LibreOffice
The Document Foundation

Base-Handbuch

Verwalten Sie ihre Daten

7.6

LibreOffice ist ein eingetragenes Markenzeichen der The Document Foundation.
Weitere Informationen finden Sie unter https://de.libreoffice.org

Copyright

Dieses Dokument unterliegt dem Copyright © 2014. Die Beitragenden sind unten aufgeführt. Sie dürfen dieses Dokument unter den Bedingungen der GNU General Public License (*http://www.gnu.org/licenses/gpl.html*), Version 3 oder höher, oder der Creative Commons Attribution License (*http://creativecommons.org/licenses/by/3.0/*), Version 3.0 oder höher, verändern und/oder weitergeben.

Warennamen werden ohne Gewährleistung der freien Verwendbarkeit benutzt.

Fast alle Hardware- und Softwarebezeichnungen und weitere Stichworte und sonstige Angaben, die in diesem Buch verwendet werden, sind als eingetragene Marken geschützt.

Da es nicht möglich ist, in allen Fällen zeitnah zu ermitteln, ob ein Markenschutz besteht, wird das Symbol (R) in diesem Buch nicht verwendet.

Mitwirkende/Autoren

Titelblatt

Robert Großkopf — basierend auf dem Design von Bayu Rizaldhan Rayes und des Gestaltungsentwurf von Drew Jensen

Vorwort

Klaus-Jürgen Weghorn — Robert Großkopf

Engl. Original: — Jean Hollis Weber

Inhalt Base

Aktuelle Ausgabe

Robert Großkopf

Vorherige Ausgaben

Robert Großkopf, Jost Lange, Michael Niedermair
Jochen Schiffers, Franklin Schiftan, Jürgen Thomas

Glossar

Erhardt Balthasar, Stefan Haas, Christian Kühl
Michael Niedermair, Florian Reisinger, Jochen Schiffers
Klaus-Jürgen Weghorn

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 15.08.2022. Basierend auf der LibreOffice Version 7.4.

Inhalt

Vorwort 12

Für wen ist dieses Buch? 13

Was finden Sie in diesem Buch? 13

Wo bekomme ich mehr Hilfe? 13

Hilfesystem 13

Freier Onlinesupport 14

Bezahlter Support und Schulungen 14

Sie sehen vielleicht etwas anderes 14

Verwenden von Tipps, Hinweisen und Warnungen 15

Wer hat dieses Buch geschrieben? 15

FAQs (häufig gestellte Fragen) 16

Einführung in Base 17

Einführung 18

Base – ein Container für Datenbankinhalte 20

Formulare – Start für die Dateneingabe 22

Tabellen – Grundlagen für die Dateneingabe 23

Abfragen – Auswertungsmöglichkeiten für eingegebene Daten 25

Berichte – Präsentationen der Datenauswertung 25

Sicherer Umgang mit einer Base-Datei 28

Eine einfache Datenbank – Testbeispiel im Detail 28

Tabellenerstellung 29

Eingabeformular 38

Mit dem Tabulator zum Unterformular 46

Navigationsleiste des Hauptformulars auch im Unterformular aktivieren 47

Eingabe eines Kontrollfeldes einschränken 47

Abfrage 49

Ausdruck im Bericht 54

Abstände zwischen den Berichtsbereichen einstellen 59

Beeinflussung eines Textfeldinhaltes durch eine Formel 59

Umstellung der Formatierung eines Textfeldes 60

Verschieben von Feldern im Report-Designer 60

Erweiterungen der Beispieldatenbank 61

Weiteres Einführungs- und Beispielmaterial 62

Datenbank erstellen 63

Allgemeines bezüglich der Erstellung einer Datenbank 64

Neue Datenbank als interne Datenbank 64

Zugriff auf externe Datenbanken 67

MySQL/MariaDB-Datenbanken 68

Erstellen eines Nutzers und einer Datenbank 69

Direkte Verbindung zu MySQL/MariaDB 69

MySQL/MariaDB-Verbindung über JDBC 69

MySQL/MariaDB-Verbindung über ODBC 70

odbcinst.ini 71

odbc.ini 71

Verbindung zur MySQL-Datenbank über den Datenbank-Assistenten 71

Die direkte Verbindung 73

Die ODBC-Verbindung 77

Die JDBC-Verbindung 78

Verbindung zu MySQL/MariaDB über das Internet 81

Zugriff auf gespeicherte Prozeduren in MySQL/MariaDB 81

PostgreSQL 82

Erstellen eines Nutzers und einer Datenbank 82

Direkte Verbindung zu PostgreSQL....83
PostgreSQL-Verbindung über JDBC....85
PostgreSQL-Verbindung über und ODBC....86
odbcinst.ini....86
odbc.ini....86
Verbindung zu PostgreSQL über das Internet....87
Autoincrement-Werte bei PostgreSQL....87
dBase-Datenbanken....88
Tabellendokumente und Tabellen in Writer-Dokumenten....90
Daten in Calc bearbeiten und in Base aktualisieren....91
Tabellen aus dem Internet über Calc automatisch aktualisiert in Base auslesen....92
Thunderbird Adressbuch....95
Texttabellen....97
Texttabellen innerhalb einer internen HSQLDB-Datenbank....97
Texttabellen als Grundlage für eine eigenständige Datenbankdatei....100
Firebird....102
Erstellen eines Nutzers und einer Datenbank....102
Direkte Verbindung zu einem Firebird Server....103
Firebird-Verbindung über JDBC....104
Firebird-Verbindung über ODBC....104
odbcinst.ini....105
odbc.ini....105
Direkte Verbindung zu einer Firebird-Datei....105
Von der externen Firebird-Datei zur Serverdatenbank....106
Benutzerverwaltung bei der externen Firebird Datenbank....108
SQLite....108
Erstellen einer Datenbank....108
SQLite-Verbindung über ODBC....109
odbcinst.ini....109
odbc.ini....109
SQLite-Verbindung über JDBC....109
Zugriff auf Access....110
Nachträgliche Bearbeitung der Verbindungseigenschaften....110
Maskierung von Tabellennamen und Feldnamen....114
Treiberverbindungen aufbewahren....115
Datenbankverbindung über SSH....116
Datenbanken in der Cloud....117
Bugs und Workarounds bei verschiedenen Datenbankverbindungen....118
MySQL/MariaDB....118
PostgreSQL....119
dBase....122
Tabellen....123
Allgemeines zu Tabellen....124
Beziehungen von Tabellen....124
Beziehungen zwischen Tabellen allgemein....125
Tabellen und Beziehungen der Beispieldatenbank....127
Tabellen Medienaufnahme....127
Tabellen Ausleihe....129
Tabellen Nutzerverwaltung....129
Erstellung von Tabellen....130
Erstellung mit der grafischen Benutzeroberfläche....132
Primärschlüssel....136
Formatierung von Feldern....137
Einstellung eines Indexes....139
Änderung bestehender Tabellen....142
Probleme bei der Änderung von Tabellen....143
Mängel der grafischen Tabellenerstellung....145

Direkte Eingabe von SQL-Befehlen....145
Tabellenerstellung....147
Tabellenänderung....151
Tabellen löschen....154
Funktionserweiterung durch Trigger bei Firebird....155
Verknüpfung von Tabellen....155
Eingabe von Daten in Tabellen....160
Eingabe über die grafische Benutzeroberfläche der Tabelle....160
Sortieren von Tabellen....163
Suchen in Tabellen....163
Filtern von Tabellen....165
Eingabemöglichkeiten über SQL direkt....167
Neue Datensätze einfügen....167
Bestehende Datensätze ändern....168
Bestehende Datensätze löschen....169
Import von Daten aus anderen Datenquellen....169
Importierte Daten an bestehende Daten einer Tabelle anfügen....170
Neue Tabelle beim Import erstellen....173
Daten Aufsplitten beim Import....175
Mängel dieser Eingabemöglichkeiten....176
Tabellen verstecken....177
Formulare....180
Formulare als Eingabeerleichterung....181
Erstellung von Formularen....181
Formulardokument in der Entwurfsansicht....181
Formulargründung über den Navigator....184
Formulargründung über ein Formularfeld....185
Externe Formulare....186
Formular-Eigenschaften....187
Eigenschaften der Kontrollfelder....190
Standardeinstellungen vieler Kontrollfelder....191
Textfeld....198
Numerisches Feld....199
Datumsfeld....200
Zeitfeld....201
Währungsfeld....201
Formatiertes Feld....202
Listenfeld....204
Kombinationsfeld....209
Markierfeld....212
Optionsfeld....213
Grafisches Steuerelement....214
Maskiertes Feld....215
Tabellen-Kontrollfeld....217
Beschriftungsfeld....218
Gruppierungsrahmen....219
Schaltfläche....222
Grafische Schaltfläche....224
Navigationsleiste....224
Drehfeld und Bildlaufleiste....226
Verstecktes Steuerelement....227
Mehrfachselektion....228
Einfaches Formular komplett erstellt....230
Felder als Gruppe hinzufügen....231
Felder anpassen....232
Felder einzeln hinzufügen....236
Tabellenkontrollfeld....237

Hauptformular und Unterformular....242
Eine Ansicht – viele Formulare....256
Fehlermeldungen bei der Eingabe in Formulare....263
Suchen und Filtern in Formularen über die Navigationsleiste....264
Datensatzsuche mit Parametern....265
Filterung mit dem Autofilter....266
Filterung mit dem formularbasierten Filter....267
Filterung über den Standardfilter....269
Fazit....271
Dateneingabe und tastaturgesteuerte Navigation....271
Drucken aus Formularen....274
Darstellungsgröße von Formularen....275
Abfragen....276
Allgemeines zu Abfragen....277
Eingabemöglichkeiten für Abfragen....277
Abfrageerstellung mit der grafischen Benutzeroberfläche....277
Funktionen in der Abfrage....285
Beziehungsdefinition in der Abfrage....289
Abfrageeigenschaften definieren....292
Abfragen nach Filterkriterien durchsuchen....295
Abfragen nachträglich ändern....298
Abfrageerweiterungen im SQL-Modus....298
Verwendung eines Alias in Abfragen....314
Abfragen für die Erstellung von Listenfeldern....315
Abfragen als Grundlage von Zusatzinformationen in Formularen....318
Eingabemöglichkeit in Abfragen....320
Verwendung von Parametern in Abfragen....325
Unterabfragen....327
Korrelierte Unterabfrage....328
Abfragen als Bezugstabellen von Abfragen....328
Zusammenfassung von Daten mit Abfragen....332
Schnellerer Zugriff auf Abfragen durch Tabellenansichten....333
Zeitdifferenzen berechnen....335
Berichte....337
Berichte mit dem Report-Designer....338
Die Benutzeroberfläche des Report-Designers....339
Gliederung eines Berichtes....347
Allgemeine Eigenschaften von Feldern....347
Besondere Eigenschaften des grafischen Kontrollfeldes....351
Diagramme im Bericht einbinden....352
Dateneigenschaften von Feldern....357
Funktionen im Report-Designer....358
Formeleingaben....358
Benutzerdefinierte Funktionen....365
Formeleingabe für ein Feld....367
Bedingte Anzeige....367
Bedingte Formatierung....368
Beispiele für Berichte mit dem Report-Designer....369
Rechnungserstellung....369
Ausdruck von Berichten zum aktuellen Datensatz des Formulars....376
Aufbau der Filtertabelle....376
Aufbau des Makros zum Start des gefilterten Berichtes....376
Wechselnde Einfärbung von Zeilen....377

Zweispaltige Berichte....380
Bugs und Workarounds beim Report-Designer....384
Der Inhalt eines Feldes aus einer Abfrage erscheint nicht....384
Ein Bericht lässt sich nicht ausführen....384
Datums- und Zeitwerte werden in Diagrammen nicht angezeigt....385
Gruppierungen mit wiederholendem Bereich zeigen nur den ersten Wert....385
Ein Neustart der Seitenzählung mit der Gruppe ist nicht möglich....387
Nachträgliche Bearbeitung des Berichtsdokuments....389
Andere Formen der Berichtserstellung....389
Das alte Berichtsmodul....390
Die BaseReportExtension....390
Weitere Berichtsmöglichkeiten....390
Datenbank-Anbindung....391
Allgemeines zur Datenbank-Anbindung....392
Anmeldung der Datenbank....392
Datenquellenbrowser....393
Daten in Text....395
Daten in Felder....398
Seriendruck....398
Aktuelle Dokument-Datenquelle....399
Explorer ein/aus....399
Serienbrieferstellung....399
Erstellung von Etiketten....405
Serienbriefe und Etiketten direkt erstellen....409
Serienbrieffelder mit der Maus erstellen....409
Serienbrieffelder über Feldbefehle erstellen....409
Externe Formulare....411
Vorteil externer Formulare....413
Nachteil externer Formulare....413
Datenbanknutzung in Calc....414
Daten in Calc einfügen....414
Daten aus Calc in eine Datenbank exportieren....419
Daten von einer Datenbank zu einer anderen konvertieren....421
Daten über die Zwischenablage in eine Tabelle einfügen....421
Datenimport aus PDF-Formularen....421
Erstellen eines PDF-Formulars....422
Auslesen der Daten aus dem PDF-Formular....423
Datenbank-Aufgaben....430
Allgemeines zu Datenbankaufgaben....431
Datenfilterung....431
Datensuche....433
Suche mit LIKE....433
Suche mit LOCATE oder POSITION....435
Bilder und Dokumente mit Base verarbeiten....440
Bilder in die Datenbank einlesen....440
Bilder und Dokumente verknüpfen....440
Dokumente mit absoluter Pfadangabe verknüpfen....442
Dokumente mit relativer Pfadangabe verknüpfen....442
Verknüpfte Bilder und Dokumente anzeigen....443
Dokumente in die Datenbank einlesen....444
Bildnamen ermitteln....446
Bildnamen aus dem Speicher entfernen....447
Bilder und Dokumente auslesen und anzeigen....447
Diagramme in Formulare einbinden....448

Diagramme aus dem Writer importieren....449
Abfragen erstellen und als Ansichten speichern....449
Abfrage für ein Säulendiagramm....449
Abfrage für ein Kreisdiagramm....450
Abfrage für ein XY-Diagramm....450
Diagramme über ein Makro anpassen....452
Übersicht über die Datenbank: BaseDocumenter – Extension....454
Codeschnipsel....459
Aktuelles Alter ermitteln....460
Geburtstage in den nächsten Tagen anzeigen....461
Tage zu Datumswerten addieren....462
Zeiten zu Zeitstempeln addieren....464
Laufenden Kontostand nach Kategorien ermitteln....465
Zeilennummerierung....466
Zeilenumbruch durch eine Abfrage erreichen....469
Gruppieren und Zusammenfassen....470
Mehrere Werte in einem Feld speichern....471
Makros....473
Allgemeines zu Makros....474
Der Makro-Editor....476
Benennung von Modulen, Dialogen und Bibliotheken....477
Makros in Base....478
Makros benutzen....478
Makros zuweisen....479
Ereignisse eines Formulars beim Öffnen oder Schließen des Fensters....479
Ereignisse eines Formulars bei geöffnetem Fenster....480
Ereignisse innerhalb eines Formulars....481
Bestandteile von Makros....482
Der «Rahmen» eines Makros....482
Variablen definieren....482
Arrays definieren....483
Zugriff auf das Formular....484
Zugriff auf Elemente eines Formulars....485
Zugriff auf die Datenbank....485
Die Verbindung zur Datenbank....485
SQL-Befehle....486
Vorbereitete SQL-Befehle mit Parametern....486
Datensätze lesen und benutzen....487
Mithilfe des Formulars....488
Ergebnis einer Abfrage....489
Mithilfe eines Kontrollfelds....490
Datensätze wechseln und bestimmte Datensätze ansteuern....491
Datensätze bearbeiten – neu anlegen, ändern, löschen....491
Inhalt eines Kontrollfelds ändern....491
Zeile einer Datenmenge ändern....492
Zeilen anlegen, ändern, löschen....492
Kontrollfelder prüfen und ändern....493
Englische Bezeichner in Makros....493
Eigenschaften bei Formularen und Kontrollfeldern....494
Schrift....494
Formular....494
Einheitlich für alle Arten von Kontrollfeld....495
Einheitlich für viele Arten von Kontrollfeld....495
Textfeld – weitere Angaben....495
Numerisches Feld....496
Datumsfeld....496
Zeitfeld....497
Währungsfeld....497

Formatiertes Feld....497
Listenfeld....498
Kombinationsfeld....499
Markierfeld, Optionsfeld....500
Maskiertes Feld....500
Tabellenkontrollfeld....500
Beschriftungsfeld....500
Gruppierungsrahmen....501
Schaltfläche....501
Navigationsleiste....501
Methoden bei Formularen und Kontrollfeldern....501
In einer Datenmenge navigieren....501
Datenzeilen bearbeiten....502
Einzelne Werte bearbeiten....504
Parameter für vorbereitete SQL-Befehle....505
Arbeit mit UNO-Befehlen in Formularen....505
Bedienbarkeit verbessern....506
Automatisches Aktualisieren von Formularen....506
Filtern von Datensätzen....507
Daten über den Formularfilter filtern....510
Filterdialog über einen Button starten....510
Durch Datensätze mit der Bildlaufleiste scrollen....512
Daten aus Textfeldern auf SQL-Tauglichkeit vorbereiten....513
Beliebige SQL-Kommandos speichern und bei Bedarf ausführen....514
Werte in einem Formular vorausberechnen....514
Die aktuelle Office-Version ermitteln....516
Wert von Listenfeldern ermitteln....516
Listenfelder durch Eingabe von Anfangsbuchstaben einschränken....517
Listenfelder mit eingeschränkter Auswahl....519
Listenfelder zur Mehrfachauswahl nutzen....522
Datumswert aus einem Formularwert in eine Datumsvariable umwandeln....524
Eingabemöglichkeiten in Feldern einschränken....524
Suchen von Datensätzen....525
Suchen in Formularen und Ergebnisse farbig hervorheben....527
Rechtschreibkontrolle während der Eingabe....531
Kombinationsfelder als Listenfelder mit Eingabemöglichkeit....533
Textanzeige im Kombinationsfeld....534
Fremdschlüsselwert vom Kombinationsfeld zum numerischen Feld übertragen....536
Kontrollfunktion für die Zeichenlänge der Kombinationsfelder....543
Datensatzaktion erzeugen....543
Navigation von einem Formular zum anderen....543
Datensatz im Formular direkt öffnen....545
Tabellen, Abfragen, Formulare und Berichte öffnen....546
Hierarchische Listenfelder....547
Filterung des Formulars mit hierarchischen Listenfeldern....548
Hierarchische Listenfelder in der Formulareingabe nutzen....551
Zeiteingaben mit Millisekunden....553
Ein Ereignis – mehrere Implementationen....554
Eingabekontrolle bei Formularen....555
Erforderliche Eingaben absichern....557
Fehlerhafte Eingaben vermeiden....560
Abspeichern nach erfolgter Kontrolle....562
Primärschlüssel aus Nummerierung und Jahreszahl....563
Datenbankaufgaben mit Makros erweitert....564
Verbindung mit Datenbanken erzeugen....564
Daten von einer Datenbank in eine andere kopieren....565
Direkter Import von Daten aus Calc....566
Zugriff auf Abfragen....571
Datenbanksicherungen erstellen....571

Interne Datenbanken sicher schließen....576
Tabellenindex heruntersetzen bei Autowert-Feldern....576
Drucken aus Base heraus....577
Druck von Berichten aus einem internen Formular heraus....577
Start, Formatierung, direkter Druck und Schließen des Berichts....577
Druck von Berichten aus einem externen Formular heraus....579
Serienbriefdruck aus Base heraus....580
Drucken über Textfelder....581
Drucken über Tabellen in Writer....582
Einfacher Druck von Text in Tabellen....582
Tabellendruck mit formatierten Zellen....584
Tabellendruck mit Bildern in der Tabelle....585
Rechnungen mit Übertrag im Tabellendruck....587
Aufruf von Anwendungen zum Öffnen von Dateien....588
Aufruf eines Mailprogramms mit Inhaltsvorgaben....590
Aufruf einer Kartenansicht zu einer Adresse....591
Mauszeiger ändern....592
Änderung beim Überfahren eines Links....592
Änderung bei gedrückter Strg-Taste und Mausklick....592
Formulare ohne Symbolleisten präsentieren....593
Formulare ohne Symbolleisten in einem Fenster....593
Formulare im Vollbildmodus....596
Formular direkt beim Öffnen der Datenbankdatei starten....596
Markierfelder durch Schaltflächen ersetzen....596
MySQL-Datenbank mit Makros ansprechen....598
MySQL-Code in Makros....598
Temporäre Tabelle als individueller Zwischenspeicher....598
Filterung über die Verbindungsnummer....599
Gespeicherte Prozeduren....599
Automatischer Aufruf einer Prozedur....599
Übertragung der Ausgabe einer Prozedur in eine temporäre Tabelle....600
PostgreSQL und Makros....600
Autowertrückgabe mit Returning....600
Datentyp «Array»....601
Dialoge....602
Dialoge starten und beenden....602
Einfacher Dialog zur Eingabe neuer Datensätze....603
Dialog zum Bearbeiten von Daten in einer Tabelle....605
Dialog zum Bearbeiten von Daten aus einer Tabellenübersicht....611
Fortschrittsbalken für den Ablauf mehrerer Prozeduren....616
Fehleinträge von Tabellen mit Hilfe eines Dialogs bereinigen....617
Makrozugriff mit Access2Base....625
Wartung....626
Allgemeines zur Wartung von Datenbanken....627
Datenbank komprimieren....627
Autowerte neu einstellen....627
Datenbankeigenschaften abfragen....627
Daten exportieren....628
Tabellen auf unnötige Einträge überprüfen....629
Einträge durch Beziehungsdefinition kontrollieren....629
Einträge durch Formular und Unterformular bearbeiten....630
Verwaiste Einträge durch Abfrage ermitteln....632
Datenbankgeschwindigkeit....632
Einfluss von Abfragen....632
Einfluss von Listenfeldern und Kombinationsfeldern....632
Einfluss des verwendeten Datenbanksystems....633

Anhang 634
Barcode 635
Datentypen des Tabelleneditors 635
Ganzzahlen 635
Fließkommazahlen 635
Text 636
Zeit 636
Sonstige 636
Datentypen in StarBasic 637
Zahlen 637
Sonstige 637
Eingebaute Funktionen und abgespeicherte Prozeduren 638
Numerisch 638
Text 640
Datum/Zeit 643
Datenbankverbindung 646
System 648
Window-Funktionen bei Firebird 649
Ranking-Funktionen 649
Navigationsfunktionen 650
Aggregatfunktionen als Windowfunktionen 651
Migration HSQLDB → Firebird 652
Steuerzeichen zur Nutzung in Abfragen 655
Einige uno-Befehle zur Nutzung mit einer Schaltfläche 656
Informationstabellen der HSQLDB 657
Informationstabellen der Firebird-Datenbank 658
Datenbankreparatur für *.odb-Dateien 660
Wiederherstellung der Datenbank-Archivdatei 661
Weitere Informationen zur Datenbank-Archivdatei 662
Behebung von Versionsproblemen 669
Weitere Tipps 669
Datenbankverbindung zu einer externen HSQLDB 669
Parallelinstallation von interner und externer HSQLDB 672
Änderung der Datenbankverbindung zur externen HSQLDB 673
Änderung der Datenbankverbindung für einen Mehrbenutzerbetrieb 674
Autoinkrementwerte mit der externen HSQLDB 675
Umgang mit der internen Firebird-Datenbank 677
Funktionserweiterungen und -änderungen in Base im Laufe der LO-Versionen 677
Glossar 680
Weitere Informationen 694
Dokumentation / Handbücher 694
Noch Fragen? 695
Mitarbeit bei LibreOffice 695
Spenden 695
Stichwortverzeichnis 696

Vorwort

Für wen ist dieses Buch?

Jeder, der sich in das Modul Base von LibreOffice einarbeiten und tiefer einsteigen will, findet hier die Möglichkeit. Sei es, dass Sie noch nie mit Datenbanken und damit mit einem Datenbanksystem (DBMS, **D**ata**b**ase **m**anagement **s**ystem) gearbeitet haben oder sei es, dass Sie ein anderes Datenbanksystem aus einer OfficeSuite oder ein eigenständiges Datenbanksystem wie beispielsweise MySQL gewohnt sind.

Was finden Sie in diesem Buch?

Dieses Buch führt Sie in die gebräuchlichsten Funktionen von LibreOffice-Base ein:

- Einführung
- Datenbank erstellen
- Ein- und Ausgabe der Datenbank: Tabellen, Formulare, Abfragen, Berichte
- Aufgaben einer Datenbank
- Makros
- Pflege und Wartung von Datenbanken
- und einiges mehr

Gegenüber der Vorversion des Handbuchs sind neben kleinen Ergänzungen die folgenden Abschnitte hinzu gekommen oder deutlich erweitert worden:

- Fehlermeldungen beim Import von Daten (Kapitel «Tabellen»)
- Dateneingabe und tastaturgesteuerte Navigation (Kapitel «Formulare»)
- Drucken über Tabellen in Writer, mit Bilder, Formatierung, Rechnungsübertrag usw. (Kapitel «Makros»)
- Listenfelder mit eingeschränkter Auswahl je nach Datensatz (Kapitel «Makros»)

Zu den Kapiteln «Bugs und Workarounds»: Es existieren viele Bugs seit der ersten Version von LibreOffice. Gerade zum Umgehen solcher Bugs ist es sinnvoll, eben solche Workarounds nutzen zu können.

Wo bekomme ich mehr Hilfe?

Dieses Buch, wie auch die anderen LibreOffice-Handbücher, das eingebaute Hilfesystem und die Benutzer-Supportsysteme setzen voraus, dass Sie mit Ihrem Computer und den Basisfunktionen wie dem Starten eines Programms, Öffnen und Speichern von Dateien vertraut sind.

Hilfesystem

LibreOffice besitzt ein umfangreiches Hilfesystem.

Um zu dem Hilfesystem zu gelangen, drücken Sie *F1* oder wählen Sie **LibreOffice Hilfe** aus dem Hilfemenü. Zusätzlich können Sie wählen, ob Sie Tipps, Erweitere Tipps und den Office-Assistenten einschalten (**Extras→ Optionen→ LibreOffice→ Allgemein**).

Wenn die Tipps eingeschaltet sind, platzieren Sie den Mauszeiger über eines der Symbole um eine kleine Box («Tooltip») angezeigt zu bekommen. Darin befindet sich eine kurze Erklärung der Funktion des Symbols. Um noch mehr Erklärungen zu erhalten, wählen Sie **Hilfe → Direkthilfe** und halten den Mauszeiger über das Symbol.

Freier Onlinesupport

Die LibreOffice-Gemeinschaft, manchmal wegen ihrer Internationalität auch Community bezeichnet, entwickelt nicht nur die Software LibreOffice, sondern bietet auch kostenfreie Unterstützung durch Freiwillige an. Mehr dazu in *Tabelle 1: Kostenlose Unterstützung für Nutzer von LibreOffice* und auf dieser Webseite: *http://de.libreoffice.org/get-help/community-support/* .

Tabelle 1: Kostenlose Unterstützung für Nutzer von LibreOffice

Freier LibreOffice-Support	
• FAQ	• Antworten auf häufig gestellte Fragen (*F*requently *A*sked *Q*uestions): *https://wiki.documentfoundation.org/Faq/de* *https://de.libreoffice.org/discover/frequently-asked-questions/*
• Dokumentation	• Handbücher und andere Dokumentationen: *http://de.libreoffice.org/get-help/documentation/* (Download Handbücher) *http://wiki.documentfoundation.org/Documentation/de* (Handbüchererstellung und Download)
• Mailinglisten	• Freier Community-Support durch ein Netzwerk an unabhängigen, erfahrenen Benutzern: *http://de.libreoffice.org/get-help/mailing-lists/* *https://www.mail-archive.com/users@de.libreoffice.org/*
• Forum	• Wir unterhalten ein eigenes Forum: *https://ask.libreoffice.org/c/german/6*
• Internationaler Support	• Die LibreOffice-Webseite in Ihrer Sprache *http://de.libreoffice.org/* • Die internationalen Mailinglisten: *http://wiki.documentfoundation.org/Local_Mailing_Lists*

Benutzer können umfassenden Onlinesupport aus der Community über Mailinglisten und Foren bekommen. Andere Webseiten, die von Nutzern geführt werden, bieten auch kostenlose Tipps und Anleitungen.

Für das Base-Handbuch sei besonders auf die Seite *https://www.familiegrosskopf.de/robert/* verwiesen. Hier liegen viele Beispieldatenbanken zum Download. Auch die Unterstützung bei der Erstellung neuer Datenbanken bietet der Autor an.

Bezahlter Support und Schulungen

Auf LibreOffice spezialisierte Firmen bieten bezahlten Support und Service an. Eine Liste mit Firmen kann bei der Mailingliste angefragt werden.

Sie sehen vielleicht etwas anderes

LibreOffice läuft auf Windows, Linux, Mac OS X, FreeBSD und anderen Unix-Varianten, von denen jedes Betriebssystem unterschiedlichste Versionen hat, und kann von den Nutzern bezüglich Schriftarten, Farben und Themen angepasst werden.

Die Bilder in diesem Buch wurden unter OpenSUSE Linux erstellt. Die Fenster sind dem Stil «Clearlooks» nachempfunden, als Symbolstil wurde weitgehend «Tango» genutzt. «Tango» ist als Extension zu LO verfügbar. Der Stil wird nicht mehr direkt mit paketiert, weil dort neuere Symbole wohl fehlen. Bei Base macht sich dies bisher allerdings nicht bemerkbar. Neuere Screenshots wurden wegen der fehlenden Symbole mit dem Symbolstil «Colibre» erstellt.

Verwenden von Tipps, Hinweisen und Warnungen

In den Handbüchern werden besondere Informationen zusätzlich gekennzeichnet: Tipps, Hinweise und Warnungen.

Tipp

Ein Tipp beschreibt eine praktische aber nicht wesentliche Information, die nicht in den Textfluss passt.

Hinweis

Ein Hinweis enthält Informationen mit Bezug auf den Text. Er erklärt, beschreibt oder kommentiert eine Aussage, um die Aufmerksamkeit des Lesers darauf zu richten.

Vorsicht

Warnungen (Vorsicht) zeigen Operationen, die zu Datenverlust führen können.

Wer hat dieses Buch geschrieben?

Dieses Buch wurde durch Freiwillige der LibreOffice-Gemeinschaft geschrieben. Gewinne aus dem Verkauf einer gedruckten Version werden für die Gemeinschaft bzw. die Stiftung «The Document Foundation» verwendet.

FAQs (häufig gestellte Fragen)

Zu LibreOffice gibt es immer wieder häufig gestellte, allgemeine Fragen. Hier ein paar Antworten:

Wie ist LibreOffice lizenziert?

LibreOffice wird unter der von der Open Source Initiative (OSI) anerkannten Lesser General Public License (LGPL) vertrieben. Die LGPL-Lizenz ist auf der LibreOffice Website verfügbar: *http://www.libreoffice.org/download/license/*

Darf ich LibreOffice an jeden vertreiben?

Ja.

Auf wie vielen Computern kann ich LibreOffice installieren?

Auf so vielen, wie Sie möchten.

Darf ich es verkaufen?

Ja.

Darf ich LibreOffice in meinem Unternehmen kostenlos verwenden?

Ja.

Ist LibreOffice in meiner Sprache verfügbar?

LibreOffice wurde in sehr vielen Sprachen übersetzt, immer wieder kommen Sprachen dazu, so dass die von Ihnen gewünschte Sprache sehr wahrscheinlich unterstützt wird. Darüber hinaus gibt es sehr viele Rechtschreib-, Silbentrenn- und Thesaurus-Wörterbücher für Sprachen und Dialekte, für die es keine lokalisierte Programmoberfläche gibt. Die Wörterbücher sind auf der LibreOffice Website erhältlich unter: *http://de.libreoffice.org/discover/templates-and-extensions/* .

Wie können Sie es kostenlos anbieten?

LibreOffice wird von Freiwilligen entwickelt und gepflegt und hat die Unterstützung von mehreren Unternehmen.

Ich schreibe eine Software-Anwendung, darf ich Programmcode von LibreOffice in meinem Programm einbauen?

Sie können dies im Rahmen der Parameter, die in der LGPL gesetzt sind, tun. Lesen Sie hierzu die Lizenzvereinbarung: *http://www.libreoffice.org/download/license/*

Wozu brauche ich Java, um LibreOffice laufen zu lassen? Ist es in Java geschrieben?

LibreOffice ist nicht in Java geschrieben, es wird in der Sprache C++ geschrieben. Java ist eine von mehreren Sprachen, die verwendet werden, um die Software zu erweitern. Das Java JDK / JRE ist nur für einige Funktionen erforderlich. Am wichtigsten davon ist die relationale Datenbank-Engine HSQLDB.
Hinweis: Java ist kostenlos erhältlich. Wenn Sie nicht möchten, dass Java verwendet wird, können Sie weiterhin nahezu alle Funktionen anderer Programmteile von LibreOffice nutzen. Base ist mit seiner Verbindung zu einer internen Java-Datenbank HSQLDB (sofern nicht Firebird genutzt wird), dem Report-Builder und vielen Assistenten allerdings ausgenommen.

Wie kann ich zu LibreOffice beitragen?

Sie können mit der Entwicklung und Pflege von LibreOffice in vielerlei Hinsicht helfen, und Sie brauchen kein Programmierer zu sein. Zum Einstieg finden Sie auf dieser Webseite weitere Informationen: *http://de.libreoffice.org/community/get-involved/*

Einführung in Base

Einführung

Im täglichen Büroeinsatz werden häufig Tabellenkalkulationen dazu benutzt, Datensammlungen zu erstellen um anschließend damit eventuell noch kleine Berechnungen durchzuführen. Die Tabellensicht ist sofort da, der Inhalt einfach einzugeben – da fragen sich dann viele Nutzer, warum es denn eine Datenbank sein sollte. Dieses Handbuch versucht den Unterschied zwischen Tabellenkalkulation und Datenbank herauszuarbeiten. Zu Beginn wird erst einmal kurz dargestellt, was eine Datenbank denn überhaupt leisten kann.

Hinweis

In der Fachsprache wird statt von Datenbanken von einer Benutzeroberfläche und einem «Datenbanksystem» gesprochen. Dieser Begriff umfasst das «Datenbankmanagementsystem» (DBMS) und den eigentlichen Datenbestand, die «Datenbank».
Base bietet einen Zugriff auf verschiedene Datenbanksysteme über eine grafische Benutzeroberfläche. Base arbeitet standardmäßig mit dem eingebetteten Datenbanksystem «HSQLDB». Seit der Version LO 4.2 ist außerdem das Datenbanksystem «Firebird», zuerst als «experimentelle Funktion», hinzugekommen. Mit der Version LO 5.3 ist auf die Version 3.0 des Firebird-Datenbanksystems, ebenfalls zunächst noch als «experimentelle Funktion» umgestellt worden. Ab Version LO 6.1 funktionierte «Firebird» zwischendurch versuchsweise standardmäßig. Ob und wann die interne «HSQLDB» durch «Firebird» ersetzt wird steht auch bei Erscheinen von LO 7.4 noch nicht fest, da die Integration von «Firebird» in die GUI bisher nicht die Qualität der Integration der «HSQLDB» erreicht.
Dieses Handbuch zeigt an verschiedenen Stellen den Unterschied zwischen den beiden internen Datenbanksystemen auf, «(HSQLDB, ~~FIREBIRD~~)» würde bedeuten: Funktioniert mit der HSQLDB, nicht aber mit Firebird. Unterschiede sind entsprechend gekennzeichnet.

Vorsicht

Wenn **Extras → Optionen → LibreOffice → Erweitert → Experimentelle Funktionen** aktiviert wird, dann wird ein Migrationsmodul aktiviert, das eingebettete HSQLDB-Datenbanken nach Firebird zu migrieren versucht. Dieses Modul erfordert anschließend viele zusätzliche Eingriffe, da Funktionen in beiden Systemen teilweise nicht identisch sind. Die Migration sollte tunlichst nicht versucht werden, bevor eine Sicherungskopie der *.odb-Datei erfolgt ist. Siehe dazu auch die Hinweise im Kapitel «Datenbank erstellen», im Anhang des Handbuches und auf *https://wiki.documentfoundation.org/Documentation/HowTo/MigrateFromHSQLDB/de*.

Das gesamte Handbuch bezieht sich in der Hauptsache auf zwei Beispieldatenbanken. Die eine Datenbank hat die Bezeichnung «Medien_ohne_Makros.odb», die andere Datenbank ist entsprechend mit Makros erweitert worden und trägt die Bezeichnung «Medien_mit_Makros.odb». Beide Datenbanken sollen einen Bibliotheksbetrieb ermöglichen: Medienaufnahme, Nutzeraufnahme, Medienverleih und alles, was damit verbunden ist wie z. B. das Anmahnen säumiger EntleiherInnen.

Daneben sind den einzelnen Kapiteln viele Beispieldatenbanken[1] beigefügt, die nur auf spezielle Probleme zugeschnitten sind:

- Beispiel_Sport.odb (Kapitel «Einführung in Base»)
- Beispiel_CSV_Einbindung.odb (Kapitel «Datenbank erstellen»)
- Beispiel_Cursorsprung_Subform_Mainform.odb (Kapitel «Formulare»)
- Beispiel_Bericht_bedingte_Einblendung_von_Grafiken.odb (Kapitel «Berichte»)

1 Alle Beispieldatenbanken können auf *http://de.libreoffice.org/get-help/documentation/* oder *https://wiki.documentfoundation.org/Documentation/de#Handbuch_f.C3.BCr_Base_.28Datenbank-Programm.29* herunter geladen werden. Weitere Beispiele auch auf: *https://www.familiegrosskopf.de/robert/index.php?&Inhalt=base_beispiele*

- Beispiel_Bericht_Rechnung.odb (Kapitel «Berichte»)
- Beispiel_Bericht_Zeilen_Farbwechsel_Spalten.odb (Kapitel «Berichte»)
- Beispiel_PDFFormular_Import.odb (Kapitel «Datenbank-Anbindung»)
- Beispiel_Autotext_Suchmarkierung_Rechtschreibung.odb (Kapitel «Datenbank-Aufgaben»)
- Beispiel_Formular_Eingabekontrolle.odb (Kapitel «Datenbank-Aufgaben» und «Makros»)
- Beispiel_Dokumente_einlesen_auslesen.odb (Kapitel «Datenbank-Aufgaben»)
- Beispiel_Baseformular_mit_Diagramm.odb (Kapitel «Datenbank-Aufgaben»)
- Beispiel_Arrayfeld.odb (Kapitel «Datenbank-Aufgaben»)
- Beispiel_Listenfeld_Mehrfachauswahl.odb (Kapitel «Makros»)
- Beispiel_Suchen_und_Filtern.odb (Kapitel «Makros»)
- Beispiel_Direktberechnung_im Formular.odb (Kapitel «Makros»)
- Beispiel_Combobox_Listfeld.odb (Kapitel «Makros»)
- Beispiel_Fortlaufende_Nummer_Jahr.odb (Kapitel «Makros»)
- Beispiel_Datenbank_Serienbrief_direkt.odb (Kapitel «Makros»)
- Beispiel_Mailstart_Dateiaufruf.odb (Kapitel «Makros»)
- Beispiel_Daten_Import.odb (Kapitel «Makros»)
- Beispiel_Dialoge.odb (Kapitel «Makros»)
- Beispiel_hierarchische_Listenfelder.odb (Kapitel «Makros»)
- Beispiel_Druck_Writer_Tabellen.odb (Kapitel «Makros»)

Bis auf die Datenbank «Beispiel_CSV_Einbindung.odb» liegen alle Datenbanken sowohl in einer Fassung für die interne HSQLDB als auch für FIREBIRD vor. Firebird kann *.csv-Dateien nicht als Datenquelle direkt nutzen und vor allem nicht in *.csv-Dateien schreiben.

Hinweis

Wie jede Software läuft auch LO-Base nicht vollkommen fehlerfrei. Besonders ärgerlich sind hier die «Regressionen», also Rückschritte von einer vorhergehenden Version zur gerade aktuellen Version. Der folgende Link führt zu den momentan noch offenen Regressionen: *https://bugs.documentfoundation.org/buglist.cgi?bug_status=UNCONFIRMED&bug_status=NEW&bug_status=REOPENED&bug_status=NEEDINFO&component=Base&keywords=regression&keywords_type=allwords&product=LibreOffice&query_format=advanced&resolution=---*

Ein Blick auf die Bug-Liste kann also helfen, Unterschiede zwischen dieser Dokumentation und eigener Programmversion zu verstehen.

Hinweis

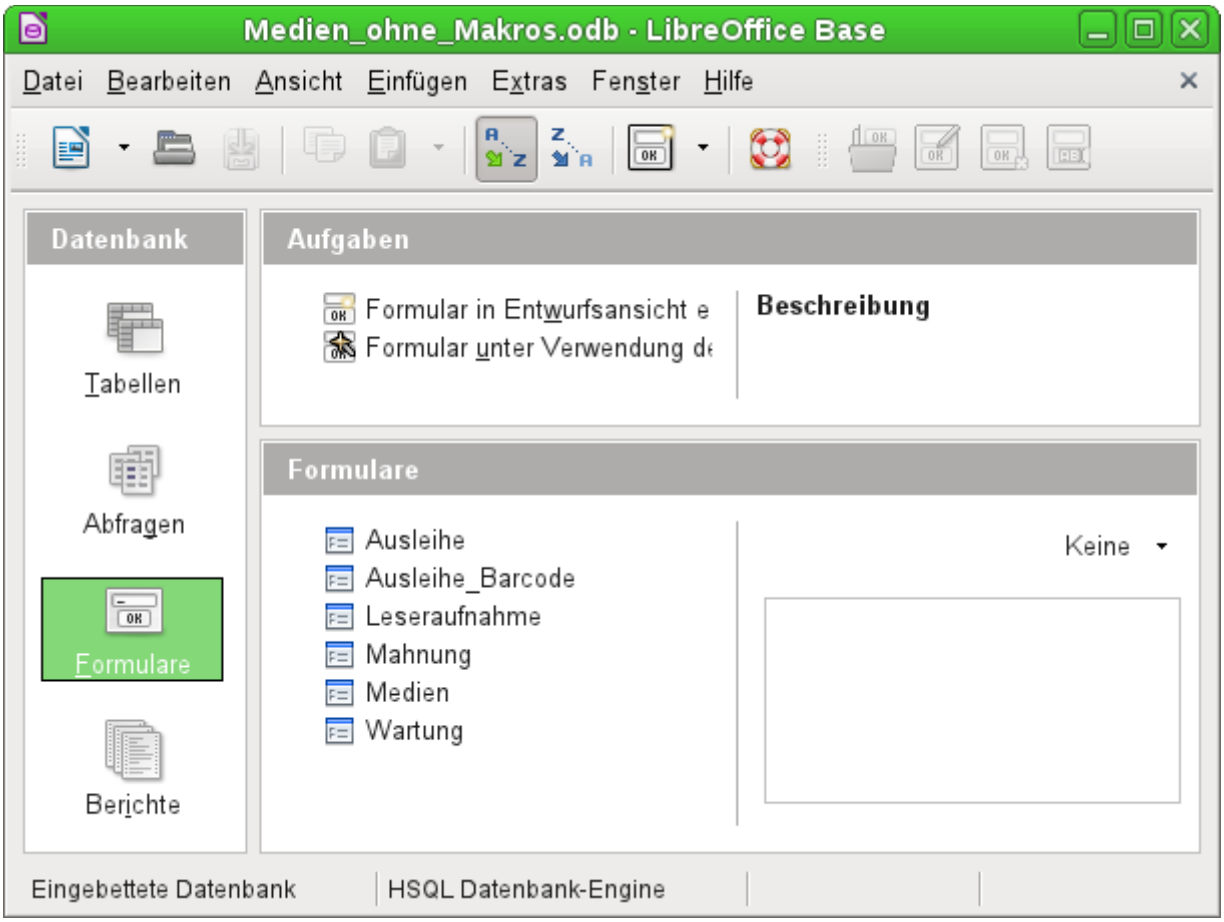
Base-Dateien sind Dateien, in denen die interne Datenbank mit abgespeichert wird. Diese Datenbankdateien sollten immer wieder gesichert werden, damit der Datenverlust gering gehalten wird. Besonderes Augenmerk gilt hier der Migration von der internen HSQLDB zur Firebird-Datenbank. Auch sollte eine Base-Datei mit interner Datenbank immer nach Beendigung der Arbeit abgespeichert werden. Ein Zuklappen des Laptops kann sonst beim Erwachen des Systems zu Datenverlust führen.

Hinweis

In diesem Handbuch werden Anführungszeichen mit unterschiedlichen Intentionen verwendet:
"Tabellenname"."Feldname" > Tabellen und Felder in Datenbanken werden in doppelten Anführungszeichen oben wieder gegeben.
'Wert' > Werte, die in die Datenbank eingegeben oder aus ihr gelesen werden, sollen in einfachen Anführungszeichen erscheinen.
«Hervorhebungen» > Alles, was sonst hervorgehoben werden soll, erscheint in der französischen Form der Anführungszeichen.

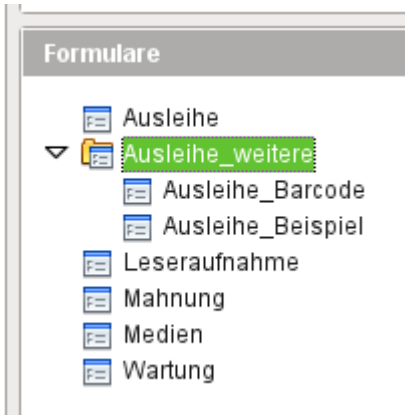
Base – ein Container für Datenbankinhalte

Eine Base-Datei ist eigentlich nur ein gepacktes Verzeichnis, in dem Informationen für die verschiedenen Arbeitsbereiche von Base stecken. In der täglichen Nutzung startet eine Base-Datei erst einmal mit der folgenden Ansicht:



Zur Arbeitsumgebung von Base gehören insgesamt vier Arbeitsbereiche: **Tabellen**, **Abfragen**, **Formulare** und **Berichte**. Je nach gewähltem Arbeitsbereich können bestimmte Aufgaben zur Neuerstellung der Elemente in Angriff genommen oder fertiggestellte Elemente aufgerufen werden.

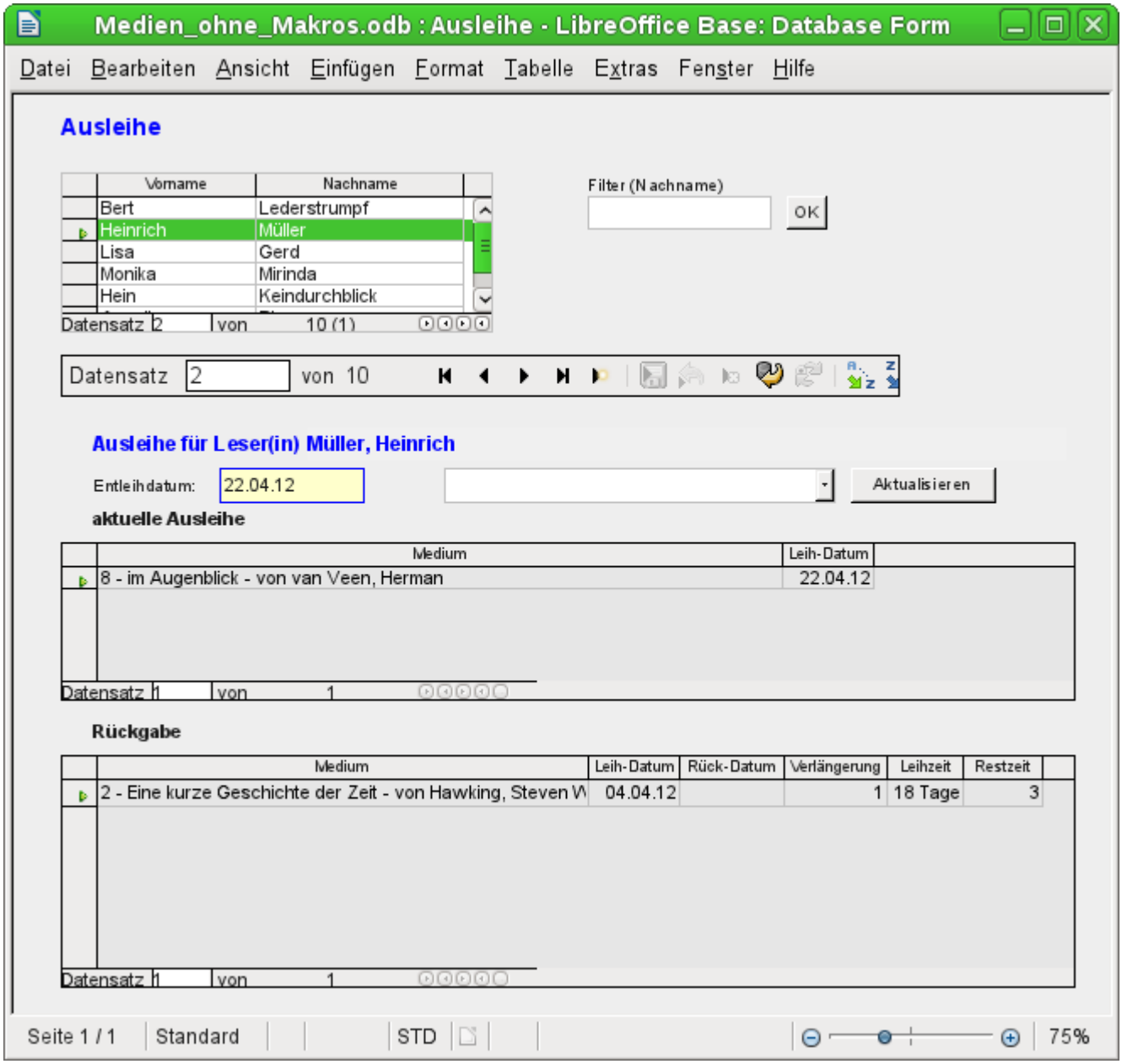
In den Arbeitsbereichen **Formulare** und **Berichte** können die jeweiligen Elemente auch innerhalb einer Verzeichnisstruktur angeordnet werden:



Dies geht entweder direkt beim Abspeichern über den Speicherdialog oder durch Neugründung von Verzeichnissen über **Einfügen → Ordner**. Wird ein Formular über den Assistenten erstellt, so steht der Speicherort in dem Ordner leider nicht zur Verfügung. Hier müsste dann das Formular anschließend in den Unterordner verschoben werden.

Obwohl die Basis für eine Datenbank durch Tabellen gebildet wird, startet Base mit der Formularansicht, weil Formulare in der Regel die Elemente sind, mit denen die tägliche Datenbankarbeit vonstatten geht. Über die Formulare werden Einträge in die Tabellen vorgenommen und Inhalte aus den Tabellen ausgewertet.

Formulare – Start für die Dateneingabe



Einfache Formulare bilden lediglich eine Tabelle wie im oben sichtbaren Tabellenkontrollfeld mit den Namen ab. Dieses Formular erfüllt durch seine Struktur einige zusätzliche Punkte.

- Die Auswahl der Personen kann mit Filtern eingeschränkt werden. Die einfache Eingabe des Buchstaben 'G' erzeugt eine Auswahl aller Personen, deren Nachname mit 'G' beginnt.
- Neue Mediennutzer können direkt in das Tabellenkontrollfeld eingegeben werden.
- Die Ausleihdaten des aktuell markierten Nutzers werden in den darunterliegenden Feldern angezeigt. Dabei wird auch noch einmal der Name des Nutzers deutlich hervorgehoben. Hat ein Nutzer ein Buch entliehen und müsste es bereits zurückgegeben werden, so steht das Tabellenkontrollfeld «aktuelle Ausleihe» erst einmal nicht zur Verfügung. Stattdessen wird angegeben, dass die Ausleihe zur Zeit gesperrt ist.
- Das Entleihdatum wird mit dem aktuellen Datum vorgegeben. In dem daneben stehenden Listenfeld werden die zu entleihenden Medien ausgewählt. Dabei können keine Medien ausgewählt werden, die zur Zeit noch entliehen sind.

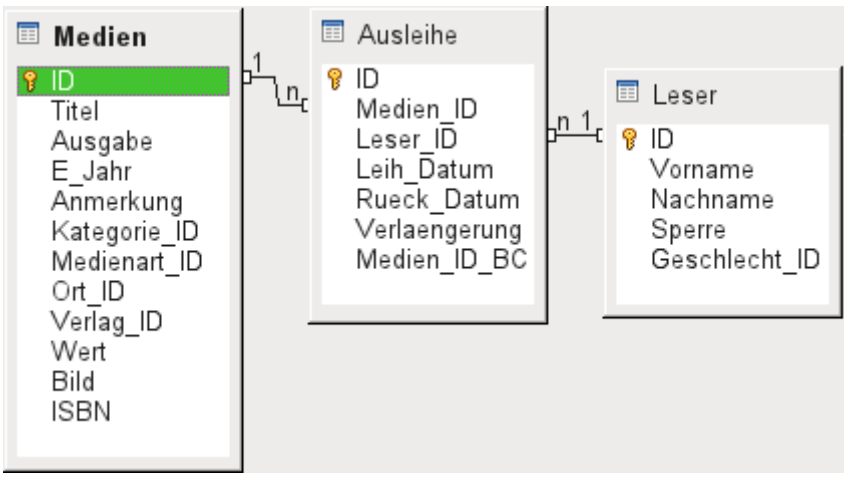
- Die ausgewählten Daten werden mit dem Button Aktualisieren in das Tabellenkontrollfeld für den aktuellen Ausleihvorgang übernommen.
- Im Tabellenkontrollfeld für die Rückgabe ist es nicht möglich, einen Datensatz einfach zu löschen. Nur die Felder «Rück-Datum» und «Verlängerung» können bearbeitet werden. War ein Mediennutzer vorübergehend gesperrt und hat seine entliehenen Medien zurückgegeben, so lässt sich die Ausleihe über den Button Aktualisieren wieder freischalten.

Alle diese Funktionen können ohne den Einsatz von zusätzlicher Programmierung mit Makros gewährleistet werden, wenn entsprechend an den Formularen gefeilt wird.

Tabellen – Grundlagen für die Dateneingabe

Die Tabellen in einer Datenbank hängen in einem großen Geflecht zusammen. Eine Tabelle bezieht Informationen aus einer anderen oder gibt Informationen an andere Tabellen weiter. Dies wird als «Relation» bezeichnet.

Die Tabelle "Ausleihe" steht in direkter Beziehung zu den Tabellen "Medien" und "Leser".

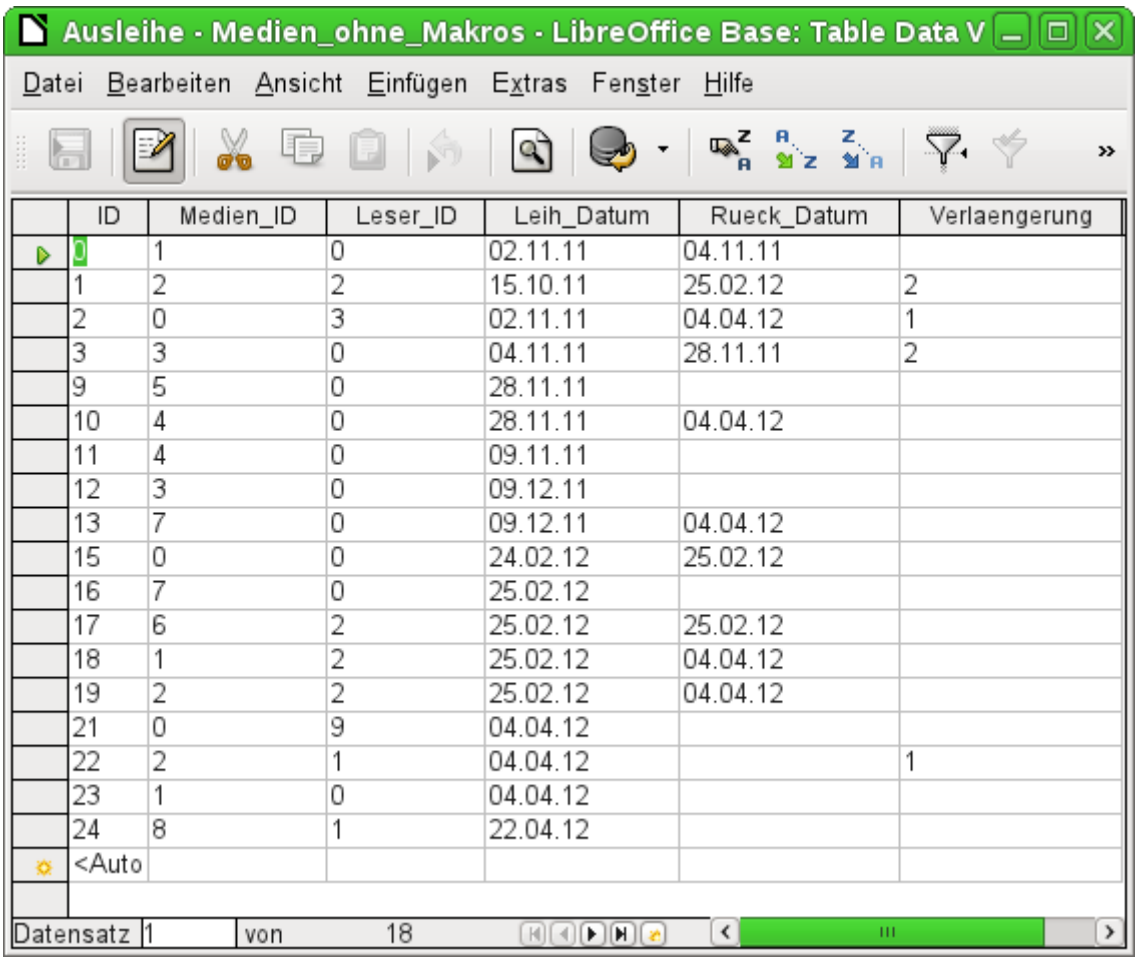


Statt in der Ausleihe den Titel eines Buches abzuspeichern, wird dort nur eine Zahl abgespeichert. Das eindeutige Kennzeichen eines Datensatzes der Tabelle "Medien" ist in dem Feld "ID" gespeichert. Das Feld ist das Schlüsselfeld der Tabelle "Medien", der Primärschlüssel.

In der Tabelle "Ausleihe" wird auch nicht jedes Mal der Lesername eingetragen. Hier gibt die Tabelle "Leser" Auskunft. Auch sie hat ein Primärschlüsselfeld. Der Wert dieses Feldes kann dann in die Tabelle "Ausleihe" eingetragen werden.

Die Beziehungen zwischen den Tabellen haben hier den Vorteil, dass die Schreibarbeit im Formular erheblich reduziert wird. Statt bei jeder Ausleihe ein Medium zumindest unverwechselbar und für jeden erkennbar aufzuschreiben und Leser mit Vor- und Nachnamen zu notieren, wird einfach im Inhalt der anderen Tabellen nachgesehen. Schließlich wird dasselbe Medium später noch öfter ausgeliehen und derselbe Leser kann schon beim ersten Entleihvorgang mehrere Medien entleihen.

Relationen helfen auch, Fehler durch unterschiedliche Schreibweisen z.B. von Lesernamen zu vermeiden.



ID	Medien_ID	Leser_ID	Leih_Datum	Rueck_Datum	Verlaengerung
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	
11	4	0	09.11.11		
12	3	0	09.12.11		
13	7	0	09.12.11	04.04.12	
15	0	0	24.02.12	25.02.12	
16	7	0	25.02.12		
17	6	2	25.02.12	25.02.12	
18	1	2	25.02.12	04.04.12	
19	2	2	25.02.12	04.04.12	
21	0	9	04.04.12		
22	2	1	04.04.12		1
23	1	0	04.04.12		
24	8	1	22.04.12		
<Auto					

Die Tabellengrundlage für das oben vorgestellte Formular sieht erst einmal recht nüchtern aus. In der oben abgebildeten Tabelle werden bis auf die Nutzerneueingabe alle Eingaben gemacht, die in dem Formular möglich sind. Dabei werden im Formular die Verbindungen dieser Tabelle zu den anderen Tabellen der Datenbank genutzt.

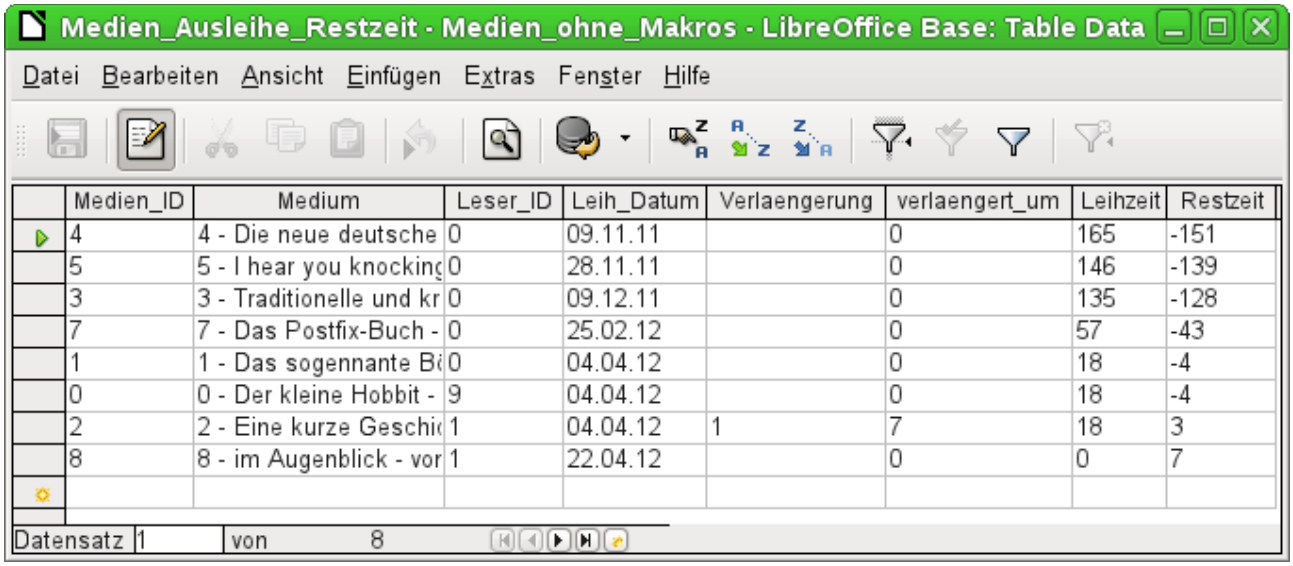
- Das erste Feld zeigt den für die meisten Datenbanken unabdingbaren, unverwechselbaren Inhalt, den Primärschlüssel ("ID"), der automatisch geschrieben wird.
  Mehr dazu im Kapitel «Beziehungen zwischen Tabellen allgemein».
- Das zweite Feld "Medien_ID" speichert den Primärschlüssel der Tabelle "Medien". Es verweist über die Nummer auf den entsprechenden Inhalt in dieser Tabelle. Solch ein Verweis auf einen Primärschlüssel wird als Fremdschlüssel bezeichnet. Im Formular werden statt des Fremdschlüssels über ein Listenfeld der Titel und der Verfasser angezeigt. Das Listenfeld gibt im Hintergrund den Fremdschlüssel an die Tabelle weiter.
- Das dritte Feld "Leser_ID" speichert den Primärschlüssel der Tabelle "Leser", also nur eine Nummer, die auf den Leser verweist. Im Formular wird aber der Nachname und der Vorname angegeben. Wie in der Tabelle zu sehen ist, hat der Leser mit der Primärschlüsselnummer '0' sehr viele Medien entliehen. Die Tabelle kann beliebig oft den einzigartigen Primärschlüssel der Tabelle "Leser" als Fremdschlüssel "Leser_ID" abspeichern. Auf keinen Fall darf aber ein Leser, der in dieser Tabelle über den Fremdschlüssel verzeichnet ist, gelöscht werden. Sonst wäre nicht mehr nachvollziehbar, wer denn nun das jeweilige Medium entliehen hat. Die Datenbank macht in den Standardeinstellungen so ein Löschen unmöglich. Der Fachbegriff dafür ist die Gewährung der «referentiellen Integrität».
- Im vierten Feld wird das Ausleihdatum abgespeichert. Ist dieses Datum abgespeichert und entspricht das Datum nicht dem aktuellen Datum, so erscheint der entsprechende Datensatz zu dem entsprechenden Leser im Formular im untersten Tabellenkontrollfeld zu Rückgabe der Medien.

- Im letzten Feld wird eine Verlängerung ermöglicht. Was hier eine 1, 2 usw. bedeutet wird an anderer Stelle festgelegt. Dafür enthält die Datenbank eine gesonderte Tabelle mit der Bezeichnung "Einstellungen".

Diese Eingaben reichen aus, um letztlich einen Bibliotheksbetrieb in Gang zu halten.

Abfragen – Auswertungsmöglichkeiten für eingegebene Daten

Abfragen zeigen eine Sicht auf die Tabellen. Sie bringen Inhalte mehrerer Tabellen zusammen in einer Übersicht. Abfragen werden nur in der Abfragesprache «SQL» gespeichert. Sie sind also keine neuen Tabellen, auch wenn sie von der Ansicht her in Base erst einmal gleich erscheinen.



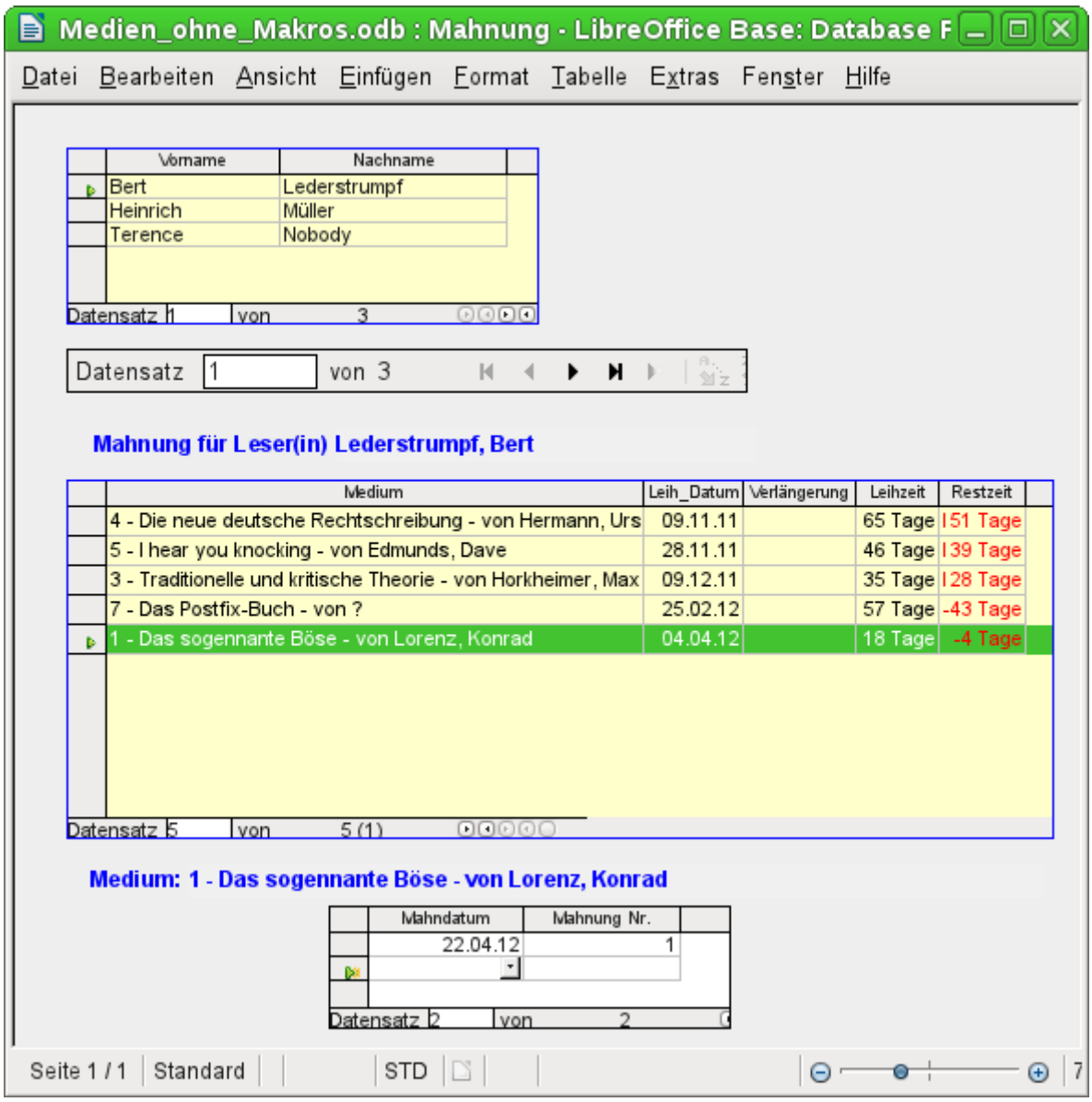
Medien_ID	Medium	Leser_ID	Leih_Datum	Verlaengerung	verlaengert_um	Leihzeit	Restzeit
4	4 - Die neue deutsche	0	09.11.11		0	165	-151
5	5 - I hear you knocking	0	28.11.11		0	146	-139
3	3 - Traditionelle und kr	0	09.12.11		0	135	-128
7	7 - Das Postfix-Buch -	0	25.02.12		0	57	-43
1	1 - Das sogennante Bi	0	04.04.12		0	18	-4
0	0 - Der kleine Hobbit -	9	04.04.12		0	18	-4
2	2 - Eine kurze Geschic	1	04.04.12	1	7	18	3
8	8 - im Augenblick - vor	1	22.04.12		0	0	7

Diese Abfrage listet alle Medien auf, die zur Zeit entliehen sind. Darüber hinaus berechnet sie, wie lange die Medien bereits entliehen sind und wie lange sie noch entliehen werden dürfen. Während das Feld "Medien_ID" das Fremdschlüsselfeld ausliest, wird in dem Feld "Medium" aus dem Primärschlüssel, dem Titel und dem Autor des Mediums ein zusammenhängender Text gebildet. Dieses Feld wird dann im Formular unter dem Untertitel «Rückgabe» benötigt. Einige Felder der Abfrage dienen dabei als Verbindungsfelder zu dem eigentlichen Ausleihformular mit der Tabelle "Ausleihe", nämlich die Felder "Medien_ID" und "Leser_ID".

- Alle Medien, bei denen das Rückgabedatum in der Tabelle "Ausleihe" nicht ausgefüllt ist, werden aufgelistet. Die Medien sind hier als zusätzliche Übersicht auch noch entsprechend benannt.
- Der Bezug zu den Lesern ist hergestellt über den Primärschlüssel der Tabelle "Leser"
- Aus dem Ausleihdatum "Leih_Datum" und dem aktuellen Datum wird die Zeitdifferenz in Tagen als "Leihzeit" angegeben.
- Von der Ausleihzeit, die je nach Medienart unterschiedlich sein kann, wird die Leihzeit abgezogen. Daraus wird die verbleibende Restzeit gebildet.
- Bei einer Verlängerung wurde in der Tabelle "Einstellungen" vermerkt, dass der Wert '1' für eine Verlängerung von 7 Tagen steht. Im vorletzten Datensatz mit der "Medien_ID" '2' ist so eine Verlängerung bei der Restzeit berücksichtigt worden.

Berichte – Präsentationen der Datenauswertung

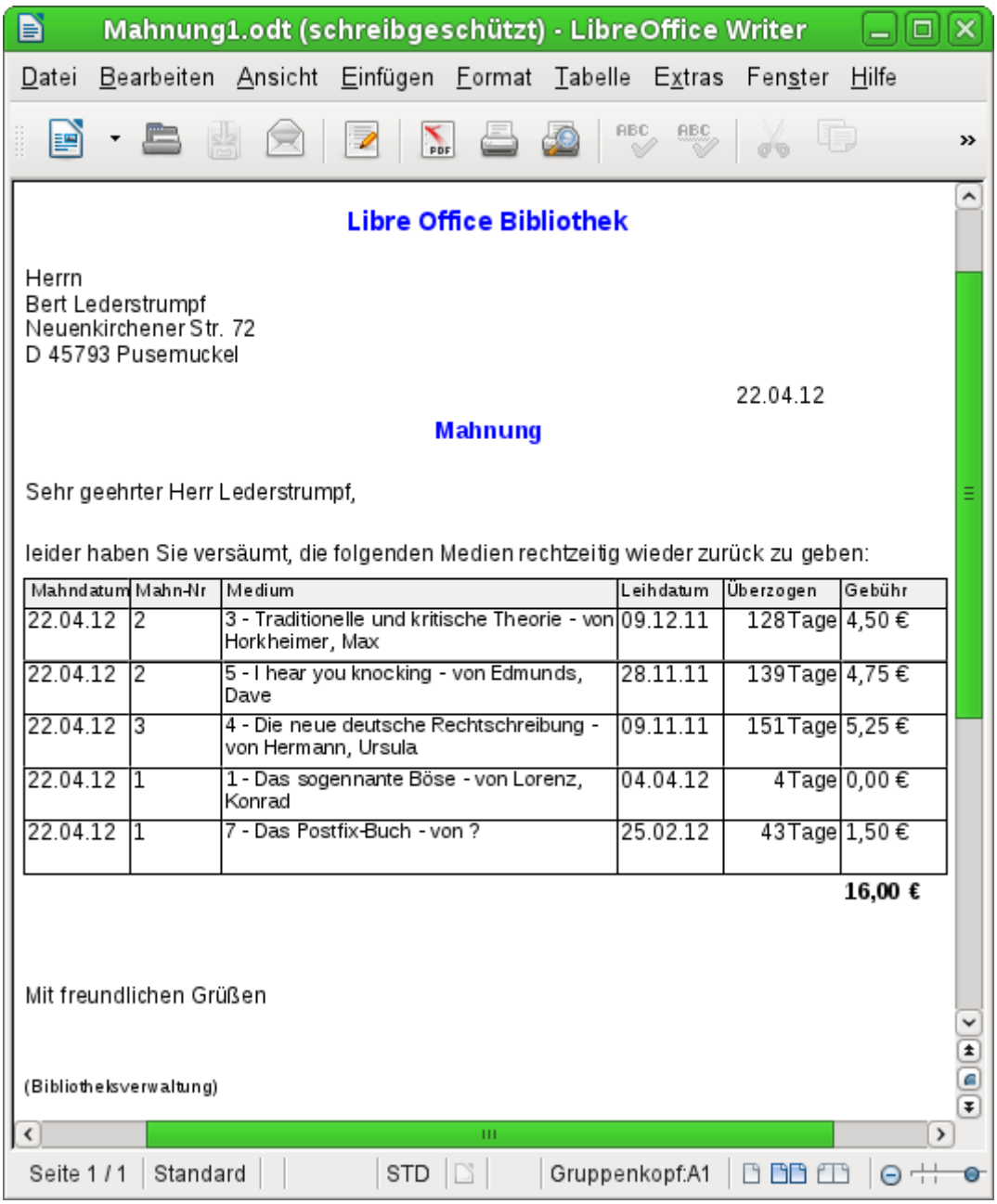
Über Berichte werden die Daten so aufbereitet, dass sie sinnvoll ausgedruckt werden können. Formulare wie das folgende sind nicht dazu geeignet, z. B. einen sauber formatierten Brief auszugeben.



Bevor ein aktueller Bericht in Form einer Mahnung ausgedruckt werden kann, müssen in diesem Formular erst einmal die Mahnungen bestätigt werden. Im oberen Tabellenkontrollfeld stehen dafür alle Namen der Leser, die ein Medium entliehen haben und für die eine negative Restzeit vermerkt ist.

Für jedes anzumahnende Buch wird im unteren Tabellenkontrollfeld ein Mahndatum festgelegt. Dies dürfte bei der Abarbeitung von Mahnungen das aktuelle Datum sein. Die Mahnungsnummer wird nicht extra geschrieben. Sie wird anhand der bisher ergangenen Mahnungen einfach durch Addition ermittelt.

Dieses Formular benötigt in der Fassung ohne Makros noch eine Eingabe durch den Nutzer. In der Fassung, die mit Makros arbeitet, wird einfach das Datum automatisch geschrieben und anschließend der Bericht zum Mahnungsdruck aufgerufen.



Mit Hilfe einer Abfrage lässt sich aus den getätigten Eingaben solch eine Mahnung zum Ausdruck fertig erstellen. Der Nutzer der Datenbank braucht dazu lediglich bei den Berichten den Bericht Mahnung auszuwählen und kann dann einen entsprechenden Mahnbrief an alle die Personen schicken, bei denen im vorher angegebenen Formular eine Mahnung bearbeitet wurde.

In so einem Bericht stehen also gegebenenfalls auf den folgenden Seiten jeweils auf einer Seite weitere Mahnungen für andere Personen. Sollte ein Leser so viele Medien entliehen haben, dass der Platz auf einer Seite nicht ausreicht, so wird die Tabelle auf der Folgeseite einfach fortgesetzt.

Ein so erstellter Bericht ist also umfangreicher als ein Serienbrief, der mit Writer erstellt wird: er stellt automatisch alle Datensätze zusammen, die gedruckt werden sollen und ordnet den zusätzlichen Text entsprechend an.

Ein ähnlicher Brief wie in der obigen Abbildung lässt sich sonst nur unter Zuhilfenahme von Makros realisieren, wie es in *Drucken über Textfelder* beschrieben wird.

Sicherer Umgang mit einer Base-Datei

Tabellen der internen Datenbank, Abfragen, Formulare und Berichte werden in einer Base-Datei gespeichert. Durch diese Vielzahl an Elementen, von denen z.B. die Datenbank auch noch während der Arbeit in den Arbeitsspeicher ausgelagert wird, ist solch eine Datei ein Element, mit dem sorgsam umzugehen ist. Manchmal sind Bugmeldungen zu lesen, die vor diesem Hintergrund deutlich machen, dass eine Datenbankdatei eben einen etwas sorgfältigeren Umgang benötigt als z.B. eine Textdatei, die mit dem Writer geschrieben wird.

Die folgenden Hinweise sollten deshalb beim Umgang mit einer Base-Datei berücksichtigt werden:

- Eine geöffnete Datenbankdatei sollte möglichst nicht über **Speichern unter** mit einem anderen Namen abgespeichert werden. Wenn es schon nicht anders geht, so sollten Tabellen, Abfragen, Formulare und Berichte vorher geschlossen werden. Besser ist es, die Datenbankdatei zu schließen und eine Kopie der Datei zu erstellen.
  Der Report-Designer ist ein Addon. Er wird zwar inzwischen nicht mehr als gesonderte Erweiterung sichtbar, arbeitet aber weitgehend unabhängig von der Datenbankdatei. Das Umbenennen der Datenbankdatei über **Speichern unter** entzieht dem Report-Designer seine Grundlage.
- Die Abspeicherung einer Tabelle, einer Abfrage, eines Formulars oder eines Berichtes hat nicht zur Folge, dass die komplette Datenbankdatei abgespeichert wird. Diese Abspeicherung muss separat erfolgen. Nach wichtigen Schritten bei der Erstellung von Elementen für die *.odb-Datei muss deshalb immer das Element und anschließend die Datenbankdatei gespeichert werden.
  Besonders trifft dieses Speicherverhalten auf die Arbeit mit dem Report-Designer zu. Die Erstellung eines Berichtes ist noch die labilste Komponente innerhalb der Base-Datei. Deshalb nach jedem Schritt Bericht und *.odb-Datei sichern. Ist der Bericht erstellt, so funktionieren die Berichte selbst ohne besondere Probleme.
- Eine einmal fertig gestellte *.odb-Datei, die anschließend nur noch mit Daten gefüttert wird, wird nicht wieder über den Speicherbutton abgespeichert. Die Speicherung der Datensätze erfolgt letztlich beim Schließen der *.odb-Datei. Der Inhalt der Datenbank wird erst dann zurück in die Datei geschrieben. Ein Absturz an dieser Stelle kann zu Datenverlust führen. Deshalb sollte eine Strategie entwickelt werden, wie Sicherungskopien rechtzeitig erstellt werden. Im Kapitel «Makros» befindet sich ein Makro, mit dem solch eine Sicherungskopie beim Öffnen einer Datenbankdatei automatisch erstellt wird. Ebenso ist dort eine Möglichkeit aufgeführt, wie auch zwischendurch gesichert werden kann.
  Einem deutlich höheren Sicherheitsstandard genügen schließlich externe Serverdatenbanken wie MySQL/MariaDB oder PostgreSQL. Für diese könnte dann Base als Frontend mit Abfragen, Formularen und Berichten dienen.

Vorsicht

Für Firebird vorläufig noch: Daten werden nicht automatisch gespeichert. Das Speichersymbol in Tabellen zeigt zwar eine Speicherung beim Datensatzwechsel an. Diese Speicherung muss aber noch im Hauptfenster der Base-Datei bestätigt werden. Ohne diese Speicherung gehen alle neu eingegebenen Daten verloren.

Eine einfache Datenbank – Testbeispiel im Detail

Die Erstellung einer Datenbank wird in «Neue Datenbank als interne Datenbank» abgehandelt. Dort werden neben der internen HSQLDB oder der internen Firebird-Datenbank verschiedene andere Datenbanksysteme eingebunden.

Das folgende Beispiel basiert auf der Standarddatenbank HSQLDB, die mit LibreOffice als interne Datenbank installiert wird. Unterschiede zur internen Firebird-Datenbank werden an entsprechender Stelle gesondert gekennzeichnet.

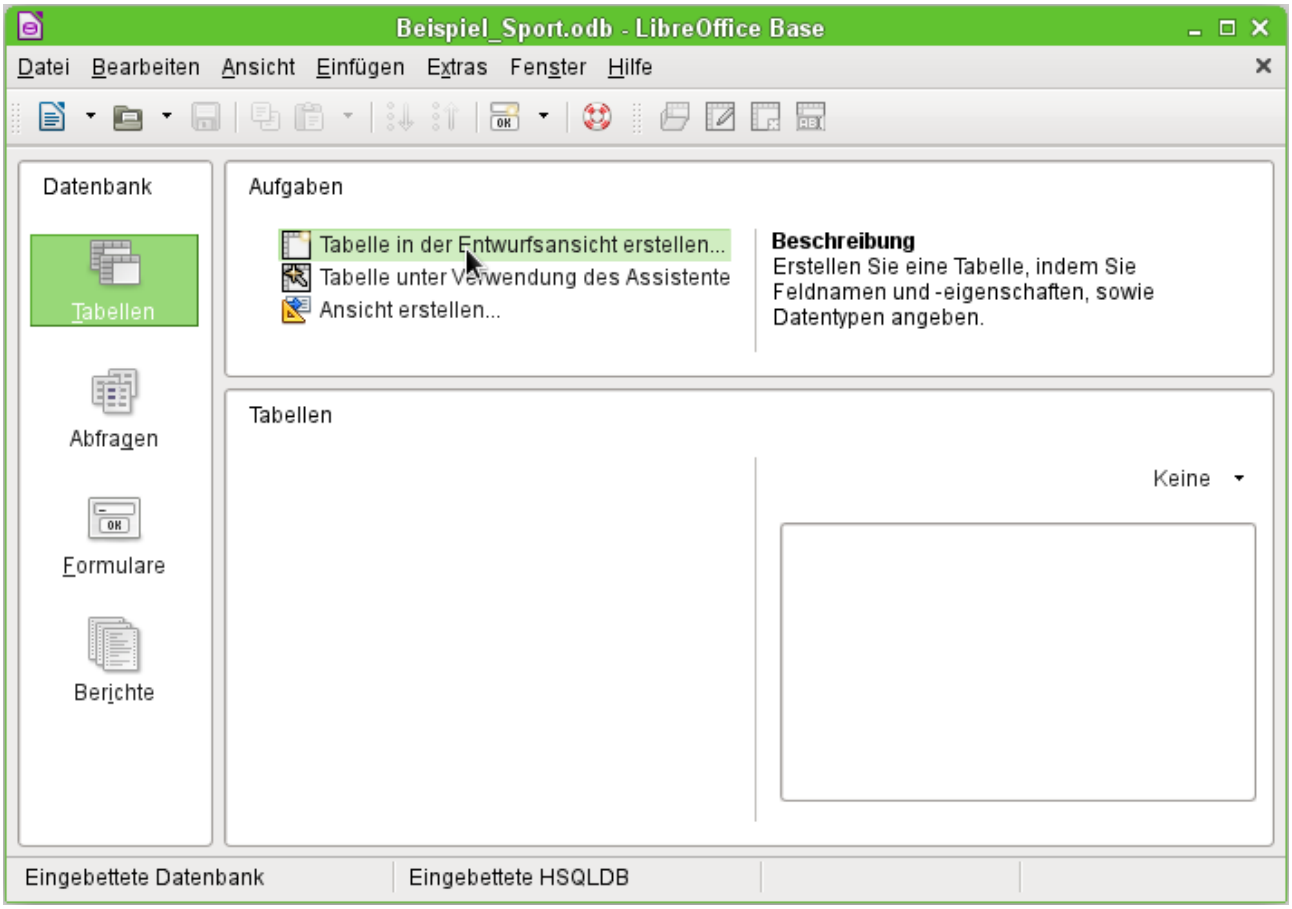
Es wird zuerst eine *Neue Datenbank als interne Datenbank* erstellt, die nicht angemeldet wird.

Erstes Kennzeichen bei der Erstellung einer Datenbank ist, dass zum Abschluss des Assistenten zuerst einmal ein Speicherort gesucht wird und die Datei abgespeichert wird.

Die Datenbank soll dazu dienen, einen Sportwettkampf in verschiedenen Disziplinen zu organisieren. Als Bezeichnung wurde deshalb «Beispiel_Sport»[2] gewählt.

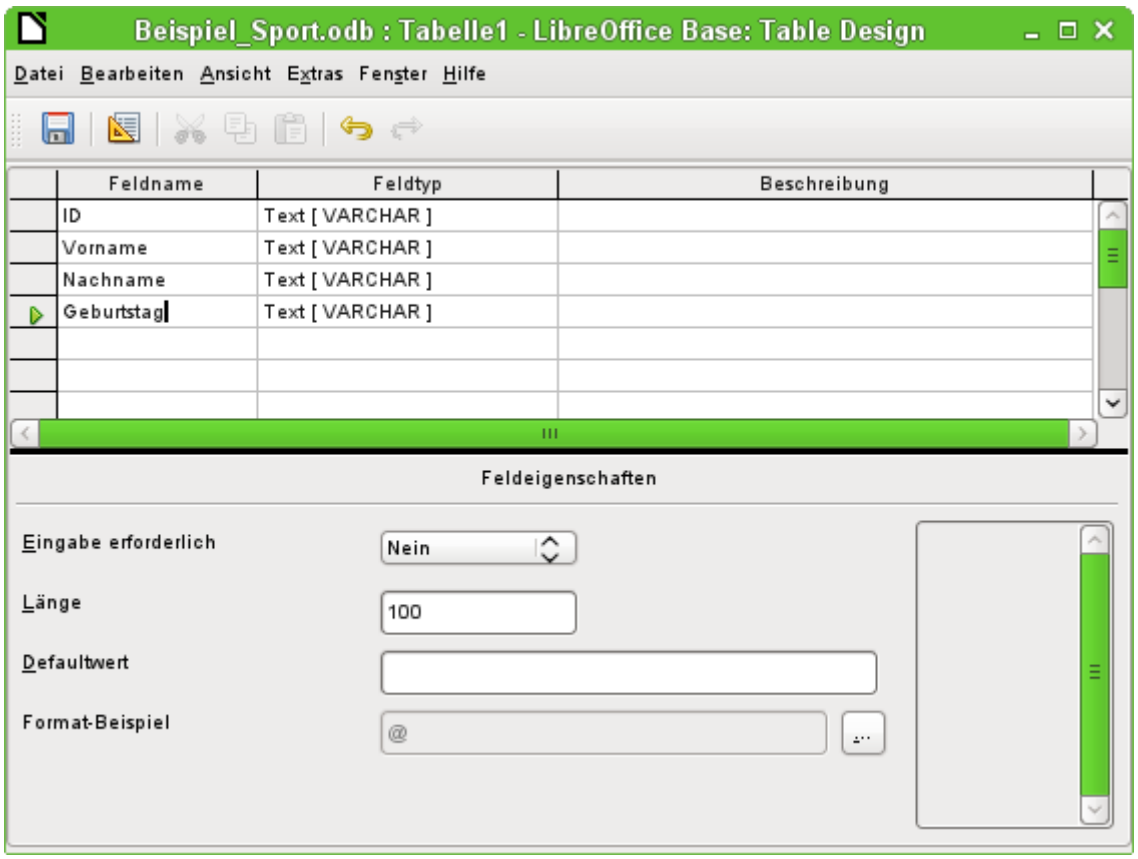
Tabellenerstellung

Sobald die Abspeicherung erfolgt ist, erscheint der Startbildschirm der Benutzeroberfläche für die Datenbank. Standardmäßig sind zuerst auf der linken Seite im Bereich **Datenbank → Tabellen** ausgewählt. Die Tabellen sind das zentrale Speicherelement für die Daten. Ohne Tabellen keine Datenbank.



Tabellen → Aufgaben → Tabelle in der Entwurfsansicht erstellen … wird angeklickt. Es öffnet sich der folgende Bildschirm:

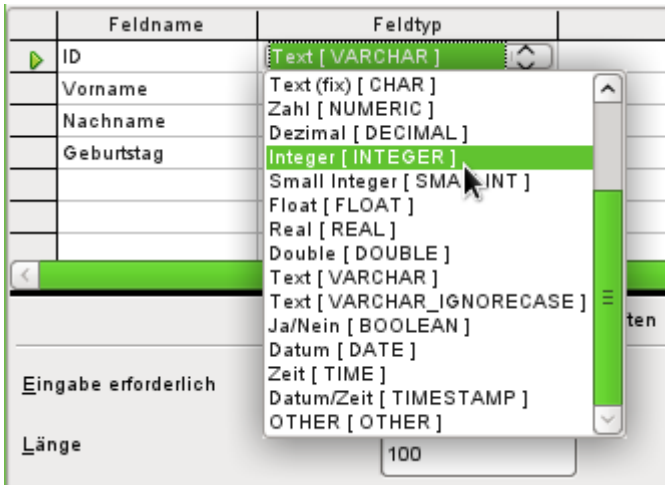
2 Die Datenbank «Beispiel_Sport.odb» ist den Beispieldatenbanken für dieses Handbuch beigefügt.



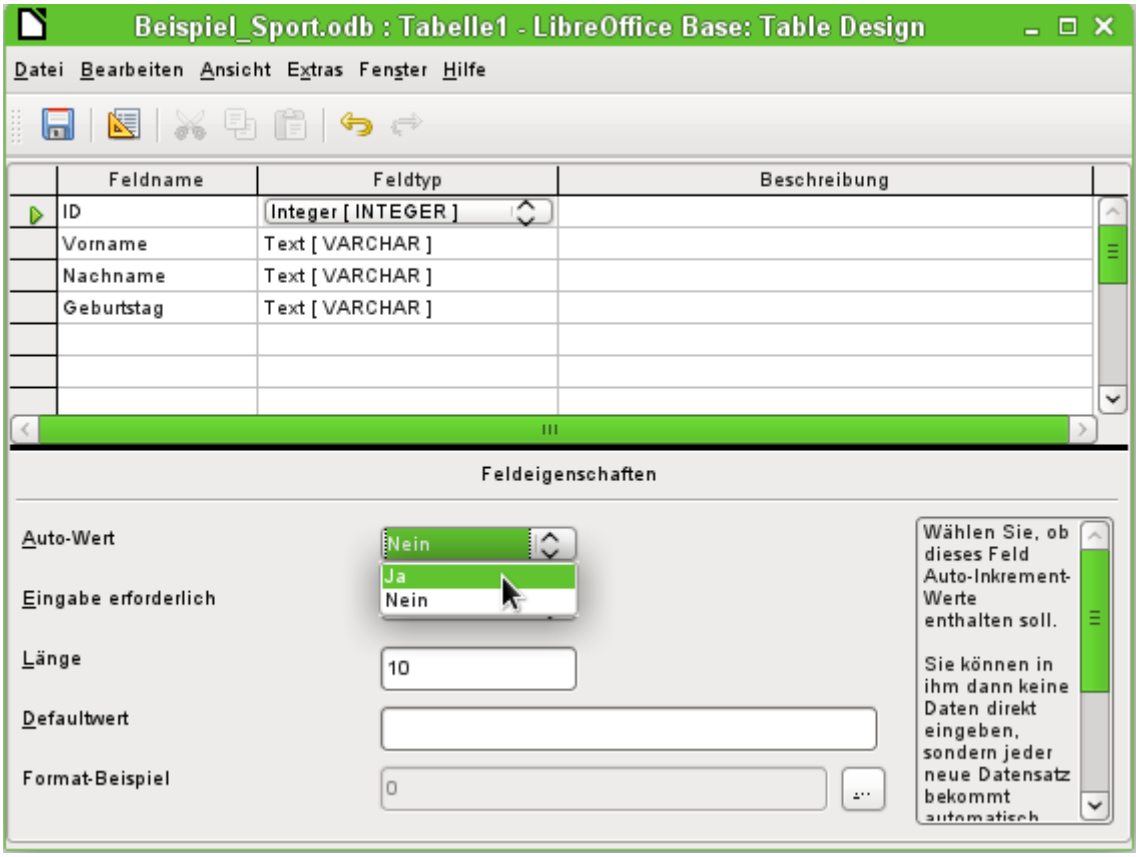
Hier werden zuerst einmal die verschiedenen Feldnamen für die erste Tabelle eingetragen. Die Tabelle soll die Teilnehmer und Teilnehmerinnen enthalten. Sie ist hier erst einmal auf die wesentlichen Bestandteile reduziert.

Die Feldnamen "Vorname", "Nachname" und "Geburtstag" dürften klar sein. Zusätzlich wurde ein Feld mit der Bezeichnung "ID" eingetragen. Dieses Feld soll später einen Wert aufnehmen, der die einzigartige Eigenschaft eines jeden Datensatzes ist. So ein einzigartiges Schlüsselfeld ist für die eingebettete Datenbank notwendig. Ansonsten lassen sich keine Datensätze in die Tabelle eingeben. Dieses Schlüsselfeld wird bei Datenbanken als «Primärschlüssel» bezeichnet.

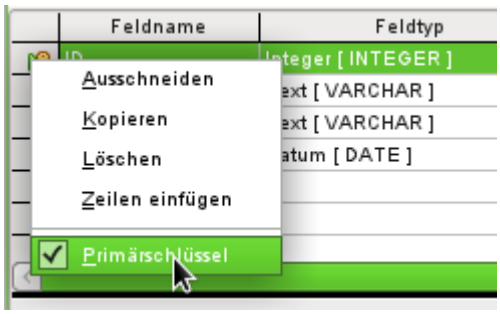
Es könnte natürlich auch ein anderes Feld für diese Eigenschaft genutzt werden. Wird aber z. B. der "Nachname" alleine dafür genutzt, so können nicht zwei Personen mit gleichem Nachnamen abgespeichert werden. In diesem Fall könnte es helfen, zwei Felder zusammen zu einem gemeinsamen Primärschlüssel zu erklären. Eine Garantie, dass das auf Dauer funktioniert, ist das aber nicht. Deshalb wird hier die einfachere Variante bevorzugt.

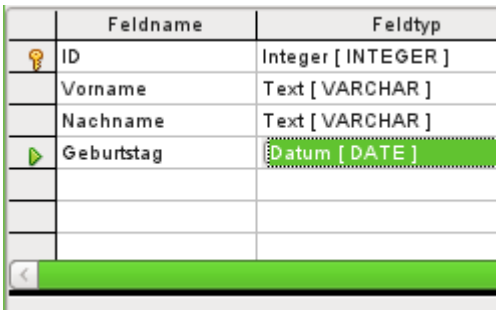


Im zweiten Arbeitsgang werden jetzt die Feldtypen für die bereits benannten Felder aus der Liste ausgewählt. Das Feld "ID" wird auf den **Feldtyp → Integer** eingestellt. Es wird dann Ganzzahlen speichern. Dieser Feldtyp hat den Vorteil, dass er bei der eingebetteten Datenbank automatisch mit der nächst höheren Ganzzahl versehen werden kann.



Die Feldeigenschaft des Feldes "ID" wird bearbeitet. Für dieses Feld wird die automatische Einsetzung von aufsteigenden Zahlenwerten aktiviert: **Feldeigenschaften → Auto-Wert → Ja**.





Durch die Einstellung des Auto-Wertes sollte beim Verlassen der Feldtypenauswahl auf dem Zeilenkopf ein Schlüsselsymbol zu sehen sein. Das zeigt an, dass dieses Feld den Primärschlüssel der Tabelle bildet. Wird nicht der Auto-Wert gewählt, so kann der Primärschlüssel auch über das Kontextmenü der Maus (**rechte Maustaste → Primärschlüssel**) ausgewählt werden.

Für den Geburtstag ist noch der **Feldtyp → Datum** einzustellen. Dadurch wird gewährleistet, dass nur gültige Datumseingaben aufgenommen werden und außerdem mit Hilfe des Datums Sortierungen durchgeführt oder z. B. das Alter berechnet werden kann.

Welche Felder müssen auf jeden Fall einen Inhalt haben? Diese Tabelle macht ohne Inhalt im Feld "Nachname" und "Datum" keinen Sinn. Schließlich soll die Person erkannt werden und einer Altersklasse zugeordnet werden können. Bei "Nachname" und "Datum" sollte also **Feldeigenschaften → Eingabe erforderlich → Ja** gewählt werden.

Die Tabelle wird nun unter dem Namen "Teilnehmer" abgespeichert. Anschließend können Daten eingegeben werden. Im Feld "ID" ist die Eingabe nicht nötig. Sie wird automatisch beim Abspeichern des Datensatzes erledigt.

Tipp

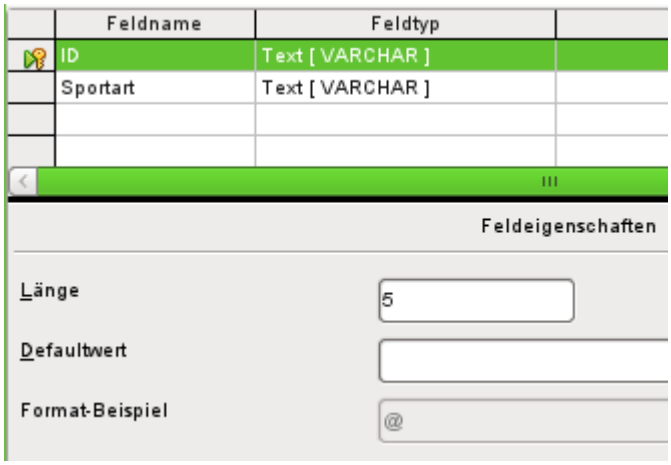
Bei der intensiveren Nutzung von Base vor allem mit Makros kann es zu Verwechselungen von Tabellenbezeichnungen, Abfragebezeichnungen, Formularbezeichnungen und Berichtsbezeichnungen führen. Um hier eine deutlichere Unterscheidung zu ermöglichen, werden häufig Kürzel vor die Namensbezeichnungen gesetzt:
tbl_Name > Tabellenbezeichnung («tbl» steht für «table»)
viw_Name > Ansichtsbezeichnung («viw» steht für «view»)
qry_Name > Abfragebezeichnung («qry» steht für «query»)
frm_Name > Formularbezeichnung («frm» steht für «form»)
rpt_Name > Berichtsbezeichnung («rpt» steht für «report»)
Die für dieses Handbuch erstellten Beispieldatenbanken folgen dieser Namensgebung nicht. Eine entsprechende Benennung wäre bei umfangreichen Datenbanken von Anfang an sinnvoll. Die umfangreichsten Datenbanken dieses Handbuches sind aber bereits entstanden, bevor dieser Vorschlag zur Benennung auftauchte.

Hinweis

Die Datenbankdatei ist ein gepacktes Verzeichnis von einzelnen Dateien. Das Speichern eines einzelnen Elementes wie der Tabelle wird deshalb nicht direkt in die Datenbankdatei selbst geschrieben. Deswegen muss nach der Erstellung von Tabellen, Abfragen, Formularen und Berichten anschließend auch der Speicherbutton für die Datenbankdatei selbst betätigt werden.
Einzig das Abspeichern von eingegebenen Daten funktioniert beim Verlassen der Datenzeile automatisch.

Die Teilnehmer können jetzt eingegeben werden. Folgende Informationen fehlen allerdings auf den ersten Blick:

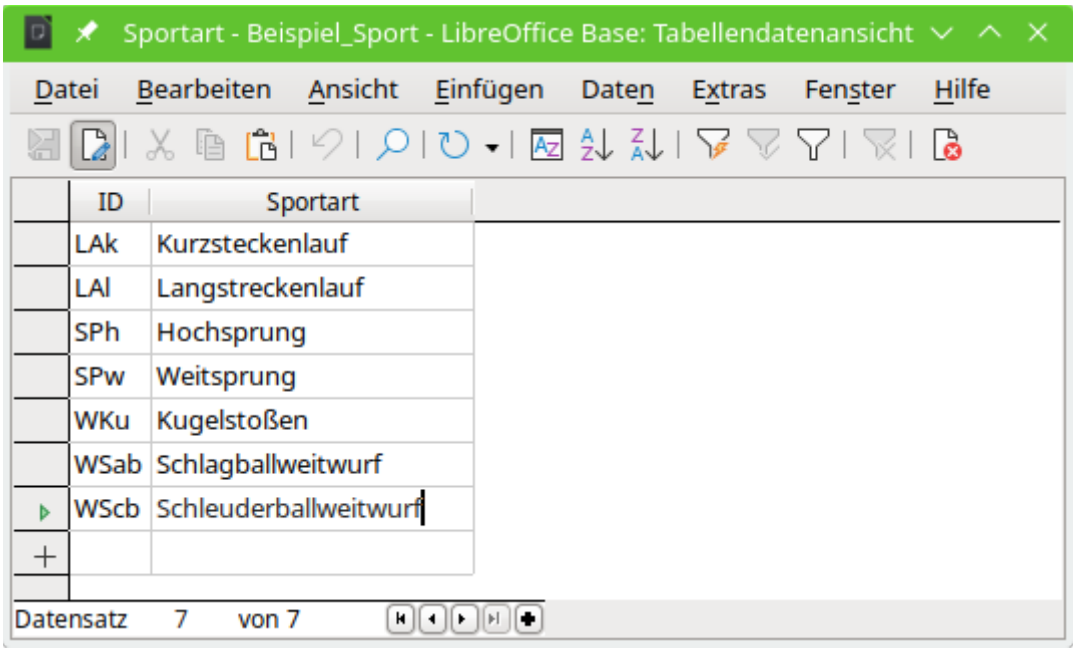
- Eine Liste der Sportarten, die die Teilnehmer belegen wollen.
- Für Wettkämpfe die Unterscheidung von Teilnehmern und Teilnehmerinnen.



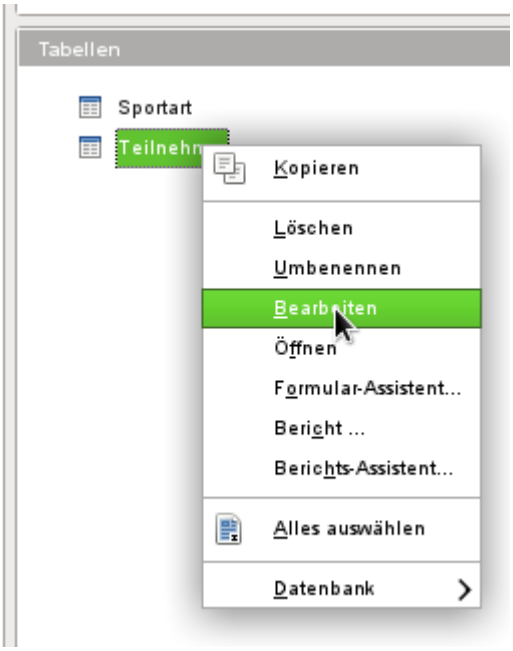
Es wird eine Tabelle "Sportarten" erstellt. Da nicht so sehr viele unterschiedliche Sportarten in Frage kommen, wird hier nicht der automatisch hoch zählende Primärschlüssel gewählt. Stattdessen wird der **Feldtyp → Text** belassen, aber auf 5 Zeichen begrenzt. Die 5 Zeichen reichen aus, um damit auch passende Kürzel für die Sportarten zu finden.

Der Feldtyp «Text» hat den Vorteil, dass schon bei einem Blick auf den Primärschlüssel "ID" klar wird, um welche (ausgeschriebene) "Sportart" es sich handelt. Das Feld "Sportart" wird über **Feldeigenschaften → Eingabe erforderlich → Ja** zum Pflichtfeld erklärt.

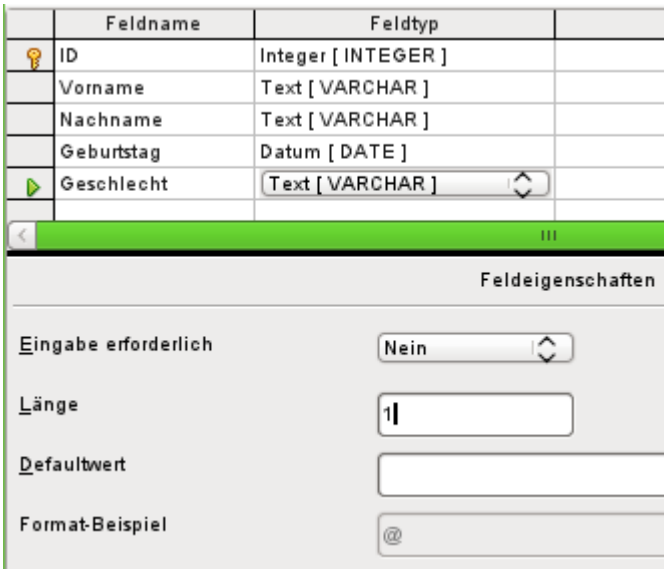
Anschließend kann diese Tabelle zur Eingabe von Daten geöffnet werden (Doppelklick auf die Tabelle oder **rechter Mausklick → Öffnen**).



So könnte dann der Inhalt der Tabelle nach der Eingabe von Daten aussehen. Ohne Daten in dieser Tabelle stehen nachher in dem Formular keine Sportarten zur Auswahl.



Die Tabelle "Teilnehmer" soll noch einmal zum Bearbeiten, nicht zur Dateneingabe, geöffnet werden. Dies geschieht über das Kontextmenü der Tabelle.



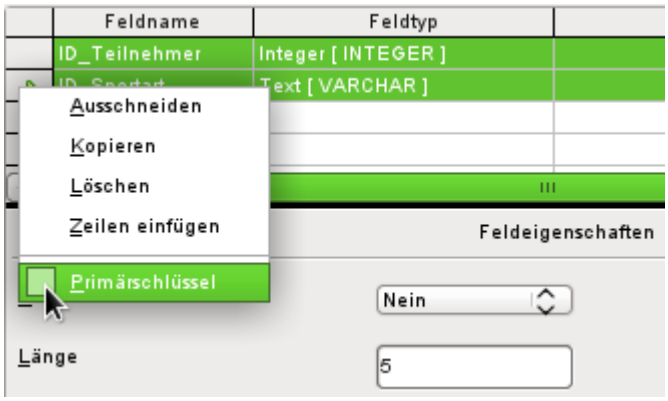
Der Tabelle wird das Feld "Geschlecht" hinzugefügt. Ein neues Feld kann mit der grafischen Benutzeroberfläche nur am Ende der Tabelle hinzugefügt werden. Über SQL ist auch die Möglichkeit gegeben, neue Felder an einer bestimmten Position einzufügen.

Die Länge des Textes in diesem Feld beschränkt sich auf einen Buchstaben. Es reicht also 'm' und 'w', ggf. auch 'd' als Eingabe.

Auch diese Eingabe ist notwendig, damit bei den Wettbewerben nach dem Geschlecht unterschieden werden kann. Mit **Feldeigenschaften → Eingabe erforderlich → Ja** soll das abgesichert werden.

Hinweis

Die nachträgliche Einstellung eines Feldes auf **Eingabe erforderlich → Ja** setzt voraus, dass entweder noch keine Daten in der Tabelle sind oder alle Datensätze für das entsprechende Feld mit einer Eingabe versehen sind. Gegebenenfalls müssen also zuerst sämtliche Datensätze aus "Teilnehmer" mit einem Eintrag bei "Geschlecht" versehen werden.



Irgendwie müssen die beiden Tabellen verbunden werden, so dass jedem Teilnehmer **mehrere Sportarten** und jeder Sportart **mehrere Teilnehmer** zugeordnet werden können. Dies geschieht durch eine Tabelle, in der die Werte der beiden Primärschlüssel der Tabellen "Teilnehmer" und "Sportart" abgespeichert werden sollen. Da nicht mehr als die Kombination dieser Felder gespeichert werden soll, sind beide Felder zusammen der Primärschlüssel für diese Tabelle. Um beiden Feldern einen Primärschlüssel zuzuweisen, wird zunächst am Zeilenkopf eine Zeile markiert, danach die Großschreibtaste (*Umschalt* oder *Shift*) betätigt und die zweite Zeile

zusätzlich markiert. Über das Kontextmenü der Maus lässt sich, wenn die Taste *Umschalt* losgelassen wurde, jetzt der Primärschlüssel festlegen.

Damit wirklich die entsprechenden Werte aus "Teilnehmer" und "Sportart" aufgenommen werden können, müssen die Felder genau dem Feldtyp entsprechen, den sie speichern sollen. "ID_Teilnehmer" muss also den Feldtyp «Integer» haben. "ID_Sportart" muss den Feldtyp «Text» haben und außerdem, wie das Feld "ID" aus der Tabelle "Sportart", auf 5 Zeichen begrenzt werden.

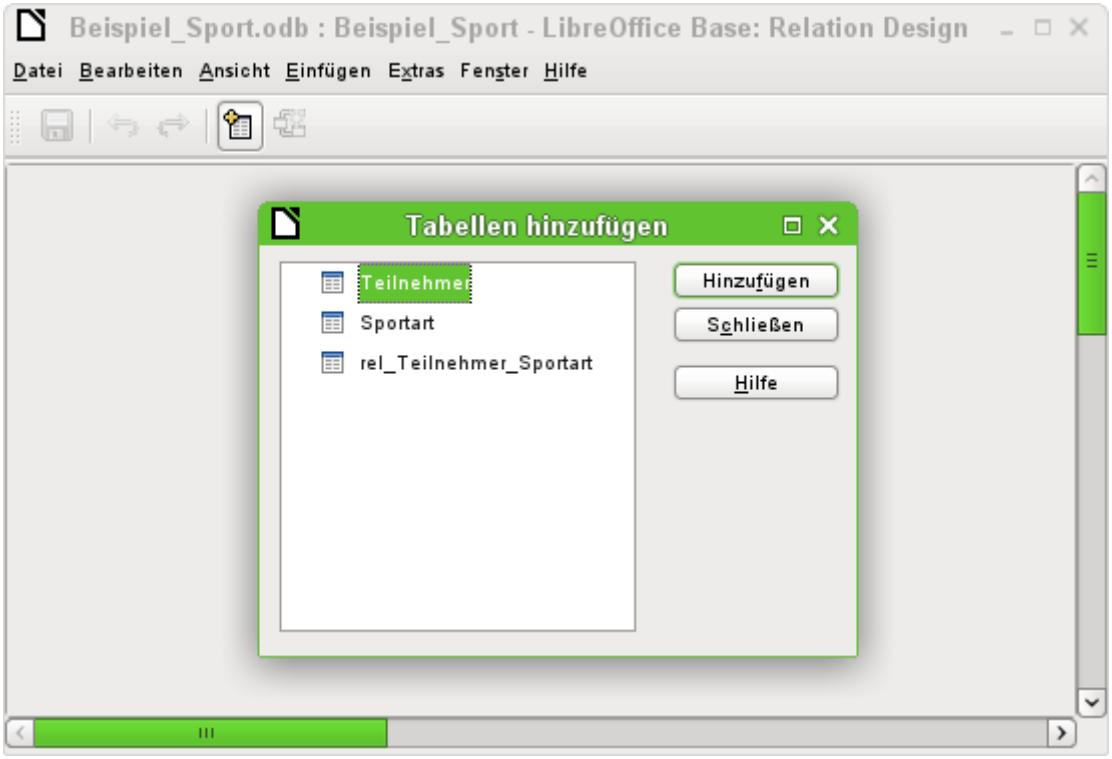
Die Tabelle wird unter dem Namen "rel_Teilnehmer_Sportart" abgespeichert.

Tipp

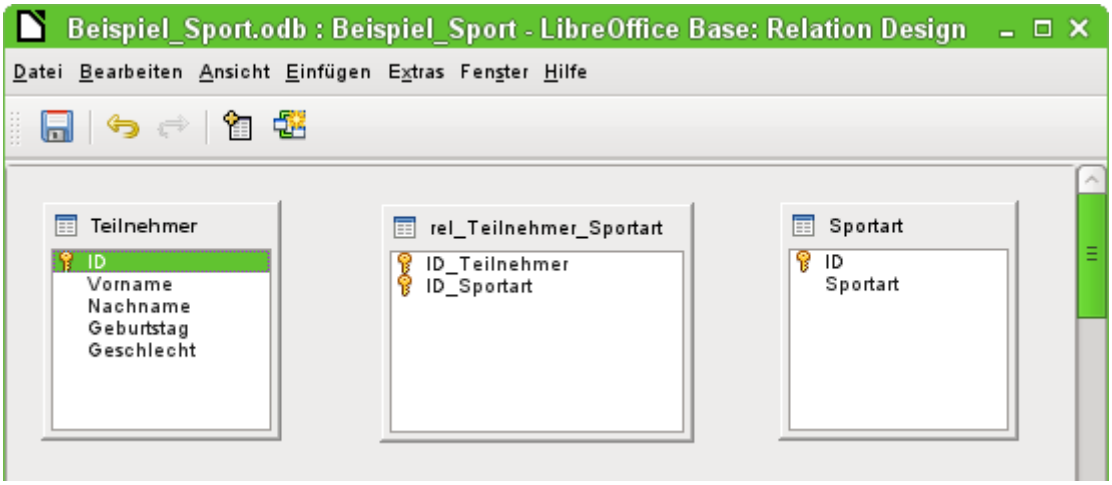
In diese Verbindungstabelle würden auch z.B. die Ergebnisse eines Wettkampfes gehören. Sollen allerdings mehrere Wettkämpfe abgehalten werden, so ist zusätzlich ein Wettkampfdatum in den gemeinsamen Primärschlüssel mit aufzunehmen.

Sind die Tabellen fertiggestellt, so sollte eine entsprechende Verbindung der Tabellen fest definiert werden. So kann verhindert werden, dass in der Tabelle "rel_Teilnehmer_Sportart" z. B. eine Nummer für einen Teilnehmer auftaucht, die gar nicht in der Tabelle "Teilnehmer" verzeichnet ist.

Über **Extras → Beziehungen** wird das Fenster für die Beziehungsdefinition geöffnet.

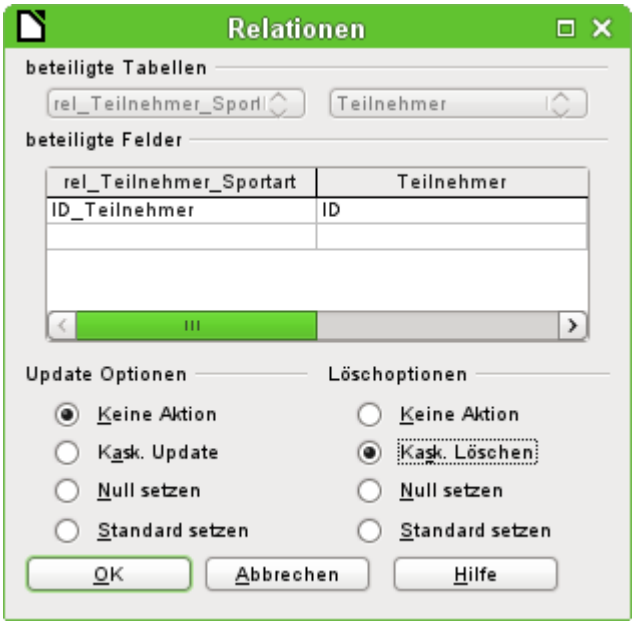


Für die Beziehungsdefinition sind alle bisher erstellten Tabellen notwendig. Die einzelnen Tabellen werden markiert und über Hinzufügen in den Beziehungsentwurf aufgenommen. Anschließend wird der Dialog «Tabellen hinzufügen» geschlossen.



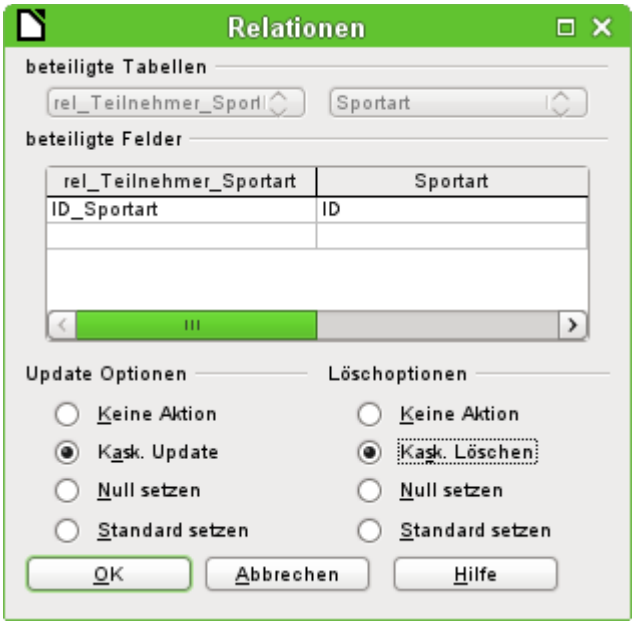
Bei den hinzugefügten Tabellen sind alle Felder aufgeführt. Die Primärschlüsselfelder sind zusätzlich mit einem Schlüsselsymbol gekennzeichnet. Die vorgesehenen Rechtecke für die Tabellen können übrigens beliebig verschoben und in der Größe verändert werden.

Das Feld "Teilnehmer"."ID" wird mit der linken Maustaste markiert. Die Maustaste wird gehalten und die Maus auf das Feld "rel_Teilnehmer_Sportart"."ID_Teilnehmer" gezogen. Der Cursor zeigt eine Verknüpfung an. Die Maustaste wird über dem Feld "rel_Teilnehmer_Sportart"."ID_Teilnehmer" los gelassen. Es erscheint der folgende Dialog zur Definition der Beziehung («Relation»).



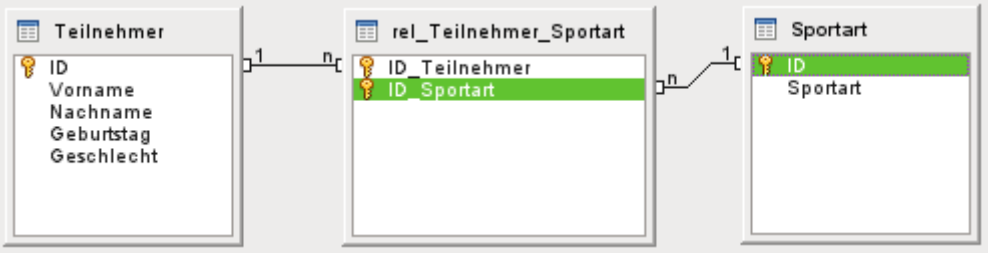
Das Feld "rel_Teilnehmer_Sportart"."ID_Teilnehmer" soll nicht verändert werden, wenn "Teilnehmer"."ID" verändert würde. Das ist die Standardeinstellung. "Teilnehmer"."ID" wird sowieso nicht verändert, da es sich um ein automatisch hoch zählendes Feld handelt und dort kein Eingriff vorgenommen werden soll.

Der Datensatz der Tabelle "rel_Teilnehmer_Sportart" soll gelöscht werden, wenn "ID_Teilnehmer" gleich "Teilnehmer"."ID" ist und "Teilnehmer"."ID" gelöscht wird. Wenn also aus der Teilnehmer-Tabelle ein Teilnehmer entfernt wird, dann werden alle entsprechenden Datensätze aus der Tabelle "rel_Teilnehmer_Sportart" entfernt. Dieses Vorgehen wird als «kaskadierendes Löschen» bezeichnet.



Im nächsten Schritt werden "rel_Teilnehmer_Sportart"."ID_Sportart" und "Sportart"."ID" durch Ziehen der Maus bei gedrückter linker Maustaste verbunden. Auch hier soll ein Datensatz gelöscht werden, wenn die entsprechende Sportart gelöscht wird.

Für die Datenänderung bei "Sportart"."ID" ist allerdings eine andere Variante gewählt worden. Wird "Sportart"."ID" geändert, so wird entsprechend "rel_Teilnehmer_Sportart"."ID_Sportart" angepasst. So kann gegebenenfalls das Kürzel von maximal 5 Zeichen für eine Sportart problemlos angepasst werden, auch wenn bereits viele Datensätze in der Tabelle "rel_Teilnehmer_Sportart" stehen. Dieser Vorgang wird als «kaskadierendes Update» bezeichnet.

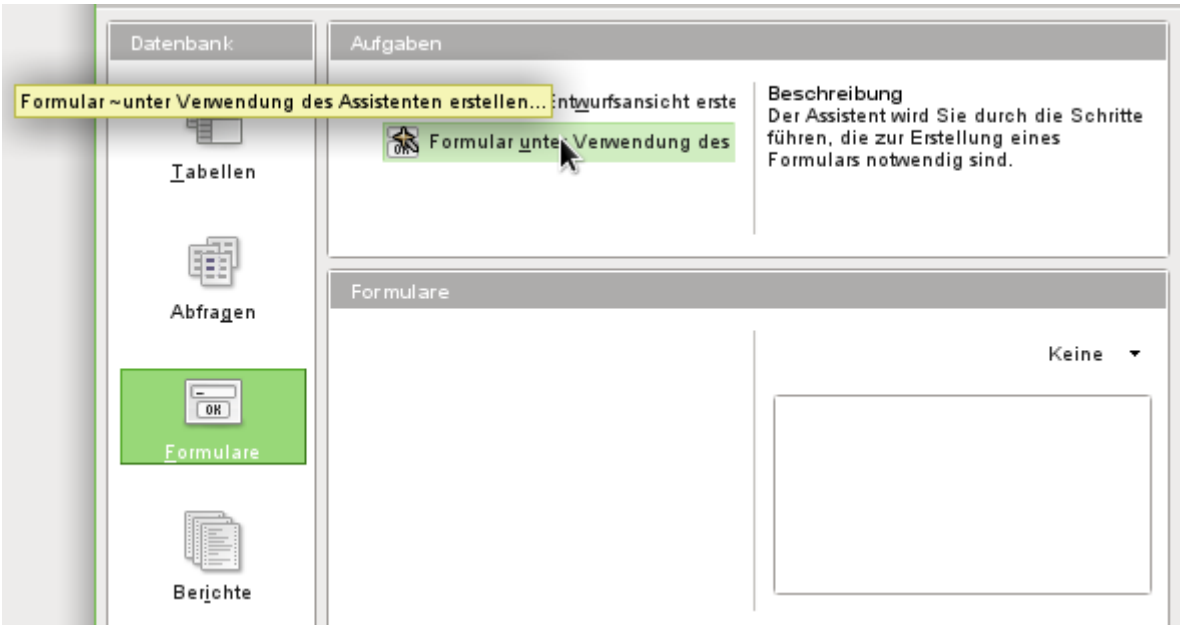


Die Felder sind nun fertig verbunden. An den Enden der Verbindungen erscheint jeweils eine «1» und ein «n». Ein Teilnehmer kann mehrmals in der Tabelle "rel_Teilnehmer_Sportart" auftauchen. Eine Sportart kann ebenfalls mehrmals in der Tabelle "rel_Teilnehmer_Sportart" auftauchen. Nur die Kombination von Teilnehmer und Sportart gibt es in der Tabelle nur maximal ein Mal. Aus zwei 1:n-Beziehungen wird so über die dazwischen gesetzte Tabelle "rel_Teilnehmer_Sportart" eine n:m-Beziehung.

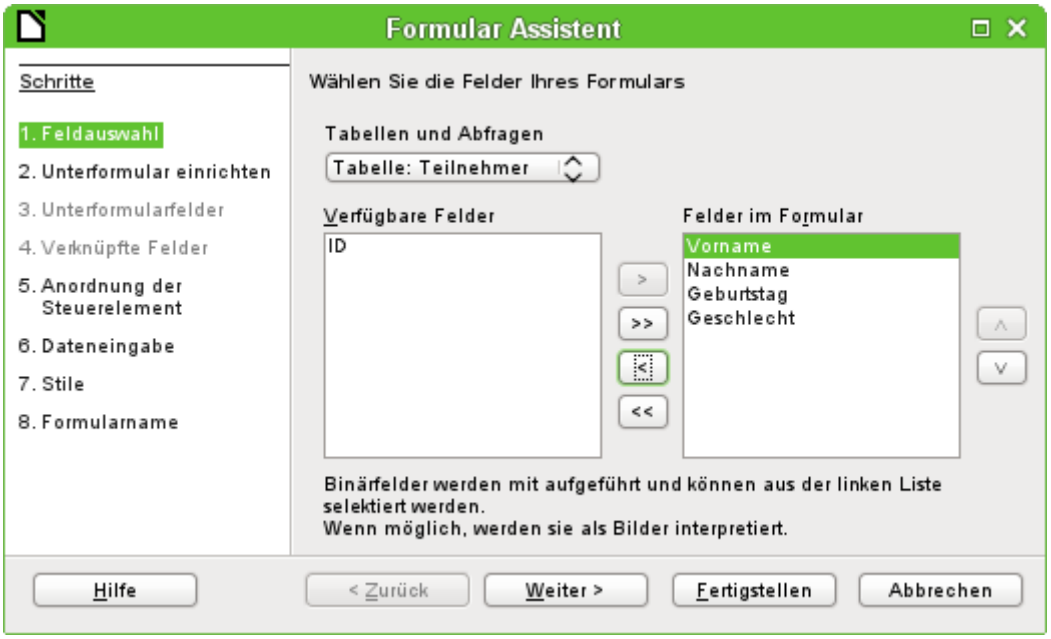
Solch eine Tabellenkonstruktion lässt sich nur schlecht über die Eingabe in Tabellen mit Inhalt füllen. Es müssten alle 3 Tabellen geöffnet sein, wenn einem Teilnehmer eine Sportart zugewiesen werden soll. In der Tabelle "Teilnehmer" muss "Teilnehmer"."ID" gesucht werden und nach "rel_Teilnehmer_Sportart" übertragen werden. In der Tabelle "Sportart" muss "Sportart"."ID" gesucht und entsprechend in "rel_Teilnehmer_Sportart" eingetragen werden. Zu umständlich: Ein Formular löst das eleganter.

Eingabeformular

Formulare können direkt in der Entwurfsansicht oder mit Hilfe des Assistenten erstellt werden. Die Erstellung mit dem Assistenten greift dabei auch alten Hasen so gut unter die Arme, dass diese eben schnell den Assistenten durchlaufen lassen und anschließend ein erst einmal vom Grund her erstelltes Formular ihren Wünschen anpassen. Das ist häufig der deutlich zeitsparendere Weg.



Zuerst wird unter **Datenbank** der Arbeitsbereich für **Formulare** gewählt. Dann wird unter **Aufgaben → Formular unter Verwendung des Assistenten erstellen …** ausgeführt.



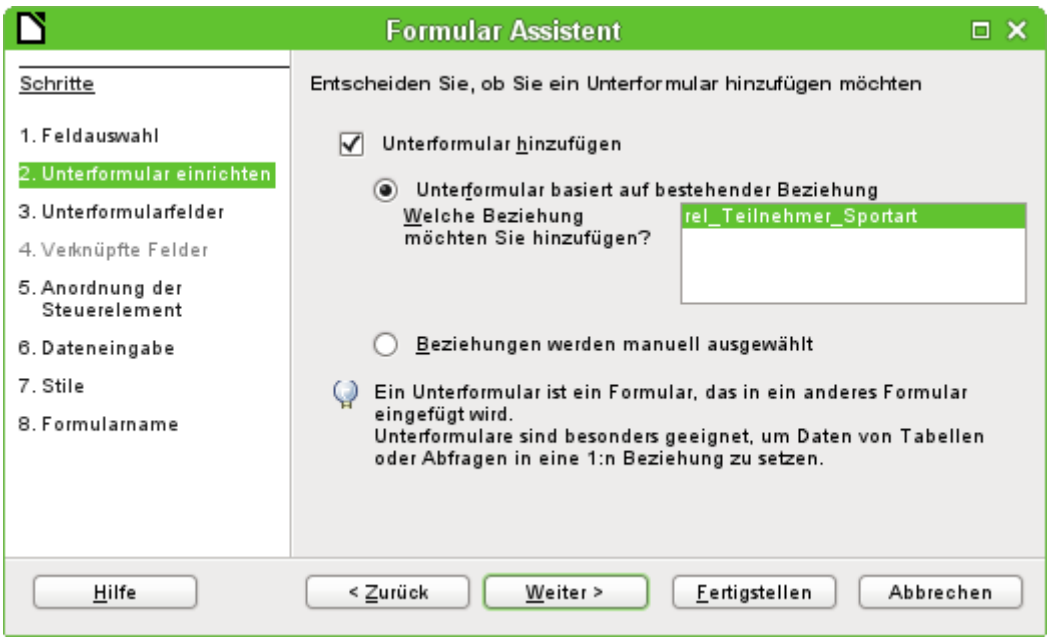
In der Hauptsache sollen in die Tabelle "Teilnehmer" Daten eingegeben werden. Die Tabelle "Sportarten" wird direkt mit den wenigen notwendigen Sportarten bestückt und nur selten ergänzt.

Von der Tabelle "Teilnehmer" werden alle Felder bis auf das Primärschlüsselfeld "ID" benötigt. Das Primärschlüsselfeld wird ja automatisch mit einem entsprechend unverwechselbarem Wert gefüllt.

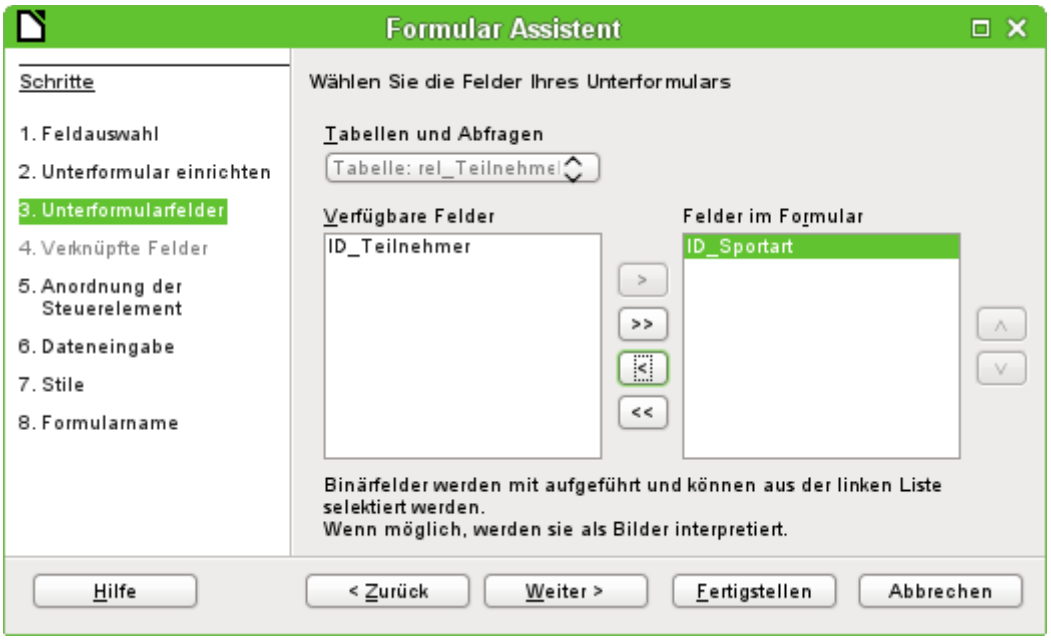
Die Felder im Formular werden über die Pfeiltasten zwischen den beiden Auswahlkästchen bestückt.

Hinweis

~~Firebird~~ erkennt zur Zeit noch nicht die in den Beziehungen erstellten Tabellenverknüpfungen. Für Firebird muss deshalb darauf geachtet werden, dass auch das Feld "ID" in das Formular übernommen wird. Die Verknüpfungen zum folgenden Unterformular werden nicht automatisch erkannt. Auch für die dort auszusuchende Tabelle müssen zuerst einmal alle Felder übernommen werden.

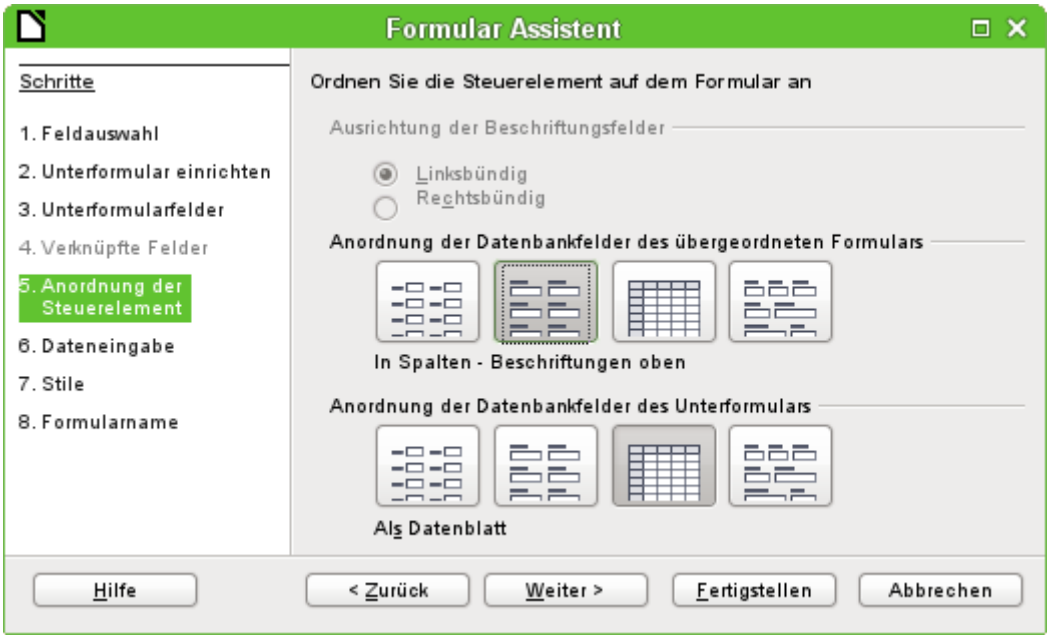


Es soll ein Unterformular eingerichtet werden, in dem die Sportarten dem Teilnehmer zugewiesen werden können. Die Beziehung für das Unterformular wird aus der vorher definierten Beziehung ausgelesen. Allerdings muss sie hier noch durch einen Klick auf die Tabelle "rel_Teilnehmer_Sportart" bestätigt werden.



Aus der Tabelle "rel_Teilnehmer_Sportart" wird nur das Feld "ID_Sportart" benötigt. Das Feld "ID_Teilnehmer" wird über die Verbindung von Hauptformular zu Unterformular mit dem Wert versehen, der in der Teilnehmer-Tabelle für den Primärschlüssel des aktuellen Datensatzes steht.

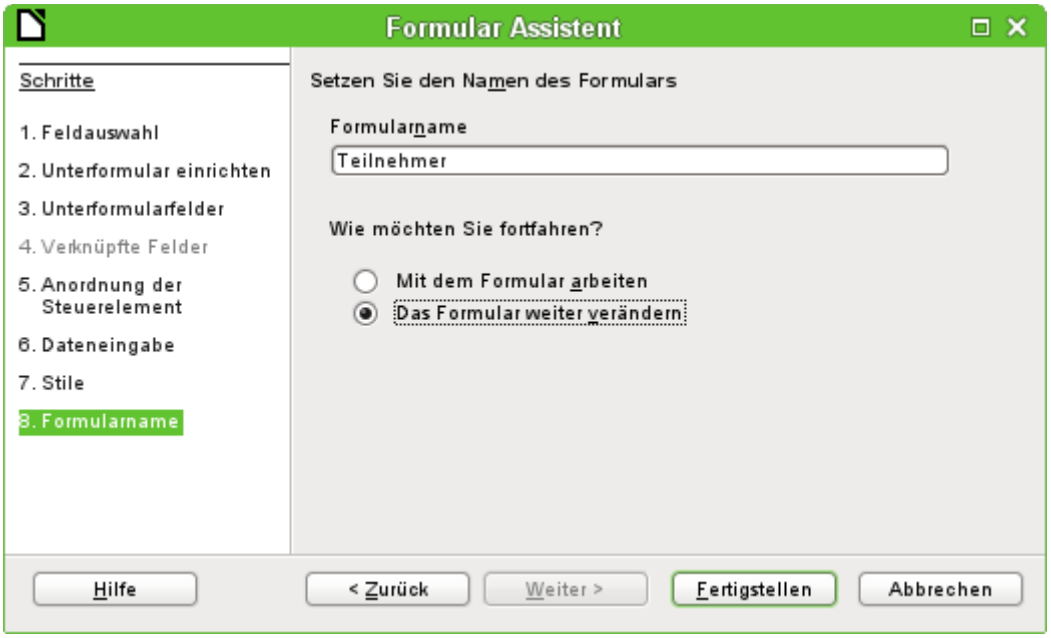
Dass diese Verknüpfung bereits geregelt ist, ist auch an dem 4. Schritt des Assistenten zu erkennen: Der Schritt **Verknüpfte Felder** ist inaktiv.



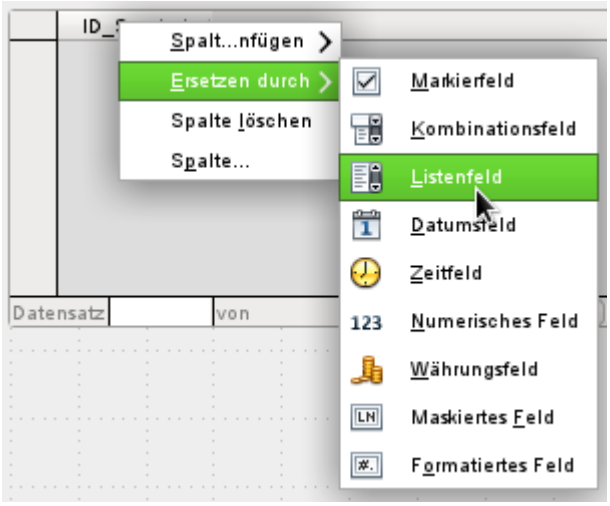
Wie die Elemente im Hauptformular und wie im Unterformular angeordnet sind, ist letztlich egal. Damit aber die Zuordnung von Daten auch für ungeübte einfach zu ersehen ist, sollte für das Hauptformular in unserem Beispiel nicht das Tabellenkontrollfeld gewählt werden. Im Unterformular soll das Feld später alle Sportarten des Teilnehmers anzeigen. Deswegen an dieser Stelle am besten das Tabellenkontrollfeld belassen.

Schritt 6 sollte so bleiben, wie er vorgesehen ist, also **Dateneingabe → Das Formular zeigt existierende Daten an**. Damit ist dann eine Neueingabe ebenso möglich wie eine Veränderung bereits bestehender Daten.

Schritt 7 ist reine Geschmackssache. Nur Vorsicht: manche Stile bergen unverhoffte kontrastarme Darstellungen vor allem in Tabellenkontrollfeldern. Hier muss dann gegebenenfalls die Schriftfarbe der Tabellenkontrollfelder nachjustiert werden.



Mit dem Formular soll noch nicht gearbeitet werde, weil das Unterformular noch die Eingabe des Kürzels für die Sportarten erfordert. Hier wäre eine Auswahl der Sportarten nach dem kompletten Namen wünschenswert. Deswegen: **Das Formular weiter verändern**.



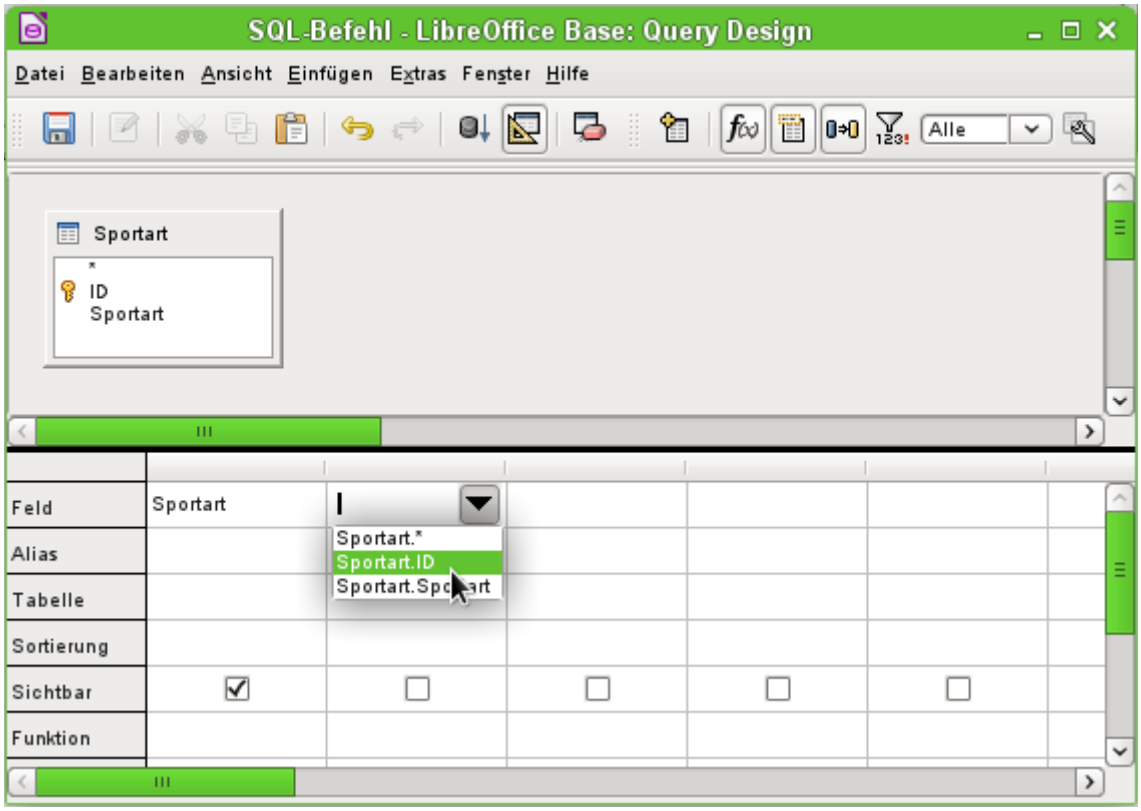
Zuerst wird mit der rechten Maustaste auf den Tabellenkopf des Tabellenkontrollfeldes geklickt. Dort steht «ID_Sportart». Im Kontextmenü wird **Ersetzen durch → Listenfeld** ausgewählt.

Anschließend soll das Listenfeld bearbeitet werden, damit es auch die entsprechenden Daten anzeigen kann. Wieder über das Kontextmenü geht es zu **Spalte...**

Es öffnen sich die Eigenschaften des ausgewählten Listenfeldes. **Daten → Art des Listeninhaltes → SQL** soll gewählt werden. Mit Hilfe von SQL (StructuredQueryLanguage – Standardabfrage-sprache für Datenbanken) soll das Feld seinen Inhalt aus der Tabelle "Sportart" erhalten.

Ist **SQL** ausgewählt, so erscheint in der Zeile **Listeninhalt** rechts ein Button mit 3 Punkten ... . Wird dieser Button gedrückt, so öffnet sich der Editor für die Erstellung von Abfragen. Die im Folgenden zu erstellenden Abfrage wird hier nur zusammengestellt und schließlich in dem Listenfeld selbst gespeichert.

Im Dialog «Tabelle oder Abfrage hinzufügen» erfolgt ein Klick auf **Sportart → Hinzufügen → Schließen**.



Für die erste Spalte wird in dem Abfrage-Editor das Feld "Sportart" ausgewählt. Dieses Feld soll im Listenfeld sichtbar sein.

Die zweite Spalte soll mit dem Feld "ID" bestückt werden. Dieses Feld gibt nachher seinen Wert an die Tabelle weiter, auf der das Unterformular basiert. Es werden also die ausgeschriebenen Worte dargestellt und die passenden Kürzel abgespeichert.

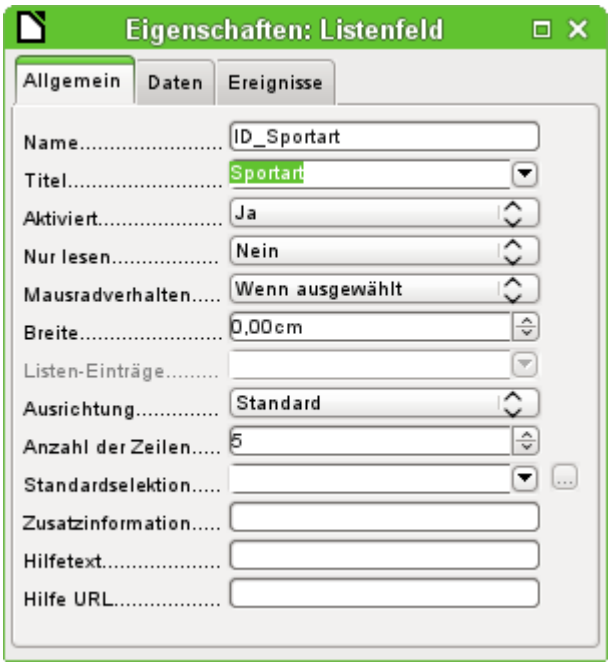
Die Abfrage wird mit dem Speicherbutton in die Eigenschaften des Listenfeldes übertragen. Der Editor wird anschließend geschlossen.

Im Listeninhalt ist jetzt der folgende Text zu lesen:

```
001 SELECT "Sportart", "ID" FROM "Sportart"
```

Das ist der SQL-Code, der aus der Betätigung des Editors erwachsen ist. Umgangssprachlich ausgedrückt heißt der Code: Wähle von der Tabelle "Sportart" als erstes die "Sportart" und als zweites den dazugehörigen Schlüsselwert aus.

Diese Abfrage stellt das Minimum dar, das ausgewählt werden sollte. Natürlich könnte auch eine Sortierung eingebaut werden. Mit geschickter Wahl der in "ID" abgespeicherten Kürzel ergibt sich aber bereits eine brauchbare Zusammenstellung der Sportarten. Wird nämlich keine Sortierung vorgegeben, so erfolgt die Sortierung immer nach dem Primärschlüsselfeld. Um später in dem Listenfeld Sportarten zu sehen, müssen diese natürlich zuerst einmal in die Tabelle "Sportart" entsprechend eingegeben werden.



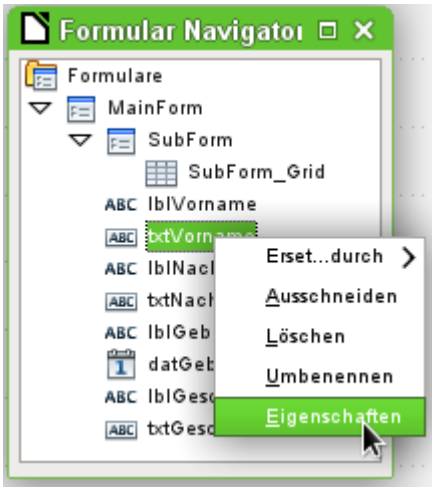
Nun soll noch der Titel geändert werden, der für das Feld "ID_Sportart" eingetragen wurde. Es wird zwar weiterhin "ID_Sportart" abgespeichert. Sichtbar ist aber "Sportart". Deswegen: **Allgemein → Titel → Sportart**.

Soll z.B. das Design bei anderen Feldern angepasst werden, so lassen sich diese am besten über den Formularnavigator ansteuern. Werden Felder direkt angeklickt, so werden nicht nur die Eingabefelder, sondern auch die Beschriftungsfelder markiert. Durch den Assistenten wurden sie zu Gruppen zusammengefasst. Das erfordert dann weitere Aktionen im Kontextmenü der Auswahl.

Der Formularnavigator wird über die Symbolleiste «Formular-Entwurf» am unteren Bildschirmrand gestartet.

Hinweis

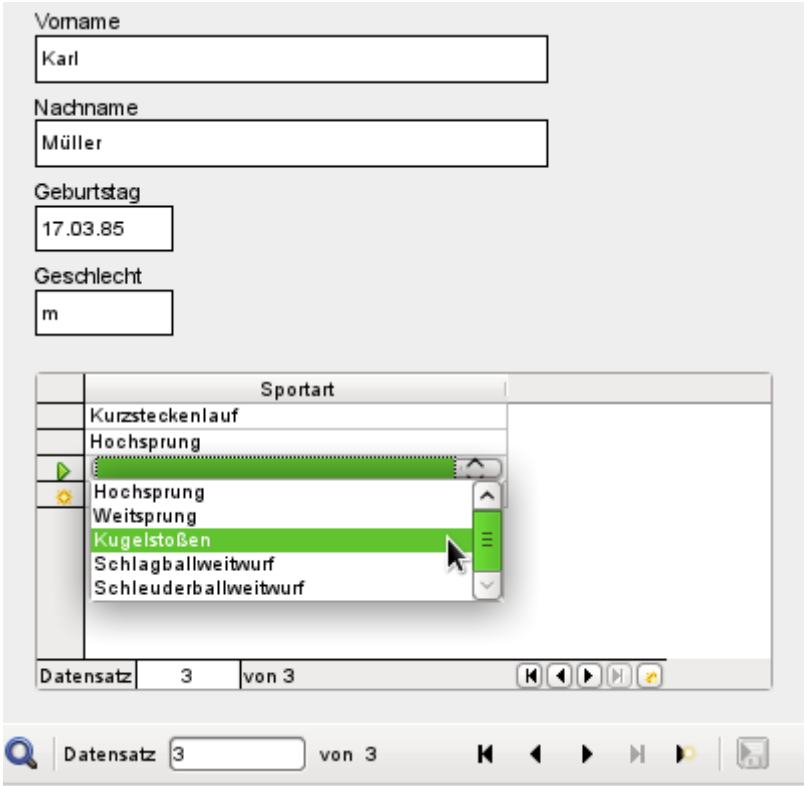
Manchmal kann es vorkommen, dass die Symbolleisten für die Erstellung von Formularen beim Bearbeiten der Formulare nicht korrekt geöffnet werden. Der Formularnavigator ist dann nicht zu finden.
Die Symbolleisten lassen sich über
Ansicht → Symbolleisten → Formular-Entwurf sowie **Formular-Steuerelemente** einblenden. Sollten die Symbolleisten danach auch bei der Dateneingabe sichtbar sein, so muss hier entsprechend wieder die Ansicht ausgeschaltet werden.



Über den Formularnavigator kann jedes Feld einzeln aufgesucht werden. Im Kontextmenü sind dann die Eigenschaften des Feldes aufrufbar. Die Eigenschaften werden nach dem Verlassen der Eigenschaften automatisch gespeichert. Es kann auch bei geöffnetem Eigenschaftsdialog von einem Feld zum anderen gesprungen werden. Hierbei wird auch der jeweilige Zwischenstand gespeichert.

Wenn die Gestaltung nun beendet wurde, kann das Formular gespeichert und geschlossen werden. Auch eine Speicherung der Base-Datei ist jetzt wieder sinnvoll.

Wird jetzt das Formular zur Eingabe von Daten aufgerufen, dann könnte es ungefähr wie folgt aussehen. In das folgende Formular wurde zum Test ein Datensatz eingetragen. Nach der Eingabe des Geschlechtes wurde erst einmal der Button *Speichern* betätigt. Dann konnten auch Sportarten gewählt werden.



Bei der Nutzung des Formulars fallen einige Unannehmlichkeiten auf:

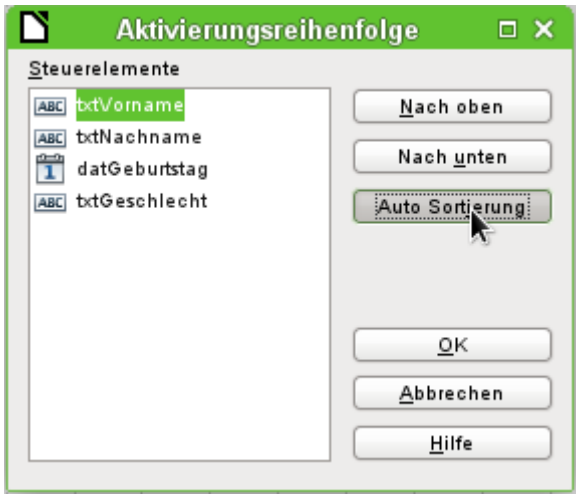
1. Das Unterformular mit den Sportarten ist nicht direkt erreichbar. Wird mit dem Tabulator navigiert, so springt der Tabulator nach der Eingabe des Geschlechtes direkt zum nächsten Teilnehmerdatensatz.
2. Die Navigationsleiste zeigt, wenn sie im Unterformular steht, die Datensatzzahl des Unterformulars an. Sie sollte besser nur durch das Hauptformular navigieren.
3. Das Geschlecht wird zur Zeit auswendig je nach Benutzervorliebe eingegeben. In der Tabelle ist lediglich die Länge des Feldes auf 1 Zeichen begrenzt worden. Hier muss also eine sicherere Eingabe gewährleistet werden.

Zur Lösung dieser Probleme muss das Formular nicht für die Dateneingabe, sondern zum Bearbeiten geöffnet werden:

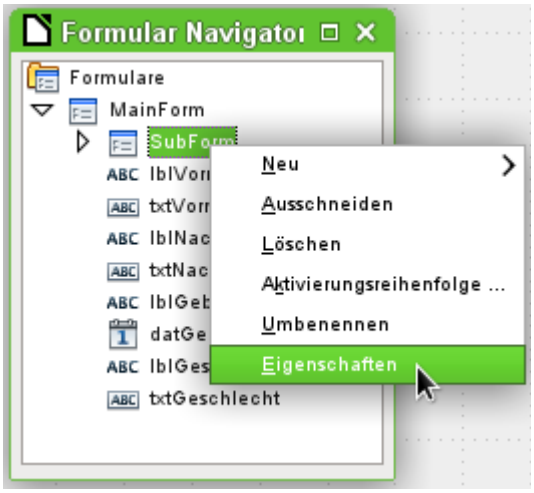
Mit dem Tabulator zum Unterformular

Um mit dem Tabulator nach der Eingabe in das Feld "Geschlecht" nicht gleich weiter zum nächsten Teilnehmer-Datensatz zu rutschen ist einmal die Bearbeitung der **Aktivierungsreihenfolge**, erreichbar über die Symbolleiste «Formular-Entwurf», notwendig.

Mit der Auswahl *Auto Sortierung* erfolgt nicht nur die Sortierung der angezeigten Steuerelemente, sondern auch die automatische Weiterleitung in das Unterformular. Das ist zwar aus dem Dialog nicht ersichtlich, wird aber im Hintergrund so geregelt.



Navigationsleiste des Hauptformulars auch im Unterformular aktivieren



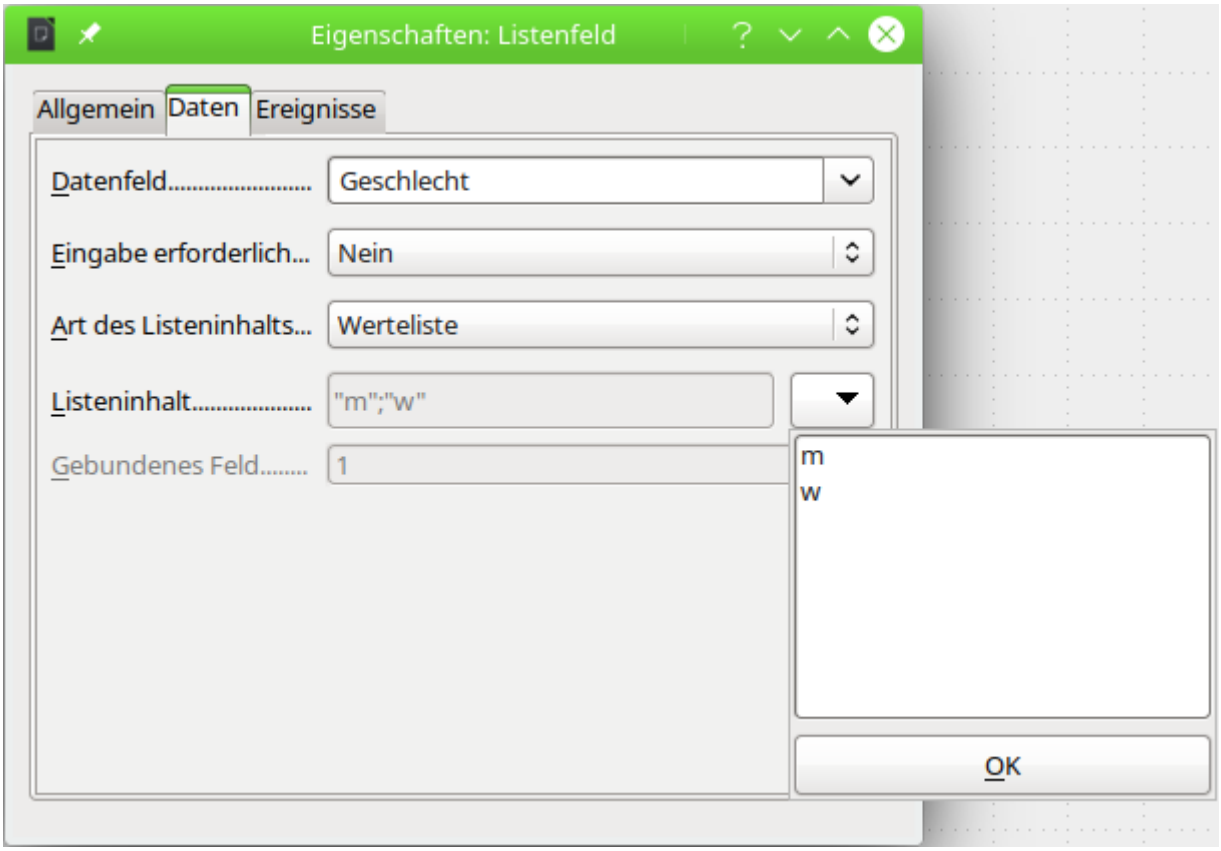
Der Formularnavigator wird geöffnet. Die Eigenschaften des Unterformulars werden aufgesucht.

Unter **Daten → Symbolleiste Navigation** wird der Wert von «Ja» auf «Übergeordnetes Formular» geändert. Anschließend zeigt die Symbolleiste Navigation immer die Nummer des Datensatzes aus der Tabelle "Teilnehmer" (Hauptformular) an.

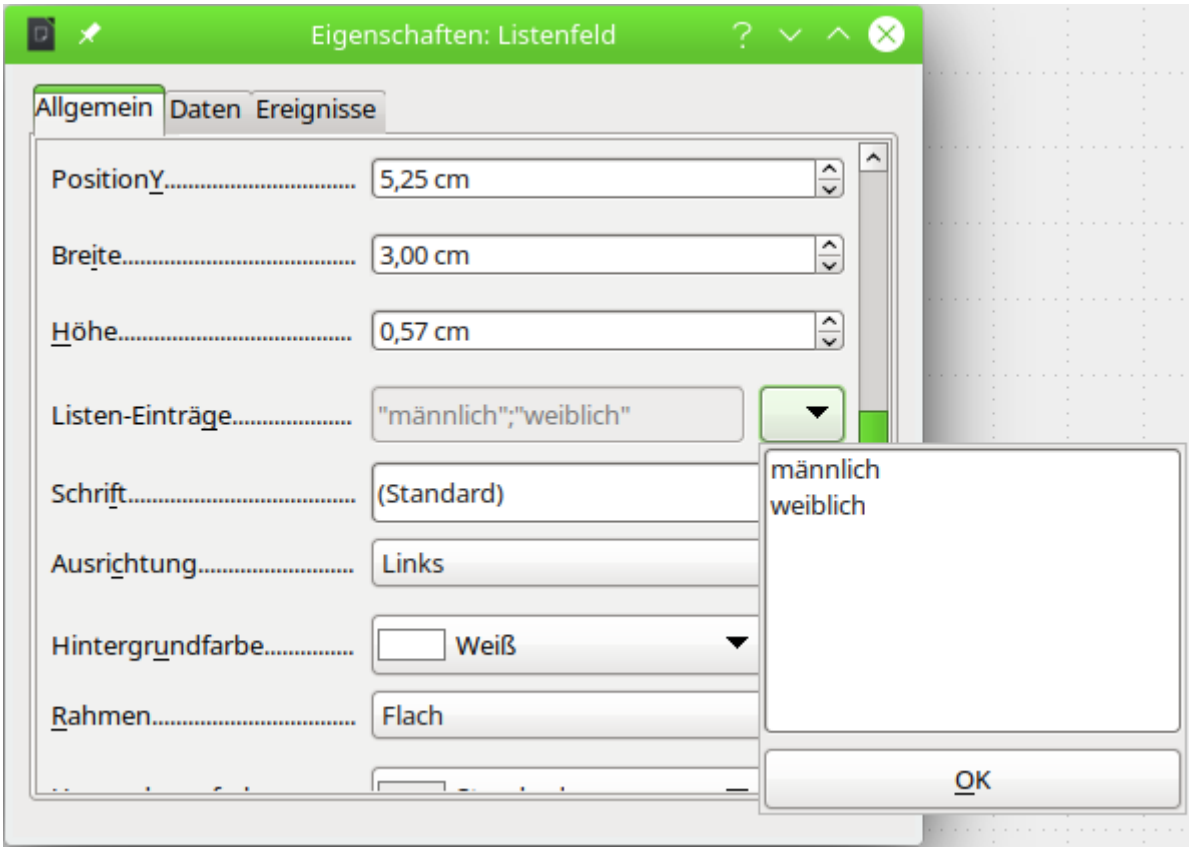
Eingabe eines Kontrollfeldes einschränken

Für die Einschränkung der Eingabe darf das Kontrollfeld kein einfaches Texteingabefeld sein. Im Hauptkapitel Formulare findet sich für die Eingabe des Geschlechtes die Lösung über einen «Gruppierungsrahmen». Hier soll deshalb eine andere Lösung, nämlich noch einmal über ein Listenfeld, vorgestellt werden.

Zuerst wird das Formularfeld für die Eingabe des Geschlechts über den Navigator markiert. Über das mit der rechten Maustaste erreichbare Kontextmenü wird **Ersetzen durch → Listenfeld** ausgewählt. Anschließend wird über das Kontextmenü **Eigenschaften** ausgewählt.



Daten → Art des Listeninhalts bleibt auf dem Wert «Werteliste» stehen. In **Daten → Listeninhalt** werden die Kürzel 'm' und 'w' über den Button ▼ in das Feld untereinander eingegeben. Diese Kürzel sind die Daten, die an die Tabelle "Teilnehmer" weiter gegeben werden.



In **Allgemein → Listen-Einträge** wird das angegeben, was durch das Listenfeld angezeigt werden soll. Das kann auch 'm' und 'w' sein. Es muss auf jeden Fall die gleiche Reihenfolge haben wie die Einträge unter **Daten**.

Jetzt wird noch unter **Allgemein → Aufklappbar → Ja** gewählt. Hierzu muss in den allgemeinen Eigenschaften recht weit unten gesucht werden.

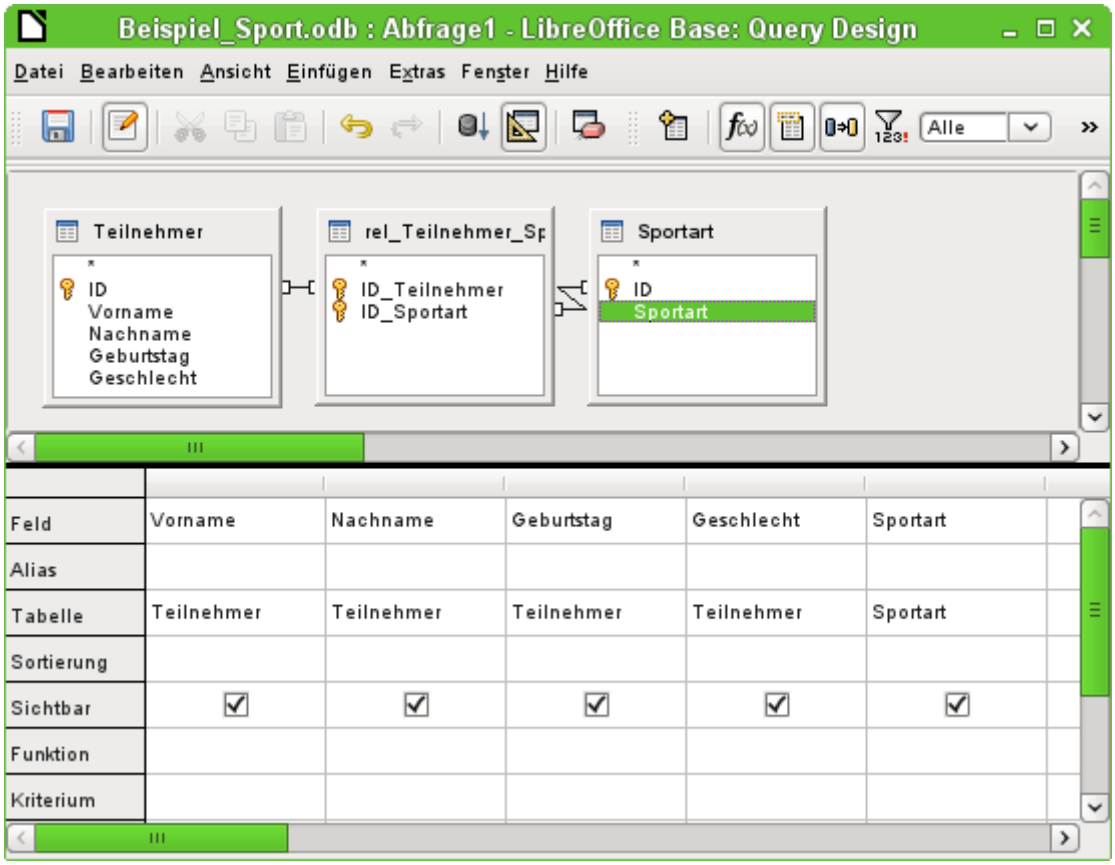
Damit ist das Eingabeformular so weit, dass die auffälligsten Unannehmlichkeiten beseitigt wurden. Das Formular und die Datenbankdatei müssen wieder abgespeichert werden. Die Eingabe von Teilnehmern und Teilnehmerinnen sowie die Zuordnung zu Sportarten kann beginnen.

Damit die folgenden Schritte sinnvoll sind, sollten erst einmal einige Datensätze eingegeben werden. Dabei sollte darauf geachtet werden, dass die Teilnehmer und Teilnehmerinnen vom Alter und von der Sportart auch einmal in Konkurrenz zueinander treten können. Sonst sehen die anschließenden Abfragen und Berichte doch recht nichtssagend aus.

Abfrage

In einer Abfrage können z. B. Inhalte verschiedener Tabellen zusammengefasst werden. Mit dem ersten Zugriff sollen erst einmal alle Teilnehmer mit den Sportarten dargestellt werden, die sie belegt haben.

Der Start dazu geht über **Abfragen → Abfrage in der Entwurfsansicht erstellen …** . Es erscheint ein Dialog zur Tabellenauswahl, aus dem wir alle Tabellen für die Abfrage auswählen. Am übersichtlichsten ist es, wenn dabei die Tabelle "rel_Teilnehmer_Sportarten" als zweite Tabelle ausgewählt wird. Dann sind auch gleich die Verbindungen gut zu erkennen.



Die Felder, die in der Abfrage dargestellt werden sollen, können entweder durch einen Doppelklick auf die Feldnamen in den Tabellen oder durch Auswahl in der Zeile **Feld** eingestellt werden. Bei der Auswahl in der Zeile werden die Felder sämtlicher aufgeführter Tabellen aufgelistet. Um die Felder den Tabellen richtig zuordnen zu können, werden sie in Abfragen nach

"Tabellenname"."Feldname" bezeichnet. Wird statt des Feldnamen ein «*» eingesetzt, so bedeutet dies, dass alle Felder der entsprechenden Tabelle angezeigt werden sollen.

Eine Abfrage sollte grundsätzlich vor dem Abspeichern auch einmal ausgeführt werden um zu sehen, ob sie denn wirklich das erhoffte Ergebnis liefert. Hierzu kann entweder direkt der oben gezeigten Button angeklickt, alternativ auch *F5* gedrückt oder der Weg über **Bearbeiten → Abfrage ausführen** gewählt werden.

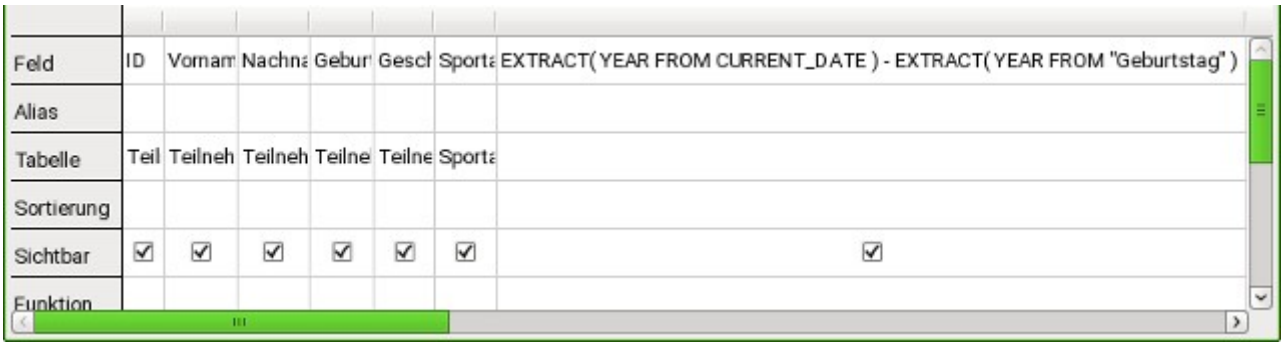
Vorname	Nachname	Geburtstag	Geschlecht	Sportart
Karl	Müller	17.03.85	m	Kurzstecken
Karl	Müller	17.03.85	m	Hochsprung
Karl	Müller	17.03.85	m	Kugelstoßer
Mia	Keller	07.04.78	w	Kurzstecken
Mia	Keller	07.04.78	w	Weitsprung
Mia	Keller	07.04.78	w	Kugelstoßer
Dirk	Anders	28.09.87	m	Kurzstecken
Dirk	Anders	28.09.87	m	Langstrecke
Dirk	Anders	28.09.87	m	Weitsprung

Datensatz 1 von 26

Die Abfrage zeigt jetzt alle Kombinationen der Teilnehmer und Sportarten an. Hat ein Teilnehmer mehrere Sportarten belegt, so werden mehrere Datensätze angezeigt. Teilnehmer ohne Sportarten erscheinen nicht.

Um bei einem Wettkampf unterschiedliche Altersgruppen zusammen zu stellen, ist der Geburtsjahrgang entscheidend. Es kommt darauf an, wie alt eine Person in dem jeweiligen Jahr wird.

Der Geburtsjahrgang kann mit unterschiedlichen Funktionen ermittelt werden. Die folgende Eingabe zeigt die Variante, die für Personen mit Englischkenntnissen vielleicht auch so erschließbar ist:



Feld	ID	Vornam	Nachna	Gebur	Gesch	Sporta	EXTRACT(YEAR FROM CURRENT_DATE) - EXTRACT(YEAR FROM "Geburtstag")
Alias							
Tabelle	Teil	Teilneh	Teilneh	Teilne	Teilne	Sporta	
Sortierung							
Sichtbar	☑	☑	☑	☑	☑	☑	☑
Funktion							

```
001 EXTRACT (YEAR FROM CURRENT_DATE) - EXTRACT (YEAR FROM "Geburtstag")
```

Über **`EXTRACT(YEAR FROM CURRENT_DATE)`** wird das aktuelle Jahr bestimmt. Entsprechend wird auch über **`EXTRACT(YEAR FROM "Geburtstag")`** das Jahr des Geburtstags ermittelt.

Diese und viele andere Funktionen, die mit der eingebauten HSQLDB oder der FIREBIRD-Datenbank zusammen arbeiten, sind im Anhang des Handbuches beschrieben.

	ID	Vorname	Nachname	Geburtstag	Geschlecht	Sportart	EXTRACT(YEAR FROM CURRENT_DATE) - EXTR
▶	0	Karl	Müller	17.03.85	m	Kurzstecken	33
	1	Mia	Keller	07.04.78	w	Kurzstecken	40
	2	Dirk	Anders	28.09.87	m	Kurzstecken	31
	4	Mia	Maria	01.03.08	w	Kurzstecken	10
	5	Evi	Lotta	15.07.07	w	Kurzstecken	11

Datensatz | von 41 *

Wird die Abfrage ausgeführt, so erscheint die entsprechende Berechnung für das Jahr 2014, da diese Abfrage eben in dem entsprechenden Jahr ausgeführt wurde.

Etwas störend dürfte dabei sein, dass der gesamte Code, den wir als Feld eingegeben haben, eben auch in der Spaltenüberschrift erscheint. Dem wird abgeholfen, indem für den Code ein alternativer Name eingegeben wird, unter dem das Ergebnis erscheinen soll.

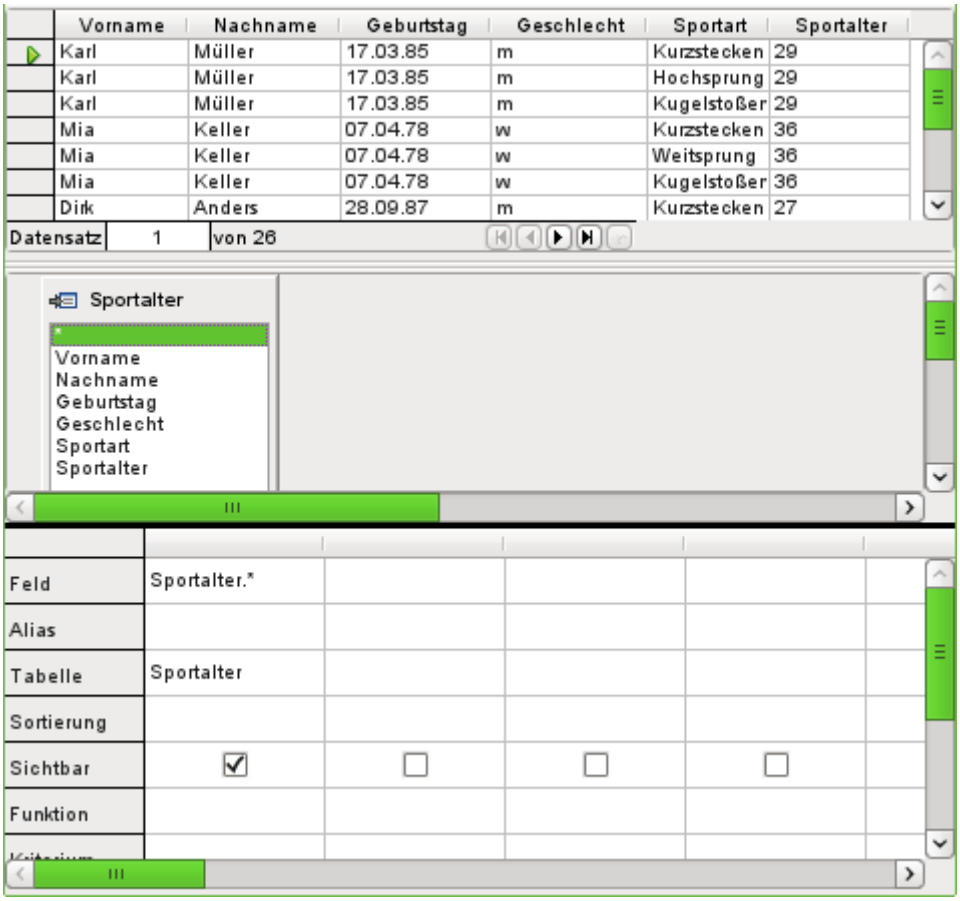
Feld	EXTRACT(YEAR FROM CURRENT_DATE) - EXTRACT(YEAR FROM "Geburtstag")
Alias	Sportalter
Tabelle	

In der Zeile «Alias» wird die Bezeichnung «Sportalter» eingegeben. Hier sollte nicht «Alter» stehen, weil die Person vielleicht erst im Laufe des Jahres dieses Alter erreicht. Im Kapitel *Aktuelles Alter ermitteln* steht beschreiben, wie das tatsächliche Alter ermittelt werden kann.

Sportart	Sportalter
Kurzstecken	29
Hochsprung	29
Kugelstoßer	29
Kurzstecken	36
Weitsprung	36

Wird die Abfrage erneut ausgeführt, so erscheint in dem Spaltenkopf nicht mehr die Formel sondern die Bezeichnung «Sportalter».

Damit der Code für den Anfang nicht zu verwirrend wird, sollte die Abfrage unter dem Namen "Sportalter" abgespeichert werden. Diese Abfrage dient dann als Grundlage für die nächste Abfrage, die das "Sportalter" jetzt einer "Altersklasse" zuordnet.

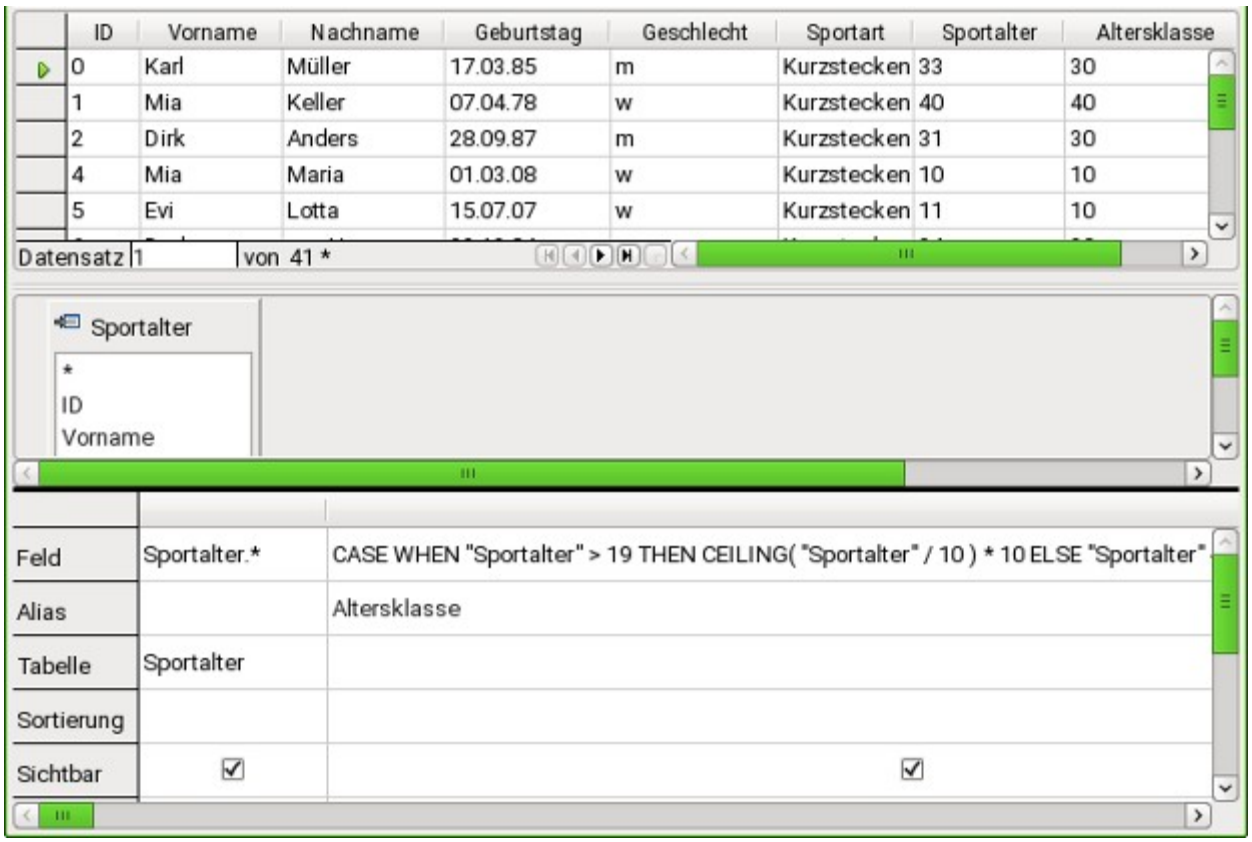


Die Abfrage "Sportalter" wurde als Grundlage für die 2. Abfrage ausgesucht. Vor der Bezeichnung «Sportalter» im Tabellenarbeitsbereich ist ein anderes Symbol zu sehen als bei den Tabellen der ersten Abfrage. Dieses Symbol zeigt an, dass die Grundlage dieser Abfrage eben eine Abfrage und keine Tabelle ist.

Durch Doppelklick auf «*» oder Auswahl von «Sportalter.*» werden alle Felder ausgewählt. Die gesamte Abfrage von vorher wird also übernommen. Nur die Formel ist nicht mehr zu sehen.

Auf das Feld "Sportalter" soll nun zugegriffen werden. Das Sportalter bestimmt ja, in welcher Altersklasse die jeweiligen Personen starten.

Damit die Ermittlung nicht allzu kompliziert wird soll hier folgende Zuordnung nachvollzogen werden: Unter 20 Jahren werden die Teilnehmer immer in zwei Jahrgängen, beginnend mit 0, zusammen gefasst. Ab 20 Jahren werden Gruppen für jeweils 10 Jahre gebildet, also von 20-30 Jahren usw.



Solche Formeln gehören eigentlich längst nicht mehr zum Einstiegs-Repertoire. Eine einfachere Zuteilung der Altersklassen wäre sogar mit Hilfe des folgenden Berichtes möglich. Für differenziertere Zuteilung sei erneut auf den Anhang des Handbuches verwiesen.

```
001 CASE WHEN "Sportalter" > 19 THEN CEILING( "Sportalter" / 10 ) * 10 ELSE
    "Sportalter" - MOD( "Sportalter", 2 ) END
```

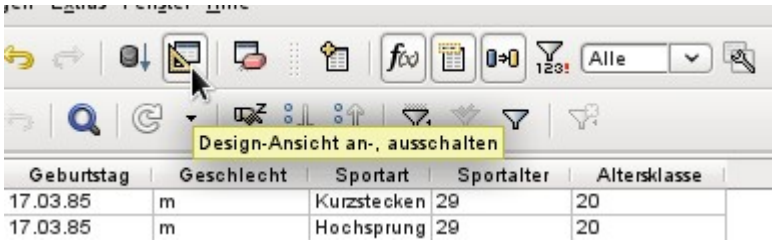
CASE WHEN Überprüfung einer Bedingung, wenn korrekt (**THEN**), dann diesen Wert nehmen, wenn nicht korrekt (**ELSE**), dann den anderen Wert nehmen.
In die deutsche Sprache übertragen bedeutet das so viel wie:
Für den Fall, dass das Sportalter über 19 Jahren ist, dividiere das Sportalter durch 10 und gib nur den Ganzzahlwert wieder. Diesen Ganzzahlwert multipliziere wieder mit 10.
Ist das Sportalter bis einschließlich 19 Jahren, so nehme den Wert des Sportalters und subtrahiere davon den Rest, der sich ergibt, wenn das Sportalter durch 2 dividiert wird. Das ergibt dann z. B. Für 19:

19/2 = 18, Rest 1, also 19 – 1 = 18.

Alle Teilnehmer über 19 Jahren werden damit einer Gruppe für das jeweilige Jahrzehnt zugewiesen. Alle Teilnehmer darunter werden einer Gruppe zugewiesen, die der jeweiligen niedrigeren geraden Zahl entspricht.

Dieser Formel wurde wieder ein Alias zugewiesen, in diesem Falle der Alias "Altersklasse".

Die Abfrage kann jetzt unter dem Namen "Meldung" gespeichert werden.



Das Ausschalten der Design-Ansicht ist hier eigentlich nicht notwendig, da alle Eingaben in der Design-Ansicht ohne größere Probleme möglich sind. Den Code von Abfragen einmal genauer anzusehen, kann allerdings nie schaden. Schließlich gibt es auch SQL-Formulierungen, die nur schlecht in die Design-Ansicht passen oder die dort gar nicht möglich sind. Dafür wird dann die direkte Eingabe im SQL-Code genutzt.

ID	Vorname	Nachname	Geburtstag	Geschlecht	Sportart	Sportalter	Altersklasse
0	Karl	Müller	17.03.85	m	Kurzsteckenlauf	33	30
1	Mia	Keller	07.04.78	w	Kurzsteckenlauf	40	40
2	Dirk	Anders	28.09.87	m	Kurzsteckenlauf	31	30
4	Mia	Maria	01.03.08	w	Kurzsteckenlauf	10	10
5	Evi	Lotta	15.07.07	w	Kurzsteckenlauf	11	10

Datensatz 1 von 41 *

```
SELECT "Sportalter".*, CASE WHEN "Sportalter" > 19 THEN
CEILING( "Sportalter" / 10 ) * 10 ELSE "Sportalter" -
MOD( "Sportalter", 2 ) END "Altersklasse" FROM "Sportalter"
```

Im Code sind Felder und Tabellen in doppelte Anführungszeichen gesetzt und in einem ockerfarbigen Farbton dargestellt. Begriffe des SQL-Codes sind in blau gehalten, Funktionen der Datenbank in grün. Die Bezeichnung **AS** erscheint in neueren Versionen von LO nicht mehr.

```
001 SELECT
002    "Sportalter".*,
003    CASE WHEN "Sportalter" > 19 THEN CEILING( "Sportalter" / 10 ) * 10
004       ELSE "Sportalter" - MOD( "Sportalter", 2 ) END
005    AS "Altersklasse"
006 FROM "Sportalter"
```

Der Teil mit der Formel wurde ja schon erwähnt. Hier jetzt der Rahmen.

```
001 SELECT "Sportalter".*, ... AS "Altersklasse" FROM "Sportalter"
```

Wähle aus der Tabelle "Sportalter" alle Datensätze aus und zusätzlich das, was durch die Formel bestimmt wird. Das, was durch die Formel bestimmt wird, bezeichne als "Altersklasse".

Der Code unterscheidet nicht zwischen Tabellen und Abfragen als Grundlage der Daten. Er funktioniert deshalb nur in der grafischen Benutzeroberfläche von Base. Die Benennung von Abfragen muss dafür immer anders sein als die Benennung von Tabellen.

Sobald in einer Abfrage bei ausgeschalteter Design-Ansicht der Button *SQL* (für **SQL-Kommando direkt ausführen**) gedrückt wird, wird die Datenbank mit einer Fehlermeldung reagieren. Die eingebettete Datenbank kennt die Abfrage "Sportalter" nicht, auf die sich die aktuellen Abfrage "Meldung" bezieht.

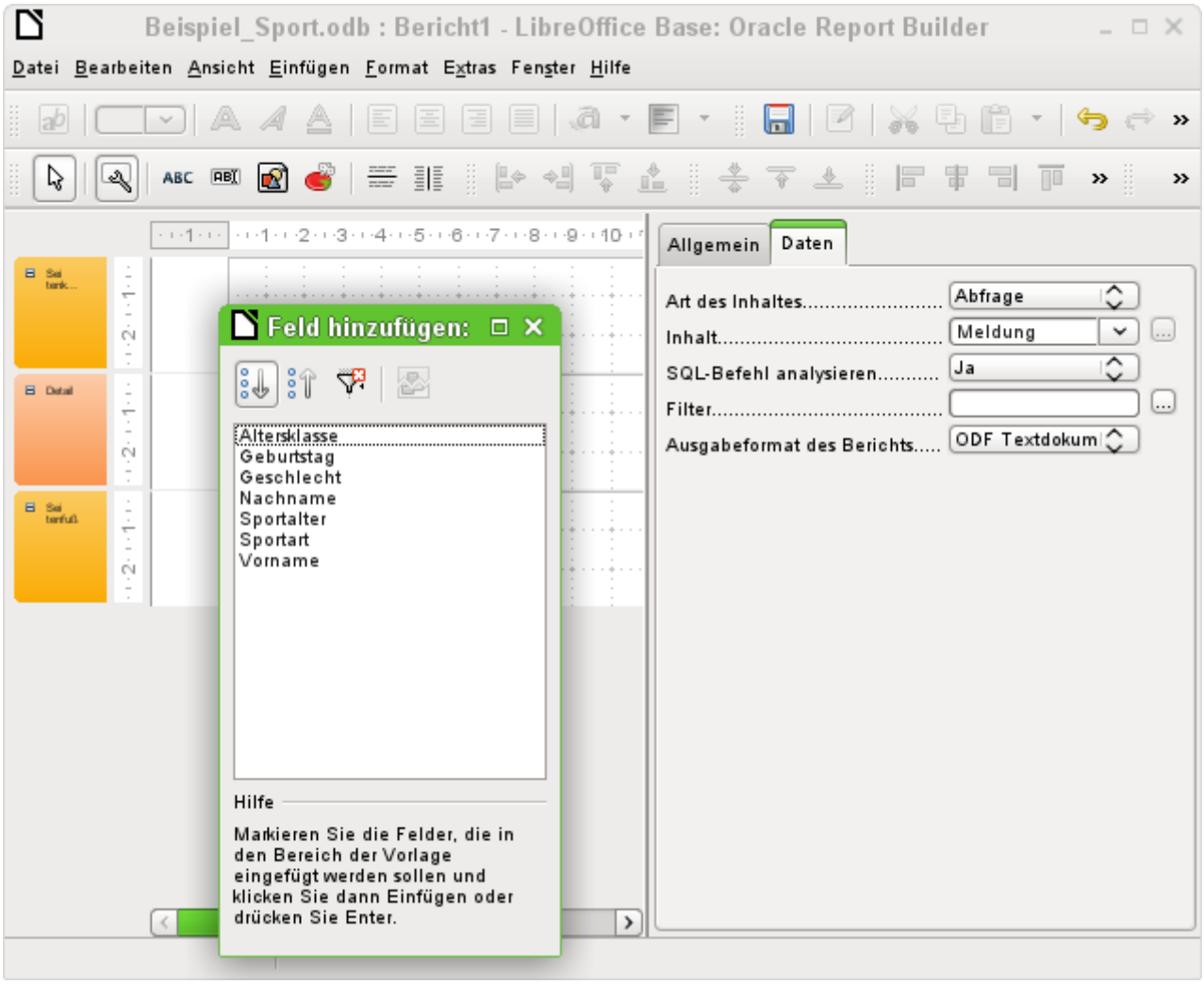
Ausdruck im Bericht

Über **Berichte → Bericht in der Entwurfsansicht erstellen …** wird ein Bericht erstellt, der eine übersichtliche Teilnehmerliste, sortiert nach Sportart, Geschlecht und Altersklasse darstellen soll.

Wird der Editor gestartet, so ist zuerst der Dialog «Feld hinzufügen» im Vordergrund. Für diesen Dialog wird die von der alphabetischen Sortierung her erste Tabelle als Basis genommen. Es muss für den Bericht aber die Abfrage als Datenquelle herausgesucht werden.

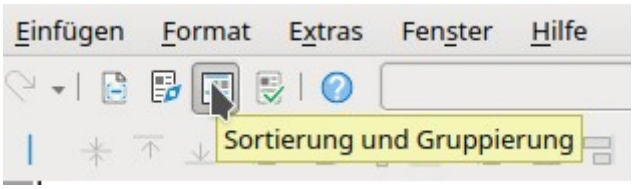
Im Berichtsfenster befindet sich auf der rechten Seite die Übersicht über die Eigenschaften des gerade aktiven Objektes. Sollte die Leiste nicht sichtbar sein, so muss sie über **Ansicht → Steuerelement-Eigenschaften…** ausgewählt werden.

Aus den Eigenschaften wird im Reiter **Daten → Art des Inhaltes → Abfrage** ausgewählt. Als **Inhalt** wird dann die Abfrage "Meldung" gewählt, die als letzte zusammenfassende Abfrage erstellt wurde.

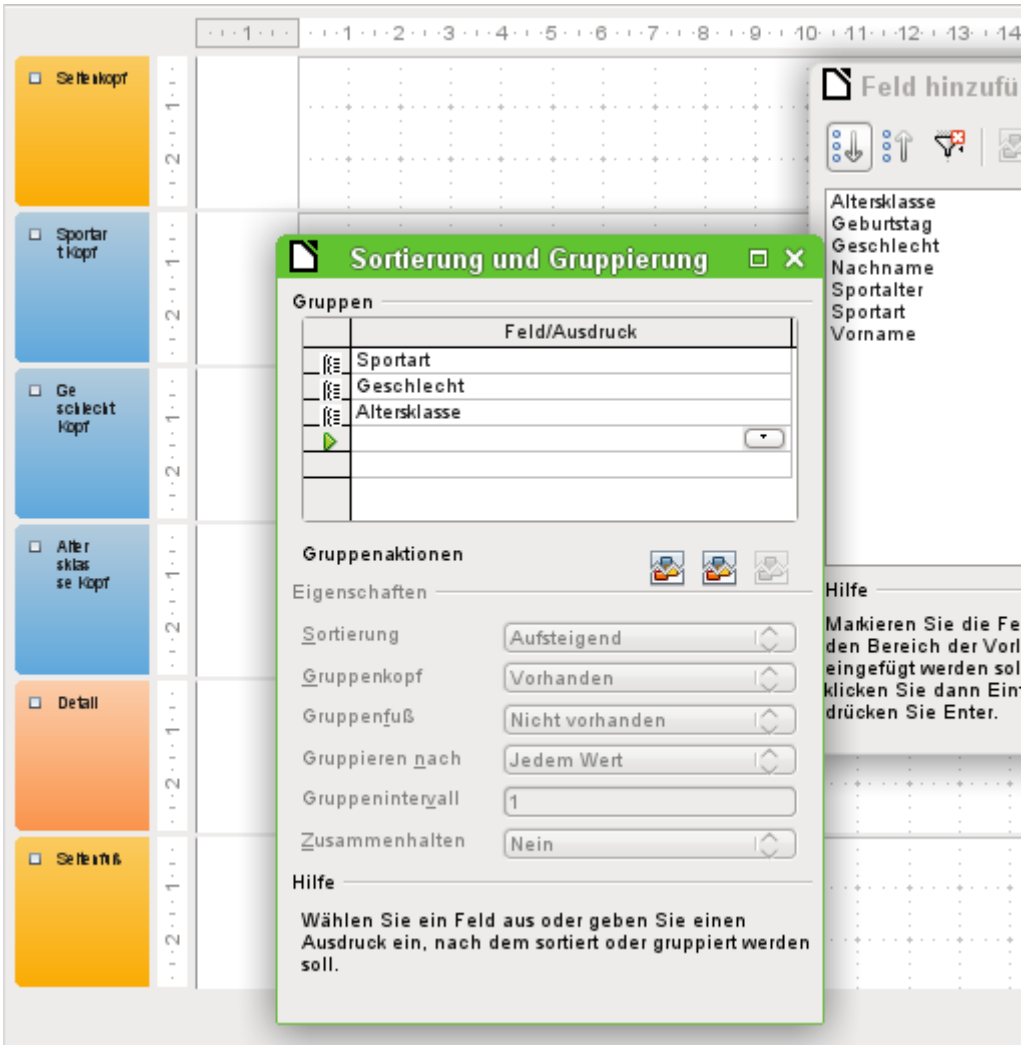


Der Report-Designer zeigt jetzt sämtliche Felder der ausgesuchten Abfrage an.

Im nächsten Schritt wird der Sortier-Dialog ausgewählt:



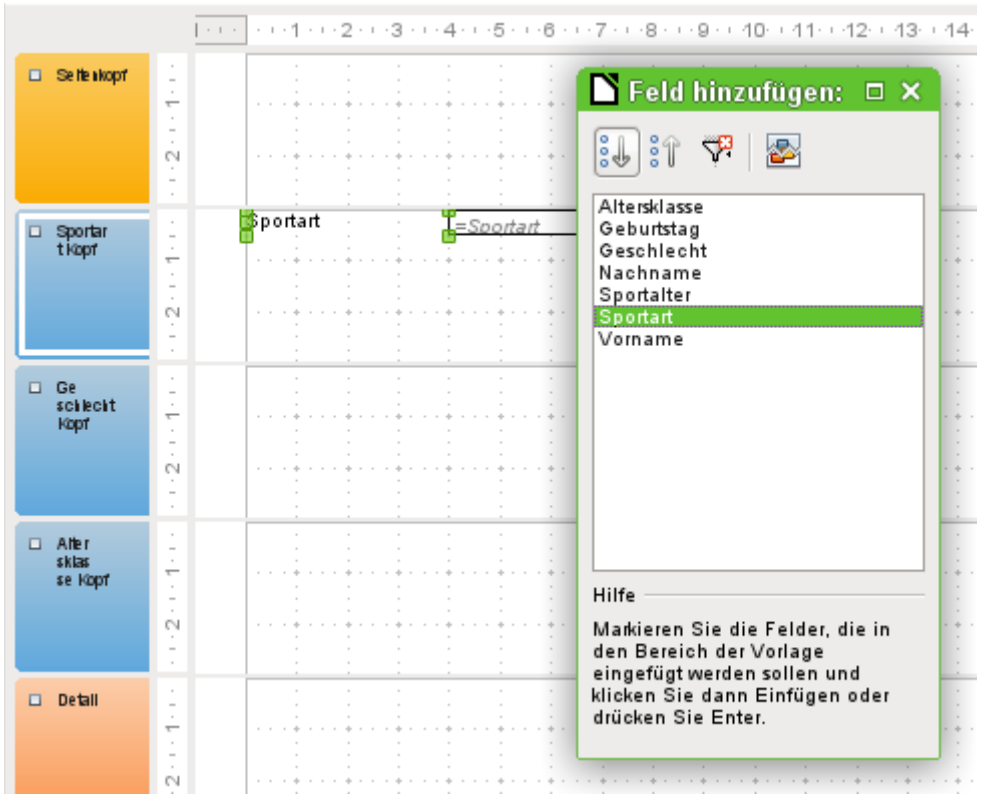
Die Sortierung kann auch über **Ansicht → Sortierung und Gruppierung** angezeigt werden.



Im Sortierungs- und Gruppierungs-Dialog werden nacheinander die Felder "Sportart", "Geschlecht" und "Altersklasse" ausgewählt.

Auf der linken Seite des Berichtes erscheint bei jeder Sortierungsauswahl ein Gruppenkopf für die Sortierung. Die Eigenschaften der Sortierung und Gruppierung werden einfach erst einmal so gelassen wie im Standard vorgesehen.

Der Dialog «Sortierung und Gruppierung» kann nun geschlossen werden.

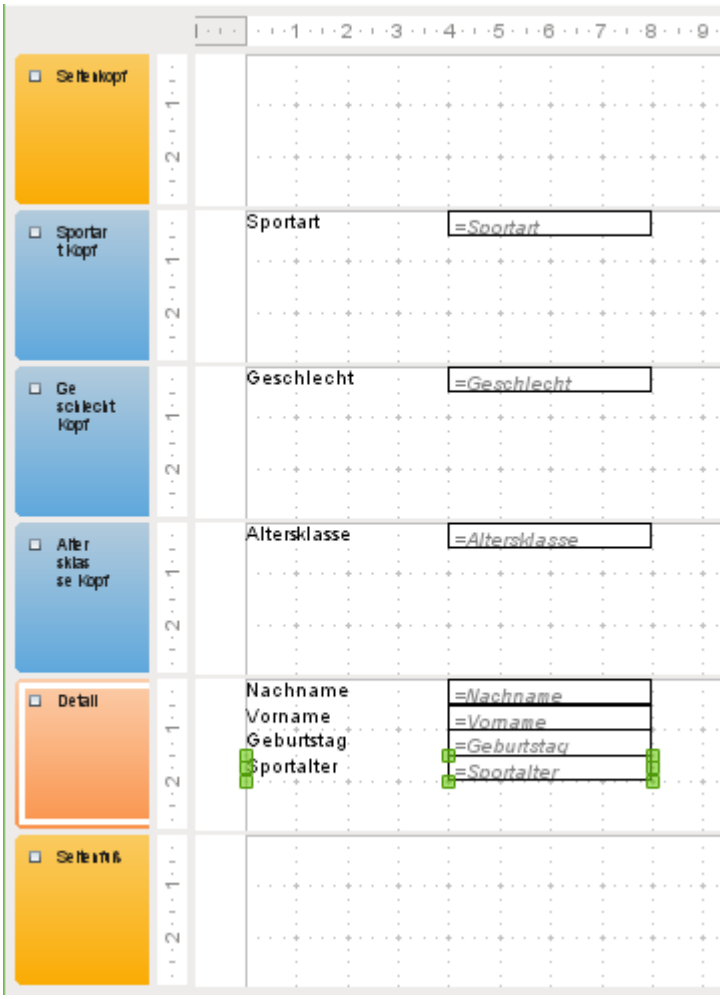


Der Gruppenkopf «Sportart Kopf» wird ausgewählt. Dies ist an der weißen Umrandung des Kopfes sichtbar. Anschließend wird im Dialog «Feld hinzufügen:» mit einem Mausklick auf das Feld «Sportart» ein Beschriftungsfeld und ein Textfeld eingefügt, das den Inhalt des Feldes «Sportart» aufnehmen soll.

Mit dem gleichen Verfahren werden die Felder «Geschlecht» und «Alter» den passenden Gruppenköpfen zugewiesen.

Sämtliche verbleibenden Felder werden in den Bereich «Detail» eingefügt.

Der Berichtsentwurf dürfte jetzt wie folgt aussehen:



Anschließend sollte der Bericht erst einmal abgespeichert werden. Als Name würde sich so etwas wie «Teilnehmerliste» anbieten.

Auch die Datenbankdatei selbst sollte abgespeichert werden, da sonst die Speicherung des Berichtes erst einmal nur temporär vorgenommen wird.

Hinweis

Gerade beim Gestalten eines Berichtes mit dem Report-Designer fallen oft Instabilitäten des Programms auf. Ist ein wichtiger Schritt gelungen, so sollte auf jeden Fall sowohl der Bericht als auch die Datenbankdatei gespeichert werden.
Die spätere Ausführung eines Berichtes ist von diesen Instabilitäten zum Glück nicht betroffen.

Wird dieser Bericht mit entsprechenden Daten ausgeführt, dann ergibt sich in etwa das folgende Bild:

Der Beginn des Berichtes zeigt hier zwei männliche Teilnehmer an, die Hochsprung machen wollen und der Altersklasse 20 angehören.

Bei der Ausführung des Berichtes fallen einige Designmängel auf:

1. Die Abstände zwischen den Gruppenköpfen und dem Inhalt des Detailbereichs sind viel zu groß.
2. Das Geschlecht wird mit «m» bzw. «w» angegeben. Besser wäre eine Bezeichnung wie «Herren» und «Damen».
3. Bei der Altersklasse steht, vermutlich wegen der Division in der Abfrage, eine Zahl mit einer Nachkommastelle, wobei als Dezimaltrenner ein Punkt gesetzt wird.
4. Die Bezeichner für die dargestellten Felder aus dem Bereich «Detail» («Nachname», «Vorname» usw.) sollten besser gemeinsam für alle Inhalte des Bereiches als Tabellenköpfe über den Inhalten stehen.

Abstände zwischen den Berichtsbereichen einstellen

Die Abstände zwischen den Bereichen können direkt an den Tabellenköpfen durch Schieben mit der Maus verringert werde, Es ist auch möglich, einen Bereich zu markieren und in den Eigenschaften des Gruppenkopfes die Höhe über eine Zahleneingabe zu regeln. Dafür darf kein Feld, sondern nur der Gruppenkopf markiert sein.

Es ist nicht möglich, einen Bereich kleiner zu wählen als die darin existierenden Beschriftungs- und Textfelder sind.

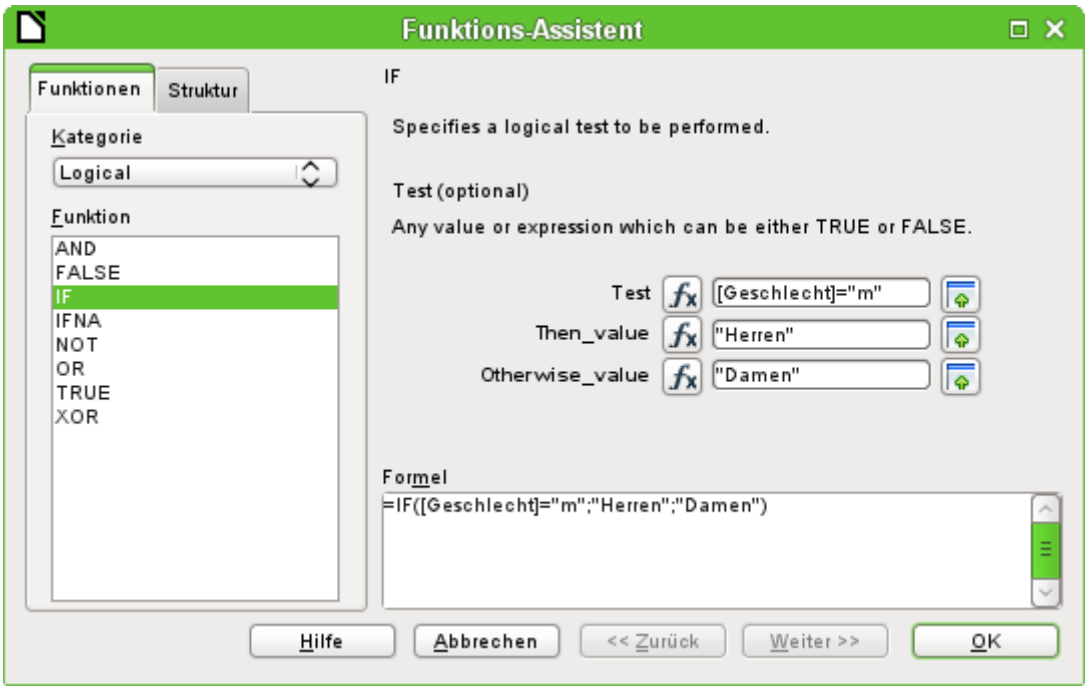
Der Bereich «Altersklasse» sollte dabei so groß gewählt werden, dass er noch die Beschriftungsfelder aus dem Bereich «Detail» aufnehmen kann.

Außerdem muss bei der Auswahl der Abstände berücksichtigt werden, dass eine folgende Gruppe nicht zu nah unter der vorhergehenden Gruppe erscheint. Die Beschriftungs- und Textfelder benötigen gegebenenfalls auch einen Abstand nach oben. Wird dieser Abstand nach oben nicht gewünscht, so kann damit nachgeholfen werden, dass zusätzlich zu den Gruppenköpfen auch Gruppenfüße angezeigt werden. Eine solche Einstellung ist über **Ansicht → Sortierung und Gruppierung** für jede Gruppe möglich.

Beeinflussung eines Textfeldinhaltes durch eine Formel

Die Bezeichnung des Geschlechtes aus der Tabelle reicht für die Ansprüchen einer Teilnehmerliste nicht aus. Es könnte eine entsprechende Umbenennung bereits in der Abfrage erfolgen. Da die Abfrage aber bereits erstellt wurde, werden jetzt die Möglichkeiten des Berichtes genutzt.

Das Textfeld «=Geschlecht» wird markiert. In den Eigenschaften auf der rechten Seite des Report-Designers wird der Reiter **Daten → Datenfeld** aufgesucht. Dort wird der Button mit den 3 Punkten … betätigt. Es erscheint der **Funktions-Assistent**. Alle Funktionen sind hier allerdings nicht übersetzt, sondern in englischer Sprache erreichbar.



In der Kategorie «Logical» steckt die Funktion «IF». Die Vorgehensweise bei dieser Funktion ist gleich der Funktion «WENN» aus LO-Calc.

Ein Testwert wird eingegeben. Dabei ist das Feld der Abfrage, aus der die Daten ausgelesen werden, in eckige Klammern zu setzen. Texte müssen in doppelte Anführungszeichen gesetzt werden. Wenn im Feld "Geschlecht" der Wert «m» steht, dann soll in dem Feld des Berichtes «Herren» angezeigt werden. Steht dort kein «m», dann wird «Damen» angezeigt.

Die Eingabe wird mit OK bestätigt.

Damit ist die Formulierung des Feldes entsprechend umgestellt.

Umstellung der Formatierung eines Textfeldes

Das Feld, das den Inhalt der Datenbank aufnehmen soll, wird zwar in Berichten als «Textfeld» bezeichnet. Es kann aber genau so formatiert werden wie Felder in Tabellen im Writer oder in Calc.

Das Feld «=Altersklasse» wird markiert. Bei den Eigenschaften am rechten Bildschirmrand wird **Allgemein → Formatierung** aufgesucht. Die Formatierung steht dort auf «Text». Der Button mit den 3 Punkten (…) wird gedrückt. Es öffnet sich der Formatierungs-Dialog, der auch aus Calc, dem Writer oder bei Formularerstellungen bekannt ist. Hier wird **Kategorie → Zahl** ausgewählt und mit OK bestätigt.

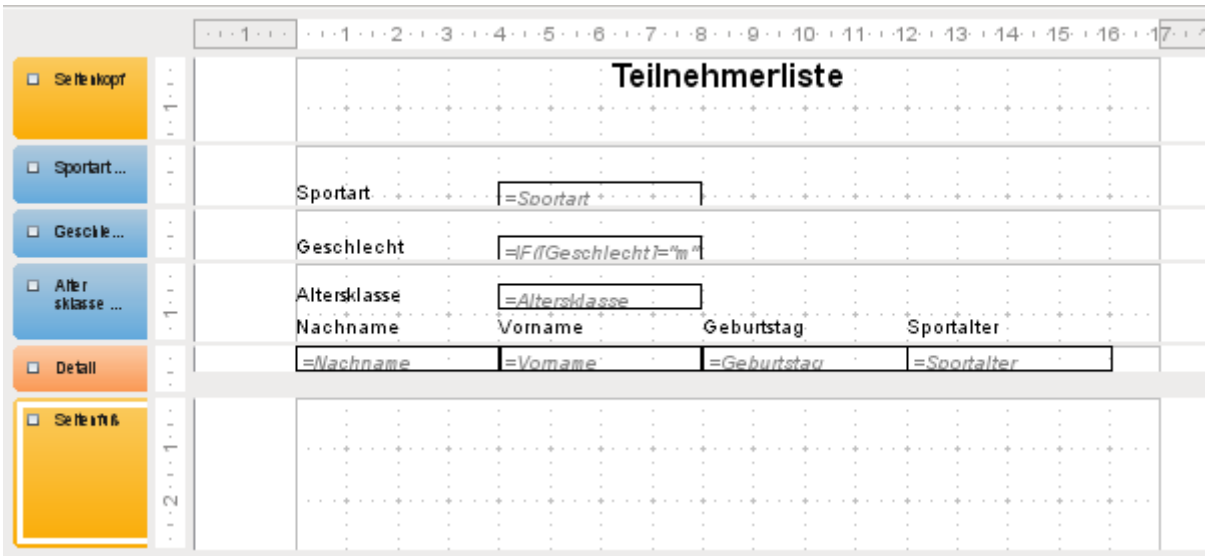
Die Formatierung der Anzeige ist nun von Text auf Zahl umgestellt.

Verschieben von Feldern im Report-Designer

Felder können beim Report-Designer auch über die Grenzen eines Bereiches hinweg verschoben werden. Im Zielbereich muss allerdings genügend Platz für das Feld vorhanden sein. Es kann nämlich nicht ein Feld über einem anderen liegen.

Die Positionierung mit der Maus ist je nach System so ungenau, dass am besten reichlich Platz vorhanden sein sollte. Sind erst einmal die Felder in einem Bereich vorhanden wie hier dem Gruppenkopf «Altersklasse», so lassen sie sich auch mit den Pfeilfeldern und mit Tastatureingaben genau positionieren.

Zur Positionierung mit der Tastatur ist in den Eigenschaften **Allgemein → Position X** und **Allgemein → Position Y** auf zu suchen.



Hier wurde noch ein Beschriftungsfeld für die Überschrift hinzugefügt. Die Eingabe des Textes für das Beschriftungsfeld erfolgt in den Eigenschaften des Feldes.

Der Seitenfuß wurde in den Eigenschaften auf **Sichtbar → Nein** eingestellt. Er nimmt sonst am unteren Rand des Dokumentes zu viel Platz weg. Hierbei sollte bedacht werden, dass die verfügbare Höhe bereits durch die Größe der Seitenränder zusätzlich verringert wird. Standardmäßig sind alle Seitenränder bei einer DIN-A4-Seite auf 2 cm eingestellt.

Weitere Formatierungsmöglichkeiten für den Bericht werden in dem gesonderten Kapitel *Berichte mit dem Report-Designer* beschreiben.

Erweiterungen der Beispieldatenbank

Das vorgestellte Beispiel ist natürlich nur der erste Schritt für eine Datenbank im Sportbereich. Zuerst einmal sollte auf jeden Fall ein Feld ergänzt werden, das die jeweiligen Leistungen aufnimmt. Solch ein Feld wäre gut in der Tabelle "rel_Teilnehmer_Sportart" unter zu bringen.

Sollen die Eingaben allerdings für mehrere Wettkämpfe in gleichen Sportarten gelten, so ist außerdem noch ein Datumsfeld oder ein anderes Feld, das dem jeweiligen Wettkampf zugeordnet werden kann, in die Tabelle "rel_Teilnehmer_Sportart" aufzunehmen. Das Feld muss dann auch Teil des gemeinsamen Primärschlüssels sein.

Vielleicht käme noch die Vereinszugehörigkeit hinzu. Dazu würde ein Feld in der Tabelle "Teilnehmer" ausreichen. Bei vielen gleichen Vereinen bietet sich aber auch eine separate Tabelle "Vereine" sowie der entsprechende Fremdschlüssel in der Tabelle "Teilnehmer" an.

Dann muss natürlich, wie bei allen Wettbewerben, ermittelt werden, wer denn nun in der jeweiligen Altersklasse auf welcher Platzierung gelandet ist. Sortierung ist hier gefragt, die vielleicht wieder als Bericht in einer Ergebnisliste landet.

Schön wäre dann, wenn jeder Teilnehmer (wieder über einen Bericht) eine gestaltete Urkunde mit den persönlichen Leistungen und der Platzierung erhält.

Solche Erweiterungen sind ohne weiteres möglich. Das sollen die kommenden Kapitel des Handbuches zeigen.

Weiteres Einführungs- und Beispielmaterial

Neben den beiden Hauptdatenbanken für das Handbuch «Medien_ohne_Makros.odb» und «Medien_mit_Makros.odb» steht noch ein ganzes Bündel von Beispieldatenbanken mit einer ausführlichen Erklärung an gleicher Stelle wie das Handbuch zum Download bereit.

Ein mehrstufiges Videotutorial von F3K Total, das von den Grundlagen her den Einstieg ermöglicht ist gut von dieser Seite aus zu starten: *http://de.openoffice.info/viewtopic.php?f=27&t=54234#p237478*

Anlaufpunkte für Fragen zu Base sind im deutschsprachigen Raum vor allem die Foren *https://ask.libreoffice.org/c/german/6* (Forum von LibreOffice – Dokument Foundation) und *http://www.-libreoffice-forum.de/viewforum.php?f=10* .

Datenbank erstellen

Allgemeines bezüglich der Erstellung einer Datenbank

In LibreOffice gibt es das Programmmodul «Base». Dies stellt eine grafische Benutzeroberfläche für Datenbanken zur Verfügung. Daneben existiert, in LibreOffice eingebunden, die interne Datenbank «HSQLDB» sowie die interne Datenbank «Firebird». Diese internen Versionen der Datenbanken können nur als Datenbanken von einem Nutzer betrieben werden. Die gesamten Daten sind in der *.odb-Datei mit abgespeichert und diese Datei ist nur für einen Benutzer nicht schreibgeschützt zu öffnen.

Neue Datenbank als interne Datenbank

Sofern nicht von vornherein eine Multiuserdatenbank geplant ist oder erst einmal erste Erfahrungen mit einer Datenbank gesammelt werden sollen, ist die interne Datenbank gut nutzbar. Diese lässt sich später auch mit ein paar Kniffen noch zur externen Datenbank umwandeln, auf die dann auch mehrere User gleichzeitig zugreifen können, wenn der Datenbankserver läuft. Eine Beschreibung dazu in Bezug auf die HSQLDB erfolgt im Anhang dieses Handbuches im Kapitel *Datenbankverbindung zu einer externen HSQLDB*. Der Umstieg bei Firebird zur externen Variante ist in den Kapiteln *Direkte Verbindung zu einer Firebird-Datei* und *Von der externen Firebird-Datei zur Serverdatenbank* beschrieben.

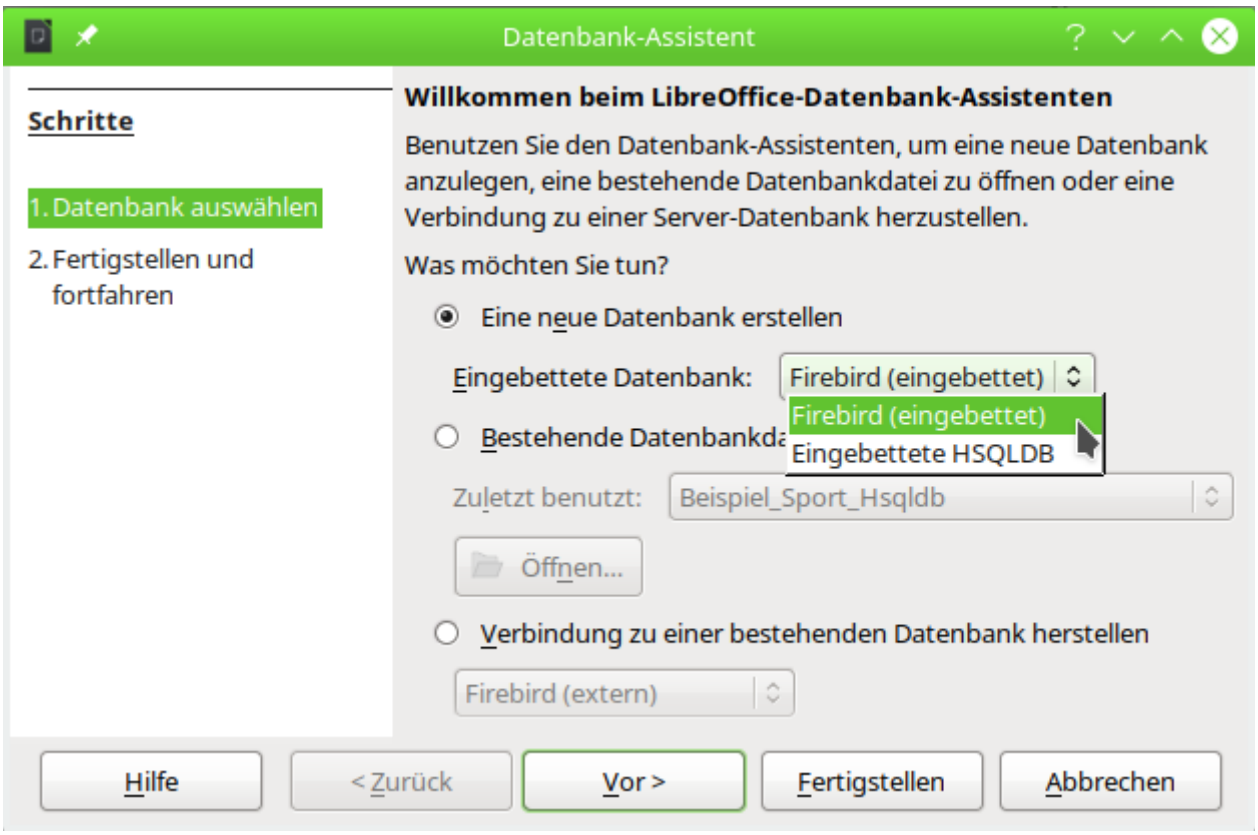
Hinweis

Die interne HSQLDB benötigt (wie auch der ReportBuilder und die Assistenten) zwingend Java. Häufigster Fehler bei der Erkennung von Java durch LibreOffice ist, dass Nutzer LibreOffice in einer 64bit-Version installiert haben, Java aber in der 32bit-Version. LibreOffice und Java müssen in der gleichen Version vorliegen.
Für MAC-User ist außerdem wichtig zu wissen, dass die JavaRuntimeEnvironment (JRE) nicht für den Betrieb von Base ausreicht. Hier ist das JavaDevelopementKit (JDK) notwendig. Manche Java-Versionen passen hier allerdings nicht korrekt zu LibreOffice. Die Versionen von AdoptOpenJDK scheinen hier besser zu funktionieren: *https://adoptopenjdk.net/releases.html?variant=openjdk15&jvmVariant=hotspot*
LibreOffice arbeitet ja nach Version auch nur mit bestimmten Versionen von Java zusammen. Dies fällt z.B. auf, wenn ältere Versionen von LibreOffice mit einer aktuellen Java-Laufzeitumgebung eine fehlende Java-Version anmahnen. Unter MAC scheint es außerdem vor zu kommen, dass zu neue Versionen des JDK zwar angezeigt werden, aber mit Base trotzdem nicht zusammen arbeiten. Hier hilft immer ein Blick auf *https://ask.libreoffice.org/tag/base*. Dort werden auch unkonventionelle Lösungen wie die JDK von *https://adoptium.net/temurin* empfohlen.

Hinweis

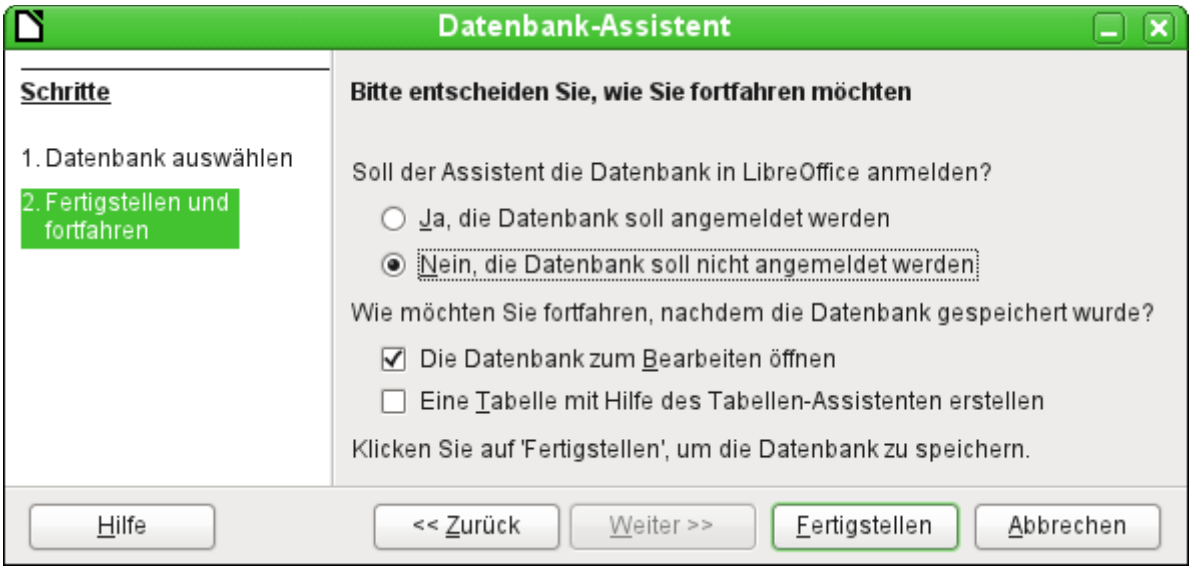
Die Nutzung großer Datenmengen kann bei **Java** zu Geschwindigkeitsverlusten führen. Dies kann von der verwendeten Java-Version abhängen. Ein einfacher Test bei direktem Ausführen von SQL ergibt für eine Tabelle von 15 Spalten und 30.000 Datensätzen deutliche Unterschiede für die Zeit, die für das Scrollen zum letzten Datensatz notwendig sind. Die interne HSQLDB braucht hier fast 30 Sekunden, Firebird hingegen schafft das in 1 Sekunde.
Auch bei externen Datenbanken lässt sich das in Abhängigkeit von dem Treiber beobachten. Die MariaDB mit der direkten Verbindung von LibreOffice aus liegt hier mit Firebird gleichauf. Die gleiche Datenbank, jetzt aber über JDBC mit LibreOffice verbunden, ist genauso langsam wie die interne HSQLDB.

Die Erstellung einer internen Datenbank erfolgt direkt über den Eingangsbildschirm von LO mit einem Klick auf den Button Datenbank oder über **Datei → Neu → Datenbank.** Der Datenbank - Assistent startet mit dem folgenden Bildschirm:



Das Optionsfeld «Neue Datenbank erstellen» ist für eine neue interne Datenbank angewählt. Standardmäßig ist dies die eingebettete HSQLDB. Ab LO 4.2 steht zusätzlich die interne Firebird Datenbank zur Verfügung, bis Version LO 6.0 müssen dafür aber noch die experimentellen Funktionen (**Extras → Optionen → LibreOffice → Erweitert → Optionale Funktionen**) aktiviert werden. Auch ab LO 6.4.3 ist dies wieder notwendig, da die Integration weiter Probleme bereitet. Nur so steht hier auch «Firebird (eingebettet)» zur Verfügung.

Die weiteren Optionen dienen dazu, eine bestehende Datei zu öffnen oder eine Verbindung zu einer externen Datenbank wie z.B. einem Adressbuch, einer MySQL-Datenbank o.ä. aufzubauen.



Eine in LibreOffice angemeldete Datenbank kann von den anderen Programmteilen von LibreOffice als Datenquelle genutzt werden (z. B. Serienbrief). Diese Anmeldung kann aber

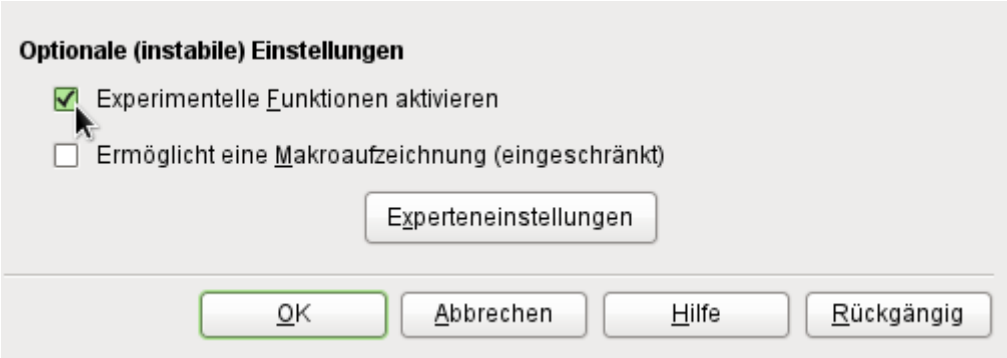
auch zu einem späteren Zeitraum erfolgen. Deshalb ist an dieser Stelle erst einmal das Optionsfeld **Nein, die Datenbank soll nicht angemeldet werden**, ausgewählt.

Bei den Markierfeldern ist lediglich «Die Datenbank zum Bearbeiten öffnen» markiert. Assistenten zur Erstellung von Tabellen, Abfragen usw. werden in diesem Handbuch nicht genutzt. Ihre Funktion ist lediglich für den einfachen Einstieg in die Materie geeignet.

Die Datenbankdatei wird fertiggestellt, indem direkt nach einem Namen und einem Speicherort gefragt wird. An der entsprechenden Stelle wird dann die *.odb-Datei erzeugt, die zur Aufnahme von Daten aus der internen Datenbank, zur Speicherung von Abfragen, Formularen und Berichten gedacht ist.

Im Gegensatz zu den anderen Programmteilen von LibreOffice wird also die Datei abgespeichert, bevor der Nutzer überhaupt irgendwelche für ihn sichtbare Eingaben getätigt hat.

Die interne Firebird Datenbank könnte in Zukunft die interne HSQLDB ersetzen. Zur Vereinfachung des Wechsels wird an einem Migrationstool gearbeitet. Dieses Tool bietet die Migration von HSQLDB zu Firebird an, wenn über **Extras → Optionen → LibreOffice → Erweitert → Optionale (instabile) Einstellungen** die experimentellen Funktionen aktiviert werden:



Die HSQLDB ist auch für LibreOffice 7.4 die Standarddatenbank für jeden Nutzer; die Firebird-Datenbank hat mit LO 5.3 ein Update auf die aktuelle Firebird-Version 3.0 vollzogen, ist aber bisher nur experimentell zu empfehlen. Die Datenbank-Beispiele beziehen sich weiterhin auf die HSQLDB, sind aber so angepasst, dass die meisten Funktionen direkt auf Firebird übertragbar sind. Gegebenenfalls werden Alternativen in Firebird direkt aufgezeigt.

Hinweis

Die Nutzung von Firebird ist mit der Version LO 6.4.3 wieder in den experimentellen Status zurückversetzt worden. Einige Fehler der internen Version sind so beschaffen, dass sie bei Abfragen gleich die ganze Datenbank zum Absturz bringen lassen und den Inhalt von Tabellen erst nach einem Neustart von Base wieder anzeigen.

Vorsicht

Mit den experimentellen Funktionen erscheint bei jedem Öffnen einer internen HSQLDB-Datenbank (über den Zugriff auf den Tabellencontainer) ab LO 6.1 der folgende Dialog:

Hier ist äußerste Vorsicht geboten! Später drücken und dann erst einmal eine Sicherungskopie der Datenbankdatei machen. Auf keinen Fall einfach bestätigen! Die Migration funktioniert nicht einwandfrei, kann dies auch vermutlich nie.

1. Sicherheitskopie der HSQLDB-Datenbankdatei anfertigen.
2. Funktionen nach der im Anhang des Handbuches unter *Migration HSQLDB → Firebird* stehenden Liste so weit wie möglich anpassen.
3. Ansichten, die nicht direkt umgestellt werden können, vom SQL-Code her kopieren und als Abfragen abspeichern.
4. Tabellennamen und Spaltennamen dürfen in Firebird nur maximal 31 Zeichen lang sein. Gegebenenfalls also anpassen.
5. Reine Texttabellen (eingebundene *.csv-Tabelle etc.) sind unter Firebird nicht möglich, müssen also anderweitig ersetzt werden. Prinzipiell gibt es auch bei Firebird eine Möglichkeit, bestimmte Textdateien (nicht: *.csv-Dateien) einzubinden. Diese Funktion ist aber aus Sicherheitsgründen standardmäßig nicht eingeschaltet.

Zugriff auf externe Datenbanken

Alle externen Datenbanken müssen zuerst einmal existieren. Angenommen es soll der Zugriff auf die Datenbank einer eigenen Homepage erfolgen. Dort muss zuerst einmal die Datenbank selbst mit einem Namen und Zugangsdaten gegründet worden sein, bevor externe Programme darauf zugreifen können.

Existiert nun diese externe Datenbank, so kann sie, je nach vorhandener Treibersoftware, zur Erstellung von Tabellen und anschließender Dateneingabe und -abfrage genutzt werden.

Über **Datei → Neu → Datenbank** wird der Datenbankassistent geöffnet und die Verbindung zu einer bestehenden Datenbank hergestellt. Die Liste der hier aufgeführten Treiber, die mit Datenbanken verbinden, variiert etwas je nach Betriebssystem und Benutzeroberfläche. Immer dabei sind aber

- dBase
- JDBC
- MySQL
- ODBC
- Oracle JDBC
- PostgreSQL

- Firebirddatei
- Tabellendokument
- Text
- sowie verschiedene Adressbücher

Je nach gewählter Datenbank variieren nun die Verbindungseinstellungen. Diese können auch gegebenenfalls später noch korrigiert werden, wenn erst einmal die *.odb-Datei erstellt worden ist.

Bei manchen Datenbanktypen können keine neuen Daten eingegeben werden. Sie sind deshalb nur zum Suchen von Daten geeignet (z.B. Tabellendokument, Adressbücher). Die Beschreibungen in den folgenden Kapiteln beziehen sich ausschließlich auf die Verwendung von LibreOffice Base mit der internen Datenbank HSQLDB. Die meisten Ausführungen lassen sich auf Datenbanken wie MySQL, PostgreSQL etc. übertragen und entsprechend anwenden.

Hier soll nur kurz an ein paar Beispielen vorgestellt werden, wie der Kontakt zu anderen externen Datenbanken hergestellt wird.

Hinweis

In Abhängigkeit vom benutzten Treiber ist es möglich, gleichzeitig auf mehrere Datenbanken des jeweiligen Servers zuzugreifen. Wird z.B. in MySQL/MariaDB mit der direkten Verbindung ein Kontakt zu einer Datenbank hergestellt, so werden in Base gleichzeitig alle anderen Datenbanken auf dem Server angezeigt, bei denen der angegebene Nutzer Leserecht hat. Es ist also problemlos möglich, Daten von einer Datenbank zu einer anderen zu kopieren, zusammenhängende Abfragen durchzuführen usw.

Hinweis

ODBC-Verbindungen werden in Linux direkt über die Anweisungen in den Dateien «odbcinst.ini» und «odbc.ini» erstellt. Für Windows existiert hier eine GUI, die bei der Erstellung helfen soll. Dort scheint es dann aber vor zu kommen, dass für LibreOffice nicht nur die existierenden eingetragenen Datenquellen angeboten werden, sondern auch leere Datenquellen.
Auf die Erstellung von ODBC-Verbindungen unter Windows kann hier nicht weiter Bezug genommen werden, da zum einen mehrere Treiber existieren, die GUI zu viele Einstellungsmöglichkeiten lässt (die einer GUI als Allroundwerkzeug geschuldet sind) und außerdem mir als Autor das Betriebssystem nicht zur Verfügung steht.

MySQL/MariaDB-Datenbanken

MySQL-Datenbanken oder auch MariaDB-Datenbanken können auf drei verschiedene Weisen mit Base verbunden werden. Die einfachste und schnellste Art ist die direkte Verbindung mit dem MySQL-Connector. Daneben steht noch die Verbindung über JDBC und ODBC zur Verfügung.

Hinweis

In MySQL und MariaDB ist es möglich, Daten in Tabellen ohne ein Primärschlüsselfeld einzugeben und zu ändern. Die GUI von Base zeigt diese Tabellen zwar an, bietet aber keine Eingabe- bzw. Änderungsmöglichkeit.
Wer Tabellen ohne Primärschlüssel verwenden möchte, kann stattdessen über **Extras → SQL**, oder innerhalb von Formularen über Makros, die Tabellen mit Daten versorgen.

Erstellen eines Nutzers und einer Datenbank

Nachdem MySQL bzw. MariaDB installiert ist, sollten nacheinander die folgenden Schritte vollzogen werden:

1. Der Administrationsuser in MySQL heißt «root». Für Linuxnutzer ist hier wichtig, dass es sich dabei nicht um den Administrator des Linuxsystems handelt. Diesem Nutzer wird direkt nach der Installation erst einmal ein Passwort zugewiesen. Dies kann je nach System auch schon vorher bei der Installation erfolgt sein. In der Konsole wird nach der Anmeldung als «sudo»
   mysql -u root -p
   eingegeben. Am Anfang ist kein Passwort vorhanden, so dass bei der Eingabeaufforderung auch nur *Enter* gedrückt wird. Gegebenenfalls muss **-p** für die Passworteingabe auch weg gelassen werden.
   Es erscheint die Eingabeaufforderung
   mysql>
   Alle folgenden Eingaben werden jetzt auf dieser mysql-Konsole erstellt. Die Passwörter können sich unterscheiden, je nachdem, ob die Anmeldung von dem momentanen Rechner («localhost») oder von einem anderen Rechner zum MySQL-Serverrechner («host») erfolgt.
   SET PASSWORD FOR root@localhost=PASSWORD('Passwort');
   SET PASSWORD FOR root@host=PASSWORD('Passwort');
   Windows-Nutzer geben statt der zweiten Zeile
   SET PASSWORD FOR root@'%'=PASSWORD('Passwort');
   ein.
2. Als Sicherheitsmaßnahme werden alle eventuell bestehenden anonymen Nutzer gelöscht.
   DELETE FROM mysql.user WHERE User='';
   DELETE FROM mysql.db WHERE User='';
   FLUSH PRIVILEGES;
3. Eine Datenbank mit dem Namen «libretest» wird erstellt.
   CREATE DATABASE libretest;
4. Alle Rechte an der Datenbank «libretest» werden dem Nutzer «lotest» übergeben, der sich durch das Passwort «libre» ausweisen soll.
   GRANT ALL ON libretest.* TO lotest IDENTIFIED BY 'libre';

Damit ist die Datenbank vorhanden, mit der die folgenden Verbindungen aufgebaut werden.

Direkte Verbindung zu MySQL/MariaDB

Für die **MariaDB** und die **MySQL-Datenbank** existiert eine direkte Verbindung mit Base. Diese Verbindung funktioniert auf jeden Fall mit der MariaDB, eventuell aber nicht einwandfrei mit neueren Versionen von MySQL ab Version MySQL 8. Das liegt daran, dass aus Lizenzgründen vom MySQL-C++-Verbinder zum MariaDB-C-Verbinder gewechselt wurde. Der MariaDB-C-Verbinder steht unter der LGPL-Lizenz, die kompatibel zu der LibreOffice-Lizenz ist.[3]

Für MySQL-Versionen ab MySQL 8 sollte daher eher eine JDBC oder ODBC-Verbindung genutzt werden.

MySQL/MariaDB-Verbindung über JDBC

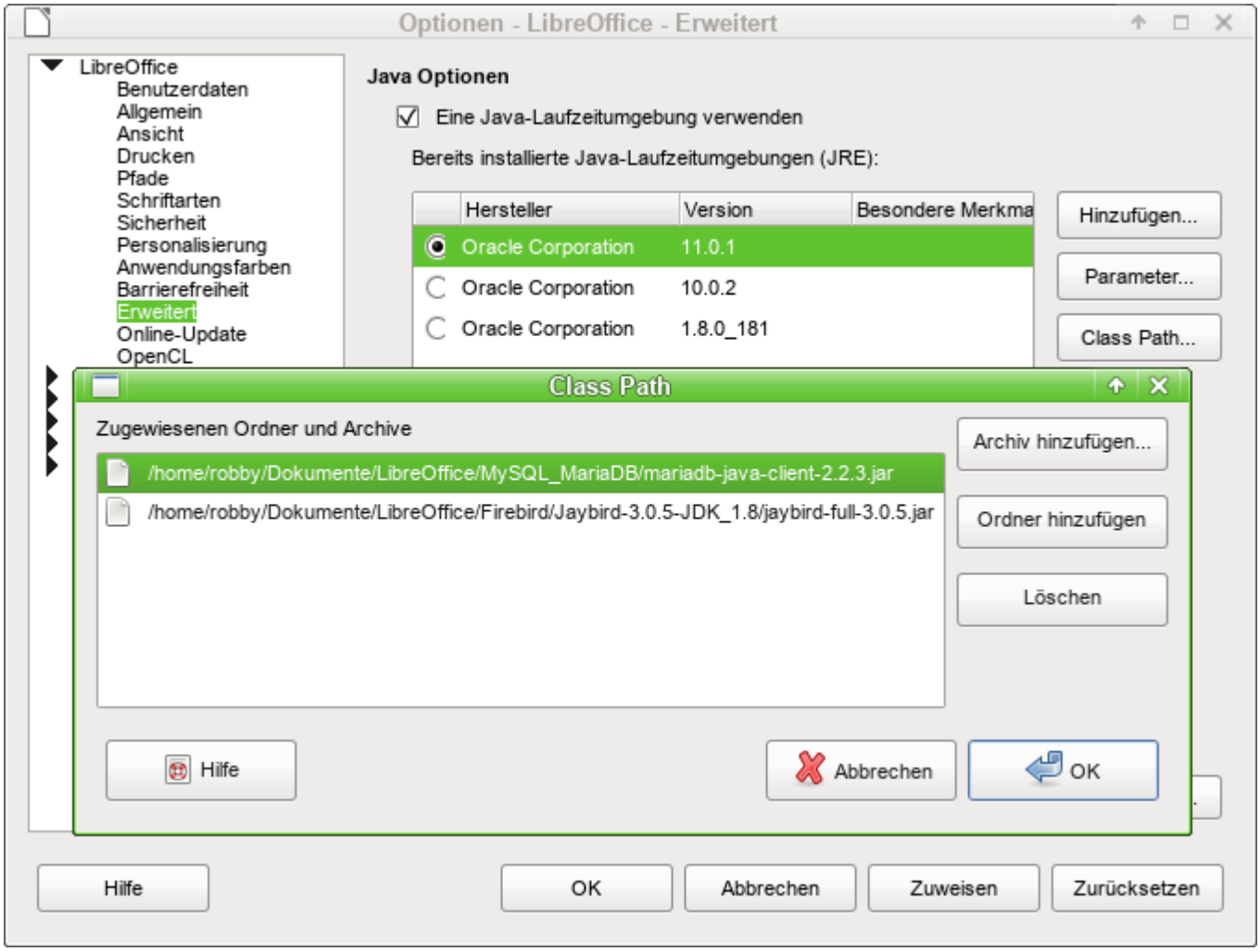
Als allgemeiner Zugang zu MySQL ist der Zugang über JDBC oder ODBC zu wählen. Um den JDBC-Zugang nutzen zu können, wird der mysql-connector-java.jar in der jeweils zur Datenbank passenden Fassung benötigt. Für MySQL kann das entsprechende Paket, das diesen Connector enthält, hier herunter geladen werden: *https://dev.mysql.com/downloads/connector/j/* . Dieses Java-Archiv wird am besten in das Verzeichnis kopiert, aus dem auch die aktuelle Java-Version von LibreOffice geladen wird. Als Speicherort bietet sich das Unterverzeichnis «...Javapfad.../

3 Siehe *https://wiki.documentfoundation.org/ReleaseNotes/6.2/de*

lib/ext/» an. In Linux-Systemen wird dies durch die Softwareverwaltung oft automatisch erledigt.

Für eine Nutzung durch lediglich einen Benutzer kann der connector allerdings auch einfach irgendwo im Benutzerverzeichnis liegen. Dann ist allerdings notwendig, den **Class Path** anzugeben.

Gegebenenfalls kann das Java-Archiv auch separat über **Extras → Optionen → LibreOffice → Erweitert → Java Optionen→ Class Path** in den Class-Path aus jeder beliebigen Stelle der Festplatte übernommen werden. Für diese Option sind dann auch keine Systemverwalterrechte notwendig.



Über **Class Path → Archiv hinzufügen…** wird die zu der Datenbank passende *.jar-Datei gesucht und in LibreOffice eingebunden.

Der Treiber wird bis Version 5.1 mit **`com.mysql.jdbc.Driver`** angesprochen. Neuere Treiber benötigen den Eintrag **`com.mysql.cj.jdbc.Driver.`**

Für die MariaDB bietet sich der JDBC-Treiber von *https://downloads.mariadb.com/Connectors/java/* an. Der Treiber kann ebenso in den Class-Pfad eingebunden werden und wird mit **`org.mariadb.jdbc.Driver`** angesprochen.

MySQL/MariaDB-Verbindung über ODBC

Für eine Verbindung über ODBC muss natürlich erst einmal die entsprechende ODBC-Software installiert sein. Details dazu werden hier nicht weiter beschrieben.

Nach Installation der entsprechenden Software kann es passieren, dass LO den Dienst verweigert, weil es die libodbc.so.1 nicht findet. Hier existiert in den meisten Systemen inzwischen die libodbc.so.2 . Auf diese Datei muss ein entsprechender Verweis mit dem Namen libodbc.so.1 in dem gleichen Verzeichnis abgespeichert werden.

In den für das System notwendigen Dateien odbcinst.ini und odbc.ini müssen Einträge ähnlich den folgenden existieren:

odbcinst.ini

```
001 [MySQL]
002 Description = ODBC Driver for MySQL
003 Driver = /usr/lib64/libmyodbc5.so
```

odbc.ini

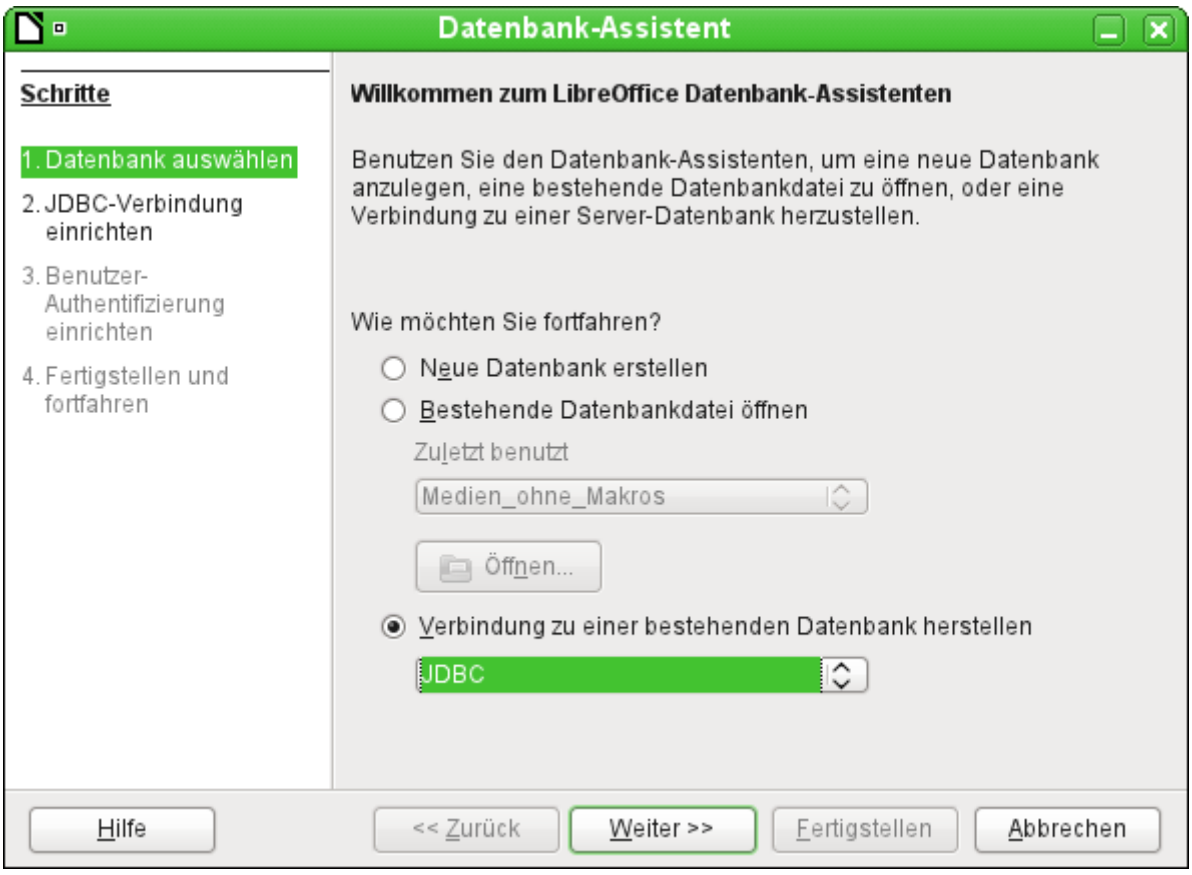
```
001 [MySQL-test]
002 Description = MySQL database test
003 Driver = MySQL
004 Server = localhost
005 Database = libretest
006 Port = 3306
007 Socket =
008 Option = 3
009 Charset = UTF8
```

Diese beiden Dateien liegen bei einem Linux-System im Verzeichnis /etc/unixODBC. Ohne den Eintrag zur Art des verwendeten Zeichensatzes kommt es bei Umlauten zu Problemen, auch wenn die Einstellungen in MySQL/MariaDB und Base übereinstimmen.

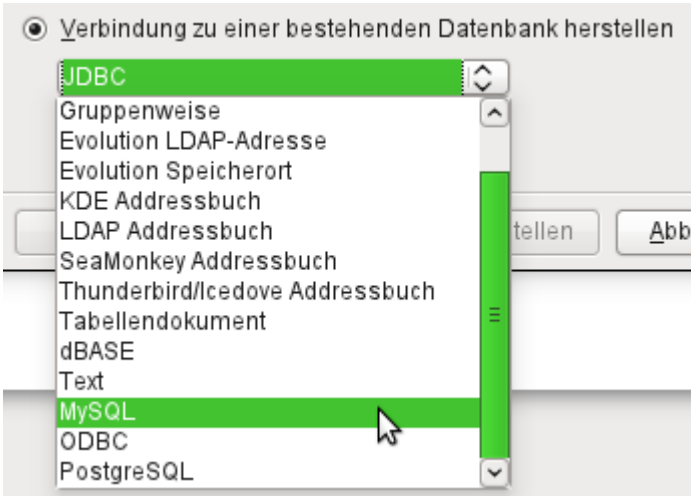
Details zu den Verbindungsparametern sind im *MySQL-Referenzhandbuch* zu finden.

Verbindung zur MySQL-Datenbank über den Datenbank-Assistenten

Der Zugriff auf eine existierende MySQL-Datenbank erfolgt mit der direkten Verbindung in den folgenden Schritten:



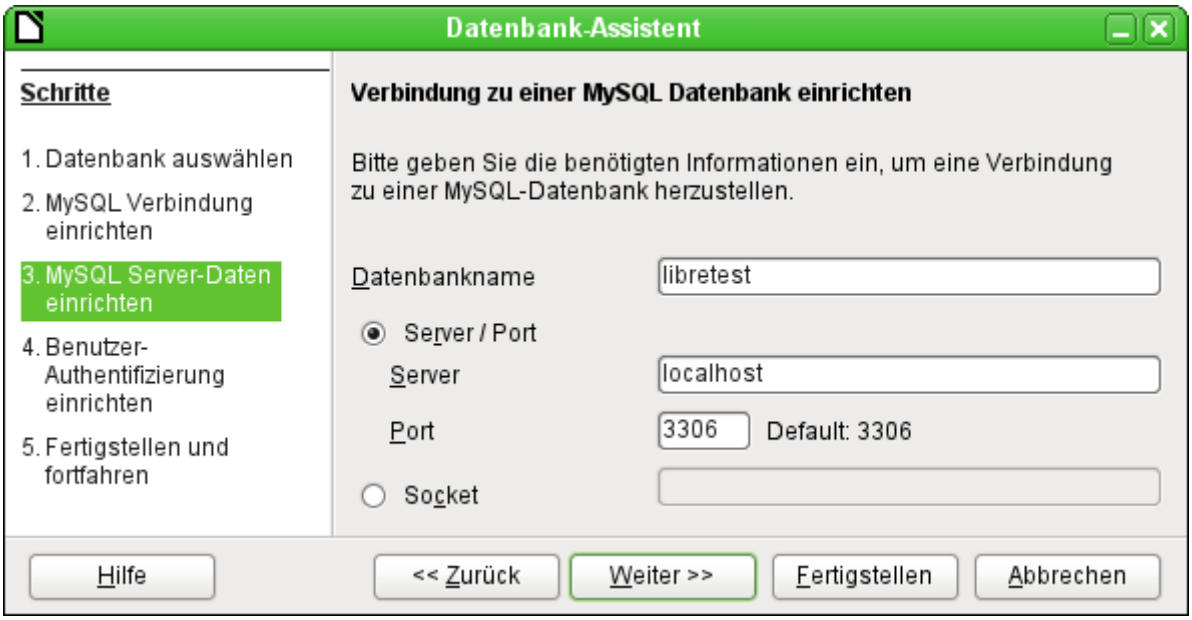
Über «*Neue Datenbank erstellen*» ist nur die Erstellung einer Datenbank im internen HSQLDB-Format möglich. Die Zusammenarbeit mit anderen Datenbanken kann nur erfolgen, wenn die Datenbank selbst bereits existiert. Dazu muss also **Verbindung zu einer bestehenden Datenbank herstellen** gewählt werden.



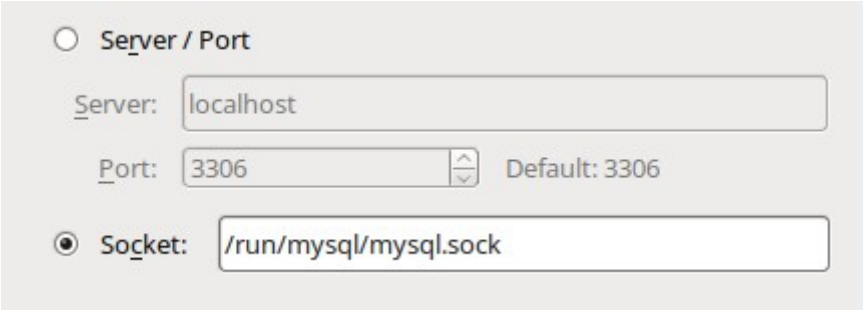
Hier wird aus den, teilweise betriebssystemspezifischen, Datenbanken die MySQL-Variante ausgewählt.

Die direkte Verbindung

Die direkte Verbindung ist diejenige, die von der Geschwindigkeit her für die **MariaDB** am besten gewählt werden sollte. Allerdings hat sie auch ein paar kleine Bugs, die gegebenenfalls besonders berücksichtigt werden müssen. So ist es z. B. dringend erforderlich, in Formularen bei Textfeldern die maximale Länge einzustellen, weil sonst die Eingabe einfach auf den bisher maximal eingegebenen Wert gekürzt wird.



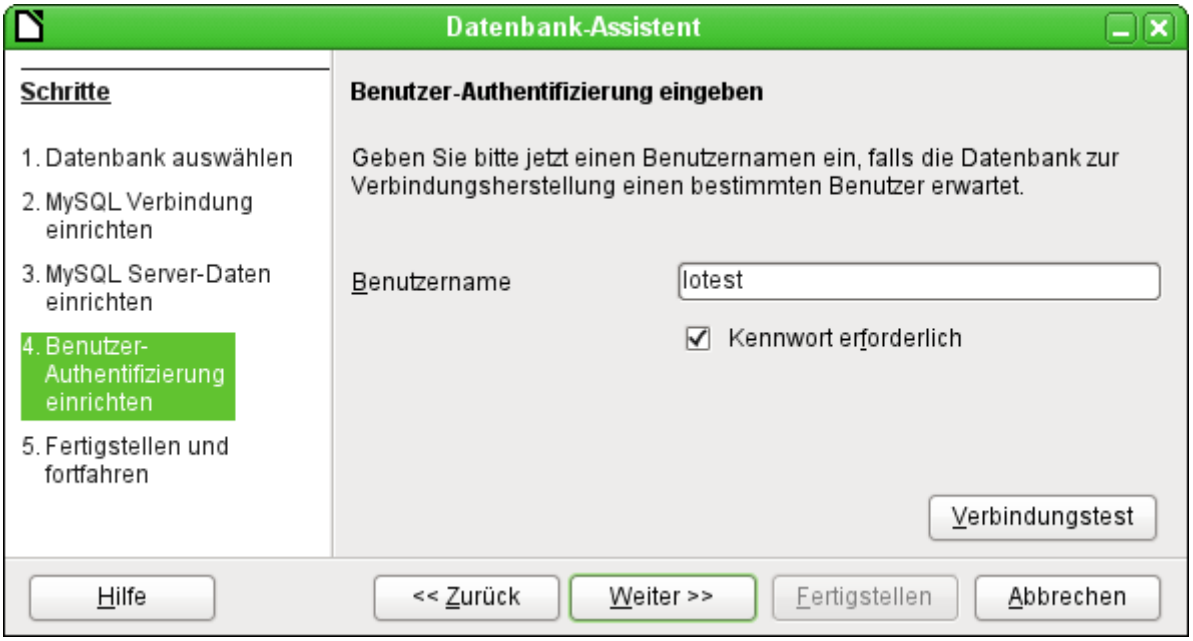
Der Datenbankname muss bekannt sein. Befindet sich der Server auf dem gleichen Rechner wie die Benutzeroberfläche, unter der die Datenbank erstellt wird, so kann als Server «localhost» gewählt werden. Ansonsten kann die IP-Adresse (z.B. 192.168.0.1) oder auch je nach Netzwerkstruktur der Rechnername oder gar eine Internetadresse angegeben werden. Es ist also ohne weiteres möglich, mit Base auf die Datenbank zuzugreifen, die vielleicht auf der eigenen Homepage bei irgendeinem Provider liegt.



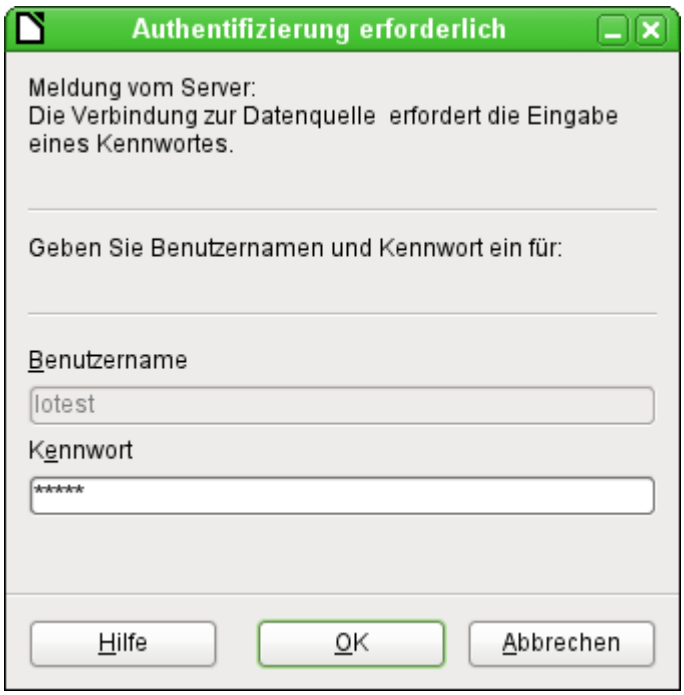
Alternativ zu den Serverangaben kann (unter Linux) auch direkt der Socket eingegeben werden. Der Standardsocket kann dafür aus der Datei /etc/my.cnf ausgelesen werden.

Auch der Socket benötigt einen laufenden MariaDB/MySQL-Server. Es ist also nicht möglich, ohne gestarteten Server auf die Datenbanken in dem Server zuzugreifen. Auch die Nutzung einer anderen Instanz als der des systemweiten Servers ist nicht möglich.

Bei der Arbeit mit Base über das Internet sollte sich der Nutzer allerdings darüber bewusst sein, wie seine Verbindung zu der Datenbank gestaltet ist. Gibt es eine verschlüsselte Verbindung? Wie erfolgt die Passwortübertragung?

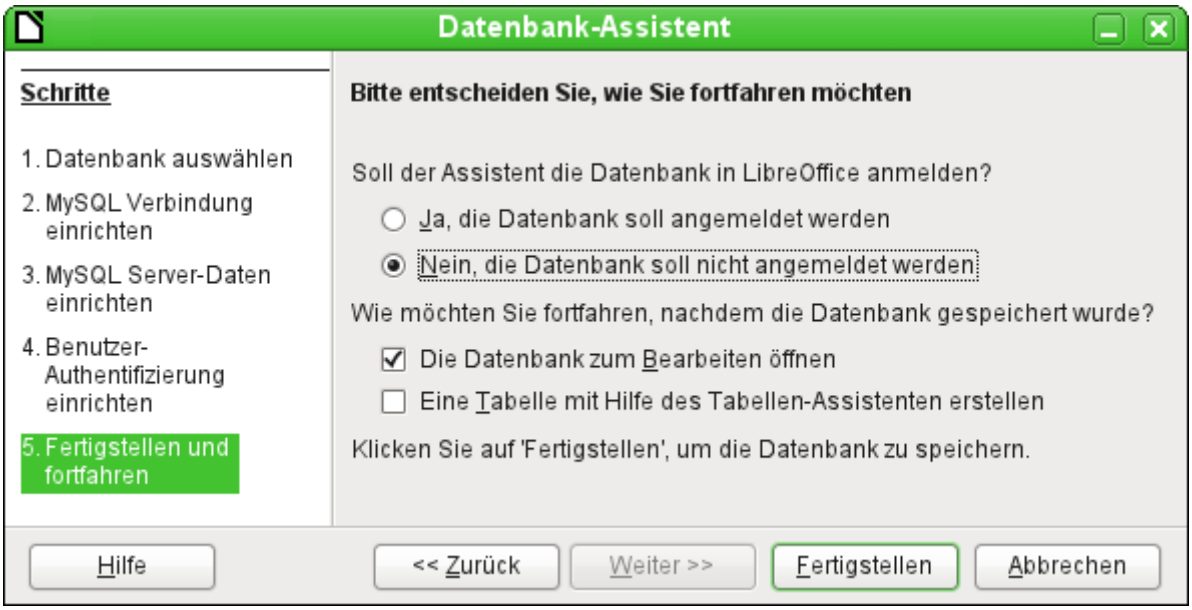


Jede über das Netz erreichbare Datenbank sollte mit einem Benutzernamen und einem Kennwort geschützt sein. Hier wird direkt getestet, ob die Verbindung klappt. Wichtig ist natürlich, dass der entsprechende Benutzer in MySQL bzw. der MariaDB entsprechend für den benannten Server eingerichtet wurde.



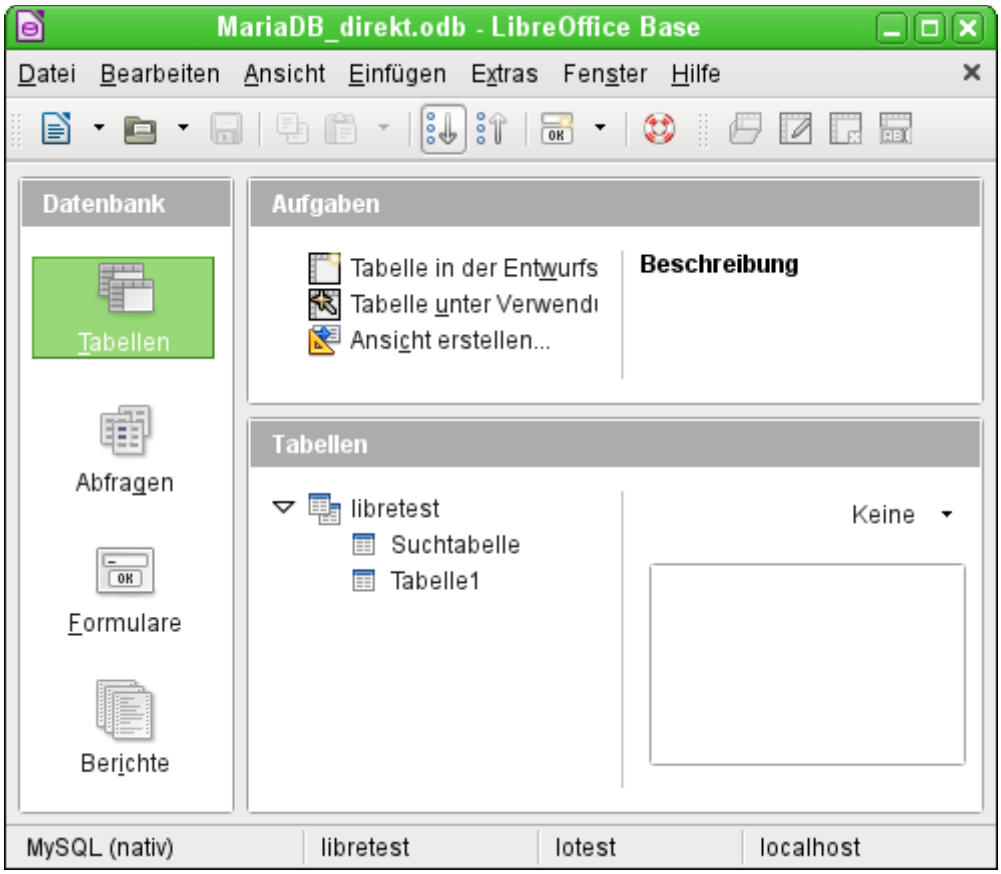
Dieses Fenster erscheint anschließend bei jedem Start der Datenbankdatei, wenn das erste Mal auf die MySQL-Datenbank zugegriffen wird.

Der Verbindungstest startet die Authentifizierung mit vorgegebenem Benutzernamen. Nach Passworteingabe erfolgt die Meldung, ob die Verbindung erfolgreich hergestellt werden kann. Läuft z.B. MySQL zur Zeit nicht, so kommt natürlich an dieser Stelle eine Fehlermeldung.



Auch hier wird die Datenbank nicht angemeldet, da sie nur zu ersten Tests aufgebaut wurde. Eine Anmeldung ist erst notwendig, wenn andere Programme wie z.B. der Writer für einen Serienbrief auf die Daten zugreifen sollen.

Der Assistent beendet die Verbindungserstellung mit dem Abspeichern der gewünschten Datenbankverbindung. In dieser *.odb-Datei liegen jetzt lediglich diese Verbindungsinformationen, die bei jedem Datenbankstart ausgelesen werden, so dass auf die Tabellen der MySQL-Datenbank zugegriffen werden kann.



Ansicht der geöffneten Datenbankdatei mit Tabellenübersicht und in der Fußzeile der Benennung des verwendeten Treibers «MySQL (nativ)», des Datenbanknamens «libretest», des Nutzers der Datenbank «lotest» und des Servers, auf dem die Datenbank läuft, nämlich «localhost».

Über Fertigstellen wird die Base-Datei erstellt und die Ansicht auf die Tabellen der MySQL-Datenbank geöffnet. Die Tabellen der Datenbank werden unter dem Namen der Datenbank selbst aufgeführt.

Beim manchen Treibern wird nur die Datenbank «libretest» angezeigt, für die auch die Verbindung bestimmt war. Andere Treiber von LO bieten auch andere MySQL- bzw. MariaDB-Datenbanken auf dem gleichen Server zur Auswahl, für die der Nutzer mindestens eine Leseberechtigung hat.

Auch mit den Treibern für nur eine Datenbank ist aber ein Zugriff auf die anderen Tabellen z.B. für Abfragen möglich, sofern natürlich der angegebene Datenbanknutzer, im obigen Fall also «lotest», mit seinem Passwort auf die Daten zugreifen kann. Im Gegensatz zu den bisherigen nativen LO-Treibern ist hier allerdings kein schreibender Zugriff auf andere Datenbanken des gleichen MySQL-Datenbankservers möglich.

Im Unterschied zu der internen Datenbank von Base taucht bei Abfragen entsprechend in MySQL für die Definition der Tabelle immer auch der Datenbankname auf, hier z.B.

```
001 ... FROM "libretest"."Klasse" AS "Klasse" ...
```

Hier ist es also auf jeden Fall notwendig, der Kombination aus Datenbankname und Tabellenname mit dem Zusatz «AS» einen alternativen Alias-Namen zuzuweisen. Genaueres siehe dazu in dem Kapitel *Verwendung eines Alias in Abfragen*.

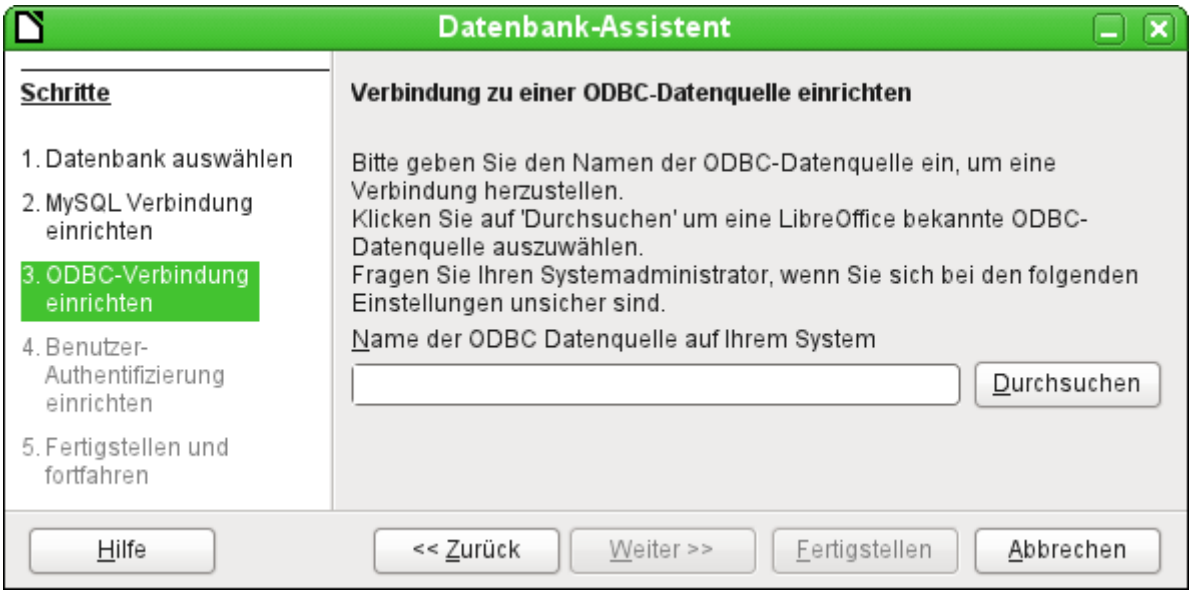
Hinweis

Durch diese Aliaszuweisung werden Abfragen, die aus mehreren Tabellen bestehen, nicht mehr editierbar, auch wenn alle anderen Bedingungen erfüllt sind. Dann muss in der durch die GUI erstellten Abfrage im SQL-Code **`"libretest"."Klasse" AS`** (also der Hinweis auf den Datenbanknamen) entfernt werden. Dann gelingt auch eine bearbeitbare Abfrage über mehrere Tabellen.

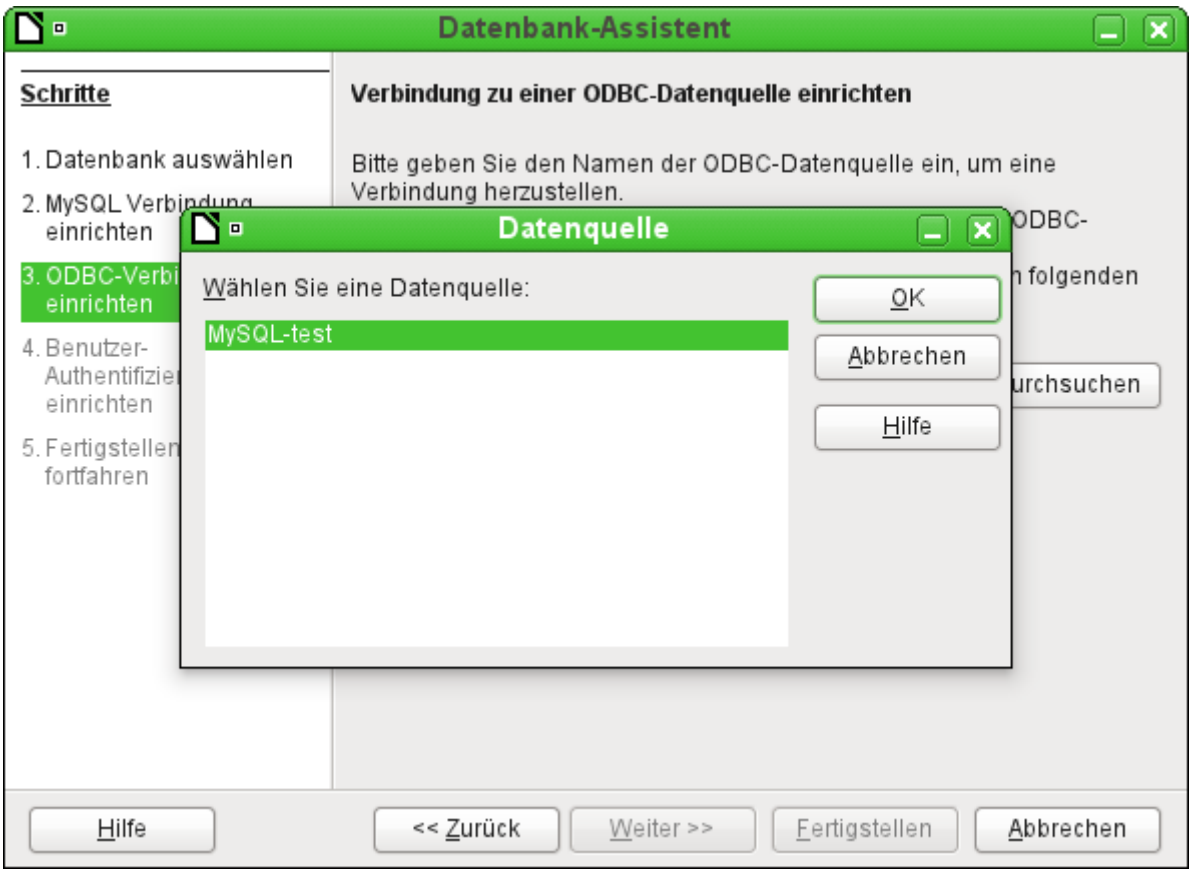
Tabellen können in der Datenbank erstellt und gelöscht werden. Automatisch hoch zählende Zahlenfelder («Autowert») funktionieren und lassen sich auch bei der Tabellenerstellung auswählen. Sie starten bei MySQL mit dem Wert '1'.

Die ODBC-Verbindung

Die ersten Schritte zur ODBC-Verbindung sind gleich denen zur direkten Verbindung. Wird beim zweiten Schritt dann die ODBC-Verbindung für MySQL gewählt, dann erscheint das folgende Fenster des Datenbank-Assistenten:



Die ODBC Datenquelle hat nicht unbedingt den gleichen Namen wie die Datenbank in MySQL selbst. Hier muss der Name eingetragen werden, der auch in der Datei «odbc.ini» steht. Die einfachste Möglichkeit besteht hier darin, über den Button Durchsuchen den Namen aus der «odbc.ini» direkt auszulesen.

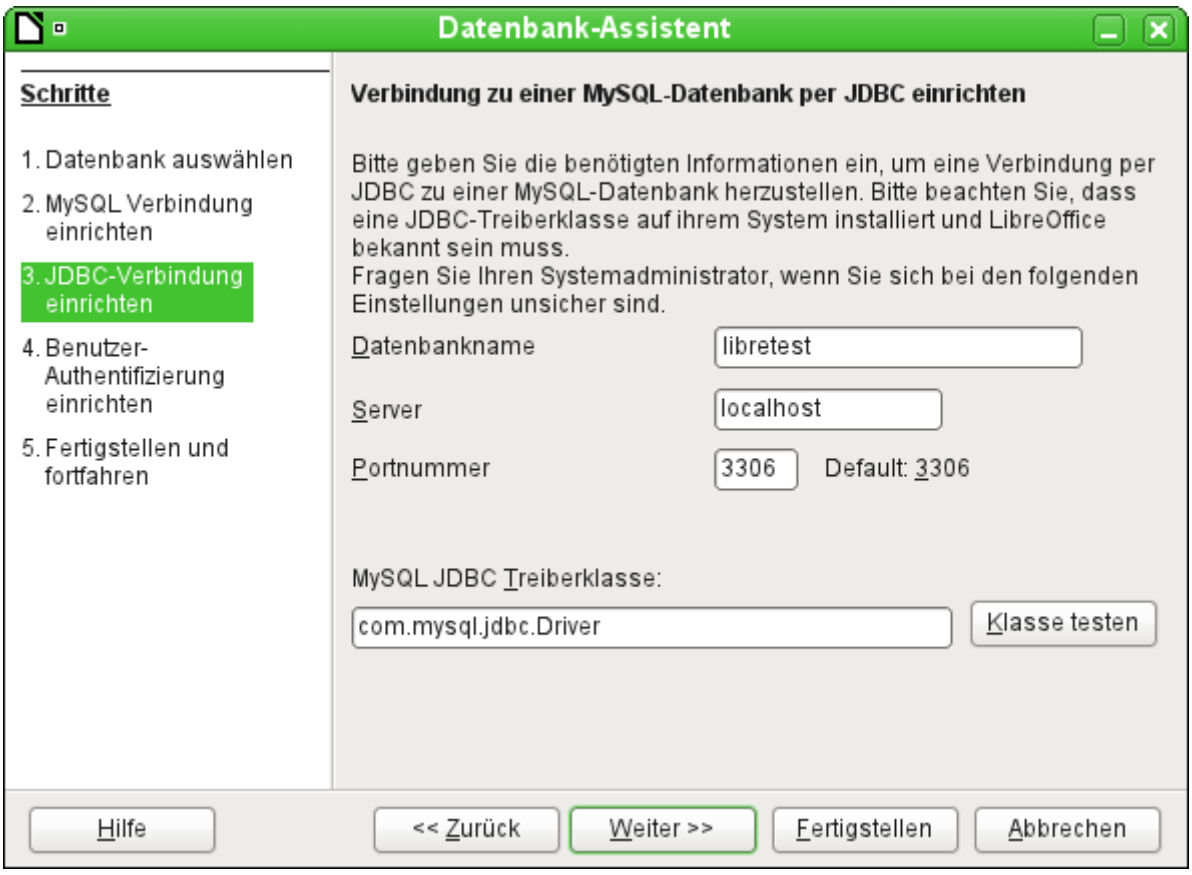


Beim Test erscheint der Name aus der «odbc.ini». Auch wenn hier wieder nur mit einer Datenbank verknüpft wird, können andere Tabellen des MySQL-Servers sehr wohl gelesen werden.

Schritt 4 und 5 laufen wieder identisch zur Direktverbindung ab.

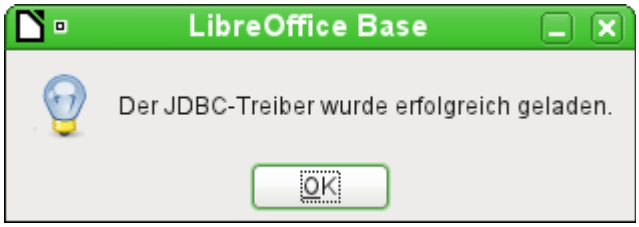
Die JDBC-Verbindung

Auch bei der JDBC-Verbindung sind die ersten Schritte gleich, hier unterscheidet sich wieder nur Schritt 3 von den anderen Schritten des Assistenten:



Der Assistent fragt hier die gleichen Informationen ab wie bei der direkten Verbindung. Der Datenbankname ist der, der auch in MySQL selbst verwandt wird.

Über «Klasse testen» wird überprüft, ob das Archiv mysql-connector-java.jar über Java erreichbar ist. Entweder muss dieses Archiv im Pfad der ausgewählten Java-Version liegen oder direkt in LibreOffice eingebunden werden.



Alle weiteren Schritte sind wieder identisch zu denen der vorherigen Verbindungen. Die Verbindungen zu anderen Datenbanken des gleichen MySQL-Datenbankservers funktionieren ebenfalls nur lesend.

Hinweis

Bei Verwendung des MariaDB-Treibers muss die JDBC-Treiberklasse mit «org.mariadb.jdbc.Driver» angegeben werden.
Für MySQL wird in neueren Version nach der Version 5 die JDBC-Treiberklasse «com.mysql.cj.jdbc.Driver» benötigt.

Hinweis

Der JDBC-Zugang mit MySQL/MariaDB kann auch direkt über die Auswahl **JDBC** statt des Untermenüs **MySQL → JDBC** erfolgen. Dies kann besonders dann sinnvoll sein, wenn dem Treiber Parameter mitgegeben werden sollen.
Ohne Parameter trennt z.B. der JDBC-Treiber wie auch der direkte Treiber zu einer MySQL-Datenquelle im Internet nach recht kurzen Pausenzeiten die Verbindung. Mit der folgenden Einstellung wird dies vermieden:

```
001 jdbc:mysql://«Host der Datenbank»:3306/«Datenbankname»?autoReconnect=true
```

Bei neueren Treibern (ab Version 8.*) für MySQL ist die Verbindung wegen einer Zeitzoneneinstellung nur über die Angabe von Parametern möglich, wenn nicht Servereinstellungen beeinflusst werden können:

```
001 jdbc:mysql://localhost/«Datenbankname»?
    autoReconnect=true&useUnicode=true&useJDBCCompliantTimezoneShift=true
    &useLegacyDatetimeCode=false&serverTimezone=UTC
```

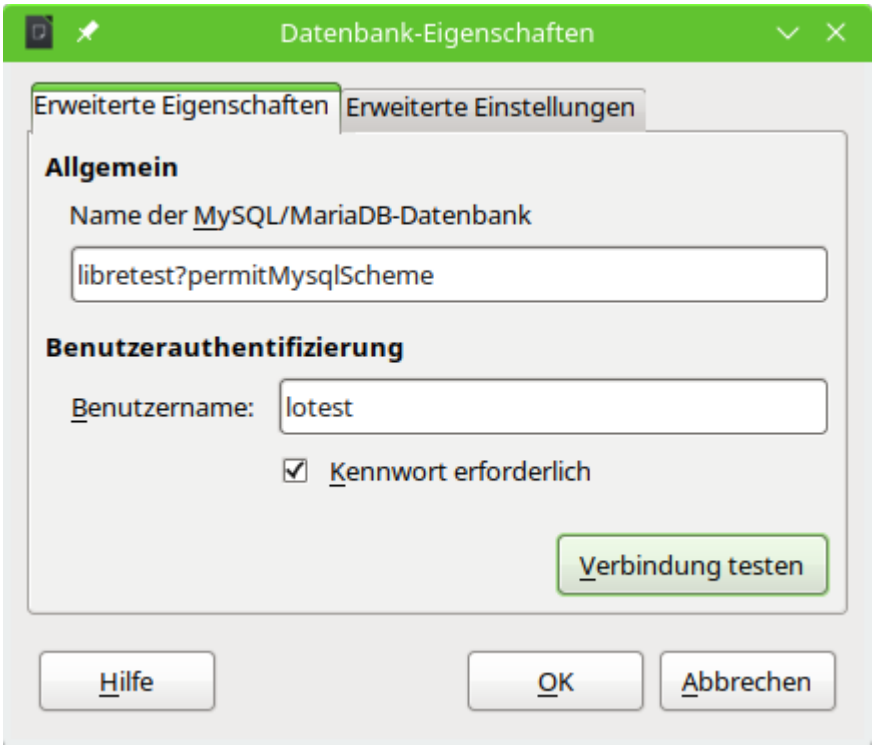
Dies bietet alle erforderlichen Einstellungen, zusätzlich auch noch die Einstellung des Zeichensatzes.
Kommt eine Fehlermeldung wie «RSA public key is not available on client side (option serverRsaPublicKeyFile not set)», dann hilft eventuell

```
001 jdbc:mysql://localhost:3306/«Datenbankname»?
    allowPublicKeyRetrieval=true&useSSL=false
```

Eine Übersicht aller Einstellungen bietet *https://dev.mysql.com/doc/connector-j/5.1/en/connector-j-reference-configuration-properties.html#connector-j-reference-set-config*

Hinweis

JDBC-Treiber für die MariaDB ab MariaDB Connector/J 3.0.0 benötigen zur Kompatibilität mit den bisherigen Verbindungsstrings zum Datenbanknamen den Zusatz **`?permitMysqlScheme`**. Dieser Zusatz kann direkt bei der Erstellung der Verbindung oder nachträglich über **Bearbeiten → Datenbank → Eigenschaften → Erweiterte Eigenschaften → Allgemein** dem Namen der Datenbank hinzugefügt werden.



Verbindung zu MySQL/MariaDB über das Internet

Die verschiedenen Verbindungen zu MySQL/MariaDB wurden oben jeweils mit der Beziehung zum örtlichen Server «localhost» getestet. Sie funktionieren grundsätzlich auch als Verbindungen zu Datenbanken eines Providers im Internet. Allerdings ist hier zu beachten, dass der Server des Providers für Kontakte von außerhalb ein **wait_timeout** eingestellt haben kann. Nach einer entsprechenden Zeit ohne Datenaustausch wird dann die Verbindung unterbrochen. Je nach Einstellung des Servers kann es erlaubt sein, diese Zeitspanne zu beeinflussen.

```
001 SHOW SESSION VARIABLES LIKE 'wait_timeout';
002 SET SESSION wait_timeout=600;
```

Zuerst wird die Einstellung des Servers über **Extras → SQL** ermittelt. Die Standardeinstellung eines Servers ohne entsprechende Konfiguration liegt bei 28800 Sekunden, also 8 Stunden. Ist dies nicht der Fall, dann kann für die Session eine Zeitspanne (hier: 600 Sekunden oder 10 Minuten) eingegeben werden. Sobald in dieser Zeit wieder eine Abfrage läuft wird die Zeitspanne bis zur nächsten Unterbrechung wieder auf 10 Minuten gesetzt.

Ist die Einstellung der Zeitspanne nicht möglich, so hilft das folgende Makro:

```
001 GLOBAL boStop AS BOOLEAN
```

```
001 SUB Reconnect
002    DIM oDatasource AS OBJECT
003    DIM oConnection AS OBJECT
004    DIM oSQL_Command AS OBJECT
005    boStop = false
006    DO
007       WAIT 30000 'Zeitangabe in Millisekunden
008       oDatasource = thisDatabaseDocument.CurrentController
009       oConnection = oDatasource.ActiveConnection()
010       oSQL_Command = oConnection.createStatement()
011       oSQL_Command.executeQuery("SELECT NOW()")
012    LOOP WHILE boStop = false
013 END SUB
```

```
001 SUB StopConnect
002    boStop = true
003 END SUB
```

Zuerst wird eine globale Variable erstellt, damit die Schleife in der Prozedur **Reconnect** auch unterbrochen werden kann. In der Prozedur wird alle 30 Sekunden eine einfache Abfrage gestellt, die nicht weiter ausgewertet wird. So bleibt der Kontakt zum Server erhalten, wenn **wait_timeout** größer als 30 Sekunden ist und die Netzverbindung einwandfrei funktioniert. Die Einstellung der Zeitspanne ist hier dem Test überlassen. Mit der Prozedur **StopConnect** schließlich lässt sich die Prozedur **Reconnect** wieder unterbrechen.

Wird dieses Makro mit dem Öffnen der Datenbank aufgerufen, so muss natürlich innerhalb der ersten 30 Sekunden auf jeden Fall der erste Kontakt zur Datenbank mit Passworteingabe hergestellt werden.

Bei der JDBC-Verbindung lässt sich hier mit dem Parameter **autoReconnect=true** eine Verbindung direkt wieder herstellen. Die erste Unterbrechung findet dann trotzdem statt, bei einem zweiten Kontaktversuch ist die Verbindung aber wieder da.

Bei der direkten Verbindung funktioniert das nicht. Stattdessen muss die Datenbankdatei geschlossen und wieder geöffnet werden um erneut einen Kontakt herzustellen. Die Verbindungsdaten wie Nutzername und Passwort müssen aber nicht neu eingegeben werden, solange LibreOffice selbst nicht beendet wird..

Zugriff auf gespeicherte Prozeduren in MySQL/MariaDB

Auf dem Datenbankserver können neben Tabellen und Ansichten auch Prozeduren gespeichert werden. Werden diese Prozeduren direkt auf der Konsole von MySQL aufgerufen, so können dort zum Teil Tabellenansichten ähnlich einer sonst in Base sichtbaren Ansicht («View») abgeru-

fen werden. In Base sind diese Tabellenansichten nicht sichtbar. Ein Aufruf der Prozeduren unter **Extras → SQL** ist problemlos möglich, nur erfolgt leider keine Ausgabe der anzuzeigenden Inhalte. Um Inhalte der gespeicherten Prozeduren («Stored Procedures») zu sehen kann der folgende Weg beschritten werden:

- Statt einer Prozedur wie
  ```
  CREATE PROCEDURE AlleNamen()
  BEGIN
     SELECT * FROM `libretest`.`Namen`;
  END
  ```
  wird das Ergebnis der Prozedur in eine temporäre Tabelle geschrieben:
  ```
  CREATE PROCEDURE AlleNamen()
  BEGIN
     DROP TEMPORARY TABLE IF EXISTS `libretest`.`TempNamen`;
     CREATE TEMPORARY TABLE `libretest`.`TempNamen` AS SELECT *
        FROM `libretest`.`Namen`;
  END
  ```
- Anschließend wird unter **Extras → SQL** die Prozedur aufgerufen:
  ```
  CALL AlleNamen();
  ```
- Jetzt befinden sich die Ausgabedaten in einer temporären Tabelle, die nur für den aktuellen Nutzer von MySQL sichtbar ist. Die Ausgabedaten werden in einer Abfrage über
  ```
  SELECT * FROM `libretest`.`TempNamen`
  ```
  sichtbar.

Die Abfrage funktioniert nur dann einwandfrei, bei der Erstellung der Prozedur auch die Datenbank (in diesem Fall "libretest") mit angegeben wird. Wie das Ganze etwas besser automatisiert ablaufen kann ist im Kapitel *MySQL-Datenbank mit Makros ansprechen* beschrieben.

PostgreSQL

Für die PostgreSQL-Datenbank gibt es wie bei MySQL/MariaDB einen direkten Treiber von LO, der von vornherein mit installiert wird. Damit der Kontakt auch sicher passt, zuerst aber eine kurze Anleitung zu den ersten Schritten nach der Installation von PostgreSQL.

Erstellen eines Nutzers und einer Datenbank

Die folgenden Schritte sind nach einer Installation über den Paketmanager in OpenSUSE erforderlich. Sie ähneln vermutlich denen unter anderen Betriebssystemen.

1. Dem Nutzer «postgres» muss zuerst ein Passwort zugewiesen werden. Das kann mit dem Administrationstool des Betriebssystems erfolgen.
2. Der PostgreSQL-Server muss vom Administrator gestartet werden, z. B. **`service postgresql start`** oder **`rcpostgresql start`**.
3. Der Nutzer «postgres» muss sich mit
   `su postgres`
   auf der Konsole anmelden.
4. Ein einfacher Datenbanknutzer, hier «lotest», sollte mit Passwortzugang erstellt werden:
   `createuser -P lotest`
5. Damit der Datenbanknutzer anschließend auch auf die zu gründende Datenbank zugreifen kann, muss in der Datei /var/lib/pgsql/data/**pg_hba.conf** ein Eintrag geändert werden. In dieser Datei werden u.a. die Methoden zur Identifizierung der Nutzer auf verschiedenen Ebenen festgelegt. Die Methode, mit der LO-Base kommunizieren kann, ist die Methode «password» oder «md5», nicht, wie voreingestellt, die Methode «ident».
6. Der Systemnutzer «postgres» meldet sich über «psql» an:
   `psql -d template1 -U postgres`

7. Der Systemnutzer «postgres» erstellt die Datenbank «libretest»:
   `CREATE DATABASE libretest;`

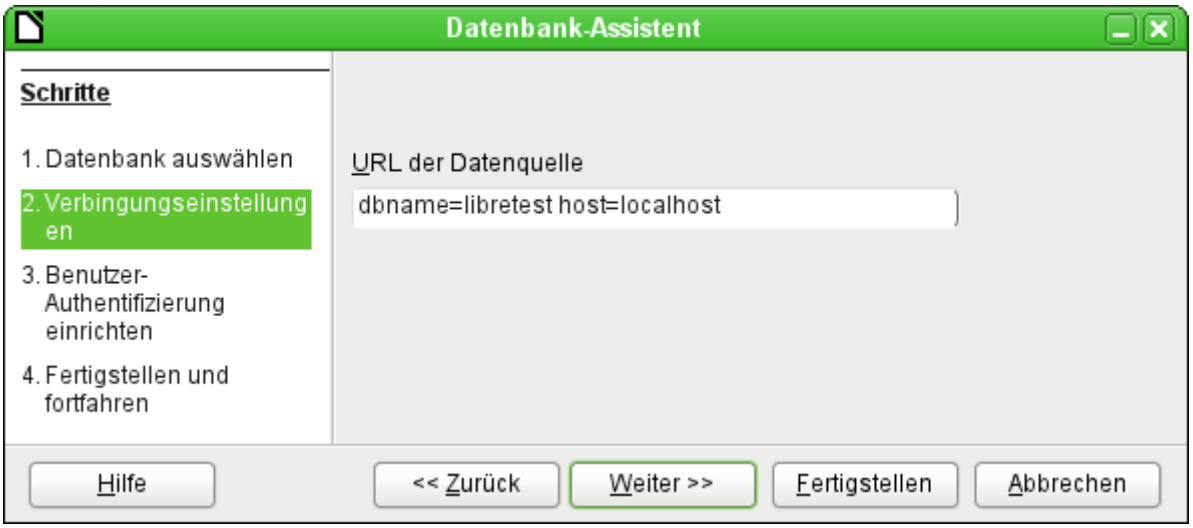
Der Nutzer «lotest» hat nicht die gleichen Rechte wie der Nutzer «postgres». Meldet sich der Nutzer «postgres» direkt beim System über **`su postgres`** an, so kann er beispielsweise zusätzliche Schemas für eine Datenbank erstellen:

1. Der Systemnutzer «postgres» meldet sich über «psql» an:
   `psql -d libretest -U postgres`
2. Der Systemnutzer «postgres» erstellt das Schema «reports»:
   `CREATE SCHEMA reports;`
3. Der Systemnutzer «postgres» setzt den Nutzer «lotest» als Eigentümer für das Schema ein:
   `ALTER SCHEMA reports OWNER TO lotest;`

Der Nutzer «lotest» kann jetzt auf dieses Schema zugreifen. Allerdings erscheint es nicht direkt in der Übersicht. Wird aber über die GUI in Base eine Tabelle erstellt, so kann diese in dem Schema «reports» gespeichert werden und anschließend wird das Schema auch in der Liste der Tabellen angezeigt.

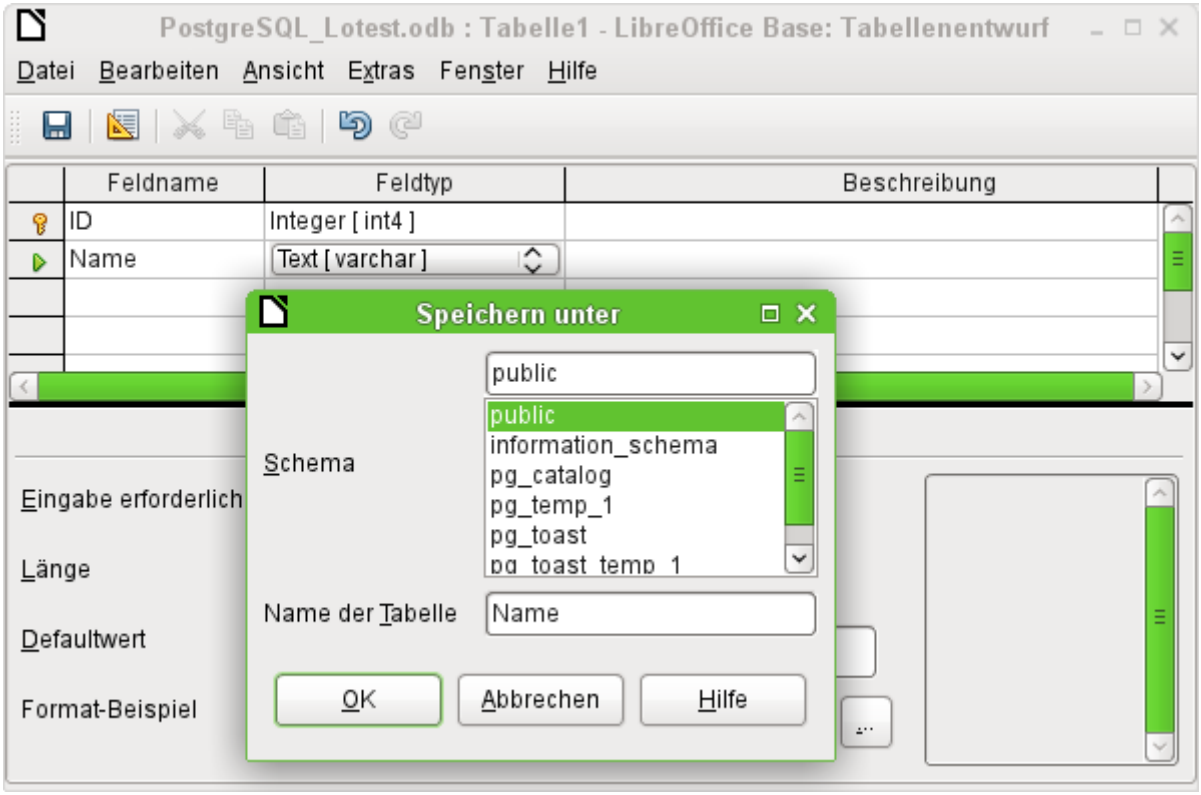
Direkte Verbindung zu PostgreSQL

Im Assistenten wird der Eintrag «postgres» ausgewählt. Die Verbindungseinstellung geschieht über die Angabe des Datenbanknamens («dbname») und des Hostes. Unter bestimmten Umständen ist es erforderlich, statt der einfachen Angabe eines Hostes den kompletten Host einschließlich des vollständigen Domainnamens anzugeben. Gegebenenfalls muss auch noch separat «port=5432» hinzugefügt werden. Dies sollte aber nur notwendig sein, wenn ein anderer als der Standardport gewählt wird.



Die Benutzer-Authentifizierung sieht genauso aus wie die unter MySQL. Neben «dbname» und «host» können hier noch «port», «connect_timeout» und andere Einträge erstellt werden.

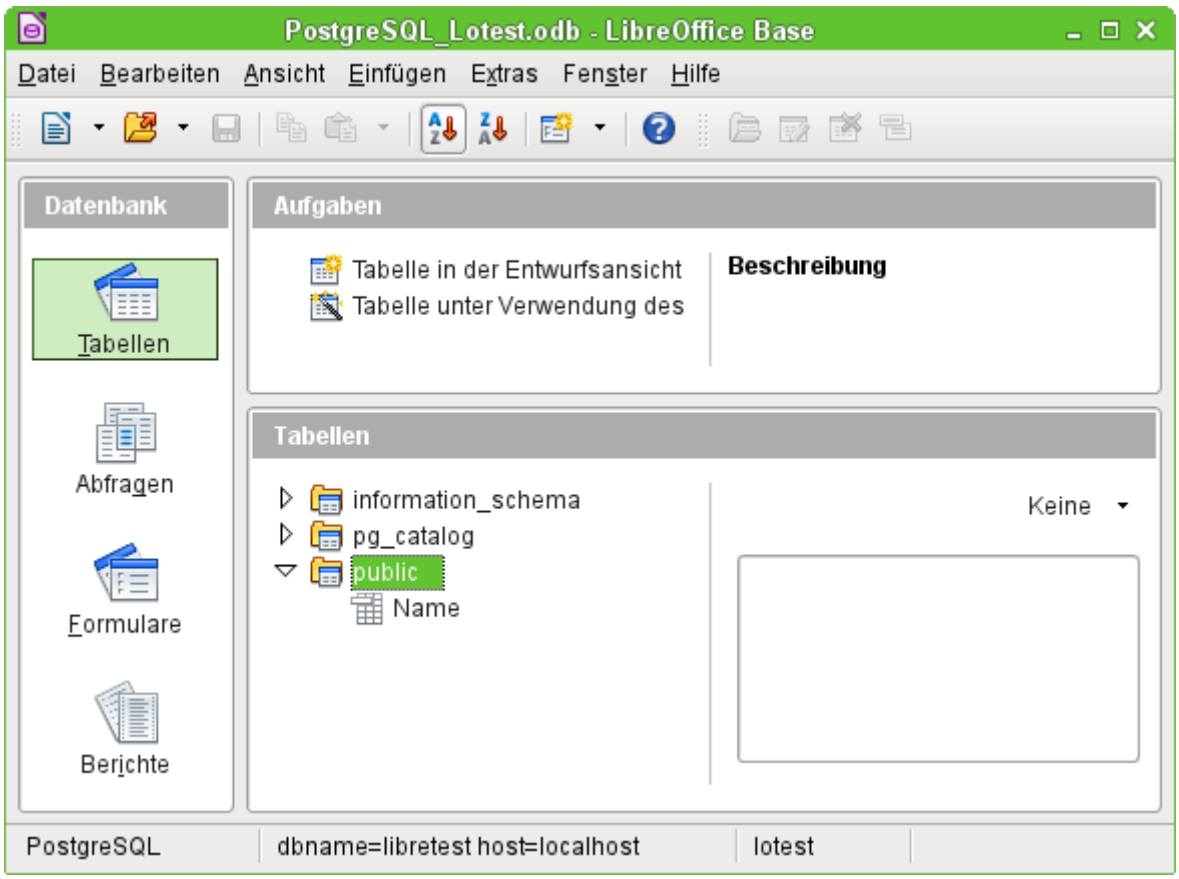
Wird die Datenbank das erste Mal geöffnet, so erscheinen für den User erst einmal eine Unmenge an Tabellen aus dem Bereich «information_schema». Diese Tabellen sind ebenso wie die im Bereich «pg_catalog» für den normalen Nutzer nur lesbar, nicht schreibbar. Diese verschiedenen Bereiche bezeichnet PostgreSQL als «Schema». Der Nutzer legt neue Tabellen im Schema «public» an.



Der Dialog «Speichern unter» zeigt hier verschiedene Schema von PostgreSQL auf. Tatsächlich speicherbar ist allerdings nur in dem Schema «public», solange keine weitreichenderen Nutzerrechte für den entsprechenden Anmeldenamen vergeben wurden.

Hinweis

Werden Tabellen z.B. aus der internen HSQLDB nach PostgreSQL kopiert, so schlägt der Importassistent den einfachen Tabellennamen aus der HSQLDB, z.B. «Tabelle1», vor. Der Import mit diesem Namen schlägt allerdings fehl. Stattdessen muss vor den Namen die Schemabezeichnung ergänzt werden: aus «Tabelle1» wird «public.Tabelle1».



In der Tabellenübersicht von PostgreSQL erscheinen die unterschiedlichen Schemata. Im Schema «public» ist die abgespeicherte Tabelle «Name» zu sehen.

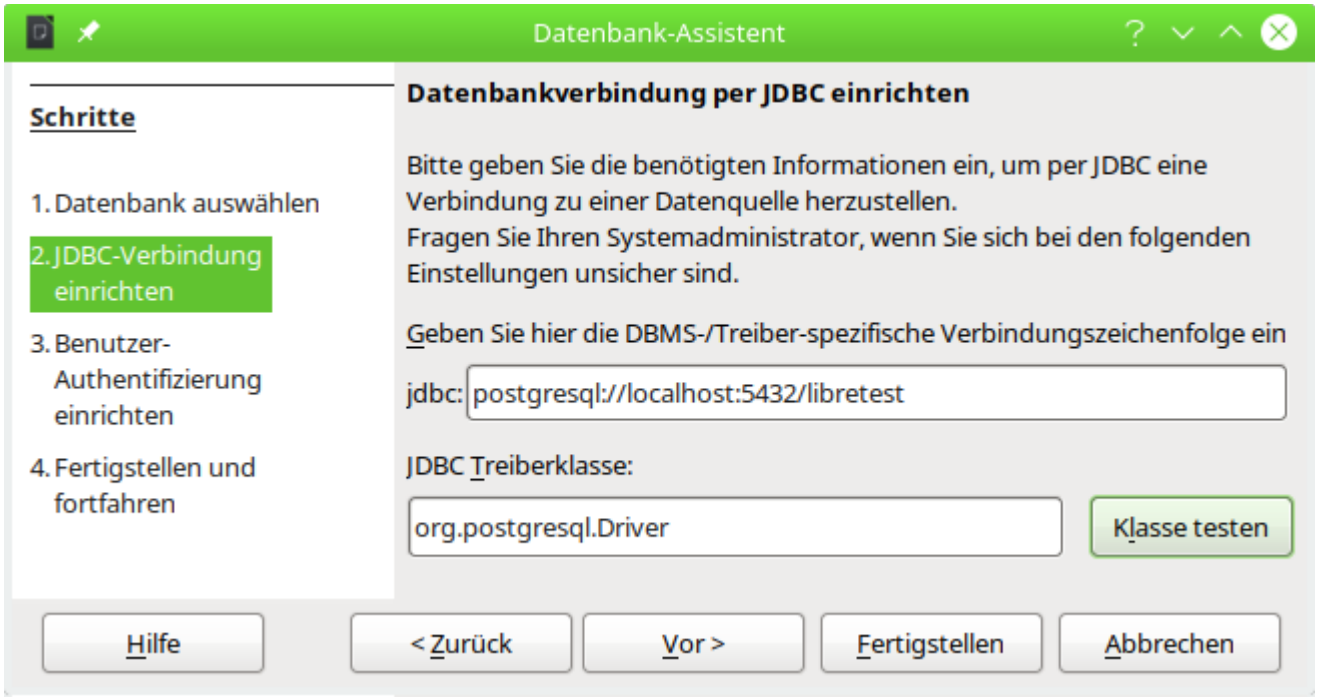
Hinweis

Die Ansicht auf andere Verzeichnisse als das Verzeichnis "public" kann auch über **Extras → Tabellenfilter** eingeschränkt werden.

PostgreSQL-Verbindung über JDBC

Neben dem direkten Treiber ist natürlich auch die Verbindung über JDBC oder ODBC möglich.

Während der direkte Treiber z. B. keine Erstellung von Ansichten in der GUI erlaubt oder immer mit der Tabellenansicht in dem Tabellenverzeichnis "information_schema" beginnt, lässt die **JDBC**-Verbindung die Erstellung von Ansichten zu und zeigt standardmäßig nur das Tabellenverzeichnis mit dem Schemanamen "public" oder andere selbst erstellte Schemata an. Nur in diesem Bereich werden schließlich neue Tabellen des Nutzers erstellt. Auch ist der JDBC-Treiber der einzige, der in der GUI automatisch hoch schreibende Primärschlüsselfelder in der GUI erzeugt.



Hier die direkte Eingabe der Datenbankverbindung für einen lokalen Server zur Datenbank «libretest».

Der **JDBC**-Treiber wiederum hat Probleme bei der Verwendung von Abfragen mit Aliasbezeichnungen für Tabellen. Wenn ein Primärschlüssel automatisch hoch geschrieben werden soll, so wird dies bei Abfragen mit Aliasbezeichnungen nicht korrekt ausgelesen. Dies führt dann zu angeblichen 0-Werten als Primärschlüssel und Datenverlust, wenn nach dem ersten Abspeichern Daten in so einem Datensatz geändert werden.

PostgreSQL-Verbindung über und ODBC

Aus diesem Grund hier auch noch funktionierende Standardeinstellungen für die **ODBC**-Verbindung von PostgreSQL. Leider ist auch diese Verbindung nicht frei von Mängeln. So lassen sich hier über die GUI keine Ansichten erstellen, solange noch keine Ansicht existiert. Dem kann aber abgeholfen werden, indem eine erste Dummyansicht über **Extras → SQL** erstellt wird:

```
004 CREATE VIEW "public"."vtest" AS SELECT * FROM "public"."Tabellenname"
```

Wird danach die Datenbankdatei abgespeichert und wieder neu geladen, so können Ansichten auch über die GUI mit der **ODBC**-Verbindung erstellt werden.

In den für das System notwendigen Dateien odbcinst.ini und odbc.ini müssen Einträge ähnlich den folgenden existieren:

odbcinst.ini

```
001 [PSQL]
002 Description=PostgreSQL
003 Driver64=/usr/lib64/psqlodbcw.so
004 UsageCount=1
```

odbc.ini

```
001 [PostgreSQL-libretest]
002 Description = PostgreSQL connection to libretest
003 Driver = PSQL
004 Database = libretest
005 Servername = localhost
006 ReadOnly = No
007 RowVersioning = No
```

```
008 ShowSystemTables = No
009 ConnSettings = SET SEARCH_PATH TO libretest;
```

Mit diesen Einstellungen ist eine Verbindung unter Linux möglich. Serielle Felder, die in PostgreSQL der Ersatz für Autoincrement-Werte sind, wurden anstandslos ausgelesen. Auch die Nutzung von Aliasbezeichnungen in Abfragen beeinflussten nicht, wie bei dem JDBC-Treiber, die korrekte Rückgabe der Werte nach Neueingaben.

PostgreSQL hat die Eigenart, dass innerhalb einer Datenbank unterschiedliche Schemata für unterschiedliche Zwecke angelegt werden können. Häufig wird ein anderer Schemaname als «public» beim Import von Daten aus anderen Datenbanken gewählt. Der Eintrag für die ConnSettings bewirkt, dass alle Schemata aus der Datenbank «libretest» angezeigt werden. Soll nur ein Schema angezeigt werden, so wird der Name dieses Schemas, das sich in der Datenbank befindet, eingetragen.

Verbindung zu PostgreSQL über das Internet

Neben den Hinweisen zur *Verbindung zu MySQL/MariaDB über das Internet* existiert auch bei PostgreSQL eine direkte Möglichkeit, die Dauer einer Verbindung zu beeinflussen. Allerdings bleibt hier bei einem Standardwert von '0' die Verbindung beständig bestehen. Diese Variable ist erst ab der Version PostgreSQL 14 verfügbar.

```
001 SHOW idle_session_timeout;
002 SET idle_session_timeout TO '10000';
```

Die Angabe ohne Maßeinheit bezieht sich hier auf Millisekunden. Würde also statt der '0' eine '10000' gesetzt, so würde die Verbindung nach einer Zeit von 10 Sekunden Untätigkeit automatisch unterbrochen. Eine Einstellung dieser Variablen für einen entsprechenden fortdauernden Kontakt ist also nicht sinnvoll, da standardmäßig keine Unterbrechung erfolgt.

```
001 SHOW idle_in_transaction_session_timeout;
002 SET idle_in_transaction_session_timeout TO '10000';
```

Mit dieser Funktion wird eine Abfrage oder andere Aktion unterbrochen, wenn sie die entsprechende Zeit in Millisekunden benötigt hat. Zuerst wird nachgeschaut, auf welchen Wert diese Form des Timeouts steht. Auch hier setzt eine '0' eine Unterbrechung komplett aus. Falls es bei Tests vorkommt, dass eine fehlerhafte Abfrage das System zu lange beschäftigt, so kann mit dieser Einstellung entgegengewirkt werden.

Autoincrement-Werte bei PostgreSQL

PostgreSQL unterstützt nicht direkt Autoincrement-Werte. Stattdessen kann ein Feld, das automatisch hochzählende Werte generieren soll, als Feld des Typs **SERIAL** definiert werden.

```
001 CREATE TABLE "public"."Test" (
002 "ID" SERIAL PRIMARY KEY
003 );
```

Auch über

```
001 CREATE TABLE "public"."Test" (
002 ID INTEGER PRIMARY KEY GENERATED BY DEFAULT AS IDENTITY
003 );
```

kann mit neueren Versionen von PostgreSQL ein AutoWert-Feld erzeugt werden.

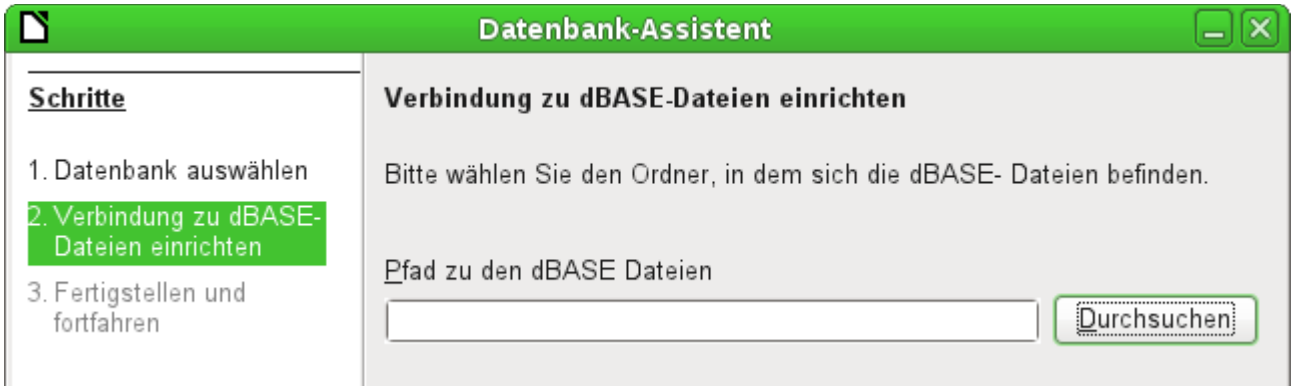
Hier wird ein automatisch hoch zählendes Feld "ID" als Primärschlüssel definiert. Die entsprechende Eingabe unter **Extras → SQL** erzeugt die Tabelle "Test". Wird anschließend über **Ansicht → Tabellen aktualisieren** die Anzeige der Tabelle ermöglicht, so kann die Tabelle in dem GUI-Editor anschließend weiter bearbeitet werden (**rechte Maustaste auf der Tabelle → Bearbeiten**).

Über den JDBC-Treiber ist es möglich, direkt Felder des Typs **SERIAL** zu erstellen. Unter **Bearbeiten → Datenbank → Erweiterte Einstellungen → Generierte Werte** darf dazu aber **kein Eintrag** erfolgen. Wird dort SERIAL eingetragen, so wird die Abspeicherung der Tabelle mit einer Fehler-

meldung unterbrochen. Der dort eingeblendete Begriff für die Erstellung des automatisch hoch zählenden Feldes muss dann entfernt werden, damit das Abspeichern gelingt.

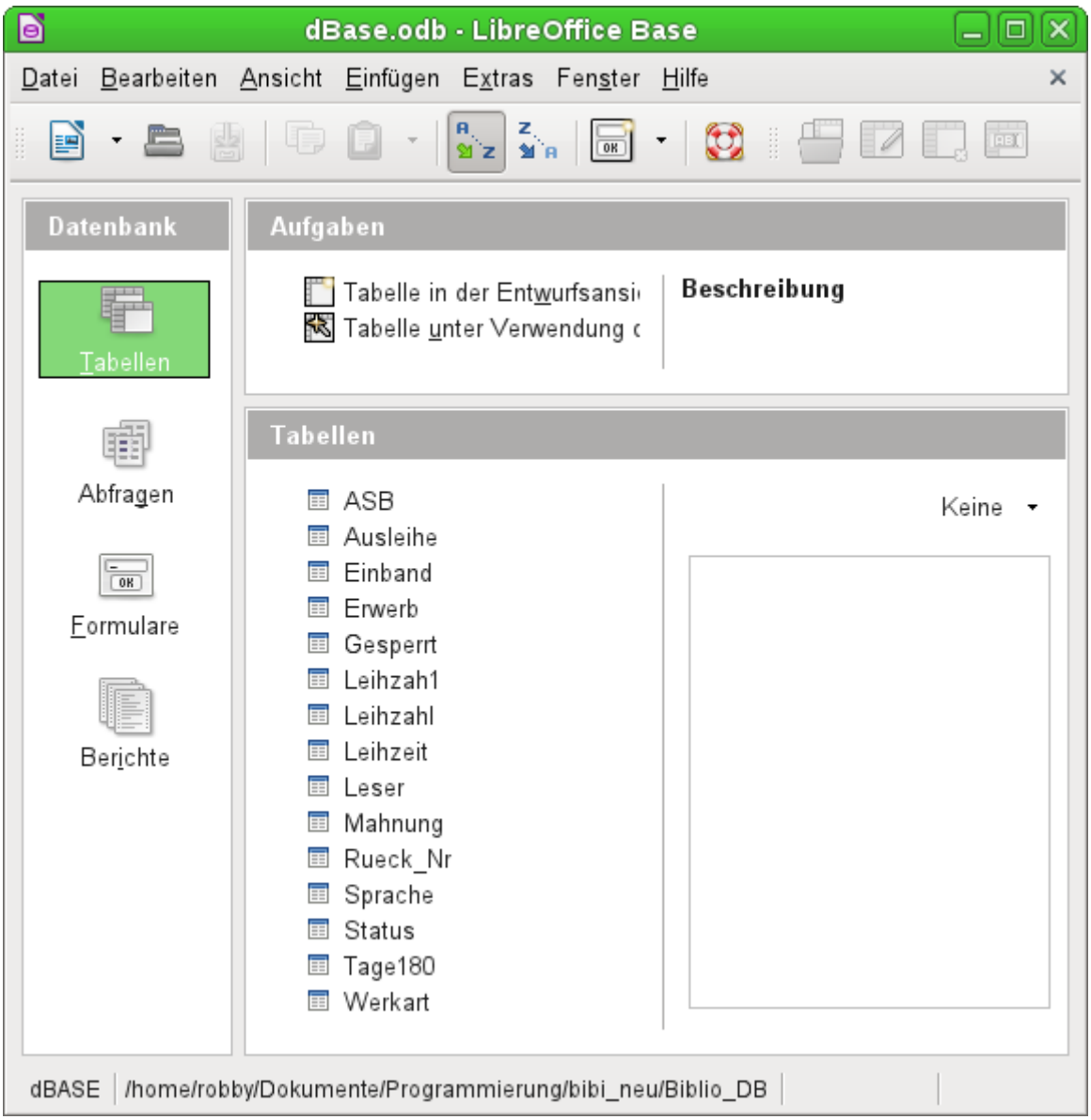
dBase-Datenbanken

Bei dBase handelt es sich um ein Datenbankformat, dass alle Daten in einem beliebigen Verzeichnis in separaten Tabellen bereitstellt. Verknüpfungen zwischen den Tabellen müssen über die Programmstruktur erledigt werden. dBase hat keine Möglichkeit, intern zu verhindern, dass z.B. in der Medien-Datenbank ein Medium gelöscht wird, der Verweis darauf aber weiterhin in der Tabelle zum Entleihen von Medien erhalten bleibt.



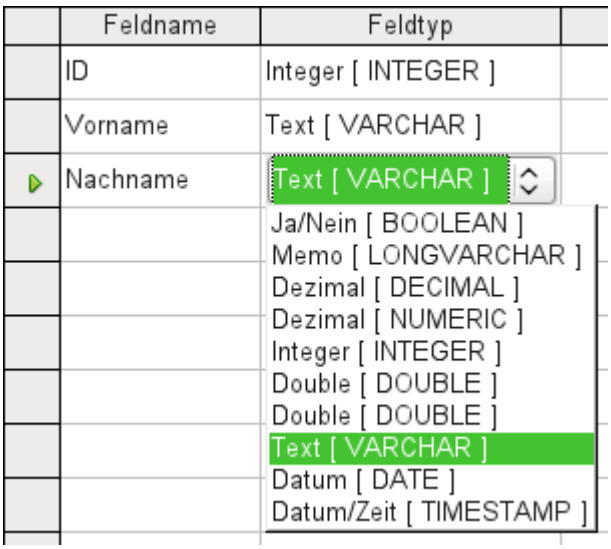
Die Verbindung wird zu einem Verzeichnis hergestellt. Alle *.dbf-Dateien, die sich in diesem Verzeichnis befinden, werden anschließend angezeigt und können über Formulare miteinander verbunden werden.

Tabellen in dBase haben kein unverwechselbares Feld für jeden Datensatz («Primärschlüssel»). Sie können also vom Prinzip her so beschrieben werden wie Tabellenblätter in Calc.



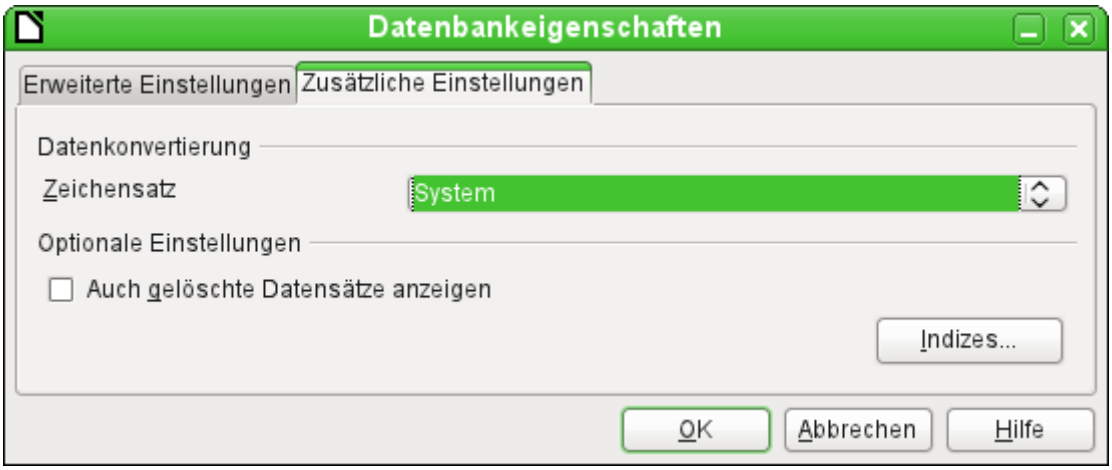
Tabellen können in Base neu erstellt werden. Sie werden dann einfach als neue Dateien in das angegebene Verzeichnis geschrieben.

Für neue Tabellen stehen deutlich weniger Feldtypen zur Verfügung als für die interne HSQLDB. Dabei sind noch einige Bezeichnungen der folgenden Abbildung als Verdoppelungen anzusehen.



dBase bietet sich besonders an, wenn es um Weitergabe und vielfältige Verarbeitungsmöglichkeit von Daten geht. Schließlich können auch Tabellenkalkulationen dBase-Tabellen direkt einlesen.

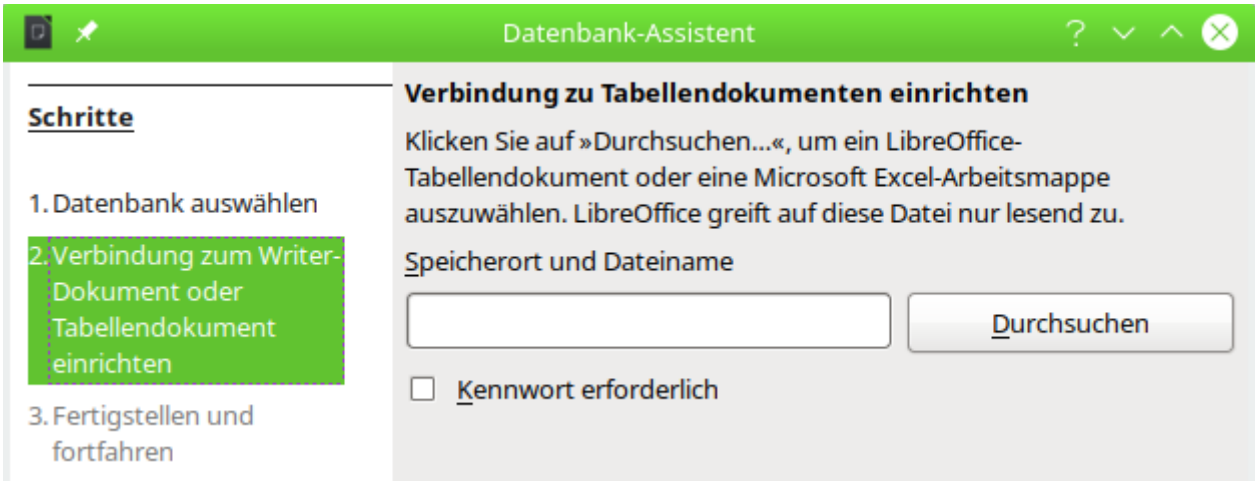
Base übernimmt einfach die Codierung, die dem Betriebssystem entspricht. Alte dBase-Dateien weisen dadurch leicht Fehler beim Import von Sonderzeichen auf. Der Zeichensatz kann anschließend über **Bearbeiten → Datenbank → Eigenschaften → Zusätzliche Einstellungen** entsprechend korrigiert werden:



Tabellendokumente und Tabellen in Writer-Dokumenten

Tabellenquellen für Datenbanken können auch Tabellendokumente sein. Wird aber eine Calc-Tabelle als Grundlage genommen, so ist jedes Editieren der Tabelle ausgeschlossen. Selbst wenn das Calc-Dokument auch noch geöffnet wird, wird dort ein Schreibschutz gesetzt.

Auch eine Verbindung zu Writer-Dokumenten ist möglich. Dort werden dann bestehende Tabellen in Base angezeigt. Auch diese sind wie die Calc-Tabellen schreibgeschützt. Der Name der Tabellen kann im Writer-Dokument über den Navigator oder die Tabelleneigenschaften festgelegt werden.



Der Dialog für die Verbindung zu Tabellendokumenten und Writer-Dokumenten ist gleich. Lediglich aus den Schritten ist erkennbar: Beide Dokumenttypen können genutzt werden.

Die einzige Abfrage ist letztlich, an welcher Position das Tabellendokument liegt und ob die Tabelle passwortgeschützt ist. Base öffnet dann das Tabellendokument und übernimmt alle Tabellenblätter mit ihrem Inhalt. Aus der ersten Zeile bildet Base die Feldbezeichnungen, aus den Tabellenblattbezeichnungen die Tabellenbezeichnungen. Auch Daten, die aus externen Quellen in Calc eingebunden sind (z. B. laufend aktualisierte Webseiten), lassen sich so in Base lesen.

Enthält das Tabellenblatt Datenbankbereiche, sichtbar über den Navigator oder **Daten → Bereich auswählen**, so werden diese Bereiche zusätzlich jeweils als Tabellen angezeigt.

Beziehungen zwischen den Tabellen lassen sich in Base nicht erstellen, da Calc nicht Grundlage für eine relationale Datenbank ist. Auch lassen sich Abfragen nicht über eine Tabelle hinaus erstellen.

Natürlich lassen sich in Calc vorbereitete Daten sehr wohl nach Base importieren und dort dann entsprechend verwerten. Siehe hierzu das Kapitel *Import von Daten aus anderen Datenquellen*.

Hinweis

Daten aus einer Calc-Tabelle werden in Base automatisch mit einem entsprechenden Format versehen. Allerdings wählt Base ein anderes Format als das Textformat nur dann, wenn alle Zeilen einen entsprechenden Wert enthalten. Das Textformat kann nicht geändert werden und kann dann bei weiteren Verarbeitungen zu Problemen führen, wenn z. B. Zahlen in eine andere Datenbank übertragen werden sollen. Hier hilft es dann, die entsprechenden Standardformatierungen in Calc einzustellen. Das ist für Dezimalzahlen dann eben der Punkt als Dezimaltrenner und keine weitere Formatierung wie z. B. ein Währungszusatz.

Daten in Calc bearbeiten und in Base aktualisieren

Innerhalb von Base sind Daten aus Calc nicht editierbar. Sollen trotzdem Daten verändert werden, so sind die folgenden Schritte zu beachten:

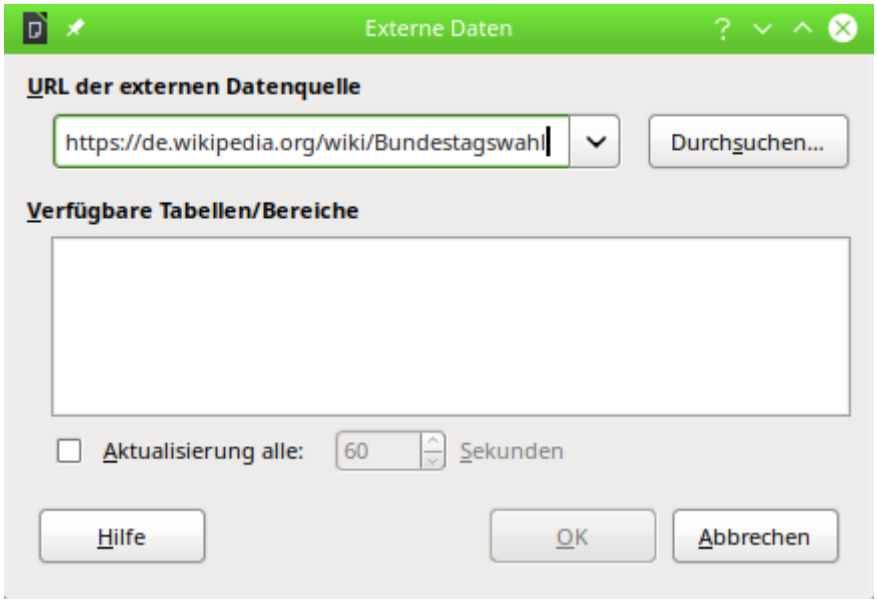
1. Die Base-Datei muss mit einem Zugriff auf die Tabellen geöffnet sein, bevor die damit verbundene Calc-Datei geöffnet wird.
2. Die Calc-Datei wird mit einem Schreibschutz geöffnet. Hier auf Dokument bearbeiten klicken.
3. Werden Daten im vorhandenen Datenbereich geändert, so muss in der Base-Tabelle der Button zum Aktualisieren (oder **Daten → Aktualisieren**) betätigt werden. Die geänderten Daten sind jetzt in der Base-Datei zu sehen. Vorsicht! Die Daten sind in Calc nicht abgespeichert.

4. Werden Daten unterhalb des Bereiches in einer neuen Zeile hinzugefügt, so bekommt die Tabelle in Base von der zusätzlichen Zeile nichts mit. In diesem Fall die Tabelle schließen und **Ansicht → Tabellen aktualisieren** im Hauptmenü der Base-Datei betätigen. Danach die Tabelle wieder öffnen und die zusätzliche Zeile erscheint.

Tabellen aus dem Internet über Calc automatisch aktualisiert in Base auslesen

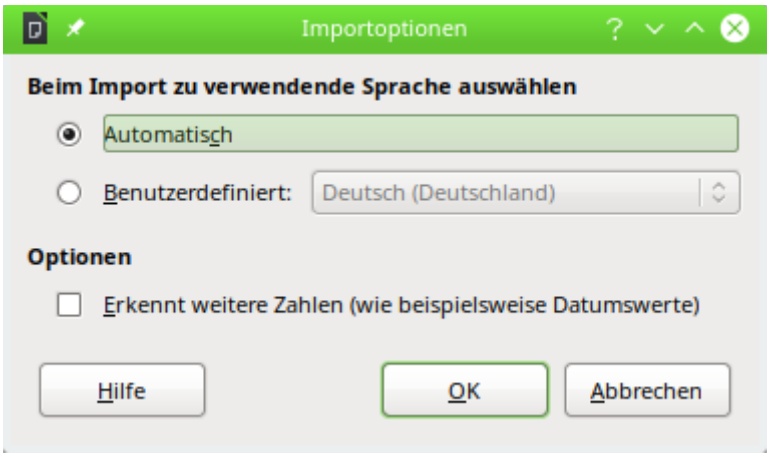
In Calc ist es möglich, auf externe Daten aus dem Internet zuzugreifen. Diese Daten lassen sich mit einer beständigen Aktualisierung versehen. So ist es auch möglich, mit Base eben über Calc auf solche Daten zuzugreifen.

In Calc wird **Tabelle → Verknüpfung zu externen Daten...** aufgerufen.

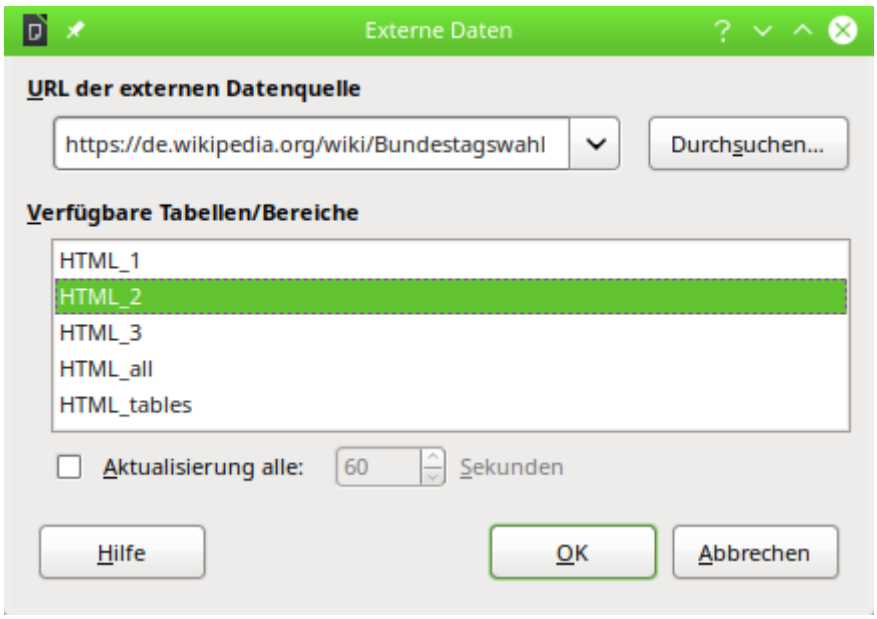


Hier wurde als Quelle der Wikipediaartikel zu Bundestagswahlen ausgesucht. Bei dem Artikel lohnt sich zwar nicht die Aktualisierung, aber das Prinzip lässt sich daran auch nachvollziehbar erklären.

Ist der Link zu der externen Datenquelle eingegeben, so muss diese Eingabe mit einem *Return* betätigt werden. Ansonsten passiert nichts. Erst danach beginnt die Suche nach vorhandenen Inhalten auf der Seite.

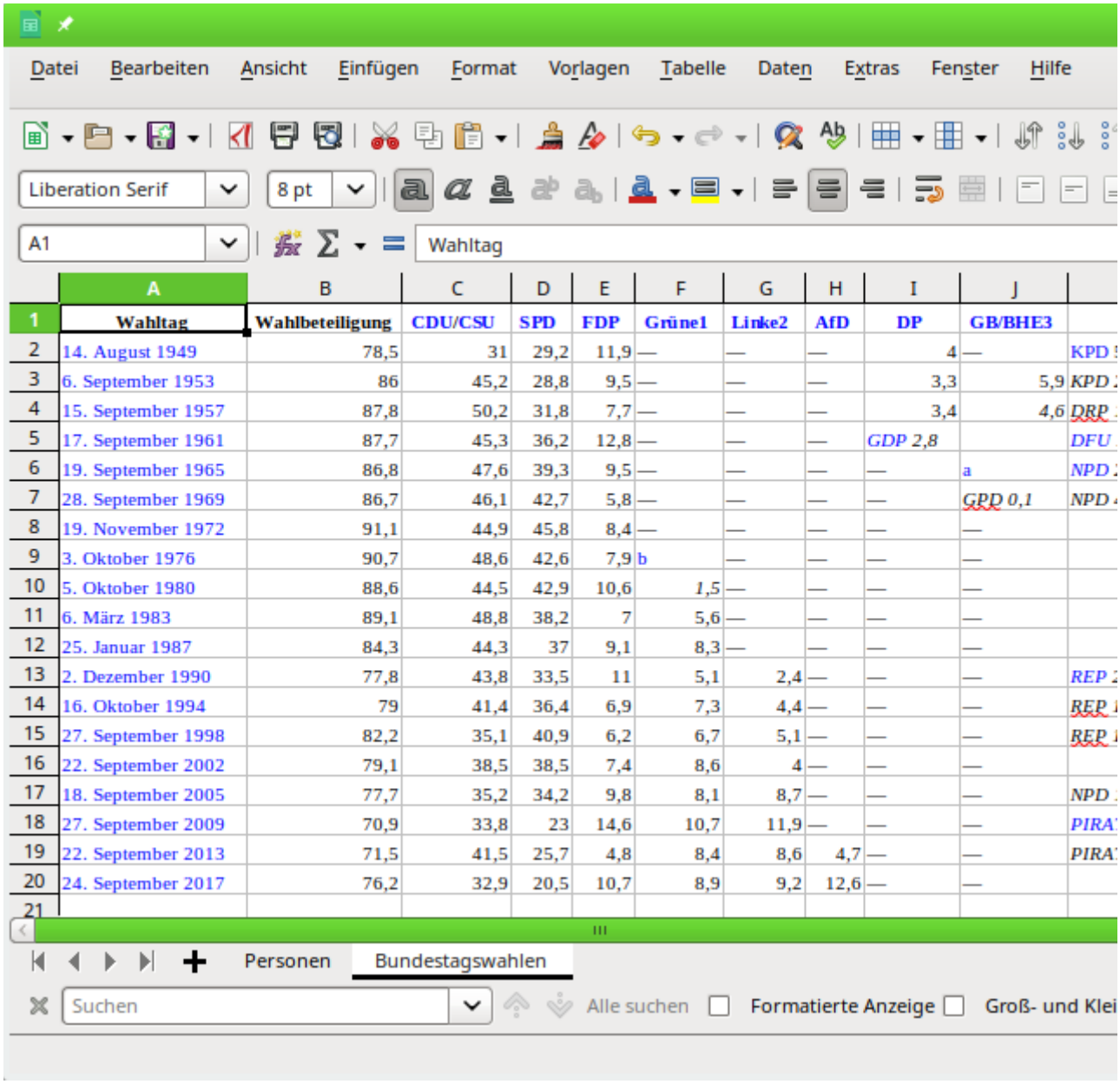


Wenn Calc entsprechende Informationen aus der Seite geladen hat erscheinen zuerst die Importoptionen. Da die Tabellen in Wikipedia in dem gewählten Beispiel sowieso eine Mischung aus Text und Zahlen darstellen wird auch Base daraus nur Textfelder machen. Deswegen ist hier z. B. das Erkennen weiterer Zahlen nicht aktiviert worden.



Die Tabellen in dem Wikipediaartikel werden von oben nach unten durchnummeriert. Hier ist die zweite Tabelle **HTML_2** ausgewählt, die die prozentualen Anteile für die verschiedenen Parteien bei sämtlichen Bundestagswahlen aufzeigt. Die unteren beiden Einträge sind für die Nutzung in Base nicht geeignet. Bei **HTML_all** wird der gesamte Inhalt ohne Rücksicht auf Tabellen geladen, bei **HTML_tables** alle 3 Tabellen auf eine Seite des Calc-Dokumentes gepackt, so dass Base daraus versuchen würde, eine Gesamttabelle zu erstellen.

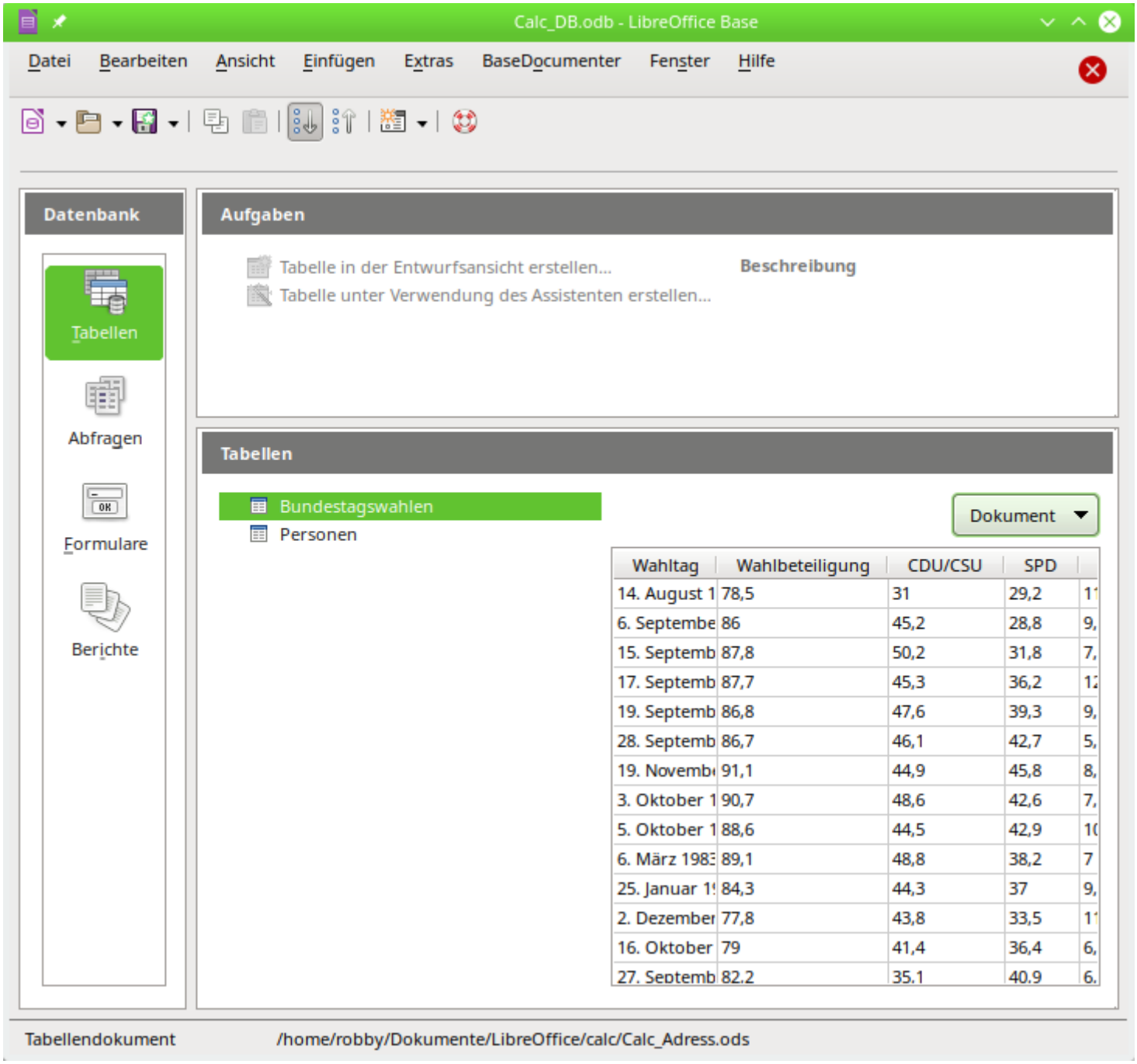
Jetzt könnte noch die automatische Aktualisierung eingestellt werden, so dass z. B. alle 60 Sekunden im Netz nachgeschaut wird, ob die Daten noch Bestand haben. So etwas macht wohl eher bei Aktienkursen als bei einer Tabelle zur Bundestagswahl Sinn.



	A	B	C	D	E	F	G	H	I	J	
1	Wahltag	Wahlbeteiligung	CDU/CSU	SPD	FDP	Grüne1	Linke2	AfD	DP	GB/BHE3	
2	14. August 1949	78,5	31	29,2	11,9	—	—	—	4	—	KPD
3	6. September 1953	86	45,2	28,8	9,5	—	—	—	3,3	5,9	KPD
4	15. September 1957	87,8	50,2	31,8	7,7	—	—	—	3,4	4,6	DRP
5	17. September 1961	87,7	45,3	36,2	12,8	—	—	—	GDP 2,8		DFU
6	19. September 1965	86,8	47,6	39,3	9,5	—	—	—	—	a	NPD
7	28. September 1969	86,7	46,1	42,7	5,8	—	—	—	—	GPD 0,1	NPD
8	19. November 1972	91,1	44,9	45,8	8,4	—	—	—	—	—	
9	3. Oktober 1976	90,7	48,6	42,6	7,9	b	—	—	—	—	
10	5. Oktober 1980	88,6	44,5	42,9	10,6	1,5	—	—	—	—	
11	6. März 1983	89,1	48,8	38,2	7	5,6	—	—	—	—	
12	25. Januar 1987	84,3	44,3	37	9,1	8,3	—	—	—	—	
13	2. Dezember 1990	77,8	43,8	33,5	11	5,1	2,4	—	—	—	REP
14	16. Oktober 1994	79	41,4	36,4	6,9	7,3	4,4	—	—	—	REP
15	27. September 1998	82,2	35,1	40,9	6,2	6,7	5,1	—	—	—	REP
16	22. September 2002	79,1	38,5	38,5	7,4	8,6	4	—	—	—	
17	18. September 2005	77,7	35,2	34,2	9,8	8,1	8,7	—	—	—	NPD
18	27. September 2009	70,9	33,8	23	14,6	10,7	11,9	—	—	—	PIRA
19	22. September 2013	71,5	41,5	25,7	4,8	8,4	8,6	4,7	—	—	PIRA
20	24. September 2017	76,2	32,9	20,5	10,7	8,9	9,2	12,6	—	—	
21											

Die so eingelesenen Daten brauchen nicht weiter formatiert zu werden. das geschieht in Base. Für die Datenbankdatei ist lediglich wichtig, wie die Tabelle, hier «Bundestagswahlen» heißt. Denn dies ist auch der so erkannte Tabellenname.

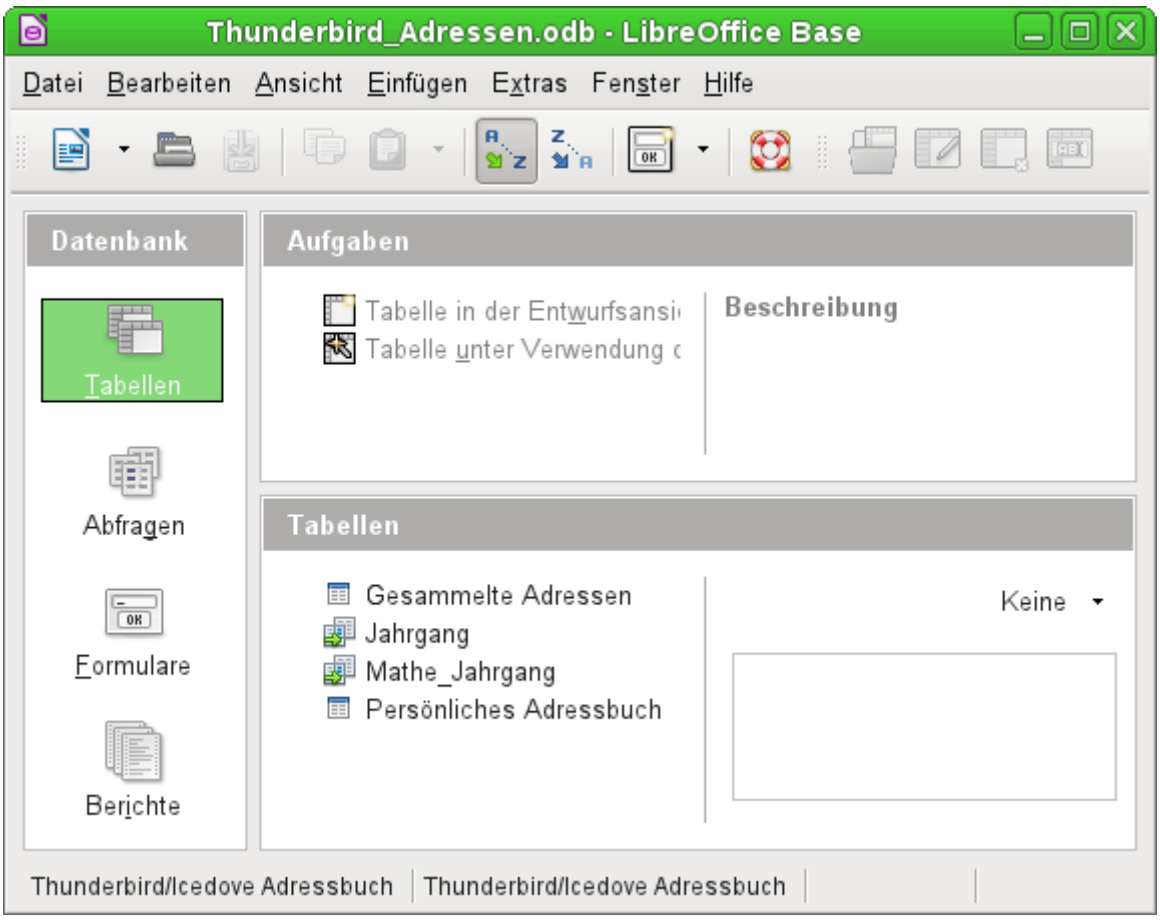
Bei Tabellen, die so in Calc eingelesen werden, werden die besten Ergebnisse erzielt, wenn die Vorlage im Internet die Daten sauber in Zeilen und Spalten darstellt. Zeilenumbrüche in einer Zelle führen zu zusätzlichen Datenzeilen, die dann leere Felder in den anderen Spalten hervorrufen. Zusätzliche unbedarfte Formatierungen der Urheber einer solchen Tabelle können dazu führen, dass nach jeder Zeile mit Inhalt eine Leerzeile entsteht. Das werden dann auch leere Tabellenzeilen in Base. Ein Löschen von Zeilen bringt hier nichts, da schließlich der Inhalt regelmäßig aktualisiert wird. Da müsste dann schon innerhalb von Calc aus der Importtabelle in eine weitere Tabelle eingelesen werden, die diese Formatierungsfehler ausgleicht.



In der Datenbank stehen jetzt die Namen der Tabelle aus dem Calc-Dokument. Hier wurde die Dokumentansicht für die Tabellen aktiviert, so dass ausschnitthaft direkt zu sehen ist, welcher Inhalt in der Tabelle angezeigt wird.

Thunderbird Adressbuch

Die Verbindung zu einem Adressbuch wie z.B. dem Thunderbird-Adressbuch sucht der Assistent automatisch. Er fragt lediglich nach einem Speicherort für die *.odb-Datei, die er erstellen soll.



Alle Tabellen werden dargestellt. Wie in Calc sind die Tabellen aber nicht editierbar. Base macht die Adressen damit nur für Abfragen und z. B. für die Verwendung in Serienbriefen zugänglich.

Die direkte Verbindung funktioniert nur zu Versionen vor Thunderbird 78. Die im Jahr 2020 erschienen Version ist auf SQLite als Datenbankmodul umgestiegen. Zum Kontakt zu dieser Datenbank siehe auch das Kapitel zu *SQLite*.

Die Verbindung über JDBC ist hier unter Linux am einfachsten zu erstellen:

JDBC URL: **`jdbc:sqlite:/home/<nutzername>/.thunderbird/….default/abook.sqlite`**

Treiberbezeichnung: **`org.sqlite.JDBC`**

Das Adressbuch «abook.sqlite» enthält die Adressen aus dem persönlichen Adressbuch. Existieren hier weitere Versionen wie «abook.v2.sqlite», «abook.v3.sqlite» usw., dann sind dies Backups, die zumindest teilweise bei erneutem Umstieg vom alten zum neuen Adressbauch entstanden sind. Das Adressbuch «history.sqlite» sollte dann die gesammelten Adressen enthalten.

Leider ist die Aufmachung der Tabellen in der SQLite-Version des Adressbuches nicht ganz so einfach gestaltet. Etwas problematisch ist auch, dass die Daten jetzt teilweise beschreibbar sind. Das kann bei einigen der Codes dazu führen, dass bei falscher Verknüpfung die Daten innerhalb des Adressbuches nicht mehr stimmig sind. Hier ist also Vorsicht geboten.

Die Daten sind nicht beschreibbar, wenn gleichzeitig Thunderbird geöffnet ist. Beim JDBC-Treiber kann die Beschreibbarkeit auch dadurch unterbunden werden, wenn **Bearbeiten → Datenbank → Erweiterte Einstellungen → Besondere Einstellungen → Die Rechte vom Datenbanktreiber ignorieren** abgewählt bzw. auf 'Nein' gesetzt wird.

Am besten ist es, die erforderlichen Daten durch Abfragen zusammen zu fassen und eine Datenänderung innerhalb von Thunderbird vor zu nehmen. Hier eine Abfrage, die Namen, Mailadressen und Listenzugehörigkeit anzeigt:

```
001 SELECT "a"."card", "a"."value" "DisplayName", ( SELECT "value" FROM
    "properties" WHERE "card" = "a"."card" AND "name" = 'PrimaryEmail' )
    "PrimaryEmail", "lists"."name" "list"
002 FROM "properties" AS "a" LEFT JOIN "list_cards" ON "list_cards"."card" =
    "a"."card" LEFT JOIN "lists" ON "lists"."uid" = "list_cards"."list"
003 WHERE "a"."name" = 'DisplayName'
```

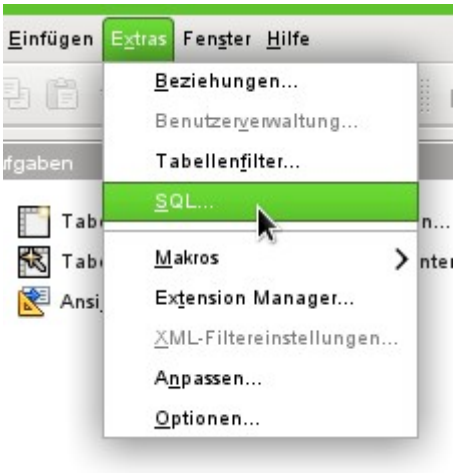
Texttabellen

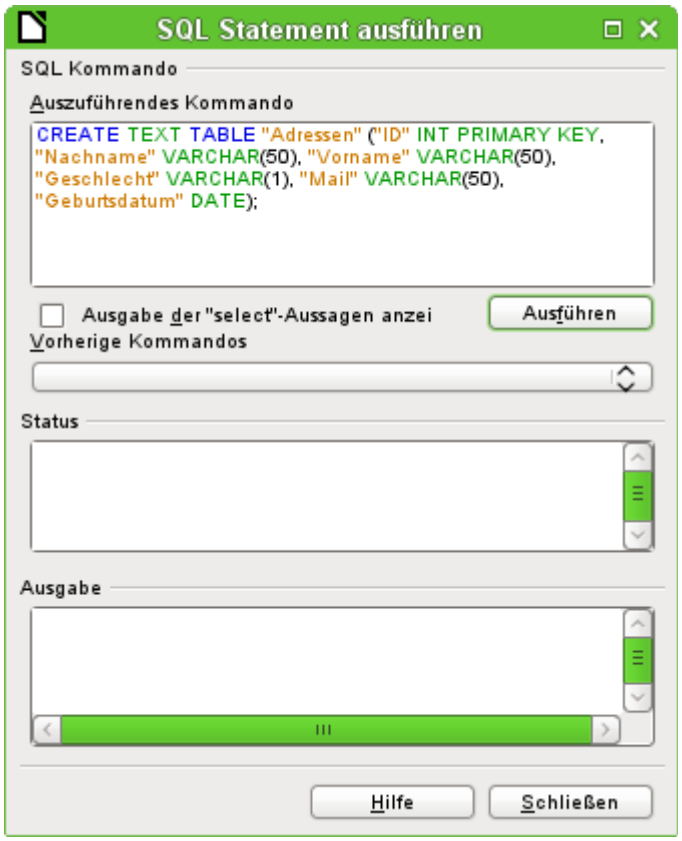
In Base gibt es die Möglichkeit, eine komplette Datenbank mit Zugriff auf Texttabellen zu erstellen. Gleichzeitig können auch Texttabellen innerhalb einer internen HSQLDB-Datenbank angesprochen werden.

Texttabellen innerhalb einer internen HSQLDB-Datenbank

Ein gängiges Austauschformat zwischen Datenbanken ist das *.csv-Format[4]. Daten werden hier in einer durch einfache Textbearbeitungsprogramme les- und schreibbaren Form gespeichert. Einzelne Felder werden durch Kommata getrennt. Enthält ein Feld Text mit einem Komma, so werden die Textfelder mit doppelten Anführungszeichen versehen. Ein neuer Datensatz beginnt nach einem Zeilenumbruch.

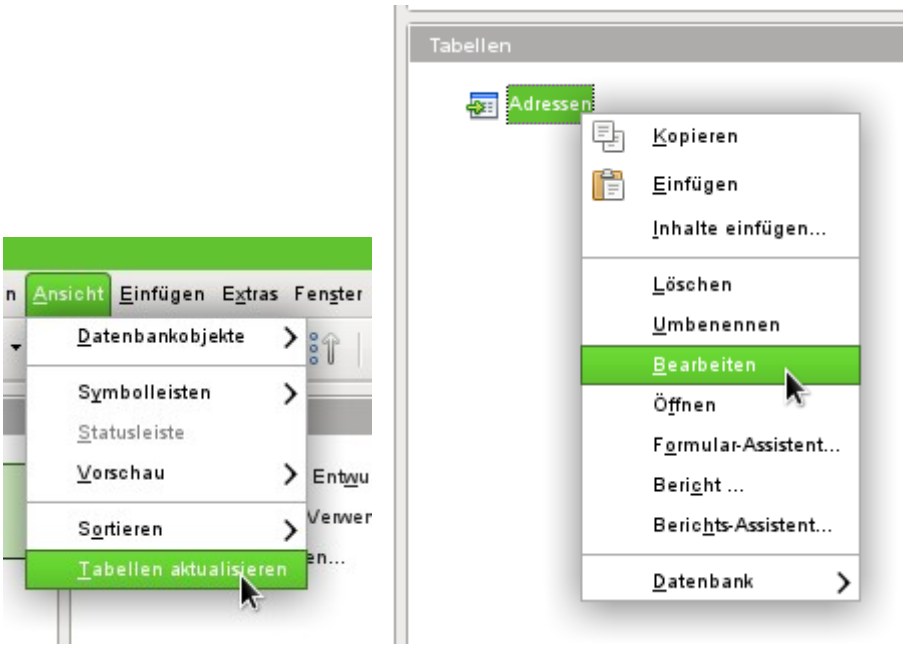
Soll zum Beispiel der Inhalt eines Adressbuches, das nicht direkt von den Treibern von Base unterstützt wird, eingelesen werden, so kann dies entweder über einen Import einer solchen *.csv-Datei erfolgen (hierzu ist gegebenenfalls ein Zwischenschritt über Calc notwendig) oder die Datei wird direkt als Texttabelle in die Datenbank eingefügt. Um dort bearbeitet werden zu können, benötigt die *.csv-Datei allerdings ein Feld, das als eindeutiges Feld («Primärschlüssel») erkennbar ist.



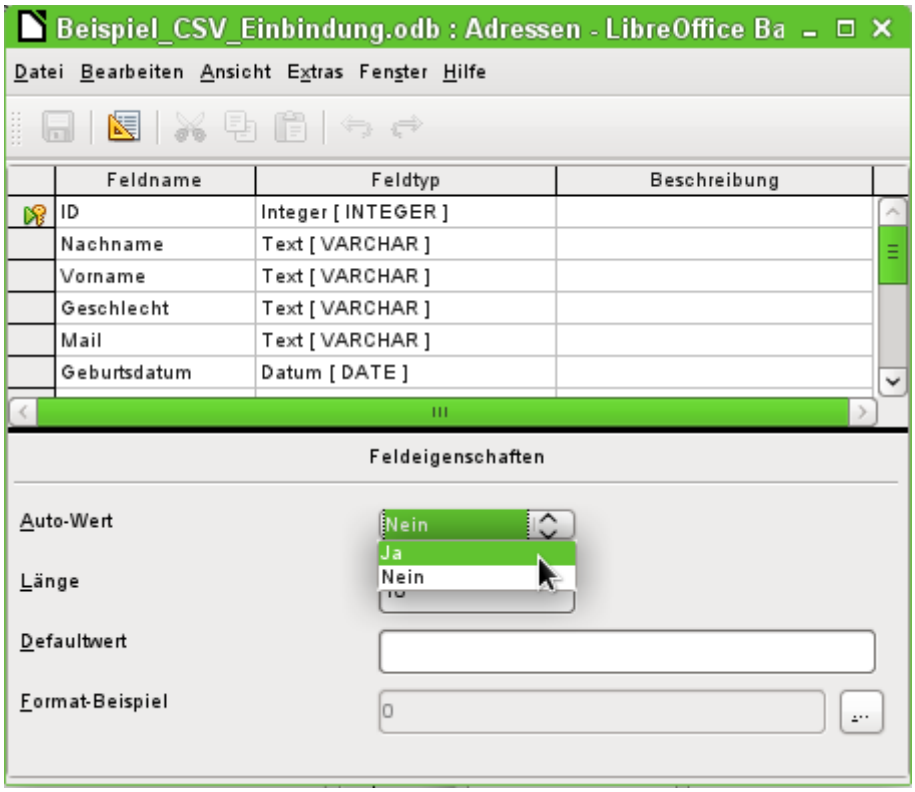


4 CSV-Dateien lassen sich aus jeder Tabelle oder Abfrage durch Export über Calc erzeugen. Die für OpenOffice 2 geschriebene Extension exportcsv.oxt (*https://extensions.openoffice.org/node/18410*) erzeugt direkt aus Base heraus *.csv-Dateien. Die Extension funktioniert zusammen mit LibreOffice 7 einwandfrei.

Der Zugriff zu einer Texttabelle kann nicht über die grafische Benutzeroberfläche erstellt werden.[5] Stattdessen wird über **Extras → SQL** eine Texttabelle erstellt. Die Felder in der Texttabelle müssen dem entsprechen, was die Texttabelle in der entsprechenden Reihenfolge liefert. Das Feld "ID" muss z. B. positive Ganzzahlen enthalten, das Feld "Geburtsdatum" z. B. ein Datum in der Schreibweise Jahr – Monat – Tag.



Die Tabelle ist nicht direkt auf der Benutzeroberfläche sichtbar. Falls jetzt noch eine Ergänzung erfolgen soll, muss über **Ansicht → Tabellen aktualisieren** die Tabelle verfügbar gemacht werden. Das Symbol der Tabelle macht deutlich, dass es sich nicht um eine «normale» Tabelle der Datenbank handelt.



5 Diesem Handbuch liegt die Datenbank «Beispiel_CSV_Einbindung.odb» bei.

Die Tabelle wurde zum Bearbeiten geöffnet und das Primärschlüsselfeld auf einen automatisch hoch zählendes Feld umgestellt. Dies kann auch direkt mit der Eingabe des SQL-Befehls erfolgen:

```
001 CREATE TEXT TABLE "Adressen" ("ID" INT GENERATED BY DEFAULT AS
    IDENTITY, ...
```

Über **Extras → SQL** muss jetzt noch die Verbindung zur externen Texttabelle erstellt werden. Diese Texttabelle liegt im gleichen Verzeichnis wie die Datenbankdatei selbst.

```
001 SET TABLE "Adressen" SOURCE "Adressen.csv;encoding=UTF-8";
```
[6]

Anschließend steht die Texttabelle ganz normal zur Eingabe zur Verfügung. Allerdings ist hierbei Vorsicht geboten:

- Die Texttabelle kann gleichzeitig auch von externen Textprogrammen oder Tabellenkalkulationen geöffnet und bearbeitet werden. Datenverlust ist dann nicht auszuschließen.
- Änderungen an bereits geschriebenen Datensätzen führen dazu, dass die entsprechende Zeile in der Originaldatei geleert wird und in der neuen Fassung an den Schluss der Tabelle angefügt wird. Die obige Ansicht präsentiert z.B. sauber 4 beschriftete Zeilen mit richtig sortierter ID. Die Originaldatei zeigt hingegen nach einer Änderung im 2. Datensatz die folgende Reihenfolge, geordnet nach der ID: 1, Leerzeile, 3, 4, 2

Für die Verbindung zur Textdatei stehen verschiedene Parameter zur Verfügung.

```
001 SET TABLE "Adressen" SOURCE
    "Adressen.csv;ignore_first=false;all_quoted=true;encoding=UTF-8";
```

Mit «ignore_first=true» wird die erste Zeile der Datei nicht eingelesen. Dies ist sinnvoll, wenn es sich dabei um die Feldbezeichnungen handelt. Der interne Standardwert der HSQLDB steht hier auf «false».

Standardmäßig werden von der HSQLDB Texte nur in doppelte Anführungszeichen gesetzt, wenn sie ein Komma enthalten, da das Komma die Standardtrennung der Felder ist. Soll jedes Feld in doppelte Anführungszeichen gefasst werden, so ist «all_quoted=true» zu setzen.

Zu weiteren Parametern siehe: *http://hsqldb.org/doc/guide/ch09.html#set_table_source-section*.

Mit

```
001 SET TABLE "Adressen" READONLY TRUE;
```

wird schließlich davor geschützt, dass in die Tabelle überhaupt etwas geschrieben wird. So steht dann die Tabelle als normale Adresstabelle wie eine Tabelle aus Adressbüchern z.B. von Mailprogrammen schreibgeschützt zur Verfügung. Die gesonderte Erstellung eines Schreibschutzes ist allerdings nur notwendig, wenn für die Tabelle erst einmal ein Primärschlüssel erstellt wurde.

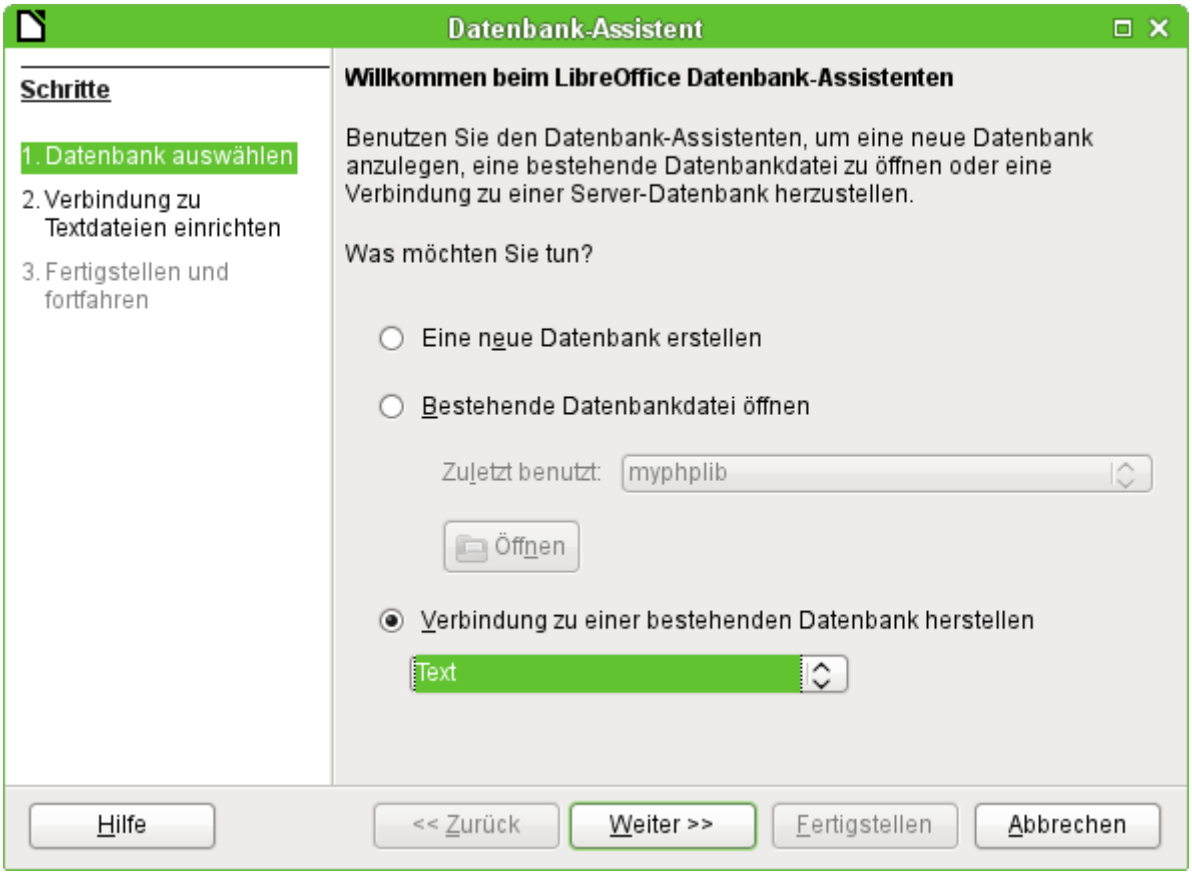
6 Die Angabe «encoding=UTF-8» funktioniert bei vielen Systemen. Eventuell muss auch statt «UTF-8» «ansi» gewählt werden.

Hinweis

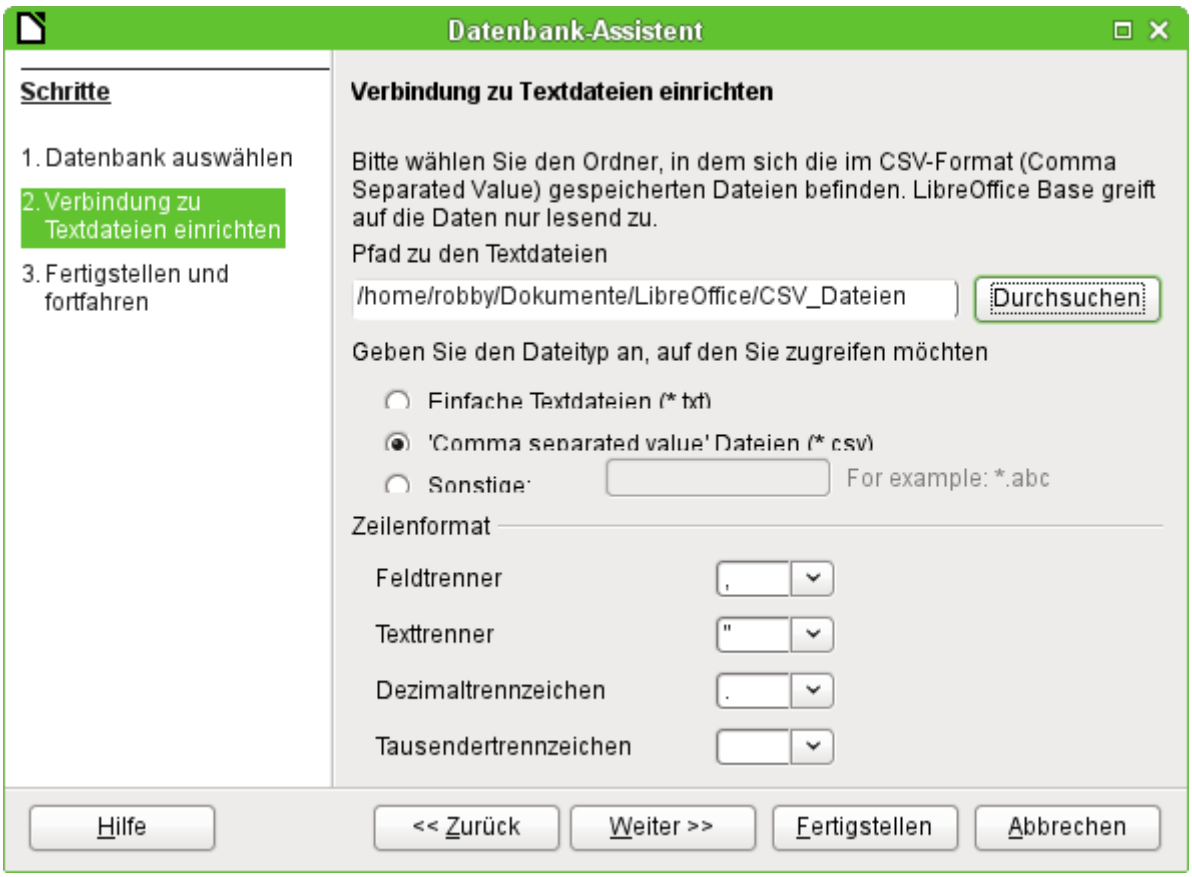
Texttabellen innerhalb der internen HSQLDB scheinen zur Zeit die einzige Möglichkeit zu sein, Zeilenumbrüche aus externen Tabellen wie z.B. einer Calc-Tabelle beim Import in eine HSQLDB zu erhalten:
Die Calc-Tabellen müssen jeweils als einzelne *.csv-Dateien exportiert werden. Zeilenköpfe sind zu entfernen. Für jede *.csv-Datei wird eine Texttabelle erstellt. Mehrzeilige Texte müssen in LONGVARCHAR-Spalten abgelegt sein. Anschließend werden diese Textdateien wieder kopiert und als normale Tabellen in die HSQLDB-Datenbank eingefügt.

Texttabellen als Grundlage für eine eigenständige Datenbankdatei

Wie im vorhergehenden Beispiel werden als Datengrundlage in dem folgenden Beispiel *.csv-Dateien genutzt. Über Base wird ein Verzeichnis, in dem die *.csv-Dateien liegen als Datenverzeichnis eingebunden.

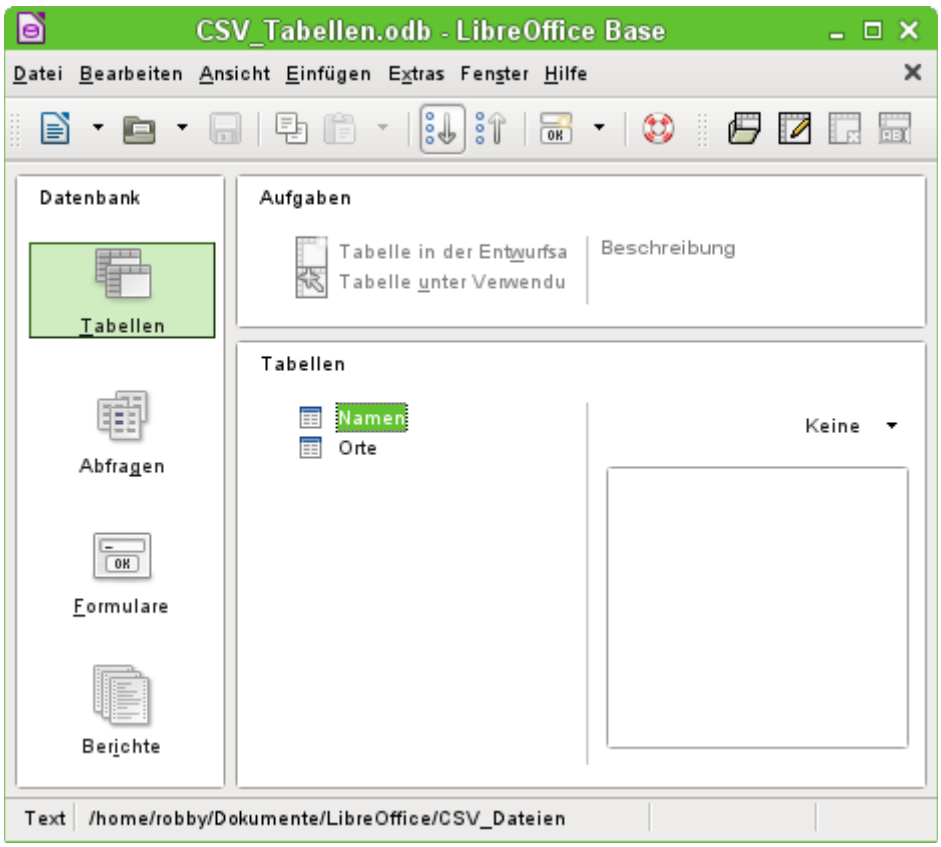


Startpunkt ist die Verbindung zu einer bestehenden Datenbank. Hier ist als Format «Text» gewählt.



Der Pfad zu den Textdateien wird ausgesucht. Innerhalb des Verzeichnisses werden später alle dem Dateityp entsprechenden Dateien aufgelistet. Für das *.csv-Format ist hier die zweite Option gewählt.

Bereits bei der Auswahl ist schon deutlich beschrieben: Auf die Tabellen kann nur lesend zugegriffen werden, nicht schreibend.



In der Tabellenansicht sind dann alle Tabellen mit dem Dateinamen aus dem aufgezeigten Verzeichnis ohne die Dateinamenserweiterung zu sehen. Die Aufgaben zum Erstellen von Tabellen stehen nicht zur Verfügung. Die Tabellen selbst sind nur lesbar, nicht schreibbar.

Auch der Zugriff auf die Tabellen mit Abfragen ist beschränkt auf eine Tabelle und keine Funktionen.

Wird eine solche Datenbank genutzt, um einmal kurz in einer *.csv-Datei nach entsprechenden Datensätzen zu suchen oder um eine *.csv-Datei in eine andere Datenbankdatei über die Kopierfunktion einzufügen, so erfüllt diese Text-Datenbank ihren Sinn. Die entsprechende *.csv-Datei wird nur in das definierte Verzeichnis geschoben und kann dann direkt durchsucht oder kopiert werden. Für weitergehende Datenbankaufgaben ist diese Text-Datenbank wohl nicht gedacht.

Firebird

Die etwas in die Jahre gekommene alte HSQLDB-Version der internen Datenbank wird ab LO 6.1 schrittweise durch eine interne Firebird-Datenbank ersetzt. Seit Version LO 4.2.* ist die interne Firebird-Datenbank erst einmal nur als «experimentelle Funktion» zusätzlich wählbar. Das liegt einfach daran, dass zu dem Zeitpunkt längst noch nicht alle Funktionen richtig integriert werden konnten. Um aber zu sehen, was Firebird leisten kann, wird hier die Verbindung zu einer externen Firebird-Datenbank dargestellt.

Da die Dokumentation dazu nicht ganz so umfangreich ist wie z.B. zu MySQL oder PostgreSQL hier zuerst einmal die wichtigsten Schritte bei der Installation.

Erstellen eines Nutzers und einer Datenbank

Unter Linux stehen für Firebird entsprechende Pakete in der Paketauswahl bereit. Hier muss nach der Installation eigentlich nur noch der Server eingeschaltet werden. Die folgenden Schritte haben unter OpenSUSE 15.0 zu einer funktionierenden Datenbank mit Firebird geführt:

1. Notwendig für die Installationsschritte bis zur funktionierenden Datenbank sind Administratorrechte auf dem Rechner. Dem Systemuser «firebird» sollte ein Passwort zugeordnet werden.
2. Der Nutzer «firebird» muss sich mit
   su firebird
   auf der Konsole anmelden.
3. **sysdba** ist der Nutzername für den Admin-Account. Die folgenden Schritte funktionieren nur, wenn der Firebird Server nicht läuft.
4. Eine Passwortänderung geht im Terminal als Nutzer «firebird» über:
   isql-fb -user sysdba
   Je nach System kann der Befehl auch **isql** statt **isql-fb** lauten. In OpenSUSE ist gibt es andere Pakete, die den Befehl «isql» belegen.
5. Danach geht es weiter in der SQL-Konsole. Zuerst muss eine leere Datenbank erstellt werden, damit auf die weiteren Befehle zugegriffen werden kann:
   SQL> **CREATE DATABASE 'fbtest.fdb';**
   SQL> **CREATE USER SYSDBA password 'neuesPasswort';**
   SQL> **commit;**
6. Erst nach diesen Einstellungen kann der Server gestartet werden.
7. Ein neuer Nutzer sollte erstellt werden:
   SQL> **connect localhost:employee.fdb user SYSDBA password neuespasswort;**
   Als Antwort hierauf erscheint die Meldung
   Database: localhost:employee.fdb, User: SYSDBA
   Hiernach wird dann eingegeben:
   SQL> **CREATE USER lotest password 'libre';**
   Der neue Nutzer mit dem Namen **lotest** und dem Passwort **libre** wird also erst erstellt, wenn eine Verbindung zu einer Datenbank, hier der zu Firebird gehörigen festen Beispieldatenbank «employee.fdb», existiert.
8. Danach kann die Datenbank für den neuen Nutzer erstellt werden:
   SQL> **CREATE DATABASE 'libretest.fdb' user 'lotest' password 'libre';**
9. Auf die direkte Nachfrage nach dem Commit muss mit y geantwortet werden:
   Commit current transaction (y/n)?y
   Committing.
10. Mit
    SQL> **quit;**
    wird die Eingabe im SQL-Modus des Tools **isql-fb** beendet.

Werden die Angaben als Systemadministrator «root» getätigt, so ist die Datenbank nicht dem richtigen Benutzer für den Netzwerkbetrieb zugeordnet. Die Datenbankdatei muss als Nutzer und als Gruppe «firebird» zugewiesen bekommen. Sonst klappt der Kontakt später nicht.

Direkte Verbindung zu einem Firebird Server

Die direkte Verbindung zu einem Firebird Server ist mit dem internen Treiber möglich. Allerdings bietet die GUI hier zur Zeit keine weitere Hilfe an. Auch funktioniert diese Verbindung nur so lange, wie die interne Datenbankversion Firebird 3 und die Version des Servers gleich sind. **Firebird-Dateien für Firebird 4 können mit dem internen Treiber nicht geöffnet werden**.

Es wird wie bei der Verbindung zu einer externen Firebird-Datei vorgegangen. Allerdings lässt sich die Datei nicht über das System über Durchsuchen… ermitteln, da bei einem richtigen Server natürlich das entsprechende Verzeichnis für den Normaluser nicht zugänglich ist. Der Pfad, der einzugeben ist, ist gleich dem lokalen Pfad, der auch in der *odbc.ini* steht:

```
001 localhost:/srv/firebird/libretest.fdb
```

Der Serverpfad selbst muss also bekannt sein. Um die Datenbank von außen zu erreichen muss natürlich «localhost» durch die IP des Servers ersetzt werden.

Die Benutzerverwaltung kann hier allerdings nicht über die Verbindung erfolgen, da dieses Element bei dem internen Treiber nicht implementiert ist. Ein einmal erstellter Benutzer mit entsprechenden Rechten kann allerdings mit diesen Rechten genutzt werden.

Firebird-Verbindung über JDBC

Zuerst muss das Jar-Archiv in LO eingebunden werden. Irritierend ist, dass es kein Archiv mit einer Bezeichnung firebird-*.jar gibt. Der aktuelle JDBC-Treiber ist auf der Seite *http://www.firebirdsql.org/en/jdbc-driver/* zu finden. Der Treiber beginnt mit der Bezeichnung Jaybird...

Aus der *.zip-Datei muss nur das Archiv **jaybird-full-4.0.5.jar** (oder aktuellere Version) kopiert und entweder in den Java-Pfad der Installation gelegt oder als Archiv direkt in LO eingebunden werden. Siehe dazu auch die entsprechende Einbindung bei MySQL.

Für die Angaben bei der JDBC-Installation sind folgende Parameter wichtig, die einen besonderen Kompatibilitätsmodus für ApacheOpenOffice und LibreOffice berücksichtigen. Ohne den Zusatz :**oo** kann es vorkommen, dass einige Tabellen keinen Inhalt zeigen, dieser aber über Abfragen sehr wohl sichtbar wird.

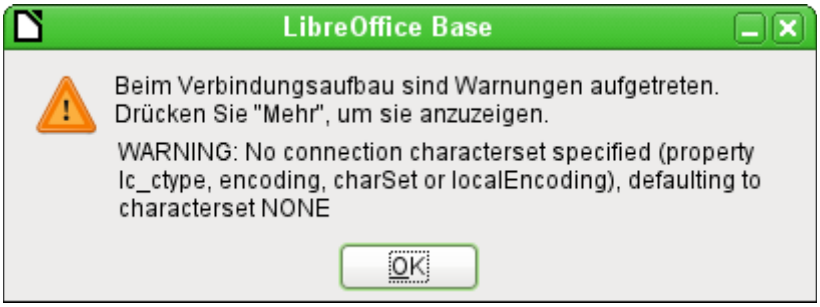
JDBC URL: **jdbc:firebirdsql:oo://host[:port]/<path_or_alias>**

Treiberbezeichnung: **org.firebirdsql.jdbc.FBDriver**

In obigen Beispiel wurde dies zu der URL

```
001 jdbc:firebirdsql:oo://localhost/libretest.fdb?charSet=UTF-8
```

Ohne Zeichensatzangabe erscheint die folgende Fehlermeldung:



Bei der Erstellung von Tabellen muss darauf geachtet werden, dass die Formatierung der entsprechenden Felder («Feldeigenschaften») von Beginn an stimmt. Bei allen Zahlenwerten setzt sonst LO als Default-Format merkwürdigerweise ein Währungsfeld ein.

Nachträgliche Änderungen z.B. der Feldeigenschaften von Feldern einer Tabelle sind nicht möglich, wohl aber die Erweiterung der Tabelle oder das Löschen der Felder.

Firebird-Verbindung über ODBC

Aus dem Netz muss zuerst der passende ODBC-Treiber heruntergeladen werden: *http://www.firebirdsql.org/en/odbc-driver/*. Dieser Treiber besteht meist aus einer einfachen Datei, der libOdbcFb.so. Diese Datei wird meist mit Administratorrechten in einen vom System allgemein zugänglichen Pfad gelegt. Sie muss ausführbar sein.

Bei Linux-Systemen kann es wegen fehlender Bibliotheken zu einigen Problemen führen. Siehe hierzu auch *http://www.firebirdsql.org/manual/de/qsg2-de-installing.html* Die lassen sich aber durch entsprechende Symlinks beheben.

Die libOdbcFb.so benötigt die libodbc.so.1. Hier existiert in den meisten Systemen inzwischen die libodbc.so.2 . Auf diese Datei muss ein entsprechender Verweis mit dem Namen libodbc.so.1 in dem gleichen Verzeichnis abgespeichert werden.

Auch die libgds.so wird anschließend als fehlend angemahnt. Dies kann durch einen Link zu /usr/lib64/libfbclient.so.2 gelöst werden.

Weitere benötigte Dateien lassen sich mit

```
001 readelf -d libOdbcFb.so
```

über die Konsole von Linux ermitteln.

In den für das System notwendigen Dateien odbcinst.ini und odbc.ini müssen Einträge ähnlich den folgenden existieren:

odbcinst.ini

```
001 [Firebird]
002 Description = Firebird ODBC driver
003 Driver64 = /usr/lib64/libOdbcFb.so
```

odbc.ini

```
001 [Firebird-libretest]
002 Description = Firebird database libreoffice test
003 Driver = Firebird
004 Dbname = localhost:/srv/firebird/libretest.fdb
005 SensitiveIdentifier = Yes
```

Diese beiden Dateien liegen bei einem Linux-System im Verzeichnis /etc/unixODBC. Die Variable SensitiveIdentifier muss auf jeden Fall auf «Yes» stehen, damit die Eingabe in Tabellen funktioniert, deren Namen und Feldbezeichnungen nicht in Großbuchstaben angegeben sind.

Zur Erstellung der Verbindung von LO mit der entsprechenden Datenquelle siehe *Die ODBC-Verbindung.*

Die ODBC-Verbindung ist leider zur Zeit unter 64bit-Linux-Systemen nicht funktionstüchtig, scheint aber unter Windows zu funktionieren. Das Öffnen der Datenbank klappt zwar, aber bereits beim Aufrufen einer Tabelle stürzt LibreOffice sofort ab. Dies liegt vermutlich daran, dass der ODBC-Connector von Firebird zuletzt im Mai 2017 aktualisiert wurde, viele der Bibliotheken, auf die sich der Treiber bezieht, aber schon lange überholt sind. Da funktioniert der JDBC-Treiber und die interne Verbindung von LibreOffice eindeutig besser.

Direkte Verbindung zu einer Firebird-Datei

Wird eine Serverdatenbank nicht benötigt, dann ist auch die direkte Verbindung zu einer Firebird-Datei möglich. Hier muss lediglich, wie z.B. bei Tabellendokumenten, der Pfad zu der Firebird-Datei herausgesucht werden. Die Verbindung wird dann automatisch hergestellt. Die Firebird-Datei muss dazu allerdings ab LibreOffice 5.3 mit einer Firebird-Version ab Firebird 3 erstellt worden sein. Über den Assistenten kann so eine Datei direkt erstellt werden.

Aus einer internen Datenbankdatei lässt sich eine externe Datenbankdatei erstellen:

- Das temporäre Verzeichnis, das in LibreOffice definiert ist, muss mit dem Dateimanager aufgesucht werden. Dort erscheint automatisch ein neues Unterverzeichnis, wenn LibreOffice gestartet wird. Dieses Unterverzeichnis ist nicht an dem Namen, sondern an dem Zeitpunkt der Erstellung zu erkennen.
- Die Datenbank mit der internen Firebird-Datenbankdatei muss gestartet werden.[7]
- Mit einem Klick auf **Datenbank → Tabellen** muss der Tabellencontainer geöffnet werden, damit die Verbindung zur internen Datenbankdatei erstellt wird.
- Jetzt werden in dem neu erstellten temporären Unterverzeichnis mehrere Unterverzeichnisse erstellt.
- In einem dieser Unterverzeichnisse steckt neben «firebird.fbk» die daraus erstellte «firebird.fdb». Diese Datei kann jetzt kopiert und als externe Datenbankdatei weiter verwandt werden. Ein Passwortschutz für diese Datei besteht nicht.
- Nach dem Schließen der internen Datenbankdatei wird das temporäre Verzeichnis automatisch gelöscht.

7 In der *.odb-Datei liegt zur Zeit die Datei «firebird.fbk», eine Backupdatei der Daten.

Sind bereits Formulare, Abfragen, Berichte und Makros erstellt worden, so wäre es schön, die alte Datenbank anschließend mit der neuen Verbindung zu nutzen. In der GUI lässt sich das leider nicht umstellen. Hierfür muss die Datenbankdatei mit einem Packprogramm geöffnet werden und in der Datei **content.xml** die folgende Änderung in dem Tag **db:connection-data** vorgenommen werden:

```
001 <db:connection-data><db:connection-resource xlink:href="sdbc:embedded:firebird"
    xlink:type="simple"/><db:login db:is-password-required="false"/></db:connection-
    data>
```

wird zu

```
001 <db:connection-data><db:connection-resource
    xlink:href="sdbc:firebird:file:///home/robby/Dokumente/Datenbanken/Firebird/
    firebird.fdb" xlink:type="simple"/><db:login
    db:is-password-required="false"/></db:connection-data>
```

Statt der Referenz auf die interne Firebird Datenbank wird hier die Referenz auf eine externe Datei gesetzt. Sollte bei der Pfadeingabe ein Fehler passieren, so kann auch unter **Bearbeiten → Datenbank → Eigenschaften** gegebenenfalls ein anderer Pfad zu der Datenbank angegeben werden.

Ohne Packprogramm ist es am einfachsten, über **Datei → Neu → Datenbank → Verbindung zu einer bestehenden Datenbank herstellen → Firebird (extern)** eine neue Datenbank mit der Verknüpfung zur externen firebird.fdb zu erstellen. Formulare, Berichte, Abfragen und Makros müssen dann von der alten zur neuen Datenbank kopiert werden.

Von der externen Firebird-Datei zur Serverdatenbank

Die externe Datenbankdatei kennt noch keine Benutzerverwaltung. Alle Tabellen, Ansichten und Trigger sind dort unter dem Administrationsnutzer von Firebird (SYSDBA) erstellt worden, können aber durch den aktuellen Benutzer der Datenbankdatei genauso weiter benutzt werden.

Die Schritte wie bei der Erstellung einer neuen Datenbank sind nicht notwendig. Die folgenden Schritte haben unter OpenSUSE 15.3 zu einer funktionierenden Serverdatenbank mit Firebird 3 geführt:

1. Notwendig für die Installationsschritte bis zur funktionierenden Datenbank sind Administratorrechte auf dem Rechner. Der Firebird Server muss ausgeschaltet sein.
2. Die externe Firebird-Datei «extern.fdb» wird in das Verzeichnis kopiert, das von dem Firebird Server als Serververzeichnis angesehen wird. Hier ist das **/srv/firebird**.
3. Der Datei wird der Nutzer und die Gruppe «firebird» zugewiesen.
4. Dem Systemuser «firebird» sollte ein Passwort zugeordnet werden.
5. Der Nutzer «firebird» muss sich mit
   su firebird
   auf der Konsole anmelden.
6. Auf der Konsole wird
   isql-fb -user SYSDBA extern.fdb
   eingegeben. Dadurch wird der Nutzer SYSDBA mit der Datenbank verbunden.
7. Anschließend wird mit
   SQL> **CREATE OR ALTER USER SYSDBA PASSWORD 'neuesPasswort';**
   ein Passwort erstellt, das für den Systemnutzer, der auch Standardnutzer aller internen Datenbanken von Base ist, genutzt werden kann.
8. Mit
   SQL> **quit;**
   wird die Eingabe im SQL-Modus des Tools **isql-fb** beendet.
9. Jetzt wird der Firebird Server gestartet.
10. Die JDBC-Verbindung zur Datenbank extern.fdb wird mit dem Nutzer SYSDBA und dem entsprechenden Passwort erstellt.

11. Über **Extras → SQL** wird mit
    `CREATE USER neuerUser PASSWORD 'neuesUserPasswort';`
    ein neuer Nutzer erstellt.
12. Verbindet sich der neue Nutzer mit der Serverdatenbank, so kann er zwar die Tabellen sehen, bei einem Klick auf die Tabellen erscheint aber, dass er keine Rechte zum Lesen der Tabellen hat. Dies liegt daran, dass die (vorher interne) Datenbankdatei nur den Nutzer SYSDBA kennt. Daher muss der Nutzer SYSDBA dem neuen Nutzer alle möglichen Rechte zuweisen:
    Über **Extras → SQL** werden die Rechte für <neuerUser> eingestellt. Die allgemeine Administrationsrolle zu der aktuellen Datenbank wird über
    `GRANT RDB$ADMIN TO neuerUser;`
    zugewiesen.
13. Nach dem Einloggen des neuen Nutzers muss dieser explizit angeben, dass er die Rolle gerade wahrnehmen will. Über **Extras → SQL** (oder per Makro) macht dies
    `SET ROLE RDB$ADMIN;`
    Dies ist eine Einstellung, die bei jedem Login erfolgen muss.
    Alternativ dazu kann auch zu den entsprechenden Tabellen und Ansichten ein entsprechendes Nutzerrecht vergeben werden:
    `GRANT ALL ON TABLE "Tabellenname" TO USER neuerUser;`
    Diese Zuweisung ist von der **`ROLE`** unabhängig. Sie muss deshalb nicht bei jedem Start extra ausgewählt werden.

Privilegien können auch für einzelne Tabellen detailliert erstellt werden. Für die genaueren Einstellungen sei hier auf *https://www.firebirdsql.org/en/reference-manuals/* mit der entsprechenden Dokumentation verwiesen. Für Firebird 3.0, der intern benutzten Version, existiert dort auch eine deutschsprachige «Sprachreferenz».

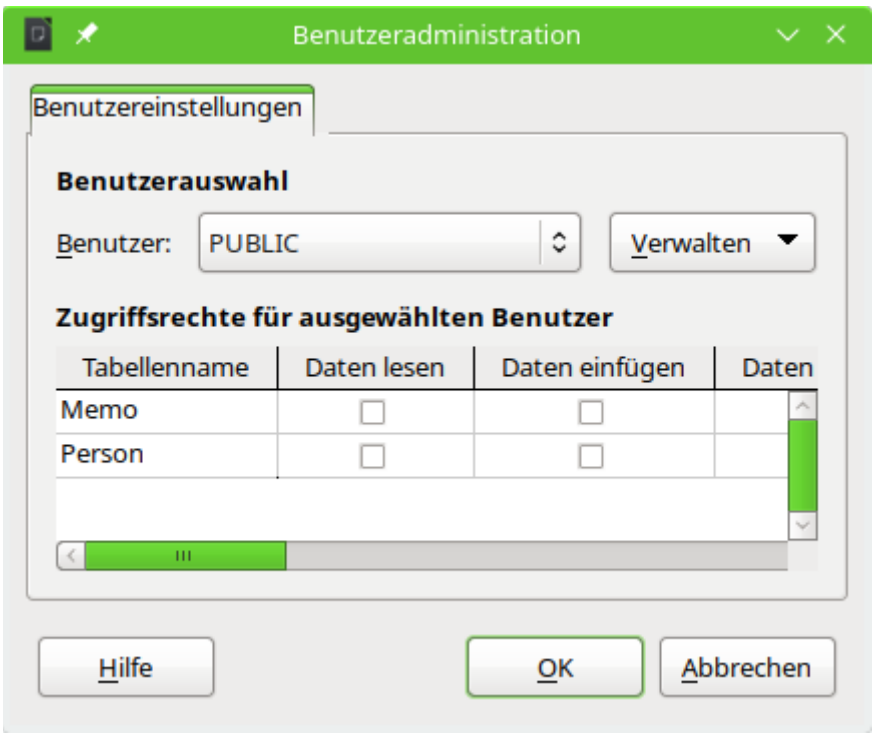
Hinweis

Der **`AutoWert`** für Primärschlüssel in Tabellen wird in der aktuellen Kombination von LO 7.3.0.3 und JDBC-Treiber nicht als **`AutoWert`** in der neuen Zeile angezeigt. Bei einer Neueingabe erscheint dort grundsätzlich '0' statt des nächsten erstellten Wertes. Erst wenn die Tabelle aktualisiert wird ist zu sehen, dass der Wert allerdings richtig in die Datenbank übernommen wird. Hier fehlt also neben der Anzeige **`<AutoWert>`** auch die Aktualisierung.

Hinweis

Firebird Datenbanken werden standardmäßig nicht komprimiert. Werden also Daten in so einer Datenbank gelöscht, so wird die Datenbankdatei vom Speicherbedarf nicht kleiner. Die freiwerdenden Lücken werden dann mit neuen Daten aufgefüllt.
Eine solche Datenbank kann nur über umständliche Backup- und Restorebearbeitung wieder weniger Platz auf dem Festspeicher einnehmen: *https://www.firebirdfaq.org/faq41/*

Benutzerverwaltung bei der externen Firebird Datenbank



Unter Firebird und einigen anderen externen Datenbanken, soweit der Treiber das zulässt, steht über **Extras → Benutzerverwaltung…** ein Dialog zur Verwaltung der Benutzerrechte für die verschiedenen Tabellen zur Verfügung. Leider scheint diese Benutzerverwaltung grundsätzlich nicht zu funktionieren. Zumindest das Hinzufügen von neuen Nutzern über **Verwalten** wäre schon sinnvoll. So könnten auch die Passwörter entsprechend in der GUI gesetzt werden. Bei den Zugriffsrechten wird auch grundsätzlich außer den Tabellennamen nicht angezeigt. Die Markierfelder reagieren hier überhaupt nicht.

Für die direkten Verbindungen zu MySQL/MariaDB oder auch PostgreSQL ist die Benutzerverwaltung von vornherein ausgeschlossen und startet gar nicht erst.

SQLite

SQLite-Datenbanken bestehen aus einer einzigen Datei. Diese Datei kann irgendwo im Dateisystem gespeichert werden und muss nur über einen entsprechenden Treiber mit Base verbunden werden.

Erstellen einer Datenbank

Das Erstellen einer Datenbank beschränkt sich darauf, eine leere Datei in einem Verzeichnis abzulegen. Es ist nicht einmal erforderlich, der leeren Datei eine bestimmte Dateiendung zu geben. Unter Linux reicht es, eine einfache Textdatei zu erstellen. Die Datei wird dann mit dem Namen libretest.db versehen. Bei der Erstellung kann es passieren, dass bereits Absätze vorhanden sind. Diese Absätze sollten gelöscht werden, so dass der Inhalt der Datei 0 Byte groß ist.

Natürlich lässt sich eine leere Datei auch auf der Konsole erstellen, die bei der Installation des Treibers mitinstalliert wird:

```
001 sqlite3 libretest.db
```

erstellt die leere Datenbankdatei. Dabei wird dann die SQLite-Konsole geöffnet. Die Datei ist noch nicht in dem Verzeichnis sichtbar, in dem der Befehl ausgeführt wird. Auf der Konsole wird jetzt nur noch

```
001 .exit
```
eingegeben und die leere Datei wird geschrieben.

SQLite-Verbindung über ODBC

Unter Linux wird der ODBC-Treiber des Systems über die Paketverwaltung installiert. Die Einstellungen in der odbcinst.ini werden dabei automatisch vorgenommen.

odbcinst.ini

```
001 [SQLITE3]
002 Description=SQLite ODBC 3.X
003 Driver=/usr/lib64/libsqlite3odbc.so
004 Setup=/usr/lib64/libsqlite3odbc.so
005 Threading=2
006 FileUsage=1
007 UsageCount=1
```

In der odbc.ini wird dann die Verbindung zu der vorher erstellten Datei vorgenommen. Die Datei wird anschließend über die entsprechende Bezeichnung, hier Sqlite3-libretest, über ODBC mit Base verbunden.

odbc.ini

```
001 [Sqlite3-libretest]
002 Description = SQLite database libreoffice test
003 Driver= SQLITE3
004 Database= /home/<nutzername>/Dokumente/LibreOffice/libretest.db
```

SQLite-Verbindung über JDBC

Nachdem die *.jar-Archiv herunter geladen und in den Class Path eingebunden wurde wird mit folgenden Angaben auf die Datenbank zugegriffen:

JDBC URL: **jdbc:sqlite:/home/<nutzername>…/libretest.db**

Treiberbezeichnung: **org.sqlite.JDBC**

Unter Windows kommt hier noch entsprechend die entsprechende Partition hinzu: **jdbc:sqlite:*C:/*…/libretest.db**.

Tabellen lassen sich direkt erstellen, aber bei dem ersten Zugriff auf eine Tabelle kommt die Meldung:

Die Tabellen lassen sich so nicht öffnen. Hier hilft eine Einstellung in **Bearbeiten → Datenbank → Einstellungen → Erweiterte Einstellungen**. Unter **Besondere Einstellungen** muss **Satztyp des Datenbanktreibers akzeptieren** angewählt werden.

Zugriff auf Access

Während unter Windows der direkte Zugriff auf Access-Datenbanken möglich ist muss unter Linux ein entsprechender Treiber die Verbindung herstellen. Hier bietet sich die JDBC-Verbindung über UcanAccess an: *http://ucanaccess.sourceforge.net/site.html.*

Der JDBC-Treiber kann wie unter «*MySQL/MariaDB-Verbindung über JDBC*» beschrieben eingebunden und genutzt werden. Die *.zip-Datei muss entpackt werden. Das entstehende Verzeichnis wird komplett mit den Unterverzeichnissen benötigt. Im Unterverzeichnis loader befindet sich ucanload.jar. Diese Datei muss über den Class-Path direkt eingebunden werden.

Für die Angaben bei der JDBC-Installation sind folgende Parameter wichtig:

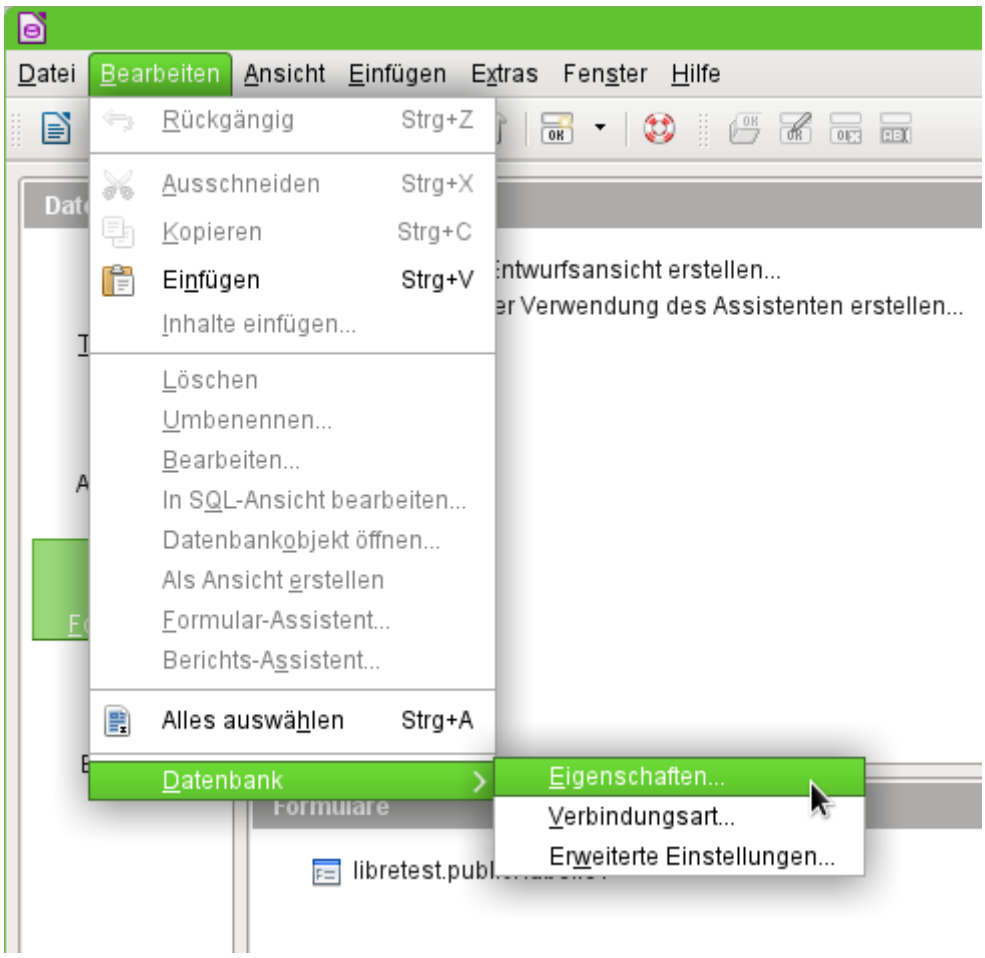
JDBC URL: **`jdbc:ucanaccess:///home/<path_or_alias>`**

Treiberbezeichnung: **`net.ucanaccess.jdbc.UcanloadDriver`**

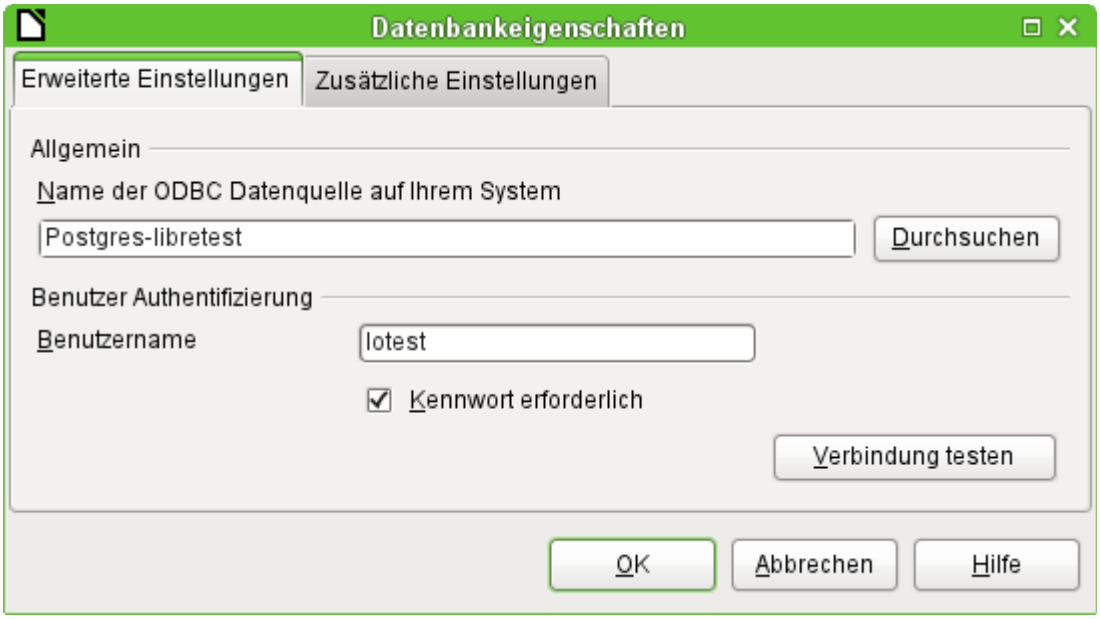
Nachträgliche Bearbeitung der Verbindungseigenschaften

Vor allem bei Verbindungen zu externen Datenbanken kann es hin und wieder vorkommen, dass eine Verbindung nicht wie gewünscht einwandfrei funktioniert. Mal ist der Zeichensatz nicht korrekt, ein anderes Mal funktionieren Unterformulare nicht einwandfrei oder es ist einfach etwas an den grundsätzlichen Verbindungsparametern zur Datenbank geändert worden.

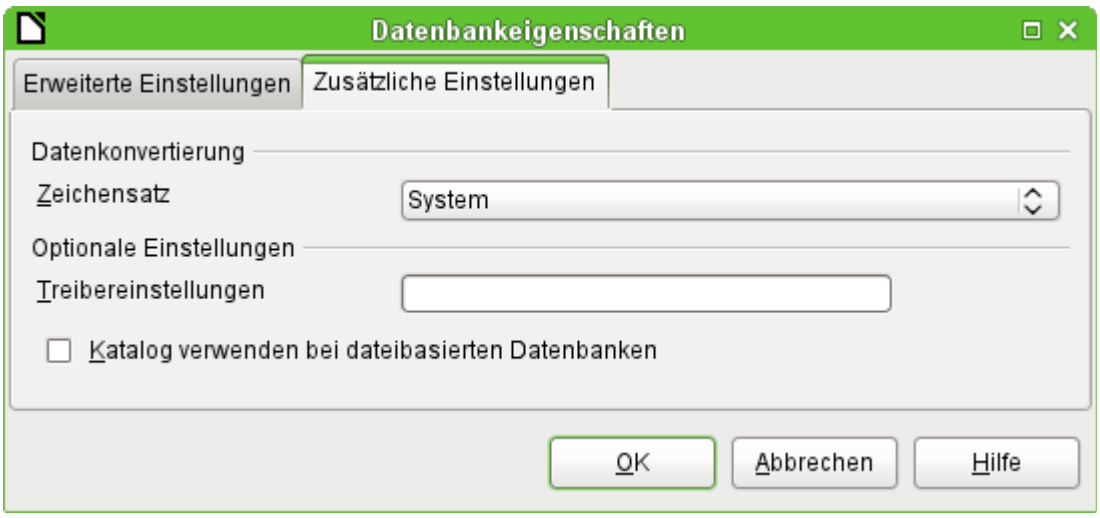
Die folgenden Screenshots zeigen die Möglichkeit auf, wie z.B. die Verbindungsparameter zu einer externen PostgreSQL-Datenbank im Nachhinein geändert werden können.



Über **Bearbeiten → Datenbank** werden die Einstellungsmöglichkeiten «Eigenschaften», «Verbindungsart» und «Erweiterte Einstellung» aufgerufen.

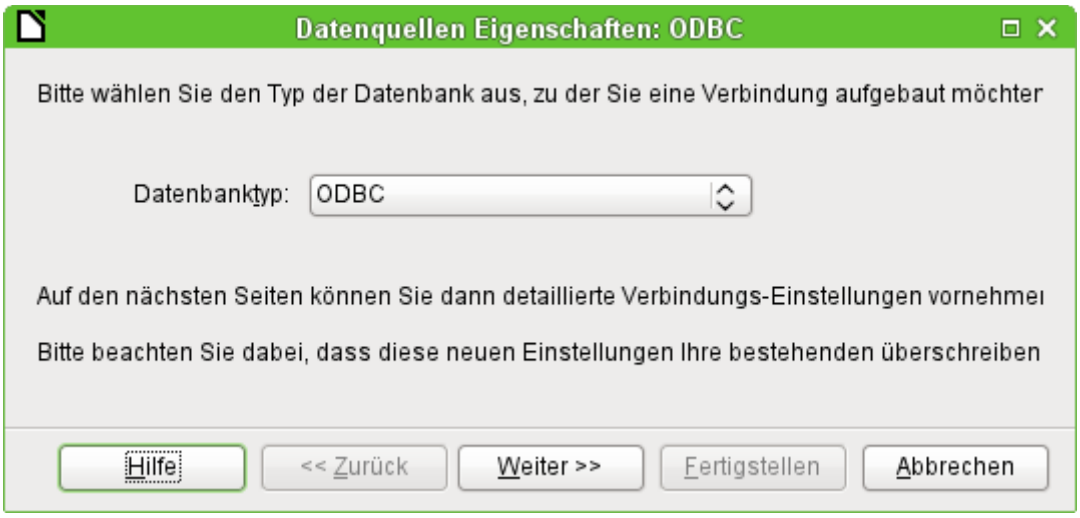


Falls sich der Name der Datenquelle geändert hat, kann dies hier geändert werde. Bei einer Verbindung über ODBC wird ja der Name, mit dem die Datenbank aufgerufen werden kann, in der odbc.ini festgelegt. Der Name ist in der Regel nicht gleich dem eigentlichen Datenbanknamen in PostgreSQL.

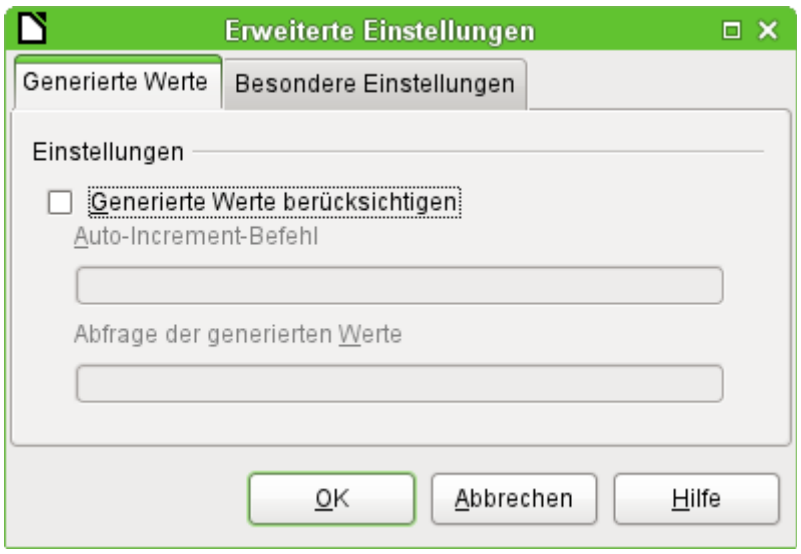


Gibt es Probleme mit dem Zeichensatz? Diese Probleme können eventuell in dem zweiten Reiter der Datenbankeigenschaften gelöst werden.

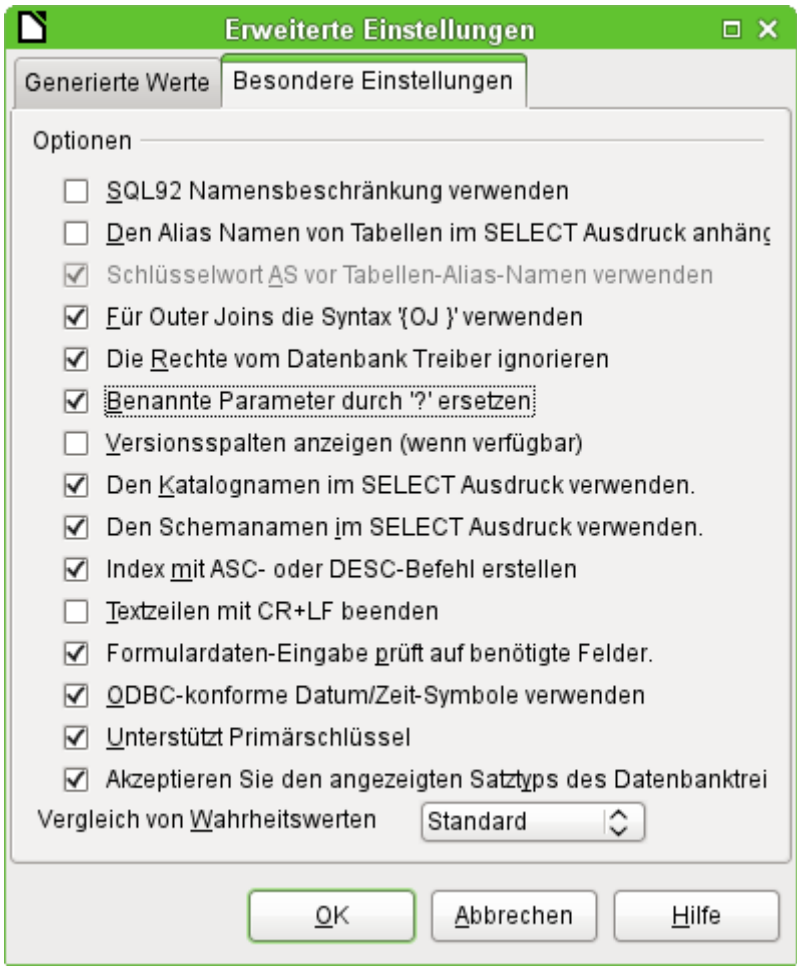
Auch eine zusätzliche spezielle Einstellung des Treibers ist möglich, wenn eventuell ein Parameter ausgeführt werden soll, der zur Zeit nicht bereits in der odbc.ini steht.



Wird die Verbindungsart ausgewählt, so kann der gesamte Kontakt zur Datenquelle geändert werden. Die darauf folgenden Schritte sind gleich denen des Datenbankassistenten ab Schritt 2. So kann z.B. von einer ODBC- auf eine JDBC- oder gar eine direkte Verbindung über die internen Treiber von LO gewechselt werden. Dies ist vor allem dann sinnvoll, wenn zuerst einmal ausgetestet werden soll, welche Verbindungsart zu dem eigenen Projekt am besten passt.



Je nach Datenbanksystem gibt es unterschiedliche Befehle um automatisch hoch zählende Werte zu erzeugen. Soll so etwas erreicht werden, was der Treiber zur Zeit nicht direkt ermöglicht, dann ist an dieser Stelle Handarbeit angesagt. Dabei ist sowohl ein Befehl für das Erstellen eines Auto-Wert-Feldes als auch ein Befehl zur Abfrage des zuletzt erstellten Wertes notwendig.



Die über **Bearbeiten → Datenbank → Erweitere Einstellungen** erreichbaren besonderen Einstellungen beeinflussen das Zusammenspiel von externer Datenbank und Base auf die unterschiedliche Art und Weise. Manche Einstellungen sind bereits ausgegraut, da sie für die zugrundeliegende Datenbank nicht änderbar sind. In dem obigen Beispiel wurde **Benannte Parameter**

durch '?' ersetzen ausgewählt. Es hatte sich gezeigt, dass sonst bei PostgreSQL die Weitergabe von Werten von einem Hauptformular zum Unterformular nicht funktionierte. Erst mit dieser Einstellung funktionierten die weitergehenden Formularkonstruktionen aus dem Kapitel «Formulare» dieses Handbuches korrekt.

Leider sind manchmal nicht alle möglichen erweiterten Einstellungen tatsächlich in der GUI verfügbar. Gegebenenfalls kann dann der Zugriff auf die in der *.odb-Datei befindlichen content.xml helfen. Hier ein Beispiel, das gerade beim Umstieg von LO 6.0 zu LO 6.1 Probleme bereitete: Unterabfragen in MySQL waren nicht mehr möglich, weil die Weitergabe des verbindenden Wertes (Parameter) unterbunden wurde. Das gleiche Problem hat auch Firebird, wenn die Firebird-Datenbank über den Migrationsassistenten erstellt wurde - zumindest in allen Versionen LO 6.*.

Der nicht funktionierende Code:

```
001 <db:driver-settings db:system-driver-settings="" db:base-dn=""
002 db:parameter-name-substitution="false"/>
```

Wird dieser Code geändert auf

```
001 <db:driver-settings db:system-driver-settings="" db:base-dn=""/>
```

so funktionieren die Unterformulare wieder mit der *.odb-Datei.

Alternativ kann auch das folgende Makro von der migrierten Base-Datei aus gestartet werden:

```
001 SUB FB_Parameter
002    DIM oSettings AS OBJECT
003    oSettings = ThisComponent.DataSource.Settings
004    oSettings.ParameterNameSubstitution = True
005 END SUB
```

Nach einmaligem Start des Makros ist der Eintrag in der Base-Datei komplett verschwunden, sofern die Base-Datei einmal abgespeichert wurde.

Maskierung von Tabellennamen und Feldnamen

Grundsätzlich ist es in SQL möglich, Tabellennamen und Feldnamen ohne Maskierung zu schreiben. Ein Code wie

```
001 SELECT VORNAME, NACHNAME FROM PERSONENTABELLE
```

funktioniert genauso wie

```
001 SELECT "VORNAME", "NACHNAME" FROM "PERSONENTABELLE"
```

Je nach Datenbanksystem setzt Base die Maskierung mit doppelten Anführungszeichen automatisch im Abfrageeditor und auch bei der Eingabe über **Extras → SQL**. Bei den internen Datenbanksystemen **HSQLDB und Firebird** kann es aber schon Probleme geben, wenn Tabellennamen und Feldnamen nicht grundsätzlich groß geschrieben werden. Werden die folgenden Eingaben in dem Abfrageeditor im SQL-Modus eingegeben und dabei die direkte Ausführung von SQL ausgewählt, so führt das bei einer Eingabe zu einer Fehlermeldung:

```
001 SELECT "Vorname", "Nachname" FROM "PersonenTabelle"
```

funktioniert, wenn eben die Felder auch Kleinbuchstaben enthalten. Das kann auch auf Sonderzeichen ausgedehnt werden.

```
001 SELECT Vorname, Nachname FROM PersonenTabelle
```

funktioniert hingegen nicht, weil der SQL-Code direkt an die Datenbank weiter gegeben wird ohne die Bezeichner zu maskieren. Die interne Datenbank sucht dann nach einer Tabelle und Feldern in Großbuchstaben, die aber nicht vorhanden sind.

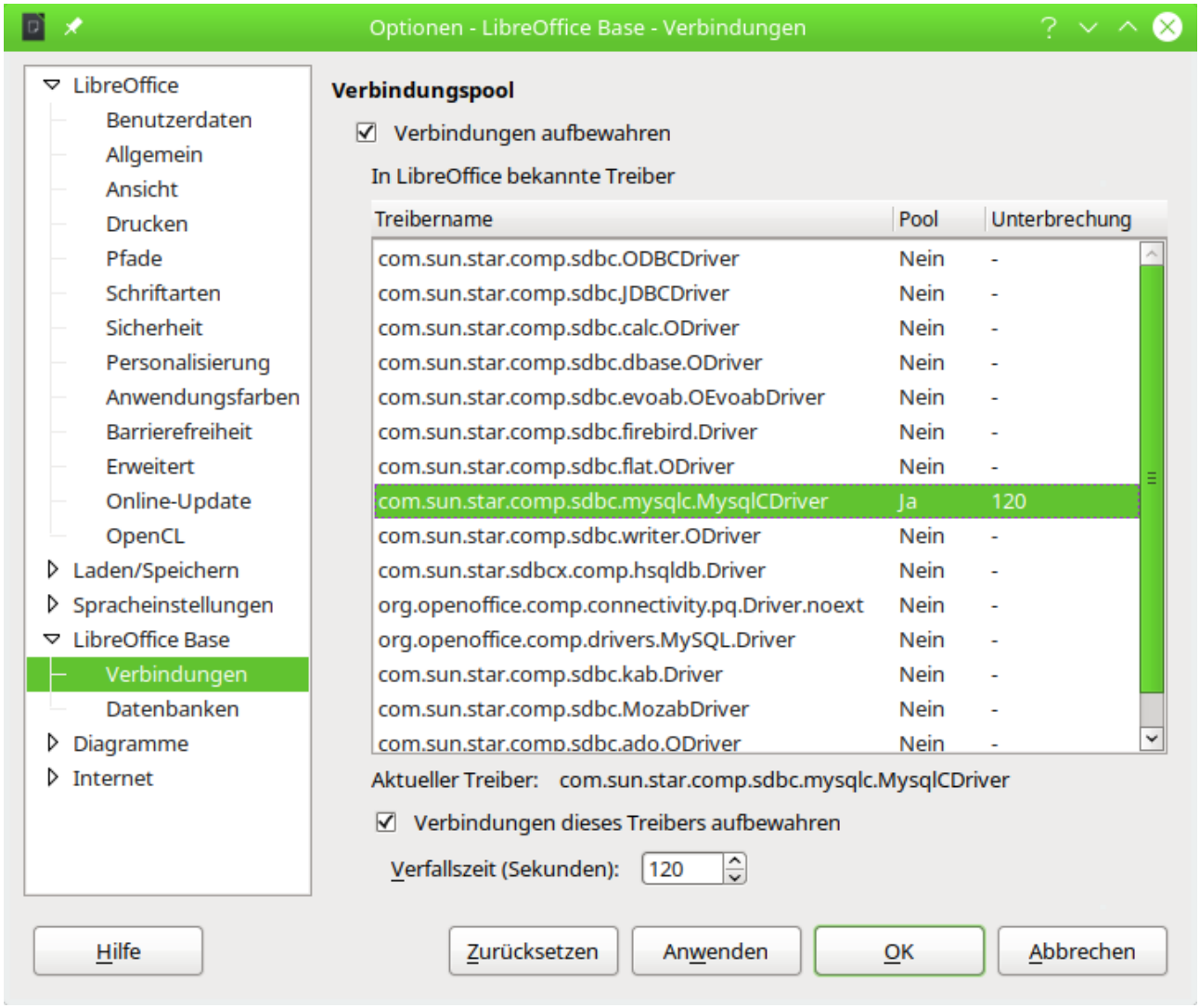
Bei externen Datenbanken hängt die Maskierung von dem entsprechenden Datenbanksystem ab. Für **MySQL/MariaDB** schreibt Base je nach Treiber direkt die passende Maskierung in den SQL-Code, der durch den Abfrageeditor erstellt wird:

```
001 SELECT `Vorname`, `Nachname` FROM `PersonenTabelle`
```

Die Maskierung für MySQL und MariaDB ist also hier nicht das doppelte Anführungszeichen sondern der Gravis `.

Treiberverbindungen aufbewahren

Unter **Extras → Optionen → LibreOffice Base → Verbindungen** gibt es die Möglichkeit, die Aufbewahrungszeit für einen Treiber einzustellen.



Hierbei geht es **nicht** um die Zeit, nach der eine **Verbindung bei fehlender Aktivität**, wie weiter oben beschrieben, nach einer bestimmten Zeit unterbrochen wird. Vielmehr soll eine Verbindung, die ein Nutzer einmal erstellt hat, anschließend auch noch gegebenenfalls für einen anderen Nutzer zur Verfügung gestellt werden. Aus dem Grunde stehen in dem Verbindungspool auch alle Treiber zur Verfügung, die LibreOffice mitliefert.

Die Nutzung dieser Einstellungen scheint nur dann sinnvoll zu sein, wenn über Makros direkt auf einen Datenbank ohne Umweg über die Datenbankdatei zugegriffen werden soll. Dann kann von

```
001 oManager = CreateUnoService("com.sun.star.sdbc.DriverManager")
```

nach

```
001 oManager = CreateUnoService("com.sun.star.sdbc.ConnectionPool")
```

gewechselt werden.

Die Aufbewahrung ist damit unabhängig von den eigentlichen Datenbankeinstellungen z.B. in MySQL/MariaDB oder PostgreSQL, bei denen es um das Halten der Verbindung zur Datenbank geht. Die Zeitabschnitte zum Halten der Verbindung sind standardmäßig bei MySQL/MariaDB 8 Stunden, bei PostgreSQL sogar unbegrenzt.

Datenbankverbindung über SSH

Die Verbindung zu Datenbankservern in Netzen kann durch eine SSH-Verknüpfung zum Server mit einem SSH-Tunnel sicher gestaltet werden. In dem folgenden Beispiel sei die IP des Ubuntu-Servers **192.168.178.32**. Der Kontakt wird hier zu einer PostgreSQL Datenbank aufgenommen.

Auf dem Server existiert ein Benutzer 'lotest' mit dem Passwort 'libre'.

1. Der SSH-Server wird installiert. Dies ist bei Linux-Systemen problemlos über die Paketverwaltung möglich, soll auch ab Windows 10 dort keine weiteren Schwierigkeiten bereiten.
2. Über **ssh localhost** wird local auf dem Server erst einmal der Server getestet und der Schlüssel exportiert
3. Über **ssh lotest@192.168.178.32** wird vom Client über die Konsole Kontakt mit dem Passwort des Servers ('libre') aufgenommen. Dabei wird der Schlüssel auf den Client exportiert. *Ab hier nur noch Kontakt über ssh zum Server*.
4. Jetzt erfolgt die Installation von PostgreSQL über ssh
   sudo apt-get update
   sudo apt-get -y install postgresql
   installiert hier die neueste Version mit allen Paketen.
5. **service postgresql start**
   startet den Server nach Eingabe des Passwortes des angemeldeten Nutzers
6. **sudo -u postgres createuser -P -d dblotest**
   erstellt in PostgreSQL den Benutzer 'dblotest', der auch Datenbanken erstellen darf. Bei der Erstellung wird nach einem Passwort für den Nutzer gefragt. Der Nutzer ist nicht identisch mit dem, der sich per ssh eingeloggt hat.
7. **sudo -u postgres createdb -O dblotest libretest**
   erstellt in PostgreSQL für den Benutzer 'dblotest' die Datenbank 'libretest'
8. Zum SSH-Server wird die Verbindung über
   logout
   geschlossen
9. Zum SSH-Server wird für PostgreSQL ein Tunnel über einen freien lokalen Port[8], hier '63333', erstellt. Soll nur der Tunnel, nicht aber die Shell benötigt werden, so kann **-N** (am Ende des folgenden Kommandos) den Kontakt der Shell unterbinden.
   ssh -L 63333:localhost:lotest@5432 192.168.178.32
10. Für die Verbindungseinstellungen unter LibreOffice gilt jetzt: Die Verbindung erfolgt über den Localhost des eigenen Rechners und den freien Post. Von der IP des Fremdrechners braucht LibreOffice nichts zu wissen.
    1. Die direkte Verbindung zum Server über LibreOffice wird wie folgt erstellt
       dbname=libretest host=localhost port=63333
    2. Die JDBC-Verbindung zum Server über LibreOffice wird wie folgt erstellt
       postgresql://localhost:63333/libretest
11. Die Eingabe des Benutzernamens 'dblotest' und des dazugehörigen Passwortes für die Datenbank ist natürlich weiterhin erforderlich. Unter PostgreSQL lässt sich aber auch eine Passwortdatei nach *https://www.postgresql.org/docs/14/libpq-pgpass.html* erstellen. Der Inhalt der Datei ist

8 Freie Ports sind in der Regel von Port 49152 – 65535 verfügbar

localhost:63333:libretest:dblotest:MeinPasswort

Die Datei wird unter Linux als **.pgpass** direkt im Homeverzeichnis abgespeichert. Sie sollte für andere Nutzer nicht lesbar sein.
Ist diese Datei für die entsprechende Datenbank vorhanden, so entfällt in Base die Angabe von Benutzername und Passwort.

Tipp

Die Angabe eines Passwortes kann nach einem Schlüsselaustausch entfallen. Auf der Konsole des Clients (nicht ssh) wird dazu angegeben:
ssh-keygen -t rsa
Alle Abfragen zu Dateiname und Passphrase, die nach diesem Befehl folgen, werden mit einem Return bestätigt. Wird der Dateiname anders gewählt, so klappt eventuelle das folgende Kommando nicht.
ssh-copy-id 192.168.178.32
kopiert den Schlüssel zum Server. Zum letzten Mal wird hier das Passwort abgefragt. Danach erfolgt das Einloggen per ssh ohne Passworteingabe.
Damit erscheint die Passwortabfrage nicht mehr und die Verbindung über ssh kann mit Hilfe von Startdateien erfolgen.
Unter **Linux** enthält die Datei **db_ssh.sh** den folgenden Inhalt:

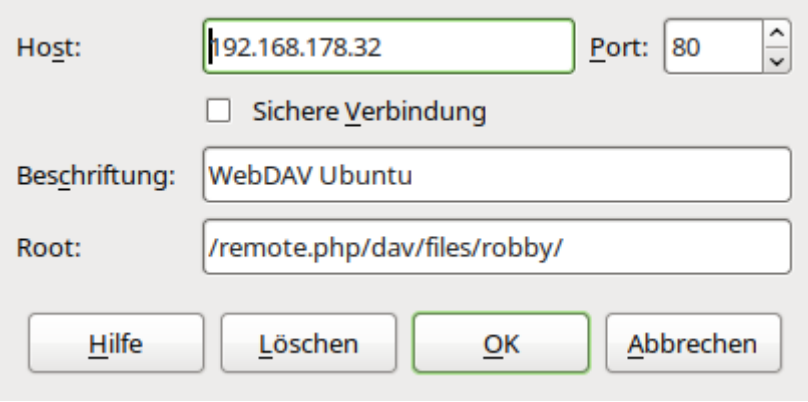
```
001 #!/bin/bash
002 ssh -L  63333:localhost:5432 192.168.178.32
```

Die Datei muss anschließend ausführbar gemacht werden.
Unter **Windows** enthält die **Date db_ssh.bat** den folgenden Inhalt:

```
001 @echo off
002 ssh -L  63333:localhost:5432 192.168.178.32
```

Datenbanken in der Cloud

Die WebDav-Einstellungen für das Laden von Dateien über **Datei → Vom Server öffnen...** sind in dem folgenden Dialog für einen internen Server ersichtlich:



Verbindung im lokalen Netz zu einer Nextcloud-Installation

Dabei ist darauf zu achten, dass der Host ohne den Zusatz **http://** oder **dav://** oder **webdav://** aufgeführt wird. Sonst wird die Verbindung falsch aufgetrennt und irrtümlich **http**, **dav** oder **webdav** als Host gelesen. Das wird dann deutlich, wenn der Dialog beim nächsten Öffnen unter **Root:** mit dem Eintrag von //192.168.178.32 (oder eben dem Hostnamen nach draußen) beginnt.

Wird der Dialog zum Öffnen der Dateien vom Server zum ersten Mal nach dem Öffnen von LibreOffice genutzt, so erfolgt die Benutzer- und Passwortabfrage.

Auf diesem Weg können Datenbankdateien geöffnet, aber nur unterschiedlich genutzt werden. Beim Zugriff auf eine interne ~~HSQLDB~~ erscheint der folgende Hinweis:

Es ist also nicht möglich, die Tabellen überhaupt zu sehen. Die interne ~~HSQLDB~~ braucht dazu zwingend den Zugriff auf den lokalen Rechner.

FIREBIRD läuft da etwas problemloser. WebDav lädt die Datei. Die Datenbank steht für Einträge und Recherchen auch über bereits erstellte Formulare zur Verfügung. Auch Berichte werden ausgeführt. Eine Erstellung neuer Formulare, Abfragen, Tabellen usw. über die grafische Benutzeroberfläche ist allerdings nicht möglich. Hierzu muss also die Datei erst auf dem lokalen Rechner bearbeitet werden und kann dann anschließend zur Nutzung über die Cloud wieder hoch geladen werden.

Dass eine Bearbeitung der *.odb-Datei nicht möglich ist ließe sich natürlich auch vorteilhaft nutzen. Wird z. B. eine Verbindung zu einem lokal auf dem Server laufenden PostgreSQL-Server genutzt, so läuft die Verbindung gleich dem, was auf dem eigenen lokalen Server eingestellt werden müsste. Es könnte also über die Cloud eine Bedienoberfläche zur Verfügung gestellt werden, die durch andere Benutzer nicht verändert werden kann, sehr wohl aber die Funktionalität der Datenbank voll unterstützt.

Bugs und Workarounds bei verschiedenen Datenbankverbindungen

Bestimmte Fehler bei Datenbankverbindungen tauchen bereits seit Beginn von LibreOffice auf. Diese Fehler werden hier aufgelistet, sofern dafür ein entsprechender Workaround existiert.

MySQL/MariaDB

Sobald bei einer MySQL- oder MariaDB-Datenbank der *JDBC-Treiber* gewählt wurde und der Kontakt zu einem Unterformular erstellt werden soll taucht leider die folgende Meldung auf: *Bug 50747*.

Diese in LO 7.4 auftauchende Meldung ist recht nichtssagend. Sie verschwindet, sobald unter **Bearbeiten → Datenbank → Erweiterte Einstellungen → Besondere Einstellungen → Benannte Parameter durch '?' ersetzen** ausgewählt wurde.

Mit keiner der Verbindungen zu MySQL/MariaDB können Datenbanknamen, die einen Punkt enthalten, korrekt wieder gegeben werden: *Bug 149434*. Wird ein Datenbankname wie "MyDB.libre.1" erstellt, dann wird neben diese Datenbank in der *.odb-Datei eine Datenbank "MyDB" angezeigt. Eine bereits in "MyDB.libre.1" anderweitig erstellte Tabelle "tbl_Person" erscheint nicht als Tabelle der Datenbank "MyDB.libre.1", sondern als Tabelle der Datenbank "MyDB" mit dem Namen "libre.1.tbl_Person". Daten können aus der Tabelle gelesen werden, wenn der korrekte Datenbankname und der korrekte Tabellenname gewählt werden. Die Tabellen sind aber schreibgeschützt. Die Verwendung von Punkten in Datenbanknamen sollte also unterlassen werden.

Wir die direkte Verbindung zu MariaDB gewählt, so kann es bei der Auswertung des Feldtyps BIT zu Problemen kommen. Während ein Feld dieses Typs in Tabellen, Abfragen und Formularen korrekt als Markierfeld dargestellt wird gibt eine Abfrage über **Extras → SQL** oder auch über Makros für das Feld einen leeren Inhalt aus: *Bug 153317*. Mit der Abfrage

```
001 SELECT CASE WHEN `Bitfeld` THEN 1 ELSE 0 END FROM `test`.`tbl_Test`
```

wird das Feld nicht leer, sondern mit den Zahlen '1' für 'Ja' und '0' für 'Nein' dargestellt.

PostgreSQL

Beim *direkten Treiber* funktioniert die Erstellung der AutoWert-Felder nicht: *Bug 60643*. Dieser Bug kann umgangen werden, indem die Tabellen mit AutoWert-Feldern in einer JDBC-Verbindung erstellt werden. Am sichersten ist aber die Erstellung über Extras → SQL:

```
001 CREATE TABLE "public"."Test" (
002 "ID" SERIAL PRIMARY KEY
003 );
```

Hiermit wird ein Feld des Typs **SERIAL** erstellt, das vom Umfang her einem **INTEGER**-Feld entspricht. Es wird als einziger Feldtyp in PostgreSQL mit dem direkten Treiber auch als <AutoWert> angezeigt: *Bug 147497*. Allerdings ist darauf zu achten, dass der Feldname kein Leerzeichen enthält. Ansonsten werden die in PostgreSQL erzeugten Werte nicht korrekt wieder an die GUI zurückgegeben: *Bug 152478*. Statt eines Rückgabewertes erscheint dann einfach eine '0'. Diese '0' verhindert eine anschließende Änderung des Datensatzes und z. B. eine Verknüpfung eines Formulars mit einem Unterformular über den AutoWert.

Ist die Tabelle wie oben erstellt, so kann sie anschließend um die notwendigen Felder im Tabelleneditor von Base ergänzt werden.

Werden in PostgreSQL Abfragen erstellt, so erstellt die GUI diese Abfragen automatisch mit einem Alias in Form des Tabellennamens:

```
001 SELECT * FROM "public"."Test" "Test"
```

Die Erstellung mit Aliasnamen kann (für *JDBC*) nicht über die erweiterten Einstellungen der Datenbankverbindung ausgeschaltet werden: *Bug 152450*. Für den direkten Treiber ist so eine Einstellung gar nicht erst verfügbar.

Dieser Bug wäre nicht weiter erwähnenswert, wenn er nicht einen weiteren Bug nach sich zieht: In Abfragen, die mit einem Alias für den Tabellennamen erstellt wurden, wird das AutoWert-Feld grundsätzlich nicht aktualisiert: *Bug 130376*. So muss also jede Abfrage, die in der GUI bearbeitet wurde, anschließend im ausgeschalteten Design-Modus von den Alias-Benennungen für die Tabellen befreit werden.

Der Alias-Bug hat dann natürlich zur Folge, dass entsprechende Konstruktionen wie korrelierende Unterabfragen zu entsprechenden Problemen führen. Es ist also eventuell erforderlich, in Formulare grundsätzlich Makros ein zu bauen, die Datensätze nach einer Neueingabe aktualisieren, damit die automatisch erstellten Werte auch angezeigt werden:

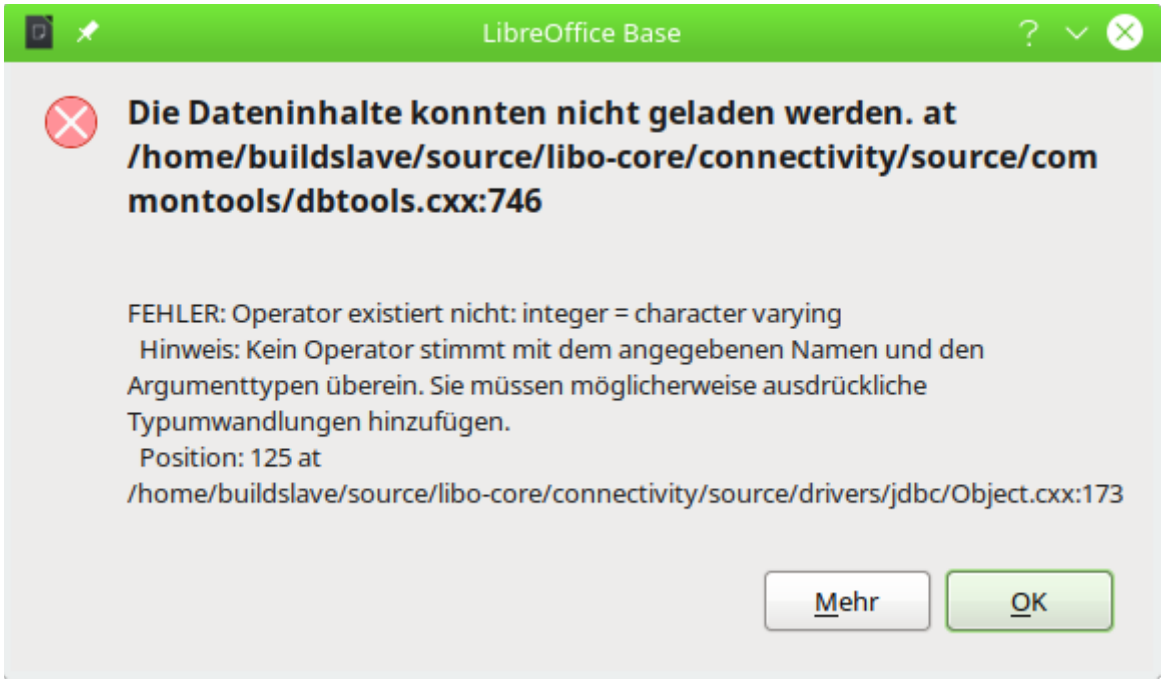
```
001 GLOBAL boNew AS BOOLEAN

002 SUB OldNew(oEvent AS OBJECT)
003    DIM oForm AS OBJECT
004    oForm = oEvent.Source
005    IF hasUnoInterfaces(oForm, "com.sun.star.form.XForm" ) THEN
006       IF oForm.getInt(1) = 0 THEN
007          boNew = True
008       ELSE
009          boNew = False
010       END IF
011    END IF
012 END SUB
```

Vor der Datensatzaktion soll zuerst einmal mit der Prozedur **OldNew** ermittelt werden, ob es sich um einen neuen Datensatz handelt. Nur dann sind die automatisch erstellten Werte ja nicht korrekt vorhanden. Diese Prozedur schreibt in die globale Variable **boNew** den Wert **True**, wenn der erste Wert in der Tabelle den Integer-Wert '0' ergibt. Dabei wird einfach davon ausgegangen, dass die erste Spalte eben den automatisch erstellten Schlüsselwert enthält.

```
001 SUB FormNew(oEvent AS OBJECT)
002    DIM oForm AS OBJECT
003    oForm = oEvent.Source
004    IF hasUnoInterfaces(oForm, "com.sun.star.form.XForm" ) THEN
005       IF boNew = True THEN
006          oForm.Reload
007          oForm.Last
008          boNew = False
009       END IF
010    END IF
011 END SUB
```

Nach der Datensatzaktion soll das Formular mit der Prozedur **FormNew** neu eingelesen werden, wenn in dem Formular ein neuer Datensatz eingespeichert wurde, also **boNew** auf **True** gesetzt wurde. Nach dem **Reload** des Formulars wird der Cursor wieder auf den letzten Datensatz gesetzt. Dies ist der Datensatz, der als neuer Datensatz erstellt wurde.

Wird für PostgreSQL der *JDBC-Treiber* genutzt, so kann es bei Formularen mit UnterUnter-Formularen zu der Meldung kommen, dass ein Parameter nicht korrekt gesetzt werden kann: *Bug 147582*.



Angeblich wird der Parameter als VARCHAR weiter gegeben, aber als INTEGER erwartet. Diese Meldung taucht nur dann auf, wenn das Unterformular noch keinen Datensatz enthält, das UnterUnterformular also komplett ohne Eingabemöglichkeit sein müsste.

Hier muss in dem UnterUnterFormular das Feld, das eigentlich als INTEGER-Feld in der Datenbank vorhanden ist, zusätzlich als VARCHAR-Feld eingebunden werden. Das geschieht mit einer Abfrage ähnlich der folgenden:

```
001 SELECT "ID", "Ort_ID", "Vorname", "Nachname",
002 CAST( "Ort_ID" AS VARCHAR ( 10 ) ) "FK"
003 FROM "public"."tbl_Mitglieder"
```

Die Tabelle "tbl_Mitglieder" ist über den Fremdschlüssel "Ort_ID" mit einer anderen Tabelle verbunden. Das UnterUnterformular, das diese Abfrage als Datenbasis nutzt, ist eigentlich mit dieser "Ort_ID" mit dem darüber liegenden Unterformular verbunden. Leider wird bei einem nicht vorhandenen Datensatz des darüber liegenden Unterformulars an das UnterUnterformular eine Variable weiter gegeben, die nicht dem Typ INTEGER entspricht – einfach keine Zahl, sondern vermutlich ein leerer Text. Deswegen wird in Zeile 2 die "Ort_ID" in einen Text mit maximal 10 Zeichen umgewandelt. Die Verbindung zwischen Unterformular und UnterUnterformular wird jetzt nicht zu "Ort_ID" gesetzt sondern zu "FK".

Leider ist so natürlich das Feld "Ort_ID" in dem UnterUnterFormular leer, wenn ein neuer Datensatz abgespeichert werden soll. Hier muss mit einem Makro gegen gesteuert werden:

```
001 SUB NewRow(oEvent AS OBJECT)
002    DIM oForm AS OBJECT
003    DIM inID AS INTEGER
004    oForm = oEvent.Source
005    IF hasUnoInterfaces(oForm, "com.sun.star.form.XForm" ) THEN
006       IF oForm.isNew THEN
007          inID = oForm.Parent.getInt(1)
008          oForm.UpdateInt(2,inID)
009       END IF
010    END IF
011 END SUB
```

Vor der Datensatzaktion wird dem Formular im 2. Feld (Zeile 8) der gleiche Wert zugewiesen, der im darüber liegenden Formular im 1. Feld steht (Zeile 7). Hier wird davon ausgegangen, dass das 1. Feld im darüber liegenden Formular den Schlüsselwert enthält, der als Fremdschlüssel an das 2. Feld in der Abfrage weiter gegeben werden soll.

Sobald bei einer PostgreSQL-Datenbank der *JDBC-Treiber* gewählt wurde und der Kontakt zu einem Unterformular erstellt werden soll taucht leider die folgende Meldung auf: *Bug 50747*.

Diese in LO 7.4 auftauchende Meldung ist recht nichtssagend. Sie verschwindet, sobald unter **Bearbeiten → Datenbank → Erweiterte Einstellungen → Besondere Einstellungen → Benannte Parameter durch '?' ersetzen** ausgewählt wurde.

dBase

Beim Auslesen der Dateien, die in einem Verzeichnis liegen, werden nur dBase-Dateien berücksichtigt, die mit der in Kleinbuchstaben geschriebene Dateiendung *.dbf versehen sind. Andere Endungen wie z.B. *.DBF werden nicht ausgelesen: *Bug 46180*. Hier brauchen einfach nur die Dateiendungen umgeschrieben zu werden.

Der Importassistent für dBase hat Probleme, numerische Feldtypen und Ja/Nein-Felder automatisch richtig zu erkennen: *Bug 53027*. Hier muss entsprechend nachgebessert werden bevor die Inhalte z. B. in die interne HSQLDB eingelesen werden sollen.

Tabellen

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.

Hinweis

Manche Inhalte geben zwar vor, Zahlen zu sein, sind aber nicht als Zahlen komplett speicherbar. Dies gilt im deutschsprachigen Raum für die «Telefonnummer» und die «Postleitzahl». Beide können auch führende Nullen enthalten. Für diese Felder muss also genauso wie für eine «Bankleitzahl» ein Feldtyp **`Text (Varchar)`** erstellt werden.

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:

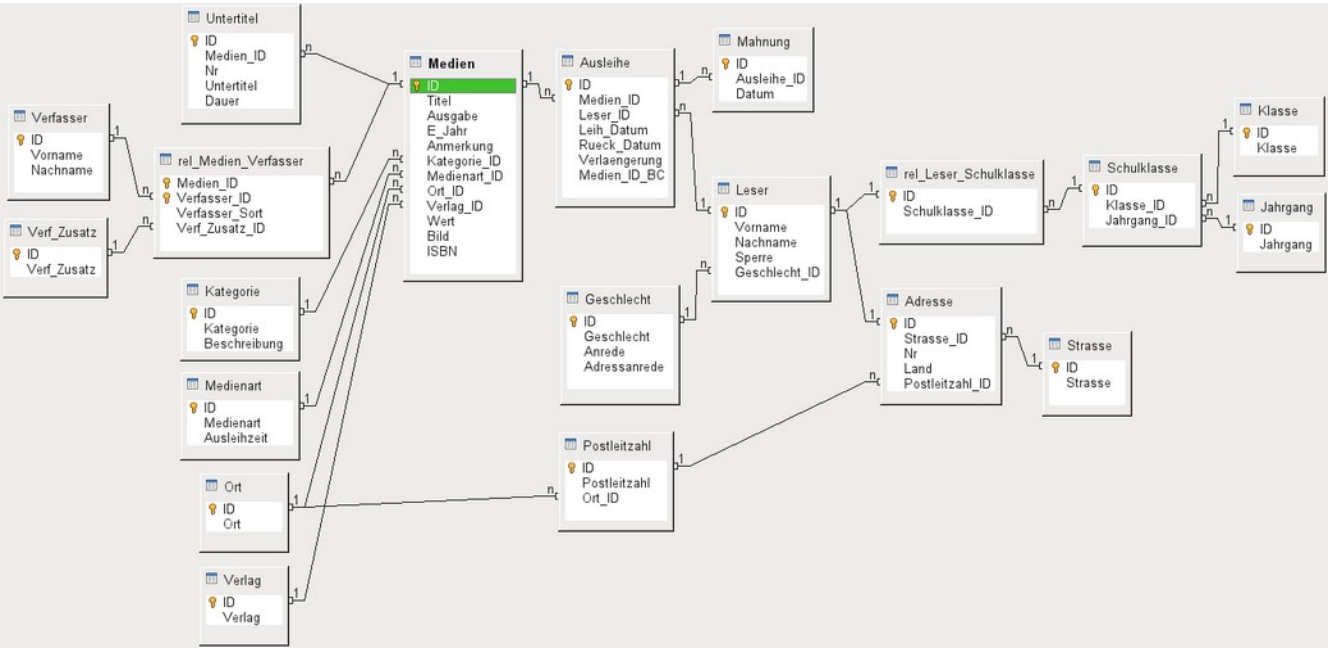


Abbildung 1: Beziehungsentwurf der Beispieldatenbank «Medien_ohne_Makros»

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.

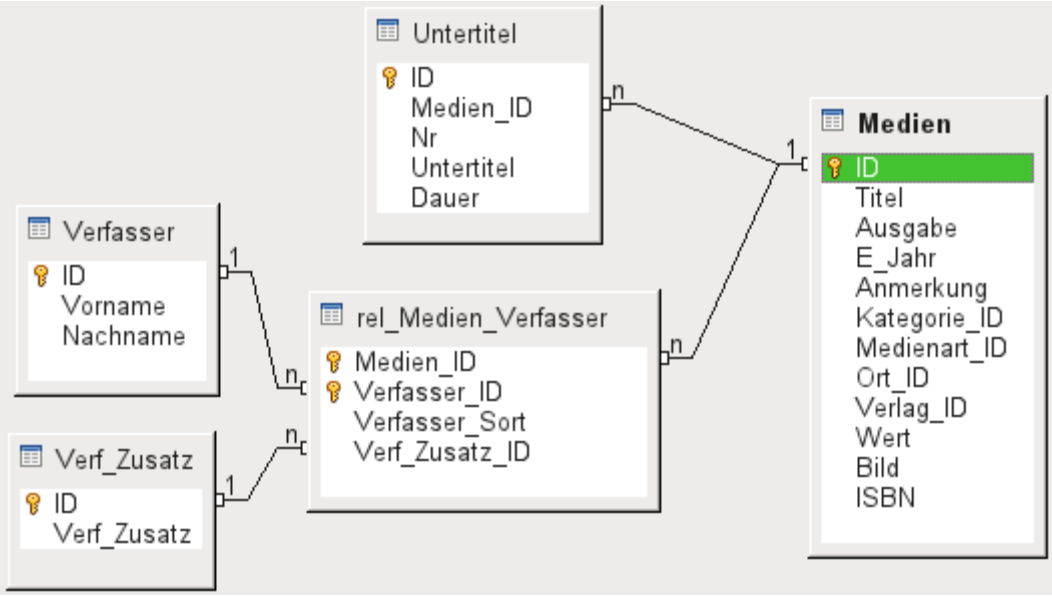


Abbildung 2: Beispiel 1:n-Beziehung; n:m-Beziehung

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:

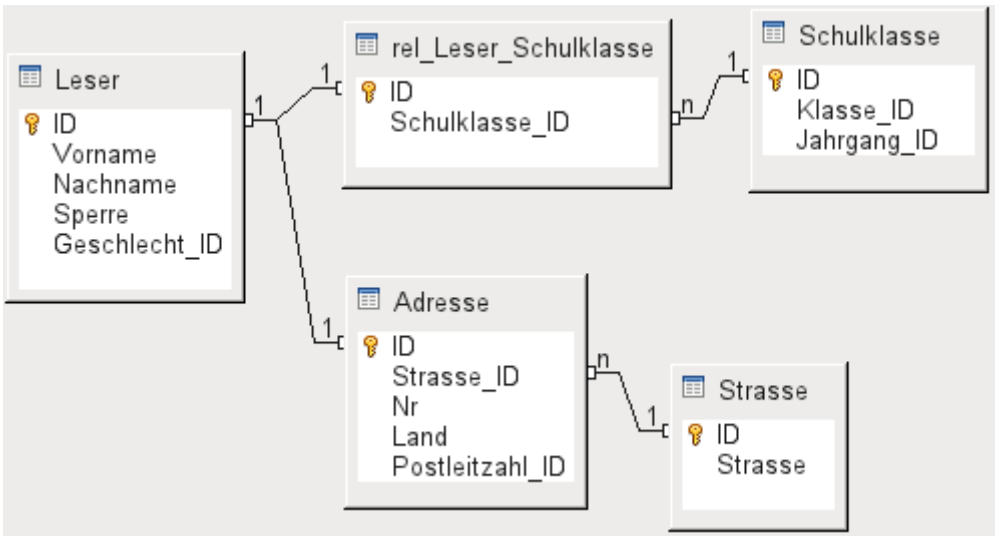


Abbildung 3: Beispiel 1:1-Beziehung

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.

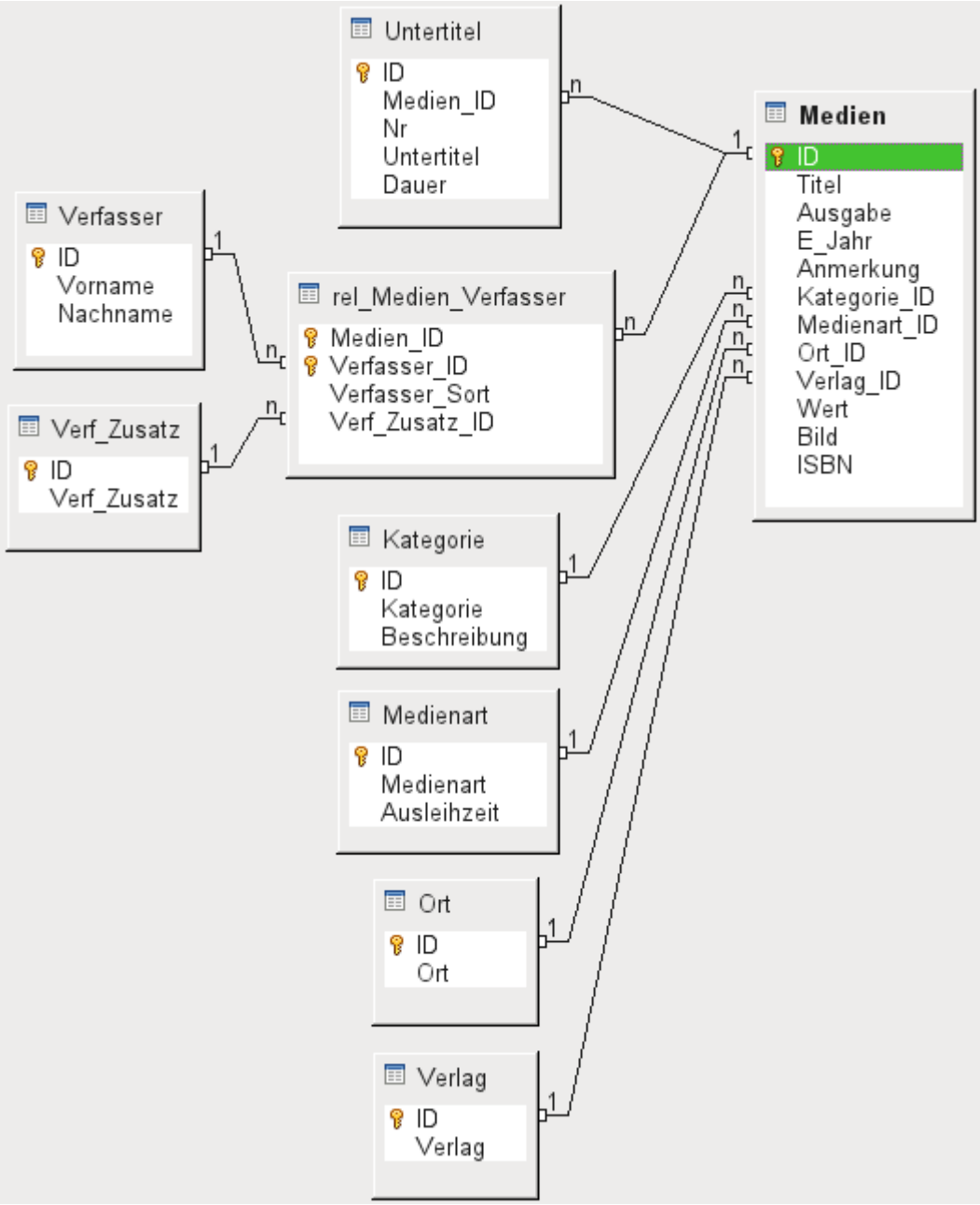


Abbildung 4: Medienaufnahme

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

Tabellen Ausleihe

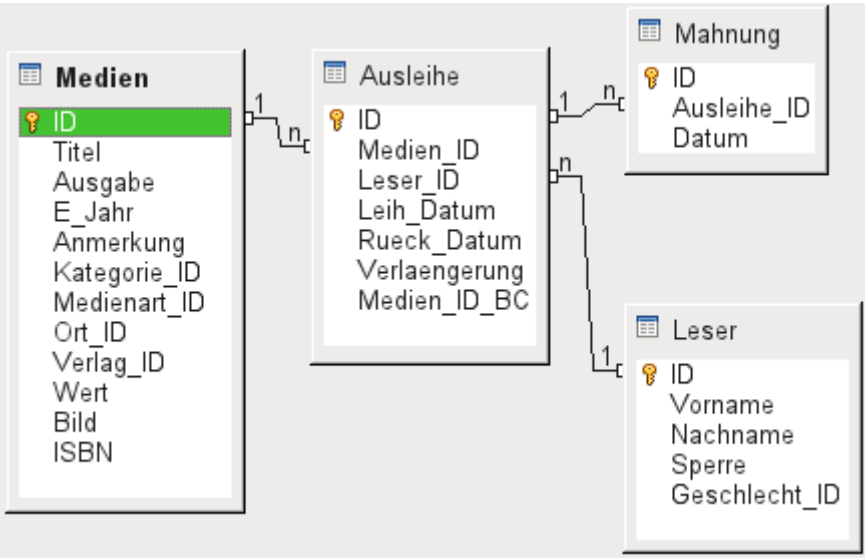


Abbildung 5: Ausleihe

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.

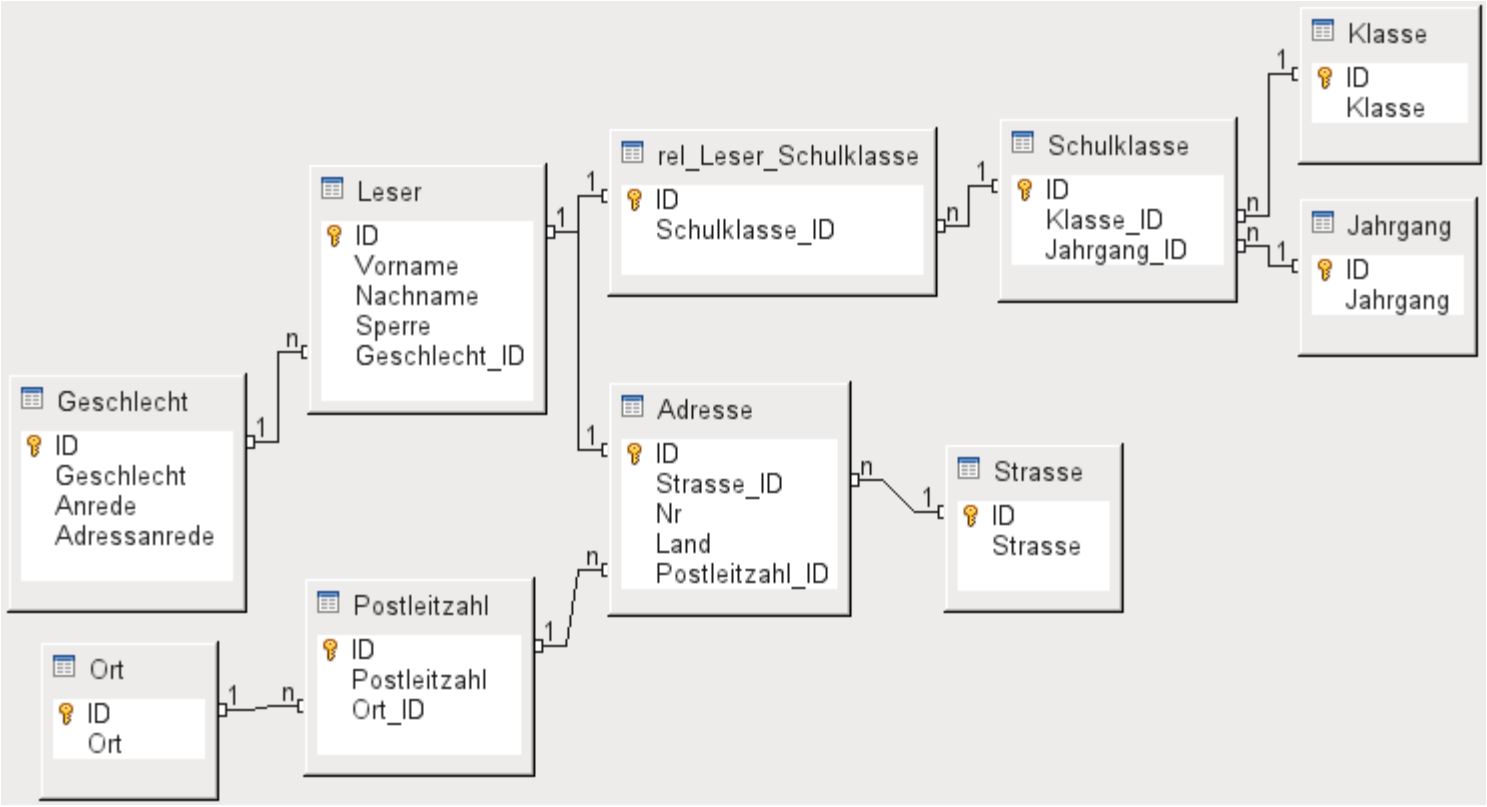


Abbildung 6: Leser - ein Schulklassenstrang und ein Adressenstrang

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.

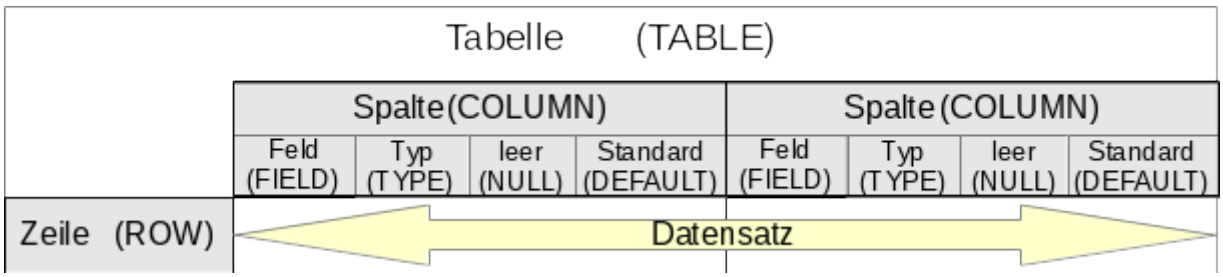
Die "**Adresse**" wird ebenfalls sehr differenziert dargestellt. Die Straße ist aus der Adresse ausgelagert, da Straßennahmen innerhalb eines Ortes häufiger wiederholt werden. Postleitzahl und Ort sind voneinander getrennt, da oft mehrere Postleitzahlen für einen Ort gelten. Für die Post sind alle Ortsbezeichnungen,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

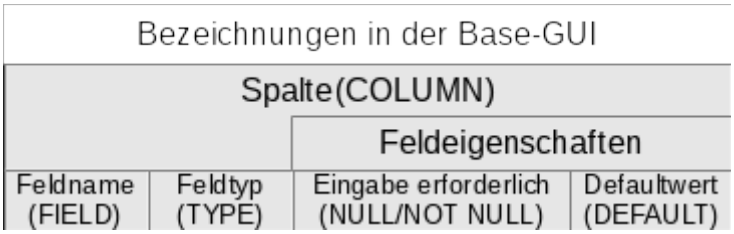
Bezeichnungen bei Tabellen:



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

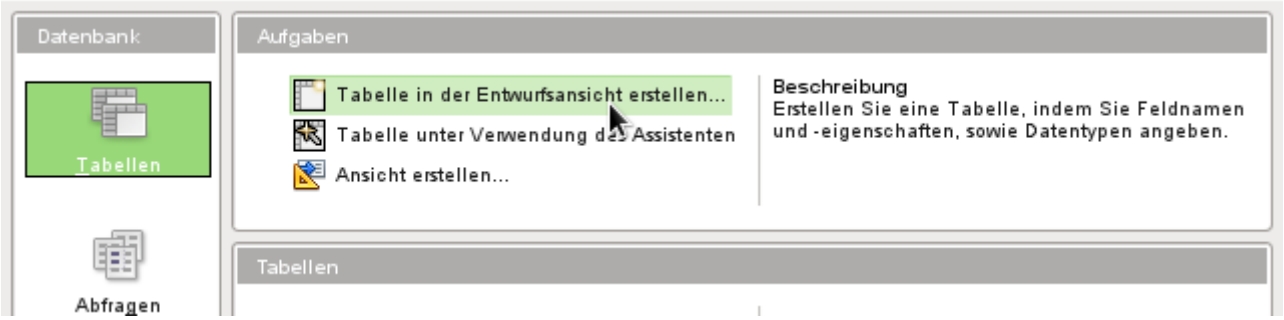
Die Firebird-Datenbank hat in der Standardeinstellung Probleme mit der Sortierung von Groß- und Kleinschreibung und auch Umlauten. Der folgende Schritt sollte **vor der Erstellung der Tabellen** in Firebird einmal ausgeführt werden.
Über **Extras → SQL** wird die direkte Eingabe geöffnet.

```
003  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.
Auch in den bestehenden Tabellen werden neu hinzugefügte Felder entsprechend eingefügt. Hier könnten also alte Felder durch neue zusätzliche Felder ergänzt, der Inhalt kopiert und die alten Felder gelöscht werden. Dabei ist aber Vorsicht geboten: Die neuen Felder sollten erst nach einer Abspeicherung der Datenbank und einem Neustart umbenannt werden. Sonst kann es passieren, dass Firebird weiter die vorher bereits bestehenden Felder nutzt.

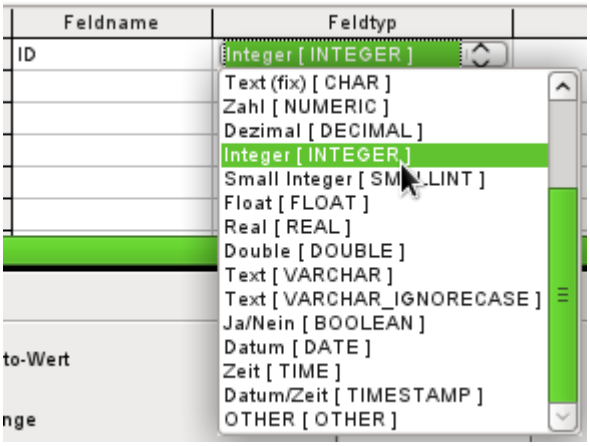
Erstellung mit der grafischen Benutzeroberfläche

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».

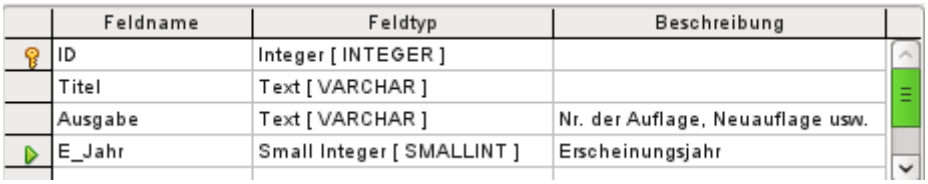
Hinweis

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.

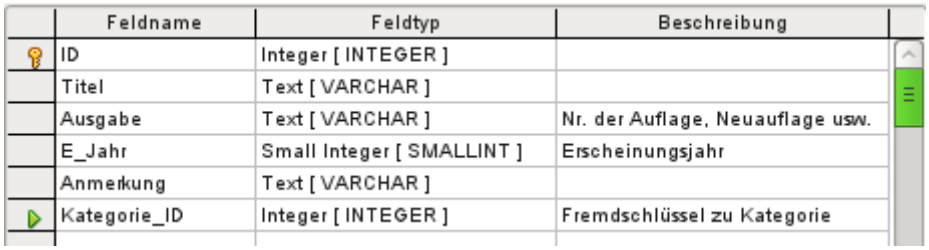
- Bei den Feldeigenschaften für das Feld «ID» wird die Eigenschaft **Auto-Wert → Ja** gesetzt. Damit wird der Primärschlüssel automatisch hoch gezählt. Die Zählung beginnt bei der internen HSQLDB-Datenbank mit '0'.
  Der Auto-Wert lässt sich nur für **ein** Feld einer Tabelle einstellen. Die Auswahl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

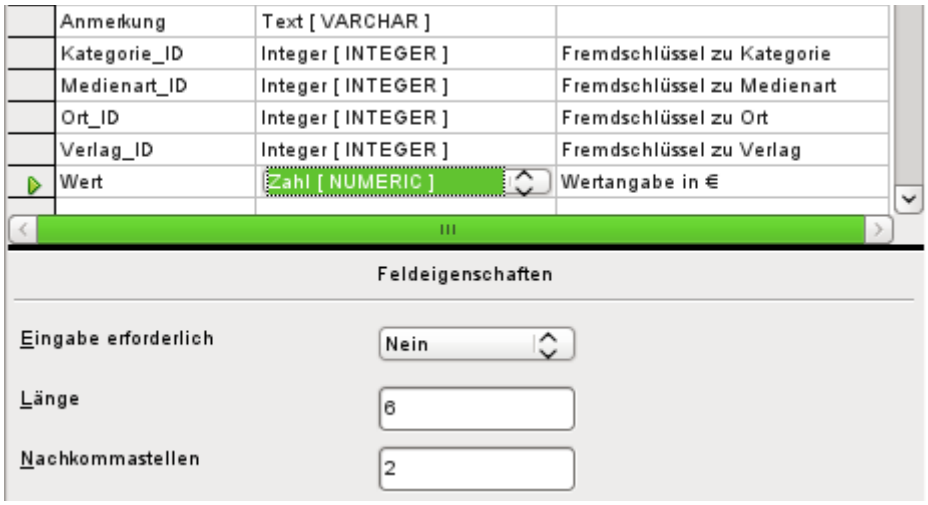
   - In den Feldeigenschaften muss die Länge des Feldes angepasst werden. Die Standardlänge ist hier bei den neueren LO-Versionen 100, sollte für den Medientitel aber auf 250 Zeichen erweitert werden.
   - In den Feldeigenschaften sollte wenigstens bei diesem Feld **Eingabe erforderlich → Ja** gewählt werden.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



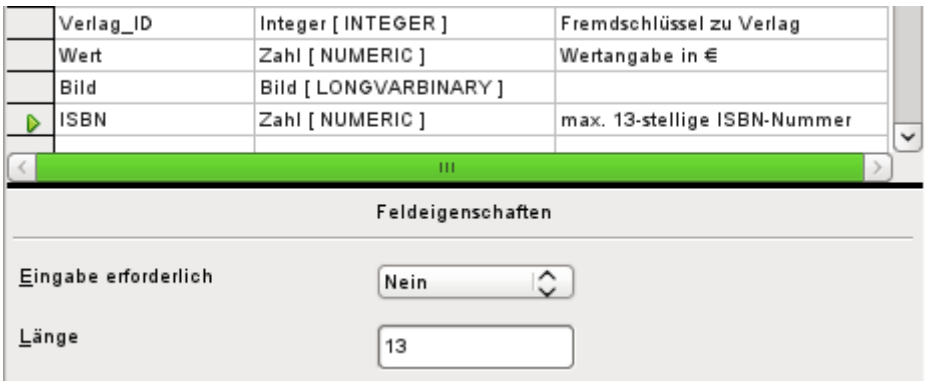
3. **Beschreibung** kann alles sein; die Spalte kann auch leer gelassen werden. Die Beschreibung dient nur zur Erklärung des Feldinhaltes für Personen, die sich später einmal den Tabellenentwurf ansehen wollen.
4. Für das Feld «E_Jahr» wird der Zahlentyp **`Small Integer [SMALLINT]`** gewählt. Dabei 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.

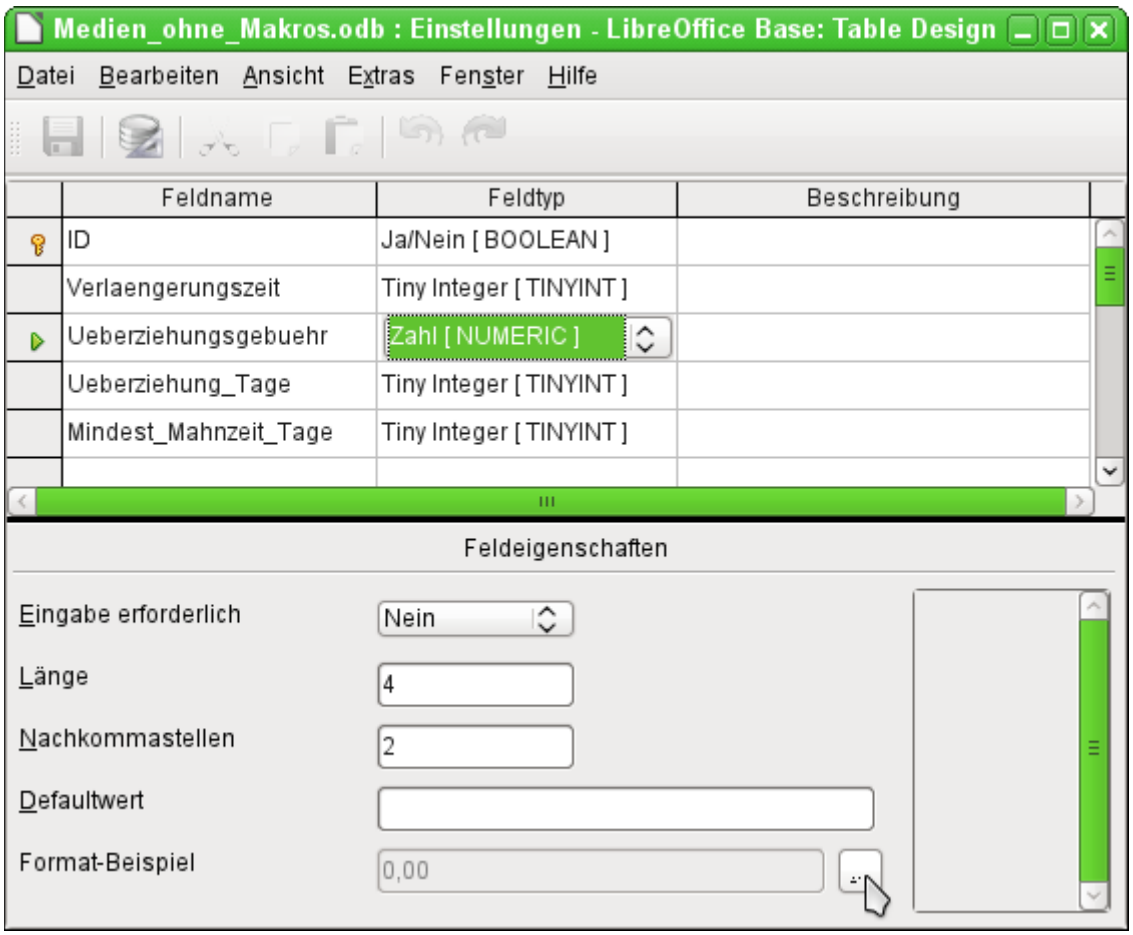
Formatierung von Feldern

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.

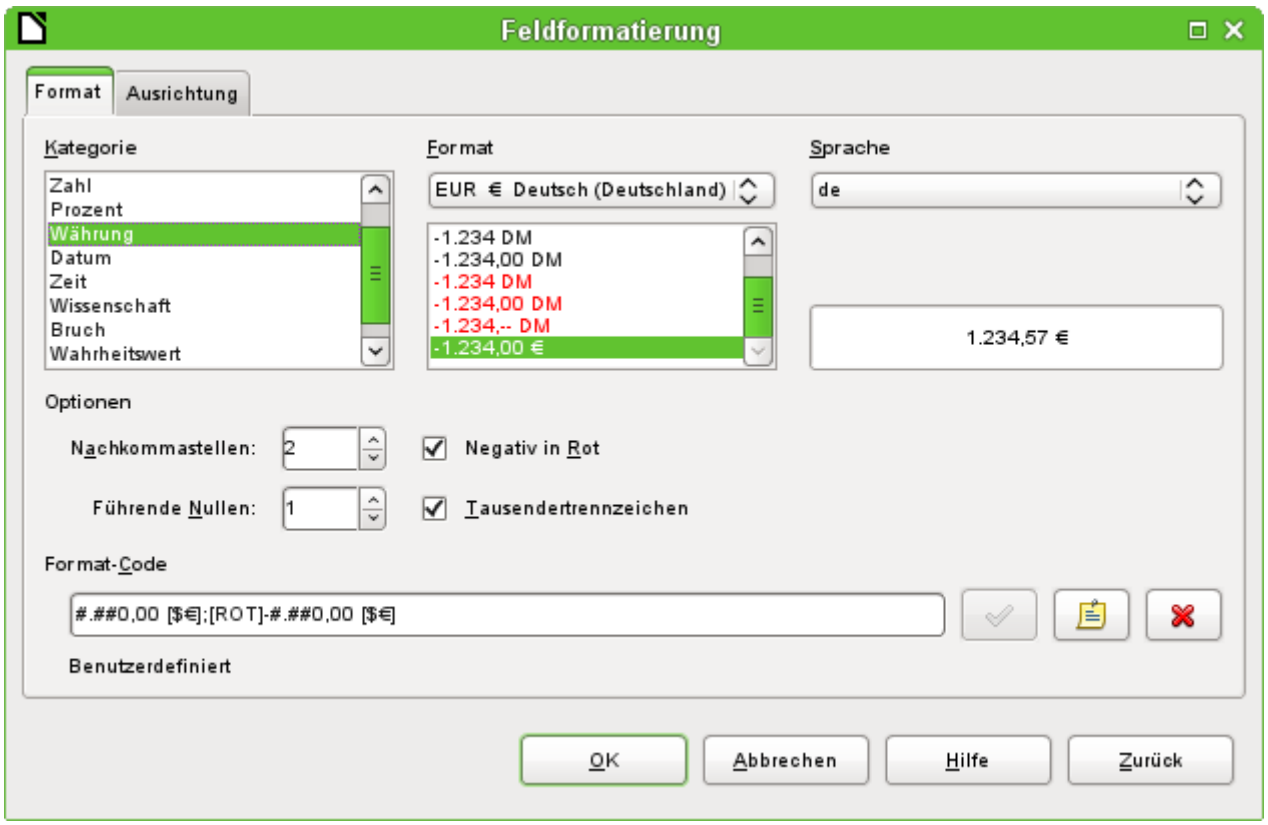
Die Formatierung zeigt die Inhalte nur an. Eine Datumsdarstellung mit einer zweistelligen Jahreszahl wird dennoch als vierstellige Jahreszahl gespeichert. Wird ein Feld für eine Zahl mit zwei Nachkommastellen erstellt, wie es bei der Überziehungsgebühr in dem folgenden Beispiel der Fall ist, so wird 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.

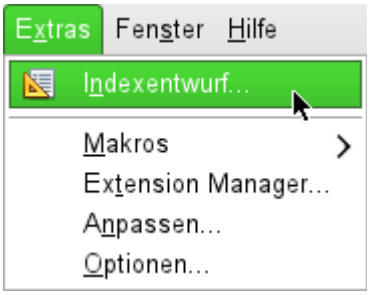


Abbildung 7: Zugriff auf den Indexentwurf

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.

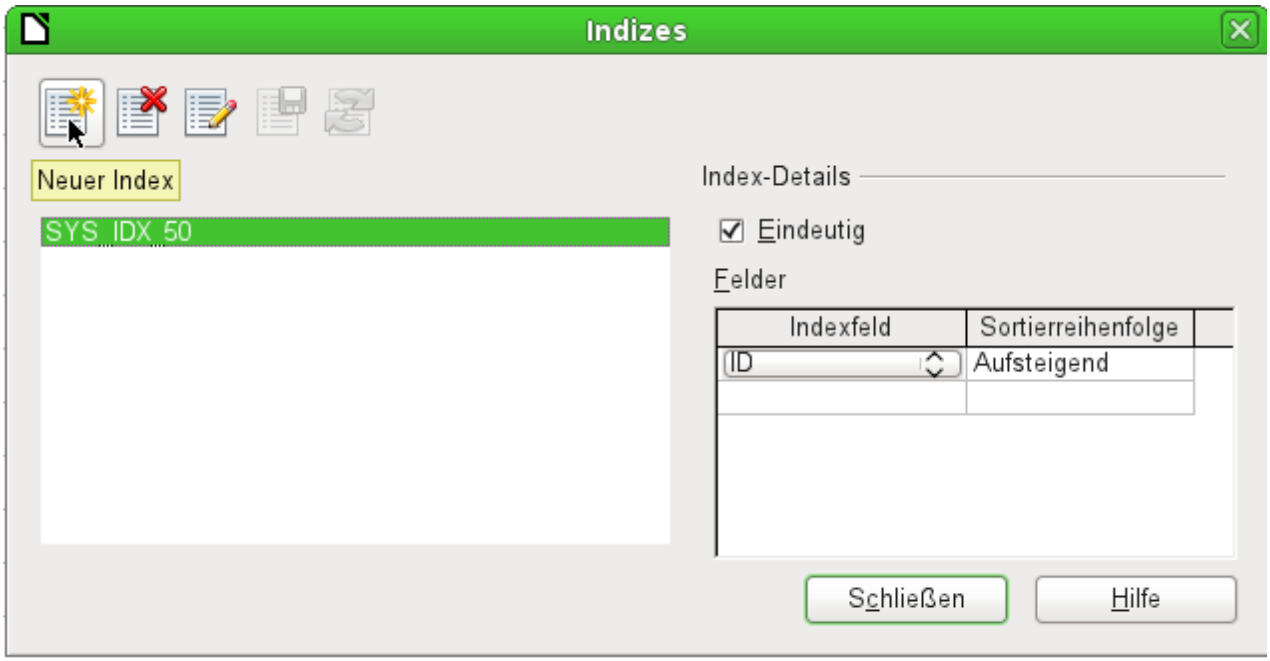


Abbildung 8: Erstellen eines neuen Indexes

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

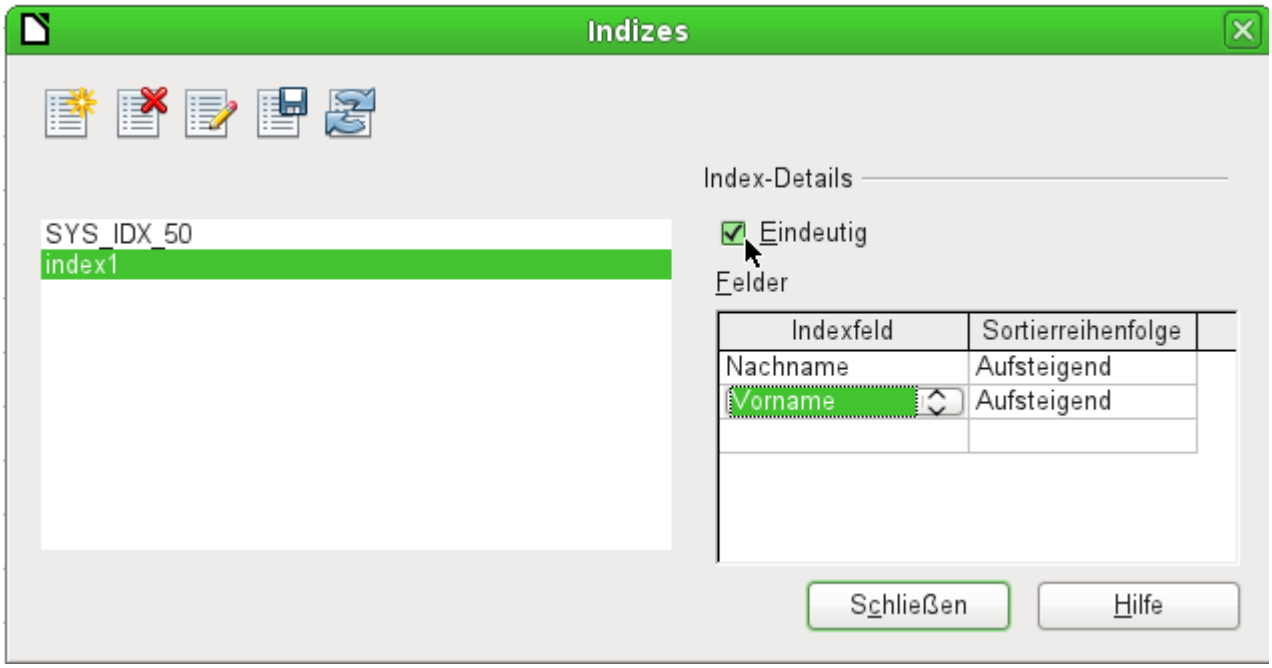


Abbildung 9: Der Index wird als «Eindeutig» definiert.

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.

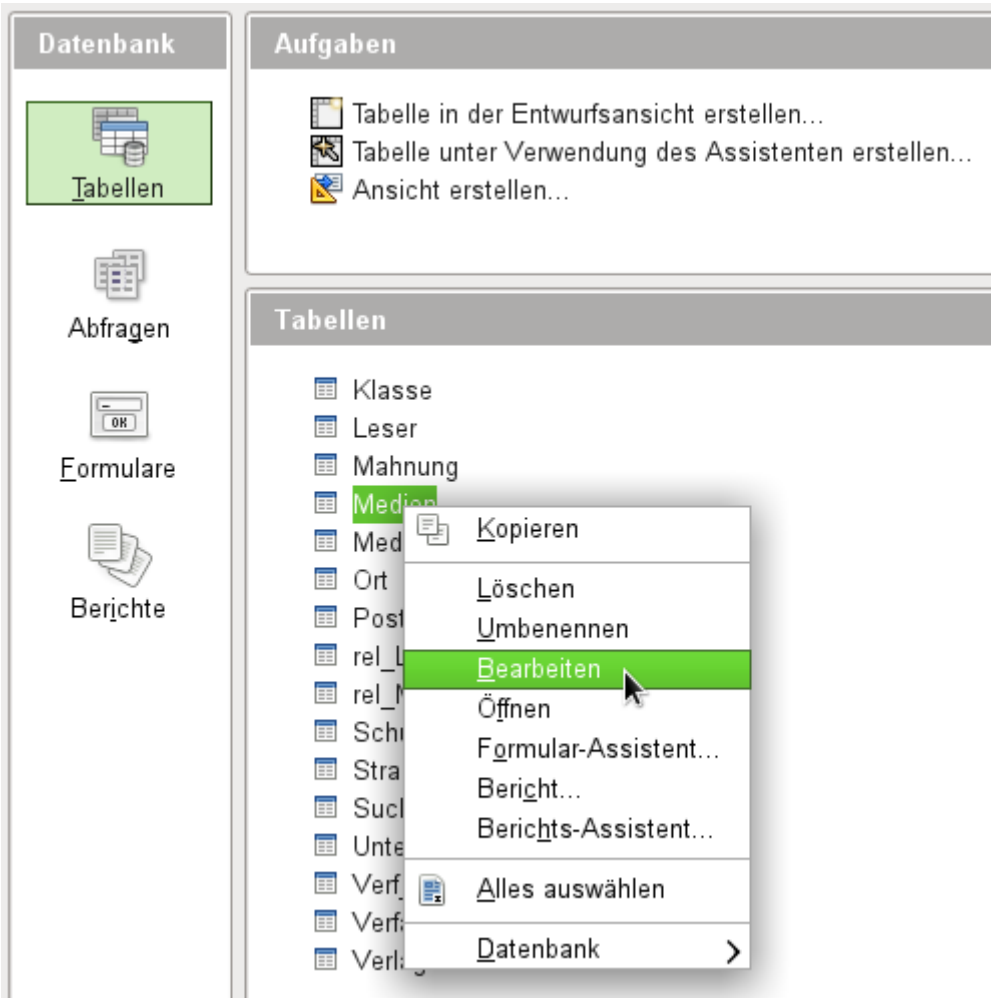
Hinweis

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.

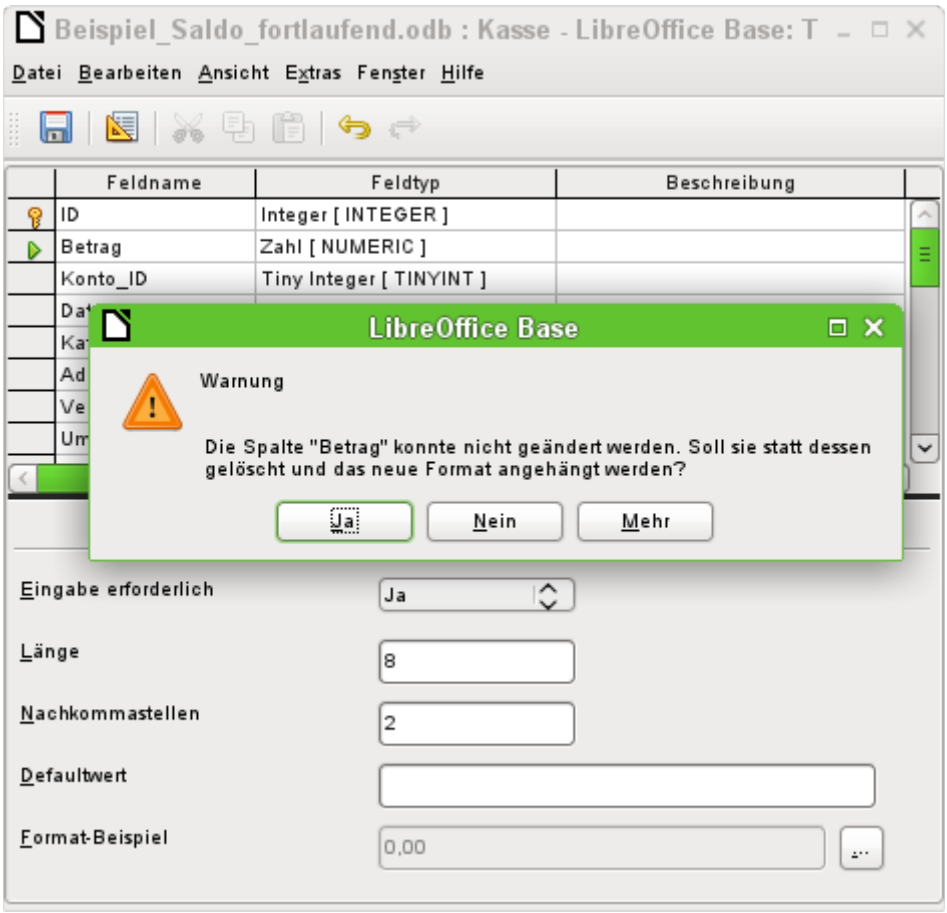
Hinweis

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.

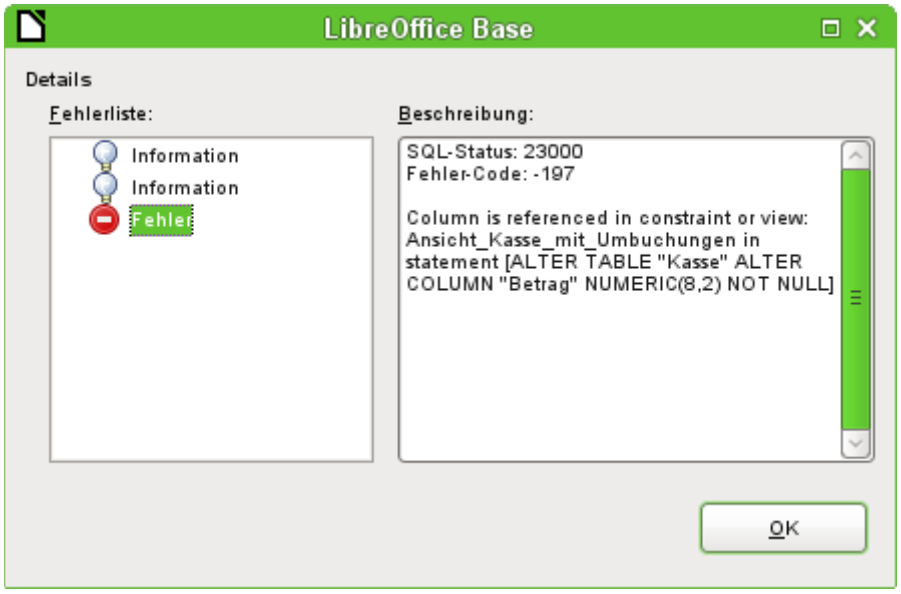


Hier sollte das Feld "Betrag" in der Eigenschaft «Eingabe erforderlich» auf «Ja» umgestellt werden. Das Warnsymbol macht bereits darauf aufmerksam: Die Änderung kann zu einem Verlust von Daten führen. Eine einfache Änderung ist nicht möglich. Bereits vorher wurde ausgeschlossen, dass in dem Feld "Betrag"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

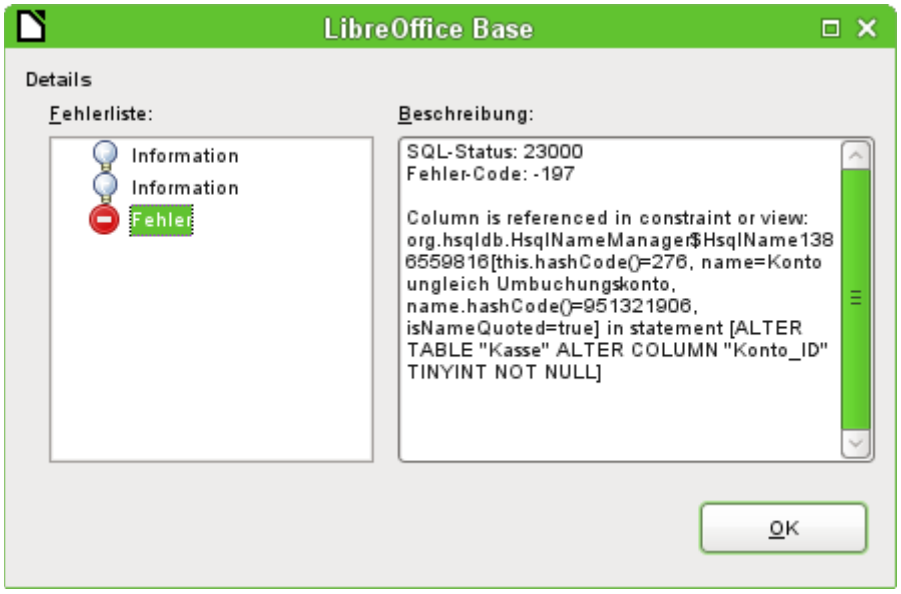
Ein Klick auf Ja führt zu einer weiteren Fehlermeldung, da die Datenbank die Löschung nicht zulässt. Ein Klick auf Nein bricht den gesamten Vorgang ab. Häufig wird die Frage nach Mehr Informationen gar nicht erst gestellt, weil die Informationen nur mit etwas gründlicherem Wissen auch sinnvoll zu einer 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.[9]

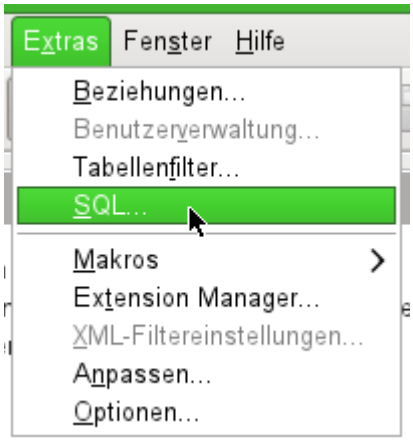
Nur über direkte SQL-Eingabe kann ein neues Feld an eine bestimmte Position innerhalb der Tabelle rutschen. Bereits erstellte Felder sind bei der internen HSQLDB nicht beweglich. Für Firebird müssen die Felder erstellt werden und können dann an die entsprechende Position verschoben werden.

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*).

9 *Bug 51605*

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.

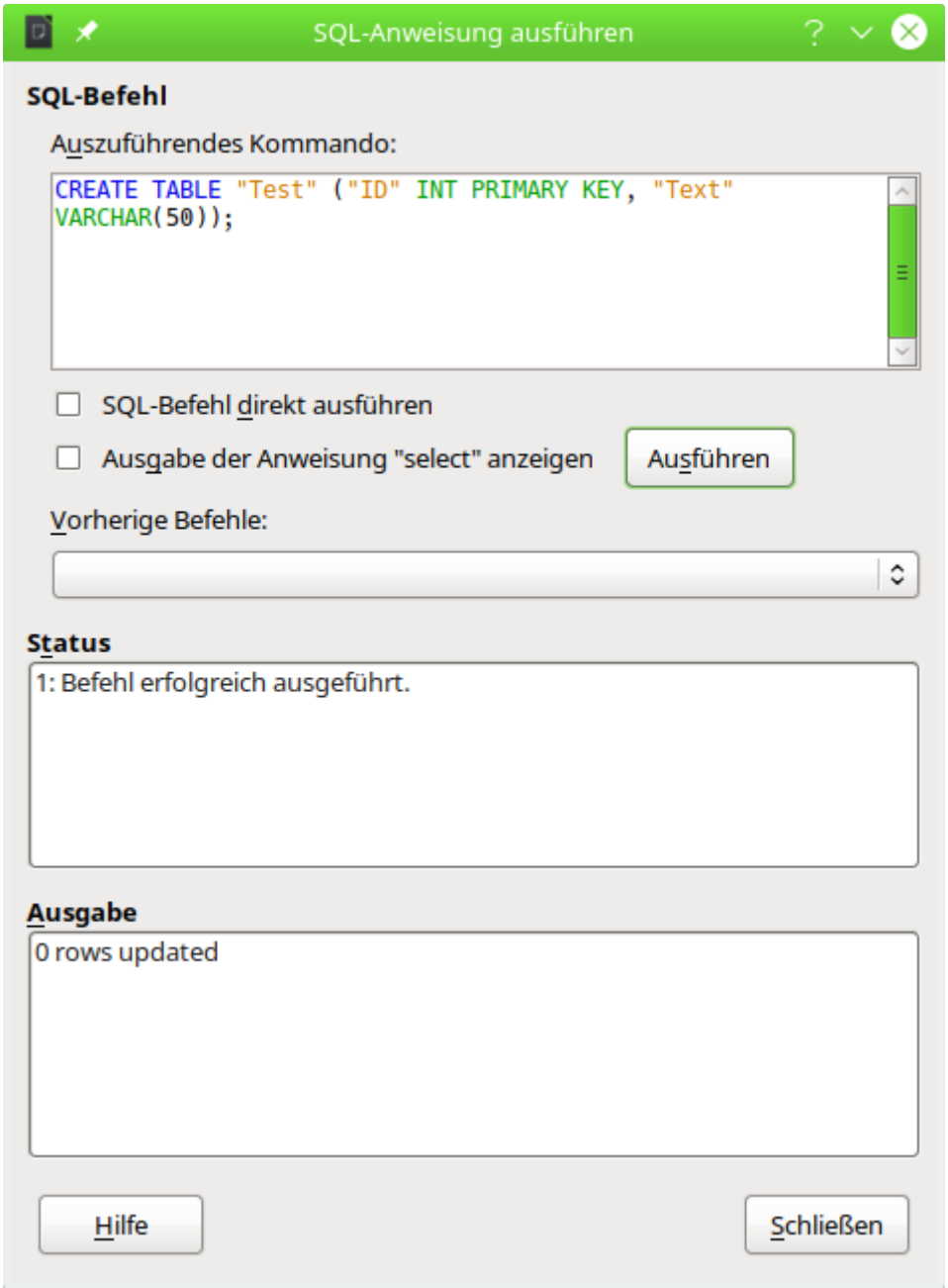


Abbildung 10: Fenster für direkte Eingabe von SQL-Befehlen

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);
001 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"];
001 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**).

Hinweis

Um in FIREBIRD einen Feldtyp zu ändern muss das Schlüsselwort TYPE hinzugefügt werden:

```
001 ALTER TABLE "Tabellenname" ALTER COLUMN "Feldname" TYPE VARCHAR(12);
```

Dies ändert den Datentyp des entsprechendes Feldes auf ein Textfeld mit maximal 12 Zeichen. Werden weniger Zeichen angegeben als erforderlich sind, um den Inhalt zu speichern, so wird der Befehl mit einer entsprechenden Meldung nicht ausgeführt. Die Zeichenzahl wird dabei angegeben, so dass eine entsprechende Korrektur möglich ist.

Hinweis

Die Längenangaben in Zeichen, die Firebird für einen Inhalt ausliest, speichern sicher diesen Inhalt. Allerdings decken sie sich nicht mit den Textlängen, die tatsächlich nötig sind, da Firebird einen recht eigenwilligen Umgang mit Sonderzeichen gegenüber ASCII-Zeichen hat und dies für die GUI von Base auf das normale Verhalten angepasst wurde. So wird z. B. eine Länge von 11 Zeichen bei einer 5-stelligen Zahl als erforderlich angegeben. Der Inhalt kann aber anschließend sehr wohl in ein Feld des Typs VARCHAR(5) eingelesen werden.
Vor diesem Hintergrund erscheint es manchmal sinnvoller, ein neues Feld mit den entsprechend gewünschten Eigenschaften zu erstellen, den Inhalt zu kopieren und anschließend das nicht mehr benötigte Feld zu löschen.

```
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[10]. 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.

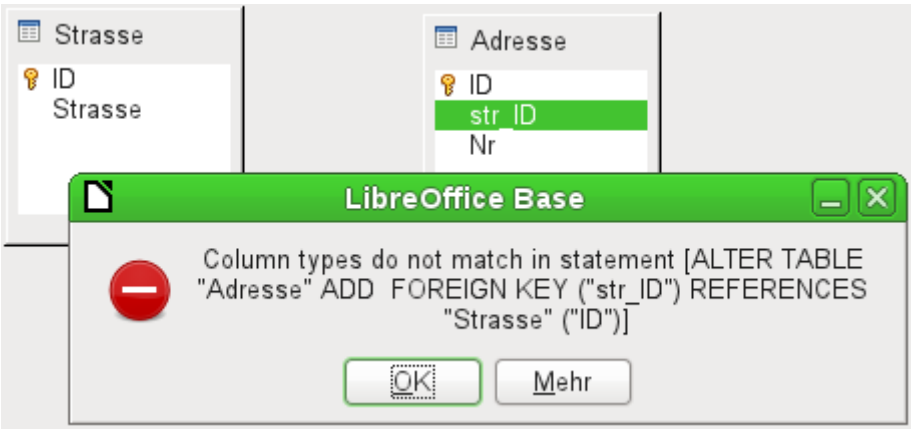
Das Löschen von Datensätzen bei verknüpften Tabellen ist nicht so einfach möglich. Angenommen es würde aus der Tabelle "Strasse" in *Abbildung 11* eine "Strasse" gelöscht, die aber durch die Verknüpfung mit der Tabelle "Adresse" in der Tabelle "Adresse" noch als Fremdschlüssel vertreten ist. Der Verweis in der Tabelle "Adresse" würde ins Leere gehen. Hiergegen sperrt

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

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.

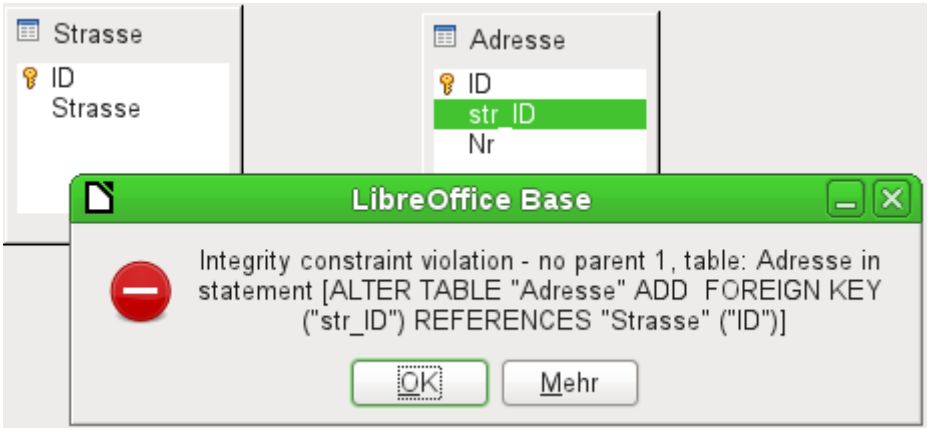
Die grundsätzlichen Verknüpfungen werden über **Extras → Beziehungen** festgelegt. Hierbei wird eine Verbindungslinie von dem Primärschlüssel einer Tabelle zum zu definierenden Sekundärschlüssel gezogen.

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.

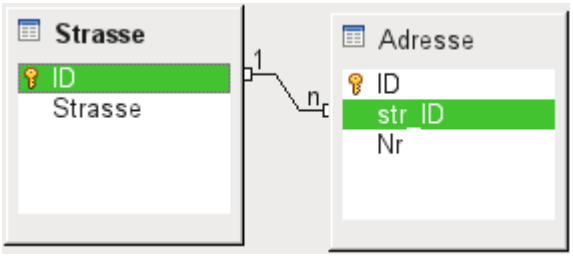
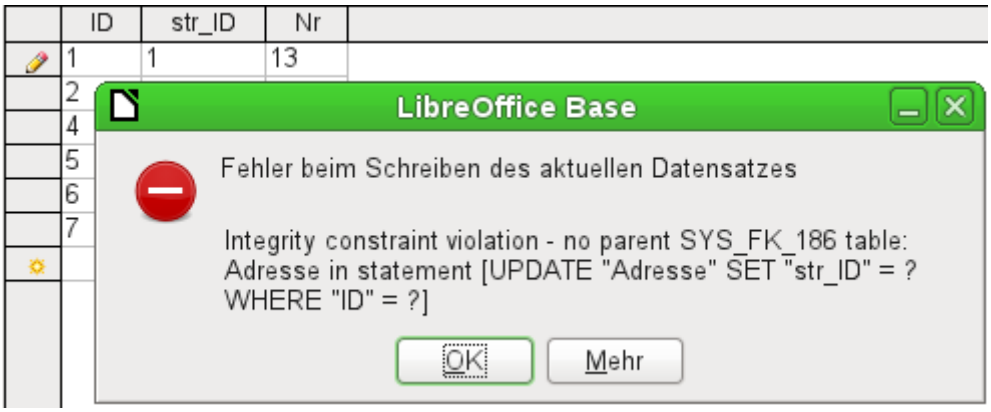


Abbildung 11: Erstellte Verbindung zwischen "Strasse" und "Adresse" mit "str_ID" als Fremdschlüssel.

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.

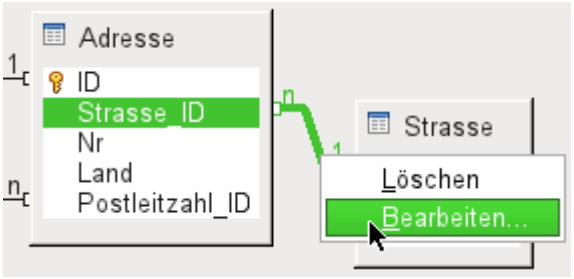


Abbildung 12: Durch Klick mit der rechten Maustaste können Verknüpfungen, hier am Beispiel der Mediendatenbank, bearbeitet werden.

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.

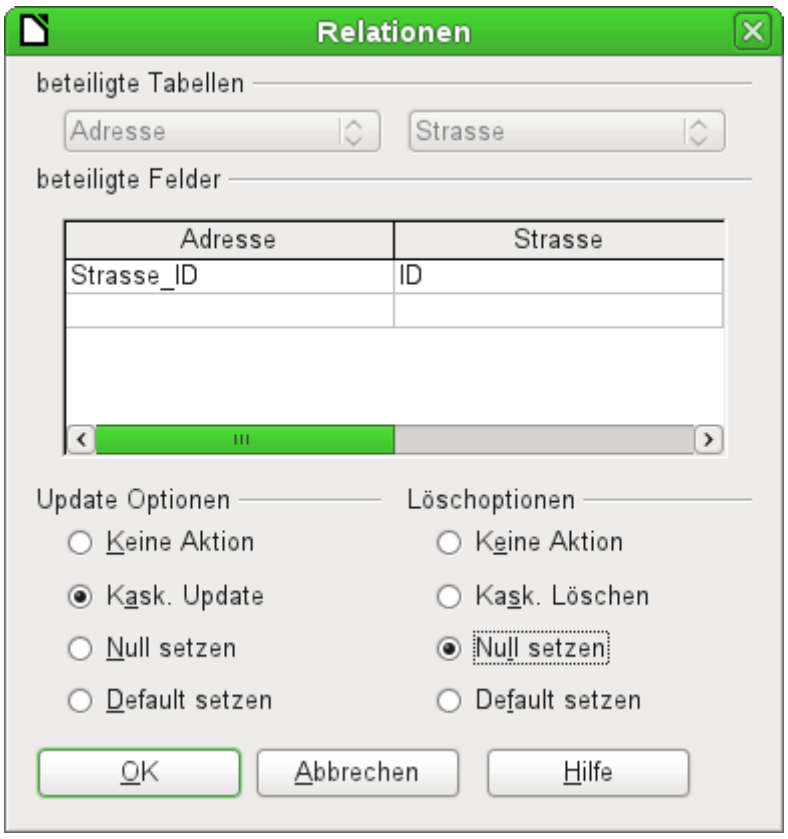


Abbildung 13: Dialog «Relationen»

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.

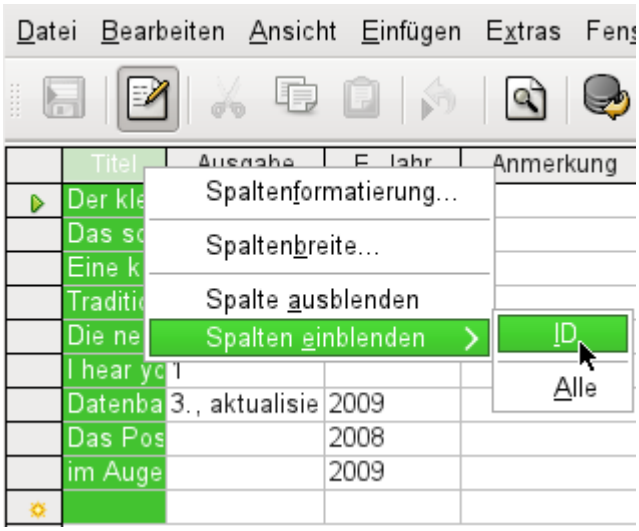


Abbildung 15: Eingabe in Tabellen - Spalten wieder einblenden.

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.

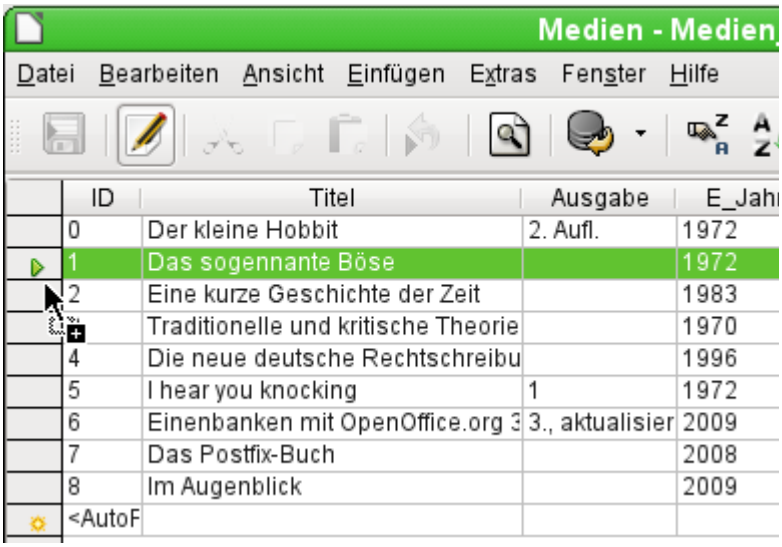
Nach Erreichen des letzten Feldes eines Datensatzes springt der Cursor automatisch in den nächsten Datensatz. Die vorhergehende Eingabe wurde dabei abgespeichert. Eine zusätzliche Abspeicherung unter **Datei → Speichern** ist bei der HSQLDB nicht nötig und nicht möglich. Die Daten sind bereits in der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

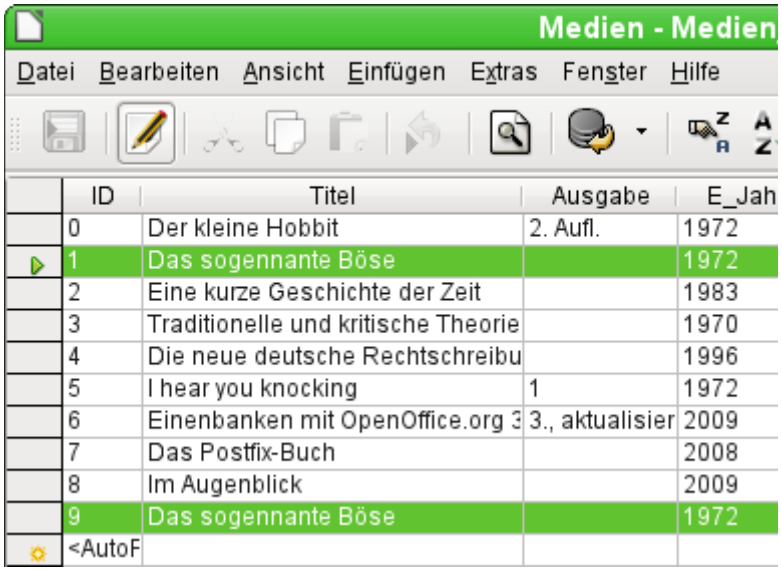
Die Änderung eines Datensatzes ist einfach möglich: Feld aufsuchen, geänderten Wert eingeben und Datenzeile wieder verlassen.

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

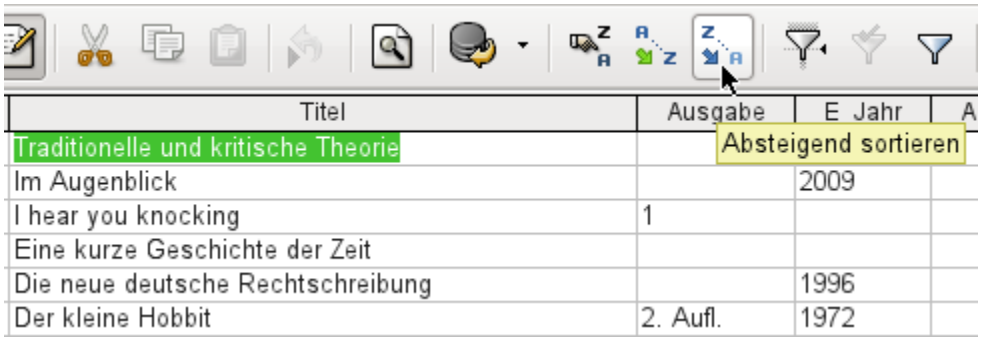


Abbildung 16: Schnelle Sortierung

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:

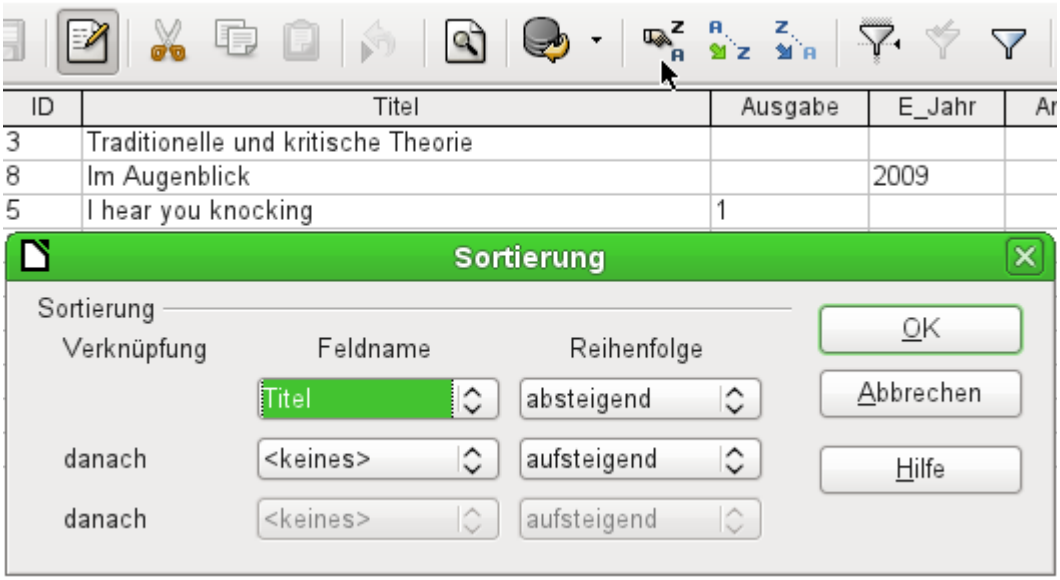
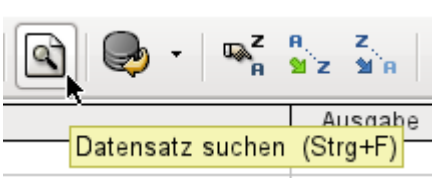


Abbildung 17: Sortierung nach mehreren Spalten

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.

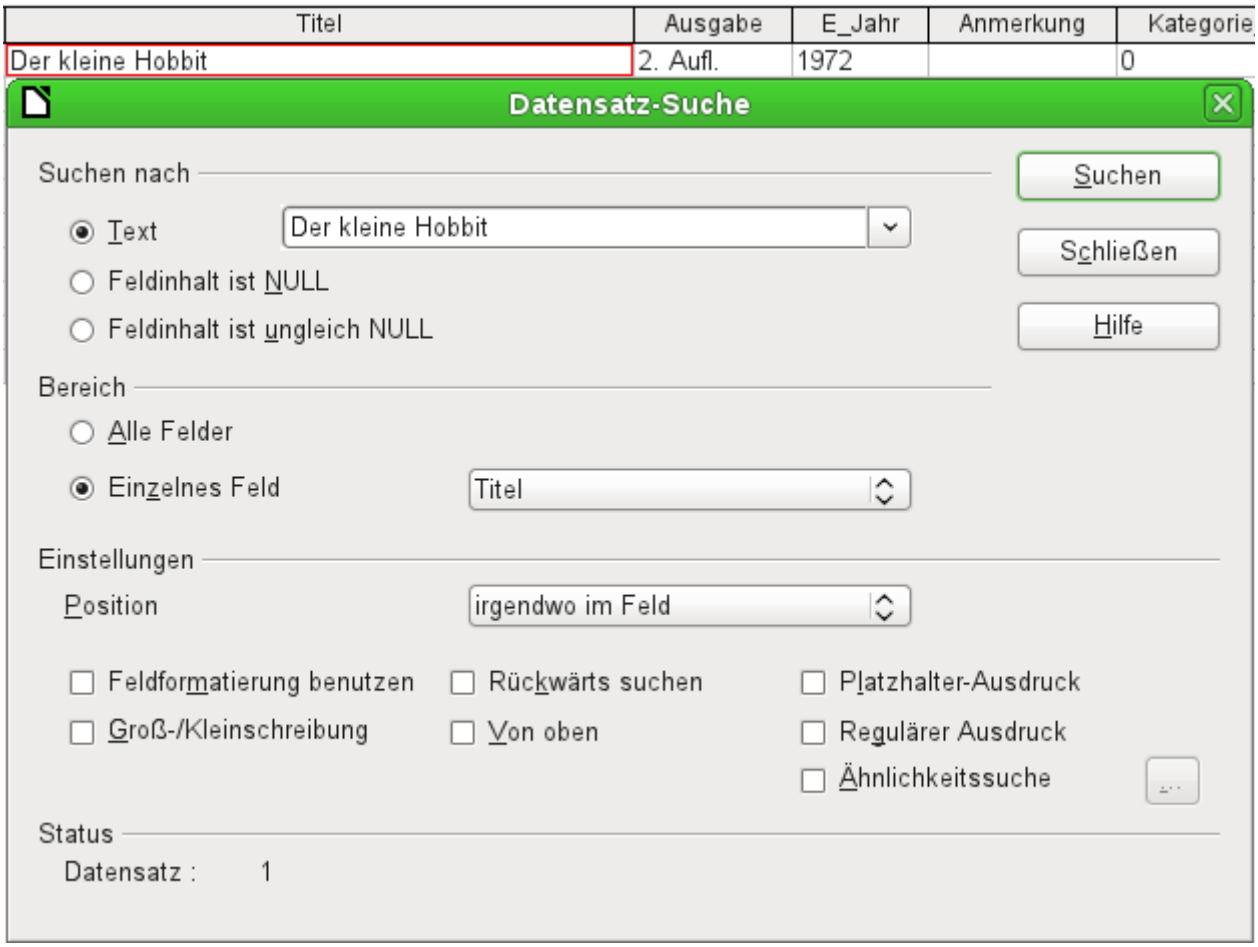


Abbildung 18: Eingabemaske zur Datensatzsuche

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.

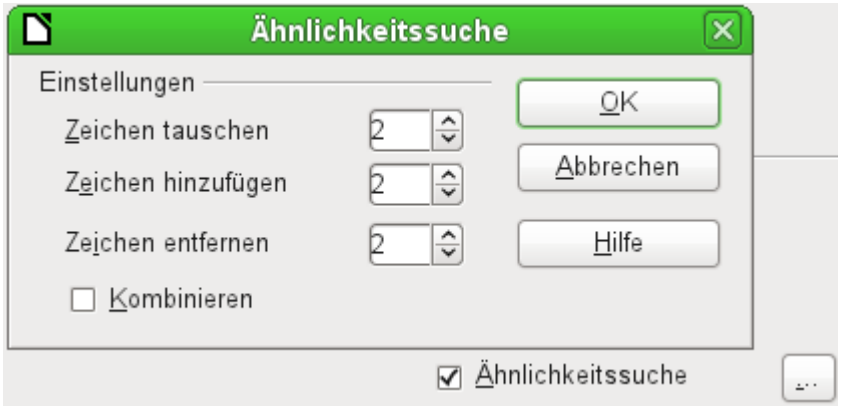


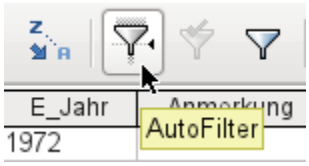
Abbildung 19: Eingrenzung der Ähnlichkeitssuche

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.

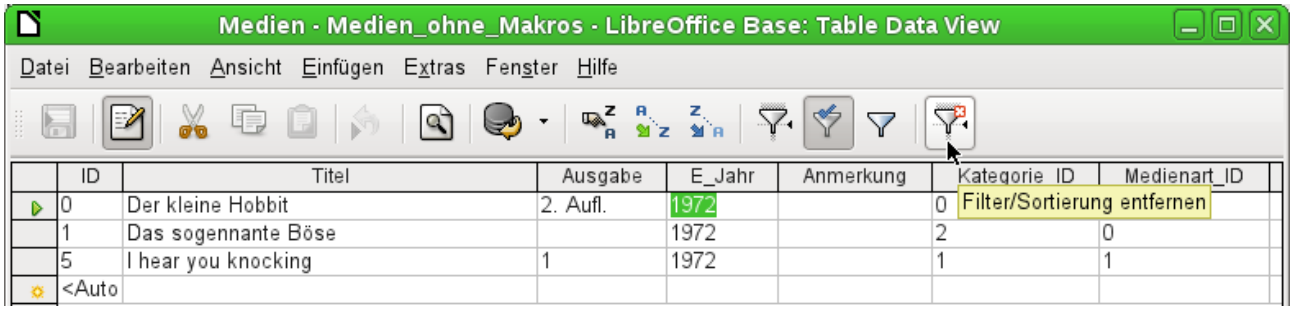
Insgesamt ist dieses Suchmodul eher etwas für Leute, die durch häufige Anwendung genau wissen, an welchen Stellen sie zum Erreichen eines Ziels drehen müssen. Für den Normaluser dürfte die Möglichkeit, Datensätze durch Filter zu finden, schneller zum Ziel führen.

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".

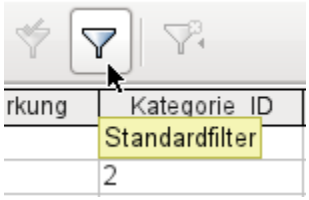


Der Filter ist aktiv. Dies ist an dem Filtersymbol mit einem grünen Haken zu erkennen. Das Filtersymbol erscheint gedrückt. Wird der Button erneut betätigt, so bleibt der Filter selbst erhalten, es werden aber wieder alle Datensätze angezeigt. So kann gegebenenfalls wieder in den Filterzustand zurückgeschaltet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

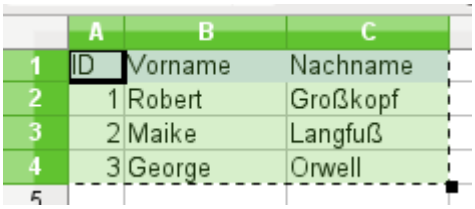
```
001 DELETE FROM "Tabellenname" WHERE "RueckgabeDatum" IS NOT NULL;
```

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

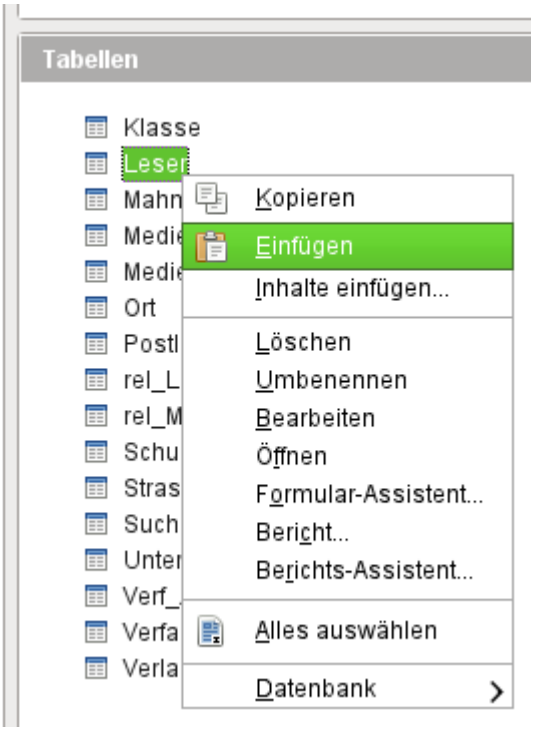
Um einen Import über die Zwischenablage zu gewährleisten, muss das Datenformat für Base lesbar sein. Dies ist bei allen Daten der Fall, die in LibreOffice geöffnet zur Verfügung stehen.
Sollen also z. B. Tabellen aus einer externen Datenbank in eine *.odb-Datei eingelesen werden, so muss zuerst zu der 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

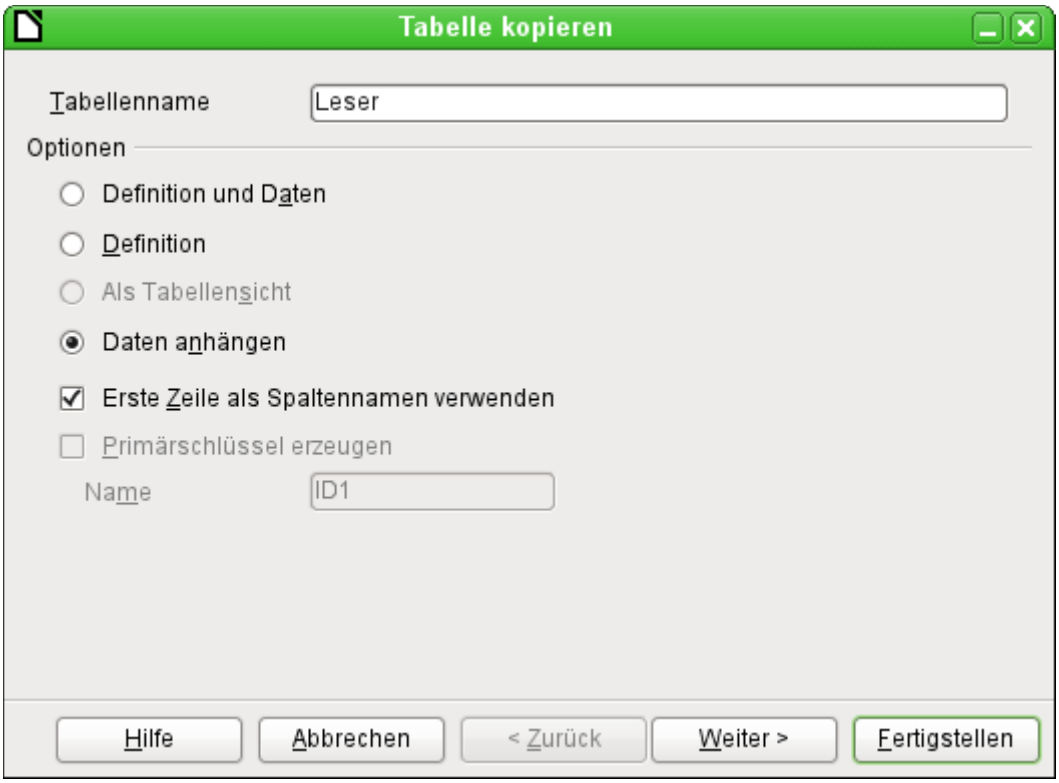
Eine kleine Beispieltabelle wird aus der Tabellenkalkulation «Calc» in die Zwischenablage kopiert.

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.

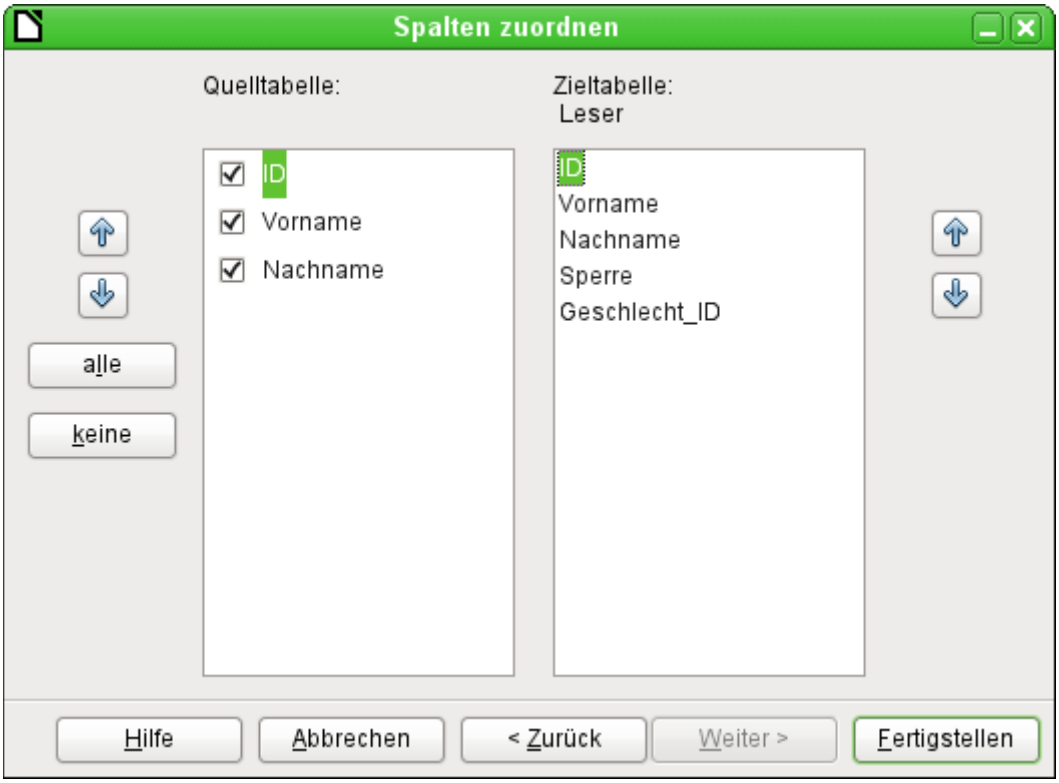


In dem Tabellencontainer wird mit der rechten Maustaste das Kontextmenü über der Tabelle aufgerufen, an die die Daten angehängt werden sollen und **Einfügen** gewählt

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.

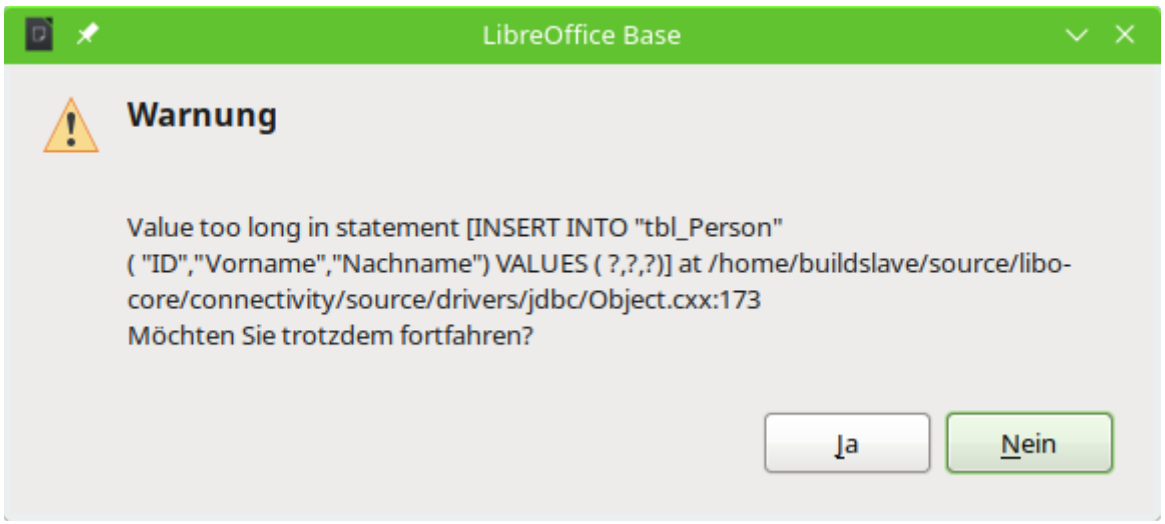


Die Spalten der Quelltabelle aus Calc und der Zieltabelle in Base müssen nicht von der Reihenfolge her, vom Namen her oder von der Anzahl her übereinstimmen. Es werden nur die auf der linken Seite ausgewählten Elemente übertragen. Die Zuordnung zwischen Quelltabelle und Zieltabelle muss mit den danebenliegenden 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.

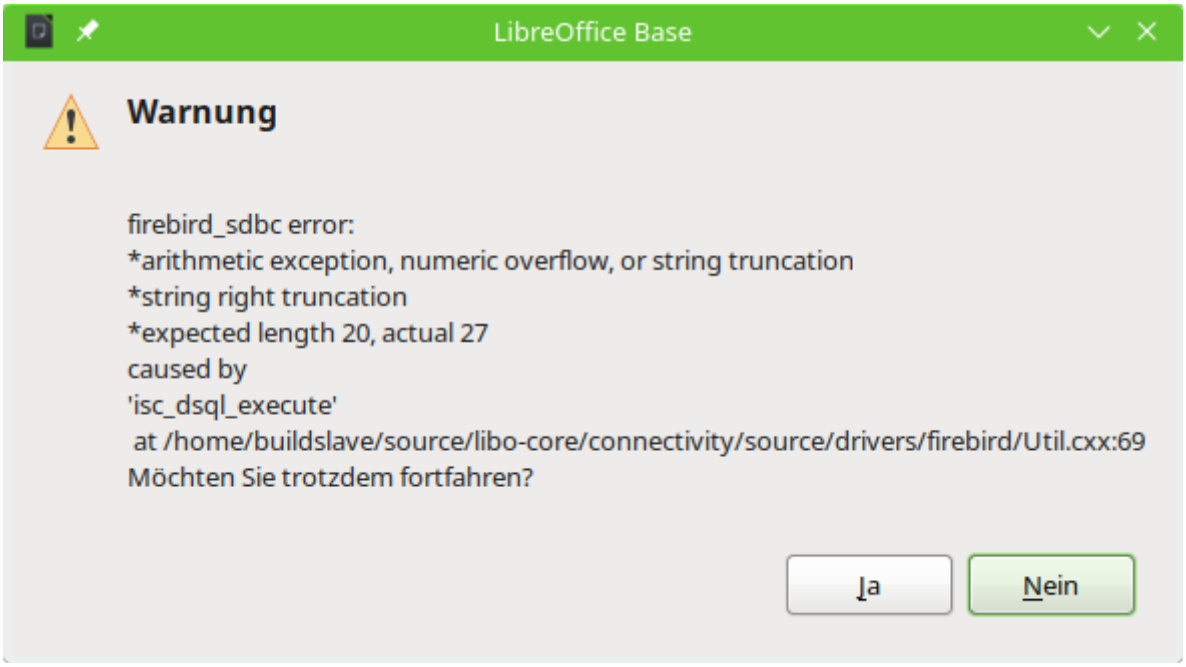


Fehlermeldung beim Import eines zu langen Wertes bei der internen HSQLDB

Diese Fehlermeldung bei der internen HSQLDB macht darauf aufmerksam, dass in einem Feld, das in die Tabelle "tbl_Person" eingefügt werden soll, der einzufügende Wert länger ist als er in der Definition der Felder der Tabelle vorgesehen wurde. Leider ist nicht ersichtlich, in welchem Feld die Länge nicht eingehalten wird. Im obigen Fall wurden die Felder für "Vorname" und "Nachname" auf VARCHAR(20) gesetzt. Die zu importierenden Daten hatten aber einen Nachnamen, der 27 Zeichen lang war.

Wird jetzt Nein gedrückt, so werden alle Datensätze bis zum Auftreten des Fehlers importiert, die nachfolgenden aber nicht. So kann der Datensatz mit dem zu langen Feld noch einigermaßen gut gefunden werden.

Wird stattdessen Ja gedrückt, dann wird der fehlerhafte Datensatz nicht importiert, die folgenden Datensätze aber sehr wohl. Es muss also anschließend nach dem Datensatz gesucht werden, der den Fehler verursacht. Außerdem muss auch noch gesucht werden, welches Feld denn den Fehler verursacht.



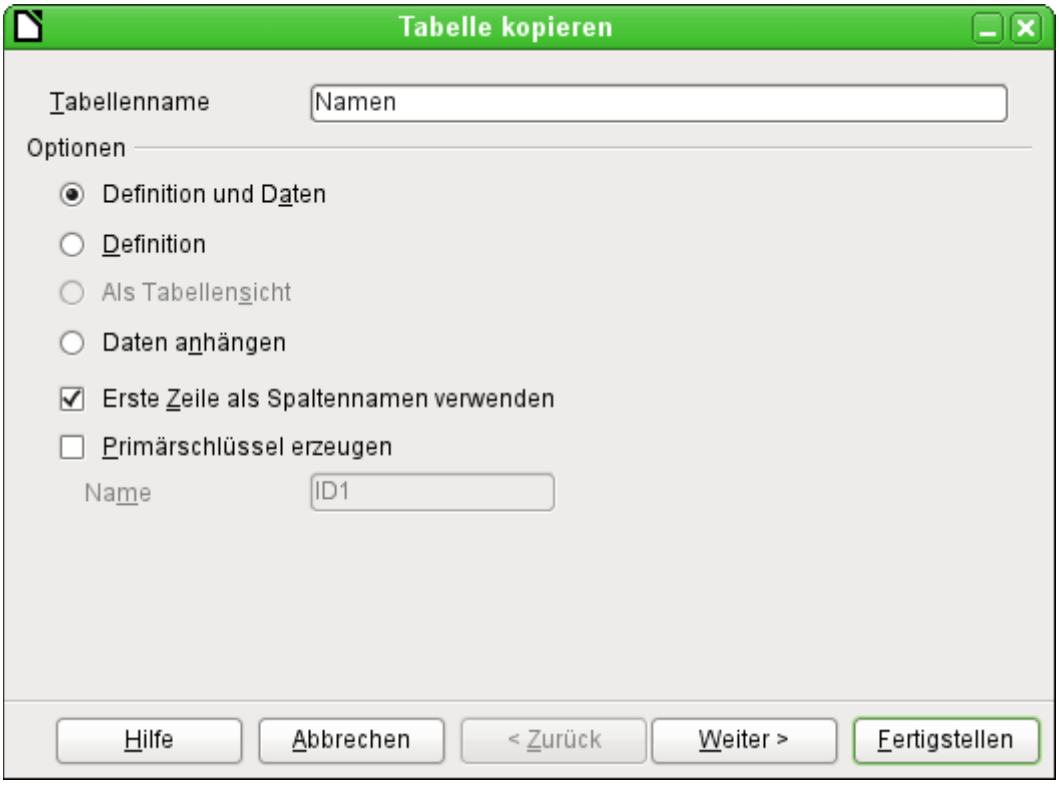
Fehlermeldung beim Import eines zu langen Wertes bei FIREBIRD

Die Meldung bei FIREBIRD zeigt lediglich an, dass irgendein Feld eben 27 Zeichen statt der erwarteten 20 Zeichen enthält. Das hilft zumindest schon einmal etwas bei der anschließenden Fehlersuche.

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.

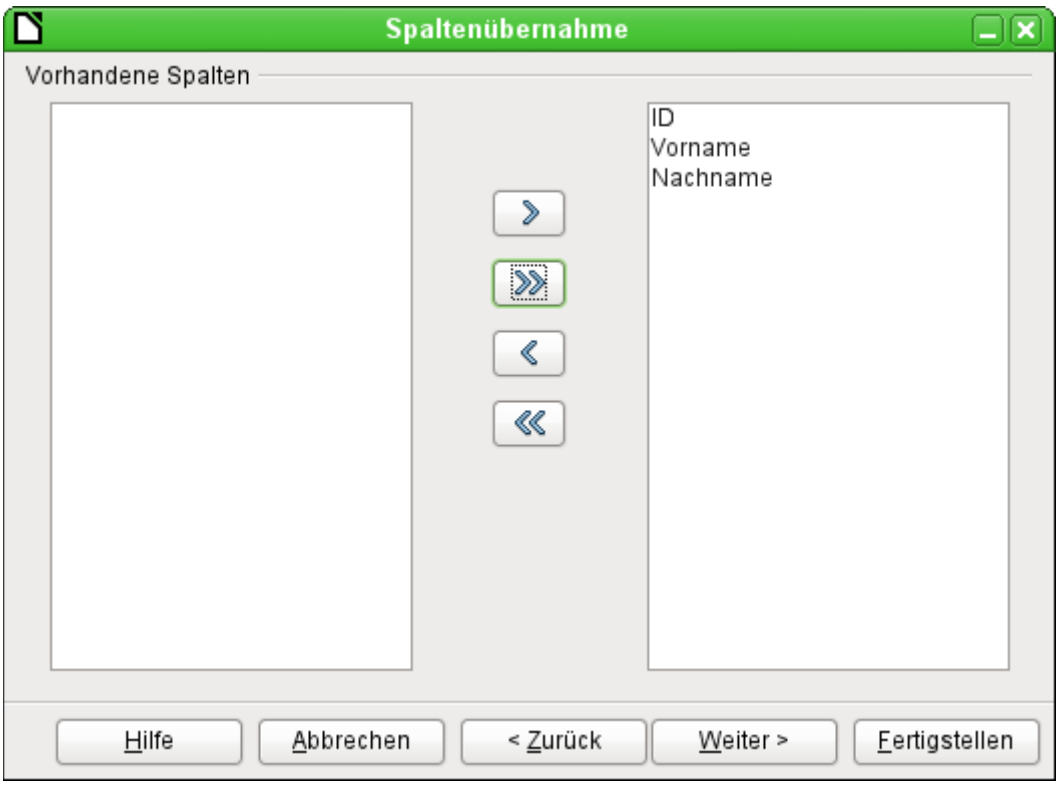
An dieser Stelle kann ein neues, zusätzliches Datenbankfeld für einen Primärschlüssel erzeugt werden. Der Name dieses Datenbankfeldes darf nicht schon als Spalte in der Calc-Tabelle existieren. Anderenfalls erscheint die Fehlermeldung:

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

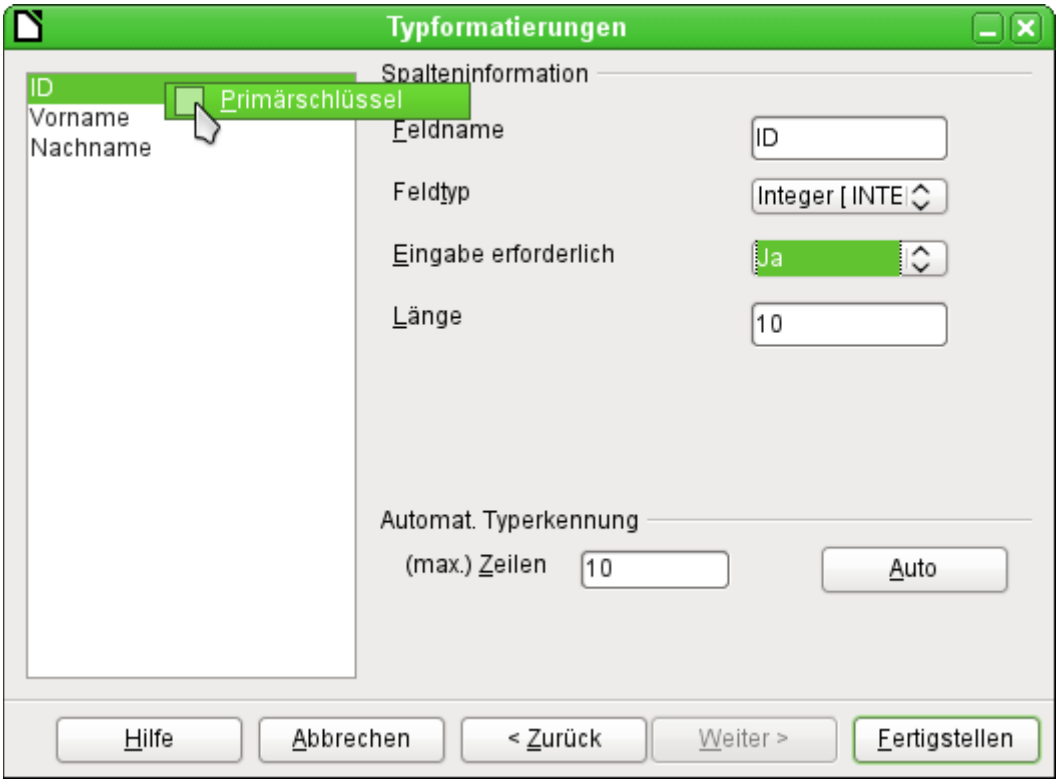
Diese Meldung gibt den Sachverhalt leider nicht ganz korrekt wieder.

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.

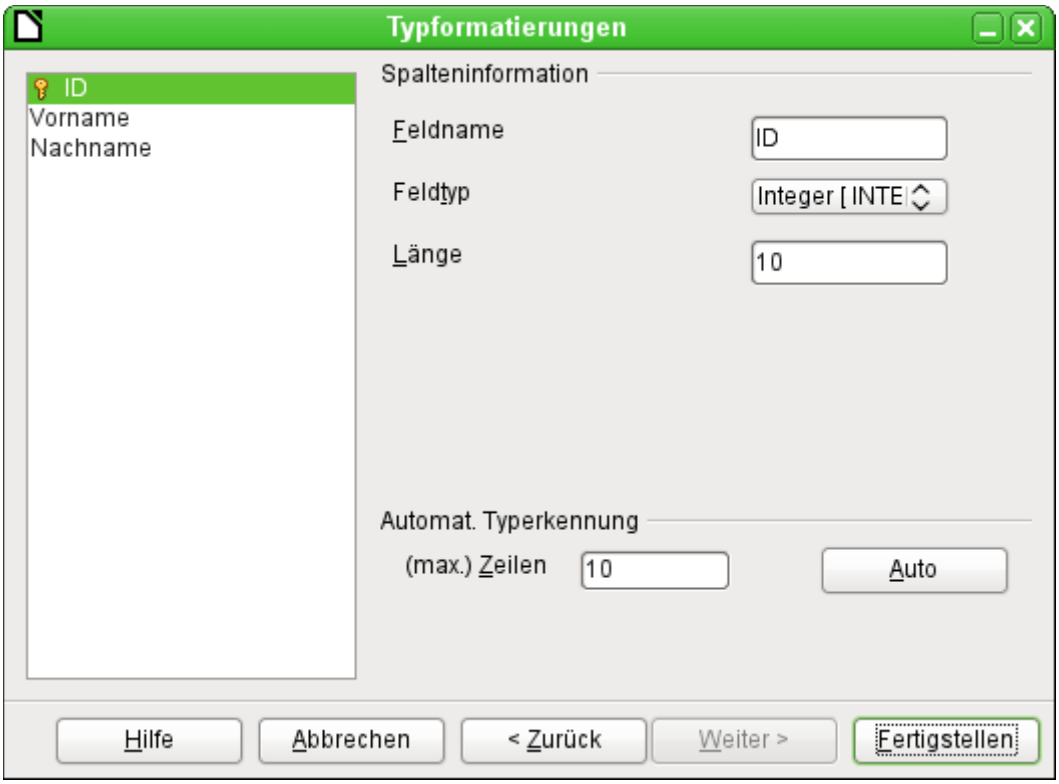


Alle vorhandenen Spalten werden übernommen.



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.

```
001 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		

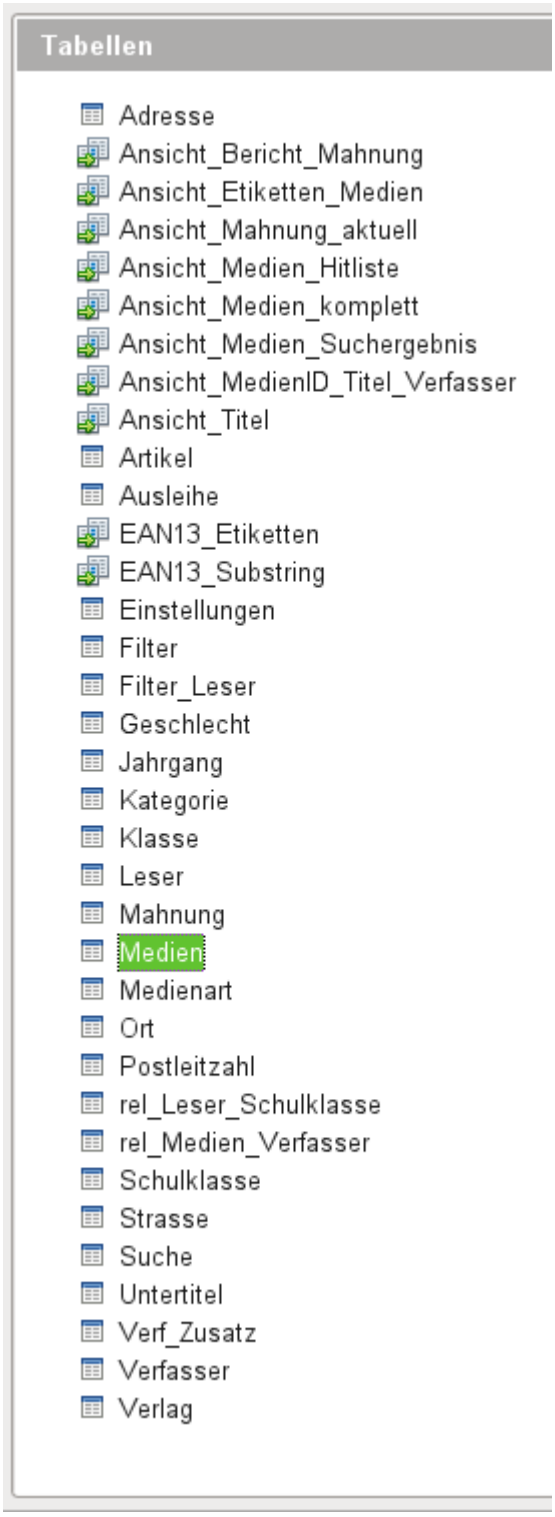
Die Ausleihtabelle besteht aus Fremdschlüsseln für das auszuleihende Medium "Medien_ID" und den entsprechenden Nutzer "Leser_ID" sowie einem Ausleihdatum "Leih_Datum". In die Tabelle werden also bei der Ausleihe zwei Zahlenwerte (Mediennummer und Benutzernummer) und ein Datum eingetragen. Der Primärschlüssel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

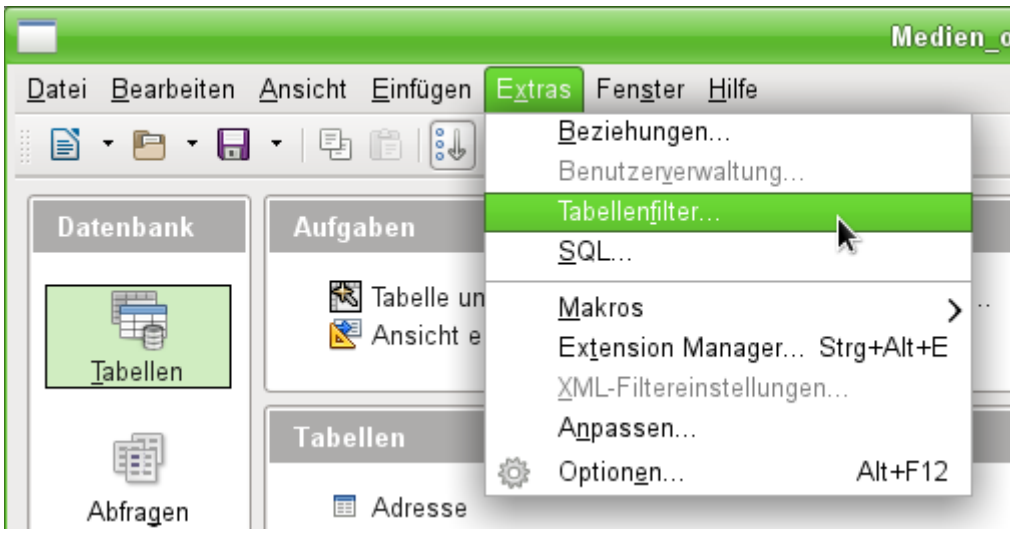
Die Eingabe in Tabellen ist in Datenbanken daher nur bei einfachen Tabellen sinnvoll. Sobald Tabellen in Relation zueinander gesetzt werden, bietet sich besser ein entsprechendes Formular an. In Formularen können diese Relationen durch Unterformulare oder Listenfelder gut bedienbar gemacht werden.

Tabellen verstecken

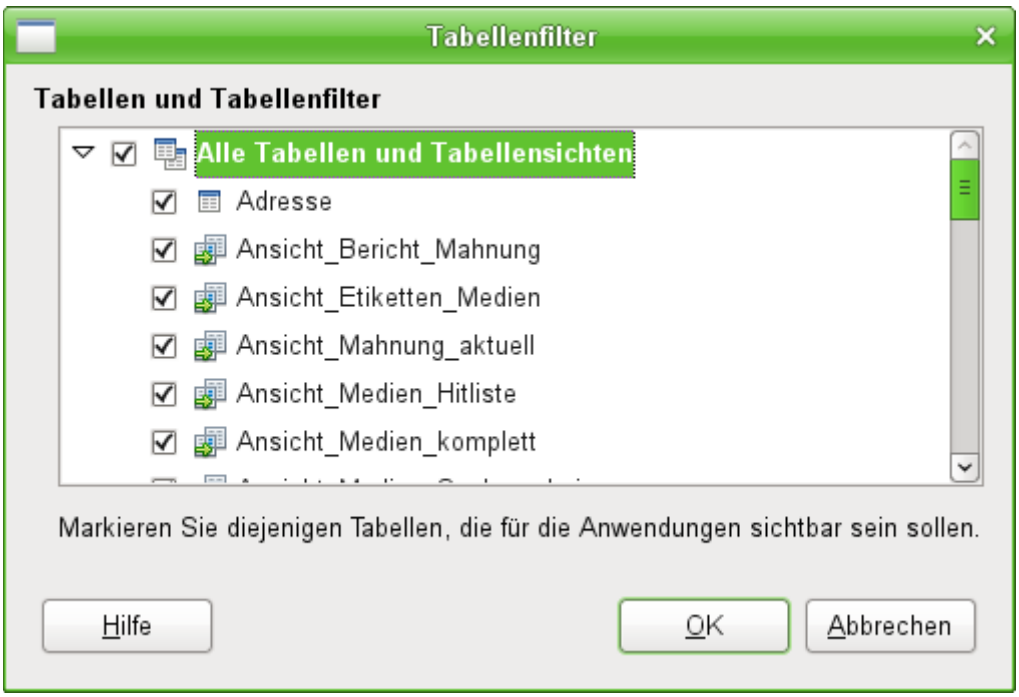
Bei einer entsprechend großen Datenbank erscheinen in der Tabellenübersicht sehr viele Tabellen und Tabellenansichten.



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.

Formulare

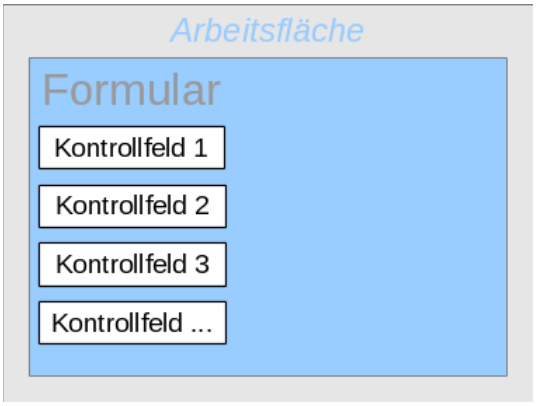
Formulare als Eingabeerleichterung

Formulare werden dann genutzt, wenn die Eingabe direkt über eine Tabelle zu unübersichtlich wird, eventuelle Fehleingaben rechtzeitig abgefangen werden sollen oder zu viele Tabellen eine direkte Verwaltung der Daten unmöglich machen.

Hinweis

Ein **Formular** ist in Base ein für den Nutzer **nicht sichtbares Konstrukt**. Es dient innerhalb von Base dazu, den Kontakt zur Datenbank zu ermöglichen.

Sichtbar sind für den Nutzer die **Kontrollfelder**, die dazu dienen, Text, Zahlen usw. einzugeben oder anzuzeigen. Diese Kontrollfelder werden über die GUI in verschiedene Feldarten unterteilt.



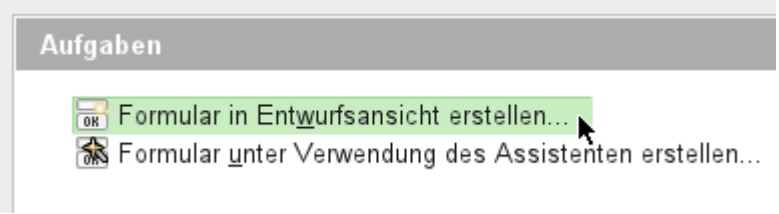
Der Begriff «Formular» hat eine doppelte Bedeutung.
Zum einen steht ein «Formular» für den gesamten Inhalt des Eingabefensters, in dem die Daten für eine oder mehrere Tabellen verwaltet werden. Hier handelt es sich genaugenommen um ein **Formulardokument**, das in der Datenbankdatei auch als komplette Writer-Datei abgespeichert wird.
Zum anderen enthält ein solches Formulardokument ein Fenster, das wiederum ein oder mehrere Formulare enthalten kann; auch für diese Teilbereiche des Fensters wird der Begriff «*Formular*» verwendet. Jedes dieser Formulare in dem Fenster des Formulardokumentes kann außerdem noch wieder Unterformulare enthalten.
Aus dem Zusammenhang sollte immer klar werden, welche Art von Formular gemeint ist, sodass es hoffentlich niemals zu Missverständnissen kommt.

Erstellung von Formularen

Der einfachste Weg zur Erstellung von Formularen ist der über den Formular-Assistenten. Mit diesem Assistenten lassen sich auch Formulare mit Unterformularen erstellen. Dieser Weg wurde bereits im Kapitel *Eingabeformular* im Kapitel «Einführung in Base» beschrieben.

Formulardokument in der Entwurfsansicht

Als Start dient uns aus dem Formularbereich die Aufgabe **Formular in Entwurfsansicht erstellen**.



Rufen wir damit den Formulareditor auf, so zeigt sich erst einmal das Fenster *Formulardokument in der Entwurfsansicht*.

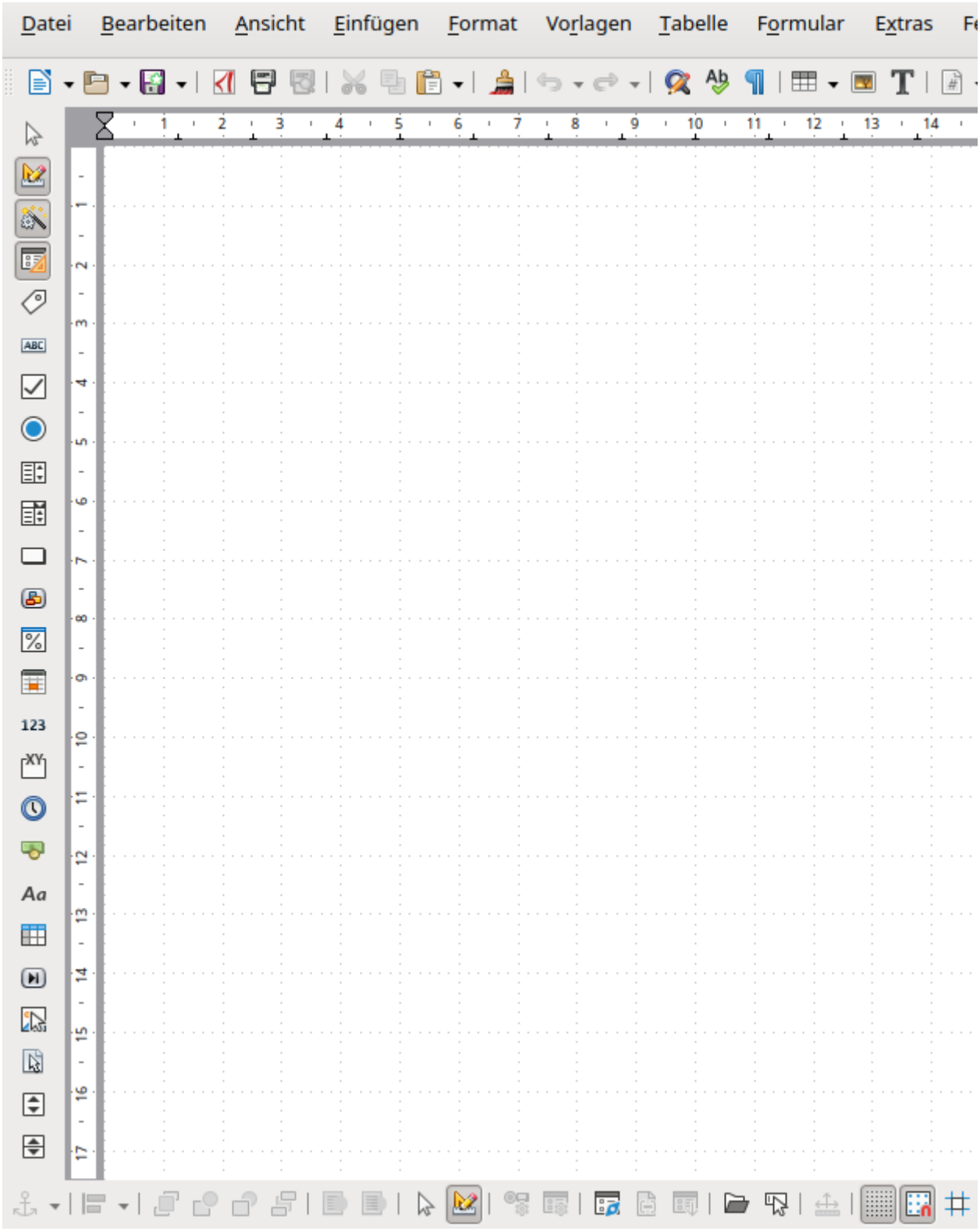


Abbildung 21: Formulardokument in der Entwurfsansicht

Am linken Rand ist die Symbolleiste «Formular-Steuerelemente» eingeblendet. Am unteren Rand ist die Symbolleiste «Formular-Entwurf» angedockt. Sollten diese Symbolleisten nicht automatisch erscheinen, so können sie über **Ansicht → Symbolleisten** angewählt werden. Ansonsten steht noch im Hauptmenü der Menüpunkt **Formular** zur Verfügung.

Die weiße Fläche weist ein gepunktetes Raster auf. Dies dient dazu, die Elemente möglichst genau positionieren zu können – vor allem im Verhältnis zueinander. Dass das Raster sichtbar

und eingeschaltet ist, ist an den Symbolen ganz rechts in der Leiste zum Formular-Entwurf zu erkennen.

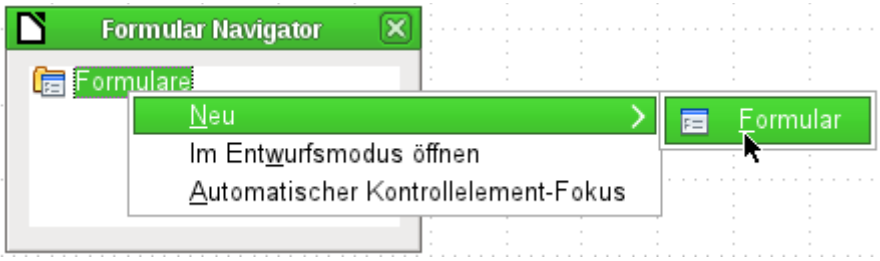
Abbildung 22: Alle verfügbaren Schaltflächen der Symbolleiste «Formular-Entwurf»

Auf der leeren Fläche soll nun ein Formular entstehen. Dies kann auf verschiedene Arten geschehen:

- Aufruf des Formular-Navigators, von dort Gründung eines Formulars sowie
- Erstellung von Formularfeldern und Gründung des Formulars über das dortige Kontextmenü.

Formulargründung über den Navigator

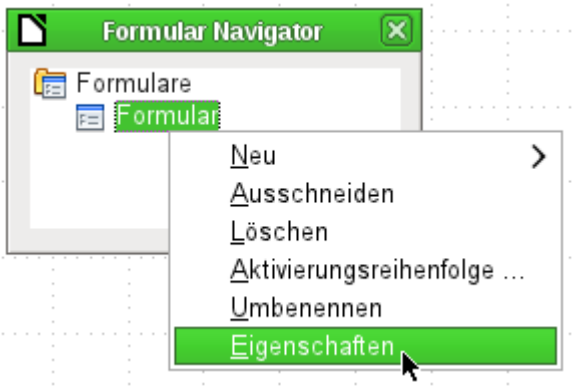
Mit dem in *Abbildung 22* abgebildeten Button Formular-Navigator wird der Navigator gestartet. Es erscheint ein Fenster, das auf lediglich ein Verzeichnis hinweist. Dies ist die höchste Ebene der Fläche, die jetzt bearbeitet wird. Sie ist mit **Formulare** benannt. Dies weist darauf hin, dass nicht nur ein, sondern ohne weiteres mehrere Formulare auf der angezeigten Fläche untergebracht werden können.



Mit einem Rechtsklick auf das Verzeichnis Formulare öffnet sich ein Kontextmenü, in dem über den Menüpunkt **Neu** ein neues **Formular** erstellt werden kann. Die weiteren Befehle des Kontextmenüs entsprechen denen der Buttons in der *Abbildung 22*.

Hinweis

Soll ein Formular automatisch mit dem Cursor im ersten Formularelement starten, so kann dies durch die Auswahl **Automatischer Kontrollelement-Fokus** bewirkt werden. Das erste Formularelement wird über die *Aktivierungsreihenfolge* geregelt.
Barrierefreiheit: Diese Einstellung ist für die Bedienung durch die Tastatur sehr wichtig. Sonst können Sehbehinderte z.B. nicht von einem Formularfeld zum nächsten navigieren. Zusätzlich sollte jedes Formularfeld mit einer Beschreibung (rechter Mausklick über dem Feld) für den Screenreader versehen werden.



Das Formular erhält standardmäßig den Namen **Formular**. Diese Bezeichnung kann direkt oder später geändert werden. Sie hat allerdings für die spätere Funktion nur dann eine Bedeutung, wenn über Makros auf Teile des Formulars zugegriffen werden soll. Spätestens dann sollten nicht zwei Elemente mit gleicher Bezeichnung in der gleichen Ebene des Verzeichnisbaums auftauchen.

Über das Kontextmenü des Formulars geht schließlich der Weg zu den Formulareigenschaften.

Formulargründung über ein Formularfeld

Über die Symbolleiste für die Formularsteuerelemente (*Abbildung 23*) stehen direkt alle Formularfelder zur Verfügung. Während die ersten vier Elemente identisch zu den entsprechenden Elementen des Formular-Entwurfs sind, folgen anschließend sämtliche Formularfelder.

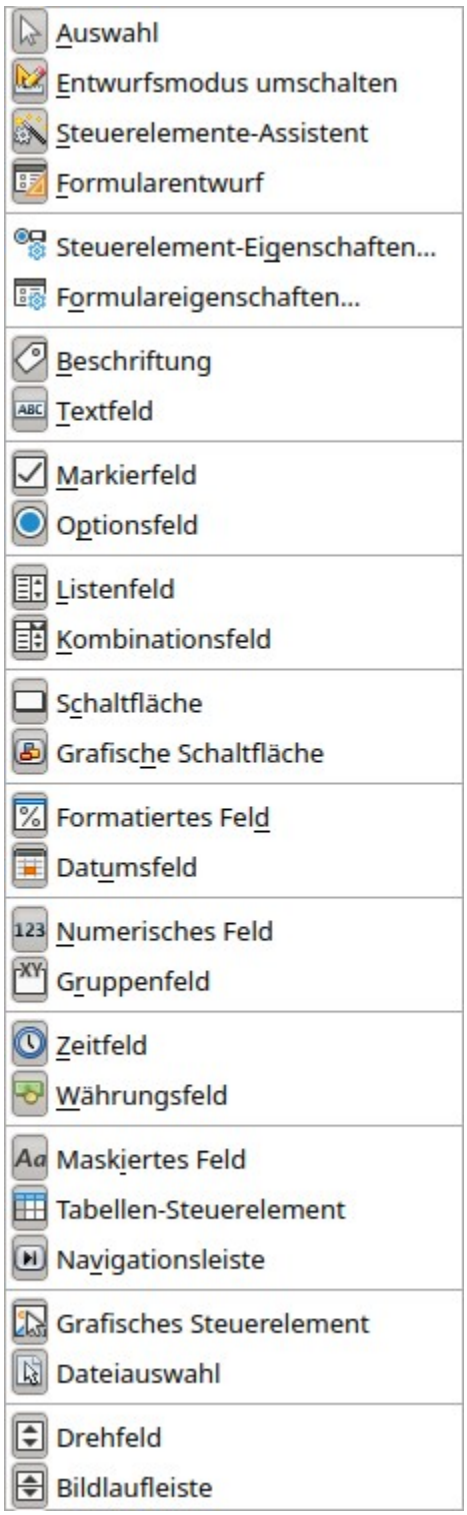
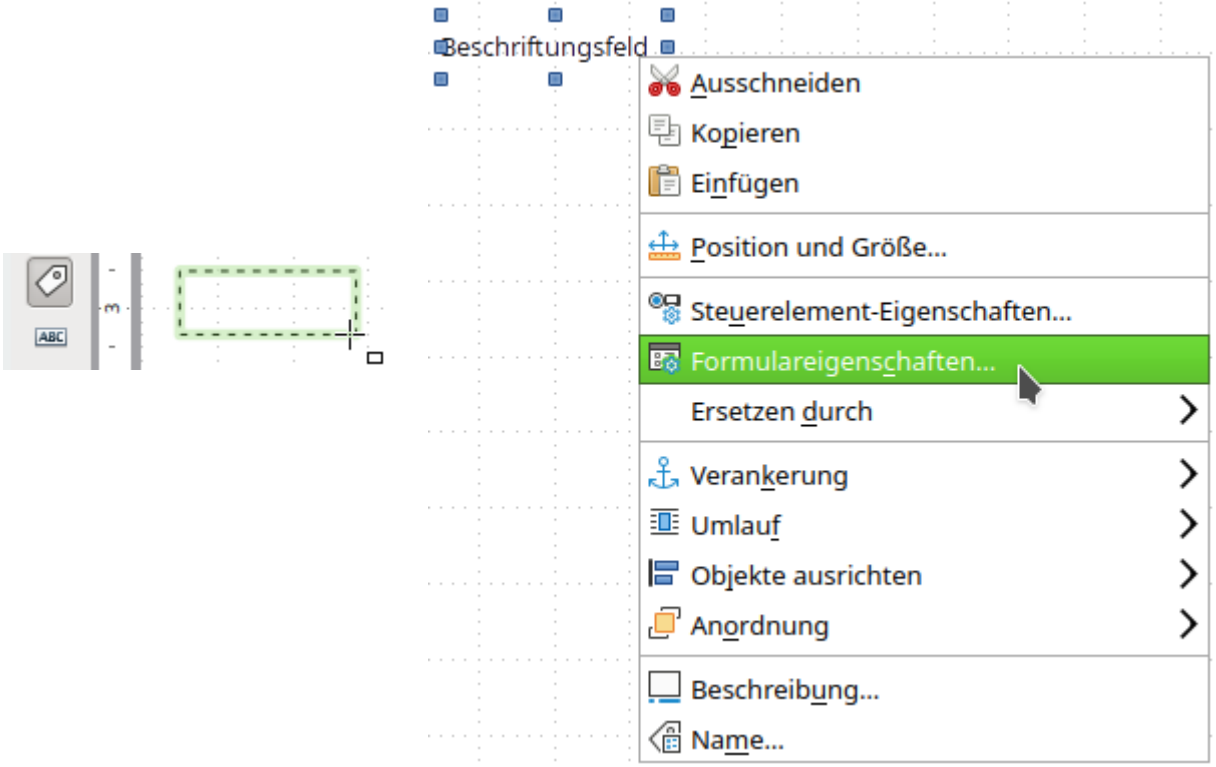


Abbildung 23: Alle verfügbaren Schaltflächen der Symbolleiste «Formular-Steuerelemente»

Hinweis

Der Inhalt der Symbolleiste «Formular-Steuerelemente» ist mit der Zeit geändert worden. Vor der Version LO 6.0 gab es eine Symbolleiste «Weitere Steuerelemente», die aus der Symbolleiste «Formular-Steuerelemente» als frei schwebende Leiste angezeigt wurde. Zur Version LO 6.3 schließlich gibt es die Symbolleiste «Weitere Steuerelemente» nicht mehr.
Neben der Symbolleiste stehen alle Steuerelemente über den neuen Menüeintrag **Formular** zur Verfügung.

Über den Aufruf eines Formularfeldes wird automatisch ein Formular mit gegründet:



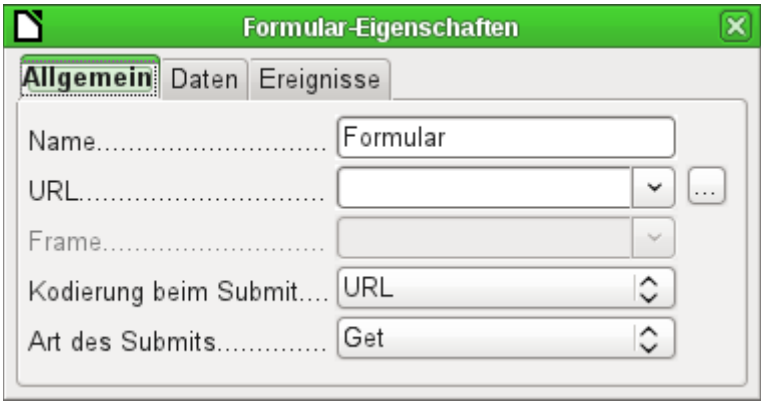
Dazu wird z. B. ein Beschriftungsfeld aufgerufen. Der Cursor verändert sein Erscheinungsbild. Es kann eine rechteckige Form auf die weiße Oberfläche des Formulars gezogen werden. Aus der gestrichelten Form entsteht anschließend ein Beschriftungsfeld. Jetzt wird zur Erstellung des Formulars das **Kontextmenü** des Kontrollfeldes aufgerufen.

Über den Menüpunkt **Formular** werden hier die Eigenschaften des nebenher gegründeten Formulars aufgerufen. Das Formular wurde mit dem Standardnamen «Formular» erstellt.

Externe Formulare

Neben den Formularen, die direkt in Base erstellt werden, gibt es auch die Möglichkeit, im Writer oder in Calc Formulare zu erstellen. Auch Formulare, die in Base erstellt wurden, lassen sich zu externen Formularen umwandeln. Dies wird im Kapitel «Datenbank-Anbindung» unter *Externe Formulare* beschrieben.

Formular-Eigenschaften

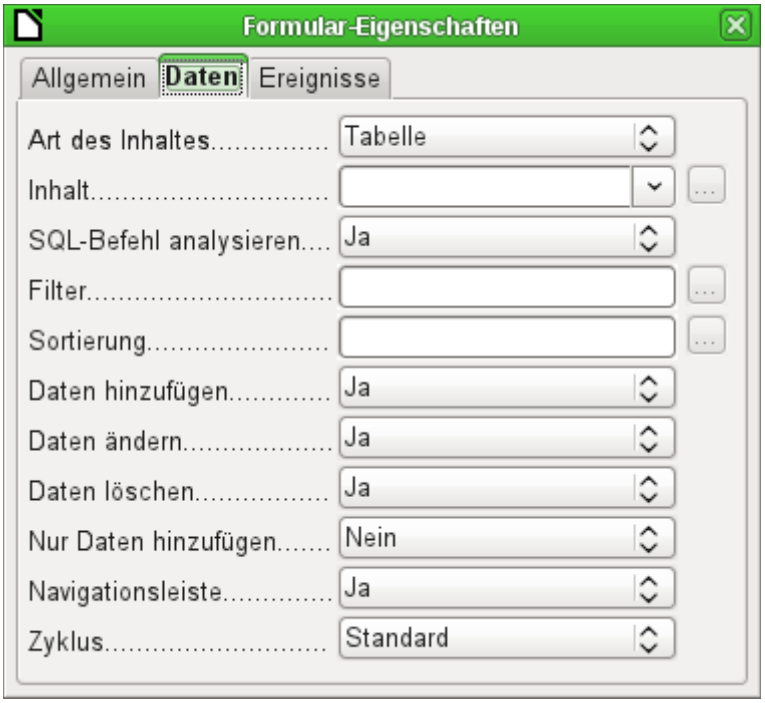


Werden die Formulareigenschaften über das Kontextmenü im Formular-Navigator oder das Kontextmenü eines Formularfeldes aufgerufen, so erscheint ein Fenster mit drei Registerreitern: **Allgemein**, **Daten** und **Ereignisse**

Allgemein

Hier kann der **Name** des Formulars geändert werden. Außerdem finden sich Einstellungsmöglichkeiten, die innerhalb von Base keine weitere Bedeutung haben. Sie zeigen lediglich die universelle Einsatzmöglichkeit des Formulareditors. So können z. B. Daten der Datenbank an ein Webformular zur weiteren Verarbeitung geschickt werden. (**URL**: Ziel, an das die Daten gesandt werden sollen | **Frame**: Teil der Zielwebseite, der eventuell separat angesprochen werden muss | **Kodierung beim Submit**: Neben dem normalen Kodieren der Zeichen für eine Weitergabe über die URL sind hier Textkodierungen und Multipartkodierungen (z. B. zum Versenden von Dateien ...) möglich. | **Art des Submits**: 'Get' (über die URL sichtbar an den Dateinamen angehängt) oder 'POST' (nicht sichtbar, auch für größere Datenmengen geeignet).

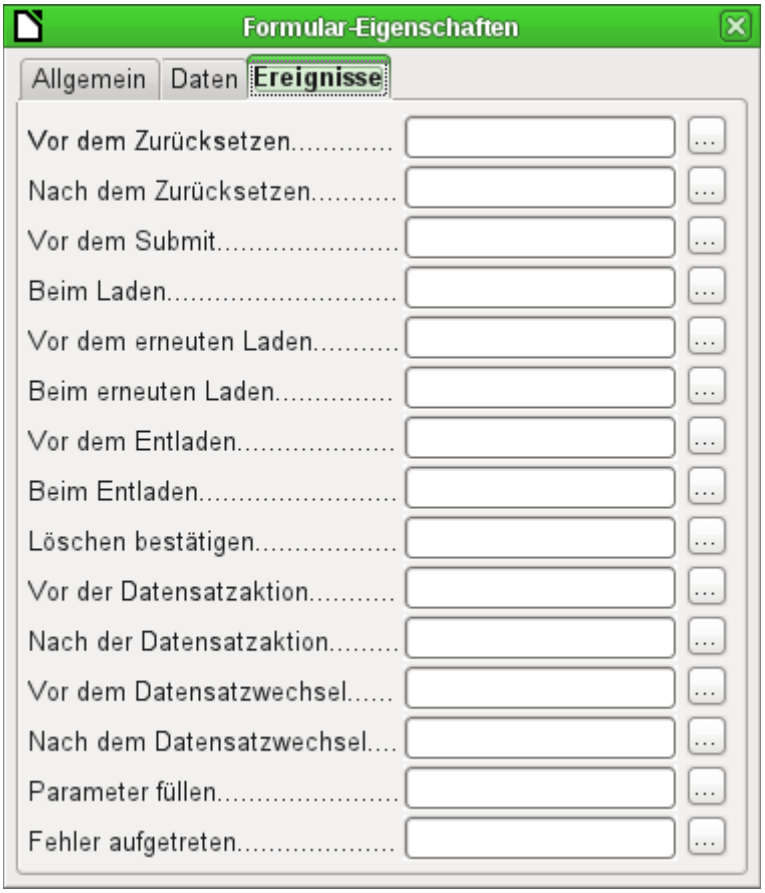
Daten



Für die Erstellung interner Formulare von Base ist dies der wichtigste Registerreiter. Hier wird zuerst der Inhalt des Formulars festgelegt.

- **Art des Inhalts**: Hier besteht die Wahl zwischen 'Tabelle', 'Abfrage' und 'SQL-Befehl'. Während Tabellen in der Regel immer für Eingaben in einem Formular genutzt werden können, so ist dies bei Abfragen eventuell nicht der Fall. Näheres dazu im Kapitel «Abfragen». Gleiches wie für Abfragen gilt auch für die direkten Eingabe eines SQL-Befehls. Hier handelt es sich dann lediglich um eine Abfrage, die nicht im Abfragecontainer von Base sichtbar ist, aber vom Prinzip her die gleiche Struktur aufweist.
- **Inhalt**: Wird unter Art des Inhaltes Tabelle oder Abfrage gewählt, so werden hier alle *verfügbaren Tabellen und Abfragen* gelistet. Soll ein SQL-Befehl erstellt werden, so besteht die Möglichkeit, den Abfrageeditor dazu über den Button mit den drei Punkten rechts von dem Inhaltsfeld aufzurufen.
- **SQL-Befehl analysieren**: Wird die Analyse des SQL-Befehls nicht zugelassen (weil z. B. mit Code gearbeitet wird, den die GUI eventuell nicht richtig deuten kann), so ist hier 'Nein' zu wählen. Allerdings schließt dies aus, dass das Formular weiterhin noch mit der Filterung oder mit der Sortierung auf die zugrundeliegenden Daten zugreifen kann.
- **Filter**: Hier kann ein Filter gesetzt werden. Hilfe dazu bietet ein Klick auf den Button rechts von dem Eingabefeld. Das Filtern entspricht dem *Filtern von Tabellen*.
- **Sortierung**: Hier kann eine Sortierung der Daten festgelegt werden. Hilfe dazu bietet ein Klick auf den Button rechts von dem Eingabefeld. Das Sortieren entspricht dem *Sortieren von Tabellen*.
- **Daten hinzufügen**: Sollen neue Daten erstellt werden können? Standardmäßig ist dies auf 'Ja' eingestellt.
- **Daten ändern**: Sollen Daten geändert werden können? Ebenfalls Standard 'Ja'.
- **Daten löschen**: Auch das Löschen von Daten wird standardmäßig ermöglicht.
- **Nur Daten hinzufügen**: Ist dies gewählt, so erscheint immer ein leeres Formular. Auf die alten Datensätze besteht kein Zugriff, sie können nicht bearbeitet oder auch nur angesehen werden. [IgnoreResult]
- **Navigationsleiste**: Das Erscheinen der Navigationsleiste am unteren Bildschirmrand kann angeschaltet oder ausgeschaltet werden. Außerdem besteht die Möglichkeit, bei einem Unterformular immer die Navigationsleiste des darüber liegenden Hauptformulars anzeigen zu lassen, so dass eine Betätigung der Navigationsleiste direkte Auswirkung auf das Hauptformular hat.
  Diese Einstellung zur Navigationsleiste betrifft nicht die Leiste, die gegebenenfalls als Formularfeld eingefügt werden kann.
- **Zyklus**: 'Standard' bedeutet hier für Base-Datenbanken, dass nach der Eingabe im letzten Feld innerhalb eines Formulars mit dem Tabulator zum ersten Feld des nächsten Datensatzes, also gegebenenfalls eines neuen Datensatzes, gesprungen wird. Dies ist für die Datenbanken gleichbedeutend mit 'Alle Datensätze'. Wird hingegen bei Datenbanken 'Aktueller Datensatz' gewählt, so bewegt sich der Cursor nur innerhalb des einen Datensatzes, beim Erreichen des letzten Feldes also zum ersten Feld des gleichen Datensatzes.
  'Aktuelle Seite' bezieht sich wieder besonders auf HTML-Formulare. Hier springt dann der Cursor vom Ende eines Formulars gegebenenfalls zum nächsten Formular auf der Seite, das weiter unten liegt.

Ereignisse



Ereignisse können Makros auslösen. Durch einen Klick auf den rechts stehenden Button … können Makros mit dem Ereignis verbunden werden.

Zurücksetzen: Das Formular wird von allen neuen Einträgen geleert, die noch nicht abgespeichert sind.

Vor dem Submit: Bevor die Formulardaten gesendet werden. Dieses Ereignis tritt nur dann auf, wenn eine URL für die Weitergabe der Daten in **Formular-Eigenschaften → Allgemein** angegeben wurde. Es wird durch eine Schaltfläche ausgelöst, bei der die **Aktion → Formular übertragen** gewählt wurde.

Beim Laden: Nur beim Öffnen des Formulars. Nicht beim Laden eines neuen Datensatzes in das Formular.

Erneutes Laden: Dies erfolgt, wenn der Inhalt des Formulars z. B. über einen Button in der Navigationsleiste aktualisiert wird.

Entladen: Nach einigen Tests scheint dies ohne Funktion zu sein. Erwartet würde der Ablauf eines Makros beim Schließen des Formulars.

Datensatzaktion: Dies ist z. B. das Abspeichern mittels Button. Im Test ergibt sich, dass die Aktion **Vor der Datensatzaktion** regelmäßig doppelt erscheint, d. h. Makros werden direkt nacheinander zweimal abgearbeitet. Dies liegt daran, dass hier unterschiedliche Funktionen (*Implementationen*) ausgeführt werden. Beide sind mit Namen versehen: **`org.openoffice.comp.svx.FormController`** und **`com.sun.star.comp.forms.ODatabaseForm`**. Wenn innerhalb des Makros mit **`oForm.ImplementationName`** der entsprechende Name abgefragt wird, so kann das Makro auf eine Ausführung begrenzt werden.

Datensatzwechsel: Bereits das Öffnen des Formulars stellt einen Datensatzwechsel dar. Beim Wechsel von einem Datensatz zum anderen innerhalb eines Formulars taucht diese

Aktion ebenfalls zweimal auf. Makros werden also auch hier zweimal hintereinander ausgeführt. Auch hier kann aber zwischen den Ursachen des Ereignisses unterschieden werden.

Parameter füllen: Dieses Makro springt ein, wenn eine Parameterabfrage in einem Unterformular aufgerufen werden soll, aber aus irgendeinem Grund der Parameter vom Hauptformular nicht richtig weiter gegeben wird. Ohne das Ereignis abzufangen, erfolgt dann beim Laden des Formulars eine Parameterabfrage.

Fehler aufgetreten: Dieses Ereignis lässt sich nicht nachvollziehen.

Eigenschaften der Kontrollfelder

Ist ein Formular erstellt, so kann es mit den sichtbaren Kontrollfeldern bestückt werden. Die Kontrollfelder sind für verschiedene Aufgaben gedacht:

- Die meisten zeigen den Inhalt aus der Datenbank an oder nehmen die Daten entgegen, die in die Datenbank eingefügt werden.
- Andere Kontrollfelder dienen zur Navigation, zum Suchen und zur Ausführung von Befehlen (Interaktion).
- Weitere Kontrollfelder sorgen für eine zusätzliche grafische Aufarbeitung des Formulars.

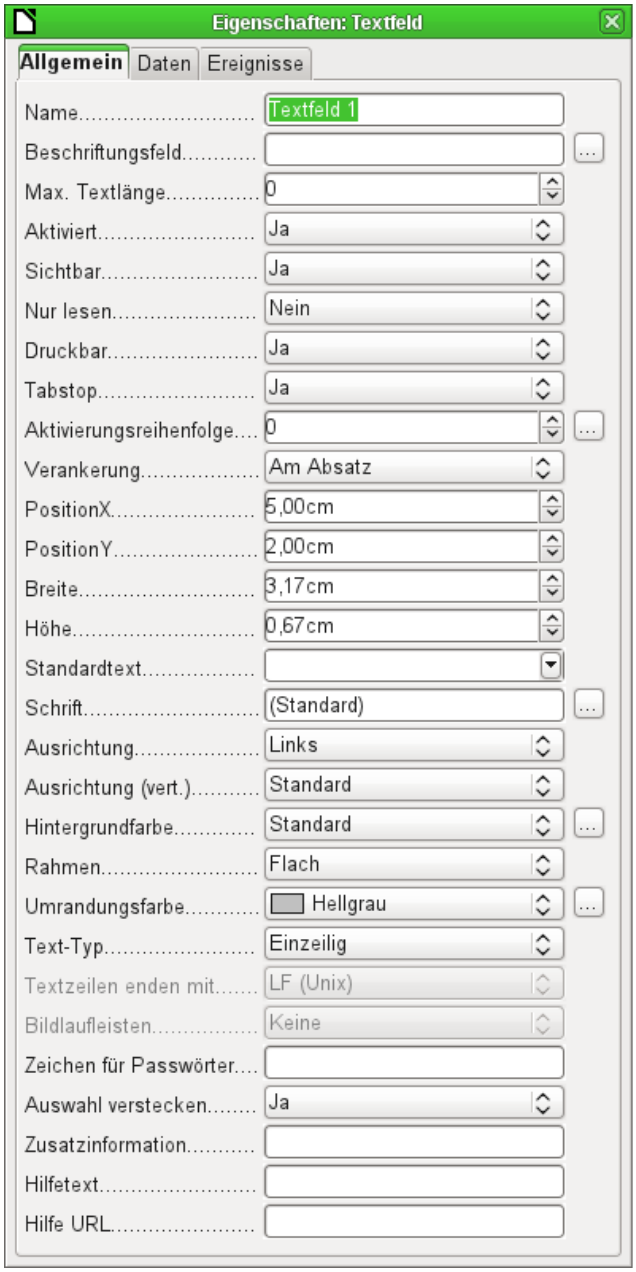
Dateneingabe und Datenanzeige	
Kontrollfeld	***Anwendungsgebiet***
Textfeld	Texteingaben
Numerisches Feld	Zahleneingabe
Datumsfeld	Datumseingabe
Zeitfeld	Zeiteingabe
Währungsfeld	Zahleneingabe, vorformatiert für Währungen
Formatiertes Feld	Anzeige und Eingabe mit zusätzlicher Formatierung wie z. B. Maßeinheiten
Listenfeld	Auswahl zwischen vielen verschiedenen Möglichkeiten, Weitergabe eines anderen als des angezeigten Wertes an die Datenbank.
Kombinationsfeld	Wie Listenfeld, nur Weitergabe des angezeigten Wertes und dazu noch die Möglichkeit, auch neue Werte einzugeben.
Markierfeld	Ja/Nein-Felder
Optionsfeld	Auswahl zwischen verschiedenen, stark begrenzten Möglichkeiten.
Grafisches Kontrollfeld	Anzeige von Bildern und seit LO 5.0 auch *.pdf-Dateien (1. Seite) aus einer Datenbank und Neueingabe von Bildern und seit LO 5.0 auch anderen Dateien, abhängig von der Einstellung **Extras → Optionen → LibreOffice → Allgemein→ Dialoge zum Öffnen/Speichern** und den Systemdialogen des Betriebssystems, in eine Datenbank über eine Pfadangabe
Maskiertes Feld	Eingabe in eine vorgefertigte Maske; grenzt die Eingabemöglichkeiten auf bestimmte Zeichenkombinationen ein.
Tabellen-Kontrollfeld	Universelles Eingabemodul, das eine ganze Tabelle bedienen kann. Integriert in dieses Kontrollfeld sind wieder viele der obigen Kontrollfelder

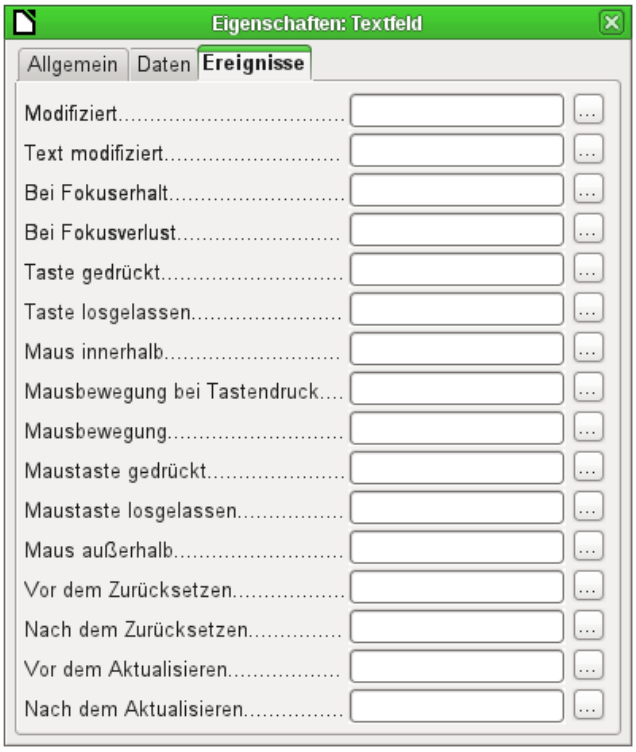
Gestaltung	
Kontrollfeld	**Anwendungsgebiet**
Beschriftungsfeld	Überschrift über das Formular, Beschriftung anderer Kontrollfelder
Gruppierungsrahmen	Linienzug um z. B. verschiedene Optionsfelder

Interaktion	
Kontrollfeld	**Anwendungsgebiet**
Schaltfläche	Button mit Beschriftung
Grafische Schaltfläche	Wie der Button, nur mit einer zusätzlich auf dem Button erscheinenden Grafik
Navigationsleiste	Leiste mit geringen Abweichungen zu der, die am unteren Bildschirmrand angezeigt wird.
Dateiauswahl	Auswahl von Dateien z. B. zum Hochladen in HTML-Formularen. - nicht weiter beschrieben
Drehfeld	Nur über Makroprogrammierung verwendbar – nicht weiter beschrieben
Bildlaufleiste	Nur über Makroprogrammierung verwendbar – nicht weiter beschrieben
Verstecktes Steuerelement	Hier kann ein Wert über Makros eingespeichert und wieder ausgelesen werden.

Standardeinstellungen vieler Kontrollfelder

Die Eigenschaften werden wie beim Formular in drei Kategorien unterteilt: Allgemein, Daten und Ereignisse. Unter Allgemein wird all das eingestellt, was für den Nutzer sichtbar ist. In der Kategorie Daten wird die Verbindung zu einem Feld der Datenbank hergestellt. Die Kategorie Ereignisse schließlich regelt Auslösemomente, die mit irgendwelchen Makros verbunden werden können. Für eine Datenbank ohne Makros spielt diese Kategorie keine Rolle.





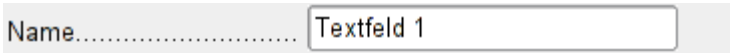
Hinweis

In den folgenden Übersichten stehen die deutschen Begriffe vor allem der Eigenschaften – genau wie im Formular-Entwurf. In Makros werden zur Beeinflussung der Felder die englischen Begriffe benötigt; diese sind in [Klammern] angegeben. So bedeutet z. B. [Printable] «Druckbar».
Beispiel:

```
001  oDoc = thisComponent
002  oDrawpage = oDoc.drawpage
003  oForm = oDrawpage.forms.getByName("Formular")
004  oFeld = oForm.getByName("Formularfeld")
005  oFeld.Printable = True 'Das Feld wird auf "Druckbar"→"Ja" gesetzt.
```

Die Eigenschaften werden einfach mit einem Punkt verbunden an das entsprechende Objekt des Formularfeldes angehängt. Weitere Hinweise stehen im Kapitel «Makros».

Allgemein

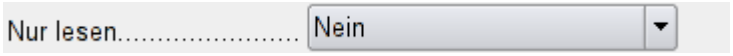


Die Bezeichnung für das Feld sollte innerhalb eines Formulars nur einmal vorkommen – Anwendung bei Zugriff über Makros.
[Name]

Gehört zu dem Feld ein Beschriftungsfeld? Hiermit wird eine Gruppierung festgelegt. Über das Beschriftungsfeld kann dann das Formularfeld mit einer Tastenkombination direkt erreicht werden.
[LabelControl]

Nicht aktivierte Felder sind nicht verwendbar und werden grau hinterlegt. Sinnvoll bei Steuerung über Makros (Entscheidung: Wenn in Feld 1 ein Wert eingegeben wird, darf in Feld 2 kein Wert eingegeben werden – Feld 2 wird deaktiviert)
[Enabled]

In der Regel 'Ja'; nicht sichtbare Felder können Werte zwischenspeichern. Anwendung z. B. bei der Erstellung von Kombinationsfeldern mit Makros.
[EnableVisible]

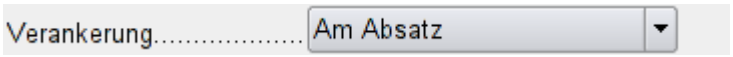


'Ja' würde eine Veränderung des Wertes ausschließen; z. B. für die Anzeige eines automatisch erstellten Primärschlüssels sinnvoll.
[ReadOnly]

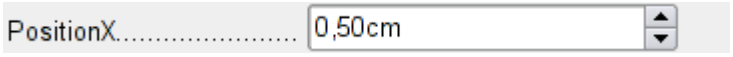
Manchmal sind Seitenausdrucke aus einem Formular sinnvoller als ein separater Bericht. Hier sollen dann eventuell nicht alle Felder erscheinen.
[Printable]

Durch ein Formular wird in der Regel mit dem Tabulator navigiert. Ein Feld, das z. B. nur gelesen wird, braucht keinen Stop des Tabulators, würde also übersprungen.
[Tabstop]

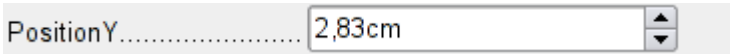
Hat das Feld einen Tabstop? Hier wird die Reihenfolge innerhalb des Formulars eingestellt.
[Tabindex]



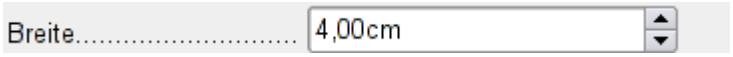
Verankerung der Grafik, die das Textfeld darstellt.



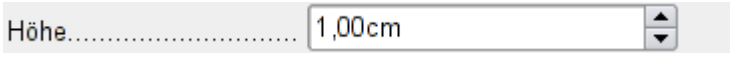
Position links oben vom linken Rand aus.
[PosSize.X]



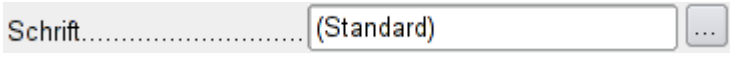
Position von oben aus.
[PosSize.Y]



Breite des Feldes
[PosSize.Width]

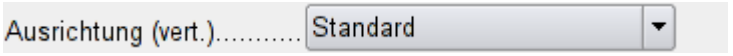


Höhe des Feldes
[PosSize.Height]

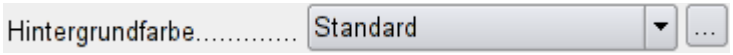


Schriftart, Schriftschnitt, Schriftgrad und Schrifteffekt sind hier einstellbar.
[Fontxxx]

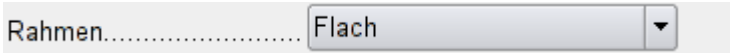
Der Eintrag wird linksbündig dargestellt.
[Align]



Standard | Oben | Mitte | Unten
[VerticalAlign]



Hintergrundfarbe des angezeigten Textfeldes
[BackgroundColor]



Rahmenform: Ohne Rahmen | 3D-Look | Flach
[Border]

Wenn ein Rahmen, dann kann hier die Umrandungsfarbe eingestellt werden.
[BorderColor]

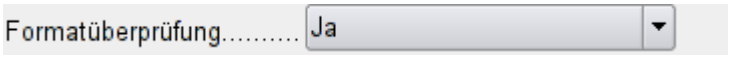
Markierter Text wird so nicht mehr als markiert angezeigt, wenn das Textfeld den Fokus verliert.
[HideInactiveSelection]

Gut nutzbar für Informationen, die mittels Makros ausgelesen werden sollen, siehe *Zusatzinformation eines Feldes* im Kapitel «Makros».
[Tag]

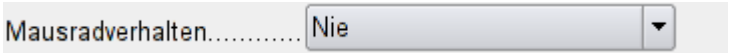
Erscheint als sogenannter Tooltip, wenn mit der Maus über das Textfeld gefahren wird.
Barrierefreiheit: Dieser Text wird von Screenreadern bei MacOS vorgelesen.
[HelpText]

Verweist z. Zt. nur auf die interne Hilfedatei. Die entsprechende Zuordnung von einer sogenannten Hilfe-ID (HID) zu den Verlinkungen in der Hilfe steht in der Datei **help/hid2file.js**. Diese Datei enthält ein Array, von dem das Element, das vor dem Doppelpunkt steht, zum Anspringen der internen Hilfe genutzt werden kann. Durch F1 abrufbar, wenn der Fokus auf dem Feld liegt.
[HelpURL]

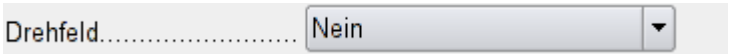
Zusätzlich sind bei Zahlenfeldern, Datumsfeldern u.ä. üblich:



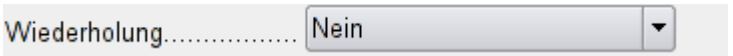
Mit eingeschalteter Überprüfung ist nur die Eingabe von Ziffern und Komma möglich.
[EnforceFormat]



'Nie' erlaubt keine Änderung mit dem Mausrad; 'Wenn ausgewählt' lässt eine Änderung zu, wenn das Feld ausgewählt ist und die Maus sich über dem Feld befindet; 'Immer' bedeutet, dass sich die Maus über dem Feld befinden muss.
[MouseWheelBehavior]



Ein Drehsymbol wird an der rechten Seite des Feldes eingeblendet.
[Spin]



Wenn ein Drehfeldpfeil länger gedrückt wird lässt sich hier einstellen, ob nicht nur zum nächsten Wert gedreht werden soll.
[Repeat]

Stellt die Verzögerungszeit ein, nach der der Druck auf die Maustaste beim Drehfeld Wiederholung interpretiert wird.
[RepeatDelay]

Daten

Datenfeld: Hier wird die Verbindung zur Tabelle hergestellt, die dem Formular zugrunde liegt.
[Model.DataField]

Leere Zeichenfolge ist NULL: Soll ein leeres Feld geleert werden (NULL) oder nur der Inhalt gelöscht werden?

Eingabe erforderlich: Dieser Eintrag sollte sich mit dem in der Tabelle decken. Dann fragt die GUI gegebenenfalls nach einer Eingabe, wenn vom Nutzer kein Wert eingegeben wurde.
[Model.InputRequired]

Filtervorschlag: Bei einer Filterung der Daten werden die Inhalte dieses Feldes als Vorschläge zwischengespeichert. Achtung – dies ist bei umfangreichen Inhalten recht speicherintensiv.
[Model.UserValueFilterProposal]

Hinweis

Den Eintrag **Leere Zeichenfolge ist NULL** enthalten nicht alle Formularfelder. So verfügt z. B. ein *Numerisches Feld* nicht über den Eintrag. Das führt dazu, dass nach einem einmal erfolgten Eintrag bei einer Löschung das Feld immer wieder auf den Wert '0' zurück springt. Die Allroundlösung ist hier ein *Formatiertes Feld*.

Ereignisse

Modifiziert: Dieses Ereignis tritt ein, wenn das Kontrollfeld geändert wurde und anschließend den Fokus verloren hatte. Das Ereignis geht verloren, wenn direkt zum nächsten Datensatz gewechselt wird. Unter den Umständen ist also eine Änderung gespeichert, ohne vorher wahrgenommen zu werden. [com.sun.star.lang.EventObject]

Text modifiziert: Direkt auf den Inhalt bezogen; kann Text, Zahl o.a. sein, tritt also nach jeder Tastatureingabe auf. [com.sun.star.awt.TextEvent]

Fokuserhalt: Der Cursor kommt in das Feld hinein. Hier sollte auf keinen Fall über das Makro eine Messagebox auf dem Bildschirm erzeugt werden. Durch das Anklicken dieser Box verliert das Formularfeld den Fokus und erhält ihn direkt danach zurück – eine Schleife wird dadurch erzeugt. Sie kann nur durch Tastatureingaben unterbrochen werden.

Fokusverlust: Der Cursor bewegt sich aus dem Feld heraus. Auch dies kann zu einem Wechselspiel führen, wenn eine zu bestätigende Bildschirmausgabe erfolgt.

Taste: Bezieht sich auf die Tastatur. Die Taste muss ausgelöst werden, wenn der Cursor in dem Kontrollfeld steht. Auch der Tabulator zum Verlassen des Kontrollfeldes löst dieses Ereignis aus. Dem Ereignis wird über den **KeyCode** bzw. **KeyChar** die auslösende Taste (Buchstabe, Zahl, Spezialtaste) mitgegeben. [com.sun.star.awt.KeyEvent]

Maus: selbsterklärend; Ereignisse treten nur ein, wenn vorher die Maus innerhalb des Feldes ist oder war («außerhalb» entspricht javascript onMouseOut). [com.sun.star.awt.MouseEvent]

Zurücksetzen: Das Formular wird von allen Daten geleert (Anlegen eines neuen Datensatzes) oder auf den alten Datenstand zurück gesetzt (Änderung eines bestehenden Datensatzes). Bei einem Formularfeld wird das Ereignis nur ausgelöst, wenn über den Button in der Navigationsleiste die Dateneingabe rückgängig gemacht wird. [com.sun.star.lang.EventOb-

ject]
Wenn ein Formular aufgerufen wird, wird nacheinander das Ereignis *Vor dem Zurücksetzen* und *Nach dem Zurücksetzen* abgearbeitet, bevor das Formular für eine Eingabe verfügbar ist.

Aktualisieren: Ist dies Ereignis an ein Kontrollfeld des Formulars gebunden, so tritt die Aktualisierung bei Fokusverlust und Sprung zu einem anderen Formularfeld auf, wenn der Inhalt des Kontrollfeldes geändert wurde. Änderungen in dem Formular werden übernommen und angezeigt. Bei Schließen eines Formular werden nacheinander die Ereignisse *Vor dem Aktualisieren* und *Nach dem Aktualisieren* abgearbeitet. [com.sun.star.lang.EventObject]

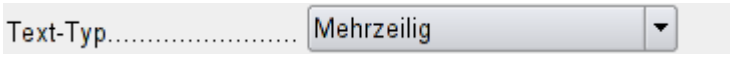
Textfeld

Neben den bereits unter *Standardeinstellungen vieler Kontrollfelder* erklärten Eigenschaften gibt es die folgenden Besonderheiten:

Allgemein

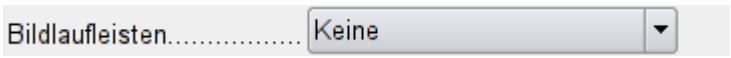
Solange der Wert '0' ist wird die Einstellung nicht berücksichtigt. In der Praxis wird hier die Zeichenlänge des Feldes aus der Datenbank übernommen, auf das sich das Textfeld bezieht.
[MaxTextLen]

Soll in einem leeren Feld ein Standardtext erscheinen? Dieser Text muss gelöscht werden, wenn ein anderer Eintrag erfolgen soll.
[DefaultText]



Mögliche Typen: 'Einzeilig', 'Mehrzeilig', 'Mehrzeilig mit Formatierungen' (wobei sich die beiden letzten nur im Tabulator unterscheiden – und ein Feld mit Formatierungen nicht an eine Datenbank angebunden werden kann). Bei mehrzeiligen Feldern ist die vertikale Ausrichtung nicht aktiv.
[MultiLine]

'Unix' oder 'Windows'? Prinzipiell funktionieren beide Endungen. Intern müssen für Windowszeilenenden aber zwei Steuerzeichen verwendet werden (CR und LF).
[LineEndFormat]



Nur bei mehrzeiligen Feldern:
'Keine', 'Horizontal', 'Vertikal', 'Beide'
Nur bei eingeschalteter Bildlaufleiste funktioniert das Scrollen mit der Maus.
[HScroll], [VScroll]

Aktiv nur bei einzeiligen Feldern.
[EchoChar]

Daten

keine weiteren Besonderheiten

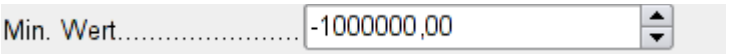
Ereignisse

keine weiteren Besonderheiten

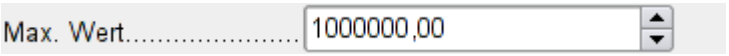
Numerisches Feld

Neben den bereits unter *Standardeinstellungen vieler Kontrollfelder* erklärten Eigenschaften gibt es die folgenden Besonderheiten:

Allgemein



Mindestwert, den dieses Feld einnehmen kann. Sollte mit dem Mindestwert übereinstimmen, der in der Tabelle erwartet wird.
[ValueMin]

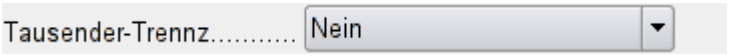


Maximalwert
[ValueMax]

Intervall-Wert für die Funktion als Scrollelement per Mausrad bzw. Drehfeld
[ValueStep]

Wert, der beim Erstellen eines neuen Datensatzes angezeigt wird.
[DefaultValue]

Anzahl Nachkommastellen, bei Integer-Feldern auf '0' zu stellen
[DecimalAccuracy]



Trennzeichen für Tausenderstellen, in der Regel der Punkt
[ShowThousandsSeparator]

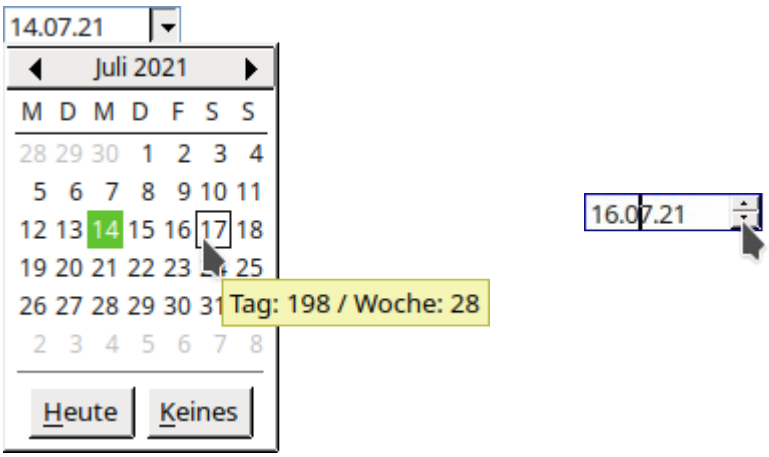
Daten

Es erfolgt keine Nachfrage, ob das Feld NULL sein soll. Eine fehlende Eingabe belässt dennoch das Feld auf NULL, nicht auf '0'. Wird allerdings erst einmal eine Eingabe gemacht, so lässt sich diese nicht komplett löschen. Statt NULL wird sie anschließend auf '0' gesetzt. Ein Filtervorschlag wird ebenfalls nicht erstellt.

Ereignisse

Es fehlt das Feld **Modifiziert**. Änderungen werden über **Text modifiziert** (hier wohl nicht wörtlich zu nehmen) angesprochen.

Datumsfeld



Das Datumsfeld kann als aufklappbares Feld und als Drehfeld definiert werden.

Beim aufklappbaren Feld ist ein Monatskalender eingeblendet. Ein Klick auf «Heute» gibt das aktuelle Datum ein. In der Tipp-Hilfe erscheint dann auch noch genau der Tag im Jahr und die entsprechende Woche.

Beim Drehfeld entscheidet der Stand des Cursors darüber, welcher Wert geändert wird. Steht der Cursor, wie im Screenshot, im Wert für den Monat, so verändert ein Klick auf das Drehfeld den Wert für den Monat.

Neben den bereits unter *Standardeinstellungen vieler Kontrollfelder* erklärten Eigenschaften gibt es die folgenden Besonderheiten:

Allgemein

Min. Datum..... 01.01.1800	Mindestwert für das Feld, einstellbar über ein Aufklappfeld, das einen Kalender bereitstellt. [DateMin]
Max. Datum..... 31.12.2200	Maximalwert. [DateMax]
Datumsformat..... Standard (kurz)	Kurzform wie 10.02.12 sowie unterschiedliche Formen auch mit '/' statt '.' oder '-' in amerikanischer Schreibweise. [DateFormat]
Standarddatum.....	Hier kann ein festes Datum vorgegeben werden. Das aktuelle Datum (Heute) beim Aufruf des Formulars muss leider (noch) durch ein Makro eingetragen werden. [DefaultDate]
Aufklappbar..... Nein	Ein Monatskalender zur Auswahl des Tages kann eingeblendet werden. [DropDown]

Daten

Es erfolgt keine Nachfrage, ob das Feld NULL sein soll. Eine fehlende Eingabe belässt dennoch das Feld auf NULL, nicht auf '0'.
Ein Filtervorschlag wird ebenfalls nicht erstellt.

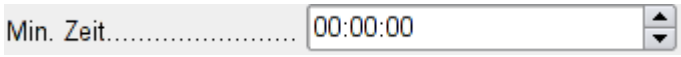
Ereignisse

Es fehlt das Feld **Modifiziert**. Änderungen werden über **Text modifiziert** (hier wohl nicht wörtlich zu nehmen) angesprochen.

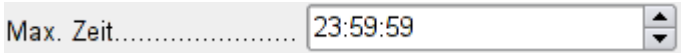
Zeitfeld

Neben den bereits unter *Standardeinstellungen vieler Kontrollfelder* erklärten Eigenschaften gibt es die folgenden Besonderheiten:

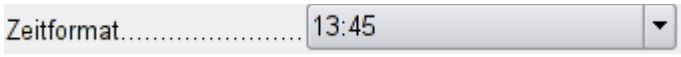
Allgemein



Mindestwert für das Feld, standardmäßig auf '0' gesetzt.
[TimeMin]



Maximalwert, standardmäßig auf 1 Sekunde unter 24 Uhr gesetzt.
[TimeMax]



Kurzform ohne Sekunden, Langform mit Sekunden sowie Benennungen mit PM (post meridiem / Nachmittag)
[TimeFormat]

Eine feste Zeit ist voreinstellbar, die aktuelle Zeit beim Abspeichern des Formulars leider (bisher) nur mit Makro.
[DefaultTime]

Daten

Es erfolgt keine Nachfrage, ob das Feld NULL sein soll. Eine fehlende Eingabe belässt dennoch das Feld auf NULL, nicht auf '0'.
Ein Filtervorschlag wird ebenfalls nicht erstellt.

Ereignisse

Es fehlt das Feld **Modifiziert**. Änderungen werden über **Text modifiziert** (hier wohl nicht wörtlich zu nehmen) angesprochen.

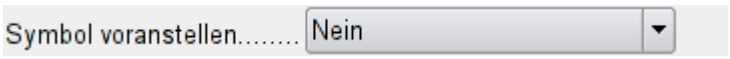
Währungsfeld

Neben den bereits unter *Standardeinstellungen vieler Kontrollfelder* erklärten Eigenschaften gibt es die folgenden Besonderheiten:

Allgemein

Min. Wert, Max. Wert, Intervall, Standardwert, Nachkommastellen und Tausender-Trennzeichen entsprechen den allgemeinen Eigenschaften *Numerisches Feld*. Daneben gibt es lediglich:

Das Symbol wird angezeigt, aber nicht in der dem Formular zugrundeliegenden Tabelle mit abgespeichert.
[CurrencySymbol]



Soll das Symbol vor oder hinter der Zahl erscheinen?
[PrependCurrencySymbol]

Daten

Es erfolgt keine Nachfrage, ob das Feld NULL sein soll. Eine fehlende Eingabe belässt dennoch das Feld auf NULL, nicht auf '0'.
Ein Filtervorschlag wird ebenfalls nicht erstellt.

Ereignisse

Es fehlt das Feld **Modifiziert**. Änderungen werden über **Text modifiziert** (hier wohl nicht wörtlich zu nehmen) angesprochen.

Formatiertes Feld

Neben den bereits unter *Standardeinstellungen vieler Kontrollfelder* erklärten Eigenschaften gibt es die folgenden Besonderheiten:

Allgemein

Min. Wert und Max. Wert und Standardwert hängen von der Formatierung ab. Hinter dem Button zur Formatierung versteckt sich ein «Allroundfeld", das Währungsfeld und Zahlenfeld meist überflüssig macht. Im Gegensatz zum einfachen Währungsfeld kann das formatierte Feld negative Beträge auch in roter Farbe darstellen.

Über den rechten Button mit den drei Punkten erscheint die Auswahl zum Zahlenformat, die auch in LibreOffice-Calc üblich ist.
[FormatKey]

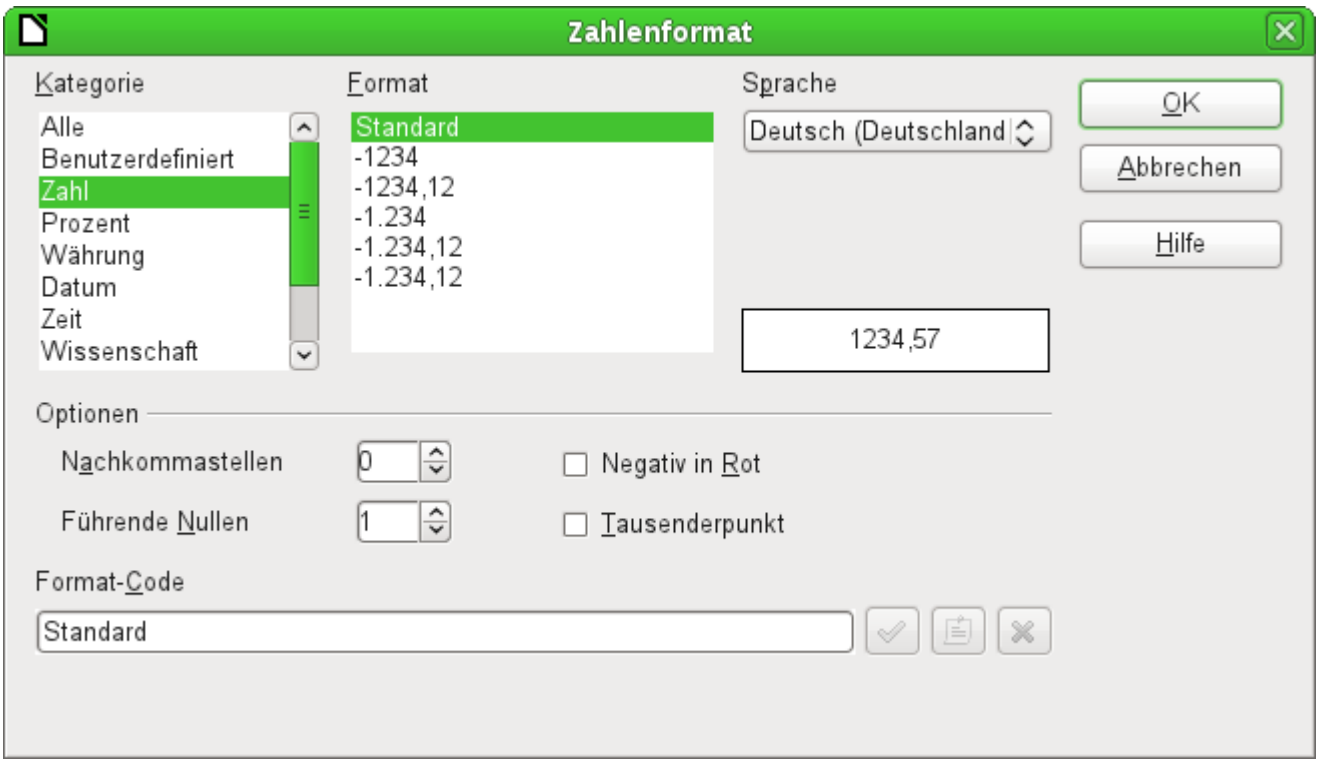


Abbildung 24: Formatiertes Feld mit allgemeinen Zahleneinstellungen

In dem Zahlenformat sind dann neben Datum, Zeit, Währung oder normalen Zahlenformaten auch Möglichkeiten gegeben, die Felder mit einer Maßeinheit wie z. B. **kg** zu bestücken. Siehe dazu die allgemeine Hilfe zu Zahlenformat-Codes.

Mit dem formatierten Feld ist es auch möglich, Timestamp-Felder aus Tabellen mit nur einem Feld darzustellen und zu beschreiben. Der Formularassistent nutzt dazu sonst ein Datumsfeld und ein Zeitfeld.

Nur mit Hilfe von Makros ist es bei Timestamp-Feldern möglich, Eingaben im Format Minuten:Sekunden,Hundertstelsekunden zu erreichen.

Das Zahlenformat erlaubt auch mit Einstellungen im Format-Code wie z.B.

```
001 [<20][BLAU]0" °C";[>30][ROT]0" °C";[SCHWARZ]0" °C "
```

Werte entsprechend farbig darzustellen. Hier im Beispiel Temperaturen unter 20°C in Blau und über 30°C in Rot, sonst in Schwarz.

Hinweis

Werden Formulare mit Hilfe des Assistenten erstellt, dann wird der Schrift in den Formularfeldern eine Farbe fest zugewiesen. Die formatierten Felder richten sich dann nicht nach dem vorgesehenen Format-Code. Dies kann in dem Steuerelement unter **Eigenschaften → Allgemein → Schrift → Button ... → Schrifteffekte → Schriftfarbe** mit der Einstellung **Automatisch** rückgängig gemacht werden. Das Steuerelement übernimmt die Eigenschaft, auch wenn es nachher wieder statt «Automatisch» «Schwarz» anzeigt. Die Farbe wird allerdings vorübergehend zurückgesetzt, wenn der Cursor in dem Feld steht.

Daten

keine weiteren Besonderheiten.

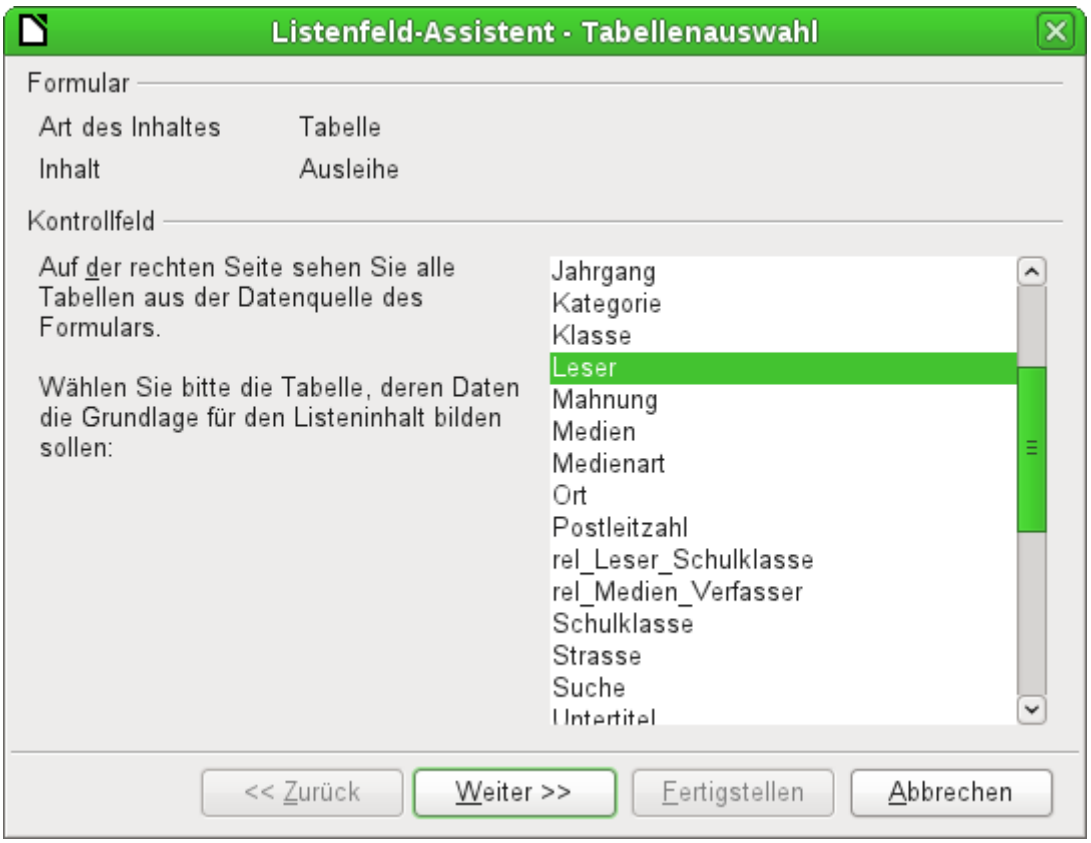
Ereignisse

Es fehlt das Feld **Modifiziert**. Änderungen werden über **Text modifiziert** (hier wohl nicht wörtlich zu nehmen) angesprochen.

Listenfeld

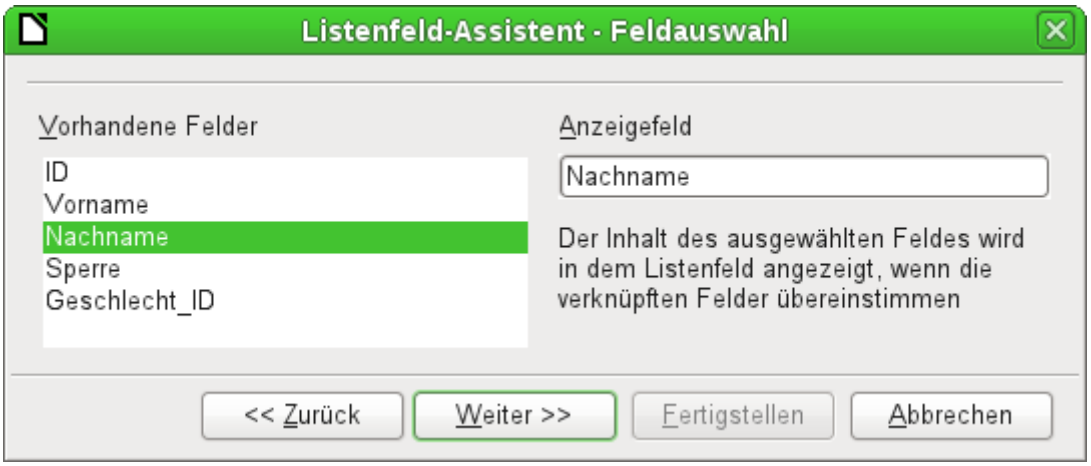
Sobald ein Listenfeld erstellt wird, erscheint standardmäßig der Listenfeldassistent. Diese Automatik lässt sich gegebenenfalls über den Steuerelemente-Assistenten an/aus (*Abbildung 23*) abschalten.

Assistent

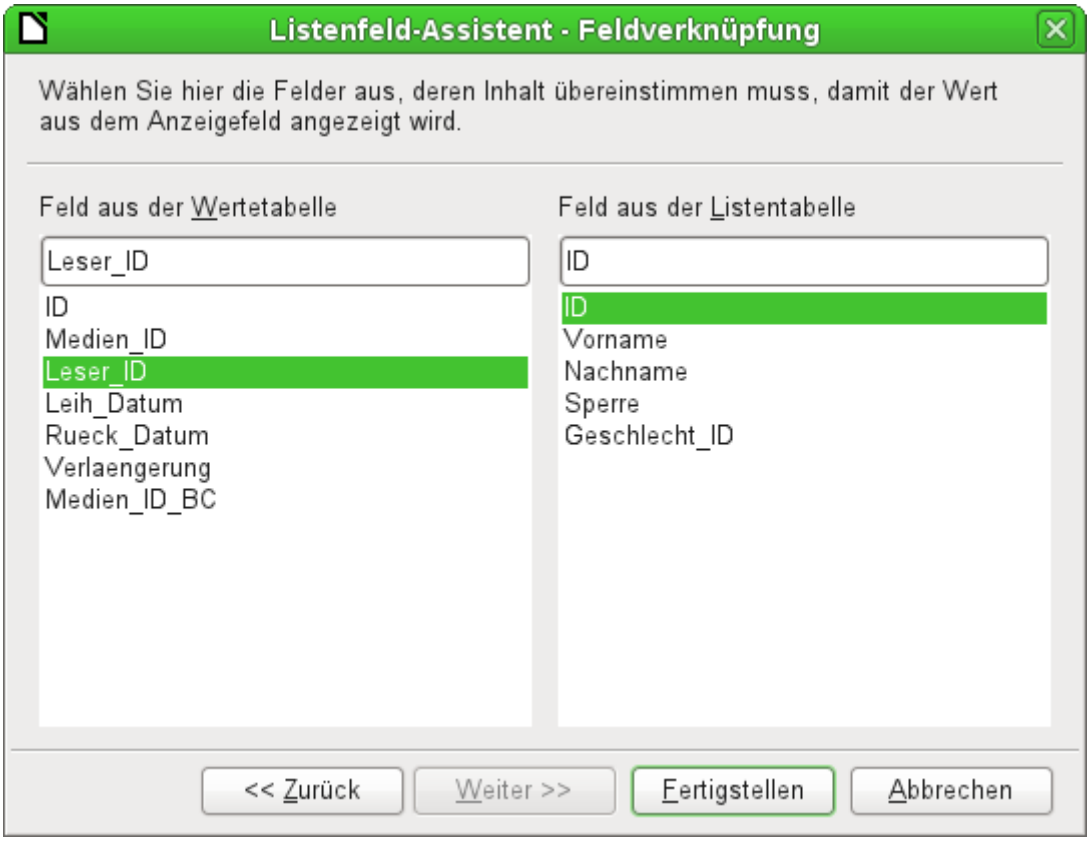


Das Formular ist bereits definiert. Es verbindet mit einer 'Tabelle', die den Namen "Ausleihe" trägt. Ein Listenfeld zeigt andere Daten für den Nutzer an als solche, die an die 'Tabelle' weitergegeben werden. Diese Daten stammen bei Datenbanken in der Regel aus einer anderen Tabelle als der, mit der das Formular verbunden ist.

In der Tabelle "Ausleihe" soll verzeichnet werden, welcher "Leser" welche Medien entliehen hat. Allerdings wird in dieser Tabelle nicht der Name des Lesers gespeichert sondern der Primärschlüssel aus der Tabelle "Leser". Die Tabelle "Leser" bildet also die Grundlage für das Listenfeld.



Das Feld "Nachname" aus der Tabelle "Leser" soll in dem Listenfeld sichtbar sein. Es ist also das *Anzeigefeld*.



Das Feld "Leser_ID" befindet sich in der dem Formular zugrundeliegenden Tabelle "Ausleihe". Diese Tabelle wird hier als *Wertetabelle* bezeichnet. Der Primärschlüssel "ID" aus der Tabelle "Leser" soll mit diesem Feld verbunden werden. Die Tabelle "Leser" wird hier als *Listentabelle* bezeichnet.

Das Listenfeld ist jetzt komplett mit Daten und Standardeinstellungen erstellt und funktionsfähig.

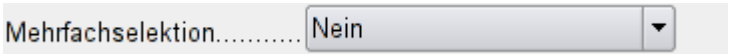
Neben den bereits unter *Standardeinstellungen vieler Kontrollfelder* erklärten Eigenschaften gibt es die folgenden Besonderheiten:

Allgemein

Die Listeneinträge wurden bereits durch den Assistenten erstellt. Hier könnten sonst auch Einträge stehen, die in keiner Tabelle der Datenbank verzeichnet sind. Mit Listeneinträgen sind hier die sichtbaren Einträge gemeint; nicht die Einträge, die über das Formular an die Tabelle weitergegeben werden. [StringItemList]

Wird das Feld als nicht aufklappbar festgelegt, so erscheinen bei Aufruf des Formulars am rechten Rand des Listenfeldes Scrollpfeile. Das Listenfeld wird dann automatisch zu einem mehrzeiligen Feld, bei dem die aktuelle Auswahl markiert ist. [Dropdown]

Falls aufklappbar wird hier die maximal sichtbare Anzahl der Zeilen eingegeben. Geht der Inhalt über mehr Zeilen, so erscheint beim Aufklappen ein Scrollbalken. [LineCount]



Sollen mehrere Werte selektiert werden können? Im obigen Beispiel ist das nicht möglich, da ein Fremdschlüssel abgespeichert wird.
In der Regel wird diese Funktion in Verbindung mit Datenbanken nicht genutzt werden, da jedes Feld eigentlich nur einen Wert einnehmen sollte.
Diese Funktion ließe sich besonders bei Datenbanken nutzen, die (wie PostgreSQL) den Datentyp eines Arrays benutzen. Das Einfügen und Auslesen ist aber auch hier nicht direkt zugänglich. Eine Auswertung des Eintrags in das Listenfeld über Makros kann hier Abhilfe schaffen. [MultiSelection] [MultiSelectionSimpleMode]

Eine Standardselektion bei einer Verbindung mit einer Datenbanktabelle, wie über den Listenfeldassistenten erzeugt, muss natürlich den Inhalt der entsprechenden Liste wiedergeben können.
In dieses Feld wird gegebenenfalls die Position des Eintrages aus der Datenliste eingegeben. [DefaultSelection]

Daten

Neben den üblichen Daten-Eigenschaften Datenfeld und Eingabe erforderlich sind hier Eigenschaften von Bedeutung, die die Verbindung von anzuzeigenden Daten und in die dem Formular zugrundeliegende Tabelle einzutragenden Daten herstellen.

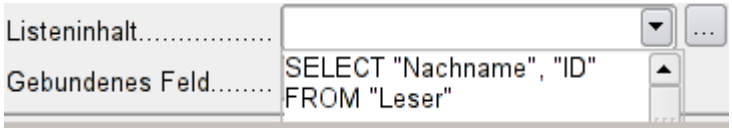
Art des Listeninhaltes: 'Werteliste', 'Tabelle', 'Abfrage', 'SQL', 'SQL (Native)', 'Tabellenfelder' [ListSourceType]

Listeninhalt → Werteliste: Sind unter **Allgemein** Listeneinträge gemacht worden, so werden hier die entsprechenden Werte eingegeben, die mit dem Formular abgespeichert werden sollen. Der Listeninhalt wird bestückt, indem bei der Eingabe die Inhalte über **Shift – Enter** aneinandergehängt werden. Es erscheint dann in dem Listeninhalt **"Wert1";"Wert2";"Wert3" …** Die Eigenschaft **Gebundenes Feld** ist *inaktiv*.

Listeninhalt → Tabelle: Hier wird eine der Datenbanktabellen ausgesucht. Dabei muss aber berücksichtigt werden, dass dann immer das erste Feld der Tabelle in dem Listenfeld angezeigt wird.
Üblicherweise sind Tabellen so aufgebaut, dass zuerst der Primärschlüssel erscheint und anschließend die Inhalte. Solche Tabellen eignen sich nur dann für ein Listenfeld, wenn tatsächlich der Primärschlüssel angezeigt werden soll.

Listeninhalt → Abfrage: Hier kann extern eine Abfrage vorformuliert werden, die dann auch als Abfrage sichtbar abgespeichert wird. Die Gestaltung solcher Abfragen wird im Kapitel 'Abfragen' erklärt. Durch die Abfrage ist es dann möglich, das Feld "ID" von der ersten Position in der ursprünglichen Tabelle an die zweite Position zu bewegen, was hier dem gebundenen Feld 1 entspricht.

Listeninhalt → SQL: Hiermit bestückt der Listenfeldassistent das Listenfeld. Die von dem Assistenten konstruierte Abfrage sieht dann so aus:



Die Abfrage ist die einfachste Möglichkeit, die sich bietet. Das Feld "Nachname" erscheint an Position 0, das Feld "ID" an Position 1. Beide werden aus der Tabelle "Leser" ausgelesen. Da das gebundene Feld das Feld 1 ist reicht diese SQL-Formulierung aus. Schon vom Standard her sollte aber hier ergänzt werden **ORDER BY "Nachname" ASC**, denn ohne diese Formulierung werden die Nachnamen so wiedergegeben, dass ihre Reihenfolge der Eingabe in die Tabelle Leser entspricht. Zusätzliches Problem dürfte sein, dass "Nachname" bei einigen Lesern gleich ist. Hier muss dann "Vorname" hinzugezogen werden können und, als letztes Mittel, wohl auch der Primärschlüssel "ID". Wie so etwas genauer formuliert wird, kann im Kapitel 'Abfragen' genauer nachgelesen werden.

Listeninhalt → SQL (Nativ): Die SQL-Formulierung wird direkt eingegeben, kein Assistent wird dazu aufgerufen. Base wertet die Abfrage nicht aus. Dies bietet sich an, wenn in der

Abfrage Funktionen stecken, die eventuell von der GUI von Base nicht verstanden werden. Dann wird die Abfrage nicht weiter auf Fehler untersucht. Genaueres zu dem direkten SQL-Modus im Kapitel 'Abfragen'.

Listeninhalt → Tabellenfelder: Hier werden die *Feldnamen* einer Tabelle aufgelistet, nicht die Inhalte. Für die Tabelle "Leser" erscheint dort also die Liste "ID", "Vorname", "Nachname", "Sperre", "Geschlecht_ID".

Hinweis

Für Zeitfelder, die auch Zeiten im Millisekunden-Bereich darstellen sollen, sind, wie unter *Zeitfelder in Tabellen* beschrieben, Timestamp-Felder erforderlich. Die Darstellung der Millisekunden funktioniert bei der Zusammenfügung von Zeichen in einem Listenfeld nicht. Hier muss der Timestamp in Text umgewandelt werden:

```
001 SELECT REPLACE(LEFT(RIGHT(CAST("Zeit" AS
    VARCHAR(30)),15),8),'.',',') AS "Listinhalt", "ID"
    FROM "Leistung_erforderlich"
```

Dies erzeugt die Anzeige in Minuten:Sekunden,Hundertstelsekunden. (HSQLDB)

Gebundenes Feld: Im Listenfeld wird ein Inhalt angezeigt, der nicht mit dem Inhalt identisch sein muss, der auch im Formular abgespeichert werden soll. Üblicherweise wird z. B. Ein Name angezeigt und der Primärschlüssel zu diesem Namen als abzuspeicherndes Feld bestimmt.

```
001 SELECT "Name", "ID" FROM "Tabelle" ORDER BY "Name"
```

Das Feld "ID" wird als Fremdschlüssel in der Tabelle gespeichert, die dem Formular zugrunde liegt. Die Zählung der Felder in Datenbanken beginnt mit 0, das an das Formular gebundene Feld hat also die Position '1'.

Wird stattdessen '0' gewählt, dann wird in dem Formular der Inhalt des Feldes "Name" abgespeichert. Das Feld "ID" könnte in der Abfrage entfallen.

Es ist sogar möglich, die Position '-1' zu wählen. Dann wird nicht der Inhalt der Abfrage sondern die Position des Eintrages in der Liste gespeichert. Der erste Datensatz hat dabei die Position 1.

Die oben aufgeführte Abfrage ergäbe z. B. die folgenden Daten:

Name	***ID***
Anneliese	2
Dorothea	3
Sieglinde	1

Im Listenfeld könnte nur der Name ausgewählt werden. Das Listenfeld wird auf den Namen «Dorothea» eingestellt. Folgende Abspeicherungen würden über das Listenfeld möglich:
Gebundenes Feld → '1' «3» wird abgespeichert, da das Feld "ID" gespeichert wird.
Gebundenes Feld → '0' «Dorothea» wird abgespeichert, da das Feld "Name" gespeichert wird.

Gebundenes Feld → '-1' «2» wird abgespeichert, da «Dorothea» als zweiter Datensatz in der Liste aufgeführt wird.

Hinweis

Die Änderung des gebundenen Feldes auf '0' oder '-1' ist erst mit der Version LO 4.1 eingeführt worden. Vorher war nur die Auswahl von Werten >= 1 möglich.

Hinweis

Ein Listenfeld zeigt in der Regel nicht nur die Werte an, die z.B. über SQL dargestellt werden. Als erstes wird ein leeres Feld (**NULL**) zur Auswahl angeboten. Dies ist notwendig, damit einmal eingegebene Daten über das Listenfeld auch wieder entfernt werden können.

*Abbildung 25: Für das Buchungskonto ist in **der** Tabelle eine Eingabe erforderlich. Das Umbuchungskonto kann auch leer bleiben.*

Soll von vornherein ausgeschlossen werden, dass das leere Feld angeboten wird, so ist die zugrundeliegende Tabelle für das Formular entsprechend einzustellen. Das Feld muss über **Feldeigenschaften → Eingabe erforderlich → Ja** so eingestellt sein, dass **NULL** nicht erlaubt ist.

Ereignisse

Neben den Standardereignissen werden folgende Ereignisse hinzugefügt:

Aktion ausführen: Wird durch Tastatureingabe ein Listenfeld dazu veranlasst, einen entsprechenden Inhalt anzuzeigen, so ist dies eine Aktion, die das Listenfeld ausführt.
Auch die Auswahl des Wertes aus einem Listenfeld entspricht dieser Aktion. Je nach System kann hier auch ein Doppelklick mit der Maus erforderlich sein.

Status geändert: Dies kann die Änderung des angezeigten Inhaltes eines Listenfeldes durch Betätigen des Aufklappbuttons sein. Es kann auch lediglich ein Klicken auf den Aufklappbutton des Feldes sein.

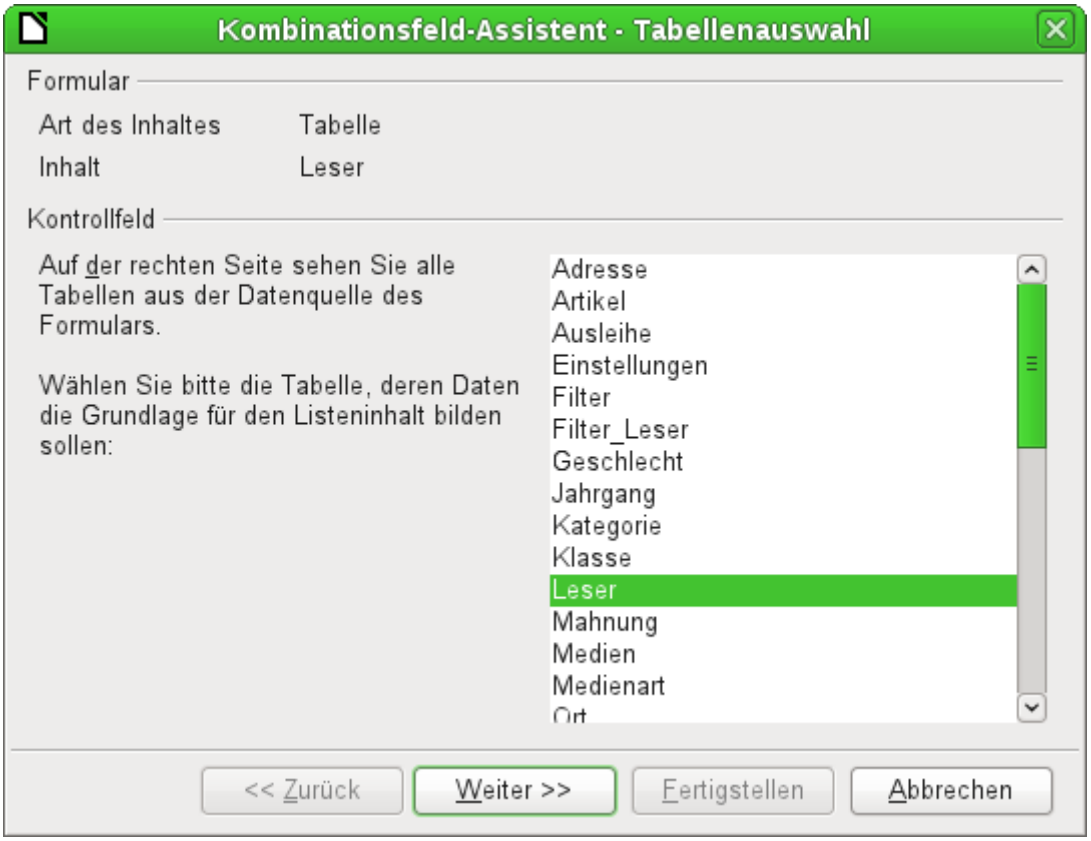
Fehler aufgetreten: Das Ereignis lässt sich leider beim Listenfeld nicht nachvollziehen.

Kombinationsfeld

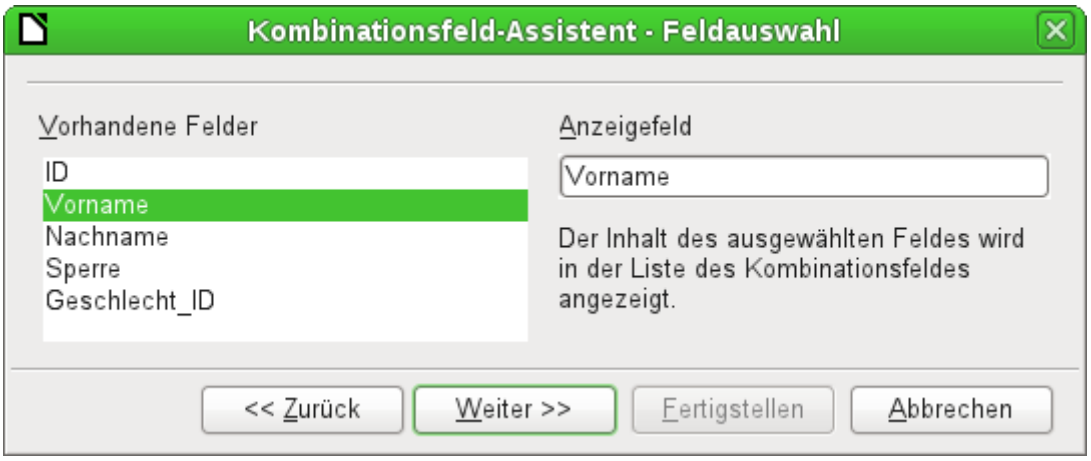
Sobald ein Kombinationsfeld erstellt wird erscheint wie beim Listenfeld standardmäßig ein Assistent. Diese Automatik lässt sich gegebenenfalls über Steuerelemente-Assistenten an/aus (*Abbildung 23*) abschalten.

Kombinationsfelder schreiben direkt den ausgewählten Wert in die dem Formular zugrundeliegende Tabelle. Deshalb ist im folgenden Beispiel die dem Formular zugrundeliegende Tabelle die Tabelle "Leser" und die für das Kontrollfeld gewählte Tabelle ebenfalls die Tabelle "Leser".

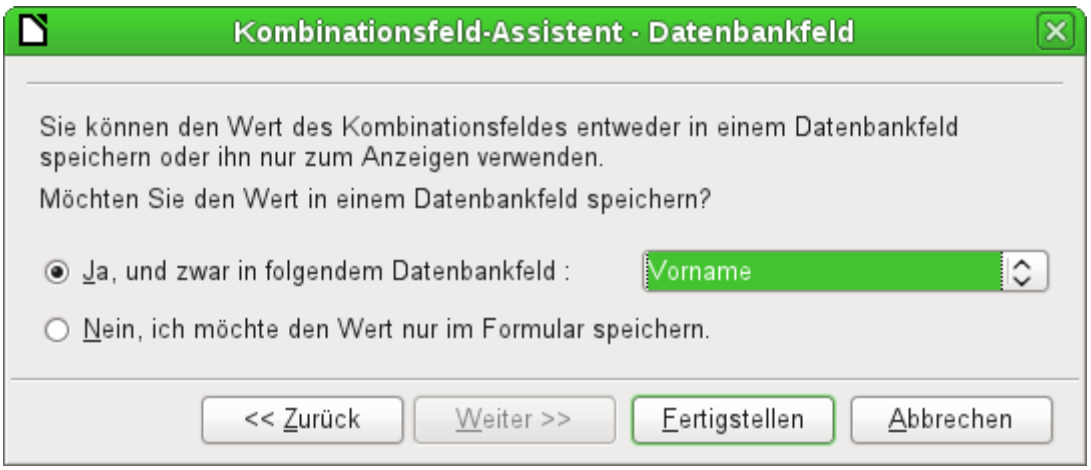
Assistent



Das Formular ist wieder vordefiniert, diesmal als Beispiel mit der Tabelle "Leser". Da die Daten, die in dem Listenfeld angezeigt werden, auch in dieser Tabelle abgespeichert werden sollen, wird als Quelle für die Daten des Listenfeldes ebenfalls die Tabelle "Leser" ausgewählt.



In der Tabelle "Leser" befindet sich das Feld "Vorname". Es soll im Kombinationsfeld angezeigt werden.



Innerhalb einer Datenbank scheint es wenig Sinn zu machen, den Wert des Kombinationsfeldes nicht in einem Feld zu speichern. Aus der Tabelle "Leser" sollen die Vornamen ausgelesen werden und für neue Leser als Vornamen zur Verfügung stehen, damit bei gleichen Vornamen eben nicht immer wieder eine neue Eingabe erfolgen muss. Das Kombinationsfeld zeigt dann hier den Vornamen an und die Texteingabe braucht nicht weiter durchgeführt zu werden.

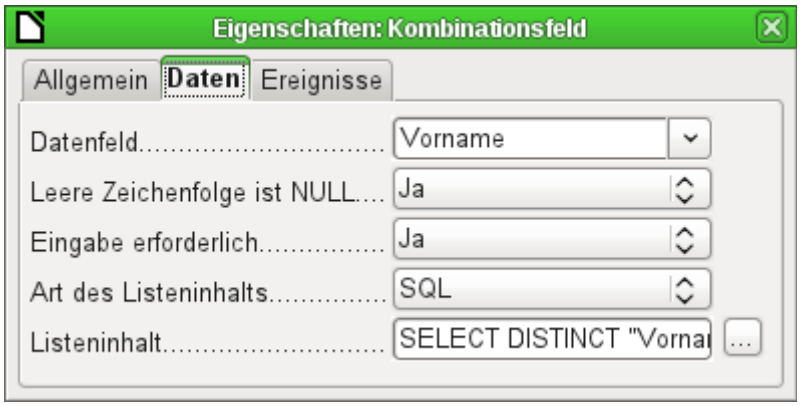
Soll ein neuer Wert eingegeben werden, so ist dies im Kombinationsfeld ohne weiteres möglich, denn das Kombinationsfeld zeigt genau das an, was es selbst an die zugrundeliegende Tabelle des Formulars weitergibt.

Neben den bereits unter *Standardeinstellungen vieler Kontrollfelder* und beim Listenfeld erklärten Eigenschaften gibt es die folgenden Besonderheiten:

Allgemein

Während der Eingabe von neuen Werten wird eine Liste der eventuell zutreffenden Werte angezeigt, die noch zur Auswahl stehen. [AutoComplete]

Daten



Die Datenfeldern entsprechen den bereits vorgestellten Standardeinstellungen und den Einstellungen bei einem Listenfeld. Der SQL-Befehl weist hier allerdings eine Besonderheit auf:

```
002 SELECT DISTINCT "Vorname" FROM "Leser"
```

Durch den Zusatz des Begriffes **DISTINCT** werden gleiche Vornamen nur einmal angezeigt. Wieder fehlt allerdings bei der Erstellung durch den Assistenten eine Sortierung der Inhalte.

Ereignisse

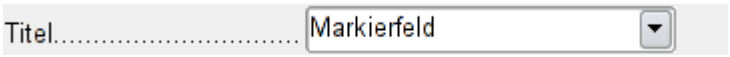
Die Ereignisse entsprechen denen beim Listenfeld.

Markierfeld

Das Markierfeld erscheint direkt als eine Kombination von Markierbox und Beschriftung dieser Box.

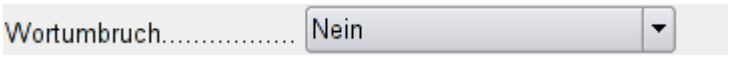
Neben den bereits unter *Standardeinstellungen vieler Kontrollfelder* erklärten Eigenschaften gibt es die folgenden Besonderheiten:

Allgemein



Die Beschriftung dieser Box, erscheint standardmäßig rechts von der Box. Zusätzlich ist noch eine Verbindung zu einem Beschriftungsfeld möglich.
[Label]

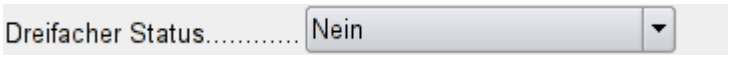
Hier gibt es, abhängig von der Auswahl in 'Dreifacher Status', bis zu drei Möglichkeiten: 'Nicht ausgewählt', 'Ausgewählt', 'Nicht definiert'. 'Nicht definiert' entspricht als Eintrag in der dem Formular zugrundeliegenden Tabelle NULL.
[State]



Standardmäßig wird der Titel nicht umbrochen. Ein zu langer Titel wird abgeschnitten, wenn das Feld nicht groß genug ist.
[MultiLine]

Hier kann eine Grafik statt bzw. zusätzlich zum Titel eingefügt werden. Dabei sollte beachtet werden, dass die Grafiken standardmäßig erst einmal nicht in das *.odb-Dokument eingebunden werden. Sinnvoll bei kleinen Grafiken ist es, die Grafiken einzubetten und nicht zu verknüpfen.
[Graphic]

Ist eine Grafik ausgewählt, so kann sie im Verhältnis zum Titel ausgerichtet werden.
[ImagePosition] ('0'='links', '1'='zentriert', '2'='rechts')



Standardmäßig gibt es für Markierfelder nur 'Ausgewählt' (Wert: '1') und 'Nicht ausgewählt' (Wert: '0'). Beim dreifachen Status kommt die Definition als 'leeres Feld' (NULL) hinzu.
[TriState]

Daten

Dem Markierfeld kann ein Referenzwert [RefValue] zugewiesen werden. In der Regel ist dies der Wert '1' für (ein) und '0' für (aus). Schließlich werden Markierfelder meist als Felder für die Auswahl von 'Ja' und 'Nein' genutzt.. Die Referenzwerte können auch andere Zahlen und Texte enthalten.

Wird ein Markierfeld an ein Textfeld gebunden, so wird an das Textfeld ab der Version LO 4.2 standardmäßig 'true' und 'false' weiter gegeben. Allerdings liest das Markierfeld auch die Werte '1' und '0' korrekt aus. Eine klare Wahl des Referenzwertes als 1 und 0 dient dann dazu, auch 1 und 0 in die Felder zu schreiben.

Ereignisse

Es fehlen die Felder **Modifiziert**, **Text modifiziert**, **Vor der Aktualisieren** und **Nach dem Aktualisieren**.

Zusätzlich erscheinen **Aktion ausführen** (siehe Listenfeld) und **Status geändert** (entspricht **Modifiziert**).

Hinweis

Markierfelder und Optionsfelder sind in Größe und Erscheinungsbild nicht veränderbar. Um ein anderes Erscheinungsbild zu erzeugen muss auf Makros zurück gegriffen werden. Siehe *Markierfelder durch Schaltflächen ersetzen.*

Optionsfeld

Das Optionsfeld entspricht bis auf eine Ausnahme in den allgemeinen Eigenschaften und die äußerlich andere Form (rund) dem eben beschriebenen Markierfeld.
Werden mehrere Optionsfelder mit gleichem Namen oder unterschiedlichem Namen, aber gleichem Gruppennamen, mit einem Tabellenfeld über das Formular verbunden, so kann von den Optionen immer nur eine Option gewählt werden. Gleiche Namen sollten möglichst vermieden werden, da dann die Navigation mit Tabulatoren zur Zeit nicht korrekt funktioniert.

Allgemein

Das Optionsfeld wird bevorzugt für Gruppierungen verwandt. Zwischen mehreren Optionen kann dann nur eine ausgewählt werden. Deswegen erscheint hier auch der Gruppenname, unter dem die Option angesprochen werden kann. [GroupName]

Daten

Siehe Markierfeld, hier werden aber die Referenzwerte [RefValue], die eingetragen wurden, auch tatsächlich an das Datenfeld weitergegeben.

Ereignisse

Siehe Markierfeld

Grafisches Steuerelement

Ein grafisches Steuerelement übernimmt die Eingabe und Ausgabe von Bildmaterial in die Datenbank. Das darunterliegende Datenfeld muss ein Binärfeld sein, wenn es das Bild direkt speichern soll. Es kann auch ein Textfeld sein. Dann wird lediglich der relative Pfad zur Datei abgespeichert. Hierbei sollte dann aber darauf geachtet werden, dass der Pfad zu der Datei auch bei einer Weitergabe der Datenbank stimmig bleibt.

Hinweis

Wird über einen Assistenten ein Formular erstellt, so erstellt der Assistent nur für Binärfelder ein grafisches Kontrollfeld. Bei Feldern, die den Pfad speichern sollen, muss dann das Textfeld mit der rechten Maustaste über **Ersetzen durch → Grafisches Steuerelement** zu einem grafischen Steuerelement umgewandelt werden.

Seit LO 5.0 können über dieses Feld beliebige Dateien, abhängig von den Dialogen der Benutzeroberfläche des Betriebssystems, in Base eingebunden werden. Angezeigt werden in dem Fall aber nur Bilder und von *.pdf-Dateien die erste Seite.

Hinweis

FIREBIRD: Der Feldtyp Bild [BLOB] funktioniert nicht mit dem grafischen Kontrollfeld. Hier wird der Feldtyp Blob [BLOB] benötigt.

Vorsicht

Bilder sollten auf jeden Fall verkleinert werden, wenn sie schon in der Datenbank gespeichert werden. Hat ein Kamerabild z.B. 3 MB, so passiert es sehr schnell bei Nutzung der internen HSQLDB, dass mit Angabe einer Java-Fehlermeldung (**`NullPointerException`**) die Abspeicherung der Daten nicht mehr möglich ist. Dies kann zum Verlust sämtlicher Daten führen, die seit dem Öffnen der Base-Datenbank in die Tabelle eingegeben wurden.

Die Eingabe in das grafische Steuerelement erfolgt entweder über einen Doppelklick mit der Maus; dann erscheint ein Dateiauswahlfenster. Oder sie erfolgt über einen Klick der rechten Maustaste; dann kann ausgewählt werden, ob die Grafik nur gelöscht oder eine andere geladen werden soll.

Das grafische Steuerelement wird in Graustufen angezeigt, wenn das Formular z.B. auf einer Abfrage beruht, die nicht bearbeitet werden kann. Dann kann natürlich auch über das Kontrollfeld keine Eingabe erfolgen.

Ein grafisches Steuerelement erhält standardmäßig erst einmal keinen Tabstop.

Neben den bereits unter *Standardeinstellungen vieler Kontrollfelder* erklärten Eigenschaften gibt es die folgenden Besonderheiten:

Allgemein

Die hier ausgesuchte Grafik wird nur im Kontrollfeld angezeigt, wenn das Formular bearbeitet wird. Für die spätere Eingabe ist sie ohne Bedeutung.
[Graphic]

'Nein': Das Bild wird nicht an das Kontrollfeld angepasst. Ist das Bild zu groß, so wird ein Ausschnitt des Bildes gezeigt. Das Bild wird nicht verzerrt.
'Seitenverhältnis beibehalten': Das Bild wird in das Kontrollfeld eingepasst, aber nicht verzerrt dargestellt.
'Autom. Größe': Das Bild wird der Größe des Kontrollfeldes angepasst und gegebenenfalls verzerrt dargestellt.
[ScaleImage] [ScaleMode]

Daten

keine weiteren Besonderheiten

Ereignisse

Es fehlen die Ereignisse **Modifiziert**, **Text modifiziert**, **Vor dem Aktualisieren** und **Nach dem Aktualisieren**.

Hinweis

Bilder werden durch das grafische Steuerelement nicht rotiert. Dies fällt bei Hochformataufnahmen auf, die die entsprechende Information in ihrem EXIF-Eintrag haben. Die Bilder müssen also vorher mit einem Bildprogramm rotiert werden. Am besten geeignet für ganze Verzeichnisse ist hier wohl ein Werkzeug wie *https://imagemagick.org/* .

```
001 mogrify -auto-orient *.JPG
```

Dieser Befehl in einem Verzeichnis rotiert alle Bilder mit JPG-Endung automatisch nach der Information im EXIF-Eintrag und ändert den EXIF-Eintrag auf die neue Orientierung. Die Bilder werden also weiter einwandfrei auch in anderen Bildbetrachtungsprogrammen in Hochformat oder Querformat dargestellt.

Maskiertes Feld

Mittels einer Eingabemaske wird hier die Eingabe in das Feld gesteuert. Zeichen werden für eine bestimmte Position vorgegeben, einzugebende Zeichen in ihrer Eigenschaft festgelegt. Die vorgegebenen Zeichen werden dann mit abgespeichert.

Neben den bereits unter *Standardeinstellungen vieler Kontrollfelder* erklärten Eigenschaften gibt es die folgenden Besonderheiten:

Allgemein

Hier wird festgelegt, welche Inhalte eingegeben werden können. [EditMask]

Diese Ansicht wird dem Formularnutzer präsentiert. [LiteralMask]

Ist die Formatüberprüfung eingeschaltet, so wirkt sie direkt bei der Eingabe. Ansonsten sind erst einmal Falscheingaben möglich, die beim Verlassen des Feldes korrigiert werden. [StrictFormat]

Der folgende Inhalt entstammt der LibreOffice-Hilfe:

Die Länge der Eingabemaske bestimmt, wie viele Stellen die mögliche Eingabe haben darf. Sollten die vom Benutzer eingegebenen Zeichen der Eingabemaske nicht entsprechen, so wird die Eingabe beim Verlassen des Kontrollfelds verworfen. Zur Definition der Eingabemaske stehen die folgenden Zeichen zur Verfügung:

Zeichen	***Bedeutung***
L	Eine Textkonstante. Diese Stelle kann nicht editiert werden. Das Zeichen wird an der entsprechenden Position der Zeichenmaske angezeigt.
a	Es können die Buchstaben a-z und A-Z eingegeben werden. Großbuchstaben werden nicht in Kleinbuchstaben konvertiert.
A	Es können die Buchstaben A-Z eingegeben werden. Eingegebene Kleinbuchstaben werden automatisch in Großbuchstaben konvertiert.
c	Es können die Buchstaben a-z und A-Z sowie die Ziffern 0-9 eingegeben werden. Großbuchstaben werden nicht in Kleinbuchstaben konvertiert.
C	An dieser Stelle können die Zeichen A-Z und 0-9 angegeben werden. Wird ein Kleinbuchstabe angegeben, wird automatisch in Großschrift umgewandelt.
N	Es können nur die Zeichen 0-9 angegeben werden.
x	Es können alle druckbaren Zeichen angegeben werden.
X	Es können alle druckbaren Zeichen angegeben werden. Wird dabei ein Kleinbuchstabe verwendet, wird automatisch in Großschrift umgewandelt.

Definieren Sie z. B. für die Zeichenmaske «__.__.2000» die Eingabemaske «NNLNNLLLLL», um dem Benutzer nur noch die Eingabe von 4 Ziffern zur Datumsangabe zu ermöglichen. Schalten sie unbedingt die Formatüberprüfung des Feldes ein.

Daten

keine weiteren Besonderheiten

Ereignisse

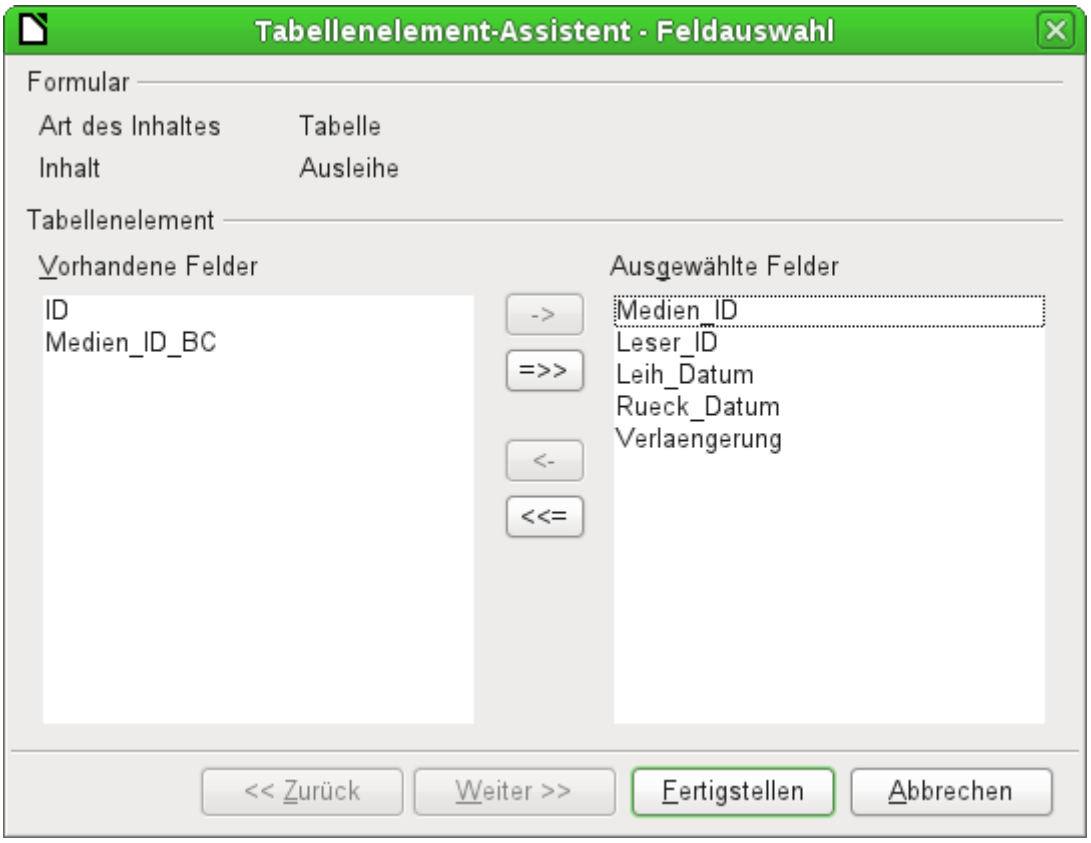
Es fehlt das Ereignis «Modifiziert»

Tabellen-Kontrollfeld

Dieses Kontrollfeld ist das umfassendste Feld. Es stellt eine Tabelle bereit, die wiederum durch Kontrollfelder für die einzelnen Spalten beschickt werden kann. Damit ist bei der Eingabe nicht nur der Blick auf die aktuell einzugebenden Daten, sondern auch auf vorher eingegebene Daten möglich, ohne mit der Navigationsleiste durch die Datensätze zu scrollen.

Nicht alle Felder, die auch im Formular möglich sind, können im Tabellen-Kontrollfeld wieder ausgewählt werden. So fehlen z. B. Schaltflächen, das grafische Kontrollfeld und das Optionsfeld.

Ein Assistent für das Tabellen-Kontrollfeld stellt in einem Fenster die Felder zusammen, die anschließend in der Tabelle zur Verfügung stehen sollen.



In dem Kontrollfeld soll die Tabelle "Ausleihe" zur Bearbeitung zur Verfügung stehen. Bis auf die Felder "ID" (Primärschlüssel) und das Feld "Medien_ID_BC" (Eingabe der Medien über einen Barcodescanner) sollen alle Felder in dem Kontrollelement benutzt werden.

Medien_ID	Leser_ID	Leih_Datum	Rueck_Datum	Verlaengerung
1,00	0,00	02.11.11	04.11.11	
2,00	2,00	15.10.11	25.02.12	2,00
0,00	3,00	02.11.11	04.04.12	1,00
3,00	0,00	04.11.11	28.11.11	2,00
5,00	0,00	28.11.11		
4,00	0,00	28.11.11	04.04.12	
4,00	0,00	09.11.11		
3,00	0,00	09.12.11		

Datensatz 1 von 18

Abbildung 26: Ergebnis des Tabellen-Kontrollfeld-Assistenten

Das erst einmal erstellte Tabellen-Kontrollelement muss hier noch weiter bearbeitet werden, da ja nur die Eingaben der Tabelle "Ausleihe" ermöglicht werden. Für Felder wie "Leser_ID" oder "Medien_ID" wäre es aber sinnvoller, statt einer Nummer den Leser bzw. das Medium selbst

auswählen zu können. Hierzu werden innerhalb des Tabellen-Kontrollelementes z. B. Listenfelder eingesetzt. Entsprechendes ist weiter unten beschrieben. Auch die Formatierung des Feldes "Verlängerung" mit zwei Nachkommastellen ist sicher nicht gewollt.

Hinweis

Wird ein Tabellenkontrollfeld ohne eingeschalteten Assistenten aufgezogen, so fehlen erst einmal die Spaltenköpfe. Die Spaltenköpfe können anschließend über einen rechten Mausklick auf die Kopfleiste (Position, an der die Spaltenköpfe erscheinen sollen) mit Hilfe eines Kontextmenüs erstellt werden.

Neben den bereits unter *Standardeinstellungen vieler Kontrollfelder* erklärten Eigenschaften gibt es die folgenden Besonderheiten:

Allgemein

In welcher Höhe sollen die einzelnen Zeilen angezeigt werden. Ohne einen Wert wird die Höhe automatisch der Schriftgröße angepasst. Mehrzeilige Textfelder würden so nur einzeilig zum Scrollen angezeigt.
[RowHeight]

Wie bei Tabellen werden am unteren Rand des Kontrollfeldes die Datensatznummer und Navigationshilfen angezeigt.
[HasNavigationBar]

Am linken Rand des Kontrollfeldes befindet sich standardmäßig der Datensatzmarkierer. Er zeigt den aktuellen Datensatz an. Über den Datensatzmarkierer wird die Löschfunktion für den gesamten Datensatz zur Verfügung gestellt.
[HasRecordMarker]

Daten

Da es sich um ein Feld handelt, das selbst keine Daten anzeigt oder abspeichert, sondern stattdessen andere Felder verwaltet, die dies tun, gibt es keine Eigenschaft Daten.

Ereignisse

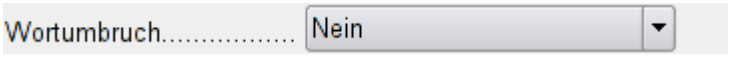
Es fehlt das Ereignis **Modifiziert** und **Text modifiziert**. Das Ereignis **Fehler aufgetreten** kommt hinzu.

Beschriftungsfeld

Neben den bereits unter *Standardeinstellungen vieler Kontrollfelder* erklärten Eigenschaften gibt es die folgenden Besonderheiten:

Allgemein

Der Titel dient als Beschriftung für ein anderes Kontrollfeld. Wird der Titel mit einem Kontrollfeld verbunden, dann kann er auch als Sprungziel für dieses Kontrollfeld festgelegt werden. Mit einem eingefügten «~» kann ein bestimmter Buchstabe des Titels als Sprungziel angegeben werden. «Na~me» definiert das «m» als Sprungziel. Befindet sich der Cursor in dem Formular, so wird die Schaltfläche über *Alt* + *m* erreicht.
[Label]



Standardmäßig wird die Beschriftung nicht umbrochen. Ein zu langer Titel wird abgeschnitten, wenn das Feld nicht groß genug ist. Der Wortumbruch hat allerdings keine Trennfunktion, so dass bei zu kleinen Feldern innerhalb eines Wortes ein Umbruch erfolgt.
[MultiLine]

Daten

keine

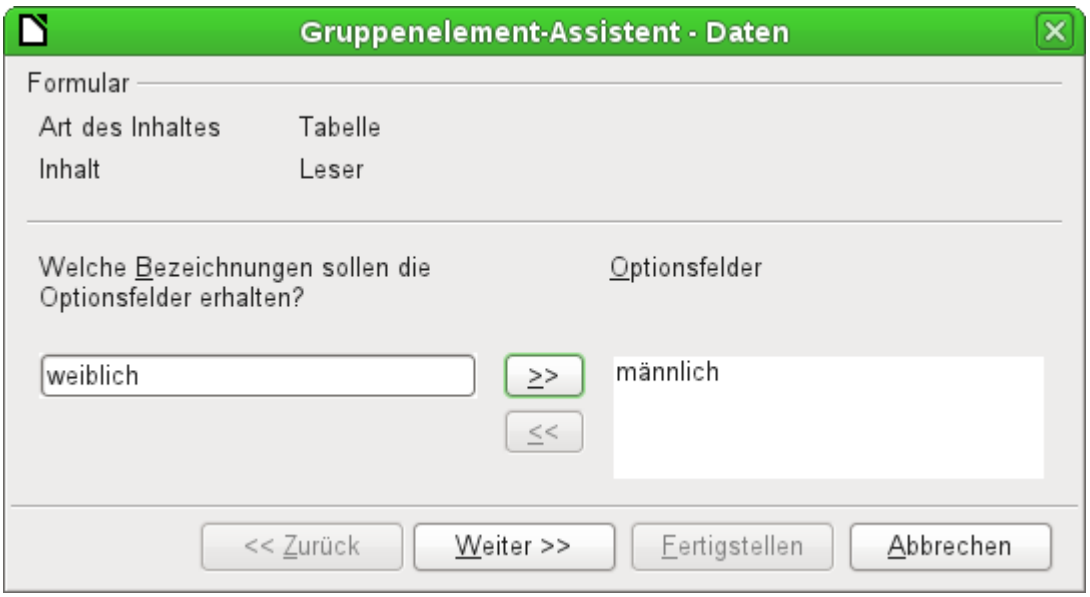
Ereignisse

Selbst das Beschriftungsfeld reagiert noch auf Ereignisse, die mit der Maus einer Taste oder dem Fokuserhalt zusammenhängen.

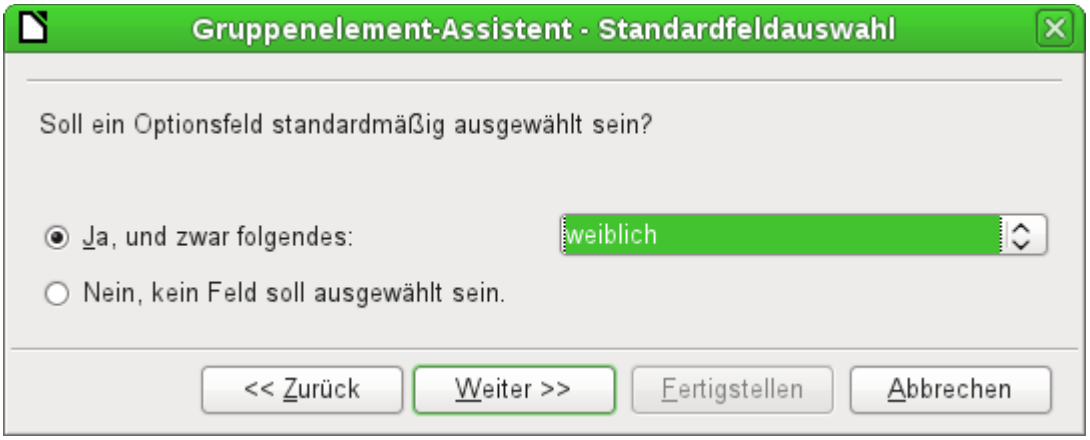
Gruppierungsrahmen

Der Gruppierungsrahmen fasst lediglich grafisch mehrere Kontrollfelder zu einer Gruppe zusammen und versieht sie mit einem zusätzlich grafisch eingebundenen Titel.

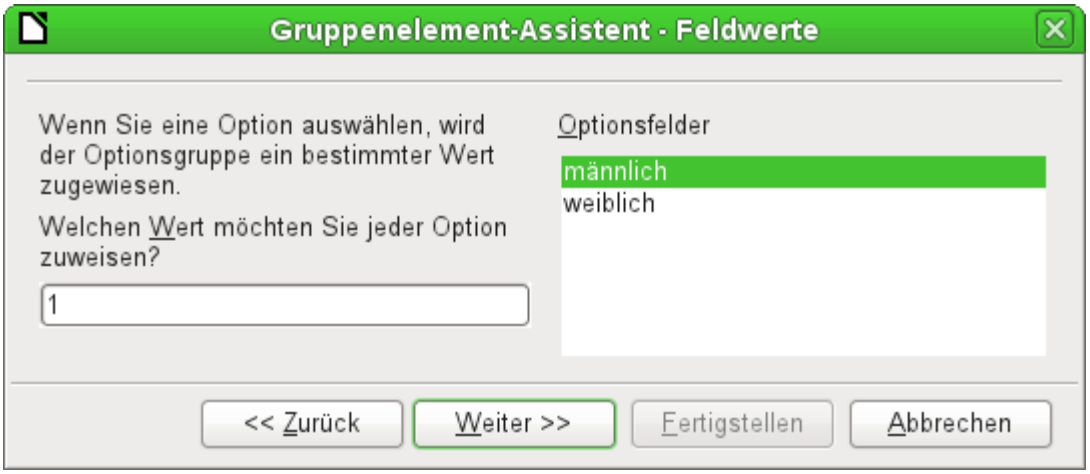
Wird ein Gruppierungsrahmen mit eingeschaltetem Assistenten aufgezogen, so geht der Assistent davon aus, dass mehrere Optionsfelder zusammen in diesem Rahmen erscheinen sollen.



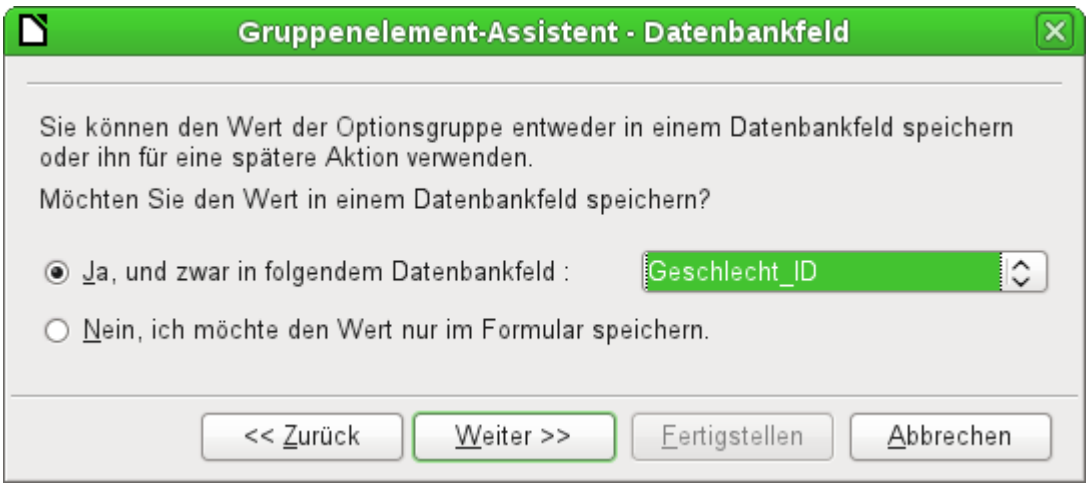
Dem Formular liegt die Tabelle Leser zugrunde. Hier wird gerade die Auswahl des Geschlechtes zusammengestellt. Die Einträge sind anschließend die Beschriftungen der Optionsfelder.



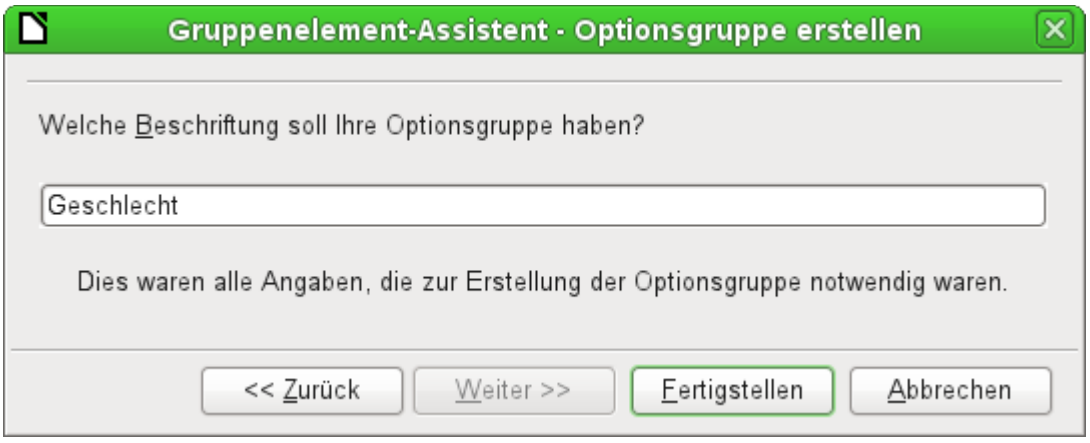
Standardmäßig wird hier vorgewählt, dass das Geschlecht «weiblich» angewählt wird. Wird kein Feld vorgewählt, so ist der Standardeintrag in die zugrundeliegende Tabelle *NULL*.



Den Optionsfeldern wird bereits standardmäßig durch den Assistenten ein jeweils separater Wert zugeteilt. Da dieser Wert in unserem Fall nicht dem des Primärschlüssels entspricht, wird er geändert, und zwar für «männlich» auf den Wert 'm', für «weiblich» auf den Wert 'w'. Diese Werte entsprechen in dem Beispiel den Primärschlüsselfeldern der Tabelle "Geschlecht"



Der jeweils in den Optionsfeldern angeklickte Wert wird an das Feld "Geschlecht_ID" der dem Formular zugrundeliegenden Tabelle "Leser" übertragen. Die Tabelle "Leser" wird also über ein Optionsfeld mit den entsprechenden Fremdschlüsseln der Tabelle "Geschlecht" versorgt.



Die Gruppe der Optionsfelder erhält einen Rahmen, in den die Beschriftung 'Geschlecht' eingefügt wird.

Wird jetzt bei aktivem Formular 'männlich' ausgewählt, so wird gleichzeitig 'weiblich' abgewählt. Dies ist ein Kennzeichen der Optionsfelder, wenn sie mit gleichem Namen versehen und mit dem gleichen Feld der dem Formular zugrundeliegenden Tabelle verbunden sind. In dem obigen Beispiel ersetzen die Optionsfelder so ein Listenfeld mit zwei Einträgen.

Neben den bereits unter *Standardeinstellungen vieler Kontrollfelder* erklärten Eigenschaften gibt es die folgenden Besonderheiten:

Allgemein

Der Titel kann neben den Standardeinstellungen verändert werden. Eine Veränderung des Rahmens (Liniendicke, Linienfarbe) ist zur Zeit nicht vorgesehen, wohl aber eine Formatierung der Schrift.

Daten

Keine, fasst schließlich Eingabefelder nur optisch zusammen.

Ereignisse

Auch der Gruppierungsrahmen reagiert noch auf Ereignisse, die mit der Maus einer Taste oder dem Fokuserhalt zusammenhängen.

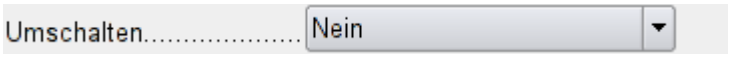
Schaltfläche

Neben den bereits unter *Standardeinstellungen vieler Kontrollfelder* erklärten Eigenschaften gibt es die folgenden Besonderheiten:

Allgemein

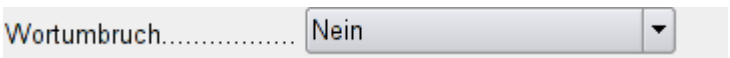
Beschriftung auf der Schaltfläche. Mit einem eingefügten «~» kann ein bestimmter Buchstabe des Titels als Sprungziel angegeben werden. «Schalt~fläche» definiert das «f» als Sprungziel. Befindet sich der Cursor in dem Formular, so wird die Schaltfläche über *Alt* + *f* erreicht.
[Label]

Die Schaltfläche erhält den Fokus, wenn auf sie geklickt wird.
Dies kann manchmal nicht erwünscht sein, wenn z.B. über die Schaltfläche Inhalt in ein anderes Formularfeld eingefügt werden soll. Dann soll bei Betätigung der Schaltfläche der Cursor in dem anderen Feld stehen bleiben.
[FocusOnClick]



Wenn 'Ja', so kann die Schaltfläche auch eingedrückt dargestellt werden. Der Schaltzustand wird wie bei einem Schalter angezeigt – im Gegensatz zu einem Taster wie sonst bei Buttons.
[Toggle]

Aktiv, wenn Umschalten auf 'Ja' gesetzt ist. Ausgewählt ist dann der eingedrückte Zustand.
[DefaultState]

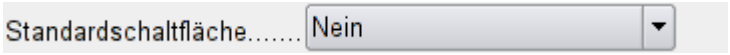


Wortumbruch, wenn die Schaltfläche zu schmal ist.
[MultiLine]

Verschiedene Aktionen, auch zusätzlich zu denen der Navigationsleiste, stehen zur Verfügung:
'Formular übertragen'
'Formular zurücksetzen'
'Dokument/Webseite öffnen'
'Erster/vorheriger/nächster/letzter Datensatz'
'Datensatz speichern'
'Dateneingabe rückgängig machen'
'Neuer Datensatz'
'Datensatz löschen'
'Formular aktualisieren'

'HTML': Datei, die hiermit aufgerufen werden soll. Hierzu muss unter **Aktion → Dokument/Webseite öffnen** gewählt werden.
Neben HTML kann dies auch genutzt werden, um bestimmte Programmmodule von LO aufzurufen. So wird mit dem hier einzutragenden Befehl **`.uno:RecSearch`** z. B. die Suchfunktion aus dem Formularnavigator auf einen Button gelegt.
Außerdem lassen sich auch z. B. Writerdateien öffnen, die anschließend per Druck einen mit der Datenbank zusammenhängenden Serienbrief ausgeben.
Siehe auch *Einige uno-Befehle zur Nutzung mit einer Schaltfläche* im Anhang des Handbuchs.
[TargetURL]

Nur für HTML-Formulare: Der Ziel-Frame (Rahmeneinteilung für verschiedene HTML-Seiten), in dem die Datei geöffnet werden soll.
[TargetFrame]



Schaltfläche wird bei 'Ja' zusätzlich umrandet. Bei mehreren alternativen Schaltflächen auf einem Formular soll dies die übliche Schaltfläche kennzeichnen. Sie wird durch Betätigung der Eingabetaste ausgelöst, wenn keine andere Aktion gerade auf die Eingabetaste reagieren muss. Nur eine Schaltfläche sollte im Formular die Standardschaltfläche sein.
[DefaultButton]

Soll eine Grafik auf der Schaltfläche erscheinen?
[Graphic] [ImageURL]

Nur aktiv, wenn eine Grafik ausgewählt wurde. Die Grafik wird im Verhältnis zum Text angeordnet.
[ImagePosition] [ImageAlign]

Daten
Keine, es werden nur Aktionen ausgeführt.

Ereignisse
Aktion bestätigen, **Aktion ausführen** und **Statusänderung**

Grafische Schaltfläche

Neben den bereits unter *Eigenschaften der Kontrollfelder* erklärten Eigenschaften gibt es die folgenden Besonderheiten:

Allgemein
Entspricht der normalen Schaltfläche. Allerdings wird diese Schaltfläche ohne Text und ohne einen sichtbaren Button dargestellt. Lediglich eine Rahmen um die Grafik kann eingestellt werden.

Eine grafische Schaltfläche erhält standardmäßig erst einmal keinen Tabstop.

Hinweis

Bereits in OpenOffice funktionierte diese Schaltfläche nicht zusammen mit den Aktionen, die auch der normalen Schaltfläche zugeordnet werden können. Die Schaltfläche lässt sich aber so nutzen, dass z. B. unter **Allgemein → Aktion → Dokument/Webeite öffnen** und anschließend unter **Allgemein → URL** `.uno:NextRecord` angegeben wird, Damit ist ein Sprung zum nächsten Datensatz möglich. Siehe dazu auch *Einige uno-Befehle zur Nutzung mit einer Schaltfläche* im Anhang des Handbuchs.
Ansonsten ist die Schaltfläche nur mit Makros nutzbar.

Daten
Keine, es werden nur Aktionen ausgeführt.

Ereignisse
Aktion bestätigen sowie alle Ereignisse, die die Maus, eine Taste oder den Fokus betreffen.

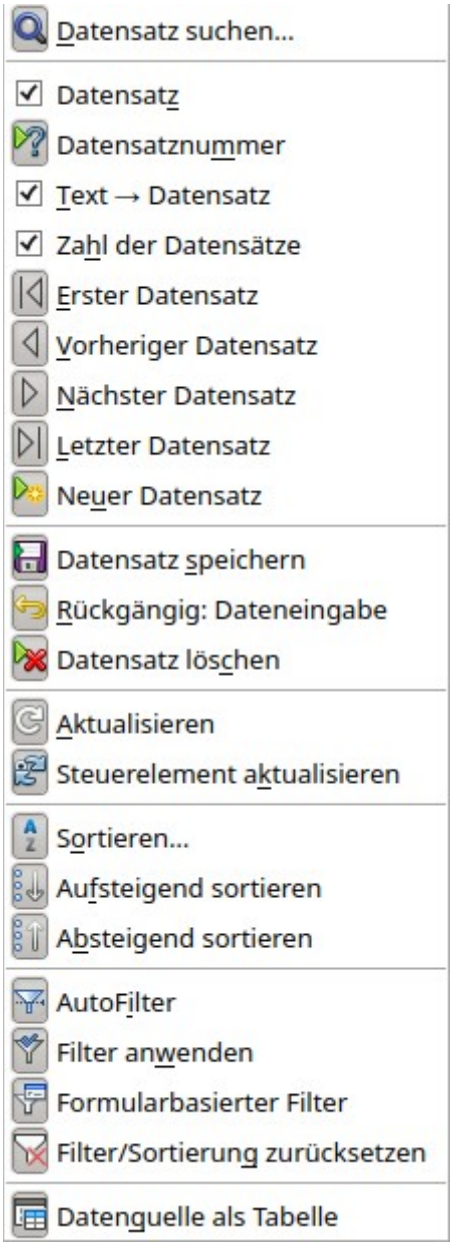
Navigationsleiste

Abbildung 27: Formularkontrollelement Navigationsleiste

Die Standardnavigationsleiste wird beim Formular am unteren Bildrand eingeblendet. Durch das Einblenden dieser Leiste kann es beim Bildschirmaufbau zu einem kurzen Zurechtschieben des Formulars kommen. Dies ist vor allem dann störend, wenn die Navigationsleiste eventuell in Teilen des sichtbaren Formulars ausgeschaltet ist.

Eine separat zu den jeweiligen Elementen eines Formulars angeordnete Navigationsleiste macht dagegen klar, durch welche Elemente mit der Leiste navigiert wird.

Die Standardnavigationsleiste bietet etwas mehr Elemente als das Formularkontrollelement Navigationsleiste. Die folgende Übersicht zeigt den gesamten Umfang der Standardnavigationsleiste.



Die Such- und Filterfunktionen werden zum Schluss des Kapitels erklärt.

Die Navigation durch die Datensatznummern dürfte von der Bedienung von Multimedia-Geräten bekannt sein. Wird bei der Datensatznummer eine konkrete Nummer eingegeben und mit der *Eingabetaste* bestätigt, so bedeutet dies einen Sprung zum entsprechenden Datensatz.

Beim Aktualisieren gibt es auch den Button Steuerelement aktualisieren. Werden Inhalte von Listenfeldern durch ein anderes Formular beeinflusst, so muss der Cursor in das Listenfeld gesetzt werden und der Inhalt kann über diesen Button neu eingelesen werden.

Neben den bereits unter *Standardeinstellungen vieler Kontrollfelder* erklärten Eigenschaften gibt es die folgenden Besonderheiten:

Allgemein

Die Größe der Symbole kann eingestellt werden. Außerdem ist auswählbar, welche Gruppen angezeigt werden. Dies sind in *Abbildung 27* von links nach rechts, bei den Symbolen durch eine senkrechte Linie getrennt, die Positionierung, die Navigation, die Datensatzaktionen sowie Filterung / Sortierung.
[IconSize]
[ShowPosition]
[ShowNavigation]
[ShowRecordActions]
[ShowFilterSort]

Daten

Keine, es werden nur Aktionen ausgeführt.

Ereignisse

Alle Ereignisse, die die Maus, eine Taste oder den Fokus betreffen.

Unabhängig von diesem Formularkontrollelement existiert natürlich auch die unten **einblendbare Navigationsleiste** mit den Elementen der obigen Abbildung.

Diese einblendbare Navigationsleiste bietet neben den Elementen des entsprechenden Formularkontrollelementes noch die allgemeine Datensatzsuche, den formularbasierten Filter und die Darstellung der dem Formular zugrundeliegenden Datenquelle in der Tabellenansicht oberhalb des Formulars. Im Gegensatz zu einer einfachen Tabellenansicht des Formularinhaltes übernimmt diese Tabellenansicht die Definition von Listenfeldern aus dem Formular. Sie ist daher eher vergleichbar mit einem fertig gestalteten Tabellen-Kontrollfeld.

Wird nicht nur mit einem Formular, sondern mit Unterformularen und Nebenformularen gearbeitet, so ist darauf zu achten, dass diese einblendbare Navigationsleiste nicht beim Formularwechsel je nach Einstellung verschwindet. Das erzeugt nur Unruhe auf dem Bildschirm.

Drehfeld und Bildlaufleiste

Beide Felder haben keine direkte Verbindung zu Daten des Formulars. Sie lassen sich nur über Makros entsprechend nutzen, wobei das Drehfeld von der Anwendung her bereits z.B. in das Zahlenfeld integriert ist.

Die Bildlaufleiste unterscheidet sich von dem Drehfeld dadurch, dass sie neben den Buttons für steigende und fallende Werte innerhalb einer vorher festgelegten Grenze auch einen Schiebeknopf haben. Der Name «Bildlaufleiste» ist dabei etwas irritierend, da über dieses Schiebereggler nicht irgendwelche Bilder verschoben werden, sondern lediglich Werte in einem Feld verändert werden können.

Beim Start eines Formulars muss der Bildlaufleiste zuerst mitgeteilt werden, welchen Wert das Feld hat, das mit ihr verbunden sein soll. Dieses Makro ist an die **Formular-Eigenschaften → Ereignisse → Nach dem Datensatzwechsel** gebunden.

```
001  SUB Bildlaufleiste_Formular(oEvent AS OBJECT)
002     DIM oForm AS OBJECT
003     DIM oFeld AS OBJECT
004     oForm = oEvent.Source
005     oFeld = oForm.getByName("Bildlaufleiste")
006     oFeld.ScrollValue = oForm.getInt(oForm.findColumn("Zahlenwert"))
007  END SUB
```

Zuerst werden die Variablen deklariert. Das auslösende Ereignis weist dabei auf das Formular hin, da das Makro schließlich an das Formular gebunden wurde. Die Bildlaufleiste wird nach dem entsprechenden Namen in dem Formular gefunden. Der Wert der Bildlaufleiste wird auf den Wert eingestellt, der in dem Feld "Zahlenwert" der dem Formular zugrundeliegenden Tabelle liegt.

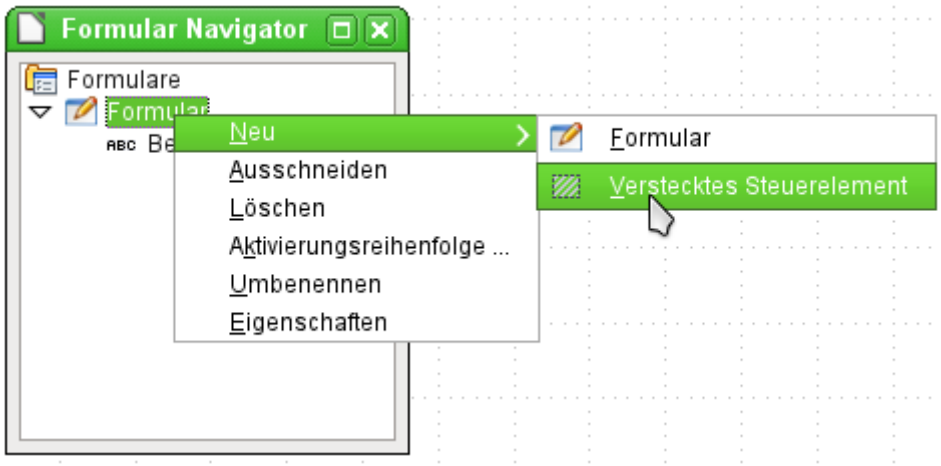
Wird der Wert der Bildlaufleiste geändert, so muss entsprechend der Wert in dem verbundenen Feld geändert werden. Dieses Makro wird an **Eigenschaften: Bildlaufleiste → Ereignisse → Beim Justieren** gebunden

```
001 SUB Bildlaufleiste_Aenderung(oEvent AS OBJECT)
002    DIM oForm AS OBJECT
003    DIM oFeld AS OBJECT
004    DIM inValue AS INTEGER
005    oFeld = oEvent.Source.Model
006    inValue = oEvent.Value
007    oForm = oFeld.Parent
008    oForm.updateInt(oForm.findColumn("Zahlenwert"),inValue)
009 END SUB
```

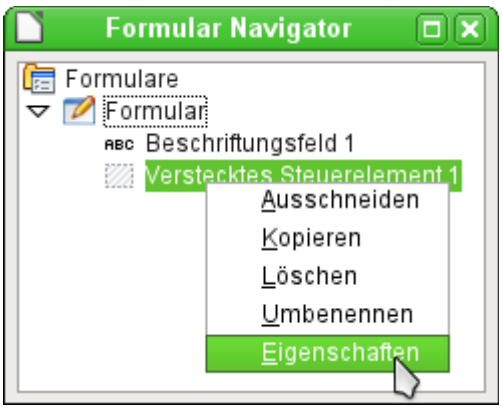
Zuerst werden die Variablen deklariert. Aus dem Feld wird der gewählte Wert als Ganzzahlenwert ausgelesen. Das dem Formular zugrundeliegende Tabellenfeld "Zahlenwert" wird auf den entsprechenden Wert eingestellt.

Details zu einem Code wie diesem sind in dem Kapitel «Makros» zu finden.

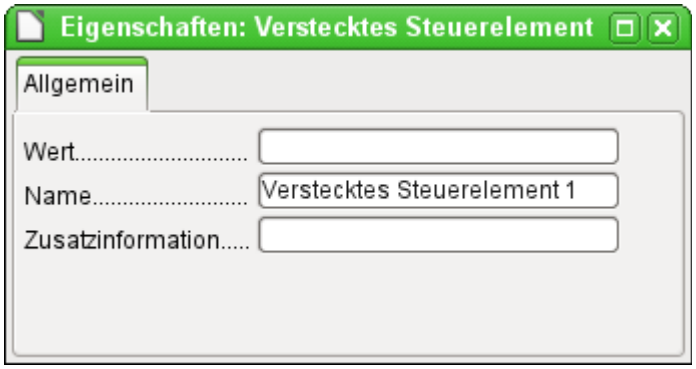
Verstecktes Steuerelement



Ein Kontrollfeld, das nicht über die Symbolleisten einfügbar ist, ist das versteckte Steuerelement. Es wird über den Formularnavigator im Kontextmenü eines Formulars neu erstellt.



Das versteckte Steuerelement erscheint wie jedes andere Kontrollfeld als Teil des Formulars. Es ist auf der Benutzeroberfläche nicht sichtbar.



Die kompletten Eigenschaften des versteckten Steuerelementes sind recht spärlich. Der Name und die Zusatzinformationen sind von den anderen Steuerelemente bekannt. Zusätzlich kann hier ein Wert angegeben werden. [HiddenValue]

Für die Nutzung ohne Makros ist dies sicher nicht weiter von Bedeutung. Bei der Arbeit mit Makros kann es aber sinnvoll sein, an einer Stelle des Formular Werte zwischenspeichern zu können, um auf sie bei einer weiteren Aktion wieder zugreifen zu können. Ein entsprechendes Anwendungsbeispiel findet sich bei den Makros im Kapitel *Hierarchische Listenfelder*.

Mehrfachselektion

Wird mit dem Auswahlpfeil (siehe *Abbildung 23*) ein größerer Bereich oder sogar sämtliche Elemente eines Formulars ausgewählt, so sind über diese Mehrfachselektion die Änderungen nach *Abbildung 28* möglich.

Abbildung 28: Allgemeine Eigenschaften von Formularfeldern in der Mehrfachselektion

Die Namen sollten hier möglichst nicht geändert werden. Schließlich lauten so plötzlich alle Elemente gleich. Das Auffinden eines einzelnen Elementes über den Formularnavigator wird erschwert, die Verarbeitung des Formulars nach Namensbezeichnung der Kontrollfelder in Makros unmöglich.

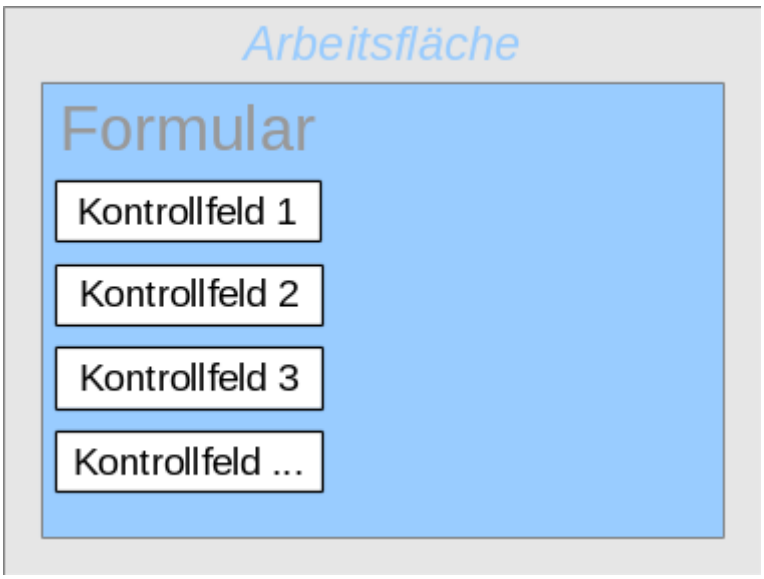
Eher bietet sich eine Mehrfachselektion an, wenn insgesamt die Schriftart, die Höhe oder die Hintergrundfarbe gewechselt werden soll. Allerdings wirkt sich natürlich die Hintergrundfarbe auch auf die Beschriftungsfelder aus.

Sollen z. B. nur die Beschriftungsfelder geändert werden, so bietet es sich an, bei gedrückter *Umschalt*-Taste mit der linken Maustaste diese Felder direkt oder im Navigator anzuklicken, mit der rechten Maustaste über eins der Felder zu gehen und die Kontrollfeldeigenschaften aufzurufen. Jetzt wird die Auswahl der änderbaren Eigenschaften größer, da ja nur gleiche Felder ausgewählt wurden. Entsprechend kann hier auch all das geändert werden, was sonst in einem Beschriftungsfeld zur Verfügung steht.

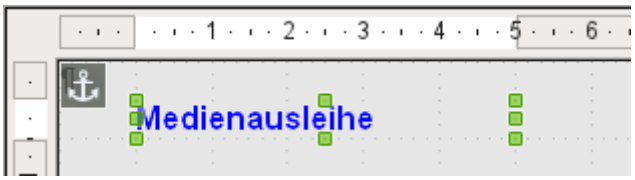
Die Möglichkeiten der Mehrfachselektion richten sich also nach der Auswahl der Felder. Kontrollfelder der gleichen Art können in allen allgemeinen Eigenschaften, die das einzelne Kontrollfeld bietet, auf diese Art gemeinsam angepasst werden.

Einfaches Formular komplett erstellt

Ein einfaches Formular stellt Formularkontrollfelder für das Schreiben oder Lesen von Datensätzen aus einer einzigen Tabelle oder Abfrage zur Verfügung. Seine Konstruktion entspricht dem folgenden Schaubild:



Am Beispiel eines einfachen Formulars für die Bibliotheksausleihe sollen hier verschiedenen Varianten eines Formulars vorgestellt werden.



Die Überschrift für das Formular wurde über ein Beschriftungsfeld erzeugt. Die Schrift wurde geändert. Das Beschriftungsfeld ist am Absatz verankert, der sich in dem Dokument an der linken oberen Ecke befindet. Über das Kontextmenü des Beschriftungsfeldes wurde ein Formular erzeugt, das mit der Tabelle Ausleihe verbunden wurde (siehe: *Formulargründung über ein Formularfeld*). Die Seite wurde außerdem mit einem einfarbigen Hintergrund versehen.

Tipp

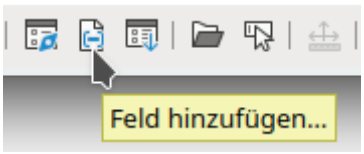
Das Formular sollte grundsätzlich nur aus Formularfeldern erstellt werden. Text gehört in die Beschriftungsfelder. Die Grundeinstellung ist, dass alle Felder an dem Absatz der linken oberen Ecke verankert werden.
Werden Zeilenumbrüche auf der Hintergrundseite eingefügt, so kann dies bei einem Wechsel des Zeichensatzes des Systems zu Verschiebungen der Formularelemente führen. Die Formularelemente sind dann nämlich an den nächstliegenden Absatz gebunden, nicht an die linke obere Ecke des Formularhintergrundes.
Die **Verankerung** am Absatz kann auch dazu führen, dass mit dem Löschen des Absatzes alle daran hängenden Formularfelder entfernt werden. Besser wäre, die Verankerung für alle Felder auf **An der Seite** einzustellen.

Hinweis

Das Einfügen von Bildern im Hintergrund kann dazu führen, dass die *.odb-Datei unnötig groß wird. Die Bilder werden nämlich in dieser Datei gespeichert. Manchmal lassen sich die Bilder zwar aus dem Formular löschen, werden aber nicht aus der *.odb-Datei entfernt. Hier hilft dann nur ein Öffnen der Datei mit einem Packprogramm. Siehe die Beschreibung dazu im Anhang.

Felder als Gruppe hinzufügen

Eine schnelle Variante, direkt Felder mit Beschriftungen einzufügen, bietet die Funktion Feld hinzufügen.



Über diese auf der Symbolleiste Formular-Entwurf (siehe *Abbildung 22*) erreichbaren Funktion lassen sich alle Felder der dem Formular zugrundeliegenden Tabelle auswählen

Über einen Doppelklick auf die Felder werden diese als Gruppierung zusammen mit einer Beschriftung immer in der Mitte des Formulars eingegliedert. Die Gruppen müssen also auf jeden Fall noch verschoben werden, damit anschließend das Formular im Einsatz wie das folgende Bild aussehen kann. Für die bessere Übersicht wurden alle unnötigen Leisten des Fensters entfernt und das Fenster entsprechend schmal zusammengeschoben, so dass nicht mehr alle Elemente der Navigationsleiste sichtbar sind. Es wurden alle Felder bis auf "Medien_ID_BC" ausgewählt, da dies speziell für die Bedienung mit einem Barcodescanner gedacht ist.

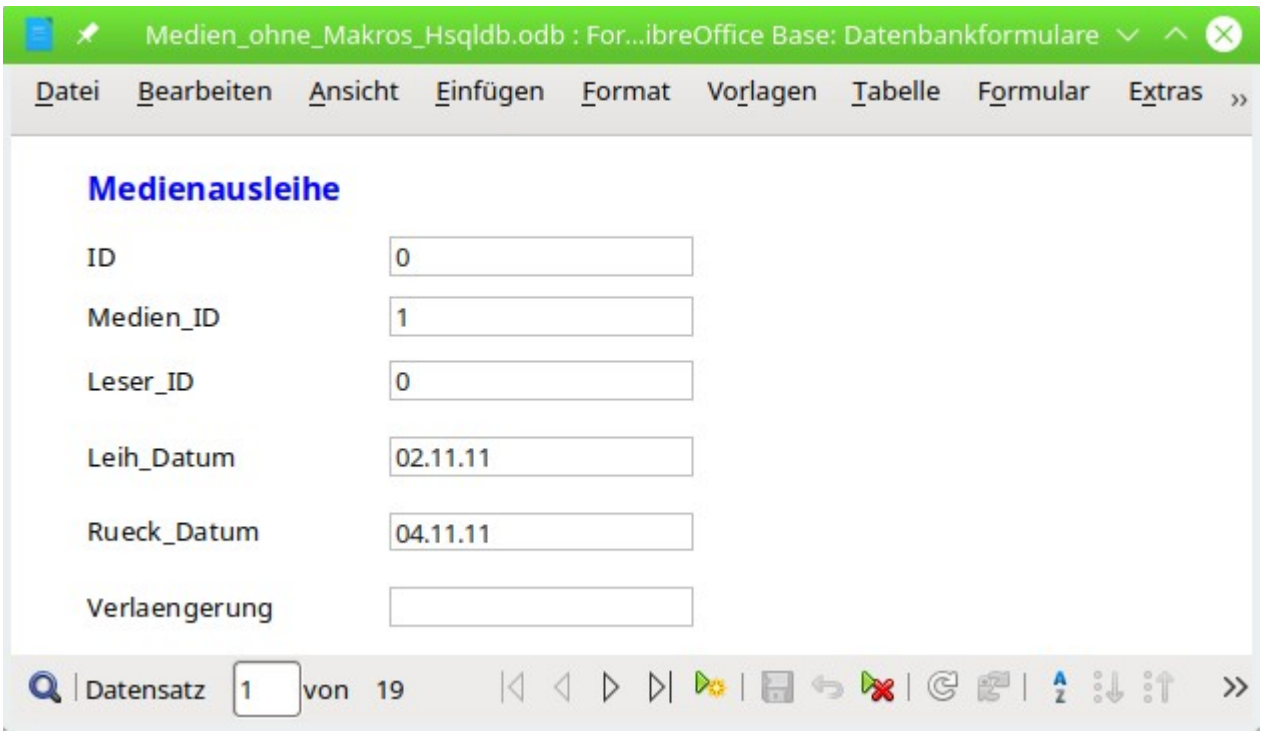


Abbildung 29: Einfaches Formular über «Feld hinzufügen"

Für alle Tabellenfelder wurden die korrekten Formularkontrollfelder ausgesucht. Zahlen werden in numerische Felder gesetzt und gleich als Ganzzahlen ohne Nachkommastellen erkannt. Datumsfelder werden ebenfalls korrekt als Datumsfelder wiedergegeben. Alle Felder werden in gleicher Breite dargestellt. Würde ein Grafisches Kontrollfeld eingesetzt, so würde hier ein quadratisches Feld erzeugt.

Felder anpassen

Gestalterisch kann jetzt einiges getan werden, indem die Felder von der Länge her angepasst werden und die Datumsfelder aufklappbar gemacht werden. Wichtiger ist aber, dass die Felder für die "Medien_ID" und die "Leser_ID" für den Nutzer verständlich werden – es sei denn, jeder Bibliotheksbesucher muss einen Ausweis mit der "ID" mitbringen und jedes Medium wird bei der Aufnahme mit der "ID" versehen. Hiervon wurde im Folgenden nicht ausgegangen.

Um einzelne Felder anzupassen, muss zuerst einmal die Gruppe betreten werden. Dies kann über einen rechten Mausklick auf die Gruppe und das entsprechende Kontextmenü erfolgen.

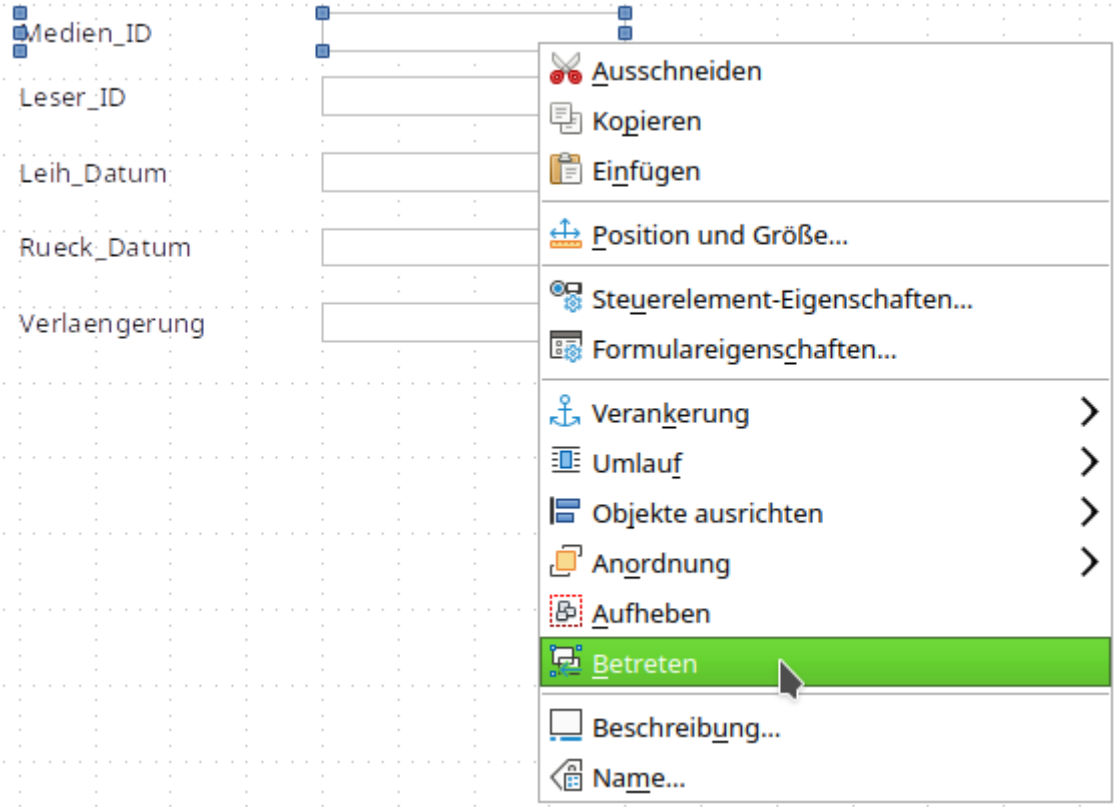
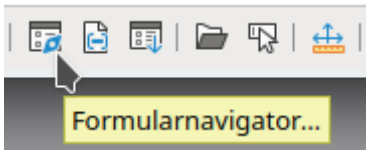


Abbildung 30: Formularkontrollelemente: Gruppe betreten

Die Gruppe kann auch direkt betreten werden, wenn die *Strg* mit einem Linksklick der Maus kombiniert wird. Mehrere Felder aus unterschiedlichen Gruppen werden mit *Strg* + *Shift* und einem Linksklick der Maus markiert.

Übersichtlicher für die späteren Verfahren ist allerdings die Arbeit mit dem Formularnavigator:



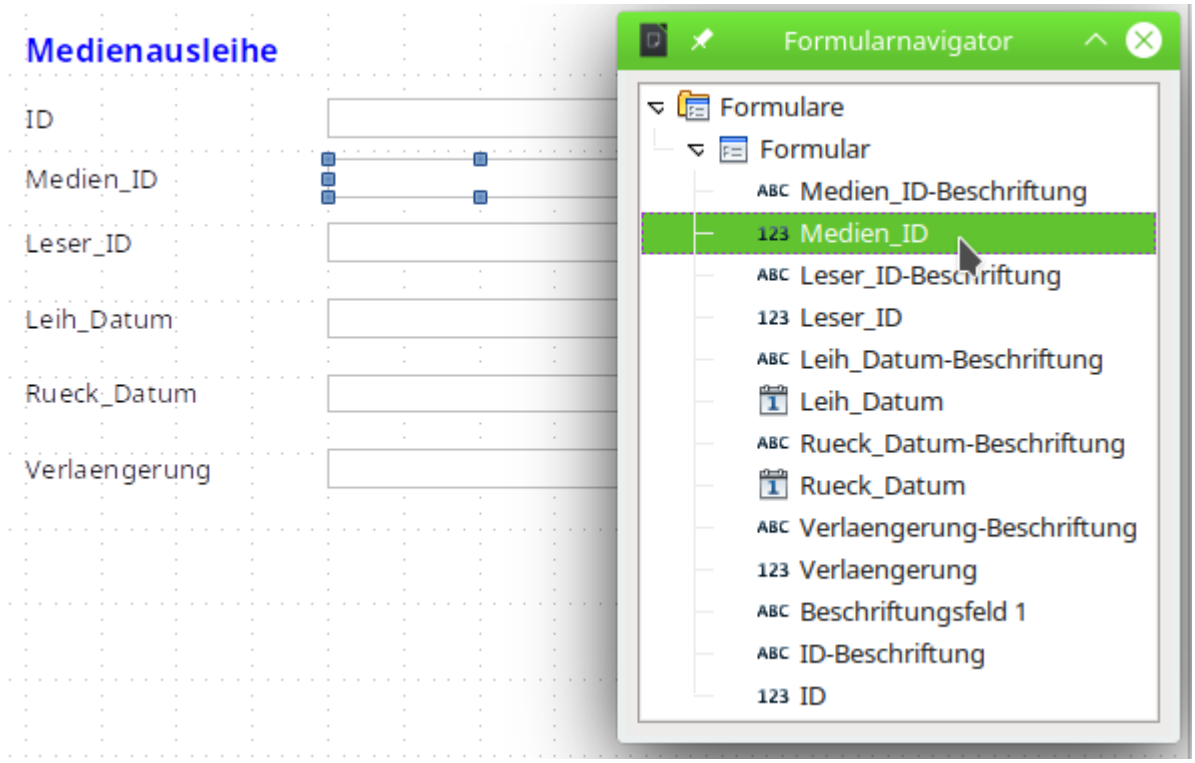


Abbildung 31: Formularkontrollelemente über den Formularnavigator direkt anwählen

Der Formularnavigator stellt alle Elemente des Formulars mit ihren Bezeichnungen dar. Für die Kontrollfelder wurden durch den Assistenten als Bezeichnungen direkt die Namen der Felder aus der dem Formular zugrundeliegenden Tabelle genommen. Die Beschriftungselemente haben den Zusatz «-Beschriftung».

Durch einen Klick auf «Medien_ID» ist dieses Feld ausgewählt. Mit einem Rechtsklick wird es über das Kontextmenü möglich, das ausgewählte Feld durch eine andere Feldart zu ersetzen:

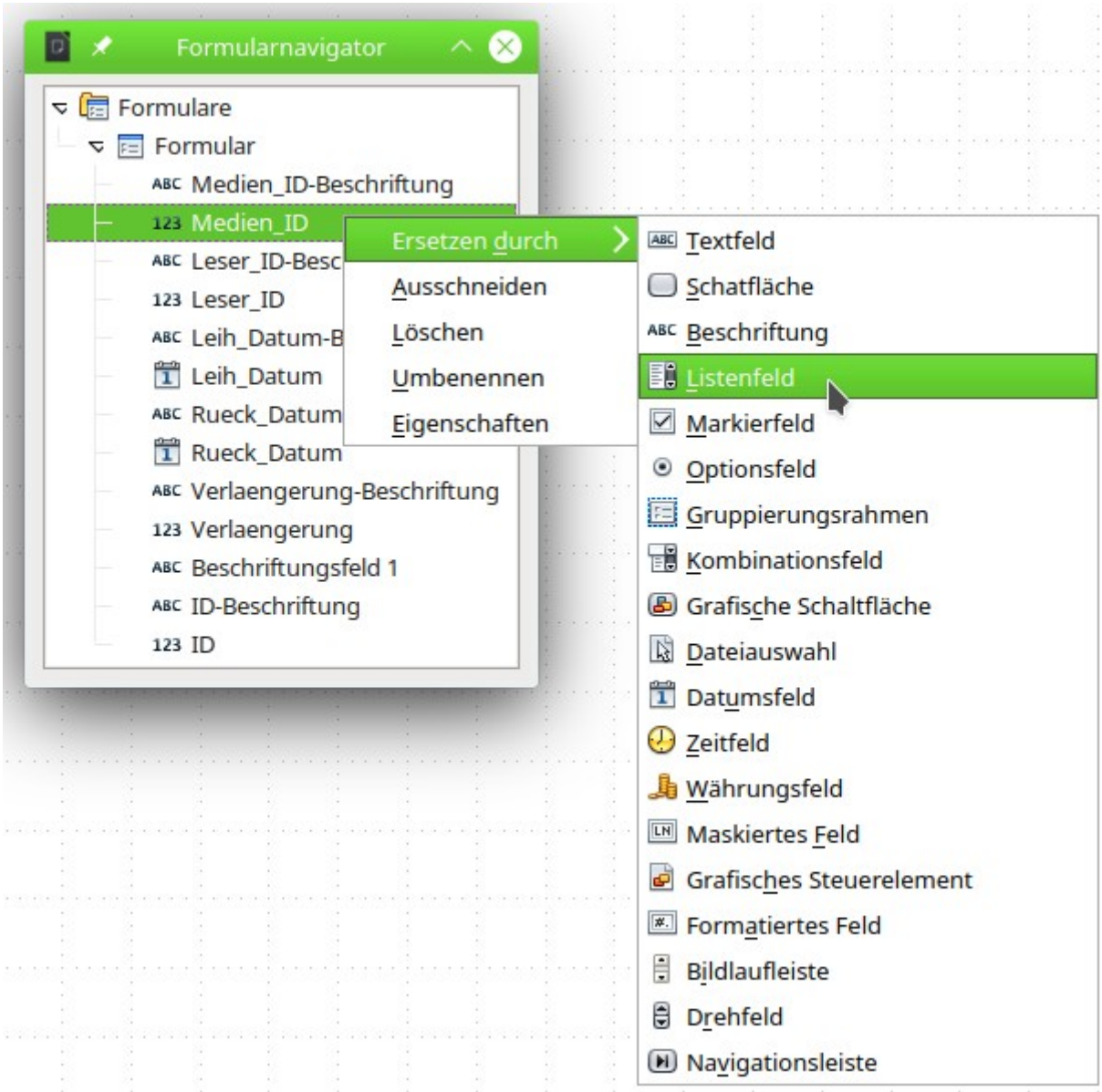
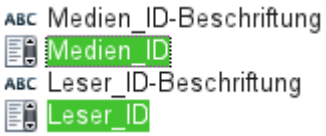


Abbildung 32: Kontrollelemente über den Formularnavigator durch andere Kontrollelemente ersetzen

Diese Ersetzung wird in dem Formular für «Medien_ID» und «Leser_ID» vorgenommen.



Die Änderung ist im Formularnavigator über die Symbole vor dem Namen des Kontrollfeldes sichtbar.

Die SQL-Abfragen für die Listenfelder können jetzt über den rechten Button mit der grafischen Benutzeroberfläche erstellt werden. Der Listenfeldassistent steht hier nicht zur Verfügung. Er springt nur automatisch ein, wenn ein Listenfeld direkt neu, nicht aber durch Umwandlung von einem anderen Feld aus, erstellt wird. Zum SQL-Befehl siehe das Kapitel *Abfragen für die Erstellung von Listenfeldern*.

Nachdem die Listenfelder über **Eigenschaften → Allgemein → Aufklappbar → 'Ja'** als aufklappbar angepasst wurden, können noch die folgenden Mängel behoben werden:

- Die Beschriftung der Listenfelder sollte auf «Medien» statt «Medien_ID» und «Leser» statt «Leser-ID» geändert werden.
- Das Kontrollfeld «ID» sollte als nicht beschreibbar erklärt werden.
- Alle Felder, die nicht beim Aufruf der Ausleihe auf jeden Fall durchlaufen werden müssen, wenn ein neues Medium ausgeliehen wird, benötigen keinen Tabstop. Ohne den Tabstop geht es schneller durch das Formular. Eventuell muss der Tabstop auch über die Aktivierungsreihenfolge (siehe *Abbildung 22*) nachjustiert werden. Lediglich die Felder «Medien», «Leser» und «Leihdatum» müssen für eine Ausleihe auf jeden Fall über den Tabulator erreichbar sein.
- Soll über das Formular die Ausleihe erfolgen, so ist es eigentlich unnötig und auch unübersichtlich, bereits zurückgegebene Medien angezeigt zu bekommen. Medien mit einem Rückgabedatum sollten ausgefiltert werden. Außerdem könnte die Reihenfolge der Anzeige nach dem Leser sortiert werden, damit Medien, die von der gleichen Person entliehen wurden, nacheinander aufgezeigt werden. Siehe dazu die Hinweise unter *Formular-Eigenschaften*. Speziell die Lesersortierung hat hier allerdings den Haken, dass lediglich nach der ID, nicht aber nach dem Alphabet sortiert werden kann, da die Tabelle für das Formular ja nur die ID enthält.

Felder einzeln hinzufügen

Das Hinzufügen einzelner Felder gestaltet sich zuerst einmal etwas aufwändiger. Die Felder müssen ausgewählt, auf der Formularfläche aufgezogen und dem Feld der dem Formular zugrundeliegenden Tabelle zugewiesen werden. Außerdem ist noch die Art des Feldes richtig einzustellen, da z. B. numerische Felder standardmäßig erst einmal 2 Nachkommastellen haben.

Beim Aufziehen der Listenfelder kommt der Assistent zum Einsatz, der die Schritte zur Erstellung eines korrekten Feldes für Ungeübte vereinfacht. Nach kurzer Anwendungsphase wird der Assistent allerdings den Ansprüchen nicht mehr gerecht, da er

- Die Einträge nicht automatisch sortiert.
- Eine Zusammenfassung von mehreren Feldern im aufzulistenden Inhalt nicht ermöglicht.

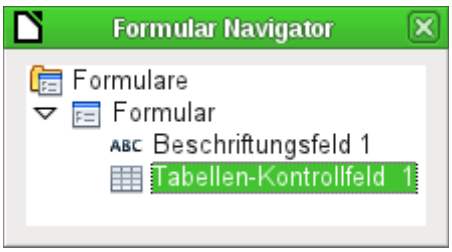
Hier muss dann immer wieder nachgebessert werden, so dass ganz schnell der SQL-Code direkt über den eingeblendeten Abfrageeditor erstellt wird.

Beim Hinzufügen einzelner Felder müsste eine Gruppierung von Feld und Beschriftung gesondert vorgenommen werden (siehe *Standardeinstellungen vieler Kontrollfelder*). In der Praxis kann sich aber die fehlende Verknüpfung auch positiv bemerkbar machen, da der Zugang zu den Eigenschaften eines Kontrollfeldes über das Kontextmenü sofort möglich ist und nicht erst ein Betreten der Gruppe erfordert. Fehlt die Verknüpfung, dann kann allerdings ein Beschriftungsfeld nicht mehr als *Sprungziel* für das entsprechende Eingabefeld genutzt werden.

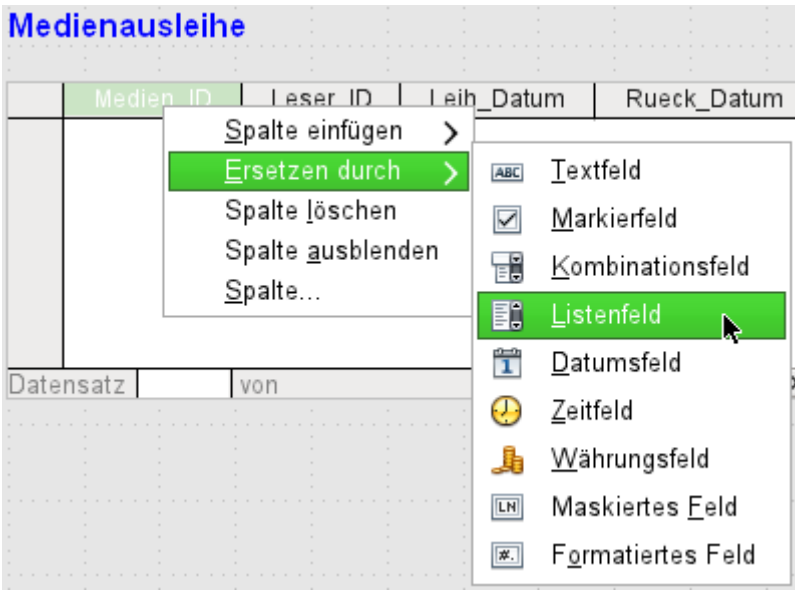
Tabellenkontrollfeld

Unter der Beschreibung zum *Tabellen-Kontrollfeld* wurde bereits über den Tabellenassistenten das entsprechende Tabellenkontrollfeld erstellt. Es hat allerdings noch einige Nachteile, die zu verbessern sind:

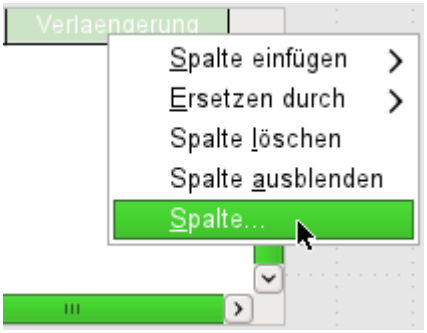
- Aus den Feldern «Medien_ID» und «Leser_ID» muss ein Listenfeld erstellt werden.
- Numerische Felder müssen gegebenenfalls auf Felder ohne Nachkommastellen umgestellt werden, da der Assistent hier grundsätzlich 2 Nachkommastellen belässt.



Die Änderung von Feldern innerhalb des Tabellen-Kontrollfeldes ist allerdings nicht auf die gleiche Art möglich, wie es oben für andere Kontrollfelder beschrieben wurde. Im Navigator endet die Beschreibung der Felder beim Tabellen-Kontrollfeld. Von den im Tabellen-Kontrollfeld liegenden Kontrollfeldern für die dem Formular zugrundeliegenden Tabelle weiß der Navigator nichts. Dies gilt in gleichem Maße übrigens später auch, wenn mittels Makros auf die Felder zugegriffen werden soll. Sie sind mit Namen nicht ansprechbar.



Die Kontrollfelder innerhalb des Tabellenkontrollfeldes werden als Spalten bezeichnet. Über das Kontextmenü ist es jetzt möglich, Felder durch andere Felder zu ersetzen. Allerdings steht nicht die ganze Palette an Feldern zur Verfügung. So fehlen z. B.Buttons, Optionsfelder oder das grafische Kontrollfeld.



Die Eigenschaften der Felder sind über das Kontextmenü unter dem Begriff **Spalte** verborgen. Hier kann dann z. B. das numerische Feld *Verlängerung* geändert werden, so dass keine Nachkommastellen mehr angezeigt werden. Auch der dort standardmäßig eingetragene minimale Wert von –1.000.000,00 macht wohl für Verlängerungen wenig Sinn. Die Anzahl dürfte wohl im positiven einstelligen Bereich bleiben.

Sobald die Eigenschaften einer Spalte aufgerufen sind, lässt sich ohne das Eigenschaftsfeld zu schließen auch eine andere Spalte aufrufen. Ohne eine separate Speicherung über eine Schaltfläche ist es so möglich, alle Felder nacheinander abzuarbeiten.

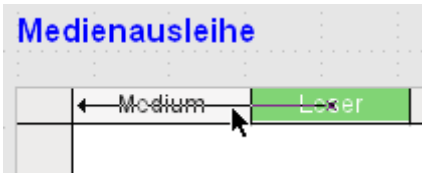
Die Speicherung erfolgt schließlich im gesamten Formular und letztlich mit dem Datenbankdokument selbst.

Die Eigenschaften dieser in das Tabellenkontrollfeld eingebauten Felder sind übrigens nicht ganz so umfangreich wie die der Felder außerhalb. Die Schrift z. B. lässt sich nur für das ganze Tabellenkontrollfeld einstellen. Außerdem gibt es hier nicht die Möglichkeit, einzelne Spalten ohne Tabulatorstop zu überspringen.

Tipp

Durch ein Formular erfolgt die Bewegung entweder mit der Maus oder mit dem Tabulator.
Gerät ein Tabulator einmal in ein Tabellenkontrollfeld, so bewegt sich der Cursor dort mit jeder Tabulatorbewegung ein Feld weiter nach rechts und bei Zeilenende schließlich zum ersten Feld des nächsten Datensatzes im Tabellenkontrollfeld.
Aus dem Tabellenkontrollfeld heraus geht es mittels **Strg + Tab** .

Die Anordnung der Spalten lässt sich über das Verschieben der Spaltenköpfe ändern:



Wird diese Verschiebung im Formularentwurf gemacht, so ist sie dauerhaft wirksam. Vorübergehend lässt sich so eine Verschiebung auch im Formular erledigen, wenn es zur Eingabe von Daten geöffnet ist.

Hinweis

Leider enthält diese Funktion zur Zeit einen Bug. Verschiebungen sind seit der Version LO 3.6 nicht mehr möglich. (*Bug 54021*)
Zur Zeit können also nur Felder neu eingefügt und auf diese Weise auch positioniert werden. Ein Kopieren eines Feldes ist bei gedrückter linker Maustaste möglich. Damit werden aber nur die ursprünglichen Formate kopiert, die beim Erstellen des Tabellenkontrollfeldes über den Assistenten erstellt wurden. Anschließende Änderungen berücksichtigt die Kopie zur Zeit nicht.

Sollen nur bestimmte Felder zum Bearbeiten offen stehen, so bietet es sich an, das Formular gleich mit mehreren Tabellenkontrollfeldern zu bestücken, denn der Tabulator wird standardmäßig von einem Tabellenkontrollfeld gefangen.

In dem in *Abbildung 34* aufgezeigten Formular wird oben die Ausgabe von Medien geregelt. Hier sind nur die direkt notwendigen Felder eingerichtet.

Im unteren Tabellen-Kontrollfeld erscheinen noch einmal alle Felder, damit auch ersichtlich ist, für welche Person und welches Medium denn die Rückgabe erfolgt. Der Code für das Listenfeld «Leser» ist weiter *unten* aus einer Abbildung ersichtlich.

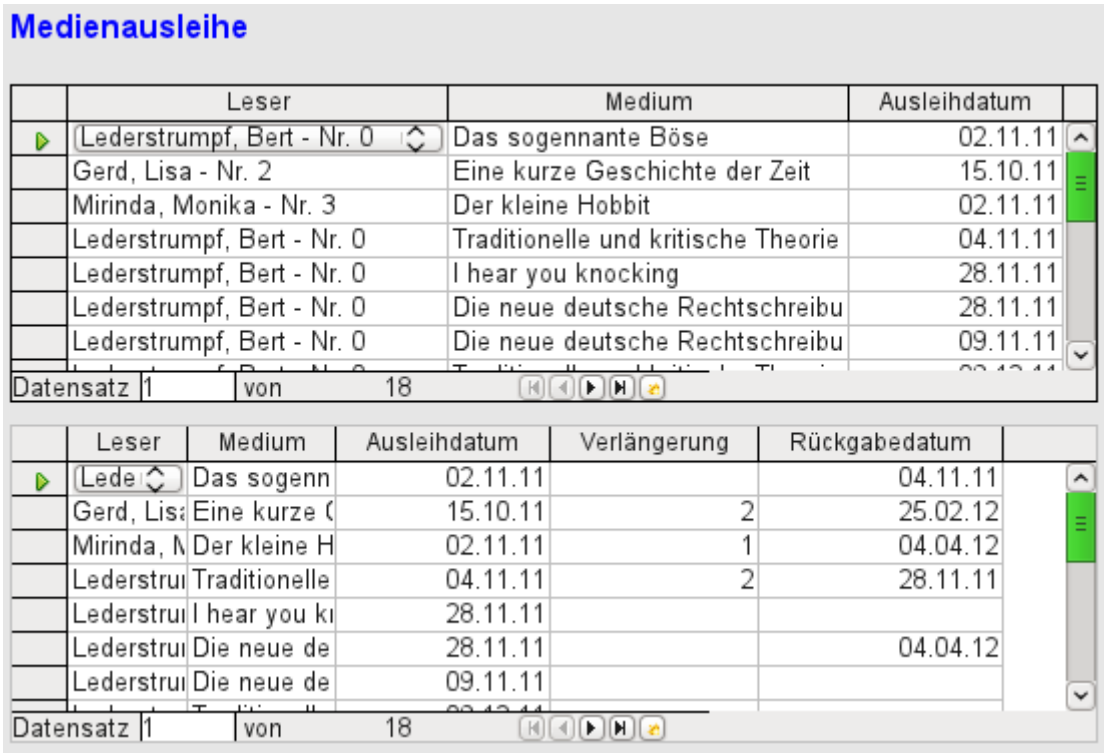
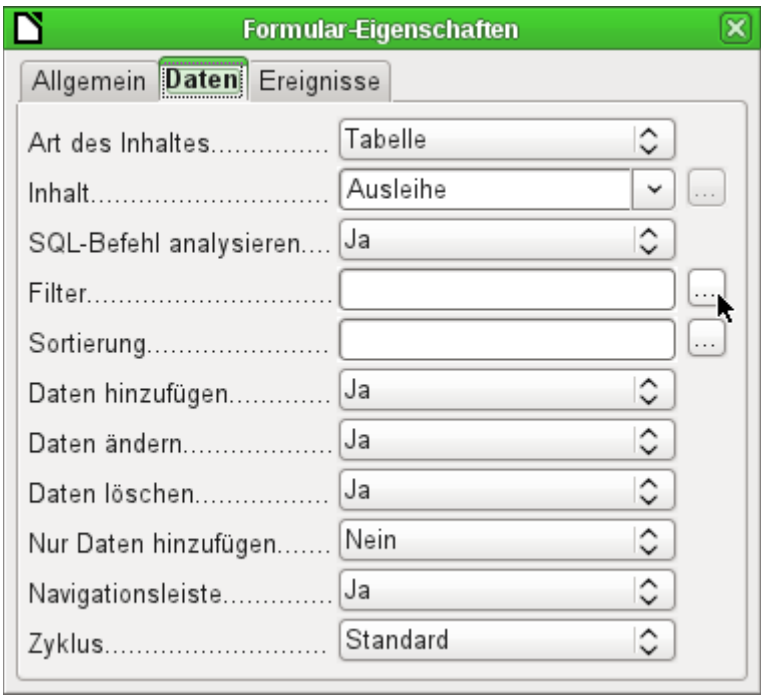


Abbildung 33: Ein Formular - mehrere Tabellen-Kontrollfelder

Diese Abbildung zeigt allerdings noch einen Schönheitsfehler, der dringend behoben werden sollte. In dem oberen Tabellenkontrollfeld fällt auf, dass das gleiche Medium mehrmals auftaucht. Dies liegt daran, dass auch die Medien angezeigt werden, die zurückgegeben wurden. Die Daten müssen für eine saubere Ausleihe also gefiltert werden. Daten mit «Rückgabedatum» brauchen gar nicht erst zu erscheinen.

Diese Filterung ist entweder über eine Abfrage oder direkt in den Formulareigenschaften möglich. Wird in den **Formular-Eigenschaften → Daten → Filter** eine Filterung erstellt, so kann diese auch bei der Eingabe in das Formular vorübergehend ausgeschaltet werden. Die Filterung mittels Abfrage wird in dem entsprechenden Kapitel «Abfragen» aufgegriffen. Hier wird jetzt der Weg über die Formulareigenschaften gezeigt:

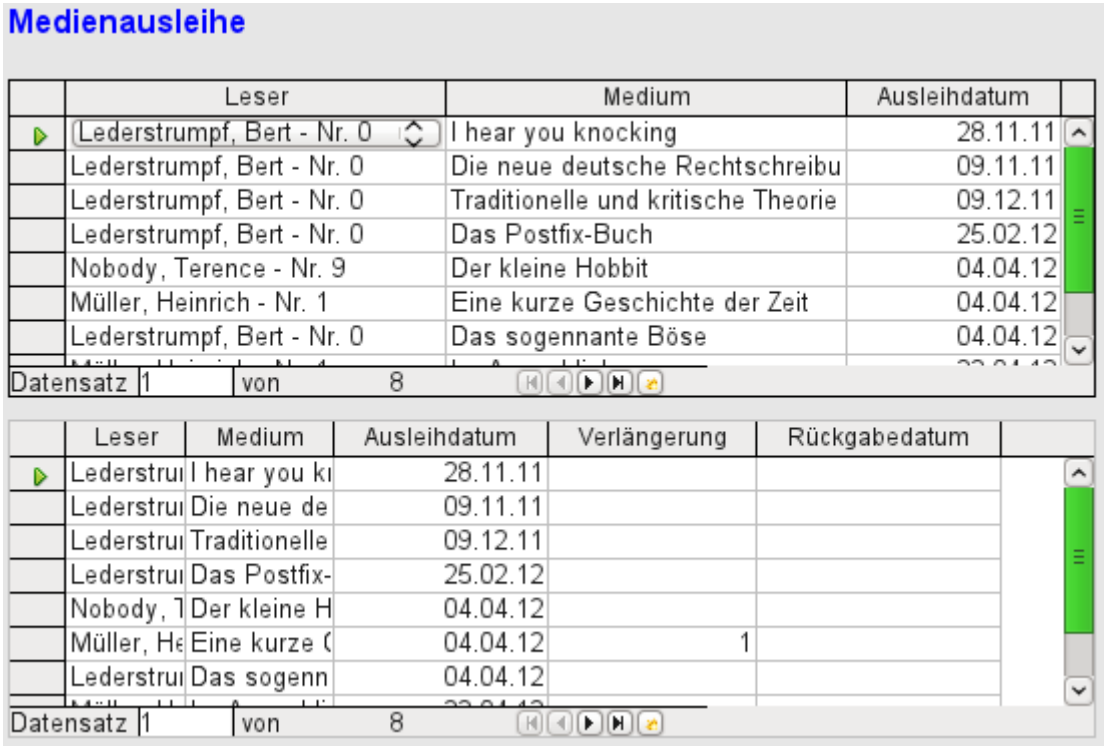


Die Filterung wird über den Button mit den drei Punkten ... gestartet. Sie könnte allerdings auch direkt in das Textfeld eingegeben werden, wenn die SQL-Formulierung bekannt ist.

In der grafischen Benutzeroberfläche lässt sich jetzt das Feld mit dem Namen "Rueck_Datum" auswählen. Angezeigt werden sollen nur die Datensätze, bei denen diese Feld 'leer' ist, wobei 'leer' für die SQL-Bezeichnung **NULL** steht.

In dem **Formular-Eigenschaften → Daten → Filter** steht dann (**"Rueck_Datum" IS NULL**). Mit Hilfe des Filters können alle Formulierungen gewählt werden, die auch in Abfragen erstellt würden. Es muss lediglich darauf geachtet werden, dass der Filter in einfachen Klammern eingetragen wird. Siehe dazu auch *Filtern von Tabellen* und die Erweiterung mit der *WHERE SQL-Expression* aus dem Kapitel «Abfragen».

Das auf diese Art bereinigte Formular sieht dann schon etwas übersichtlicher aus:



Sicher ist dies noch verbesserbar, bietet aber neben der Funktionalität der anderen Formulare den unbestreitbaren Vorteil, dass alle Medien auf einen Blick sichtbar sind.

Die Bearbeitung von Daten mit Tabellenkontrollfeldern ist ähnlich der von Tabellen. Mit einem Rechtsklick auf den Datensatzmarkierer wird bei existierenden Datensätzen die Löschung des Datensatzes angeboten, bei Neueingaben kann die Dateneingabe rückgängig gemacht oder abgespeichert werden.

Wird eine Zeile verlassen, so wird der Datensatz automatisch abgespeichert.

Noch ist das Formular zur Medienausleihe in vielen Bereichen zu verbessern.

- Es wäre wünschenswert, wenn an einer Stelle der Leser ausgewählt würde und an anderer Stelle zu diesem Leser die entliehenen Medien erscheinen.
- In der oberen Tabelle sind lauter Datensätze zu sehen, die dort eigentlich gar nicht notwendig sind. Die Medien sind ja schon entliehen. Das obere Tabellenblatt ist aber erstellt worden, um die Ausleihe zu ermöglichen. Besser wäre es, wenn hier nur ein leeres Blatt erscheint, das dann mit den neuen Ausleihen ausgefüllt wird.

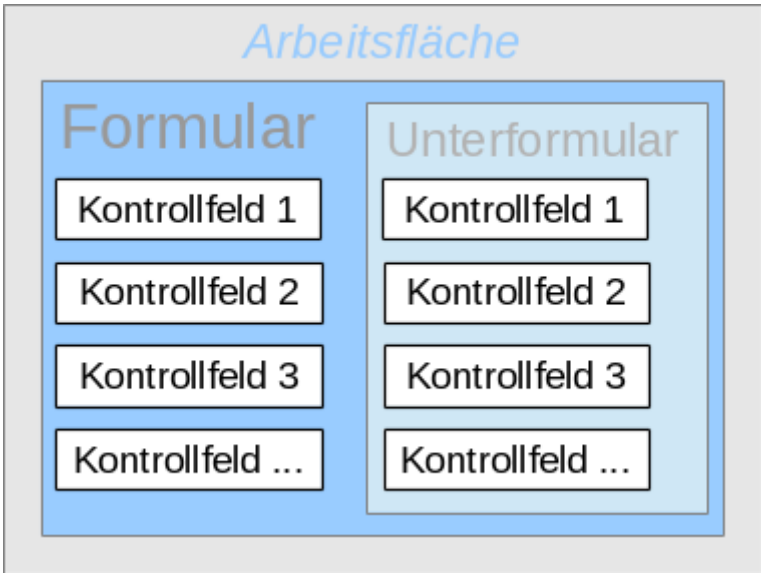
Solche Lösungen sind mit Hilfe von weiteren Formularen möglich, die hierarchisch aufeinander aufgebaut sind und eine getrennte Sicht auf die Daten ermöglichen.

Tipp

Das Tabellenkontrollfeld kann mit anderen Feldern im Formular kombiniert werden. In dem Tabellenkontrollfeld werden z. B. nur die Medientitel angezeigt, in den darunter stehenden Formularfeldern dann an diesem Datensatz Änderungen vorgenommen.
Bei der Kombination von Tabellenkontrollfeld und weiteren Formularfeldern gibt es einen kleinen, leicht behebbaren Bug. Wenn beide Feldarten zusammen in einem Formular liegen, so läuft der Cursor von den anderen Formularfeldern automatisch in das Tabellenkontrollfeld, obwohl dieses Feld standardmäßig auf **Tabstop → 'Nein'** eingestellt ist. Das kann behoben werden, indem der Tabstop einmal auf 'Ja' und anschließend wieder auf 'Nein' eingestellt wird. Dann erst wird 'Nein' wirklich übernommen.

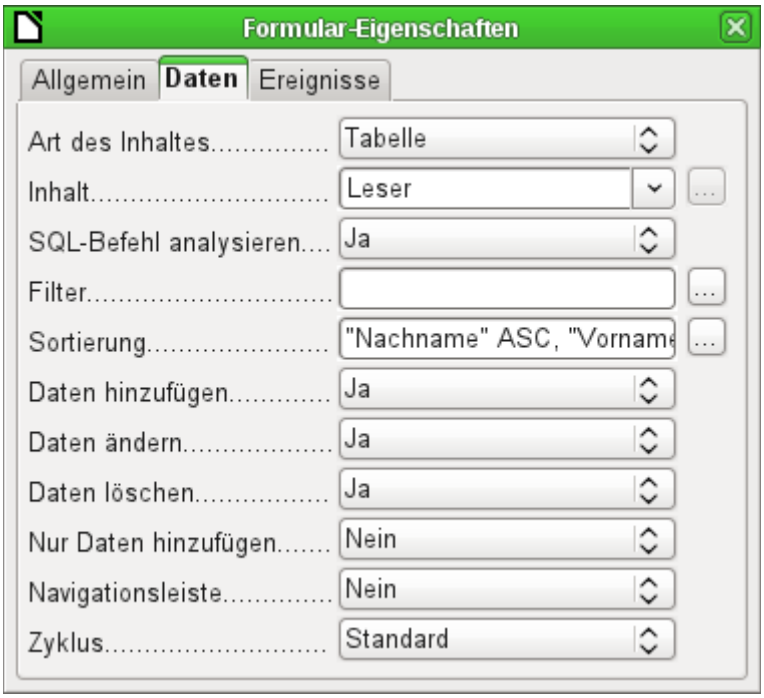
Hauptformular und Unterformular

Ein Unterformular liegt wie ein Formularkontrollfeld innerhalb eines Formulars. Wie ein Formularkontrollfeld wird es mit den Daten des (Haupt)-Formulars verbunden. Allerdings kann es als Datenquelle eine andere Tabelle oder eine Abfrage (bzw. einen SQL-Befehl) beinhalten. Für ein Unterformular ist nur wichtig, dass seine Datenquelle irgendwie mit der Datenquelle des Hauptformulars verbunden werden kann.

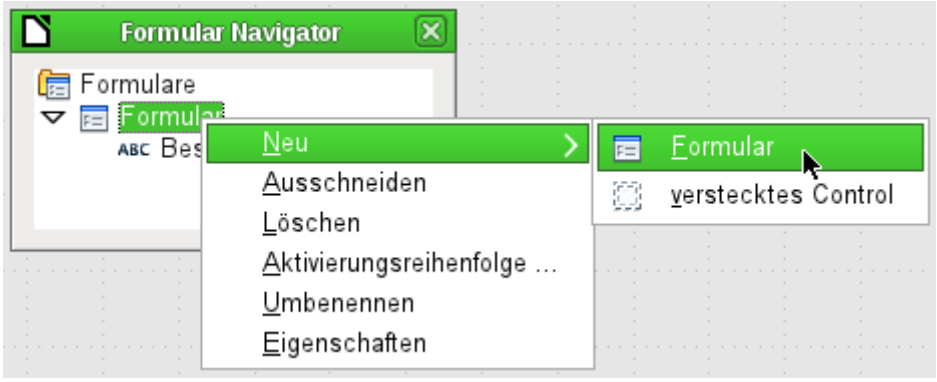


Typische Tabellenkonstruktionen, die sich für Unterformulare eignen, sind Tabellen mit einer *Eins-zu-Viele Beziehung*. Hauptformulare zeigen eine Tabelle an, zu deren Datensatz dann in den Unterformularen viele abhängige Datensätze aufgezeigt werden.

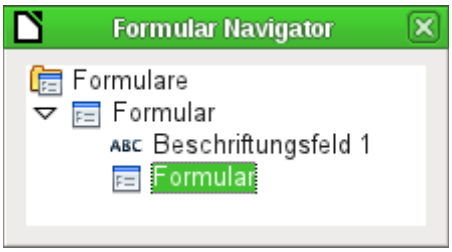
Wir nutzen jetzt erst einmal die Beziehung der Tabelle "Leser" zur Tabelle "Ausleihe" (siehe *Tabellen Ausleihe*). Die Tabelle "Leser" wird Grundlage für das Hauptformular, die Tabelle "Ausleihe" wird in dem Unterformular wiedergegeben.



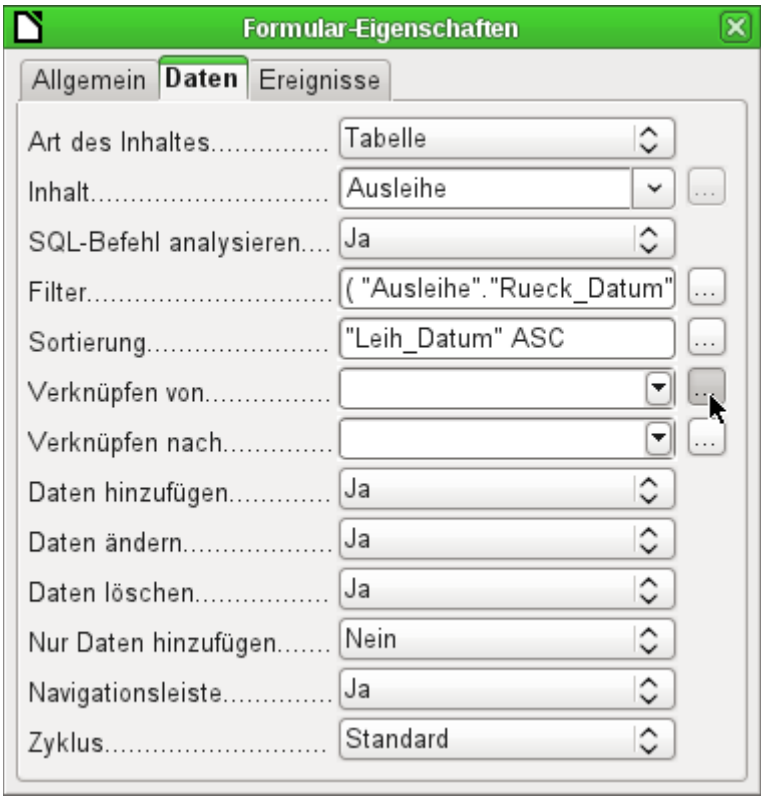
Das Hauptformular wird auf die Tabelle "Leser" eingestellt. Damit die Leser auch schnell genug gefunden werden, wird die Tabelle sortiert wiedergegeben. Auf eine Navigationsleiste wird verzichtet, weil zwischen Hauptformular und Navigationsleiste der Inhalt des Unterformulars angezeigt wird. Stattdessen soll das Formularkontrollelement *Navigationsleiste* eingebaut werden.



Durch Rechtsklick auf das Hauptformular im Formular-Navigator wird über das Kontextmenü ein neues Formular gegründet. Das Formular hat standardmäßig wieder den Namen Formular, ist jetzt aber ein Element im Unterverzeichnis des Hauptformulars.

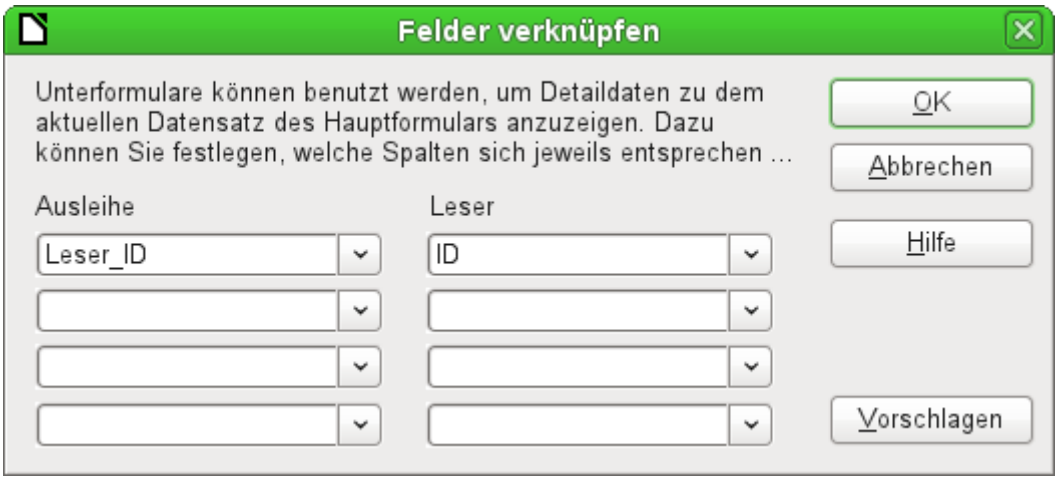


Die Eigenschaften des Unterformulars müssen jetzt entsprechend eingestellt werden, damit es die richtige Datenquelle und die zu dem entsprechenden Leser gehörigen Daten wiedergibt.



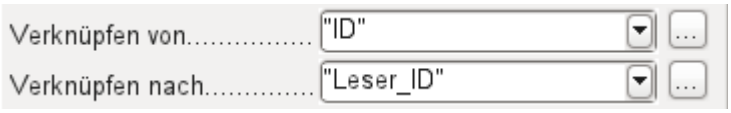
Für das Unterformular wird die Tabelle "Ausleihe" gewählt. Beim Filter wird eingestellt, dass das Rückgabedatum leer sein soll (**"Rueck_Datum" IS NULL**). Dadurch werden keine bereits zurückgegebenen Medien angezeigt. Die Datensätze sollen nach dem Entleihdatum vorsortiert werden. Diese Sortierung zeigt das am längsten entliehene Medium am weitesten oben an.

Über **Verknüpfen von** und **Verknüpfen nach** wird eine Verbindung zum Hauptformular hergestellt, in dem das Unterformular liegt. Der Button mit den drei Punkten ... zeigt wieder an, dass es hier ein helfendes Fenster für die Auswahl der Einstellungen gibt.



Unter «Ausleihe» werden die Felder der Tabelle "Ausleihe" angezeigt, unter «Leser» die der Tabelle "Leser". Die "Leser_ID" der "Ausleihe" soll gleichbedeutend sein mit der "ID" der Tabelle "Leser".

Obwohl diese Verknüpfung bereits in der Datenbank unter **Extras → Beziehungen** erstellt wurde (siehe *Verknüpfung von Tabellen*) greift die hinter dem Button Vorschlagen liegende Funktion hierauf nicht zurück und möchte stattdessen den ersten Fremdschlüssel aus der Tabelle "Ausleihe", nämlich "Medien_ID", mit "ID" aus der Tabelle "Leser" verbinden. Dies gelingt dem Assistenten zur Erstellung von Formularen besser.



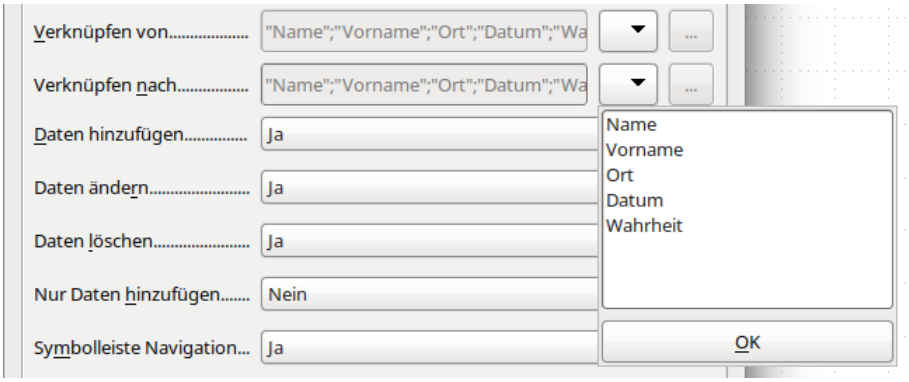
Die ausgewählte Verknüpfung von der Tabelle des Unterformulars nach der Tabelle des Hauptformulars wird jetzt mit den entsprechenden Feldern der Tabellen angegeben.

Hinweis

Es ist auch möglich, Parameterabfragen über ihre Parameter im Unterformular zu verknüpfen. Diese Parameter werden aber nicht zur Auswahl angeboten. Hier erscheinen in Verknüpfen nach nur die Felder der Tabelle oder Abfrage. Die Bezeichnung der Parameter muss also direkt in das Kombinationsfeld eingegeben werden.
Beispiel:
SELECT * FROM "Ausleihe" WHERE "Datum" < :pLeihDatum
«pLeihDatum» darf es nicht als Feld in der Tabelle Ausleihe geben. Dann kann über «pLeihDatum» diese Abfrage mit einem Datum des Hauptformulars verbunden werden.

Tipp

Der Dialog **Felder verknüpfen** zeigt die Möglichkeit an, 4 Felder zu verknüpfen. Es ist aber auch problemlos möglich, mehr Felder zu verknüpfen:



Die Begriffe werden über den Button ▼ in das Feld untereinander eingegeben.

Um jetzt ein Tabellenkontrollfeld für das Hauptformular zu erstellen, muss das Hauptformular im Formular-Navigator markiert sein. Dann zeigt bei eingeschaltetem Tabellenkontrollfeld-Assistenten der Assistent die Felder an, die im Hauptformular zur Verfügung stehen. Entsprechend wird mit dem Unterformular verfahren.

Nachdem so die Tabellenkontrollfelder aufgezogen wurden, werden die entsprechenden Änderungen durchgeführt, die schon beim einfachen Formular erklärt wurden:

- Ersetzen des numerischen Feldes «Medien_ID» im Unterformular durch ein Listenfeld.
- Umbenennung des Feldes «Medien_ID» in Medien.
- Anpassung der numerischen Felder an ein Format ohne Nachkommastellen.
- Eingrenzung der minimalen und maximalen Werte.
- Umbenennung anderer Felder, um Platz zu sparen oder Umlaute darzustellen, die bei der Feldbenennung in den Datenbanktabellen vermieden wurden.

Sortier- und Filterfunktion werden für das Hauptformular ergänzt, indem eine Navigationsleiste hinzugefügt wird. Die anderen Felder der Navigationsleiste werden nicht benötigt, da sie vom Tabellenkontrollfeld weitgehend zur Verfügung gestellt werden. (Datensatzanzeige, Datensatznavigation) bzw. durch die Bewegung im Tabellenkontrollfeld erledigt werden (Speicherung von Daten).

Das erstellte Formular könnte schließlich wie in der folgenden Abbildung aussehen:

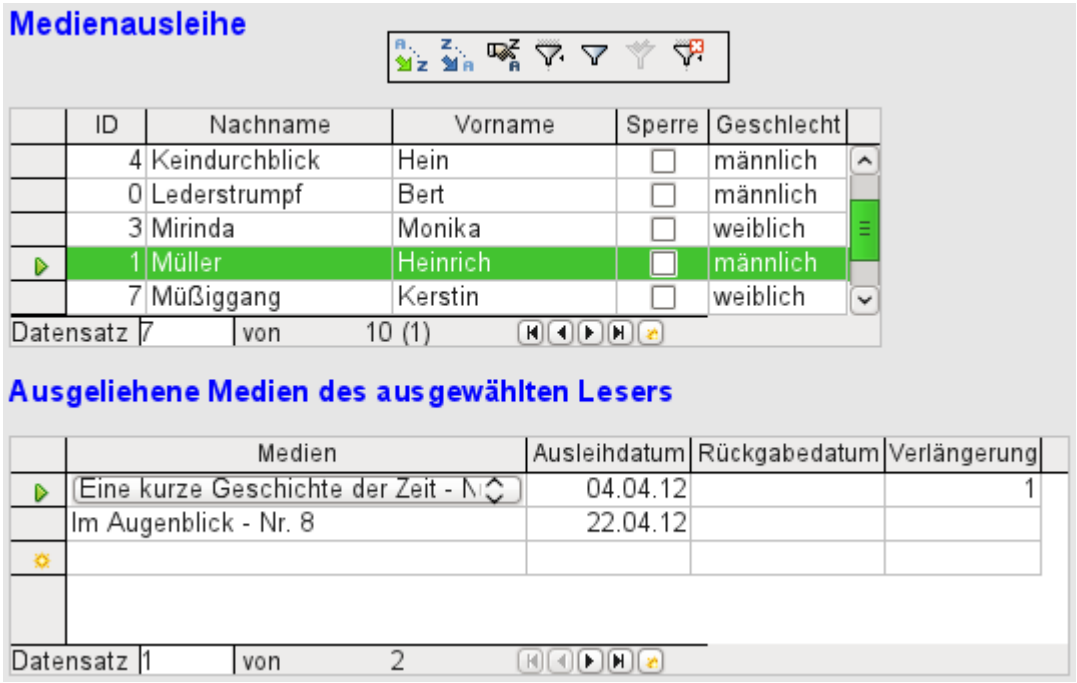


Abbildung 34: Formular, bestehend aus Hauptformular (oben) und Unterformular (unten).

Wird jetzt im Hauptformular ein Leser ausgewählt, so werden im Unterformular nur die Medien aufgezeigt, die der Leser zur Zeit entliehen hat. Wird ein Medium zurückgegeben, so erscheint dies noch so lange im Formular, bis das Formular selbst aktualisiert wird. Dies geschieht automatisch, wenn im Hauptformular ein anderer Datensatz gewählt wurde. Bei der erneuten Anwahl des ursprünglichen Lesers sind also die zurückgegebenen Medien nicht mehr in der Anzeige.

Diese verzögerte Aktualisierung ist in diesem Fall wohl auch erwünscht, da so direkt eingesehen werden kann, welche Medien denn jetzt gerade auf der Theke der Mediothek liegen und ob diese schon registriert wurden.

Diese Formularkonstruktion bietet schon deutlich mehr Komfort als die vorherige mit nur einem einzigen Formular. Allerdings gibt es noch Details, die verbesserungswürdig erscheinen:

- Medien und Ausleihdaten können geändert werden, wenn die Medien schon länger entliehen sind.
  Eine Änderung der Medien-Daten führt dazu, dass nicht mehr nachvollzogen werden kann, welches Medium denn nun noch in der Mediothek vorhanden ist und welches entliehen wurde.
  Eine Änderung des Ausleihdatums kann zu fehlerhaften bzw. nicht beweisbaren Mahnungen führen.
- Wird ein Leser nicht durch Klick auf den Datensatzmarkierer markiert, so zeigt nur der kleine grüne Pfeil auf dem Markierer an, welcher Datensatz gerade aktiv ist. Es ist auch möglich, den aktiven Datensatz komplett aus dem Tabellenkontrollfenster zu scrollen. Statt des Textes «Ausgeliehene Medien des ausgewählten Lesers» würde hier besser auch der Name erwähnt.
- Es ist möglich, mehrmals das gleiche Medium auszuleihen, ohne dass es zurückgegeben wurde.
- Es ist möglich, die Datensätze für ausgeliehene Medien einfach zu löschen.
- Auch im Hauptformular sind Änderung und Löschung von Daten möglich. Dies kann bei kleinen Mediotheken mit wenig Publikumsbetrieb sinnvoll sein. Sobald aber am Ausgabeschalter größere Hektik entsteht, ist die Bearbeitung von Nutzerdaten nicht an der gleichen Stelle vorzunehmen wie die Ausleihe.
  Eine Vereinfachung wäre schon, wenn eine Neuaufnahme ermöglicht würde, alte Daten

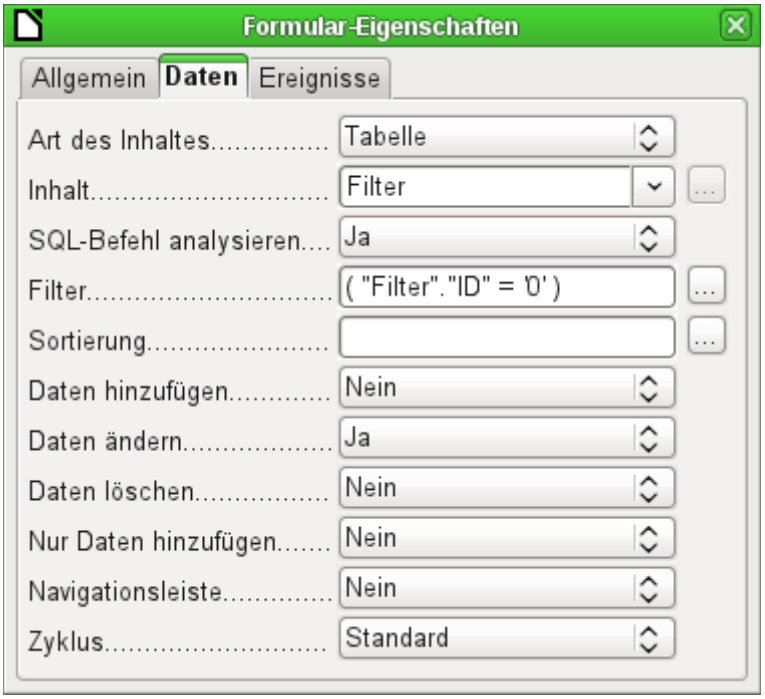
aber nicht angerührt werden dürfen. Denn ob nun Löschung oder komplette Veränderung des Namens – das Ergebnis bleibt für die Mediothek das Gleiche.

Zuerst wird einmal die Auswahl der Leser verbessert. Dies soll vor Änderungen in der Ausleihe schützen. Eine einfache Lösung wäre, keine Änderungen zuzulassen, aber neue Datensätze eingeben zu können. Dazu wird immer noch eine Suchfunktion benötigt, wenn ein Leser ein Medium entleihen will. Besser wäre, in einem Listenfeld die Leser auszusuchen und in voneinander getrennten Tabellenkontrollfeldern die Ausgabe und die Rückgabe zu erledigen.

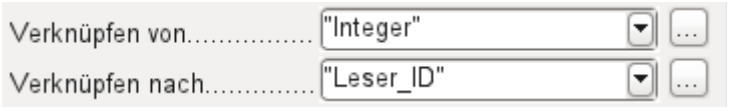
Für das Hauptformular benötigen wir eine Tabelle, in die das Listenfeld seinen mit dieser Tabelle verbundenen Wert schreiben kann. Die Tabelle muss also ein Integer-Feld und einen Primärschlüssel haben. Sie wird beständig nur einen Datensatz enthalten; daher kann das Feld "ID" als Primärschlüssel ruhig 'Tiny Integer' sein. Es reicht auch ein Feld des Typs 'Ja/Nein'. Die folgende Tabelle mit der Bezeichnung "Filter" soll also erstellt werden:

Tabellenname: Filter	
Feldname	***Feldtyp***
ID	Tiny Integer, Primärschlüssel
Integer	Integer

Die Tabelle wird mit einem Primärschlüsselwert gefüllt, und zwar dem Wert 0. Diesen Datensatz wird das Hauptformular beständig lesen und neu schreiben.

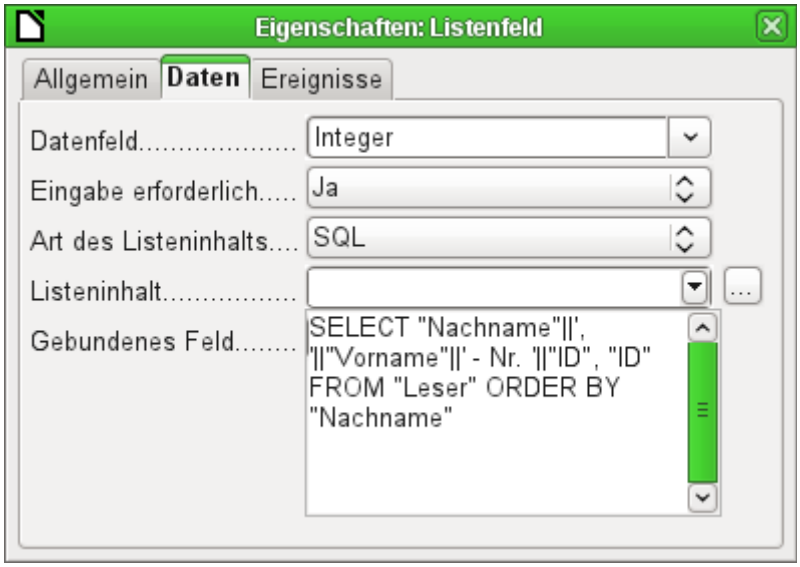


Das Hauptformular beruht auf der Tabelle "Filter". Es wird nur der Wert aus der Tabelle gelesen, bei dem der Primärschlüssel "ID" '0' ist. Es sollen keine Daten hinzugefügt werden, sondern nur der aktuelle Datensatz beständig neu geschrieben werden. Daher ist hier nur das Ändern erlaubt, eine Navigationsleiste sowieso überflüssig.



Das Hauptformular wird mit dem Unterformular so verknüpft, dass der Wert aus dem Feld "Integer" der Tabelle "Filter" gleich dem Wert aus dem Feld "Leser_ID" aus der Tabelle "Ausleihe" ist. Das Unterformular bleibt in seinen Eigenschaften ansonsten gegenüber der Vorversion unberührt.

Bevor jetzt im Hauptformular ein Listenfeld aufgezogen wird, wird erst einmal der Assistent ausgeschaltet. Mit dem Assistenten könnte nur ein Feld erzeugt werden, das lediglich einen Feldinhalt anzeigt; es wäre unmöglich, Nachname und Vorname und zusätzlich noch eine Nummer in dem Anzeigefeld des Listenfeldes zu positionieren. Wie bereits bei dem einfachen Formular wird jetzt das Listenfeld mit *Nachname, Vorname – Nr. ID* bestückt. Das Listenfeld gibt außerdem die ID an die darunterliegende Tabelle weiter.



Neben dem Listenfeld wird ein Button erstellt. Dieser Button ist allerdings Bestandteil des Unterformulars. Er soll gleichzeitig zwei Funktionen übernehmen: Abspeicherung des Datensatzes aus dem Hauptformular und Aktualisierung der Tabelle im Unterformular. Dazu reicht es, dem Button im Unterformular die Aktualisierung zuzuweisen. Der Speichervorgang für das veränderte Hauptformular wird dadurch automatisch ausgelöst.

Der Button kann einfach mit '*OK*' als Aufschrift versehen werden. Als Aktion wird **Formular aktualisieren** zugewiesen.

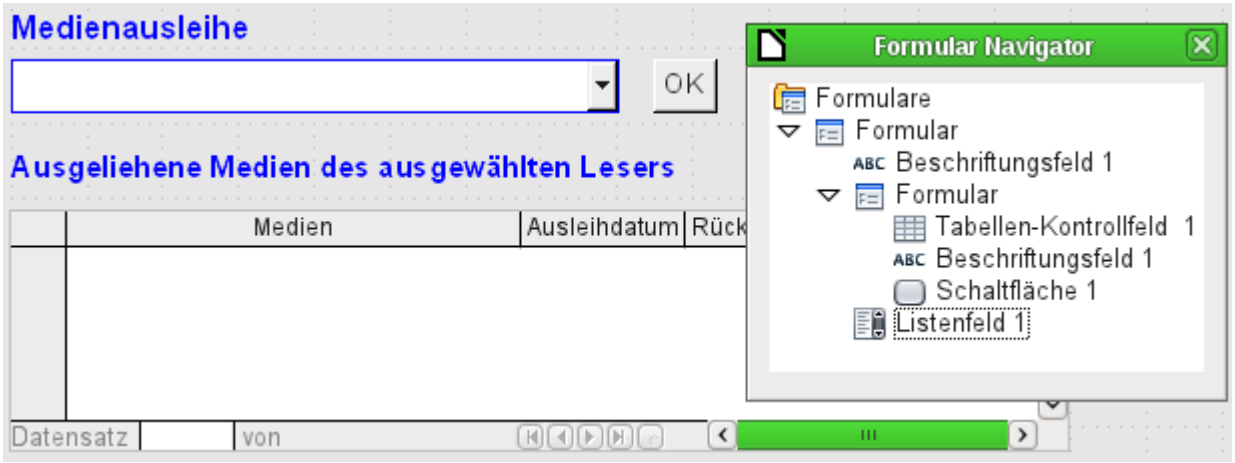


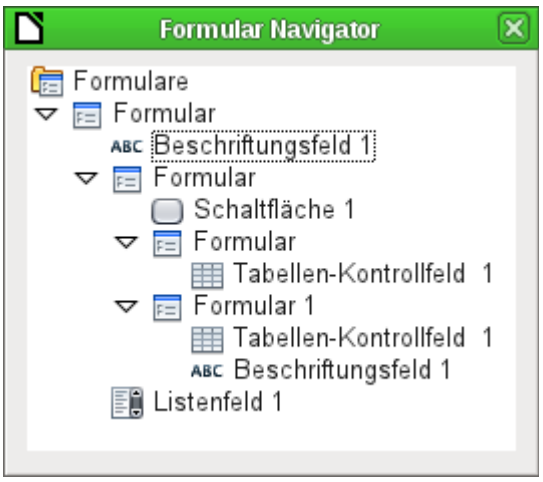
Abbildung 35: Hauptformular als Filter für ein Unterformular

Das Hauptformular besteht nur aus der Überschrift und dem Listenfeld, das Unterformular aus einer weiteren Überschrift, dem Tabellen-Kontrollfeld der vorherigen Version und dem Button.

Das Formular funktioniert jetzt schon insofern besser, als

- keine Leser mehr bearbeitet, vor allem verändert und gelöscht werden können und
- Leser schneller über das Eintippen in das Kontrollfeld gefunden werden als über Filter.

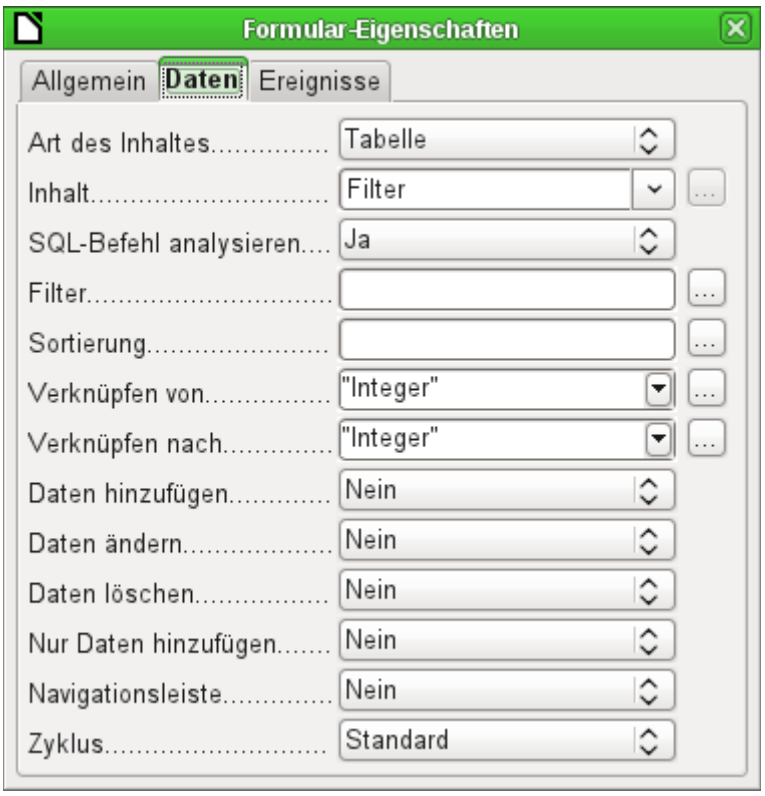
Für eine weitreichendere Funktionalität (Rückgabe ohne Änderung der vorherigen Daten) muss ein zweites Unterformular gegründet werden, das sich auf die gleiche Tabelle "Ausleihe" bezieht. Damit dennoch die Funktionalität des Listenfeldes aus *Abbildung 35* gewährleistet wird, müssen beide Unterformulare allerdings noch eine Ebene tiefer gelegt werden, also Unterformulare eines Unterformulars werden. Eine Aktualisierung von Daten verläuft in der Hierarchie nur vom Hauptformular zum Unterformular abwärts. Der Button im letzten vorgestellten Formular würde nur ein Unterformular aktualisieren, nicht aber das zweite, daneben liegende Unterformular.



Der Formularnavigator zeigt hier die verschiedenen Ebenen an. Im Hauptformular befindet sich das Beschriftungsfeld für die Formularüberschrift und das Listenfeld, in dem die Leser ausgesucht werden. Das Listenfeld steht in der Ansicht ganz unten, da es nach dem Unterformular gegründet wurde. Diese Reihenfolge der Anzeige lässt sich leider nicht beeinflussen. Das Unterformular hat lediglich eine Schaltfläche, mit der sein Inhalt aktualisiert und der des Hauptformulars gleichzeitig abgespeichert wird. Noch eine Ebene tiefer liegen dann zwei Unter-Unterformulare. Diese werden bei der Gründung bereits unterschiedlich benannt, so dass in keiner Ebene von der Benennung her Verwechselungen auftreten können.

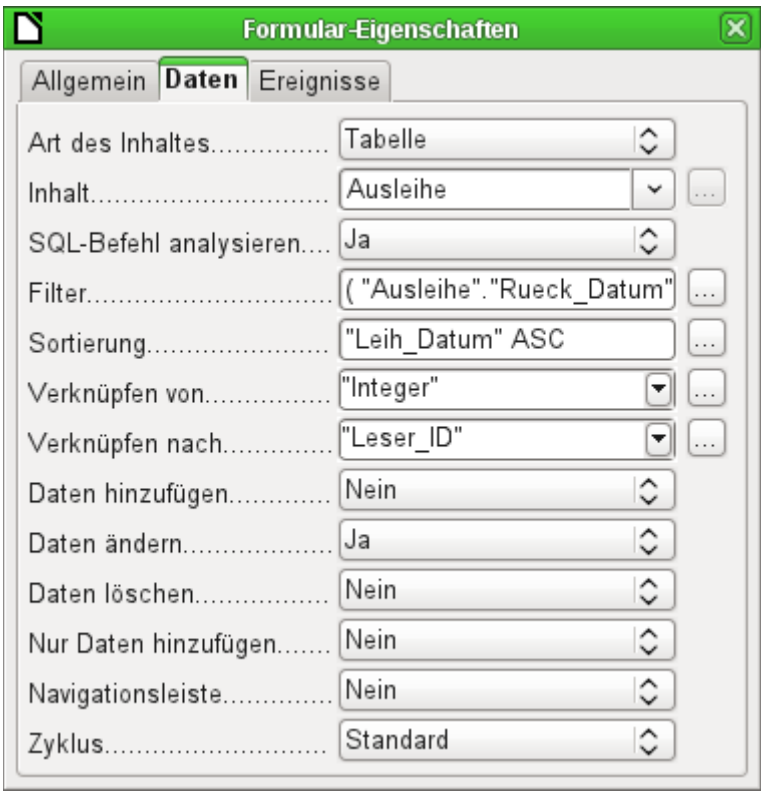
Hinweis

Grundsätzlich sind die Benennungen der Formulare und Kontrollfelder erst einmal ohne Bedeutung. Wenn sie aber über den Namen durch Makros angesprochen werden sollen, müssen sie unterscheidbar sein. Gleiche Namen in der gleichen Formularebene erlauben keine Unterscheidung.
Natürlich ist es sinnvoll, bei größeren Formularkonstruktionen aussagekräftigere Namen für die Formulare und ihre Kontrollfelder zu nutzen. Ansonsten dürfte ein Auffinden des richtigen Feldes schnell zum Problem werden.



Das Hauptformular und das Unterformular nutzen einfach die gleiche Tabelle. Im Unterformular werden keine Daten eingegeben. Deshalb stehen alle diesbezüglichen Felder auf 'Nein'. Verknüpft werden Hauptformular und Unterformular durch das Feld, dessen Wert auch an die Unter-Unterformulare weitergegeben werden soll: das Feld "Integer" der Tabelle "Filter".

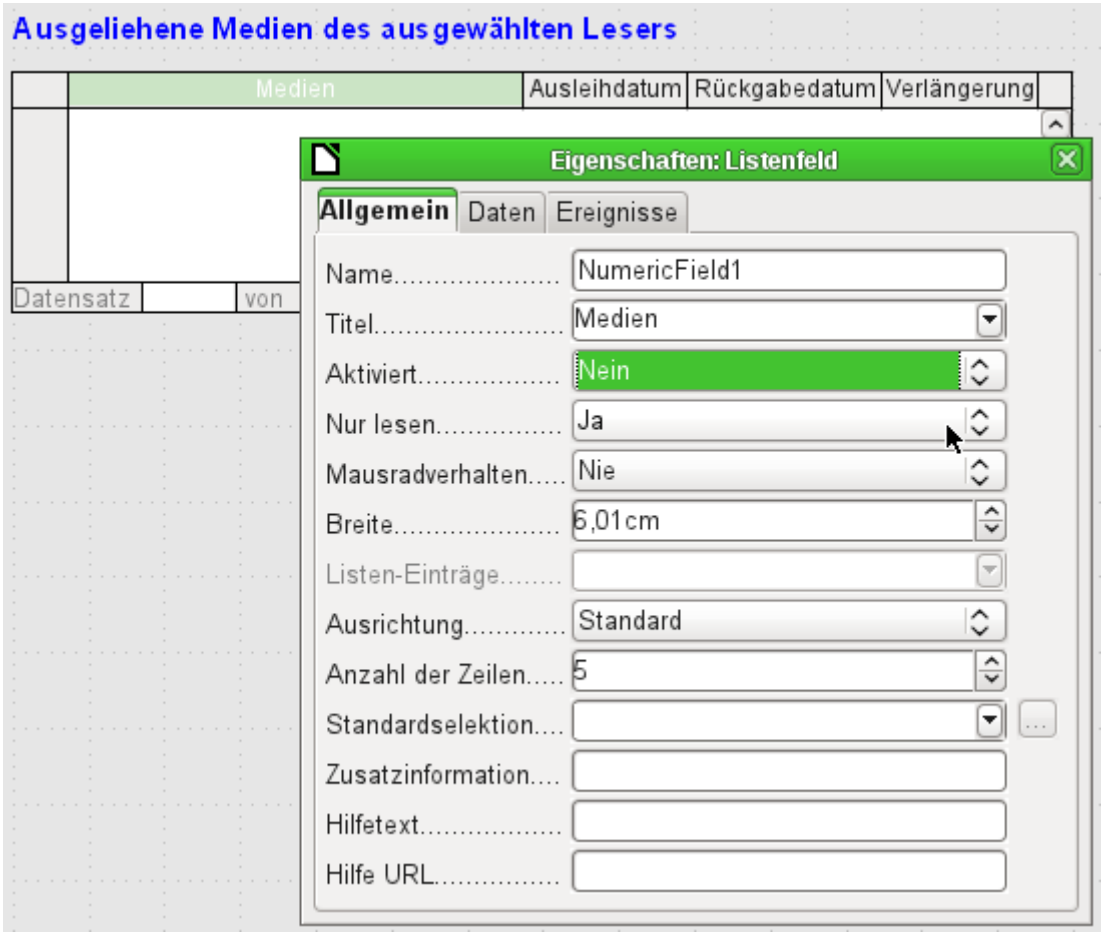
Im ersten Unter-Unterformular werden keine alten Daten angezeigt, sondern nur neue Daten verarbeitet. Hierzu reicht der Filter, der gesetzt wurde. Es werden nur Daten angezeigt, die zu der "Leser_ID" passen und deren Leihdatum leer ist (**`"Leih_Datum" IS NULL`**). Das bedeutet beim Aufruf ein leeres Tabellen-Kontrollfeld. Da das Tabellen-Kontrollfeld zwischendurch nicht laufend aktualisiert wird, bleiben die gerade neu ausgeliehenen Medien so lange in dem Tabellen-Kontrollfeld stehen, bis über den Aktualisierungsbutton OK entweder ein neuer Name ausgewählt oder auch nur die Übernahme der Daten in das zweite Unter-Unterformular veranlasst wird.



Das zweite Unter-Unterformular erfordert mehr Einstellungen. Auch dieses Formular enthält die Tabelle "Ausleihe". Hier werden aber die Daten gefiltert, bei denen das Rückgabedatum leer ist (**`"Rueck_Dat" IS NULL`**). Die Daten werden wie im vorhergehenden Formular so sortiert, dass die am längsten entliehenen Medien direkt sichtbar sind.

Wichtig sind jetzt auch die weiter unten stehenden Einträge. Alte Datensätze können geändert werden, aber es können keine neuen Datensätze hinzugefügt werden. Ein Löschen ist nicht möglich. Damit ist der erste Schritt gemacht, der notwendig ist, um später nicht Entleihdaten einfach zu löschen. Noch wäre es aber möglich, z. B. das Medium und das Entleihdatum zu ändern. Hier muss in den Eigenschaften der Spalten weiter justiert werden. Schließlich soll das Medium und das Entleihdatum nur angezeigt, aber von der Änderung ausgeschlossen werden.

Das Tabellen-Kontrollfeld wird nach der Erstellung der Formulare einfach verdoppelt. Dazu wird es markiert, anschließend kopiert, danach wird die Markierung aufgehoben und aus der Zwischenablage wieder eingeführt. Das Doppel befindet sich an der gleichen Position wie das Original, muss also noch verschoben werden. Danach können beide Tabellenkontrollfelder entsprechend bearbeitet werden. Das Tabellenkontrollfeld zur Medienrückgabe bleibt nahezu unverändert. Lediglich die Schreibrechte für die Spalten «Medien» und «Ausleihdatum» müssen geändert werden.



Während bei «Ausleihdatum» lediglich **Nur lesen** gewählt werden muss, ist dies bei Listenfeldern nicht ausreichend. Diese Einstellung verhindert nicht, dass das Listenfeld weiterhin betätigt werden kann. Wird aber **Aktiviert → Nein** gestellt, so kann dort eine Auswahl nicht mehr stattfinden. Im Tabellen-Kontrollfeld wird ein enthaltenes Listenfeld dann wie ein nicht veränderbares Textfeld angezeigt.

Im oberen Tabellen-Kontrollfeld werden alle Felder entfernt, die nichts mit der Ausleihe zu tun haben. Es beleibt lediglich das Medium als Auswahlfeld sowie das Ausleihdatum "Leih_Dat" stehen.

Wird schließlich noch die Abfrage für das Listenfeld im oberen Tabellen-Kontrollfeld entsprechend gewählt, so werden dort nur Medien angezeigt, die noch entliehen werden können. Mehr dazu im Kapitel «Abfragen».

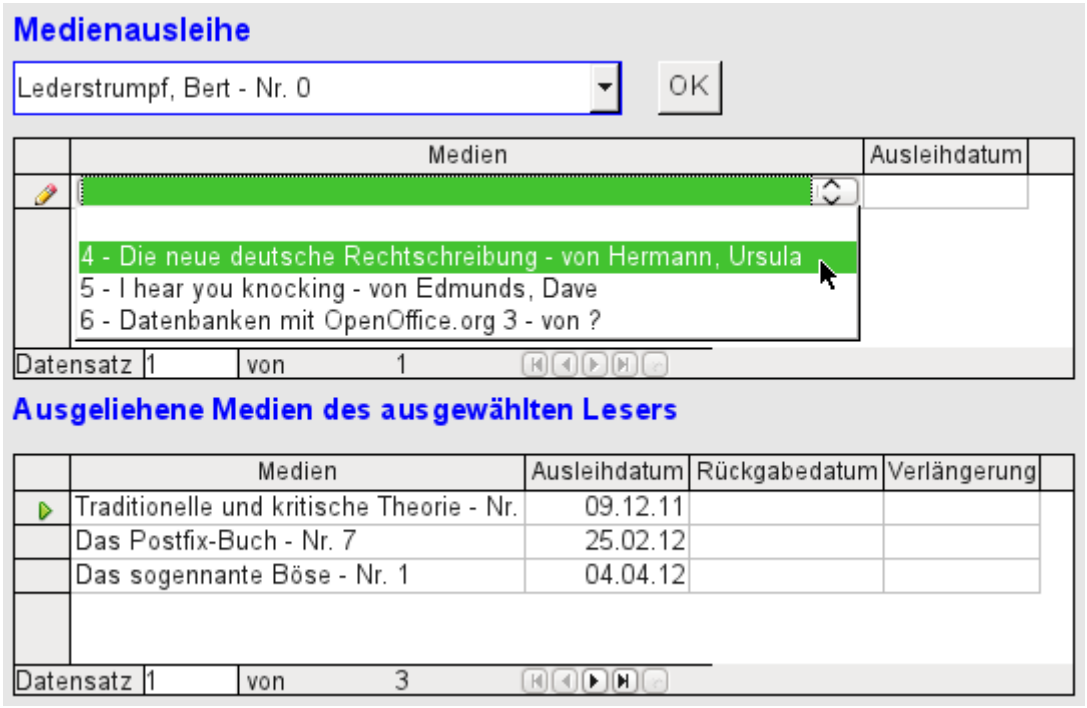
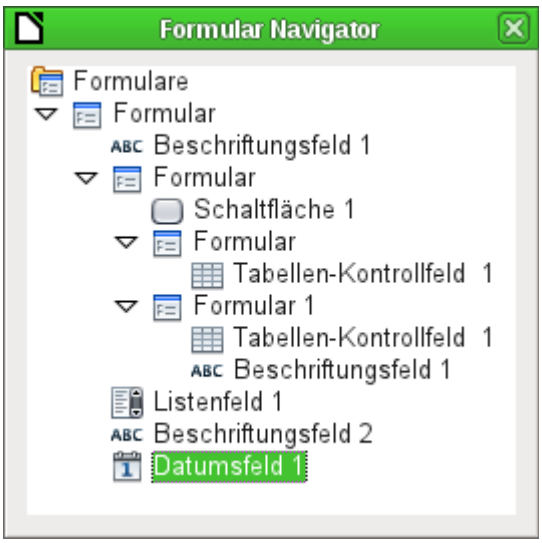


Abbildung 36: Das Auswahlfeld im oberen Unterformular zeigt nur Medien an, die nicht ausgeliehen sind.

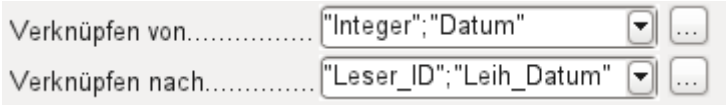
Das Formular zur Medienausleihe ist jetzt schon wesentlich besser zu bedienen. Kommt ein Leser an die Ausleihtheke, so wird der Name herausgesucht. Die zu entleihenden Medien können aus dem Listenfeld gewählt und das Entleihdatum eingestellt werden. Mit dem Tabulator geht es dann zum nächsten Datensatz.

Eine letzte Verbesserung wäre noch wünschenswert: Das Entleihdatum muss jedes Mal gewählt werden. Stellen wir uns einen Tag in der Mediothek mit vielleicht 200 Entleihvorgängen vor, vielleicht auch nur eine Person, die gleich 10 Medien auf einmal entleiht. Das wären mehrmals hintereinander die gleichen Eingabevorgänge für ein Feld. Hier muss eine Einsparmöglichkeit her.

Unser Hauptformular beruht auf einer Tabelle "Filter". Das Hauptformular arbeitet dabei immer nur mit dem Datensatz, der als Primärschlüssel die "ID" '0' hat. In die Tabelle "Filter" können ohne weiteres noch mehr Felder eingebaut werden. Da noch kein Feld enthalten ist, das ein Datum speichern kann, gründen wir einfach ein neues Feld mit dem **Feldname → Datum** und dem **Feldtyp → Datum**. In der Tabelle "Filter" wird jetzt nicht nur die "Leser_ID" ("Filter"."Integer") sondern auch das "Leih_Datum" ("Filter"."Datum") gespeichert.



Im Hauptformular erscheint jetzt zusätzlich ein Datumsfeld, außerdem noch ein Beschriftungsfeld, das auf den Inhalt des Datumsfeldes hinweist. Der Wert aus dem Datumsfeld wird in der Tabelle "Filter" gespeichert und über die Verbindung vom Unterformular zum Unter-Unterformular weitergegeben.



Die Verknüpfung zwischen beiden Formularen weist jetzt zwei Felder auf. Das Feld "Integer" wird mit dem Feld "Leser_ID" des Unter-Unterformulars verbunden. Das Feld "Datum" mit dem Feld "Leih_Datum". Damit wird das "Leih_Datum" automatisch aus der Tabelle "Filter" bei der Ausleihe in die Tabelle "Ausleihe" übertragen.

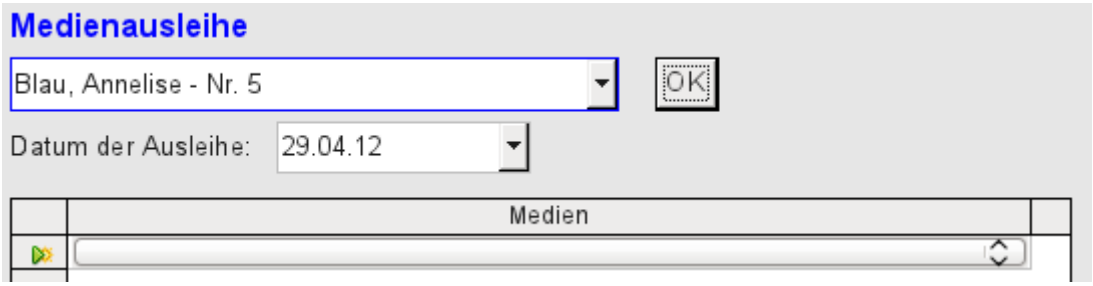
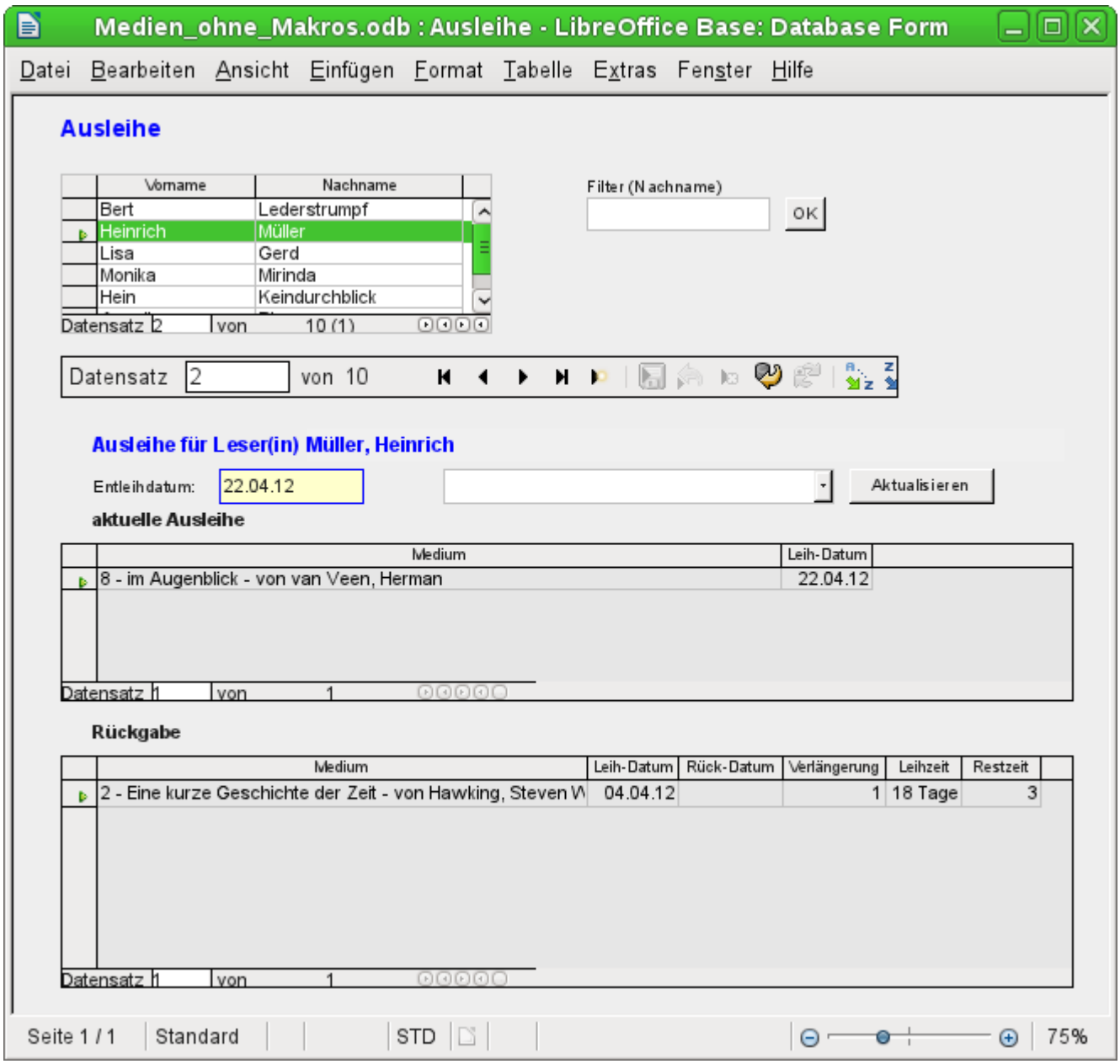


Abbildung 37: Das Datum der Ausleihe wird einmal eingestellt. Bei einem Wechsel des Lesers muss es nicht neu eingegeben werden.

Aus dem Tabellenkontrollfeld wurde jetzt auch noch das Datumsfeld entfernt. Das Tabellenkontrollfeld besteht jetzt lediglich noch aus einem Auswahlfeld. Dies wäre die ideale Voraussetzung um bei einer Mediothek noch an der Beschleunigungsschraube zu drehen. Denn eigentlich haben die Medien ja eine Nummer. Wozu müssen sie also ausgesucht werden. Da könnte doch gleich die Nummer eingetragen werden. Oder, noch besser, die Medien könnten mit Barcode-Etiketten versorgt werden. Ein Scanner dafür ist mit ca. 50.- € mittlerweile recht preisgünstig zu haben. Dann würden die Medien schneller ausgeliehen als der Entleiher sie in die Tasche packen kann.

In der Beispieldatenbank ist dies entsprechend aufgezeigt. Zur Vorstellung des ersten Formularentwurfs sollte das obige Beispiel aber erst einmal ausreichend sein.

Da allerdings das letztlich in der Beispieldatenbank «Medien_ohne_Makros.odb» vorgesehene Formular noch weiter entwickelt wurde, sollen die Erweiterungen hier noch kurz vorgestellt werden.



In die Ausleihe wurden die folgenden Eigenschaften aufgenommen:

- In einem Tabellenkontrollfeld werden die Leser und Leserinnen angezeigt. Hier können auch neue Leser und Leserinnen eingegeben werden.
- Über einen Filter, der mit der Tabelle "Filter" zusammenarbeitet, kann nach den Anfangsbuchstaben des Namens gefiltert werden. So werden bei einem 'A' nur die Personen angezeigt, deren Nachname mit 'A' beginnt. Die Filterung ist dabei unabhängig von der Eingabe von Groß- und Kleinschreibung.
- Im Untertitel wird noch einmal der Name der Person aufgezeigt, für die die Ausleihe erfolgen soll. Ist die Ausleihe für die Person gesperrt, so wird dies angezeigt.
- Das Entleihdatum ist auf das aktuelle Datum eingestellt. Dazu wurde die Filtertabelle über SQL so eingestellt, dass bei einem nicht eingegebenen Datum der Default-Wert das aktuelle Datum abspeichert.
- Die noch entleihbaren Medien werden in einem Listenfeld ausgewählt. Über den Button Aktualisieren wird die Ausleihe in das darunter stehende Tabellenkontrollfeld übertragen.

- Das mittlere Tabellenkontrollfeld dient lediglich der Anzeige der zu dem angegebenen aktuellen Datum ausgeliehenen Medien. Hier kann auch eine irrtümliche Ausleihe durch Löschen der Zeile rückgängig gemacht werden.
- Im unteren Tabellenkontrollfeld ist wie im vorher gezeigten Beispiel die Änderung von des Ausleihdatums von Medien nicht möglich. Auch eine Löschung ist nicht möglich.
- Neben der Eingabe des Rückgabedatums oder gegebenenfalls einer Verlängerung wird angezeigt, für wie viele Tage das Medium entliehen werden darf und wie viele Tage die restliche Entleihzeit beträgt.
- Geht die Restzeit in den negativen Bereich, so muss das Medium sofort zurückgegeben werden. Die Ausgabe ist deswegen gesperrt. Sie wird dadurch wieder ermöglicht, dass die Medien zurückgegeben werden. Nach der Medienrückgabe muss lediglich einmal auf Aktualisieren gedrückt werden.

Dieses Formular ist mit Hilfe von Abfragen wesentlich komplexer strukturiert als die vorher vorgestellte Variante. Mehr zu den Grundlagen ist deshalb im Kapitel «Abfragen» zu erfahren.

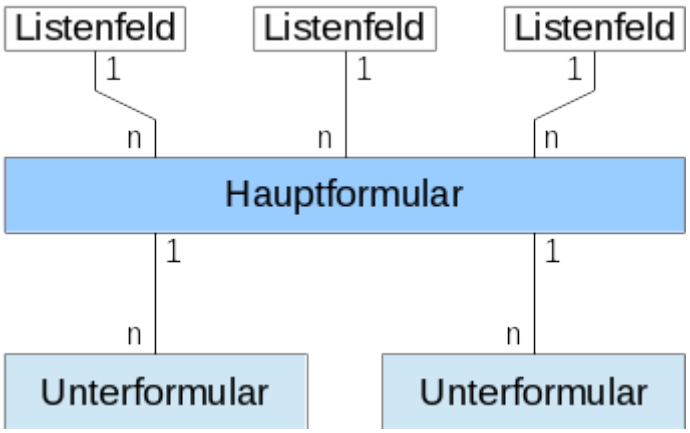
Eine Ansicht – viele Formulare

Während das Beispiel für die Ausleihe nur Eingaben in eine Tabelle erforderte (Tabelle "Ausleihe") und zusätzlich die Eingabe in der einfacheren Form auch für neue Leser ermöglichte, ist die Eingabe für die Medien wesentlich umfassender. Schließlich spielen rund um die Medien-Tabelle insgesamt 8 zusätzliche Tabellen mit (siehe *Tabellen Medienaufnahme*).

Durch die Zuordnung im Verhältnis n:1 bieten sich die Tabellen "Untertitel" und "rel_Medien_Verfasser" als Unterformulare zum Formular «Medien» an. Tabellen, die hingegen ein Verhältnis von 1:n zur Tabelle "Medien" haben, müssten eigentlich ein Formular bilden, das über dem der Tabelle "Medien" liegt. Da es sich aber um mehrere entsprechende Tabellen handelt, werden deren Werte über Listenfelder in das Hauptformular eingetragen.

Die Tabelle eines Hauptformulars steht zur Tabelle eines Unterformulars grundsätzlich im Verhältnis 1:n, in seltenen Ausnahmen im Verhältnis 1:1. Das Hauptformular beherbergt in der Regel nach längerem Gebrauch der Datenbank also eine Tabelle, die deutlich weniger Datensätze hat als die Tabelle des Unterformulars.

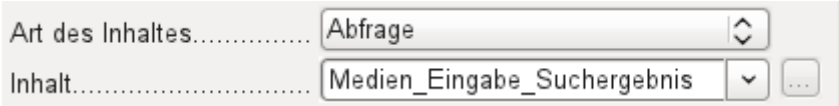
Mehrere Hauptformulare können nicht auf ein Unterformular zugreifen. Es ist also nicht möglich, viele 1:n-Beziehungen gleichzeitig über die Formularanordnung zu lösen, bei denen das Unterformular den gleichen Inhalt hat. Gibt es eine 1:n-Beziehung für die Tabelle eines Formulars, so lässt sich dies über ein Listenfeld regeln. Hier stehen wenige Begriffe aus einer anderen Tabelle zur Auswahl, deren Fremdschlüssel auf diese Art in die Tabelle des Hauptformulars eingetragen werden.



Über Listenfelder werden dem Hauptformular, das auf der Tabelle "Medien" basiert, z. B. die Inhalte der Tabellen "Kategorie", "Ort" oder "Verlag" zugewiesen. Über Unterformulare sind die

Tabelle "rel_Medien_Verfasser" und "Untertitel" mit dem Hauptformular und damit mit der Tabelle "Medien" verbunden.

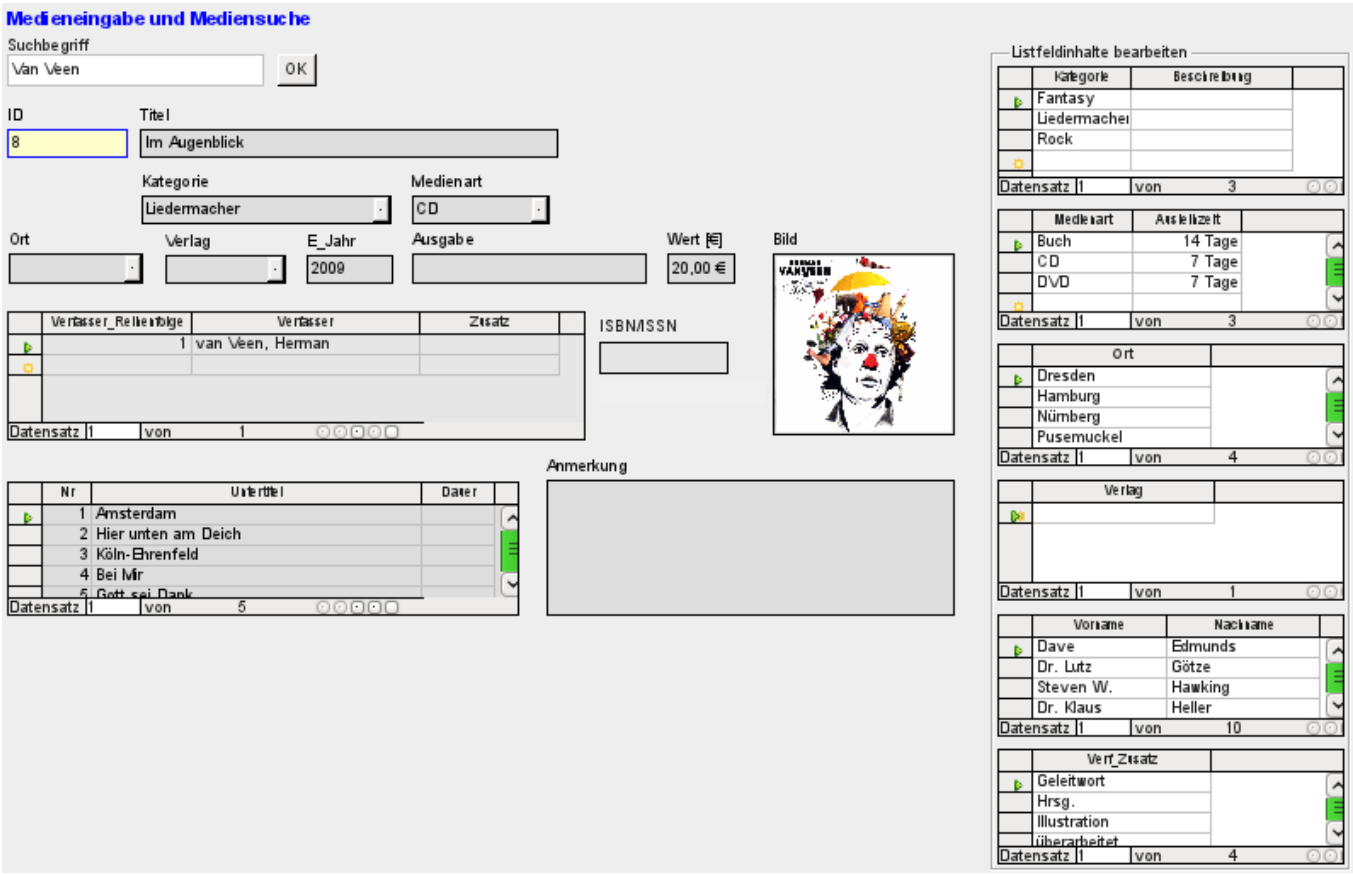
Das Unterformular für die Tabelle "rel_Medien_Verfasser" besteht außerdem wieder aus zwei Listenfeldern, damit nicht die Fremdschlüssel der Tabelle "Verfasser" und "Verf_Zusatz" (Zusätze wie 'Hrsg.', 'Gesang' usw.) direkt als Ziffern eingegeben werden müssen.



Die Datengrundlage für das Formular zur Medieneingabe ist in diesem Fall nicht eine Tabelle, sondern eine Abfrage. Dies ist notwendig, da das Formular nicht nur zur Eingabe sondern auch zur Suche benutzt werden soll. Auch wird in dem Formular noch in einem Textfeld nach dem Abspeichern darüber aufgeklärt, ob die eingegebene ISBN-Nummer korrekt ist. Auch dies ist nur über eine umfassende Abfrage möglich. Um diese Hintergründe zu verstehen, ist es also notwendig, sich erst einmal mit den Grundlagen von Abfragen in dem folgenden Kapitel auseinander zu setzen.

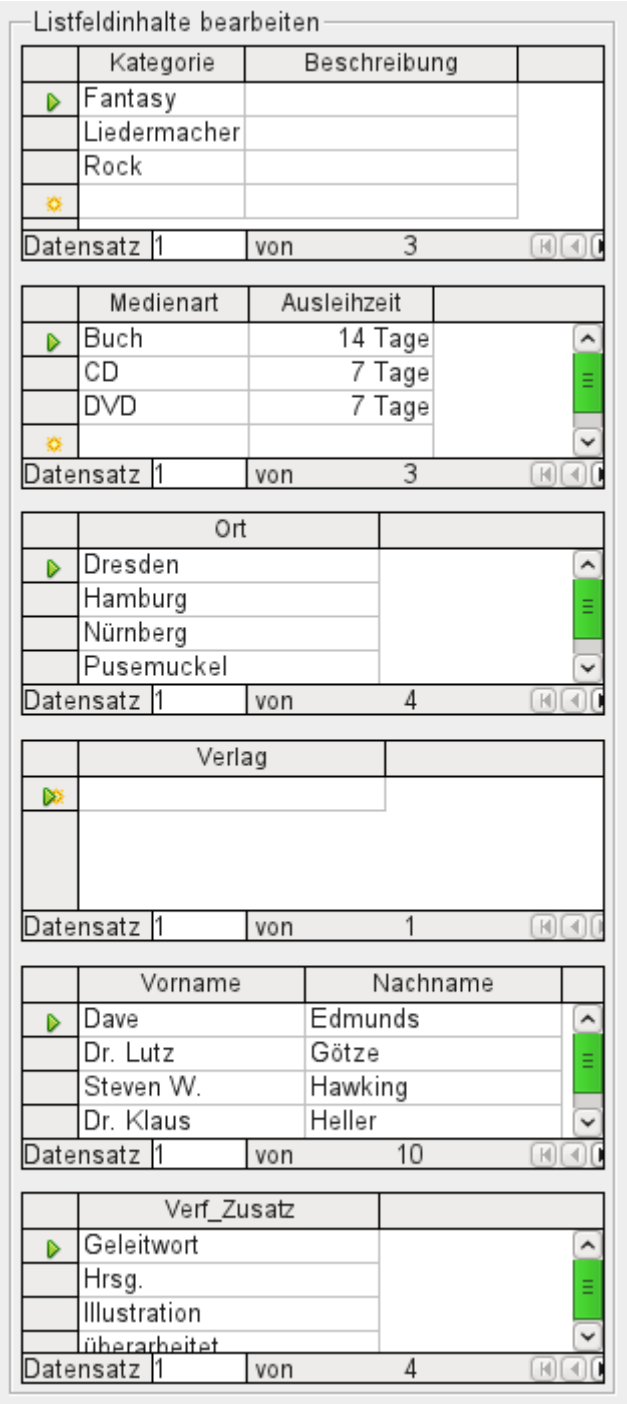
Bei dem Formular für die Medieneingabe müssen die Listenfelder meist während der Eingabe nach und nach aufgefüllt werden. Hierzu werden neben dem Hauptformular weitere Formulare eingebaut. Sie existieren unabhängig vom Hauptformular:

Das gesamte Formular zur Medieneingabe sieht so aus:



Auf der linken Seite befindet sich das Hauptformular mit Blick auf die Suche und Eingabe von neuen Medien. Auf der rechten Seite des Formulars ist durch einen Gruppierungsrahmen mit der Bezeichnung «Listfeldinhalte bearbeiten» eine Bereich abgegrenzt, der zum Auffüllen der Listenfelder (hier verkürzt: «Listfeld») im Hauptformular gedacht ist. Existiert die Datenbank erst kurz, so wird hier wohl häufig eine Eingabe erledigt werden müssen. Je mehr Eingaben allerdings in den Listenfeldern des Hauptformulars zur Verfügung stehen, desto seltener ist ein Zugriff auf die Tabellenkontrollfelder aus dem Gruppierungsrahmen notwendig.

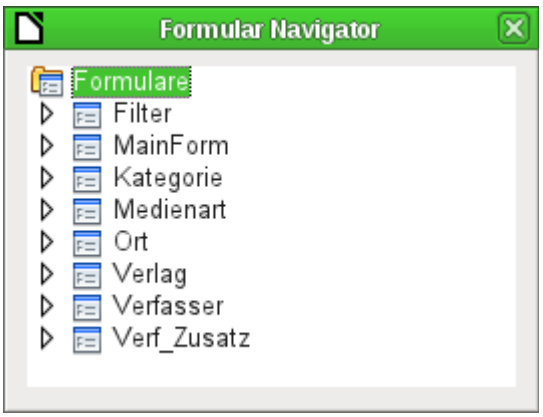
Die folgenden Tabellenkontrollfelder sind alle in einzelnen Nebenformularen zum Hautformular, dem Eingabeformular, untergebracht:



Hier werden jeweils die kompletten Daten für eine Tabelle eingegeben. Am Anfang ist es häufig erforderlich, auf diese Nebenformulare auszuweichen, da z. B. nicht viele Verfasser in der entsprechenden Tabelle bereits abgespeichert wurden.

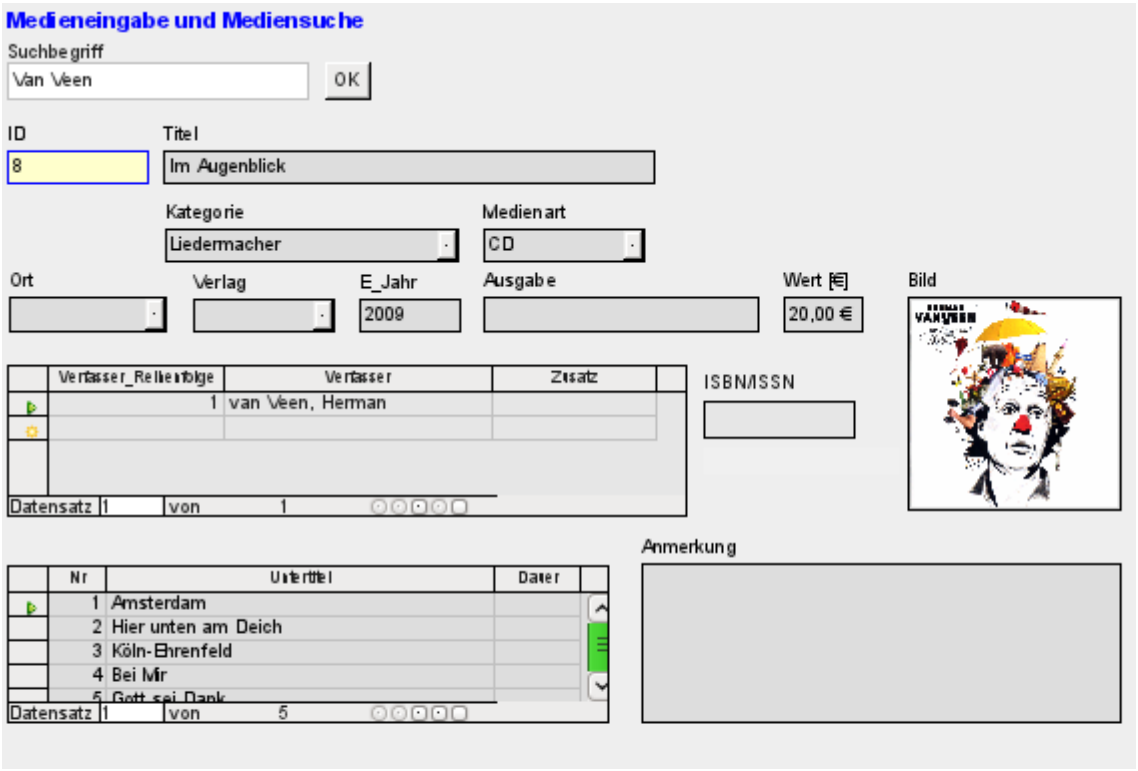
Wurde in einem der Tabellenkontrollfelder ein neuer Datensatz abgespeichert, so ist in dem Hauptformular das entsprechende Listenfeld aufzusuchen und über **Kontrollfeld aktualisieren** (siehe «Navigationsleiste») neu einzulesen.

Der Formularnavigator zeigt entsprechend viele Formulare an:



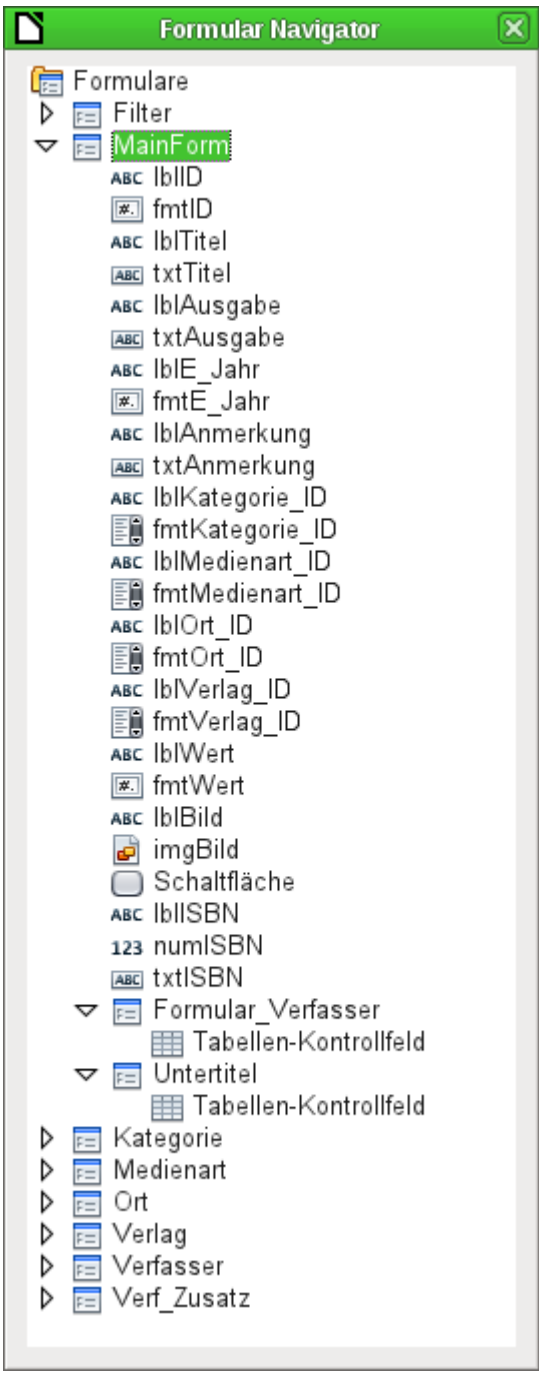
Die Formulare sind einzeln benannt, so dass sie erkennbar bleiben. Lediglich das Hauptformular hat vom Formularassistenten noch die Bezeichnung «MainForm» behalten. Insgesamt existieren also 8 Formulare parallel. Das Formular «Filter» beherbergt eine Suchfunktion, das Formular «MainForm» die Haupteingabefläche. Alle anderen Formulare stehen für je eins der oben abgebildeten Tabellenkontrollfelder.

Ohne die Tabellenkontrollfelder erscheint das Hauptformular schon etwas übersichtlicher:



Das Feld für den Suchbegriff liegt in dem *Nebenformular* «Filter», die beiden *Tabellenkontrollfelder* (für die Verfasser und für die Untertitel) liegen in *Unterformularen* zum Hauptformular der Medieneingabe.

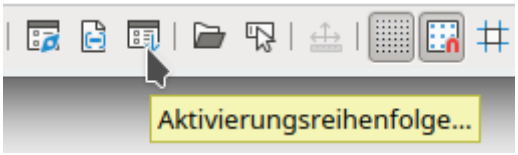
Im Formularnavigator sieht das Formular selbst dann schon wesentlich unübersichtlicher aus, da natürlich alle Kontrollfelder, auch die Beschriftungen, dort auftauchen. In den vorhergehenden Formularen waren ja die meisten Felder als Spalten von Tabellen-Kontrollfeldern im Formularnavigator nicht zu sehen.



Die Reihenfolge innerhalb des Formularnavigators lässt sich leider nicht so einfach verändern. So erscheint es z. B. sinnvoll, die Unterformulare «Untertitel» und «Formular_Verfasser» als Verzweigung direkt zum Beginn des Formulars «MainForm» auftauchen zu lassen. Innerhalb des Formularnavigators werden die einzelnen Kontrollfelder und Unterformulare einfach in der Reihenfolge aufgelistet, in der sie erstellt wurden.

Durch den Formularassistenten werden die Elemente mit bestimmten Kürzeln versehen, die neben den Symbolen andeuten, um welche Art Feld es sich handelt. Mit 'lbl' beginnen Beschriftungsfelder ('label'), mit 'txt' Textfelder usw. Beschriftungsfelder interessieren für die Eingabe von Daten eigentlich nur als zusätzliche Informationen, können aber auch als *Sprungziel* dienen. Würden sie direkt über Textrahmen erstellt, so würden sie im Formularnavigator nicht erscheinen.

Die Reihenfolge, in der die Elemente im Navigator auftauchen, hat aber nichts damit zu tun, in welcher Reihenfolge die Elemente durch einen Tabulatorsprung erreicht werden. Dies wird durch die *Aktivierungsreihenfolge* bestimmt.



Die Aktivierungsreihenfolge für ein bestimmtes Formular wird aufgerufen, indem ein Element dieses Formulars markiert wird und dann erst der Button Aktivierungsreihenfolge betätigt wird. Bei einem einzigen Formular ist diese Reihenfolge natürlich nicht notwendig, da nur ein Formular zur Verfügung steht. Die Funktion muss aber bei vielen parallel liegenden Formularen erst einmal wissen, welches Formular denn nun ausgewählt werden soll. Standardmäßig ist es sonst das erste Formular im Formularnavigator – und das enthält im obigen Beispiel nur ein Textfeld.

Über die Aktivierungsreihenfolge werden alle Elemente, die Daten an die dem Formular zugrundeliegende Tabelle weitergeben oder Aktionen hervorrufen können, in ihrer Reihenfolge zueinander festgelegt. Dies entspricht der Einstellung in den Eigenschaften in *Standardeinstellungen vieler Kontrollfelder* zur Aktivierungsreihenfolge.

In der Aktivierungsreihenfolge tauchen allerdings auch die Elemente auf, bei denen der Tabstop eigentlich abgeschaltet ist. Sie werden zwar in die Nummerierung aufgenommen, aber tatsächlich bei der Arbeit mit dem Formular über die Tastatur nicht angesprungen.

Automatisch werden die Felder nach der Lage auf dem Formularhintergrund sortiert. Je weiter oben ein Feld liegt, desto eher wird es über die Auto Sortierung angesprungen. Je weiter links ein Feld bei gleicher Höhe liegt, desto eher wird es angesprungen. Diese Sortierung funktioniert dann tadellos, wenn die Elemente wirklich genau ausgerichtet sind (Raster bei der Formularerstellung). Ansonsten muss nachgebessert werden. Hierzu wird ein Element einfach markiert und z. B. über Nach unten weiter nach unten in der Reihenfolge verlegt.

Existiert ein Unterformular, so wird bei der Auto Sortierung das Ganze so eingestellt, dass nach dem **Hauptformular direkt in das Unterformular** gesprungen wird. Bei einem Tabellenkontrollfeld führt dies über die Tastatureingabe dann dazu, dass der Cursor in diesem Unterformular gefangen ist und nur mit der Maus oder über **Strg + Tab** aus dem Subformular wieder herausbewegt werden kann.

Hinweis

Der Sprung ins Unterformular findet nur dann statt, wenn das Unterformular auch im Formularnavigator unterhalb der Felder des Hauptformulars liegt. Eine Sortierung im Formularnavigator ist leider nicht möglich, so dass gegebenenfalls ein Unterformular nach Erstellung des kompletten Hauptformulars neu erstellt werden muss. Dann können die Elemente des ursprünglichen Unterformulars dorthin verschoben und das ursprüngliche Unterformular gelöscht werden. Jetzt klappt auch der Sprung in das Unterformular, nachdem die Auto Sortierung erneut aufgerufen wurde.
Manchmal muss auch bei der Auto Sortierung für den Formularsprung getrickst werden. Sie funktioniert nur, wenn in einem Formular wenigstens 2 Felder sind, die überhaupt sortiert werden können. Bei lediglich einem Feld im Hauptformular muss dann vorübergehend ein zweites Feld aus der Datenquelle hinzugefügt werde. Wird dann die Auto Sortierung aufgerufen, so klappt der Sprung ins Unterformular auch, wenn das überflüssige Feld anschließend wieder entfernt wird.

Die Auto Sortierung funktioniert allerdings, bezogen auf das Tabellenkontrollfeld, nur einmal. Ein weiteres Unterformular mit Tabellenkontrollfeld wird nicht mit einbezogen. Parallel liegende Formulare werden also nicht berücksichtigt. Eine Auto Sortierung kann in Bezug auf ein Unterformular mit Tabellenkontrollfeld auch nicht rückgängig gemacht werden. Hierzu müsste das Unterformular komplett entfernt (oder vorübergehend in ein anderes Formular verschoben) werden.

Fehlermeldungen bei der Eingabe in Formulare

Einige Fehlermeldungen treten gerade bei den ersten Formularerstellungen gehäuft auf, so dass sie hier kurz erklärt werden sollen:

`001` **Attempt to insert null into a non-nullable column: column: ID table: Tabellenname in statement …**

Es gibt Felder, die nicht leer sein dürfen. Wird dies nur in der Tabelle definiert (**NOT NULL**), so erscheint eine englischsprachige Fehlermeldung. Erfolgt die Definition auch in den Kontrollfeldern des Formulars, dann erscheint eine deutschsprachige Meldung mit dem genauen Hinweis, welches Feld denn nun ausgefüllt werden muss.

Die obige Meldung passiert besonders häufig, wenn das Primärschlüsselfeld, hier "**ID"**, nicht in das Formular mit aufgenommen wurde, da es ja als automatisch hoch zählendes AutoWert-Feld gedacht war. Leider wurde aber die entsprechende Definition als AutoWert-Feld schlicht vergessen. Also muss dies entweder nachgeholt werden oder das Feld muss im Formular mit erscheinen, damit ein Wert eingegeben werden kann.

Die nachträgliche Definition eines Feldes als AutoWert-Feld ist manchmal etwas schwierig, wenn die Tabelle mit anderen in der Beziehungsdefinition bereits verknüpft wurde oder wenn bereits mit einer Ansicht auf die Tabelle zugegriffen wird. Hier müssen alle Verbindungen gelöst werden, damit die Tabelle gerade im Bereich des Primärschlüssels wieder editierbar wird. Der Inhalt des SQL-Befehls, mit dem z. B. eine Ansicht erstellt wurde, kann ja gegebenenfalls als Abfrage zwischendurch abgespeichert werden.

`001` **Integrity constraint violation - no parent SYS_FK_95 table: Tabellenname in statement …**

Hier besteht eine Verknüpfung der dem Formular zugrundeliegenden Tabelle mit einer anderen Tabelle. Diese Tabelle soll ein Fremdschlüsselfeld mit seinem Primärschlüssel belegen. Dies geschieht in der Regel durch Listenfelder, die den Inhalt der Tabelle richtig abfragen. Ist aber gar kein Listenfeld vorhanden oder die Konstruktion des Listenfeldes falsch, so kann in das Fremdschlüsselfeld auch irrtümlich ein Wert eingegeben werden, der in der zweiten Tabelle gar nicht als Primärschlüssel vorhanden ist. Dies wird als **Integritäts-Verletzung** bezeichnet. **'no parent SYS_FK_95'** deutet darauf hin, dass in der zweiten Tabelle, der abgebenden

Elterntabelle, der entsprechende Indexwert mit dem Namen **'SYS_FK_95'** nicht vorhanden ist.

```
001 Fehler beim Einfügen des neuen Datensatzes.
    Fehler in der Funktionsfolge
```

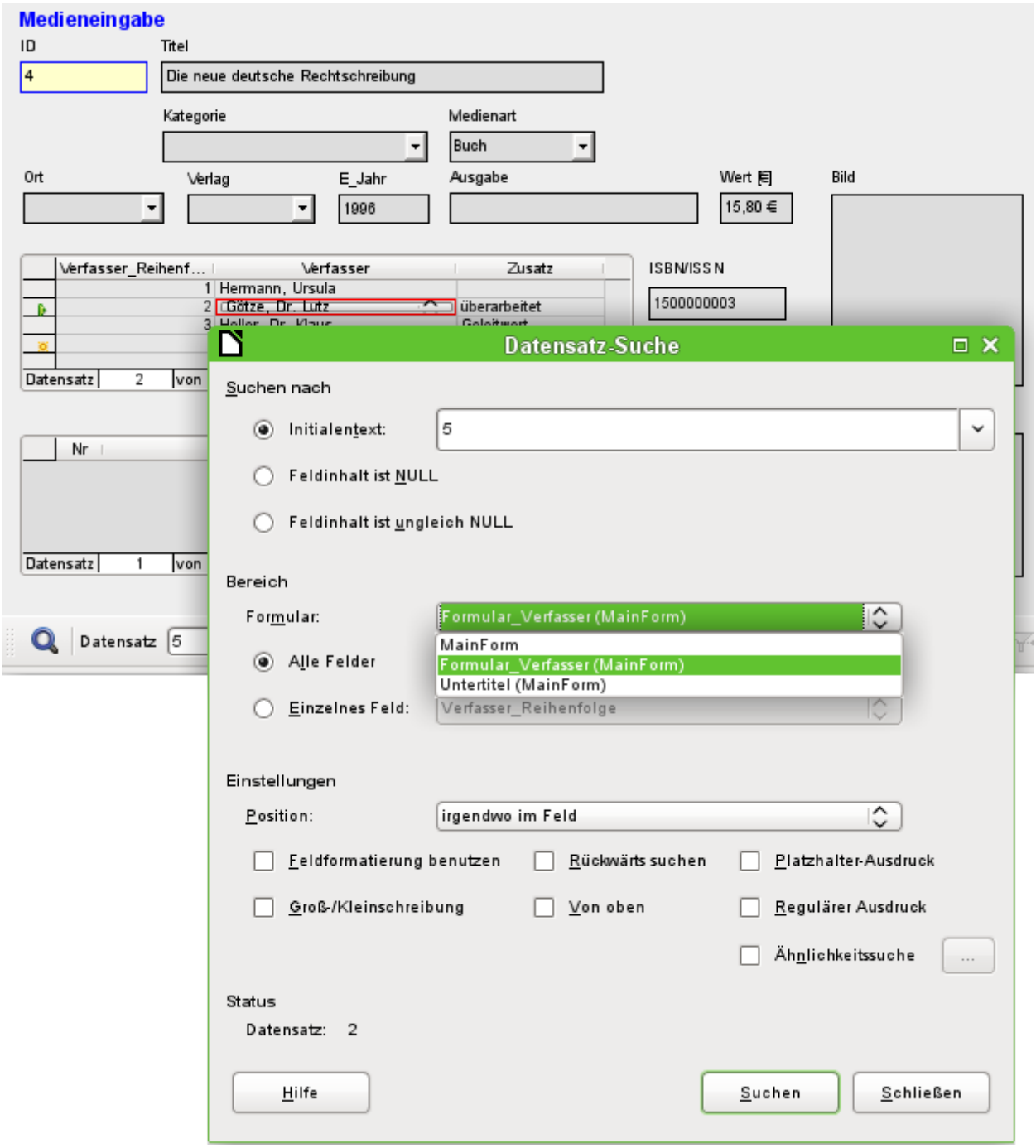
Einem Unterformular werden Daten von den darüber liegenden Formularen weitergegeben. Dies sieht Base nicht als eine Änderung eines Feldes an. Die grafische Benutzeroberfläche bietet zwar das Speichern an – nur scheint der Datensatz dann leer zu sein. Hier hilft es, ein einfaches Feld wie z. B. ein Ja/Nein-Feld in die dem Formular zugrundeliegende Tabelle einzubauen. Jetzt wird vor dem Abspeichern dieses Feld angeklickt und der Datensatz kann problemlos gespeichert werden.

Die über die anderen Formulare weitergegebenen Werte werden anscheinend erst dann in die entsprechenden Felder eingefügt, wenn wenigstens eine zusätzliche Aktion im Formular stattfindet.

Suchen und Filtern in Formularen über die Navigationsleiste

Die Navigationsleiste der Formulare bietet verschiedene Such- und Filtermöglichkeiten an. Die einzelnen Elemente der Navigationsleiste wurden bereits weiter oben aufgezählt.

Datensatzsuche mit Parametern



Die einfache Datensatzsuche mit Parametern wurde bereits im Kapitel «Tabellen» vorgestellt. Sie bietet sehr umfangreiche Einstellmöglichkeiten. Charakteristik der Suche ist, dass grundsätzlich alle Datensätze durchsucht werden und nicht die Datensätze so eingeschränkt werden, dass nur Datensätze mit dem Suchwert zu sehen sind.

Die Datensatzsuche im Formular bietet die Möglichkeit, das entsprechende Formular sowie die Unterformulare zu durchsuchen. Es ist allerdings nicht möglich, alle Formulare gleichzeitig zu durchsuchen.

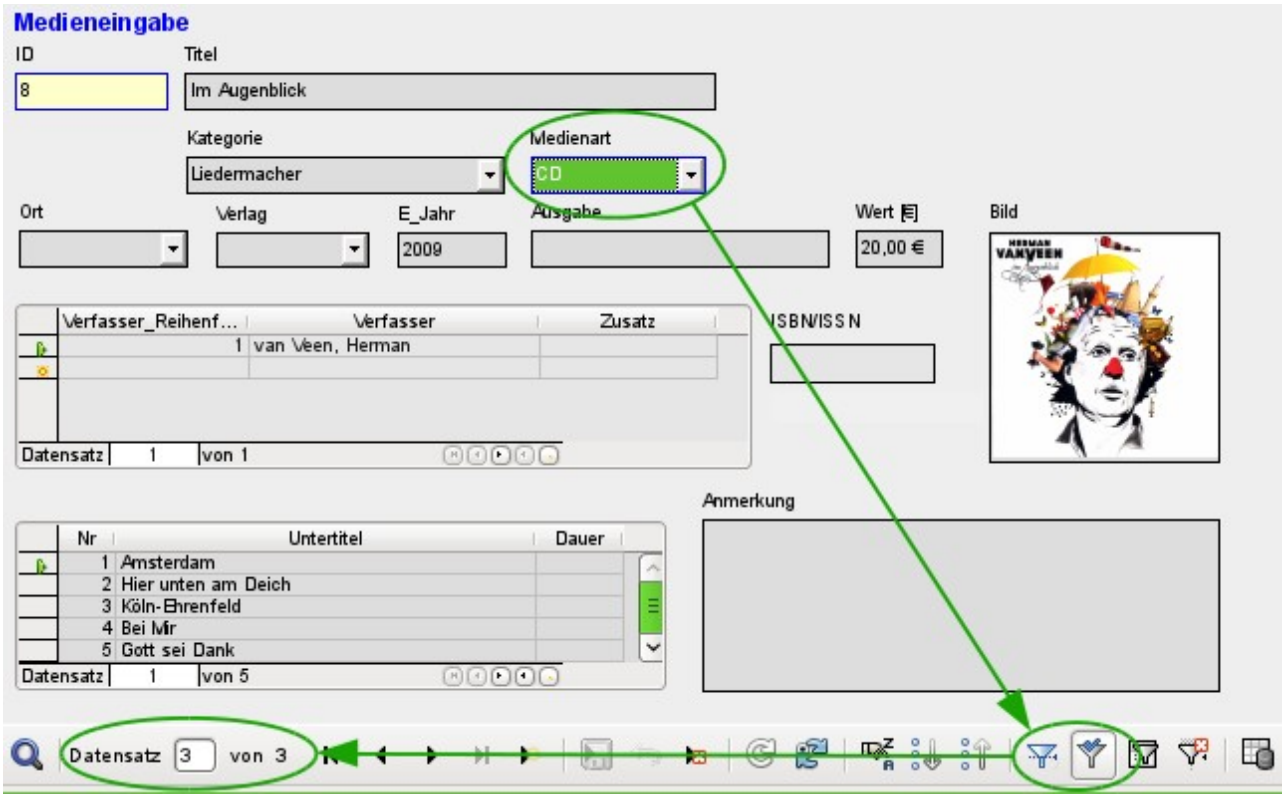
Das Bild zeigt den Zugriff auf das Unterformular «Formular_Verfasser» des Hauptformulars «MainForm». Dabei wird nach einem Listenfeld, nämlich dem Feld für den Verfasser, gesucht. Allerdings wird nicht der Text des Listenfeldes benötigt, sondern der Wert, der als Fremdschlüsselwert an dieser Stelle eingegeben wird.

Leider kennzeichnen diese Suche aber die folgenden Nachteile:

- Die Parametersuche ist für den Normalgebrauch viel zu umständlich zu handhaben.
- Die Suchfunktion selbst ist äußerst langsam, da nicht die Abfragefunktionen der Datenbank genutzt werden.
- Die Suche funktioniert nur innerhalb eines Formulars, dort dann aber auch gegebenenfalls für alle Felder. Unterformulare müssen separat angewählt werden.
- Es wird nur in dem Unterformular gesucht, das zu dem aktuellen Hauptformular passt. Einträge anderer Unterformulare lassen sich nicht ermitteln. So ist es z.B. nicht möglich, einen Untertitel eines anderen Mediums als des gerade angezeigten Mediums zu finden.
- Die Suche funktioniert bei Listenfeldern auf der Basis des Schlüsselfeldes (Fremdschlüssels), das in der Tabelle abgespeichert ist. Es lassen sich nicht die angezeigten Begriffe verwenden.

Filterung mit dem Autofilter

Der Autofilter kann direkt in der Navigationsleiste angewählt werden. Wird auf ein Feld im Hauptformular geklickt, so wird treffsicher nach diesem Feld gefiltert. Der Autofilter funktioniert auch sicher mit Listenfeldern, hat hier also einen klaren Vorteil gegenüber der händischen Eingabe von Fremdschlüsselwerten zu verbuchen.



In dem obigen Beispiel wurde einfach ein Datensatz gesucht, bei dem im Feld «Medienart» 'CD' steht. Anschließend wurde der Autofilter betätigt. Von den ursprünglich 9 Datensätzen der Medien-Tabelle werden jetzt noch 3 Datensätze in der Datensatzzählung angezeigt.

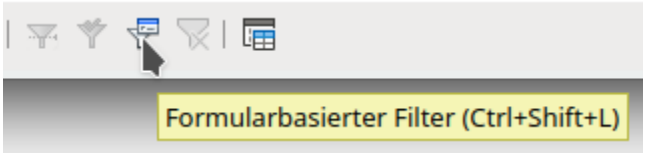
Obwohl nur in zwei Datensätzen im Unterformular eine «Nr.» '3' wie oben existiert, werden alle anderen Datensätze des Hauptformulars weiter angezeigt. Im Unterformular erscheinen hingegen nur die Felder mit «Nr.» '3'.

Der Autofilter kann die folgenden Probleme allerdings nicht lösen:

- Die Suche in Unterformularen führt dazu, dass zwar in den Unterformularen nur der entsprechende Wert angezeigt wird, die Formulare selbst aber weiter komplett angezeigt werden, auch wenn sie den entsprechenden Wert im Unterformular gar nicht verzeichnen können. Wird z.B. nach einem Medium mit einem zweiten Verfasser gesucht, so werden trotzdem auch alle Medien angezeigt, für die nur ein Verfasser verzeichnet ist. Hier bleibt das Feld für die Verfasser schlicht leer.
- Die Wahl des Filterinhaltes muss eindeutig sein. Es gibt nicht die Möglichkeit, über Ähnlichkeitssuche zu einem entsprechenden Wert zu kommen, da eben nur der Originalwert aus dem Formular verwandt wird.

Filterung mit dem formularbasierten Filter

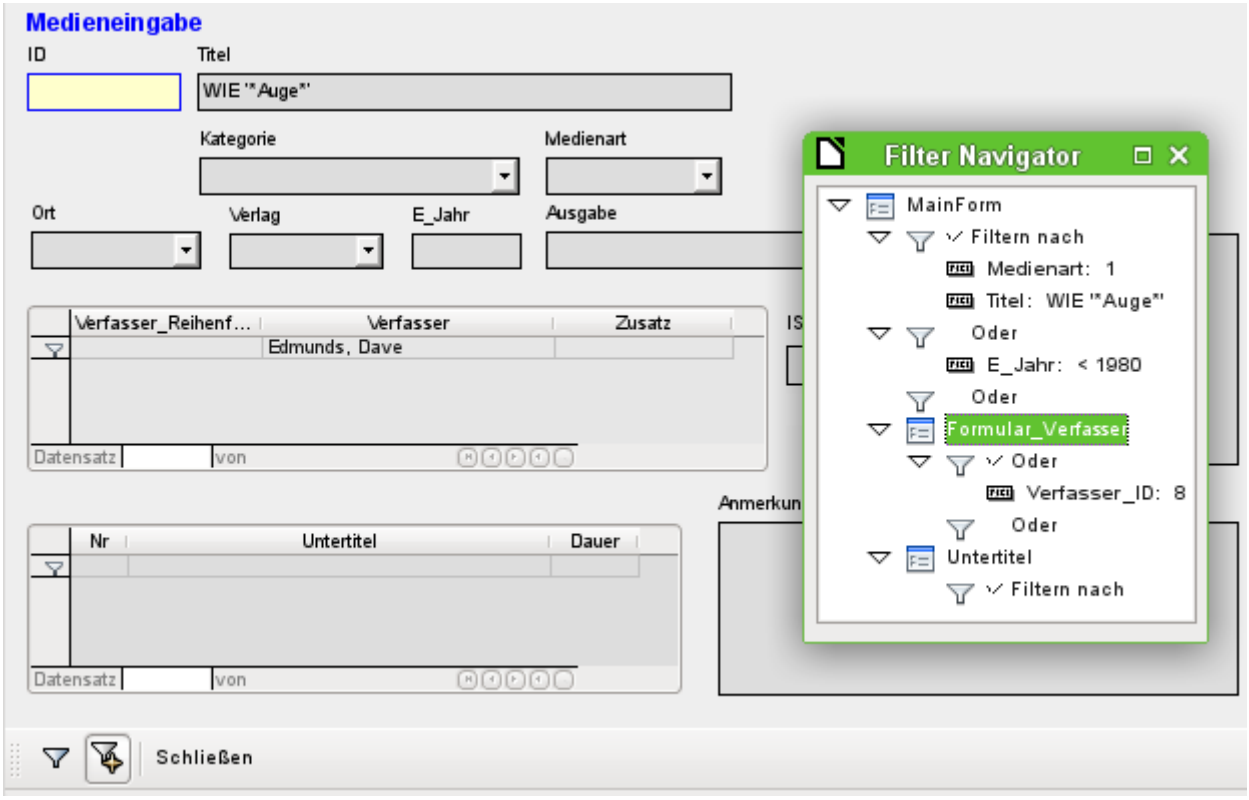
Der formularbasierte Filter bietet statt eines Filtersymbols das Formular zur Eingabe von entsprechenden Filterwerten an. Hier kann gleichzeitig nach mehreren Werten gefiltert werden. Diese Werte können entweder zusammen als gemeinsame Bedingung formuliert werden (UND) oder als unterschiedliche Bedingungen gesetzt werden (ODER).



Start des formularbasierten Filters

Die Eingabe der Filterwerte muss nicht eindeutig sein. Hier hilft dann die Formulierung von Bedingungen, wie sie bei der Filterung von Tabellen geschildert wurde. So führt ein «WIE '%Auge%'», eingetragen in das Eingabefeld für den Titel des Mediums, direkt zur Auffindung aller Titel, in denen 'Auge' enthalten ist – in der Beispieldatenbank «Medien_ohne_Makros.odb» z.B. zur Anzeige des Titels 'Im Augenblick'.

Filter für die Listenfelder werden direkt über die Listenfelder ausgewählt. Eine Eingabe von Fremdschlüsselwerten ist also nicht nötig.



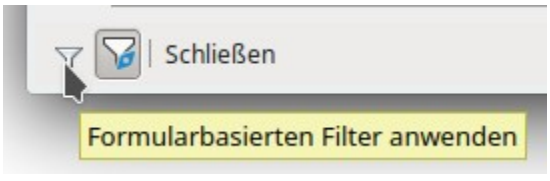
Hier werden alle Datensätze gesucht, die die Medienart 'CD' haben (entspricht dem Fremdschlüsselwert '1') und gleichzeitig im Titel irgendwo die Buchstabenfolge 'Auge' haben. Bei der Eingabe «WIE '%Auge%'» ersetzt die grafische Benutzeroberfläche das in SQL übliche Allroundzeichen für mehrere Werte «%» durch das Zeichen «*», das für viele Nutzer eher gebräuchliche Allroundzeichen.

Neben der Bedingung dürfen auch alle Datensätze des Hauptformulars angezeigt werden, bei denen die Jahresangabe «< 1980» ist.

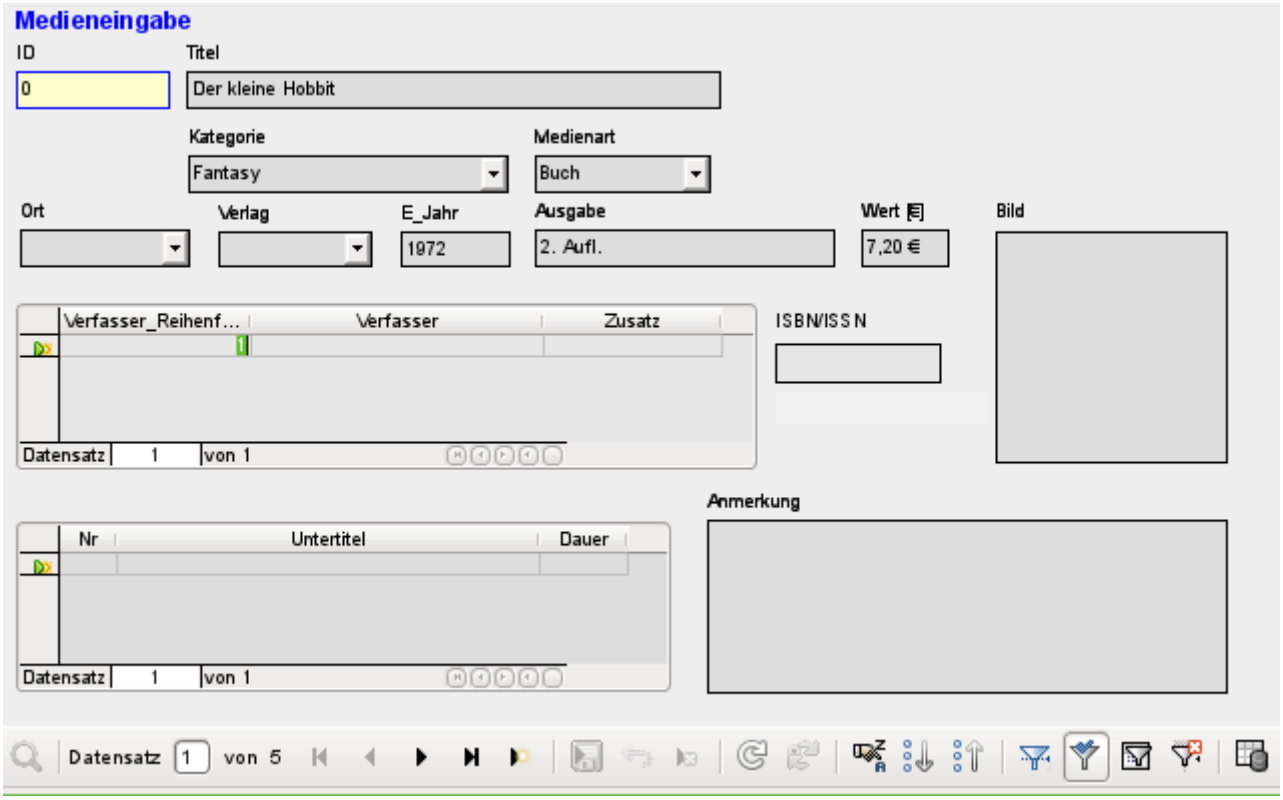
Aus dem Unterformular wird noch der Verfasser 'Edmunds, Dave' aus dem Listenfeld ausgewählt. Der entsprechende Fremdschlüsselwert ist hier die '8'.

Der formularbasierte Filter zeigt bei der Eingabe nur die Felder an, die vor dem ODER mit einem Haken versehen sind. So lassen sich auch mehrere Bedingungen für ein Feld angeben. Leider unterschlägt der Filter dabei die Anzeige des Listenfeldes «Medienart» mit dem entsprechenden Wert im Hauptformular, während dies im Tabellenkontrollfeld des Unterformulars gelingt.

Statt der Navigationsleiste werden während des Erstellens des Filters unter dem Formular lediglich drei Auswahlmöglichkeiten für den formularbasierten Filter dargestellt: Filtern nach den Vorgaben, Filterwerte hinzufügen und die Anzeige des formularbasierten Filters einfach schließen.



Der Filter wird angewandt und das Formular nach den vorgegebenen Werten gefiltert.



Ist die Filterung aktiv, so wird die Zahl der Datensätze auf die Datensätze mit einem Treffer eingeschränkt. Hier gibt es insgesamt 5 Datensätze mit Treffern. In dem obigen Bild stimmt zwar nicht der Titel und die Medienart, dafür aber die Bedingung, dass das Erscheinungsjahr vor 1980 liegen soll.

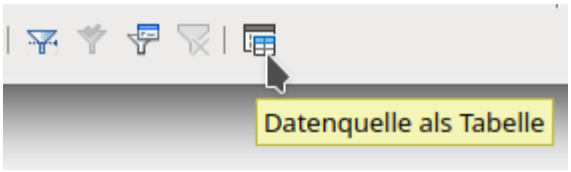
Eine Merkwürdigkeit zeigt sich im Unterformular. Da auch hier eine Filterung vorgenommen wurde, wird der Verfasser des Buches 'Der kleine Hobbit' nicht angezeigt. Der Filterwert war mit dem Verfasser 'Dave Edmunds' verbunden. Dies zeigt wieder die Logik, die hinter der Filterung in allen Fällen steckt: Es werden die Filtereigenschaften des jeweiligen Formulars genutzt. Unterformulare werden separat gefiltert und eine Filterung der Unterformulare wirkt sich nicht auf das Hauptformular aus – und umgekehrt.

Diese schon sehr gelungene Filterfunktion leidet allerdings an den folgenden Nachteilen:

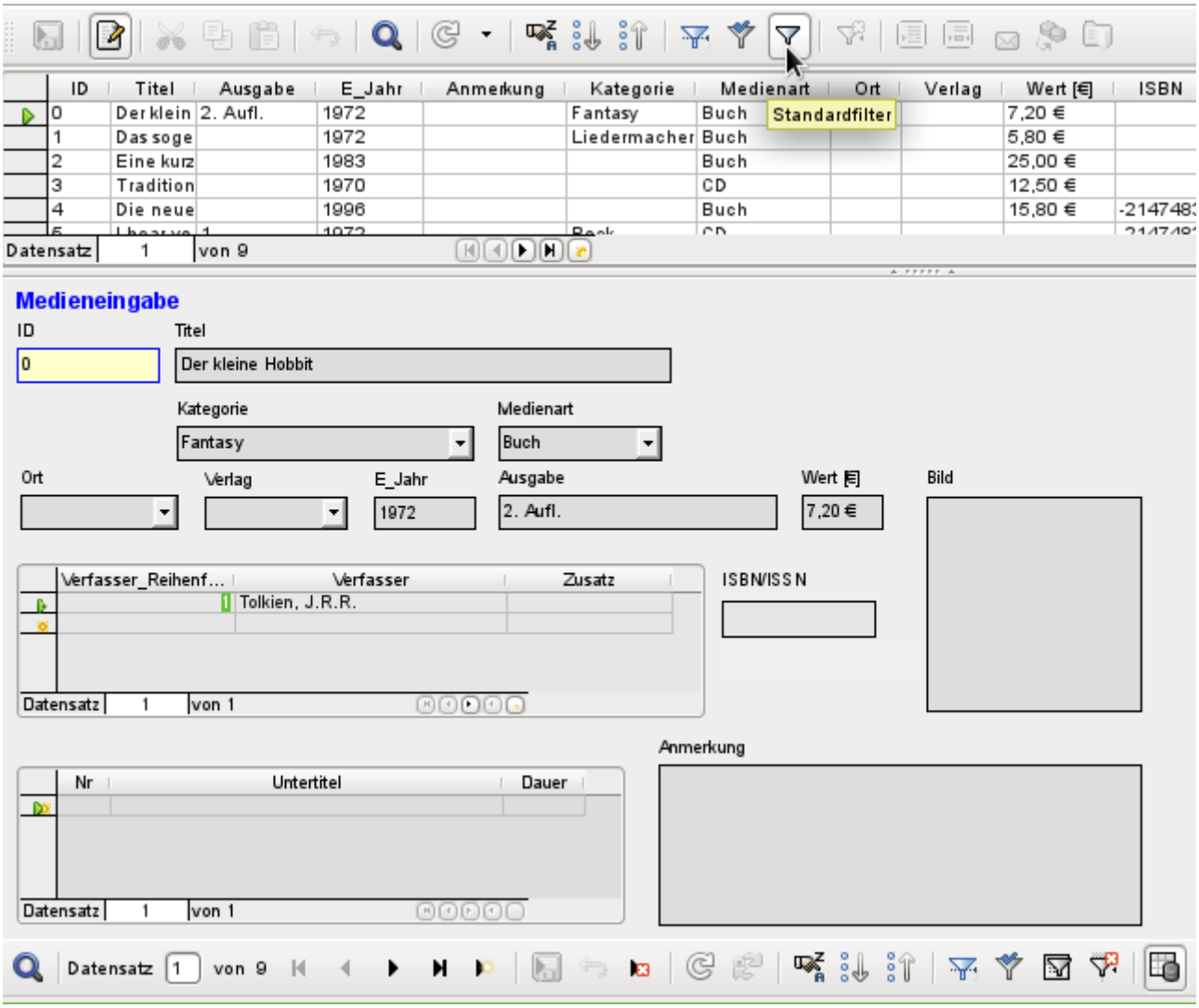
- Die Filterung funktioniert wieder nur im Hauptformular. Im Unterformular werden wie bei Autofilter zwar die fehlenden Werte unterdrückt. Dies wirkt sich aber nicht auf die Anzeige im Hauptformular aus. Es werden also im Hauptformular auch Inhalte angezeigt, die den gefilterten Inhalt aus dem Unterformular nicht vorweisen können.
- Es sind umfangreiche Kenntnisse der Bedingungsformulierung notwendig, wenn mehr als nur die kompletten Begriffe gesucht werden sollen. Hier müssen Kenntnisse über Abfragetechniken vorhanden sein, die aber den meisten suchmaschinenverwöhnten Nutzern fehlen.

Filterung über den Standardfilter

Die Filterung über den Standardfilter kann durch die Anzeige der Datenquelle als Tabelle erreicht werden. Dort gelten dann die gleichen Regeln wie bei der Filterung mit dem Standardfilter in Tabellen.



Start des Standardfilters über «Datenquelle als Tabelle»



Rechts unten im Formularnavigator wird die Datenquelle als Tabelle eingeblendet. In der Tabellensicht steht dann auch der Standardfilter zur Verfügung. Im Gegensatz zur Ansicht der eigentlichen Tabelle "Medien" werden hier die Fremdschlüsselfelder nicht mit ihren Schlüsselwerten, sondern mit den Inhalten dargestellt. Ein Klick auf das Feld «Medienart» zeigt, dass hier ein Listenfeld nach den Vorgaben des Formulars erstellt wurde.

Hinweis

Die obige Ansicht zeigt dann leider auch gleich einen Bug, der seit den Anfängen von Base wohl schon existiert: Die 13-stellige ISBN-Nummer wird nicht korrekt dargestellt. Das entsprechende Feld wird stattdessen mit der niedrigst möglichen Integer-Zahl versehen. Hat eine Zahl mehr als 9 Stellen, so ist hier also Vorsicht geboten: Datenänderung kann zu Datenverlusten führen. (*Bug 82411*)
Dieser Bug tritt auch in Tabellenkontrollfeldern auf. Die Darstellung ist korrekt, wenn statt eines Zahlenfeldes im Formular ein formatierbares Feld genommen wird.

Die Tabellenansicht des Formulars hat die folgenden Nachteile für diese Recherchemethode:

- Die Suche funktioniert nur innerhalb der einem Formular zugrundeliegenden Tabelle oder Abfrage, nicht in einem dazugehörigen Unterformular.
- In der Datenansicht werden Listenfelder angezeigt, nach denen aber nicht gefiltert werden kann. Wieder müssen die Fremdschlüssel für die Felder bekannt sein, damit eine Filterung Erfolg hat. Hat z.B. die "Medienart" 'Buch' den Primärschlüsselwert '1', so wird in der Tabelle "Medien" der Fremdschlüssel '1' gespeichert. Nach diesem Schlüsselwert ist zu suchen – obwohl die eingeblendete Ansicht 'Buch' in einem Listenfeld zeigt.

Fazit

Sämtliche Such- und Filterfunktionen eignen sich für Formulare nur eingeschränkt. Die eingebaute Suche ist viel zu langsam und schränkt die Zahl der Datensätze nicht auf die Datensätze mit entsprechenden Treffern ein. Die Filter funktionieren grundsätzlich nur innerhalb eines Formulars. Soll nach Werten im Unterformular gefiltert werden, so ist nur das aktuelle Unterformular betroffen. Es werden einfach nicht alle Datensätze korrekt ermittelt.

Aus diesem Grund wurde weiter oben in die Formulare der Beispieldatenbank eine entsprechend einfacher zu bedienende Suchfunktion integriert, die allerdings etwas Handarbeit und SQL-Kenntnis verlangt.

Dateneingabe und tastaturgesteuerte Navigation

Ein Formular kann so eingestellt werden, dass mit der Funktion **Automatischer Steuerelement-Fokus** über den Button im Formularentwurf oder über den Formular-Navigator der Cursor direkt im ersten Eingabefeld steht. Statt mit der Maus von einem Feld zum anderen zu springen, ist bei Formularen die Bewegung mit der Tabulatortaste von einem Feld zum nächsten üblich. Wird auch noch im Hauptformular die **Aktivierungsreihenfolge → Automatische Sortierung** eingestellt, so wird auch ein Sprung vom Hauptformular direkt ins Unterformular möglich.

Manchmal erfordern Formulare auch Sprünge zu einem anderen Formularfeld. Diese können natürlich mit der Maus geregelt werden, sind aber auch über Kurzbefehle anspringbar.

Dem Textfeld ist über **Eigenschaften: Textfeld → Beschriftungsfeld** ein Beschriftungsfeld zugewiesen. Die Beschriftung des Beschriftungsfeldes wurde von «Name» auf «Na~me» geändert. Dadurch erscheint im Formular jetzt ein Unterstrich unter dem «m». Das Textfeld kann jetzt bei der Eingabe mit dem Kürzel *Alt* + *m* angesprungen werden. Dies funktioniert allerdings nur, wenn der Cursor bereits in einem Formularfeld steht.

Prinzipiell können alle Buchstaben als Sprungziel genommen werden, da der Sprung innerhalb des Formulars erfolgt. Steht der Cursor aber nicht in einem Formularfeld, so werden mit den Kürzeln eventuell Sprungziele aus der LibreOffice-Oberfläche angesteuert. Mit dem Sprungziel «a» z. B. wäre dies dann statt des Feldes «Name» das Öffnen des Menüs «Ansicht».

Hinweis

Die Zuweisung eines solchen Sprungzieles funktioniert nur, wenn gleichzeitig bei den Formularfeldern durchweg die **Aktivierungsreihenfolge → '0'** gesetzt wird. Hinzu kommt noch, dass das Beschriftungsfeld vor dem Formularfeld erstellt werden muss. Der Assistent zur Erstellung von Formularen macht dies automatisch. Diese Anordnung ist im Formularnavigator zu sehen.

Tipp

Die Position im Formularnavigator entscheidet darüber, welches Feld als nächstes angesprungen wird, sofern die Aktivierung nicht über die entsprechende Reihenfolge geregelt wurde. Leider lässt sich diese Position nicht im Nachhinein durch einfaches Verschieben verändern.
Um die Postion zu verändern wird ein neues Formular parallel zu dem bestehenden Formular mit gleicher Datenanbindung erstellt. Die Felder werden jetzt aus dem alten Formular in das neue Formular in der passenden Reihenfolge hinein gezogen. Dabei werden immer zuerst die Beschriftungsfelder und dann die eigentlichen Kontrollfelder verschoben.
Auch Unterformulare müssen zuerst in dem neuen parallel erstellten Formular erstellt werden. Es reicht nicht aus, ein komplettes Unterformular zu verschieben. In dem Moment bleibt nämlich die interne Positionierung aus dem vorhergehenden Hauptformular erhalten.

Das Anspringen von Feldern mit Sprungzielen bei einer Konstruktion mit Unterformularen ist von der Formulareigenschaft **Zyklus** abhängig. Ein Sprung aus einem Unterformular zu einem Hauptformular ist nur möglich, wenn bei den Formulareigenschaften **Zyklus → Aktuelle Seite** gewählt wird.

Hinweis

Tabellenkontrollfelder zeigen unter **Allgemein → Tabulator → 'Nein'** standardmäßig an. Leider verhalten sie sich aber nicht direkt nach dem Erstellen so (*https://bugs.documentfoundation.org/show_bug.cgi?id=51727*). Damit wirklich kein Tabulatorsprung in ein Tabellenkontrollfeld stattfindet muss diese Einstellung einmal auf 'Ja' und danach wieder auf 'Nein' gesetzt werden.

Aus einem Tabellenkontrollfeld sollte der Cursor mit *Strg* + *Tab* herausspringen. Manchmal kann es passieren, dass die Tastenkombination nicht den gewünschten Erfolg erzielt. Daher hier eine völlig andere Möglichkeit:[11]

In dem Unterformular wird eine Schaltfläche angelegt. Der Titel der Schaltfläche heißt z.B. «~neu». Das «~» sorgt, wie oben erwähnt, dafür, dass der Button mit dem Kürzel *Alt* + *N* ausgelöst werden kann. In den Zusatzinformationen des Buttons wird außerdem noch der Name des Feldes im Hauptformular benannt, zu dem der Cursor springen soll. Die Schaltfläche wird über **Eigenschaften → Ereignisse → Aktion ausführen** mit dem folgenden Makro verbunden:

```
001 SUB JumpToMainform(oEvent AS OBJECT)
002    DIM oField AS OBJECT
003    DIM oForm AS OBJECT
004    DIM oDoc AS OBJECT
005    DIM oController AS OBJECT
006    DIM oView AS OBJECT
007    DIM stShortcut AS STRING
008    oField = oEvent.Source.Model
009    stShortcut = Mid(oField.Label, InStr(oField.Label, "~") + 1, 1)
010    oForm = oField.Parent.Parent
011    SELECT CASE stShortcut
012       CASE "n"
013          oForm.MoveToInsertRow()
014       CASE "ä"
015          IF oForm.isLast() THEN
016             oForm.MoveToInsertRow()
017          ELSE
018             oForm.Next()
019          END IF
020    END SELECT
```

11 Siehe hierzu auch die Beispieldatenbank «Beispiel_Cursorsprung_Subform_Mainform.odb»

```
021     oDoc = thisComponent
022     oController = oDoc.getCurrentController()
023     oView = oController.getControl(oForm.getByname(oField.Tag))
024     oView.setFocus
025 END SUB
```

Das Makro kann dazu genutzt werden, nur ins Hauptformular zurück zu springen. Es kann bei der obigen Einstellung auch dazu dienen, direkt einen neuen Datensatz anzulegen oder auch über einen weiteren Button mit dem Kürzel *Alt* + *ä* (für «nächster Datensatz») zum nächsten Datensatz zu springen. Befindet sich der Cursor des Hauptformulars allerdings schon auf dem letzten Datensatz, so muss der Sprung zu einem neuen Datensatz erfolgen. Weiteres zu Makros siehe im entsprechenden Kapitel.

Der Sprung auf die Schaltfläche gelingt nur mit einer sichtbaren Schaltfläche. Die Schaltfläche kann aber ruhig eine ganz geringe Ausdehnung oder eine Breite von 0 cm und eine Höhe von 0 cm haben. Bei der Definition einer nicht vorhandenen Breite und einer nicht vorhandenen Höhe kann es allerdings vorkommen, dass der Button als Strich im Formular über die gesamte Bildschirmbreite sichtbar wird.

Hinweis

Bei Tabellenkontrollfeldern funktioniert das Setzen des Focus nicht wie bei einfachen Formularfeldern. Hier muss das Tabellenkontrollfeld angesteuert werden und dann über

```
001 oView.CurrentColumnPosition = 2
```

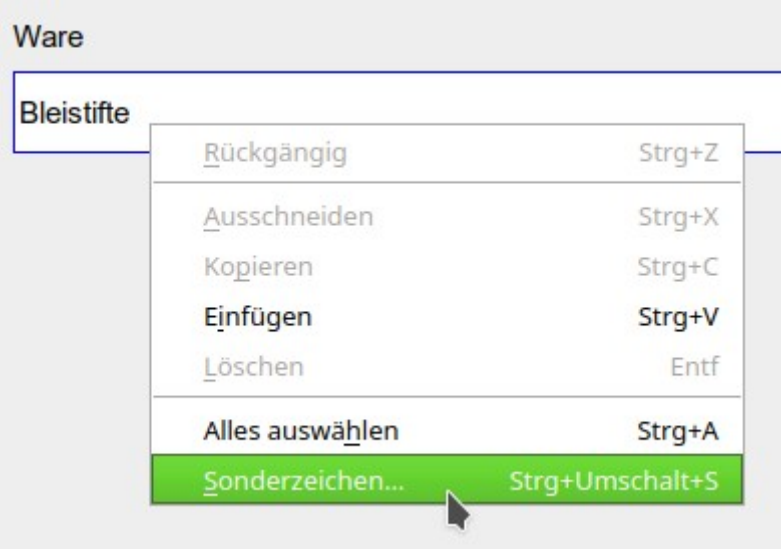
z. B. das Feld in der 2. Spalte angesteuert werden.

Weitere Möglichkeiten zur Navigation befinden sich im Kapitel «Makros».

Die **Abspeicherung** von Daten wird ausgelöst durch

- die Betätigung eines Speicherbuttons in der Navigationsleiste oder im Formular selbst
- durch den Wechsel zum nächsten Datensatz
- durch das Klicken auf ein Element, das nicht zum Formular gehört. Dies kann der Hintergrund des Formulars oder auch ein Button sein, der in einem Nebenformular liegt. Dies wird bei den Such- und Filterfunktionen häufig genutzt, um mit einem Button Daten abzuspeichern und gleichzeitig ein anderes Nebenformular zu aktualisieren.

Sollen in ein Formularfeld Sonderzeichen eingegeben werden, so geht das nicht über den Menüeintrag des Writers. Schließlich ist das Formular selbst schreibgeschützt. Nur über die Formularfelder kann ein Eintrag erfolgen.



Ist ein entsprechendes Eingabefeld aktiv, so wird mit der rechten Maustaste über das Kontextmenü der Dialog zur Eingabe von Sonderzeichen aufgerufen. Auch über *Strg* + *Umschalt* + *S* ist die Sonderzeichenauswahl erreichbar.

Hier einige feststehenden Tastenkombinationen, die im Zusammenhang mit Formularen von Bedeutung sind:

Befehl	***Tastenkombination***
Nächster Datensatz	*Strg* + *Alt* + →
Vorheriger Datensatz	*Strg* + *Alt* + ←
Letzter Datensatz	*Strg* + *Alt* + *Ende*
Erster Datensatz	*Strg* + *Alt* + *Pos 1*
Aus dem Tabellenkontrollfeld heraus	*Strg* + *Tab*
Sonderzeichen einfügen	*Strg* + *Umschalt* + *S*
Formularbasierten Filter starten	*Strg* + *Umschalt* + *L*
Befehle innerhalb einer Listbox	
Nächster Datensatz	↓ (zusammen mit *Alt* : aufklappen)
Vorheriger Datensatz	↑ (zusammen mit *Alt* : aufklappen)
Letzter Datensatz	*Ende* (zusammen mit *Alt* : aufklappen)
Erster Datensatz	*Pos 1* (zusammen mit *Alt* : aufklappen)
Listbox zuklappen	*ESC*

Weitere Befehle können beim zum Bearbeiten geöffneten Formular über **Extras → Anpassen → Tastatur** hinzugefügt werden.

Drucken aus Formularen

Formulare können so aufgebaut werden, dass ein direkter Druck der Ansicht möglich ist. Um dabei zu brauchbaren Ergebnissen zu kommen, muss allerdings darauf geachtet werden, Elemente nicht außerhalb des druckbaren Bereiches zu verschieben. Dazu wird beim Bearbeiten des Formulars **Ansicht → Drucklayout** gewählt. Die Seiteneigenschaften können eingestellt werden. Ein farbiger Hintergrund ist hier sicher nicht von Vorteil.

Jedes einzelne Formularelement kann in seinen **Eigenschaften → Allgemein → Druckbar** von einem Druck ausgeschlossen werden.

Ist die Datenbank in LibreOffice registriert worden (über **Extras → Optionen → LibreOffice Base → Datenbanken** oder direkt beim Erstellen der Datenbank), so kann ein Formular auch zum Seriendruck genutzt werden. Das Formular wird zum Bearbeiten geöffnet. Über **Ansicht → Datenquellen** oder mit F4 werden die Datenquellen zugänglich. Felder aus der Datenbank können jetzt über die Tabellenköpfe in das Formular gezogen werden. Das Formular wird anschließend abgespeichert. Wird das Formular anschließend wie zur Eingabe geöffnet, so erkennt LibreOffice, dass Serienbrieffelder enthalten sind und fragt nach, ob ein Serienbrief gedruckt werden soll.

Details zu der Erstellung von Serienbriefen sind in dem Kapitel «Datenbank-Anbindung» enthalten.

Darstellungsgröße von Formularen

Beim Öffnen von Formularen kann der Nutzer manchmal von einer unüblichen Zoom-Einstellung des Formulars überrascht werden. Die Darstellungsgröße des Formulars richtet sich leider nicht nach der Zoom-Einstellung des Formulars beim Speichern. Vielmehr übernimmt Base die jeweils letzte Einstellung des Zooms von einem vorhergehenden Writer-Dokument.

Der Zoom kann über die Zoomeinstellung oder den Maßstab am unteren rechten Fensterrand eingestellt werden. Eine andere Einstellungsmöglichkeit ist über das Drücken von *Strg* und das gleichzeitige Betätigen des Scrollrades der Maus gegeben.

Wer automatisch bei Start eines Formulars immer die gleiche Größe erreichen möchte, kann dies über das folgende Makro erreichen:

```
001  SUB Formularzoom
002     ThisComponent.CurrentController.ViewSettings.ZoomValue=100
003  END SUB
```

Dieses Makro soll beim Öffnen des Formulars ausgelöst werden. Das Formular muss zum Bearbeiten geöffnet werden. Unter **Extras → Anpassen → Ereignisse → Dokument öffnen** wird das Makro eingebunden.

Abfragen

Allgemeines zu Abfragen

Abfragen an eine Datenbank sind das mächtigste Werkzeug, was uns zur Verfügung steht, um Datenbanken sinnvoll zu nutzen. Sie fassen Daten aus unterschiedlichen Tabellen zusammen, berechnen gegebenenfalls irgendwelche Ergebnisse, filtern einen ganz bestimmten Datensatz aus einer Unmenge an Daten mit hoher Geschwindigkeit heraus. Die großen Internetdatenbanken, die viele täglich nutzen, haben ihren Hauptsinn darin, dass aus der Unmenge an Informationen durch geschickte Wahl der Schlüsselwörter schnell ein brauchbares Ergebnis für den Nutzer geliefert wird – einschließlich natürlich der zu den Suchbegriffen gehörenden Anzeigen, die zum Kauf animieren sollen.

Eingabemöglichkeiten für Abfragen

Die Eingabe von Abfragen kann sowohl in der GUI als auch direkt per SQL erfolgen. In beiden Fällen öffnet sich ein Fenster, das es ermöglicht, die Abfragen auszuführen und gegebenenfalls zu korrigieren.

Abfrageerstellung mit der grafischen Benutzeroberfläche

Die Erstellung von Abfragen mit dem Assistenten wird hier nicht dargestellt, da der Assistent zwar bei wenig Tabellen schnell zu einem Ergebnis führt, für eine größere Datenbank aber wenig zielführend ist. Hier wird stattdessen die direkte Erstellung über **Abfrage in der Entwurfsansicht erstellen** erklärt.

Nach einem Aufruf der Funktion erscheinen zwei Fenster. Ein Fenster stellt die Grundlagen für den Design-Entwurf der Abfrage zur Verfügung, das andere dient dazu, Tabellen, Ansichten oder Abfragen der Abfrage hinzuzufügen.

Da unser einfaches Formular auf der Tabelle "Ausleihe" beruhte, wird zuerst einmal an dieser Tabelle die Grundkonstruktion von Abfragen mit dem Abfrageeditor erklärt.

Aus den zur Verfügung stehenden Tabellen wird die Tabelle "Ausleihe" ausgewählt. Dieses Fenster bietet die Möglichkeit, gleich mehrere Tabellen (und unter diesen auch Ansichten) sowie Abfragen miteinander zu kombinieren. Jede gewünschte Tabelle wird markiert (linke Maustaste) und dann dem grafischen Bereich des Abfrageeditors hinzugefügt.

Sind alle erforderlichen Tabellen ausgewählt, so wird dieses Fenster geschlossen. Gegebenenfalls können später noch mehr Tabellen und Abfragen hinzugefügt werden. Ohne eine einzige Tabelle lässt sich aber keine Abfrage erstellen, so dass eine Auswahl zu Beginn schon sein muss.

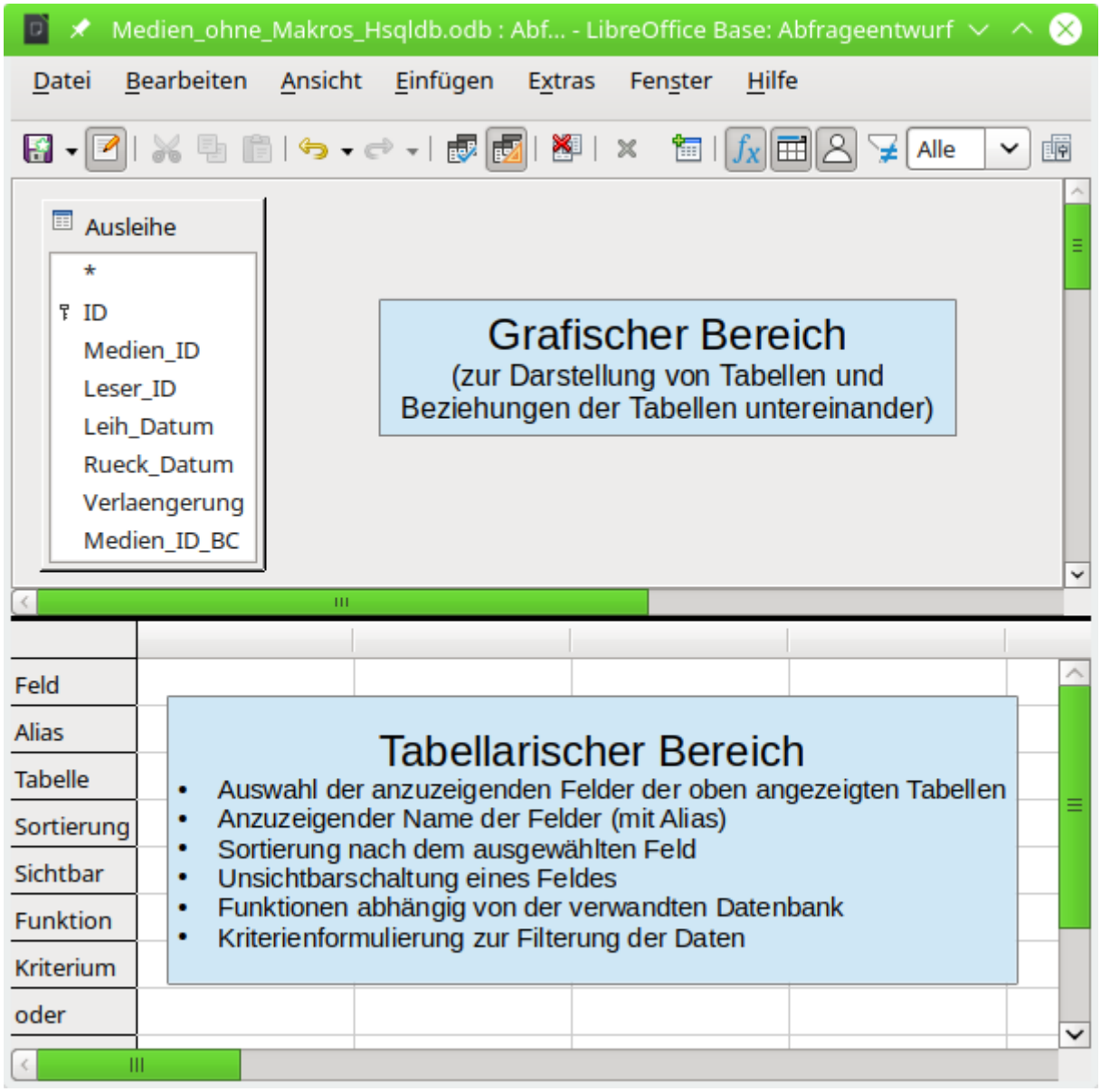


Abbildung 38: Bereiche des Abfrageentwurfs

Abbildung 38 zeigt die grundsätzliche Aufteilung des grafischen Abfrageeditors: Im grafischen Bereich werden die Tabellen angezeigt, die mit der Abfrage zusammen hängen sollen. Hier kann auch ihre Beziehung zueinander in Bezug auf die Abfrage angegeben werden. Im tabellarischen Bereich erfolgt die Auswahl der Felder, die angezeigt werden sollen, sowie Bedingungen, die mit diesen Feldern verbunden sind.

Standardmäßig werden im Abfrageeditor 2 Symbolleisten angezeigt:

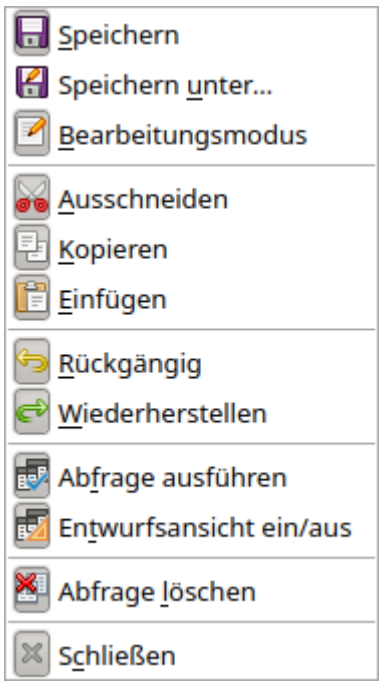


Abbildung 39: Symbolleiste «Abfrageentwurf»

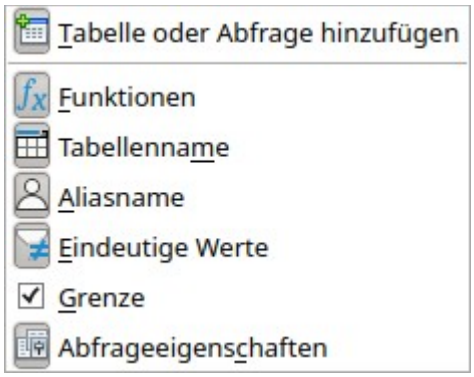
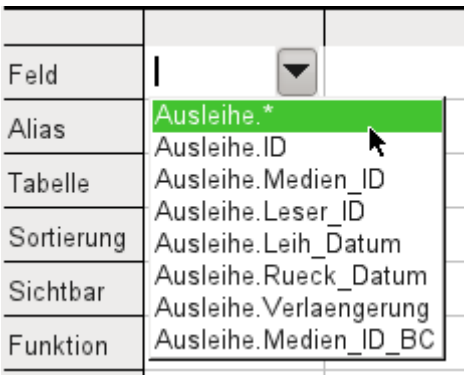


Abbildung 40: Symbolleiste «Design»

Ein Klick mit der Maustaste auf das Feld der ersten Spalte im tabellarischen Bereich öffnet die Feldauswahl:

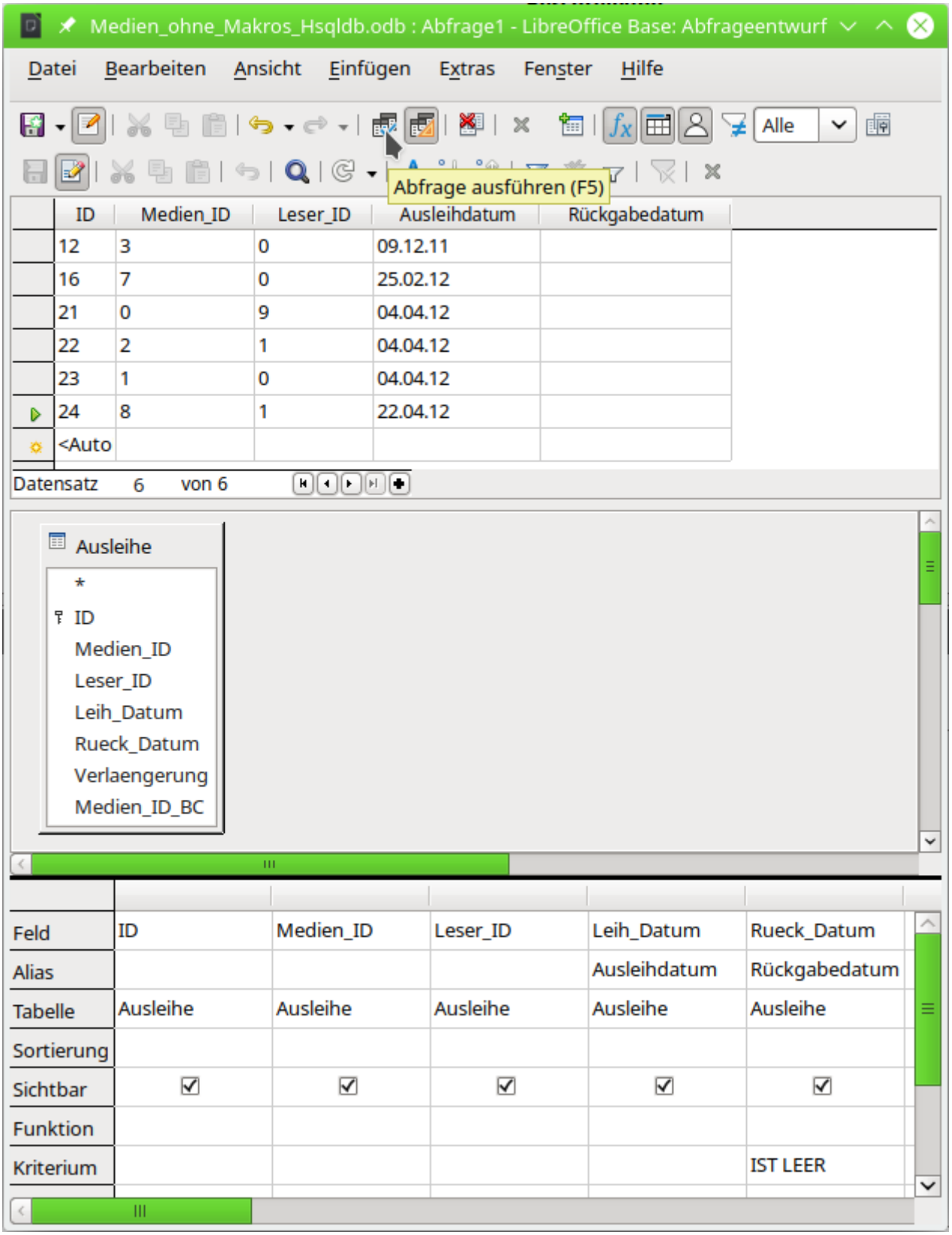


Hier stehen jetzt alle Felder der Tabelle "Ausleihe" zur Verfügung. Die Schreibweise ist: **Tabellenname.Feldname** – deshalb beginnen alle Feldbezeichnungen hier mit dem Begriff "Ausleihe.".

Eine besondere Bedeutung hat die markierte Feldbezeichnung **Ausleihe.*** . Hier wird mit einem Klick jeder Feldname der zugrundeliegenden Tabelle in der Abfrage wiedergegeben. Wenn allein diese Feldbezeichnung mit dem Jokerzeichen «*» für alle Felder gewählt wird, unterscheidet sich die Abfrage nicht von der Tabelle.

Tipp

Sollen schnell alle möglichen Felder aus Tabellen in die Abfrage übertragen werden, so kann auch direkt im grafischen Bereich auf die Tabellenübersicht geklickt werden. Mit einem Doppelklick auf ein Feld wird das Feld im tabellarischen Bereich an der nächsten freien Position eingefügt.



Es werden die ersten fünf Felder der Tabelle Ausleihe ausgewählt. Abfragen können auch im Design-Modus immer wieder testweise ausgeführt werden. Dann erscheint oberhalb der grafischen Ansicht der Tabelle eine tabellarische Übersicht über die Daten. Die testweise Ausführung von Abfragen ist vor dem Abspeichern immer sinnvoll, damit für den Nutzer klar ist, ob die Abfrage auch wirklich das erreicht, was sie erreichen soll. Manchmal wird durch einen Denkfehler ausgeschlossen, dass eine Abfrage überhaupt je Daten ausgeben kann. Ein anderes Mal

kann es passieren, dass plötzlich genau die Datensätze angezeigt werden, die ausgeschlossen werden sollten.

Grundsätzlich lässt sich eine Abfrage, die eine Fehlerrückmeldung bei der zugrundeliegenden Datenbank produziert, gar nicht erst abspeichern.

	ID	Medien_ID
▶	12	3
	16	7
	23	1
	22	2
	24	8
	21	0
✱	<Auto	

Abbildung 41: Abfrage bearbeitbar

	Medien_ID	Leser
▶	3	0
	7	0
	1	0
	2	1
	8	1
	0	9

Abbildung 42: Abfrage nicht bearbeitbar

Besonderes Augenmerk sollte in dem obigen Test einmal auf die erste Spalte des dargestellten Abfrageergebnisses geworfen werden. Auf der linken Seite der Tabelle erscheint immer der Datensatzmarkierer, der hier auf den ersten Datensatz als aktivem Datensatz hinweist. Während in *Abbildung 41* aber das erste Feld des ersten Datensatzes grün markiert ist, zeigt das erste Feld in *Abbildung 42* nur eine gestrichelte Umrandung an. Die grüne Markierung deutet bereits an, dass hier im Feld selbst etwas geändert werden kann. Die Datensätze sind also änderbar. In *Abbildung 41* ist außerdem eine zusätzliche Zeile zur Eingabe neuer Daten vorhanden, in der für das Feld "ID" schon *<AutoWert>* vorgemerkt ist. Auch hier also sichtbar, dass Neueingaben möglich sind.

Grundsätzlich sind dann keine Neueingaben möglich, wenn der Primärschlüssel der abgefragten Tabelle nicht in der Abfrage enthalten ist.

Feld	ID	Medien_ID	Leser_ID	Leih_Datum	Rueck_Datum
Alias				Ausleihdatum	Rückgabedatum

Den Feldern "Leih_Datum" und "Rueck_Datum" wurde ein Aliasname zugewiesen. Sie wurden damit nicht umbenannt, sondern unter diesem Namen für den Nutzer der Abfrage sichtbar gemacht.

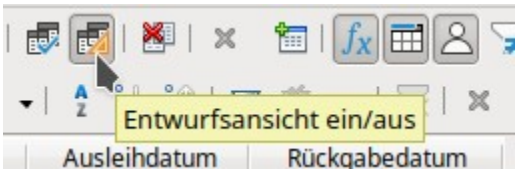
	ID	Medien_ID	Leser_ID	Ausleihdatum	Rückgabedatum
▶	12	3	0	09.12.11	

Entsprechend ist in der Tabellenansicht der Alias statt der eigentlichen Feldbezeichnung zu sehen.

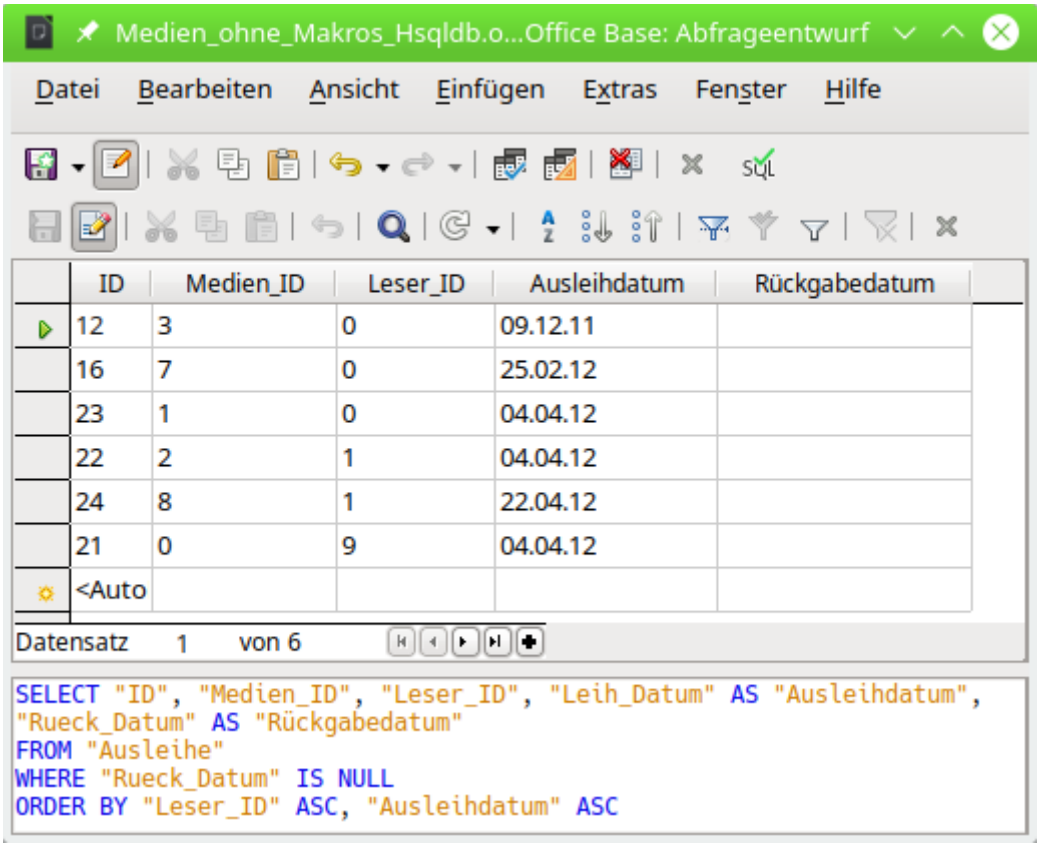
Rueck_Datum
Rückgabedatum
Ausleihe
☑
IST LEER

Dem Feld "Rueck_Datum" wurde nicht nur ein Alias, sondern auch ein Kriterium zugewiesen, nach dem nur die Datensätze angezeigt werden sollen, bei denen das Feld "Rueck_Datum" leer ist. Die Angabe erfolgt hier in deutscher Sprache, wird dann aber in der eigentlichen Abfrage in SQL übersetzt.

Durch dieses Ausschlusskriterium werden nur die Datensätze von Medien angezeigt, die ein Medium enthalten, das noch nicht zurückgegeben wurde.



Um die SQL-Sprache besser kennen zu lernen, empfiehlt es sich immer wieder einmal vom Entwurfs-Modus aus in den SQL-Darstellungsmodus zu wechseln.

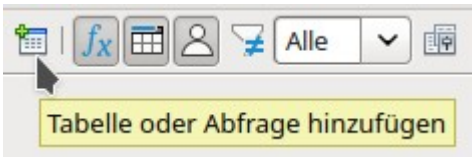


Hier wurde die durch die Auswahlen erstellte SQL-Formulierung sichtbar gemacht. Für die bessere Übersicht ist die Ansicht mit Zeilenumbrüchen versehen worden. Leider speichert der Editor diese Zeilenumbrüche standardmäßig nicht mit ab, so dass beim nächsten Aufruf die Abfrage wieder komplett als eine durchgängige Zeile mit Umbruch am Fensterrand wiedergegeben wird.

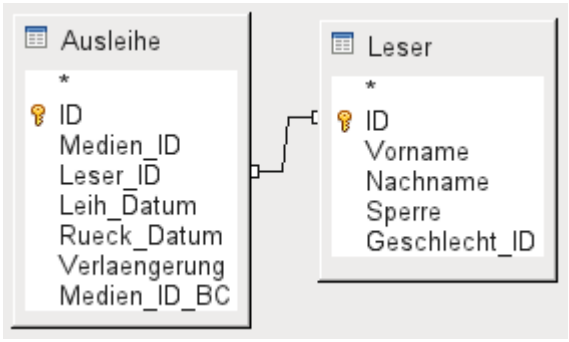
Über **SELECT** wird die Auswahl gestartet. Mit **AS** können die Aliasbezeichnungen eingeführt werden. **FROM** zeigt auf die Tabellenquelle der Abfrage. **WHERE** gibt die Bedingung für die Abfrage wieder, hier also, dass der Inhalt des Feldes "Rueck_Datum" leer ist (**IS NULL**). Mit **ORDER BY** wird die Sortierung definiert, und zwar als aufsteigend (**ASC** – ascending) für die beiden Felder "Leser_ID" und "Ausleihdatum". Diese Sortierung zeigt auch, dass die Zuweisung eines Alias das Feld "Leih_Datum" auch in der Abfrage selbst mit dem Alias ansprechbar macht.

Bisher sind die Felder "Medien_ID" und "Leser_ID" nur als Zahlenfelder sichtbar. Welchen Namen der Leser hat, bleibt unklar. Um dies in einer Abfrage anzuzeigen, muss die Tabelle Leser eingebunden werden. Um die folgende Ansicht zu erhalten, muss in den Design-Modus

zurückgeschaltet werden. Danach kann dann eine neue Tabelle in der Design-Ansicht hinzugefügt werden.



Hier können im Nachhinein weitere Tabellen oder Abfragen in der grafischen Benutzeroberfläche sichtbar gemacht werden. Sind bei der Erstellung der Tabellen Beziehungen geklärt worden (siehe Kapitel *Beziehungen zwischen Tabellen allgemein*), dann werden die Tabellen entsprechend direkt miteinander verbunden angezeigt:

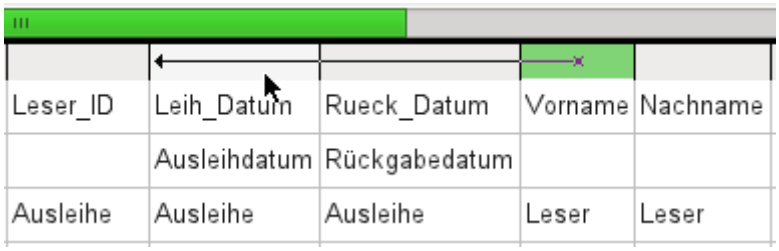


Fehlt die Verbindung, so kann hier durch ein Ziehen mit der Maus von "Ausleihe"."Leser_ID" zu "Leser"."ID" eine direkte Verknüpfung erstellt werden.

Hinweis

Eine Verbindung von Tabellen geht zur Zeit nur, wenn es sich um die interne Datenbank oder ein relationales Datenbanksystem handelt. Tabellen einer Tabellenkalkulation z. B. lassen sich so nicht miteinander verbinden. Sie müssen dazu in eine interne Datenbank importiert werden.
Um die Verbindung der Tabellen herzustellen, reicht ein Import ohne zusätzliche Erstellung eines Primärschlüssels aus.

Jetzt können im tabellarischen Bereich auch die Felder der Tabelle "Leser" ausgewählt werden. Die Felder werden dabei erst einmal am Schluss der Abfrage angeordnet.



Mit der Maus kann in dem tabellarischen Bereich des Editors die Lage der Felder korrigiert werden. Hier wird z. B. gerade das Feld "Vorname" direkt vor das Feld "Leih_Datum" verlegt.

Hinweis

In neueren Versionen von LibreOffice ist die Anzeige des Pfeils sowie des Ausgangsfeldes leider komplett verloren gegangen. Das Ziehen der Felder funktioniert im Editor trotzdem.

ID	Medien_ID	Leser_ID	Vorname	Nachname	Ausleihdatum	Rückgabedatum
12	3	0	Bert	Lederstrumpf	09.12.11	
16	7	0	Bert	Lederstrumpf	25.02.12	
23	1	0	Bert	Lederstrumpf	04.04.12	
22	2	1	Heinrich	Müller	04.04.12	
24	8	1	Heinrich	Müller	22.04.12	
21	0	9	Terence	Nobody	04.04.12	

Datensatz 1 von 6

Die Namen wurden jetzt sichtbar gemacht. Die "Leser_ID" ist eigentlich überflüssig. Auch ist die Sortierung nach "Nachname" und "Vorname" eigentlich sinnvoller als nach der "Leser_ID".

Diese Abfrage eignet sich nicht mehr für Base als Abfrage mit Eingabemöglichkeit, da zu der neu hinzugekommenen Tabelle "Leser" der Primärschlüssel fehlt. Erst wenn auch dieser Primärschlüssel eingebaut wird, ist die Abfrage wieder editierbar – allerdings komplett editierbar, so dass auch die Namen der Leser geändert werden können. Die Möglichkeit der Editierbarkeit ist also sehr vorsichtig zu nutzen, gegebenenfalls über ein Formular einzuschränken.

Selbst wenn die Abfrage weiter editierbar ist, lässt sie sich nicht so komfortabel nutzen wie ein Formular mit Listenfeldern, die zwar die Lesernamen anzeigen, aber die "Leser_ID" an die Tabelle weitergeben. Listenfelder lassen sich in eine Abfrage nicht einfügen. Sie sind den Formularen vorbehalten.

```
SELECT "Ausleihe"."ID", "Ausleihe"."Medien_ID", "Ausleihe"."Leser_ID",
"Leser"."Vorname", "Leser"."Nachname", "Ausleihe"."Leih_Datum" AS
"Ausleihdatum", "Ausleihe"."Rueck_Datum" AS "Rückgabedatum"
FROM "Ausleihe", "Leser"
WHERE "Ausleihe"."Leser_ID" = "Leser"."ID" AND
"Ausleihe"."Rueck_Datum" IS NULL
ORDER BY "Ausleihe"."Leser_ID" ASC, "Ausleihdatum" ASC
```

Wird jetzt auf die SQL-Ansicht umgeschaltet, so zeigt sich, dass alle Felder mit einer Doppelbezeichnung gekennzeichnet sind: **"Tabellenname"."Feldname"**. Dies ist notwendig, damit der Datenbank klar wird, aus welcher Tabelle die jeweiligen Feldinhalte stammen. Schließlich können Felder in unterschiedlichen Tabellen ohne weiteres den gleichen Feldnamen tragen. Bei den bisherigen Tabellenkonstruktionen trifft dies z. B. immer auf das Feld "ID" zu.

Hinweis

Folgende Abfrage funktioniert auch ohne Tabellenname vor den Feldnamen:

```
001  SELECT
002     "Ware"."ID",
003     "Abgang"."Anzahl",
004     "Ware"."Preis"
005  FROM "Ware",
006     "Abgang"
007  WHERE "Abgang"."Ware_ID" = "Ware"."ID"
```

Hierbei wird "ID" aus der Tabelle gezogen, die als erste in der **FROM**-Definition steht. Auch die Tabellendefinition in der **WHERE**-Formulierung wäre überflüssig, da "Ware_ID" nur einmal in der Tabelle "Abgang" vorkommt und die ID aus der Tabelle "Ware" genommen wird (Position der Tabelle). Die folgende Abfrage liefert also das gleiche Ergebnis:

```
001  SELECT
002     "ID",
003     "Anzahl",
004     "Preis"
005  FROM "Ware",
006     "Abgang"
007  WHERE "Ware_ID" = "ID"
```

Mit ~~Firebird~~ ist diese Abfrage nicht möglich, da sowohl die Tabelle "Ware" als auch die Tabelle "Abgang" ein Feld "ID" besitzen. Es erscheint stattdessen die Fehlermeldung «*Ambiguous field name between table Ware and table Abgang *ID»

Wird einem Feld in der Abfrage ein Alias zugewiesen, so kann es z. B. in der Sortierung mit diesem Alias ohne einen Tabellennamen angesprochen werden. Dies gilt nicht für ~~Firebird~~.

Hinweis

Der Alias in einer Abfrage kann bei der HSQLDB zur Beziehungsdefinition, zum Sortieren usw. genutzt werden. In FIREBIRD muss hingegen der ursprüngliche Ausdruck an der entsprechenden Stelle stehen.

Sortierungen werden in der grafischen Benutzeroberfläche nach der Reihenfolge der Felder in der Tabellenansicht vorgenommen. Sollte stattdessen zuerst nach "Ausleihdatum" und dann nach "Ausleihe"."Leser_ID" sortiert werden, so kann dies erzeugt werden, indem

- die Reihenfolge der Felder im tabellarischen Bereich der grafischen Benutzeroberfläche geändert wird,
- ein zweites Feld eingefügt wird, das auf unsichtbar geschaltet ist und nur die Sortierung gewährleisten soll (wird allerdings beim Editor nur vorübergehend angenommen, wenn kein Alias definiert wurde) oder
- der Text für die **ORDER BY** – Anweisung im SQL-Editor entsprechend umgestellt wird.

Die Sortierung aus der SQL-Ansicht wird in die GUI korrekt übernommen und entsprechend mit nicht sichtbaren Feldern in der grafischen Benutzeroberfläche angezeigt.

Funktionen in der Abfrage

Mittels Funktionen lässt sich aus Abfragen auch mehr ersehen als nur ein gefilterter Blick auf die Daten einer oder mehrerer Tabellen. In der folgenden Abfrage wird, abhängig von der "Leser_ID", gezählt, wie viele Medien ausgeliehen wurden.

Feld	ID	Leser_ID	Rueck_Datum
Alias	Anzahl		
Tabelle	Ausleihe	Ausleihe	Ausleihe
Sortierung			
Sichtbar	☑	☑	☐
Funktion	Anzahl	Gruppierung	
Kriterium			IST LEER

Für die "ID" der Tabelle "Ausleihe" wird die Funktion **`Anzahl`** ausgewählt. Prinzipiell ist hier egal, welches Feld einer Tabelle gewählt wurde. Die einzige Bedingung: *Das Feld darf nicht in irgendwelchen Datensätzen leer sein*. Aus diesem Grunde ist der Primärschlüssel, der ja nie leer ist, immer eine geeignete Wahl. Gezählt werden die Felder, die einen Inhalt enthalten, der von **`NULL`** verschieden ist.

Hinweis

Sobald für ein Feld eine andere Funktion als die Gruppierung gewählt wird, sollte das Feld einen **Aliasnamen** erhalten. Ansonsten kann es bei der Weiterverarbeitung z.B. in Berichten, verschachtelten Abfragen oder auch Ansichten zu Problemen kommen.

Für die "Leser_ID", die ja Rückschlüsse auf den Leser zulässt, wird als Funktion die **`Gruppierung`** gewählt. Dadurch werden die Datensätze nach der "Leser_ID" zusammengefasst. So zählt denn die Anweisung die Datensätze, die zu jeder "Leser_ID" passen.

Als Kriterium ist wie in den vorhergehenden Beispielen das "Rueck_Datum" auf **`IST LEER`** gesetzt.

	Anzahl	Leser_ID
	3	0
	1	9
▶	2	1

Datensatz 3 von 3

```
SELECT COUNT( "ID" ) AS "Anzahl", "Leser_ID"
FROM "Ausleihe"
WHERE "Rueck_Datum" IS NULL
GROUP BY "Leser_ID"
```

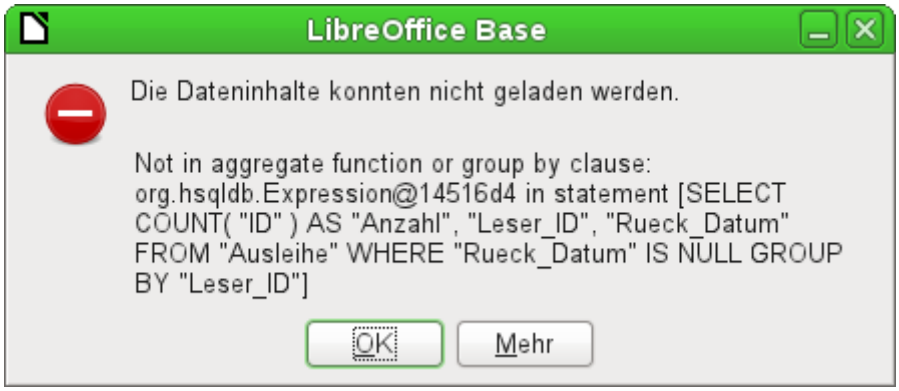
Die Abfrage zeigt im Ergebnis, dass z. B. "Leser_ID" '0' insgesamt noch 3 Medien entliehen hat. Wäre die Funktion **`Anzahl`** statt der "ID" dem "Rueck_Datum" zugewiesen worden, so würden für alle "Leser_ID" jeweils '0' entliehene Medien dargestellt, da ja "Rueck_Datum" als **`LEER`** vordefiniert ist.

Die entsprechende Formulierung für den SQL-Code ist oben wieder abgebildet. Aus dem Begriff **`Anzahl`** der deutschen GUI wird **`COUNT()`**. Aus dem Begriff **`Gruppierung`** wird der Zusatz **`GROUP BY`**.

Insgesamt stehen über die grafische Benutzeroberfläche folgende Funktionen zur Verfügung, die ihre Entsprechung zu Funktionen in der zugrundeliegenden HSQLDB haben:

Eine Erläuterung zu den Funktionen ist in dem folgenden Kapitel *Abfrageerweiterungen im SQL-Modus* nachzulesen. Die Funktionen «Sammeln», «Vereinigung» und «Durchschnitt» funktionieren mit keiner der eingebauten Datenbanken. Sie sind Standardfunktionen von Oracle.

Wird einem Feld in einer Abfrage eine Sammelfunktion hinzugefügt, so müssen alle anderen Felder der Abfrage auch mit Funktionen versehen sein, sofern die Felder sichtbar sein sollen. Dies liegt daran, dass in einem Datensatz nicht plötzlich zwischendurch Felder mehrere Datensätze abbilden können. Wird dies nicht beachtet, so erscheint die folgende Fehlermeldung:



Etwas frei übersetzt: Der folgenden Ausdruck enthält ein Feld ohne eine der Sammelfunktionen oder eine Gruppierung.

Danach wird die gesamte Abfrage aufgelistet, leider ohne das Feld konkret zu benennen. Hier wurde einfach das Feld "Rueck_Datum" als sichtbar hinzugefügt. Dieses Feld hat keine Funktion zugewiesen bekommen und ist auch nicht in der Gruppierung enthalten.

Die über den Button Mehr erreichbaren Informationen sind für den Normalnutzer einer Datenbank nicht aufhellender. Hier wird lediglich zusätzlich noch der SQL-Fehlercode aufgeführt.

Innerhalb der GUI können auch die Grundrechenarten sowie weitere Funktionen angewandt werden.

Feld	ID	Medien_ID	Leser_ID	Datum	Anzahl("Mahnung"."Datum") * 2	Rueck_Datum
Alias				Mahnzahl	Mahnbetrag	
Tabelle	Ausleihe	Ausleihe	Ausleihe	Mahnung		Ausleihe
Sortierung						
Sichtbar	☑	☑	☑	☑	☑	☐
Funktion	Gruppierung	Gruppierung	Gruppierung	Anzahl		
Kriterium						IST LEER

Hier wurden die Tabelle "Ausleihe" und die Tabelle "Mahnung" zusammen abgefragt. Aus der Zahl der Datumseinträge in der Tabelle "Mahnung" wird auf die Anzahl der Mahnungen geschlossen. Als Mahnbetrag wird in der Abfrage 2,- € festgelegt. Statt der Feldauswahl wird in das Feld einfach geschrieben: **`Anzahl(Mahnung.Datum)*2`** . Die grafische Benutzeroberfläche setzt anschließend die Anführungsstriche und wandelt den Begriff Anzahl in den entsprechenden SQL-Begriff um.

Vorsicht

Werden in der grafischen Benutzeroberfläche Zahlen mit Nachkommastellen eingegeben, so ist auf jeden Fall darauf zu achten, dass statt eines Kommas ein Punkt der Trenner für Dezimalzahlen in SQL ist. Kommata sind hingegen die Trenner der Felder. Deshalb werden einfach neue Abfragefelder gegründet, die die Nachkommastellen ausgeben.

	ID	Medien_ID	Leser_ID	Mahnzahl	Mahnbetrag
▶	12	3	0	2	4
	16	7	0	1	2
	23	1	0	1	2

Datensatz 1 von 3

```
SELECT "Ausleihe"."ID", "Ausleihe"."Medien_ID", "Ausleihe"."Leser_ID",
COUNT( "Mahnung"."Datum" ) AS "Mahnzahl",
COUNT( "Mahnung"."Datum" ) * 2 AS "Mahnbetrag"
FROM "Mahnung", "Ausleihe"
WHERE "Mahnung"."Ausleihe_ID" = "Ausleihe"."ID"
AND "Ausleihe"."Rueck_Datum" IS NULL
GROUP BY "Ausleihe"."ID", "Ausleihe"."Medien_ID", "Ausleihe"."Leser_ID"
```

Die Abfrage ermittelt jetzt für jedes noch entliehene Medium anhand der herausgegebenen Mahnungen und der zusätzlich eingefügten Multiplikation die Mahngebühren. Die folgende Abfragekonstruktion hilft weiter, wenn die Gebühren für jeden Leser berechnet werden sollen:

Feld	Leser_ID	Datum	Anzahl("Mahnung"."Datum") * 2	Rueck_Datum
Alias		Mahnzahl	Mahnbetrag	
Tabelle	Ausleihe	Mahnung		Ausleihe
Sortierung				
Sichtbar	☑	☑	☑	☐
Funktion	Gruppierung	Anzahl		
Kriterium				IST LEER

Die Felder "Ausleihe"."ID" und "Ausleihe"."Medien_ID" wurden entfernt. Sie erzeugten in der vorherigen Abfrage über die Gruppierung für jedes Medium einen separaten Datensatz. Jetzt wird nur noch nach den Lesern gruppiert. Das Abfrageergebnis sieht dann so aus:

	Leser_ID	Mahnzahl	Mahnbetrag
▶	0	4	8

Statt die Medien für "Leser_ID" '0' separat aufzulisten werden alle Felder aus "Mahnung"."Datum" zusammengezählt und die Summe von 8,- € als Mahngebühr ermittelt.

Beziehungsdefinition in der Abfrage

Werden Daten in Tabellen oder einem Formular gesucht, so beschränkt sich die Suche in der Regel auf eine Tabelle bzw. auf ein Formular. Selbst der Weg von einem Hauptformular zu einem Unterformular ist für die eingebauten Suchfunktionen nicht gangbar. Da bietet es sich dann an, zu durchsuchende Daten mit einer Abfrage zusammenzufassen.

	Titel
▶	Der kleine Hobbit
	Das sogennante Böse
	Eine kurze Geschichte der Zeit
	Traditionelle und kritische Theorie
	Die neue deutsche Rechtschreibung
	I hear you knocking
	Datenbanken mit OpenOffice.org 3
	Das Postfix-Buch
	Im Augenblick

Datensatz 1 von 9

Feld	Titel
Alias	
Tabelle	Medien
Sortierung	
Sichtbar	☑ ☐

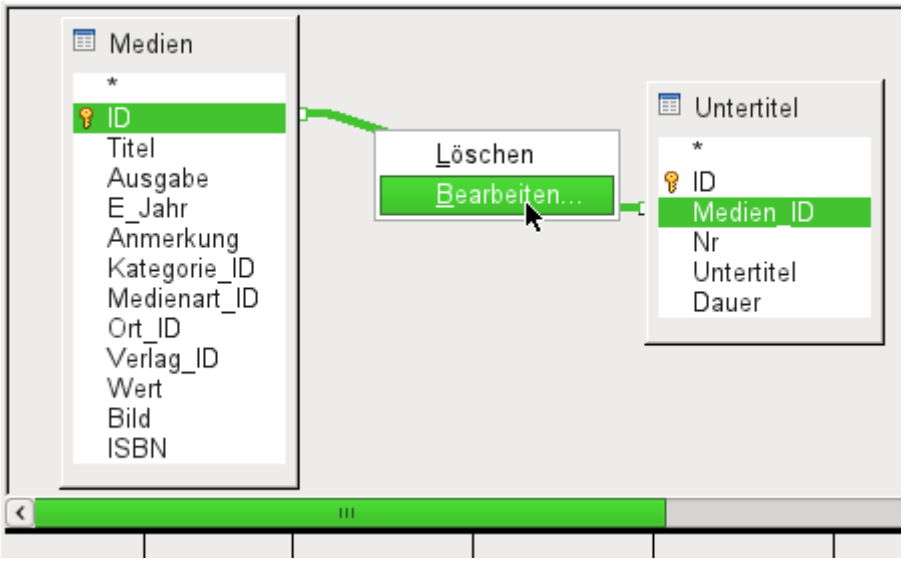
	Titel	Untertitel
▶	I hear you knocking	Youn can't catch me
	I hear you knocking	The stumble
	I hear you knocking	Sabre dance (Single version)
	Im Augenblick	Amsterdam
	Im Augenblick	Hier unten am Deich
	Im Augenblick	Köln-Ehrenfeld
	Im Augenblick	Bei Mir
	Im Augenblick	Gott sei Dank

Datensatz 1 von 8

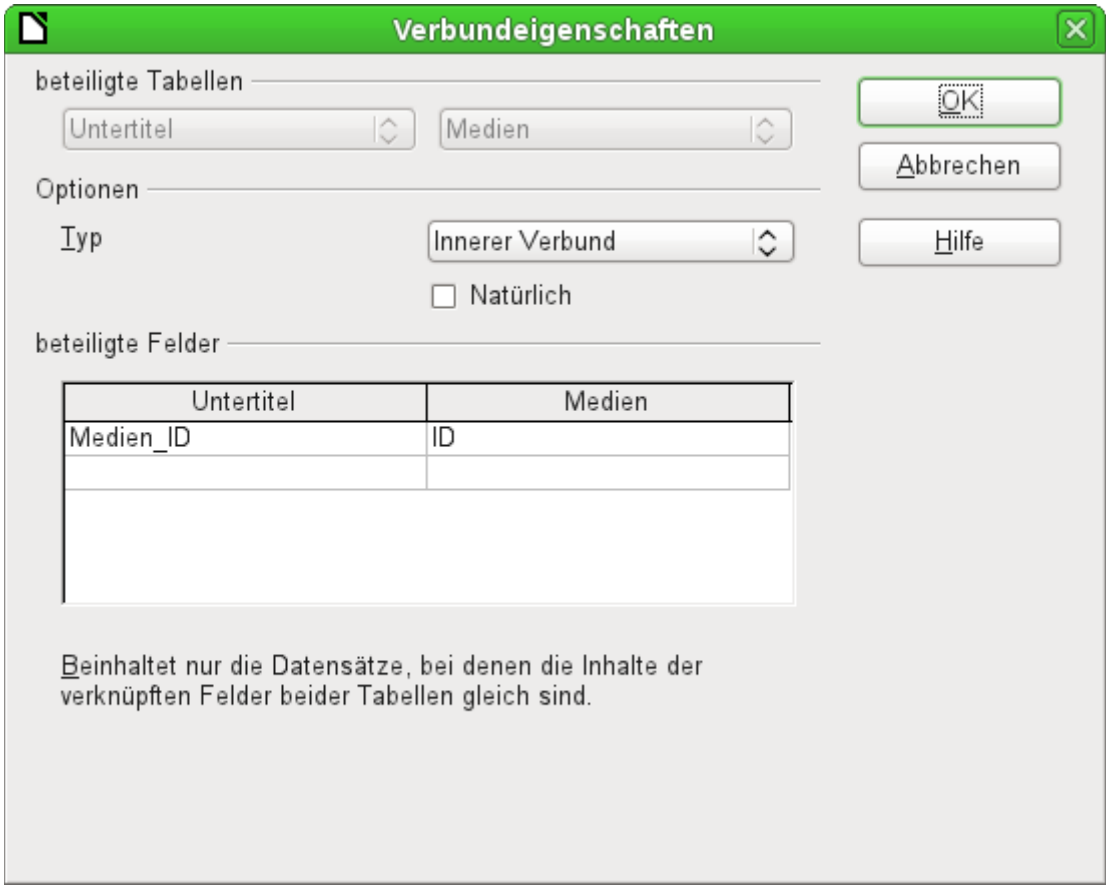
Feld	Titel	Untertitel
Alias		
Tabelle	Medien	Untertitel
Sortierung		
Sichtbar	☑	☑ ☐

Die einfache Abfrage an die "Titel" aus der Tabelle "Medien" zeigt den eingegebenen Testbestand dieser Tabelle mit 9 Datensätzen an. Wird jedoch die Tabelle "Untertitel" mit in die

Abfrage aufgenommen, so reduziert sich der Datenbestand aus der Tabelle "Medien" auf lediglich 2 "Titel". Nur für diese beiden "Titel" gibt es auch "Untertitel" in der Tabelle "Untertitel". Für alle anderen "Titel" existieren keine "Untertitel". Dies entspricht der Verknüpfungsbedingung, dass nur die Datensätze angezeigt werden sollen, bei denen in der Tabelle "Untertitel" das Feld "Medien_ID" gleich dem Feld "ID" aus der Tabelle "Medien" ist. Alle anderen Datensätze werden ausgeschlossen.



Die Verknüpfungsbedingung muss zum Bearbeiten geöffnet werden, damit alle gewünschten Datensätze angezeigt werden. Es handelt sich hier **nicht** um die Verknüpfung von Tabellen im *Relationenentwurf,* **sondern** um die Verknüpfung in einer *Abfrage*.



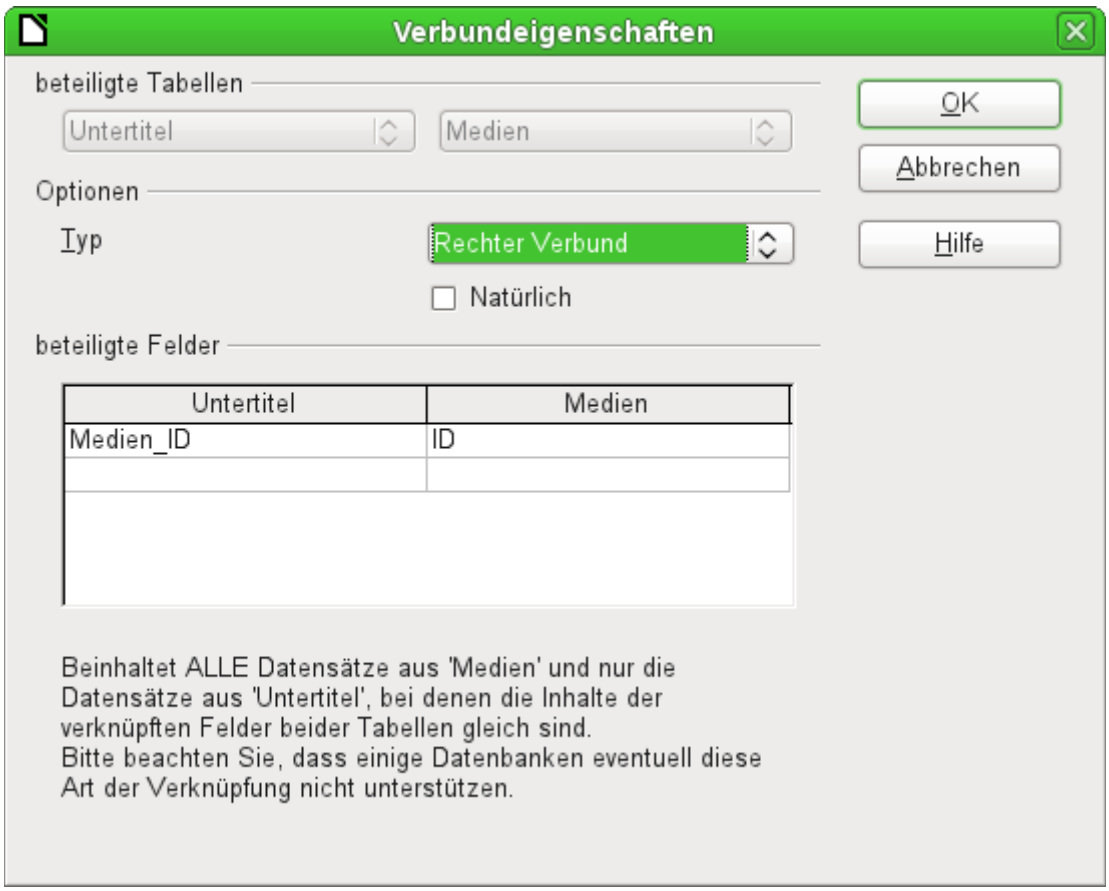
Standardmäßig steht die Verknüpfung als **Innerer Verbund** zur Verfügung. Das Fenster gibt darüber Aufschluss, wie diese Form der Verknüpfung sich auswirkt.

Als beteiligte Tabellen werden die beiden vorher ausgewählten Tabellen gelistet. Sie sind hier nicht wählbar. Die beteiligten Felder der beiden Tabellen werden aus der Tabellendefinition ausgelesen. Ist eine Beziehung in der Tabellendefinition nicht vorgegeben, so kann sie hier für die Abfrage erstellt werden. Eine saubere Datenbankplanung mit der HSQLDB sieht aber so aus, dass auch an diesen Feldern nichts zu verstellen ist.

Wichtigste Einstellung ist die Option des *Verbundes*. Hier können Verknüpfungen so gewählt werden, dass alle Datensätze von der Tabelle "Untertitel" und nur die Datensätze aus "Medien" gewählt werden, die in der Tabelle "Untertitel" auch "Untertitel" verzeichnet haben.

Umgekehrt kann gewählt werden, dass auf jeden Fall alle Datensätze aus der Tabelle "Medien" angezeigt werden – unabhängig davon, ob für sie auch "Untertitel existieren.

Die Option **Natürlich** setzt voraus, dass die zu verknüpfenden Felder in den Tabellen gleich lauten. Auch von dieser Einstellung ist Abstand zu nehmen, wenn bereits zu Beginn bei der Datenbankplanung die Beziehungen definiert wurden.



Für den **Typ → Rechter Verbund** zeigt die Beschreibung an, dass aus der Tabelle "Medien" (die in der Abbildung *rechts* angezeigte Tabelle) auf jeden Fall alle Datensätze angezeigt werden. Da es keine "Untertitel" gibt, die nicht in "Medien" mit einem "Titel" verzeichnet sind, sehr wohl aber "Titel" in "Medien", die nicht mit einem "Untertitel" versehen sind, ist dies also die richtige Wahl.

	Titel	Untertitel
▶	Der kleine Hobbit	
	Das sogennante Böse	
	Eine kurze Geschichte der Zeit	
	Traditionelle und kritische Theorie	
	Die neue deutsche Rechtschreibung	
	I hear you knocking	Youn can't catch me
	I hear you knocking	The stumble
	I hear you knocking	Sabre dance (Single version)
	Datenbanken mit OpenOffice.org 3	
	Das Postfix-Buch	
	Im Augenblick	Amsterdam
	Im Augenblick	Hier unten am Deich
	Im Augenblick	Köln-Ehrenfeld
	Im Augenblick	Bei Mir
	Im Augenblick	Gott sei Dank

Datensatz 1 von 15

Nach Bestätigung des *rechten Verbundes* sieht das Abfrageergebnis aus wie gewünscht. "Titel" und "Untertitel" werden komplett zusammen in einer Abfrage angezeigt. Natürlich kommen jetzt "Titel" wie in der vorhergehenden Verknüpfung mehrmals vor. Solange allerdings Suchtreffer nicht gezählt werden, könnte diese Abfrage im weiteren Verlauf als Grundlage für eine Suchfunktion dienen. Siehe hierzu die Codeschnipsel in diesem Kapitel, im Kapitel «Makros» (S. *526* ff.) und im Kapitel «DB-Aufgaben komplett» (S. *434* ff.).

Abfrageeigenschaften definieren

Mit der Version 4.1 von LibreOffice ist es möglich, in dem Abfrageeditor zusätzliche Abfrageeigenschaften zu definieren.

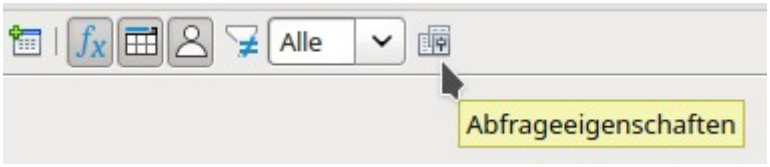
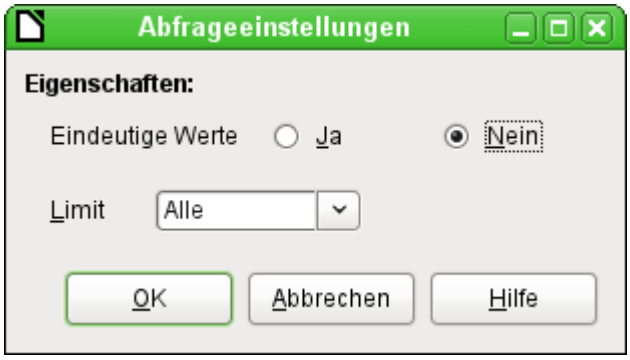


Abbildung 43: Aufruf der Abfrageeigenschaften im Abfrageeditor (ab LO 4.1)

Neben dem Button zum Aufruf der Abfrageeigenschaften befindet sich noch ein Kombinationsfeld, mit dem die Anzahl der anzuzeigenden Datensätze reguliert werden kann, sowie ein Button Eindeutige Werte. Diese Funktionen sind zusätzlich noch einmal in dem folgenden Dialog untergebracht:



alternativ:

Mit der Einstellung **Eindeutige Werte** wird beeinflusst, ob gleichlautende Datensätze in den Abfragen unterdrückt werden sollen.

Vorname	Nachname	Rueck_Datum
Lisa	Gerd	
Lisa	Gerd	
Lisa	Gerd	
Bert	Lederstrumpf	
Bert	Lederstrumpf	
Bert	Lederstrumpf	
Bert	Lederstrumpf	
Bert	Lederstrumpf	
Heinrich	Müller	
Heinrich	Müller	
Terence	Nobody	

In einer Abfrage soll ermittelt werden, welche Leser und Leserinnen noch Medien entliehen haben. Die Namen werden angezeigt, wenn das Rückgabedatum leer ist. Allerdings werden die Namen mehrmals angezeigt, wenn ein Leser oder eine Leserin noch mehrere Medien entliehen hat.

Vorname	Nachname	Rueck_Datum
Lisa	Gerd	
Bert	Lederstrumpf	
Heinrich	Müller	
Terence	Nobody	

Wird **Eindeutige Werte** ausgewählt, so verschwinden die Datensätze mit gleichem Inhalt.

Die Abfrage sind dann so aus:

```
SELECT DISTINCT
"Leser"."Vorname", "Leser"."Nachname", "Ausleihe"."Rueck_Datum"
FROM "Ausleihe", "Leser"
WHERE "Ausleihe"."Leser_ID" = "Leser"."ID" AND "Ausleihe"."Rueck_Datum" IS NULL
ORDER BY "Leser"."Nachname" ASC
```

Der ursprünglichen Abfrage

```
001 SELECT "Leser"."Vorname", "Leser"."Nachname" …
```

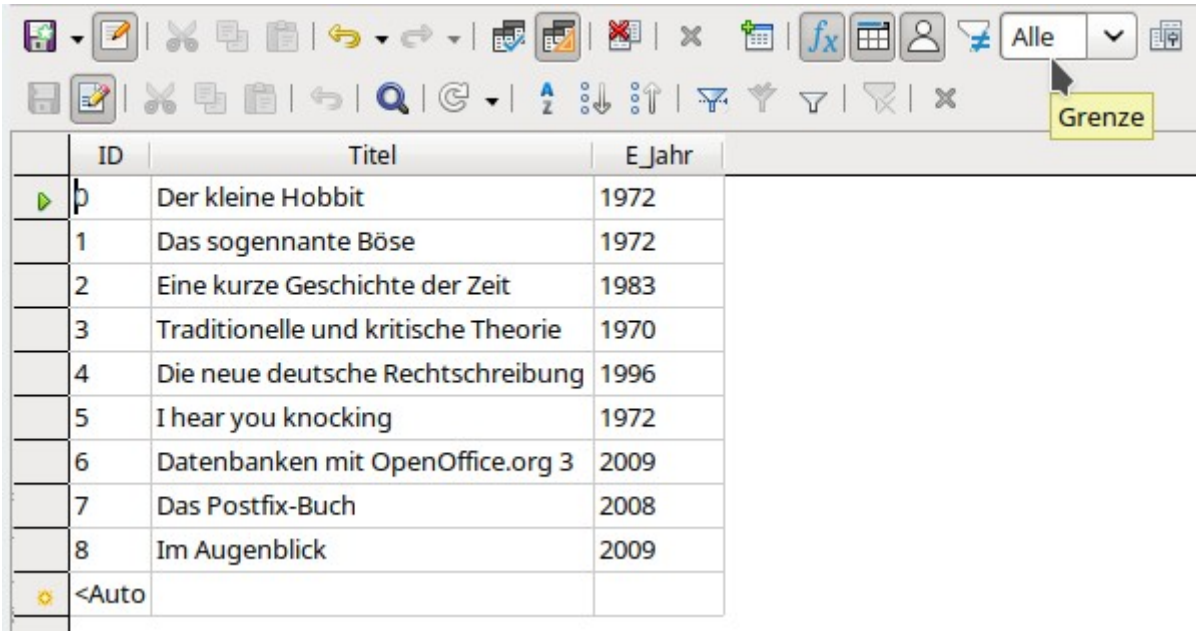
wird ein **DISTINCT** hinzugefügt:

```
001 SELECT DISTINCT "Leser"."Vorname", "Leser"."Nachname" …
```

Damit werden alle gleichlautenden Zeilen unterdrückt.

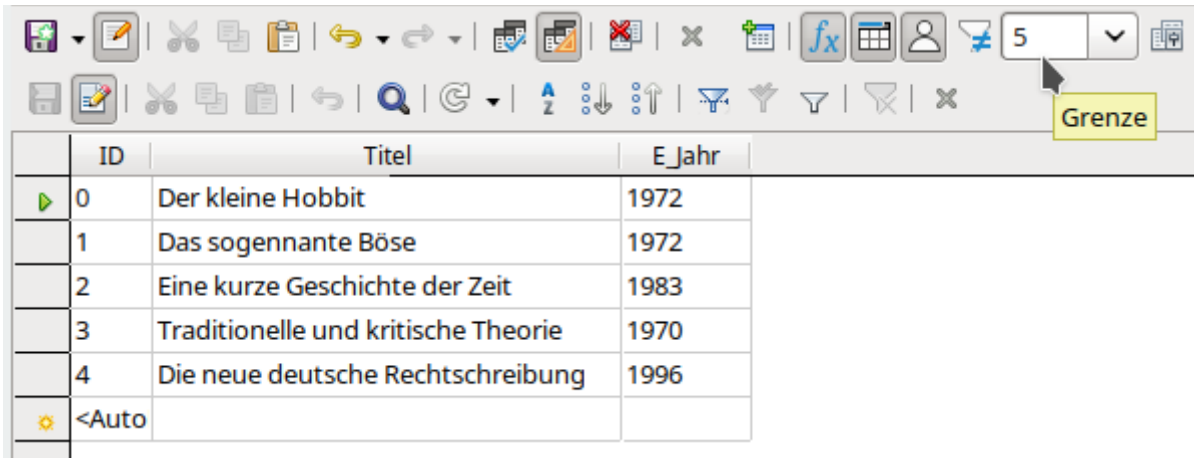
Die Auswahl eindeutiger Werte war auch in den Vorversionen möglich. Allerdings musste hier von der Design-Ansicht in die SQL-Ansicht umgeschaltet werden, um den Begriff **DISTINCT** einzufügen. Diese Eigenschaft ist also ohne Probleme abwärtskompatibel zu den Vorversionen von LO.

Mit der Einstellung **Grenze** (SQL: **Limit**) wird beeinflusst, wie viele Datensätze in der Abfrage angezeigt werden sollen. Es wird also nur eine begrenzte Zahl an Datensätzen wieder gegeben.



ID	Titel	E_Jahr
0	Der kleine Hobbit	1972
1	Das sogennante Böse	1972
2	Eine kurze Geschichte der Zeit	1983
3	Traditionelle und kritische Theorie	1970
4	Die neue deutsche Rechtschreibung	1996
5	I hear you knocking	1972
6	Datenbanken mit OpenOffice.org 3	2009
7	Das Postfix-Buch	2008
8	Im Augenblick	2009
<Auto		

Alle Datensätze der Tabelle "Medien" werden angezeigt. Die Abfrage ist editierbar, da auch der Primärschlüssel enthalten ist.



ID	Titel	E_Jahr
0	Der kleine Hobbit	1972
1	Das sogennante Böse	1972
2	Eine kurze Geschichte der Zeit	1983
3	Traditionelle und kritische Theorie	1970
4	Die neue deutsche Rechtschreibung	1996
<Auto		

Nur die ersten fünf Datensätze werden angezeigt (ID 0 bis 4). Eine Sortierung wurde nicht vorgewählt. Die Standardsortierung ist hier die nach dem Primärschlüssel, sofern nichts anderes festgelegt wurde. Die Abfrage ist trotz der Begrenzung weiterhin editierbar. Dies unterscheidet die Eingabe im grafischen Modus von der, die in früheren Versionen nur mit dem direkten SQL-Modus erreichbar ist.

```
SELECT "ID", "Titel", "E_Jahr" FROM "Medien" LIMIT 5
```

Der ursprünglichen Abfrage wurde lediglich «LIMIT 5» hinzugefügt. Die entsprechende Größe des Limits kann beliebig festgelegt werden.

Vorsicht

Die Einstellung des Limits durch die grafische Benutzeroberfläche ist nicht abwärtskompatibel. In allen LO-Versionen vor der Version 4.1 konnte ein Limit nur im direkten SQL-Modus eingegeben werden. Dort erforderte das Limit eine Sortierung (**ORDER BY …**) oder eine Bedingung (**WHERE …**).
Ohne eine entsprechend eingestellte Sortierung ist es auch nicht möglich, aus einem mit der GUI erstellten Limit eine Ansicht zu erstellen.

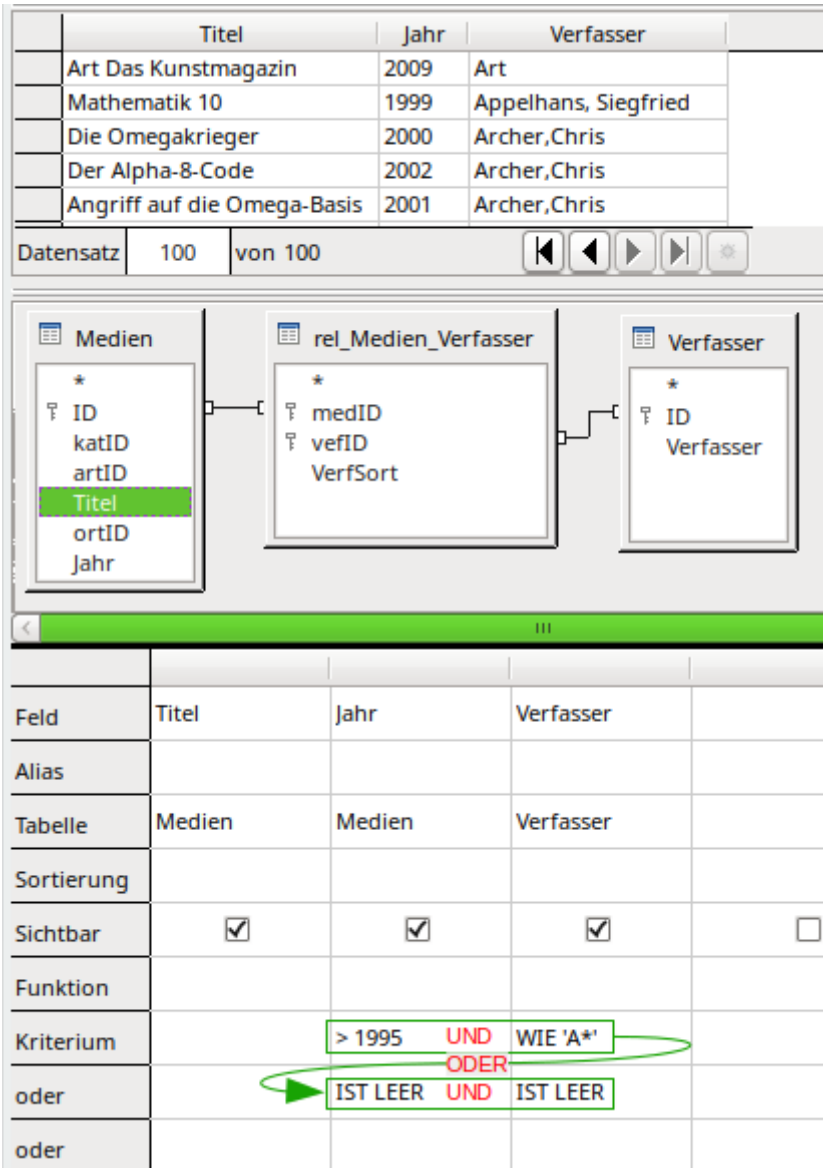
Abfragen nach Filterkriterien durchsuchen

Die Abfrage-GUI bietet eine einfache Möglichkeit, bestimmte Kriterien für die Datensuche festzulegen. Diese Kriterien werden unterhalb der Funktionen zusammengestellt.

Kriterien, die in einer Zeile nebeneinander stehen, werden mit **UND** verbunden. Es gelten die Eingaben der Zeile alle zusammen. Alle Eingaben in einer Zeile müssen also erfüllt sein, damit die Daten angezeigt werden. Diese Einträge in einer Zeile werden immer zuerst gelesen.

Kriterien, die untereinander stehen werden mit **ODER** verbunden. Es gilt entweder die Angabe in der einen Zeile oder die Angabe in der anderen Zeile.

Diese Einstellungen über die GUI sind in der Praxis manchmal nicht so leicht zu durchschauen. Deswegen hier ein Beispiel anhand von zwei Datenfeldern, das deutlich machen soll, wie die GUI-Konstruktion der Kriterien funktioniert:



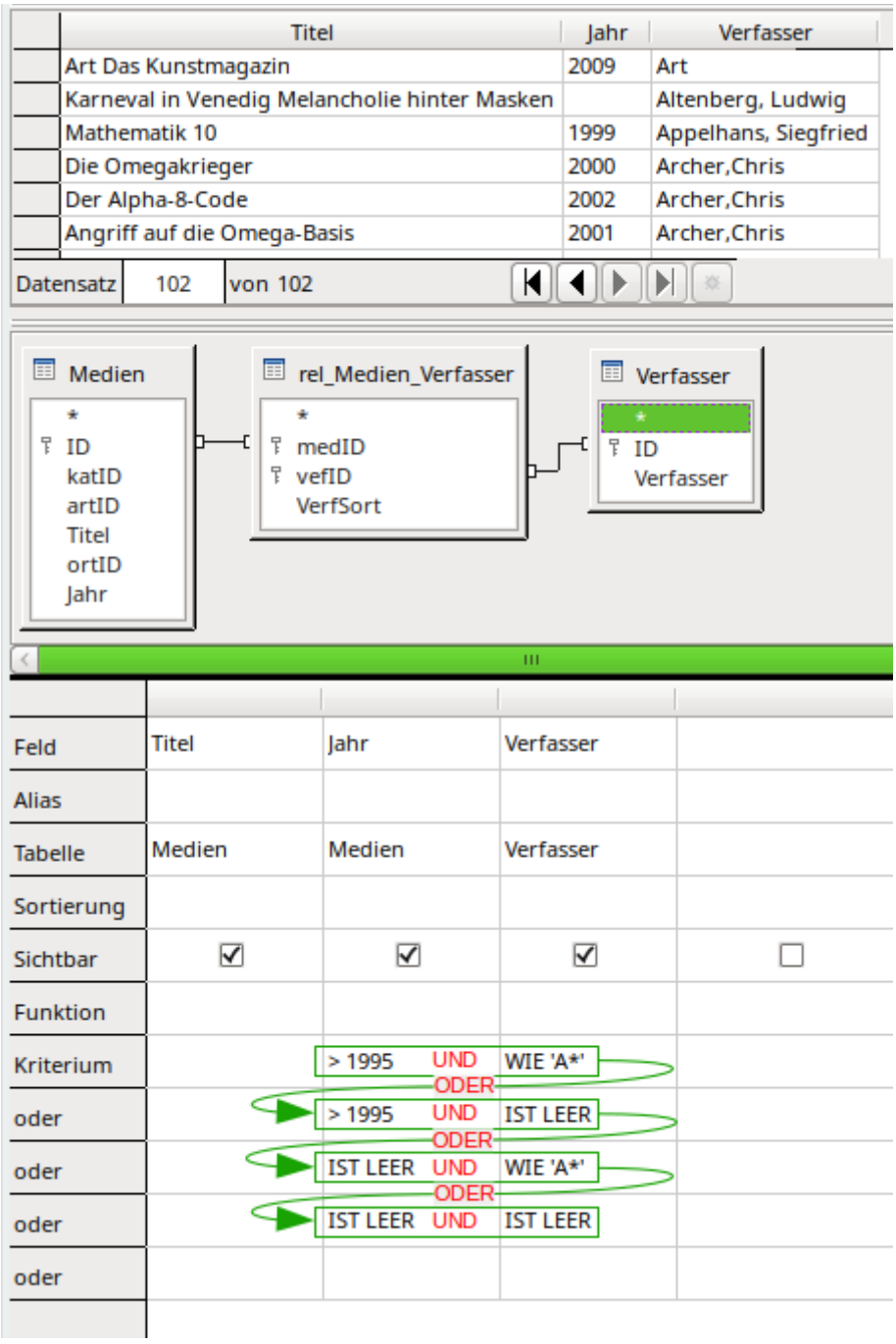
Aus den verbundenen Tabellen sollen die Datensätze angezeigt werden, bei denen das Jahr > 1995 ist oder der Eintrag für das Jahr leer ist. Außerdem sollen nur Verfasser mit 'A' beginnend angezeigt werden oder Datensätze, bei denen der Eintrag im Feld "Verfasser" leer ist.

Das in der obigen Einstellung der GUI entwickelte Schema für die Kriterien erfüllt diese Bedingungen leider *nicht*:
Das erste Kriterium ist für die Abfrage-GUI, dass das Jahr > 1995 sein muss **UND** "Verfasser" mit

'A' beginnen soll.
Das zweite Kriterium sagt aus, dass das Feld "Jahr" **UND** das Feld "Verfasser" leer sein sollen.

Damit gilt also: Entweder enthalten beide die geforderten Daten **ODER** beide sind leer, aber nicht, dass auch nur ein Feld die erforderlichen Inhalte hat und das andere leer ist. Die Kriterien werden zuerst in der Zeile und dann in der Spalte ausgelesen.

In der obigen Abfrage ist so kein Datensatz enthalten, bei dem z.B. der Eintrag für das Jahr fehlt. Die Anzahl der Datensätze ergibt hier 100.

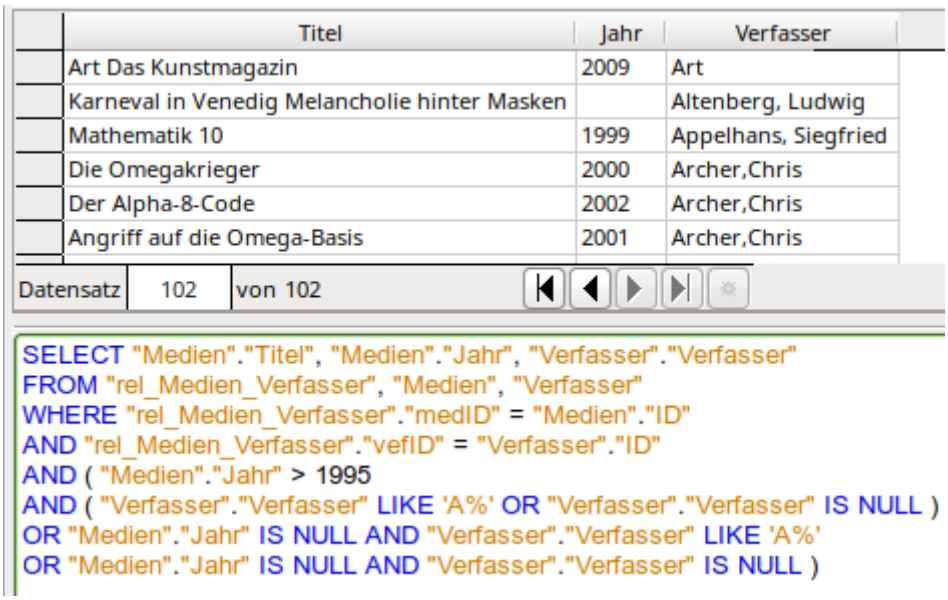


Hier ist die Anzahl der Datensätze auf 102 angestiegen. Die Abfrage hat jetzt auch Felder entdeckt, bei denen das Jahr leer geblieben ist, aber ein Eintrag im Feld "Verfasser" vorhanden war. Hier im Beispiel das Buch «Karneval ...», das im vorherigen Screenshot nicht auftaucht.

Alle notwendigen Kombinationen stehen in der GUI:

- "Jahr" **UND** "Verfasser" haben beide einen gewünschten Inhalt, sind beide nicht leer **ODER**

- "Jahr" hat einen Inhalt **UND** "Verfasser" ist leer **ODER**
- "Jahr" ist leer **UND** "Verfasser" hat einen gewünschten Inhalt **ODER**
- "Jahr" **UND** "Verfasser" sind beide leer.

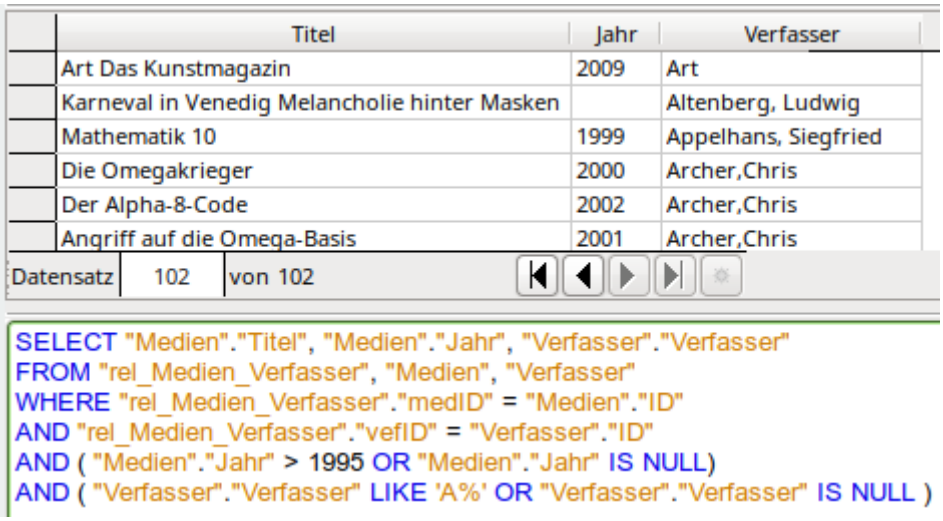


Titel	Jahr	Verfasser
Art Das Kunstmagazin	2009	Art
Karneval in Venedig Melancholie hinter Masken		Altenberg, Ludwig
Mathematik 10	1999	Appelhans, Siegfried
Die Omegakrieger	2000	Archer,Chris
Der Alpha-8-Code	2002	Archer,Chris
Angriff auf die Omega-Basis	2001	Archer,Chris

Datensatz 102 von 102

```
SELECT "Medien"."Titel", "Medien"."Jahr", "Verfasser"."Verfasser"
FROM "rel_Medien_Verfasser", "Medien", "Verfasser"
WHERE "rel_Medien_Verfasser"."medID" = "Medien"."ID"
AND "rel_Medien_Verfasser"."vefID" = "Verfasser"."ID"
AND ( "Medien"."Jahr" > 1995
AND ( "Verfasser"."Verfasser" LIKE 'A%' OR "Verfasser"."Verfasser" IS NULL )
OR "Medien"."Jahr" IS NULL AND "Verfasser"."Verfasser" LIKE 'A%'
OR "Medien"."Jahr" IS NULL AND "Verfasser"."Verfasser" IS NULL )
```

Der durch die GUI erzeugte Code sieht leider reichlich unübersichtlich aus. Der Eintrag von "Jahr" > 1995 wird über **AND** sowohl mit dem Anfangsbuchstaben des Verfassers als auch mit dem leeren Feld von "Verfasser" über **OR** verbunden. Damit werden die beiden ersten Zeilen des Kriteriums erledigt. Die anschließenden Zeilen werden dann nacheinander mit **OR** abgearbeitet.



Titel	Jahr	Verfasser
Art Das Kunstmagazin	2009	Art
Karneval in Venedig Melancholie hinter Masken		Altenberg, Ludwig
Mathematik 10	1999	Appelhans, Siegfried
Die Omegakrieger	2000	Archer,Chris
Der Alpha-8-Code	2002	Archer,Chris
Angriff auf die Omega-Basis	2001	Archer,Chris

Datensatz 102 von 102

```
SELECT "Medien"."Titel", "Medien"."Jahr", "Verfasser"."Verfasser"
FROM "rel_Medien_Verfasser", "Medien", "Verfasser"
WHERE "rel_Medien_Verfasser"."medID" = "Medien"."ID"
AND "rel_Medien_Verfasser"."vefID" = "Verfasser"."ID"
AND ( "Medien"."Jahr" > 1995 OR "Medien"."Jahr" IS NULL)
AND ( "Verfasser"."Verfasser" LIKE 'A%' OR "Verfasser"."Verfasser" IS NULL )
```

Schöner wäre es, wenn die GUI einen Code wie den obigen erzeugen könnte. Die Abfrage-GUI kann diesen Code so aber nicht erstellen, da zuerst die Zeile mit **UND** verbunden werden und anschließend die nächsten Zeilen mit **OR** angehängt werden. Hier fehlt der GUI schlicht die Möglichkeit eine Anweisung zu geben, dass zuerst die Spalten und dann die Zeilen abgehandelt werden sollen. Stattdessen werden bereits bei zwei Feldern insgesamt 4 von 5 in der GUI zur Verfügung stehenden Spalten beschrieben. Solche einfacheren Formulierungen sollten daher später zum Bearbeiten nur noch über das Kontextmenü mit **In SQL-Ansicht bearbeiten...** geöffnet werden. Sonst wird der Code auf den GUI-Code geändert.

Abfragen nachträglich ändern

Soll eine einmal erstelle Abfrage weiter bearbeitet werden, so wird mit einem rechten Mausklick über der Abfrage das **Kontextmenü** geöffnet und **Bearbeiten…** ausgewählt. Hier kann es manchmal dazu kommen, dass die GUI die Abfrage nicht richtig deuten kann. Manchmal passiert das schon beim Öffnen, so dass die Abfrage im SQL-Modus geöffnet wird. Es kann auch sein, dass die GUI die Abfrage öffnet, aber beim Umschalten in den SQL-Modus einen «Fehler in der SQL Syntax» findet, der gar nicht da ist. Schließlich hat die Abfrage vorher funktioniert.

Hier hilft es dann, direkt aus dem **Kontextmenü** über der Abfrage **In SQL-Ansicht bearbeiten…** zu wählen.

Abfrageerweiterungen im SQL-Modus

Wird von der grafischen Eingabe über **Ansicht → Design-Ansicht an-, ausschalten** die Design-Ansicht ausgeschaltet, so erscheint der SQL-Befehl, der bisher in der Design-Ansicht erstellt wurde. Für Neueinsteiger ist dies der beste Weg, die Standardabfragesprache für Datenbanken kennen zu lernen. Manchmal ist es auch der einzige Weg, eine Abfrage an die Datenbank abzusetzen, da die GUI die Abfrage nicht in den für die Datenbank notwendigen SQL-Befehl umwandeln kann.

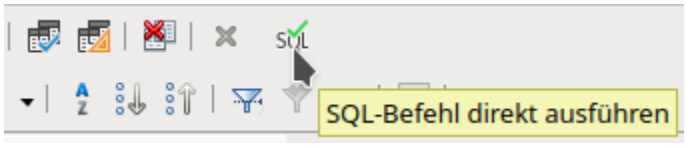
```
001 SELECT
002    *
003 FROM "Tabellenname"
```

Dies zeigt wirklich alles an, was in der Tabelle "Tabellenname" steht. Das «*» berücksichtigt sämtliche Felder der Tabelle.

```
001 SELECT
002    *
003 FROM "Tabellenname"
004 WHERE "Feldname" = 'Karl'
```

Eine deutliche Einschränkung wurde gemacht. Jetzt werden nur noch die Datensätze angezeigt, die in dem Feld "Feldname" den Begriff 'Karl' stehen haben – aber wirklich nur den Begriff, nicht z. B. 'Karl Egon'.

Manchmal sind Abfragen in Base nicht über die GUI ausführbar, da bestimmte Kommandos nicht bekannt sind. Hier hilft es dann die Design-Ansicht zu verlassen und über **Bearbeiten → SQL-Befehl direkt ausführen** den direkten Weg zur Datenbank zu wählen. Diese Methode hat allerdings den Nachteil, dass in dem angezeigten Abfrageergebnis keine Eingaben mehr möglich sind. Siehe hierzu *Eingabemöglichkeit in Abfragen*.



Die direkte Ausführung ist auch über die grafische Benutzeroberfläche erreichbar. Wie in der Abbildung zu sehen ist muss aber auch hier die Entwurfsansicht ausgeschaltet sein. Entsprechende Abfrageanweisungen sind teilweise mit `SQL` gekennzeichnet.

Hinweis

Wird eine SQL-Anweisung als direkte Anweisung geschrieben und ist SQL-Befehl direkt ausführen gewählt, so bleiben sämtliche Formatierungen des SQL-Kommandos erhalten. Hier können dann auch Kommentare in den SQL-Code eingefügt werden. Zeichen hierfür sind «**--** Kommentar...» für eine Kommentarzeile und «**/*** Kommentar ...***/**» für mehrere Kommentarzeilen direkt hintereinander.

Hier jetzt also die recht umfangreichen Möglichkeiten, an die Datenbank Fragen zu stellen und auf ein entsprechendes Ergebnis zu hoffen:

```
001 SELECT [{LIMIT <offset> <limit> | TOP <limit>}][ALL | DISTINCT]
002 { <Select-Formulierung> | "Tabellenname".* | * } [, ...]
003 [INTO [CACHED | TEMP  | TEXT] "neueTabelle"]
004 FROM "Tabellenliste"
005 [WHERE SQL-Expression]
006 [GROUP BY SQL-Expression [, ...]]
007 [HAVING SQL-Expression]
008 [{ UNION [ALL | DISTINCT] | {MINUS [DISTINCT] | EXCEPT [DISTINCT] } |
009 INTERSECT [DISTINCT] } Abfrageaussage]
010 [ORDER BY Ordnungs-Expression [, ...]]
011 [LIMIT <limit> [OFFSET <offset>]];
```

[{LIMIT <offset> <limit> | TOP <limit>}]: (HSQLDB**)**
[{FIRST <limit> | SKIP <limit>}]: (FIREBIRD**)**

Hiermit wird die Menge der anzuzeigenden Datensätze begrenzt.
HSQLDB:Mit **LIMIT 10 20** werden ab dem 11. Datensatz die folgenden 20 Datensätze angezeigt. Mit **TOP 10 (**~~HSQLDB~~**)** werden immer die ersten 10 angezeigt. Dies ist gleichbedeutend mit **LIMIT 0 10**. **LIMIT 10 0** lässt die ersten 10 Datensätze aus und zeigt alle Datensätze ab dem 11. Datensatz an.
FIREBIRD: Mit **FIRST 10** werden die ersten 10 Datensätze angezeigt. Mit **FIRST 10 SKIP 5** die Datensätze von 6 bis 15.

Den gleichen Sinn erfüllt die zum Schluss der SELECT-Bedingung erscheinende Formulierung **[LIMIT <limit> [OFFSET <offset>]] (**HSQLDB**,** FIREBIRD**)**. **LIMIT 10** zeigt lediglich 10 Datensätze an. Wird **OFFSET 20** hinzugefügt, so beginnt die Anzeige ab dem 21. Datensatz. Für die zum Schluss stehende Begrenzung ist eine Anweisung zur Sortierung (ORDER BY ...) oder eine Bedingung (WHERE ...) Voraussetzung.
FIREBIRD: Firebird kennt **LIMIT** eigentlich nicht. Es ist nur für die GUI implementiert worden. Hier gibt es **[ROWS <limit> TO <rownr>]**. Mit **ROWS 10** werden die ersten 10 Datensätze angezeigt. Mit **ROWS 10 TO 20** werden die Datensätze 10 bis 20 angezeigt. **rownr** muss also immer größer oder mindestens gleich **limit** sein.

Sämtliche Begrenzungen des anzuzeigenden Abfrageergebnisses sind bis einschließlich der Version LO 4.0 nur über die direkte Ausführung des SQL-Kommandos verfügbar. Erst ab LO 4.1 ist eine Limitierung ohne Sortierung oder Bedingung in der grafischen Benutzeroberfläche möglich. Dies wird sogar dann noch aufrecht erhalten, wenn in den direkten SQL-Modus umgeschaltet wurde. Die Limitierung kann in LO 4.1 in der SQL-Ansicht um den «OFFSET» ergänzt werden. Die folgende Abfrage ist also ab LO 4.1 editierbar:

```
001 SELECT * FROM "Tabelle" LIMIT 20 OFFSET 10
```
(HSQLDB, FIREBIRD)

Alle Eingaben in der Limitierung können nur direkt als Ganzzahl erfolgen. Es ist nicht möglich, die Eingaben durch eine Unterabfrage zu ersetzen, so dass z. B. fortlaufend die 5 letzten Datensätze einer Datenreihe angezeigt werden können.

[ALL | DISTINCT]

SELECT ALL ist die Standardeinstellung. Es werden alle Ergebnisse angezeigt, auf die die Bedingungen zutreffen. Beispiel:
SELECT ALL "Name" FROM "Tabellenname" gibt alle Namen an; kommt «Peter» dreifach und «Egon» vierfach in der Tabelle vor, so werden eben drei und vier Datensätze angezeigt.

SELECT DISTINCT "Name" FROM "Tabellenname" sorgt hingegen dafür, dass alle Abfrageergebnisse mit gleichem Inhalt unterdrückt werden. Hier würden also «Peter» und «Egon» nur einmal erscheinen. **DISTINCT** bezieht sich dabei auf den ganzen Datensatz, der in der Abfrage erfasst wird. Wird z. B. auch der Nachname erfasst, so unterscheiden sich die Datensätze mit «Peter Müller» und «Peter Maier». Sie werden also auch bei der Bedingung **DISTINCT** auf jeden Fall angezeigt.

<Select-Formulierung>

```
001 { Expression | COUNT(*) |
002 { COUNT | MIN | MAX | SUM | AVG | SOME | EVERY | VAR_POP | VAR_SAMP |
    STDDEV_POP | STDDEV_SAMP }
003 ([ALL | DISTINCT]] Expression) } [[AS] "anzuzeigende Bezeichnung"]
```

Feldnamen, Berechnungen, Zählen der gesamten Datensätze – alles mögliche Eingaben, die hier erfolgen können.

Außerdem stehen in der Felddarstellung auch verschiedene Funktionen zur Verfügung. Mit Ausnahme von **COUNT(*)** (zählt alle Datensätze) berücksichtigen die verschiedenen Funktionen keine Felder, die **NULL** sind.

COUNT | MIN | MAX | SUM | AVG | SOME | EVERY | VAR_POP | VAR_SAMP | STDDEV_POP | STDDEV_SAMP

COUNT("Name") zählt alle Felder, die einen Namen enthalten.
FIREBIRD zusätzlich: **COUNT(DISTINCT "Name")** zählt alle unterschiedlichen Namen.
MIN("Name") zeigt den ersten Namen im Alphabet. Das Ergebnis dieser Funktion ist so formatiert, wie es dem Feldinhalt entspricht. Text wird im Ergebnis Text, Ganzzahl zu Ganzzahl, Dezimalzahl zu Dezimalzahl usw.
MAX("Name") zeigt entsprechend den letzten Namen im Alphabet.
SUM("Zahl") kann nur Werte aus Zahlenfeldern addieren. Die Funktion versagt bei Datumsfeldern.

Hinweis

Berechnungen in einer Datenbank können manchmal zu erstaunlichen Ergebnissen führen. Angenommen in der Datenbank würden Zensuren einer Klassenarbeit verwaltet und mit der Berechnung sollte die Durchschnittszensur ermittelt werden.
Zuerst werden die Zensurenwerte addiert. Die Summe ergibt z. B. 80. Jetzt wird durch die Zahl der Klassenarbeiten (30) dividiert. Die Abfrage ergibt 2.
Dies liegt daran, dass es sich bei der Berechnung um eine Rechnung mit Feldern des Typs INTEGER handelt. Auch die Berechnung gibt dann nur Ergebnisse des Zahlentyps INTEGER aus. In der Berechnung muss mindestens ein Feld enthalten sein, das vom Zahlentyp DEZIMAL ist. Dies kann entweder durch Umwandlung mittels einer Funktion der HSQLDB erfolgen oder, einfacher, indem z. B. durch 30.0 dividiert wird. Dann erscheint eine Nachkommastelle, bei 30.00 zwei Nachkommastellen usw. Zu beachten ist hier, dass bei Berechnungen Dezimalzahlen mit dem in der englischen Schreibweise üblichen Dezimalpunkt dargestellt werden. Ein Komma ist bei Abfragen für die Trennung von Feldern reserviert.
Das Ergebnis wird bei der Anzeige von Nachkommastellen in der internen HSQLDB gerundet, bei Firebird aber nicht.

Tipp

HSQLDB: Auch bei Zeitfeldern versagt die Funktion. Hier kann folgendes hilfreich sein:

```
001 SELECT
002 (SUM( HOUR("Zeit") )*3600 + SUM( MINUTE("Zeit") ))*60 +
    SUM( SECOND("Zeit") ) AS "Sekunden"
003 FROM "Tabelle"
```

Die Summe wird von den Stunden, Minuten und gegebenenfalls Sekunden gebildet. Anschließend wird so erweitert, dass alles zusammen in einer Maßeinheit, hier in Sekunden, wiedergegeben wird. Es werden also weiter nur Zahlen mit der Summenformel addiert. Mit

```
001 SELECT
002 ((SUM( HOUR("Zeit") )*3600 + SUM( MINUTE("Zeit") ))*60 + SUM( SECOND("Zeit")
    )) / 3600.0000 AS "Stunden"
003 FROM "Tabelle"
```

wird aus der Addition wieder eine Zeit in Stunden mit den Minuten und Sekunden als Nachkommastellen. Daraus kann dann über entsprechende Formatierung auch wieder eine Zeitdarstellung in einer Abfrage oder einem Formular erreicht werden.
FIREBIRD: Stunde, Minute und Sekunde müssen über die Funktion **EXTRACT()** ermittelt werden. Allerdings kennt Firebird die Funktion **DATEADD**, so dass sich hier eine Möglichkeit ergibt, direkt über die Summe von Zeiten wieder eine Zeit zu erhalten:

```
001 SELECT
002 DATEADD( SUM( DATEDIFF( SECOND FROM TIME '00:00' TO "Zeit")) SECOND TO TIME
    '00:00') AS "NeueZeit"
003 FROM "Tabelle"
```

Die Zeitdifferenz zur Zeit 00:00 (Mitternacht) wird mittels **DATEDIFF** in Sekunden berechnet. Die Sekunden werden mit **SUM** addiert. Zu der Zeit 00:00 wird jetzt mittels **DATEADD** die Anzahl der Sekunden addiert und wieder als Zeit formatiert.

AVG("Zahl") zeigt den Mittelwert der Inhalte einer Spalte. Auch diese Funktion beschränkt sich auf Zahlenfelder. Hier sollte darauf geachtet werden, dass bei Ganzzahlen auch nur ein Ergebnis in Ganzzahlen ermittelt wird. **AVG("Ganzzahl" * 1.00)** zeigt dann in der HSQLDB einen Mittelwert mit maximal 2 Nachkommastellen.
FIREBIRD zusätzlich: **AVG(DISTINCT "Zahl")** erstellt den Durchschnitt aller unterschiedlichen Zahlen.
SOME("Ja_Nein"), EVERY("Ja_Nein"): SOME zeigt bei Ja/Nein Feldern (boolschen Feldern) die Version an, die nur einige Felder erfüllen. Da ein boolsches Feld die Werte 0 und 1 in der HSQLDB wiedergibt, erfüllen nur einige (**SOME**) die Bedingung 1, aber jeder (**EVERY**) mindestens die Bedingung 0. Über eine gesamte Tabelle abgefragt wird bei SOME also immer 'Ja' erscheinen, wenn mindestens 1 Datensatz mit 'Ja' angekreuzt ist. **EVERY** wird so lange 'Nein' ergeben, bis alle Datensätze mit 'Ja' angekreuzt sind. Beispiel:

```
001 SELECT
002    "Klasse",
003    EVERY("Schwimmer")
004 FROM "Tabelle1"
005 GROUP BY "Klasse";
```

Die Klassen werden alle angezeigt. Erscheint irgendwo kein Kreuz für 'Ja', so muss auf jeden Fall eine Betreuung für das Nichtschwimmerbecken im Schwimmunterricht dabei sein, denn es gibt mindestens eine Person in der Klasse, die nicht schwimmen kann. (HSQLDB, ~~FIREBIRD~~)

Hinweis

Ja/Nein-Felder können in der HSQLDB sowohl mit **TRUE** und **FALSE** als auch mit **1** und **0** abgefragt werden. FIREBIRD hingegen verlangt zwingend nach der Angabe **TRUE** bzw. **FALSE**.

VAR_POP | VAR_SAMP | STDDEV_POP | STDDEV_SAMP sind statistische Funktionen und greifen nur bei Ganzzahl- und Dezimalzahlfeldern.

Alle Funktionen ergeben 0, wenn die Werte einer Gruppe alle gleich sind.
Die statistischen Funktionen erlauben nicht die Einschränkung von **DISTINCT**. Sie rechnen also grundsätzlich über alle Werte, die die Abfrage beinhaltet. **DISTINCT** hingegen würde Datensätze mit gleichen Werten von der Anzeige ausschließen.
VAR_POP: (SUM(expr2) – SUM(expr)2 / COUNT(expr)) / COUNT(expr). Dies gibt die Populationsabweichung eines Satzes von Zahlen zurück, nachdem die Nullwerte in diesem Satz verworfen wurden
VAR_SAMP: (SUM(expr2) – SUM(expr)2 / COUNT(expr)) / (COUNT(expr) – 1). Dies gibt die Stichprobenabweichung eines Satzes von Zahlen zurück, nachdem die Nullwerte in diesem Satz verworfen wurden.
STDDEV_POP: SQRT(VAR_POP). Berechnet die Populationsstandardabweichung und gibt die Quadratwurzel der Populationsabweichung zurück.
STDDEV_SAMP: SQRT(VAR_SAMP). Berechnet die kumulative Stichproben-Standardabweichung und gibt die Quadratwurzel der Probenvarianz zurück.

[AS] "anzuzeigende Bezeichnung": Den Feldern kann in der Abfrage eine andere Bezeichnung (Alias) gegeben werden. **AS** ist zur Einführung eines Alias nicht erforderlich.

"Tabellenname".* | * [, ...]
Jedes anzuzeigende Feld kann mit seinem Feldnamen, getrennt durch Komma, angegeben werden. Werden Felder aus mehreren Tabellen in der Abfrage aufgeführt, so ist zusätzlich eine Kombination mit dem Tabellennamen notwendig: **"Tabellenname"."Feldname"**.

Statt einer ausführlichen Formulierung kann auch der gesamte Inhalt einer Tabelle angezeigt werden. Hierfür steht das Symbol «*».

[INTO [CACHED | TEMP | TEXT] "neueTabelle"] (HSQLDB**)**[12]
Das Ergebnis dieser Abfrage soll direkt in eine **neue** Tabelle geschrieben werden. Die neue Tabelle wird hier benannt. Die Definition der Feldeigenschaften der neuen Tabelle wird dabei aus der Definition der Felder, die in der Abfrage enthalten sind, erstellt.
Das Schreiben in eine Tabelle funktioniert nicht vom Abfrageeditor aus, da dieser nur anzeigbare Ergebnisse liefert. Hier muss die Eingabe über **Extras → SQL** erfolgen. Die Tabelle, die entsteht, ist anschließend erst einmal nicht editierbar, da ein Primärschlüsselfeld fehlt.

Standardmäßig wird der Inhalt in eine Tabelle des Typs **CACHED** geschrieben, der dem Typ der anderen Tabellen entspricht. Der Typ **TEMP** ist, wie bei den Tabellen beschrieben, nur begrenzt in Base nutzbar. Der Typ **TEXT** hingegen exportiert den Inhalt der Abfrage in eine kommaseparierte Textdatei, die auch von Tabellenkalkulationsprogrammen eingelesen werden kann. Die Datei liegt anschließend in dem Verzeichnis, in dem auch die *.odb-Datei der Datenbank liegt. Außerdem wird die Datei anschließend als Ansicht in dem Tabellenordner der Datenbank angezeigt. Hierbei ist allerdings zu beachten, dass ein solcher Export standardmäßig im ASCII- Zeichensatz erfolgt, Sonderzeichen also nicht berücksichtigt werden.

Um Sonderzeichen anzeigen zu können, muss der Zeichensatz geändert werden. Dies kann entweder durch einen SQL-Befehl geschehen, der den Zeichensatz für die aktuelle Sitzung einstellt, oder aber durch eine dauerhafte Einstellung in der Datenbankdatei.

Die vorübergehende Änderung ist durch die Eingabe von
SET PROPERTY "textdb.encoding" 'UTF-8';
unter **Extras → SQL** möglich. Hier kann natürlich auch entsprechend 'ansi' gewählt werden.

Zur dauerhaften Änderung des Zeichensatzes muss die Datenbankdatei entpackt werden. Das in dieser Datei liegende Verzeichnis **database** enthält eine Datei **properties**. Diese muss um einen Eintrag erweitert werden: **textdb.encoding=UTF-8** für Linux-Systeme oder **textdb.encoding=ansi** für Windows-Systeme. Wie der Zugriff auf diese Einstellungen möglich ist, wird im Anhang im Kapitel zur *Datenbankreparatur* beschrieben.

12 Für andere Datenbanken ist eine entsprechende Erweiterung verfügbar, die den Export in eine *.csv-Datei regelt: *https://extensions.openoffice.org/en/project/export-csv-base*

FROM <Tabellenliste>

```
001 "Tabellenname 1" [{CROSS | INNER | LEFT OUTER | RIGHT OUTER} JOIN
    "Tabellenname 2" ON Expression] [, ...]
```

Die Tabellen, aus denen die Daten zusammengesucht werden sollen, werden in der Regel durch Komma getrennt aufgeführt. Die Beziehung der Tabellen zueinander wird anschließend mit dem Schlüsselwort **WHERE** definiert.

Werden die Tabellen durch einen **JOIN** miteinander verbunden, so wird die Beziehung der Tabellen zueinander direkt nach der jeweils folgenden Tabelle mit dem Begriff **ON** beginnend definiert.

Ein einfacher **JOIN** bewirkt, dass nur die Datensätze angezeigt werden, auf die die Bedingung in beiden Tabellen zutrifft. Beispiel:

```
001 SELECT
002    "Tabelle1"."Name",
003    "Tabelle2"."Klasse"
004 FROM "Tabelle1",
005    "Tabelle2"
006 WHERE "Tabelle1"."KlasseID" = "Tabelle2"."ID"
```

entspricht von der Wirkung her

```
001 SELECT
002    "Tabelle1"."Name",
003    "Tabelle2"."Klasse"
004 FROM "Tabelle1"
005    JOIN "Tabelle2"
006       ON "Tabelle1"."KlasseID" = "Tabelle2"."ID"
```

Es werden hier die Namen und die dazugehörigen Klassen aufgelistet. Fehlt zu einem Namen eine Klasse, so wird der Name nicht aufgelistet. Fehlen zu einer Klasse Namen, so werden diese ebenfalls nicht aufgelistet. Der Zusatz **INNER** bewirkt hierbei keine Änderung.

```
001 SELECT
002    "Tabelle1"."Name",
003    "Tabelle2"."Klasse"
004 FROM "Tabelle1"
005    LEFT JOIN "Tabelle2"
006       ON "Tabelle1"."KlasseID" = "Tabelle2"."ID"
```

Bei dem Zusatz **LEFT** würden auf jeden Fall alle Inhalte von "Name" aus "Tabelle1" angezeigt – auch die, zu denen keine "Klasse" existiert. Beim Zusatz **RIGHT** hingegen würden alle Klassen angezeigt – auch die, zu denen kein Name existiert. Der Zusatz **OUTER** muss hier nicht unbedingt mit angegeben werden.

Tipp

Sollen auf jeden Fall alle Klassen (auch ohne Namen) und alle Namen (auch ohne Klassen) angezeigt werden, so geht dieses über **UNION** SQL ~~GUI~~:

```
001 SELECT
002    "Tabelle1"."Name",
003    "Tabelle2"."Klasse"
004 FROM "Tabelle1"
005    LEFT JOIN "Tabelle2"
006       ON "Tabelle1"."KlasseID" = "Tabelle2"."ID"
007 UNION
008 SELECT
009    "Tabelle1"."Name",
010    "Tabelle2"."Klasse"
011 FROM "Tabelle1"
012    RIGHT JOIN "Tabelle2"
013       ON "Tabelle1"."KlasseID" = "Tabelle2"."ID"
```

Hier wird lediglich einmal die Abfrage mit einem **LEFT JOIN** und dann die gleiche Abfrage mit einem **RIGHT JOIN** erstellt. **UNION** fasst die Abfrage im direkten SQL-Mode zu einer Abfrage zusammen und entfernt dabei alle Duplikate.
Der Befehl UNION ist weiter unten noch ausführlicher behandelt.

Zusätzliche Filter einzelner Tabellen sollten als **Unterabfrage** eingefügt werden:

```
001 SELECT
002    "Tabelle1".*,
003    "Tabelle2".*,
004    "Tabelle3".*,
005 FROM "Tabelle1"
006    LEFT JOIN
007    (SELECT * FROM "Tabelle2" WHERE "Name" = 'BigBoss') AS "Tabelle2"
008       ON "Tabelle1"."Tab2ID" = "Tabelle2"."ID"
009    LEFT JOIN
010    (SELECT * FROM "Tabelle3" WHERE "Ort" = 'Hintertupfingen')
011       AS "Tabelle3"
012       ON "Tabelle2"."Tab3ID" = "Tabelle3"."ID"
```

Von "Tabelle1" werden **alle** Datensätze angezeigt. Aus "Tabelle2" werden nur die Datensätze angezeigt, die im Feld "Name" 'BigBoss' stehen haben. "Tabelle3" wird an "Tabelle2" angehängt. Es werden also in "Tabelle3" nur die Datensätze angezeigt, die mit Datensätzen von "Tabelle2" zu verbinden sind, in denen im Feld "Name" 'BigBoss' steht. Zusätzlich wird die Anzeige bei "Tabelle3" auf die Datensätze begrenzt, die den "Ort" 'Hintertupfingen' enthalten.

Wird statt einer Unterabfrage die Bedingung direkt an die Beziehungsdefinition (mit **WHERE** oder **AND**) angehängt, so werden dadurch alle Datensätze gefiltert und gegebenenfalls nicht alle Datensätze von "Tabelle1" angezeigt.

```
001 SELECT
002    "Tabelle1"."Spieler1",
003    "Tabelle2"."Spieler2"
004 FROM "Tabelle1" AS "Tabelle1"
005    CROSS JOIN "Tabelle2" AS "Tabelle2"
006 WHERE "Tabelle1"."Spieler1" <> "Tabelle2"."Spieler2"
```

Beim **CROSS JOIN** müssen auf jeden Fall die Tabellen mit einem Aliasnamen versehen werden, wobei das Hinzufügen des Begriffes **AS** nicht unbedingt notwendig ist. Es werden einfach alle Datensätze aus der ersten Tabelle mit allen Datensätzen der zweiten Tabelle gekoppelt. So ergibt die obige Abfrage alle möglichen Paarungen aus der ersten Tabelle mit denen aus der zweiten Tabelle mit Ausnahme der Paarungen, bei denen es sich um gleiche Spieler handelt. Die Bedingung darf beim **CROSS JOIN** allerdings **keine Verknüpfung der Tabellen mit ON** enthalten. Stattdessen können unter **WHERE** Bedingungen eingegeben wer-

den. Würde hier die Bedingung genauso formuliert wie beim einfachen JOIN, so wäre das Ergebnis gleich:

```
001 SELECT
002    "Tabelle1"."Name",
003    "Tabelle2"."Klasse"
004 FROM "Tabelle1"
005    JOIN "Tabelle2"
006 ON "Tabelle1"."KlasseID" = "Tabelle2"."ID"
```

liefert das gleiche Ergebnis wie

```
001 SELECT
002    "Tabelle1"."Name",
003    "Tabelle2"."Klasse"
004 FROM "Tabelle1" AS "Tabelle1"
005    CROSS JOIN "Tabelle2" AS "Tabelle2"
006 WHERE "Tabelle1"."KlasseID" = "Tabelle2"."ID"
```

[WHERE SQL-Expression]

Die Standardeinleitung, um Bedingungen für eine genauere Filterung der Daten zu formulieren. Hier werden in der Regel auch die Beziehungen der Tabellen zueinander definiert, sofern die Tabellen nicht mit JOIN verbunden sind.

[GROUP BY SQL-Expression [, …]]

Wenn Felder mit einer bestimmten Funktion bearbeitet werden (z. B. **COUNT, SUM** …), so sind alle Felder, die nicht mit einer Funktion bearbeitet werden, aber angezeigt werden sollen, mit **GROUP BY** zu einer Gruppe zusammen zu fassen.
Beispiel:

```
001 SELECT
002    "Name",
003    SUM("Einnahme"-"Ausgabe") AS "Saldo"
004 FROM "Tabelle1"
005 GROUP BY "Name";
```

Datensätze mit gleichen Namen werden jetzt aufsummiert. Im Ergebnis wird jeweils **Einnahme – Ausgabe** ermittelt und darüber die Summe, die jede Person erzielt hat, aufgelistet. Das Feld wird unter dem Namen **Saldo** dargestellt.

[HAVING SQL-Expression]

Die **HAVING**-Formulierung ähnelt sehr der **WHERE**-Formulierung. Sie kann für Bedingungen eingesetzt werden, die mit Hilfe von Funktionen wie MIN, MAX formuliert werden. **Nur HAVING ist dafür geeignet, in den Bedingungen auch Berechnungen durchzuführen.** HAVING erscheint dabei nach einer eventuell vorhandenen GROUP BY – Formulierung.
Beispiel:

```
001 SELECT
002    "Name",
003    "Laufzeit"
004 FROM "Tabelle1"
005 GROUP BY "Name",
006    "Laufzeit"
007 HAVING MIN("Laufzeit") < '00:40:00';
```

Es werden alle Namen und Laufzeiten aufgelistet, bei denen die Laufzeit weniger als 40 Minuten beträgt.

[SQL Expression]

SQL-Ausdrücke werden nach dem folgenden Schema miteinander verbunden:

```
001 [NOT] Bedingung [{ OR | AND } Bedingung]
```

Beispiel:

```
001 SELECT
002    *
003 FROM "Tabellenname"
004 WHERE
005    NOT "Rückgabedatum" IS NULL
006    AND "LeserID" = 2;
```

Aus der Tabelle werden die Datensätze ausgelesen, bei denen ein "Rückgabedatum" eingetragen wurde und die "LeserID" gleich 2 ist. Das würde in der Praxis bedeuten, dass alle Medien, die eine bestimmte Person ausgeliehen und wieder zurückgegeben hat, damit ermittelt werden könnten. Die Bedingungen sind nur durch **AND** miteinander verbunden. Das **NOT** bezieht sich rein auf die erste Bedingung.

```
001 SELECT
002    *
003 FROM "Tabellenname"
004 WHERE NOT ("Rückgabedatum" IS NULL AND "LeserID" = 2);
```

Wird eine Klammer um die Bedingung gesetzt und **NOT** steht außerhalb der Klammer, so werden genau die Datensätze angezeigt, die die in den Klammern stehenden Bedingungen zusammen komplett nicht erfüllen. Das wären alle Datensätze mit Ausnahme derer, die "LeserID" mit der Nummer 2 noch nicht zurückgegeben hat.

[SQL Expression]: Bedingungen

```
001 { Wert [|| Wert]
```

Ein Wert kann einzeln oder mit mehreren Werten zusammen über zwei senkrechte Striche **||** kombiniert werden. Dies gilt dann natürlich auch für Feldinhalte.

```
001 SELECT
002    "Nachname" || ', ' || "Vorname" AS "Name"
003 FROM "Tabellenname"
```

Die Inhalte aus den Feldern "Nachname" und "Vorname" werden in einem Feld "Name" gemeinsam angezeigt. Dabei wird ein Komma und eine Leertaste zwischen "Nachname" und "Vorname" eingefügt.

```
001 | Wert { = | < | <= | > | >= | <> | != } Wert
```

Die Zeichen entsprechend den aus der Mathematik bekannten Operatoren:
{ Gleich | kleiner als | kleiner oder gleich | größer als | größer oder gleich | nicht gleich | nicht gleich }
Firebird kennt hier noch die folgenden Alternativen, die allerdings nur mit direkten SQL funktionieren:

```
001 { ~= | ^= | !> | ~> | ^> | !< | ~< | ^< }
```

{ Nicht gleich | nicht gleich | nicht größer als | nicht größer als | nicht größer als | nicht kleiner als | nicht kleiner als | nicht kleiner als }

```
001 | Wert IS [NOT] NULL
```

Das entsprechende Feld hat keinen Inhalt, ist auch nicht beschrieben worden. Dies kann in der GUI nicht unbedingt beurteilt werden, denn ein leeres Textfeld bedeutet noch nicht, dass das Feld völlig ohne Inhalt ist. Die Standardeinstellung von Base ist aber so, dass leere Felder in der Datenbank auf **NULL** gesetzt werden.

```
001 | EXISTS(Abfrageaussage)
```

Beispiel:

```
001 SELECT
002    "Name"
003 FROM "Tabelle1"
004 WHERE EXISTS
005    (SELECT
006       "Vorname"
007    FROM "Tabelle2"
```

```
008    WHERE "Tabelle2"."Vorname" = "Tabelle1"."Name")
```

Es werden die Namen aus Tabelle1 aufgeführt, die als Vornamen in Tabelle2 verzeichnet sind.

```
001 | SINGULAR(Abfrageaussage) (HSQLDB, FIREBIRD)
```

Beispiel:

```
001 SELECT
002    "Name"
003 FROM "Tabelle1"
004 WHERE SINGULAR
005    (SELECT
006       "Vorname"
007    FROM "Tabelle2"
008    WHERE "Tabelle2"."Vorname" = "Tabelle1"."Name")
```

Es werden die Namen aus Tabelle1 aufgeführt, die als Vornamen in Tabelle2 nur genau einmal verzeichnet sind.

```
001 | Wert BETWEEN Wert AND Wert
```

BETWEEN Wert1 AND Wert2 gibt alle Werte ab Wert1 bis einschließlich Wert2 wieder. Werden hier Buchstaben als Werte eingesetzt, so wird die alphabetische Sortierung angenommen, wobei Kleinbuchstaben und Großbuchstaben die gleichen Werte haben.

```
001 SELECT
002    "Name"
003 FROM "Tabellenname"
004 WHERE "Name" BETWEEN 'A' AND 'E';
```

Diese Abfrage gibt alle Namen wieder, die mit A, B, C und D beginnen (ggf. auch mit entsprechendem Kleinbuchstaben). Da als unterer Begrenzung E gesetzt wurde, sind alle Namen mit E nicht mehr in der Auswahl enthalten. Der Buchstabe E würde in einer Sortierung ganz am Anfang der Namen mit E stehen.

```
001 | Wert [NOT] IN ( {Wert [, ...] | Abfrageaussage } )
```

Hier wird entweder eine Liste von Werten oder eine Abfrage eingesetzt. Die Bedingung ist erfüllt, wenn der Wert in der Werteliste bzw. im Abfrageergebnis enthalten ist.

```
001 | Wert [NOT] LIKE Wert [ESCAPE] Wert }
```

Der **LIKE**-Operator ist derjenige, der in vielen einfachen Suchfunktionen benötigt wird. Die Angabe der Werte erfolgt hier nach folgendem Muster:
'%' steht für beliebig viele, ggf. auch 0 Zeichen,
'_' ersetzt genau ein Zeichen.
Um nach '%' oder '_' selbst zu suchen müssen die Zeichen direkt nach einem zweiten Zeichen auftauchen, das nach **ESCAPE** definiert wird.

```
001 SELECT
002    "Name"
003 FROM "Tabellenname"
004 WHERE "Name" LIKE '\_%' ESCAPE '\'
```

Diese Abfrage zeigt alle Namen auf, die mit einem Unterstrich beginnen. Als **ESCAPE**-Zeichen ist hier '\' definiert worden.

```
001 | Wert [NOT] STARTING WITH Wert }(HSQLDB, FIREBIRD)
```

Hier wird nur nach dem Beginn des Strings in einem Feld gesucht.
STARTING WITH 'Li' ergibt das Gleiche wie **LIKE 'Li%'**.

```
001 | Wert [NOT] CONTAINING Wert }(HSQLDB, FIREBIRD)
```

Ein Feld wird mit einem Wert (String) verglichen. Enthält das Feld den String oder die Zahlenkombination, so wird das Feld wiedergegeben.
CONTAINING 'Li' ergibt das Gleiche wie **LIKE '%Li%'**.

```
001 | Wert IS [NOT] DISTINCT FROM Wert }(HSQLDB, FIREBIRD)
```

Ein Wert ist dann **NOT DISTINCT** von einem anderen Wert, wenn er gleich ist oder wenn beide Inhalte **NULL** sind.
"Feld1" NOT DISTINCT "Feld2" ergibt das Gleiche wie **"Feld1" = "Feld2" OR ("Feld1" IS NULL AND "Feld2" IS NULL)**

```
001 | string-expression [NOT] SIMILAR TO <pattern> [ESCAPE <escape-char>]
    (HSQLDB, FIREBIRD)
002     <pattern>     ::= ein regulärer SQL-Ausdruck
003     <escape-char> ::= ein einzelnen Zeichen
```

SIMILAR TO[13] vergleicht eine Zeichenkette, also auch den Inhalt eines Tabellenfeldes, mit einem in SQL definierten regulären Ausdruck. Anders als in einigen anderen Sprachen muss das Muster mit der gesamten Zeichenfolge übereinstimmen, um erfolgreich zu sein - die Übereinstimmung eines Teilstrings reicht nicht aus. Wenn ein Operand **NULL** ist, ist das Ergebnis **NULL**. Andernfalls ist das Ergebnis **TRUE** oder **FALSE**.

Die folgende Syntax definiert das Format des regulären SQL-Ausdrucks. Es handelt sich um eine vollständige Top-Down-Definition. Die Definition ist sehr formell, ziemlich lang und wahrscheinlich perfekt geeignet, um alle zu entmutigen, die nicht bereits einige Erfahrungen mit regulären Ausdrücken (oder mit sehr formalen, ziemlich langen Top-down-Definitionen) haben. Deshalb wird in einem weiteren Abschnitt die Erstellung regulärer Ausdrücke an Beispielen erklärt.

Syntax des regulären SQL-Ausdrucks

```
001    <regular expression>    ::= <regular term> ['|' <regular term> ...]
002    <regular term>          ::= <regular factor> ...
003    <regular factor>        ::= <regular primary> [<quantifier>]
004    <quantifier>            ::= ?
005                             | *
006                             | +
007                             | '{' <m> [,[<n>]] '}'
008    <m>, <n>                ::= Integer ohne Vorzeichen, mit <m> <= <n>, wenn beide
                                   angegeben werden
009    <regular primary>       ::= <character>
010                             | <character class>
011                             | %
012                             | (<regular expression>)
013    <character>             ::= <escaped character>
014                             | <non-escaped character>
015    <escaped character>     ::= <escape-char> <special character>
016                             | <escape-char> <escape-char>
017    <special character>     ::= eines der Zeichen []()|^-+*%_?{
018    <non-escaped character> ::= Eines der Zeichen, das nicht ein <special character>
                                   und nicht gleich <escape-char> ist, falls definiert
019    <character class>       ::= '_'
020                             | '[' <member> ... ']'
021                             | '[^' <non-member> ... ']'
022                             | '[' <member> ... '^' <non-member> ... ']'
023    <member>, <non-member>  ::= <character>
024                             | <range>
025                             | <predefined class>
026    <range>                 ::= <character>-<character>
027    <predefined class>      ::= '[:' <predefined class name> ':]'
028    <predefined class name> ::= ALPHA | UPPER | LOWER | DIGIT | ALNUM | SPACE |
                                   WHITESPACE
```

Erstellen regulärer Ausdrücke

Zeichen

Innerhalb regulärer Ausdrücke repräsentieren sich die meisten Zeichen selbst. Die einzigen Ausnahmen sind die folgenden Sonderzeichen:

13 Übersetzung aus der Firebird 2.5 Language Reference siehe:*http://www.firebirdsql.org/en/documentation/*

```
001    [ ] ( ) | ^ - + * % _ ? {
```

...und das Escape-Zeichen, wenn es definiert ist.

Ein regulärer Ausdruck, der keine Sonder- oder Escape-Zeichen enthält, stimmt nur mit identischen Strings überein (abhängig von dem verwendeten Zeichensatz). Das heißt, es funktioniert genau wie der "=" - Operator:

```
001    'Apple' similar to 'Apple'  -- true
002    'Apples' similar to 'Apple' -- false
003    'Apple' similar to 'Apples' -- false
004    'APPLE' similar to 'Apple'  -- abhängig vom verwendeten Zeichensatz
```

Wildcards

Die bekannten SQL-Wildchards entsprechen einem einzelnen Zeichen (_)und einem String jeder beliebigen Länge (%):

```
001    'Birne' similar to 'B_rne'   -- true
002    'Birne' similar to 'B_ne'    -- false
003    'Birne' similar to 'B%ne'    -- true
004    'Birne' similar to 'Bir%ne%' -- true
005    'Birne' similar to 'Birr%ne' -- false
```

% kann auch für einen leeren String stehen.

Zeichenklassen

Ein Sammlung von Zeichen, die in Klammern eingeschlossen sind, definiert eine Zeichenklasse. Ein Zeichen in dem String entspricht einer Klasse im Muster, wenn das Zeichen in der Klasse enthalten ist:

```
001    'Citroen' similar to 'Cit[arju]oen'  -- true
002    'Citroen' similar to 'Ci[tr]oen'     -- false
003    'Citroen' similar to 'Ci[tr][tr]oen' -- true
```

Wie aus der zweiten Zeile ersichtlich ist, entspricht die Klasse nur einem einzelnen Zeichen, nicht mehreren Zeichen hintereinander.

Innerhalb einer Klassendefinition definieren zwei Zeichen, die durch einen Bindestrich verbunden sind, einen Bereich. Ein Bereich umfasst die beiden Endpunkte und alle Zeichen, die zwischen ihnen in der aktiven Sortierung liegen. Bereiche können an beliebiger Stelle in der Klassendefinition platziert werden, ohne dass spezielle Trennzeichen vorhanden sind, um sie von den anderen Elementen getrennt zu halten.

```
001    'Datte' similar to 'Dat[q-u]e'        -- true
002    'Datte' similar to 'Dat[abq-uy]e'     -- true
003    'Datte' similar to 'Dat[bcg-km-pwz]e' -- false
```

Die folgenden vordefinierten Zeichenklassen können auch in einer Klassendefinition verwendet werden:

```
001 [:ALPHA:]
    Buchstaben a..z und A..Z. Abhängig vom Zeichensatz der Datenbank enthält dies auch
    entsprechende Sonderzeichen.
002 [:DIGIT:]
    Dezimalziffern 0..9.
003 [:ALNUM:]
    Zusammenschluss von [:ALPHA:] und [:DIGIT:].
004 [:UPPER:]
    Großgeschriebene Buchstaben A..Z. Abhängig vom Zeichensatz auch Kleinbuchstaben wie
    z.B. 'ß'.
005 [:LOWER:]
    Kleingeschriebene Buchstaben a..z. Abhängig vom Zeichensatz gegebenenfalls auch
    Großbuchstaben.
006 [:SPACE:]
    Leerzeichen (ASCII 32).
007 [:WHITESPACE:]
    Vertikaler Tabulator (ASCII 9), Zeilenvorschub (ASCII 10), horizontaler Tabulator
    (ASCII 11), Seitenvorschub (ASCII 12), Wagenrücklauf (ASCII 13) und Leerzeichen
    (ASCII 32).
```

Der Einschluss einer vordefinierten Klasse hat die gleiche Wirkung wie die Aufzählung aller ihrer Mitglieder. Vordefinierte Klassen sind nur innerhalb von Klassendefinitionen zulässig. Wenn nur mit einer vordefinierten Klasse verglichen werden soll, so muss ein zusätzliches Paar von Klammern um die vordefinierte Klasse gelegt werden:

```
001    'Erdbeere' similar to 'Erd[[:ALNUM:]]eere'    -- true
002    'Erdbeere' similar to 'Erd[[:DIGIT:]]eere'    -- false
003    'Erdbeere' similar to 'Erd[a[:SPACE:]b]eere'  -- true
004    'Erdbeere' similar to [[:ALPHA:]]               -- false
005    'E' similar to [[:ALPHA:]]                    -- true
```

Wenn eine Klassendefinition mit einem Caret '^' beginnt, darf alles, was folgt, nicht im String an der Stelle existieren:

```
001    'Framboise' similar to 'Fra[^ck-p]boise'       -- false
002    'Framboise' similar to 'Fr[^a][^a]boise'       -- false
003    'Framboise' similar to 'Fra[^[:DIGIT:]]boise'  -- true
```

Wenn das Caret nicht am Anfang der Sequenz platziert wird, enthält die Klasse alles vor dem Caret, mit Ausnahme der Elemente, die auch nach dem Caret auftreten:

```
001    'Grapefruit' similar to 'Grap[a-m^f-i]fruit'  -- true
002    'Grapefruit' similar to 'Grap[abc^xyz]fruit'  -- false
003    'Grapefruit' similar to 'Grap[abc^de]fruit'   -- false
004    'Grapefruit' similar to 'Grap[abe^de]fruit'   -- false
005    '3' similar to '[[:DIGIT:]^4-8]'              -- true
006    '6' similar to '[[:DIGIT:]^4-8]'              -- false
```

Schließlich ist die bereits erwähnte Wildcard "_" eine eigenständige Zeichenklasse, die mit jedem einzelnen Zeichen übereinstimmt.

Quantoren

Ein Fragezeichen unmittelbar nach einem Zeichen oder einer Klasse weist darauf hin, dass das vorhergehende Element 0 oder 1 Mal auftreten kann:

```
001    'Hallon' similar to 'Hal?on'             -- false
002    'Hallon' similar to 'Hal?lon'            -- true
003    'Hallon' similar to 'Halll?on'           -- true
004    'Hallon' similar to 'Hallll?on'          -- false
005    'Hallon' similar to 'Halx?lon'           -- true
006    'Hallon' similar to 'H[a-c]?llon[x-z]?'  -- true
```

Ein Stern '*', der unmittelbar einem Zeichen oder einer Klasse folgt, zeigt an, dass das vorhergehende Element 0 oder mehrere Male auftreten kann:

```
001    'Icaque' similar to 'Ica*que'       -- true
002    'Icaque' similar to 'Icar*que'      -- true
003    'Icaque' similar to 'I[a-c]*que'    -- true
004    'Icaque' similar to '_*'            -- true
005    'Icaque' similar to '[[:ALPHA:]]*'  -- true
006    'Icaque' similar to 'Ica[xyz]*e'    -- false
```

Ein Pluszeichen unmittelbar nach einem Zeichen oder einer Klasse weist darauf hin, dass das vorhergehende Element 1 oder mehrere Male auftreten muss:

```
001    'Jujube' similar to 'Ju_+'               -- true
002    'Jujube' similar to 'Ju+jube'            -- true
003    'Jujube' similar to 'Jujuber+'           -- false
004    'Jujube' similar to 'J[jux]+be'          -- true
005    'Jujube' similar to 'J[[:DIGIT:]]+ujube'  -- false
```

Wenn einem Zeichen oder einer Klasse eine in geschweiften Klammern eingeschlossene Zahl folgt, muss die Überprüfung genau so oft wiederholt werden:

```
001    'Kiwi' similar to 'Ki{2}wi'     -- false
002    'Kiwi' similar to 'K[ipw]{2}i'  -- true
003    'Kiwi' similar to 'K[ipw]{2}'   -- false
004    'Kiwi' similar to 'K[ipw]{3}'   -- true
```

Wenn der Zahl ein Komma folgt, muss das Element mindestens so oft wiederholt werden:

```
001    'Limone' similar to 'Li{2,}mone'     -- false
002    'Limone' similar to 'Li{1,}mone'     -- true
003    'Limone' similar to 'Li[nezom]{2,}'  -- true
```

Wenn die geschweiften Klammern zwei Zahlen enthalten, die durch ein Komma getrennt sind, und die zweite Zahl nicht kleiner als die erste ist, dann muss das Element mindestens so oft wie die erste Zahl und höchstens so oft die zweite Zahl wiederholt werden:

```
001    'Mandarijn' similar to 'M[a-p]{2,5}rijn'   -- true
002    'Mandarijn' similar to 'M[a-p]{2,3}rijn'   -- false
003    'Mandarijn' similar to 'M[a-p]{2,3}arijn'  -- true
```

Die Quantoren ?, * und + entsprechen jeweils Kurzformen von {0,1}, {0,} und {1,}.

ODER-Ausdrücke

Reguläre Ausdrücke können mit dem '|'-Operator erzeugt werden. Eine Übereinstimmung ist dann gegeben, wenn der Argumentstring mit mindestens einem der Begriffe übereinstimmt:

```
001    'Nektarin' similar to 'Nek|tarin'         -- false
002    'Nektarin' similar to 'Nektarin|Persika'  -- true
003    'Nektarin' similar to 'M_+|N_+|P_+'       -- true
```

Unterausdrücke

Ein oder mehrere Teile des regulären Ausdrucks können in Unterausdrücken (auch als Untermuster bezeichnet) gruppiert werden, indem sie zwischen Klammern platziert werden. Ein Unterausdruck ist ein regulärer Ausdruck in seinem eigenen Bereich. Er kann alle Elemente enthalten, die in einem regulären Ausdruck erlaubt sind. Es können auch Quantifizierer hinzugefügt werden:

```
001    'Orange' similar to 'O(ra|ri|ro)nge'       -- true
002    'Orange' similar to 'O(r[a-e])+nge'        -- true
003    'Orange' similar to 'O(ra){2,4}nge'        -- false
004    'Orange' similar to 'O(r(an|in)g|rong)?e'  -- true
```

Maskieren von Sonderzeichen

Um einem Zeichen zu entsprechen, das in regulären Ausdrücken enthalten ist, muss dieses Zeichen maskiert werden. Es gibt kein Standard-Zeichen zur Maskierung (Escape-Zeichen). Das Escape-Zeichen legt der Benutzer bei Bedarf fest:

```
001    'Peer (Poire)' similar to 'P[^ ]+ \(P[^ ]+\)' escape '\'  -- true
002    'Pera [Pear]' similar to 'P[^ ]+ #[P[^ ]+#]' escape '#'   -- true
003    'Päron-Äppledryck' similar to 'P%$-Ä%' escape '$'         -- true
004    'Pärondryck' similar to 'P%--Ä%' escape '-'               -- false
```

Die letzte Zeile zeigt, dass das Escape-Zeichen bei Bedarf auch zur Maskierung des eigenen Zeichens genutzt werden kann.

Hinweis

Datumswerte können bei der internen Datenbank im Format 'YYYY-MM-DD' angegeben werden. Bei externen Datenbanken kann es aber passieren, dass das Datum unter diesen Umständen nicht korrekt gelesen werden kann. Hier kann die ältere Version {D 'YYYY-MM-DD'} oder die neuere Version {d 'YYYY-MM-DD'} zum Erfolg führen.
Entsprechende Formate existieren auch für Zeitfelder und für Datums-Zeit-Felder: {t 'HH:MI:SS[.SS]'} bzw. {ts 'YYYY-MM-DD HH:MI:SS[.SS]'}.
Entsprechend sind die Buchstaben «D» und «T» sowie die Kombination «TS» reservierte Abkürzungen, die nicht an anderer Stelle alleinstehend benutzt werden können. Eine Parameterabfrage mit dem Parameter «:D» wird z.B. nicht funktionieren.
Siehe hierzu auch die Hilfe von LO zum Stichwort «Abfrageentwurf».

[SQL Expression]: Werte

```
001 [+ | -] { Ausdruck [{ + | - | * | / | || } Ausdruck]
```

Vorzeichen vor den Werten sind möglich. Die Addition, Subtraktion, Multiplikation, Division und Verkettung von Ausdrücken ist erlaubt. Beispiel für eine Verkettung:

```
001 SELECT
002    "Nachname"||', '||"Vorname"
003 FROM "Tabelle"
```

Auf diese Art und Weise werden Datensätze in der Abfrage als ein Feld ausgegeben, in der "Nachname, Vorname" steht. Die Verkettung erlaubt also jeden der weiter unten genannten Ausdrücke.

```
001 | ( Bedingung )
```

Siehe hierzu den vorhergehenden Abschnitt

```
001 | Funktion ( [Parameter] [,...] )
```

Siehe hierzu im Anhang das Kapitel *Eingebaute Funktionen und abgespeicherte Prozeduren*.

Die folgenden Abfragen werden auch als Unterabfragen (Subselects) bezeichnet.

```
001 | Abfrageaussage, die nur genau einen Wert ergibt
```

Da ein Datensatz für jedes Feld nur einen Wert darstellen kann, kann auch nur die Abfrage komplett angezeigt werden, die genau einen Wert ergibt.

```
001 | {ANY|ALL} (Abfrageaussage, die den Inhalt einer ganzen Spalte
    wiedergibt)
```

Manchmal gibt es Bedingungen, bei denen ein Ausdruck mit einer ganzen Gruppe von Werten verglichen wird. Zusammen mit **ANY** bedeutet das, dass der Ausdruck mindestens einmal in der Gruppe vorkommen muss. Dies ließe sich auch mit der **IN**-Bedingung beschreiben. **= ANY** ergibt das gleiche Ergebnis wie **IN**, funktioniert aber nur mit einer Unterabfrage. Statt **ANY** kann in Firebird alternativ auch **SOME** genutzt werden.

Zusammen mit **ALL** bedeutet dies, dass alle Werte der Gruppe dem einen Ausdruck entsprechen müssen.

[SQL Expression]: Ausdrücke

```
001 { 'Text' | Ganzzahl | Fließkommazahl
002 | ["Tabelle".]"Feld" | TRUE | FALSE | NULL }
```

Als Grundlage dienen Werte, die, abhängig vom Quellformat, mit unterschiedlichen Ausdrücken angegeben werden. Wird nach Textinhalten gesucht, so ist der Inhalt in Hochkommata zu setzen. Ganzzahlen werden ohne Hochkommata geschrieben, ebenso Fließkommazahlen (statt Komma ist in SQL direkt der Dezimalpunkt zu setzen).
Felder stehen für die Werte, die sich in den Feldern einer Tabelle befinden. Meist werden entweder Felder miteinander verglichen oder Felder mit Werten. In SQL werden die Feldbezeichnungen besser in doppelte Anführungsstriche gesetzt, da sonst eventuell Feldbezeichnungen nicht richtig erkannt werden. Üblicherweise geht SQL ohne doppelte Anführungsstriche davon aus, dass alles ohne Sonderzeichen, in einem Wort und mit Großbuchstaben geschrieben ist. Sind mehrere Tabellen in der Abfrage enthalten, so ist neben dem Feld auch die Tabelle, vom Feld getrennt durch einen Punkt, aufzuführen.
TRUE und **FALSE** stammen üblicherweise von Ja/Nein-Feldern.
NULL bedeutet, dass nichts angegeben wurde. Es ist nicht gleichbedeutend mit 0, sondern eher mit Leer.

UNION [ALL | DISTINCT] Abfrageaussage `SQL` ~~GUI~~

Hiermit werden Abfragen so verknüpft, dass der Inhalt der 2. Abfrage unter die erste Abfrage geschrieben wird. Dazu müssen alle Felder der beiden Abfragen vom Typ her übereinstimmen. Diese Verknüpfung von mehreren Abfragen funktioniert nur über die direkte Ausführung des SQL-Kommandos.

```
001 SELECT
002    "Vorname"
003 FROM "Tabelle1"
```

```
004    UNION DISTINCT
005    SELECT
006       "Vorname"
007    FROM "Tabelle2";
```

Diese Abfrage liefert alle Vornamen aus Tabelle1 und Tabelle2; der Zusatz **DISTINCT** zeigt an, dass keine doppelten Vornamen ausgegeben werden. **DISTINCT** ist in diesem Zusammenhang die Standardeinstellung. Die Vornamen sind dabei standardmäßig nach dem Alphabet aufsteigend sortiert. Mit **ALL** werden einfach alle Vornamen aus Tabelle1 und Tabelle2 angezeigt. Sie sind jetzt standardmäßig nach dem ersten Feld der Anzeige, hier also "Vorname", sortiert.

Mit Hilfe dieser Abfragetechnik ist es auch möglich, Werte einer Datenzeile z.B. für eine Liste untereinander in einer Spalte anzuordnen. Angenommen es existiert eine Tabelle "Ware", in der der "Verkaufspreis" sowie ein "Rabattpreis_1" und ein "Rabattpreis_2" enthalten sind. Daraus soll der Inhalt für ein Kombinationsfeld erstellt werden, das genau diese Preise untereinander auflistet:

```
001 SELECT
002    "Verkaufspreis"
003 FROM "Ware" WHERE "Ware_ID" = 1
004    UNION
005    SELECT
006       "Rabattpreis_1"
007    FROM "Ware" WHERE "Ware_ID" = 1
008    UNION
009    SELECT
010       "Rabattpreis_2"
011    FROM "Ware" WHERE "Ware_ID" = 1;
```

Der Primärschlüsselwert für die Ware müsste hier natürlich entsprechend über ein Makro gesetzt werden, das dem Kombinationsfeld je nach Ware einen entsprechenden Inhalt zuweist.

Hinweis

Die Sortierung von mit UNION verknüpften Inhalten gelingt bei Firebird nicht über die Spaltennamen. Stattdessen muss die Position der Spalte übergeben werden:

```
001 SELECT
002    "Vorname", "Nachname"
003 FROM "Tabelle1"
004    UNION
005    SELECT
006       "Vorname", "Nachname"
007    FROM "Tabelle2"
008    ORDER BY 2, 1;
```

sortiert die Daten in Firebird zuerst nach dem Nachnamen (Spalte 2) und dann nach dem Vornamen (Spalte 1).

MINUS [DISTINCT] | EXCEPT [DISTINCT] Abfrageaussage SQL ~~GUI~~ (HSQLDB, ~~FIREBIRD~~)

```
001 SELECT
002    "Vorname"
003 FROM "Tabelle1"
004    EXCEPT
005    SELECT
006       "Vorname"
007    FROM "Tabelle2";
```

Zeigt alle Vornamen aus Tabelle1 mit Ausnahme der Vornamen an, die in Tabelle 2 enthalten sind. **MINUS** und **EXCEPT** führen zum gleichen Ergebnis. Sortierung ist alphabetisch. Dies funktioniert zur Zeit nur, wenn das SQL-Kommando direkt ausgeführt wird.

INTERSECT [DISTINCT] Abfrageaussage **SQL** ~~GUI~~ **(**HSQLDB~~, FIREBIRD~~**)**

```
001 SELECT
002    "Vorname"
003 FROM "Tabelle1"
004    INTERSECT
005    SELECT
006       "Vorname"
007    FROM "Tabelle2";
```

Hier werden nur die Vornamen angezeigt, die in beiden Tabellen vorhanden sind. Die Sortierung ist wieder alphabetisch. Dies funktioniert zur Zeit nur, wenn das SQL-Kommando direkt ausgeführt wird.

[ORDER BY Ordnungs-Expression [, ...]]

Hier können Feldnamen, die Nummer der Spalte (beginnend mit 1 von links), ein Alias (formuliert z. B. mit **AS**) oder eine Wertzusammenführung (siehe [SQL Expression]: Werte) angegeben werden. Die Sortierung erfolgt in der Regel aufsteigend (**ASC**). Nur wenn die Sortierung absteigend erfolgen soll, muss **DESC** angegeben werden.

```
001 SELECT
002    "Vorname",
003    "Nachname" AS "Name"
004 FROM "Tabelle1"
005 ORDER BY "Nachname";
```

ist identisch mit

```
001 SELECT
002    "Vorname",
003    "Nachname" AS "Name"
004 FROM "Tabelle1"
005 ORDER BY "Nachname" ASC;
```

ist identisch mit

```
001 SELECT
002    "Vorname",
003    "Nachname" AS "Name"
004 FROM "Tabelle1"
005 ORDER BY "Name";
```

Unter FIREBIRD funktioniert diese Sortierung allerdings nicht wie gewünscht, wenn Umlaute in den Bezeichnungen enthalten sind. Hier muss zusätzlich die Art der **Collation** angegeben werden:

```
001 SELECT
002    "Name"
003 FROM "Tabelle1"
004 ORDER BY "Name" COLLATE UNICODE ASC;
```

Leider funktioniert diese Abfrage nur im direkten SQL-Modus und auch nicht mit den Sortierpfeilen in der Abfrage- und Tabellen-GUI. Besser ist es, die *Sortierung von Groß- und Kleinschreibung und auch Umlauten* direkt vor der Erstellung der Tabellen in Firebird anzugeben.

Verwendung eines Alias in Abfragen

Durch Abfragen können auch Felder in einer anderen Bezeichnung wiedergegeben werden.

```
001 SELECT
002    "Vorname",
003    "Nachname" AS "Name"
004 FROM "Tabelle1"
```

Dem Feld "Nachname" wird in der Anzeige die Bezeichnung "Name" zugeordnet.

Wird eine Abfrage aus zwei Tabellen erstellt, so steht vor den jeweiligen Feldbezeichnungen der Name der Tabelle:

```
001 SELECT
002    "Tabelle1"."Vorname",
003    "Tabelle1"."Nachname" AS "Name",
004    "Tabelle2"."Klasse"
005 FROM "Tabelle1",
006    "Tabelle2"
007 WHERE "Tabelle1"."Klasse_ID" = "Tabelle2"."ID"
```

Auch den Tabellennamen kann ein Aliasname zugeordnet werden, der allerdings in der Tabellenansicht nicht weiter erscheint. Wird so ein Alias zugeordnet, so müssen sämtliche Tabellenbezeichnungen in der Abfrage entsprechend ausgetauscht werden:

```
001 SELECT
002    "a"."Vorname",
003    "a"."Nachname" AS "Name",
004    "b"."Klasse"
005 FROM "Tabelle1" AS "a",
006    "Tabelle2" AS "b"
007 WHERE "a"."Klasse_ID" = "b"."ID"
```

Die Zuweisung eines Aliasnamens kann auch verkürzt ohne den Zuweisungsbegriff **AS** erfolgen:

```
001 SELECT
002    "a"."Vorname",
003    "a"."Nachname" "Name",
004    "b"."Klasse"
005 FROM "Tabelle1" "a",
006    "Tabelle2" "b"
007 WHERE "a"."Klasse_ID" = "b"."ID"
```

Dies erschwert allerdings die eindeutige Lesbarkeit des Codes. Daher sollte nur in Ausnahmefällen auf die Verkürzung zurückgegriffen werden.

Über den Aliasnamen kann auch eine Tabelle mit entsprechenden Filterungen mehrmals innerhalb einer Abfrage genutzt werden:

```
001 SELECT "Kasse"."Betrag", "Kasse"."Datum",
002    "a"."Betrag" AS "Haben",
003    "b"."Betrag" AS "Soll"
004 FROM "Kasse"
005    LEFT JOIN "Kasse" AS "a"
006       ON "Kasse"."ID" = "a"."ID" AND "a"."Betrag" >= 0
007    LEFT JOIN "Kasse" AS "b"
008       ON "Kasse"."ID" = "b"."ID" AND "b"."Betrag" < 0
```

Hinweis

Die Formatierung in Abfragen wird nicht gespeichert. Base versucht, die Formatierung aus der entsprechenden Tabellenformatierung zu ermitteln. Durch die Verwendung eines Alias funktioniert das nicht mehr. Beträge wie oben werden also in Abfragen nur als Dezimalzahl, nicht aber mit dem Währungszeichen € versehen angezeigt.

Abfragen für die Erstellung von Listenfeldern

In Listenfeldern wird ein Wert angezeigt, der nicht an die den Formularen zugrundeliegenden Tabellen weitergegeben wird. Sie werden schließlich eingesetzt, um statt eines Fremdschlüssels zu sehen, welchen Wert der Nutzer damit verbindet. Der Wert, der schließlich in Formularen abgespeichert wird, darf bei den Listenfeldern nicht an der ersten Position stehen.

```
001 SELECT
002    "Vorname",
003    "ID"
004 FROM "Tabelle1";
```

Diese Abfrage würde alle Vornamen anzeigen und den Primärschlüssel "ID" an die dem Formular zugrundeliegende Tabelle weitergeben. So ist das natürlich noch nicht optimal. Die Vornamen erscheinen unsortiert; bei gleichen Vornamen ist außerdem unbekannt, um welche Person es sich denn handelt.

```
001 SELECT
002    "Vorname"||' '||"Nachname",
003    "ID"
004 FROM "Tabelle1"
005 ORDER BY "Vorname"||' '||"Nachname";
```

Jetzt erscheint der Vorname, getrennt von dem Nachnamen, durch ein Leerzeichen. Die Namen werden unterscheidbarer und sind sortiert. Die Sortierung folgt der Logik, dass natürlich nach den vorne stehenden Buchstaben zuerst sortiert wird, also Vornamen zuerst, dann Nachnamen. Andere Sortierreihenfolgen als nach der Anzeige des Feldes würden erst einmal verwirren.

```
001 SELECT
002    "Nachname"||', '||"Vorname",
003    "ID"
004 FROM "Tabelle1"
005 ORDER BY "Nachname"||', '||"Vorname";
```

Dies würde jetzt zu der entsprechenden Sortierung führen, die eher unserer Gewohnheit entspricht. Familienmitglieder erscheinen zusammen untereinander, bei gleichen Nachnamen und unterschiedlichen Familien wird allerdings noch durcheinander gewürfelt. Um dies zu unterscheiden müsste eine Gruppenzuordnung in der Tabelle gemacht werden.

Letztes Problem ist noch, dass eventuell auftauchende Personen mit gleichem Nachnamen und gleichem Vornamen nicht unterscheidbar sind. Variante 1 wäre, einfach einen Namenszusatz zu entwerfen. Aber wie sieht das denn aus, wenn ein Anschreiben mit Herr «Müller II» erstellt wird?

```
001 SELECT
002    "Nachname"||', '||"Vorname"||' - ID:'||"ID",
003    "ID"
004 FROM "Tabelle1"
005 ORDER BY "Nachname"||', '||"Vorname"||"ID";
```

Hier wird auf jeden Fall jeder Datensatz unterscheidbar. Angezeigt wird «Nachname, Vorname – ID:IDWert».

Falls es einmal passieren sollte, dass Felder für die Vornamen leer (**NULL**) sind, so wird so ein Datensatz in den Listenfeldern natürlich erst einmal nicht angezeigt. Leere Felder, verbunden mit anderen Feldern, sind ebenfalls **NULL**. Hier hilft dann eine entsprechend eingefügte Bedingung:

```
001 SELECT
002    "Nachname"||COALESCE(', '||"Vorname",''),
003    "ID"
004 FROM "Tabelle1"
005 ORDER BY "Nachname"||COALESCE(', '||"Vorname",'');
```

Damit wird das Komma und der anschließende Vorname durch einen leeren Text ersetzt, wenn das Feld "Vorname" **NULL** ist.

Im Formular Ausleihe wurde ein Listenfeld erstellt, das lediglich die Medien darstellen sollte, die noch nicht entliehen wurden. Dies wird über die folgende SQL-Formulierung möglich:

```
001 SELECT
002    "Titel" || ' - Nr. ' || "ID",
003    "ID"
```

```
004 FROM "Medien"
005 WHERE "ID"
006    NOT IN
007       (SELECT
008          "Medien_ID"
009       FROM  "Ausleihe"
010       WHERE "Rueck_Datum" IS NULL)
011 ORDER BY "Titel" || ' - Nr. ' || "ID" ASC
```

Dieses Listenfeld muss allerdings immer dann aktualisiert werden, wenn eine Ausleihe eines darin verzeichneten Mediums erfolgt ist.

Das folgende Listenfeld stellt die Inhalte mehrerer Felder in Tabellenform dar, so dass zusammengehörige Elemente direkt untereinander erscheinen:

Damit so eine Darstellung gelingt, muss zuerst einmal eine entsprechende nicht proportionale Schrift ausgewählt werden. Courier oder alle Mono-Schriftarten wie z.B. Liberation Mono können hier genutzt werden. Die Darstellung in tabellarischer Form gelingt über den SQL-Code (HSQLDB)

```
001 SELECT
002    LEFT("Ware"||SPACE(25),25) || ' - ' ||
          RIGHT(SPACE(8)||"Preis",8) || ' €',
003    "ID"
004 FROM "Waren"
005 ORDER BY ("Ware" || ' - ' || "Preis" || ' €') ASC
```

FIREBIRD kennt die Funktion **SPACE** nicht, dafür aber Funktionen wie **LPAD** und **RPAD**:
LEFT("Ware"||SPACE(25),25) wird ersetzt durch **RPAD("Ware",25)**
RIGHT(SPACE(8)||"Preis",8) wird ersetzt durch **LPAD("Preis",8)**
An den Inhalt des Feldes "Ware" werden entsprechend viele Leerzeichen angehängt, so dass "Ware" zusammen mit den Leerzeichen eine Mindestlänge von 25 Zeichen hat. Anschließend werden die ersten 25 Buchstaben dargestellt, die überflüssigen Leerzeichen also abgeschnitten.

Komplizierter wird es, wenn in dem Inhalt des Listenfeldes auch nicht druckbare Zeichen wie z.B. Zeilenumbrüche enthalten sind. Dann muss der Code entsprechend angepasst werden:

```
001 SELECT
002    LEFT(REPLACE("Ware",CHAR(10),' ')||SPACE(25), 25) || ' - ' || ...
```

Damit wird ein Zeilenvorschub in Linux durch ein Leerzeichen ersetzt. In Windows muss zusätzlich noch der Wagenrücklauf (**CHAR(13)**) entfernt werden.
FIREBIRD: Die Funktion **CHAR()** heißt dort **ASCII_Char()**.

Die Anzahl der notwendigen Leerzeichen kann übrigens auch per Abfrage ermittelt werden. So wird vermieden, dass doch einmal ein Wert aus "Ware" in seiner Länge beschnitten wird.

```
001 SELECT
002    LEFT("Ware"||SPACE(
003       (SELECT MAX(CHAR_LENGTH("Ware")) FROM "Waren")),
004       (SELECT MAX(CHAR_LENGTH("Ware")) FROM "Waren"))
```

```
005       || ' - ' || RIGHT('    '||"Preis",8) || ' €',
006    "ID"
007 FROM "Waren"
008 ORDER BY ("Ware" || ' - ' || "Preis" || ' €') ASC
```

Da der Preis rechtsbündig dargestellt werden soll, wird hier vor dem Preis mit Leerzeichen aufgefüllt und entsprechend maximal 8 Zeichen von rechts aus dargestellt. Die angestrebte Darstellung reicht also für alle Preise bis 99999,99 € aus.

Soll auch noch der Punkt aus der SQL-Darstellung durch ein Komma ersetzt werden, so ist der SQL-Code entsprechend zu ergänzen. Dies wäre über

```
005 REPLACE(RIGHT('    '||"Preis",8),'.',',')
```

möglich.

Abfragen als Grundlage von Zusatzinformationen in Formularen

Will man zusätzliche Informationen im Formular im Auge haben, die so gar nicht sichtbar wären, so bieten sich verschiedene Abfragemöglichkeiten an. Die einfachste ist, mit gesonderten Abfragen diese Informationen zu ermitteln und in andere Formulare einzustellen. Der Nachteil dieser Variante kann sein, dass sich Datenänderungen auf das Abfrageergebnis auswirken würden, aber leider die Änderungen nicht automatisch angezeigt werden.

Hier ein Beispiel aus dem Bereich der Warenwirtschaft für eine einfache Kasse:

Die Tabelle für eine Kasse enthält Anzahl und Fremdschlüssel von Waren, außerdem eine Rechnungsnummer. Besonders wenig Informationen erhält der Nutzer an der Supermarktkasse, wenn keine zusätzlichen Abfrageergebnisse auf den Kassenzettel gedruckt werden. Schließlich werden die Waren nur über den Barcode eingelesen. Ohne Abfrage ist im Formular nur zusehen:

Anzahl	***Barcode***
3	17
2	24

usw.

Was sich hinter den Nummern versteckt, kann nicht mit einem Listenfeld sichtbar gemacht werden, da ja der Fremdschlüssel über den Barcode direkt eingegeben wird. So lässt sich auch nicht über das Listenfeld neben der Ware zumindest der Einzelpreis ersehen.

Hier hilft eine Abfrage:

```
001 SELECT
002    "Kasse"."RechnungID",
003    "Kasse"."Anzahl",
004    "Ware"."Ware",
005    "Ware"."Einzelpreis",
006    "Kasse"."Anzahl"*"Ware"."Einzelpreis" AS "Gesamtpreis"
007 FROM "Kasse", "Ware"
008 WHERE "Ware"."ID" = "Kasse"."WareID";
```

Jetzt ist zumindest schon einmal nach der Eingabe die Information da, wie viel denn für 3 * Ware'17' zu bezahlen ist. Außerdem werden nur die Informationen durch das Formular zu filtern sein, die mit der entsprechenden "RechnungID" zusammenhängen. Was auf jeden Fall noch fehlt, ist das, was denn der Kunde nun bezahlen muss:

```
001 SELECT
002    "Kasse"."RechnungID",
003    SUM("Kasse"."Anzahl"*"Ware"."Einzelpreis") AS "Summe"
```

```
004 FROM "Kasse", "Ware"
005 WHERE "Ware"."ID" = "Kasse"."WareID"
006 GROUP BY "Kasse"."RechnungID";
```

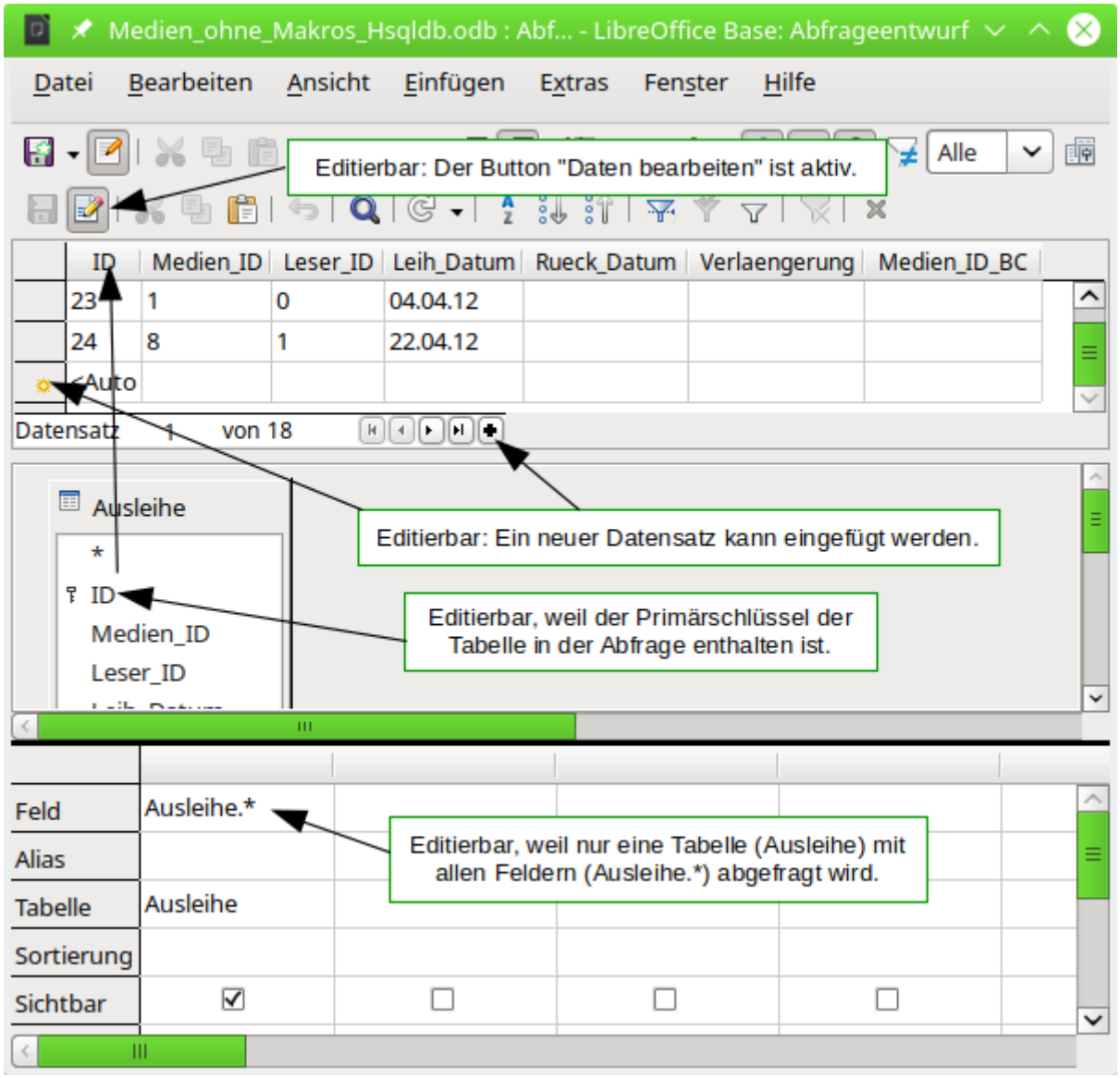
Für das Formular liefert diese Abfrage nur eine Zahl, da über das Formular natürlich wieder die "RechnungID" gefiltert wird, so dass ein Kunde nicht gleichzeitig sieht, was andere vor ihm eventuell gezahlt haben.

Tipp

Sollen in einem Formular Datumswerte dargestellt werden, die in Abhängigkeit von einem anderen Datum definiert sind (z.B.: Eine Entleihzeit für ein Medium beträgt 21 Tage – wann muss das Medium zurückgegeben werden?), dann ist dies mit Standardfunktionen der HSQLDB nicht zu machen. Es fehlt eine Funktion wie **DATEADD**.
Die Abfrage
SELECT "Datum", DATEDIFF('dd','1899-12-30',"Datum")+21 AS "RueckDatum" FROM "Tabelle"
würde in einem Formular als Datum formatiert das korrekte anvisierte Rückgabedatum ergeben. Mit dieser Abfrage werden die Tage ab dem 30.12.1899 gezählt. Der 30.12.1899 ist das Standarddatum, das z.B. auch Calc als 0-Wert nutzt.
Allerdings handelt es sich bei dem ermittelten Wert um eine Zahl, nicht um ein Datum, das anschließend in z.B. einer Abfrage weiter genutzt werden kann.
In einer Abfrage lässt sich die ermittelte Zahl schlecht nutzen, da die Formatierung von Abfragen nicht gespeichert wird. Dafür müsste eine Ansicht erstellt werden.
Achtung! Base selbst zeigt zumindest bis LO 7.4 Datumswerte, die auf Integer-Zahlen beruhen, unterschiedlich an. In Tabellen, Abfragen und formatierbaren Feldern des Formulars wird der 30.12.1899 als 0-Wert genommen. Datumsfelder hingegen berechnen aus 0 das Datum 1.1.1900.
FIREBIRD kennt die Funktion **DATEADD** und kann sogar direkt zu einem Datumswert eine Ganzzahl addieren, die dann als Tag gedeutet wird. Für FIREBIRD sähe die obige Abfrage entsprechend deutlich einfacher aus:
SELECT "Datum", "Datum"+21 AS "RueckDatum" FROM "Tabelle"

Eingabemöglichkeit in Abfragen

Um Eingaben in Abfragen zu tätigen, muss der Primärschlüssel für die jeweils zugrundeliegende Tabelle in der Abfrage enthalten sein.



Hinweis

Ist in einer Tabelle eine Spalte ausgeblendet worden, so erscheint sie nicht als Spaltenkopf bei den Daten. Dies kann nur umgangen werden, indem die Spalte wieder angezeigt wird oder die Abfrage in direktem SQL ausgeführt wird. Durch die Ausführung in direktem SQL wird eine Abfrage aber grundsätzlich nicht für Dateneingaben editierbar.

Bei der Ausleihe von Medien ist es z. B. nicht sinnvoll, für einen Leser auch die Medien noch anzeigen zu lassen, die längst zurückgegeben wurden.

```
001 SELECT
002    "ID",
003    "LeserID",
004    "MedienID",
005    "Ausleihdatum"
```

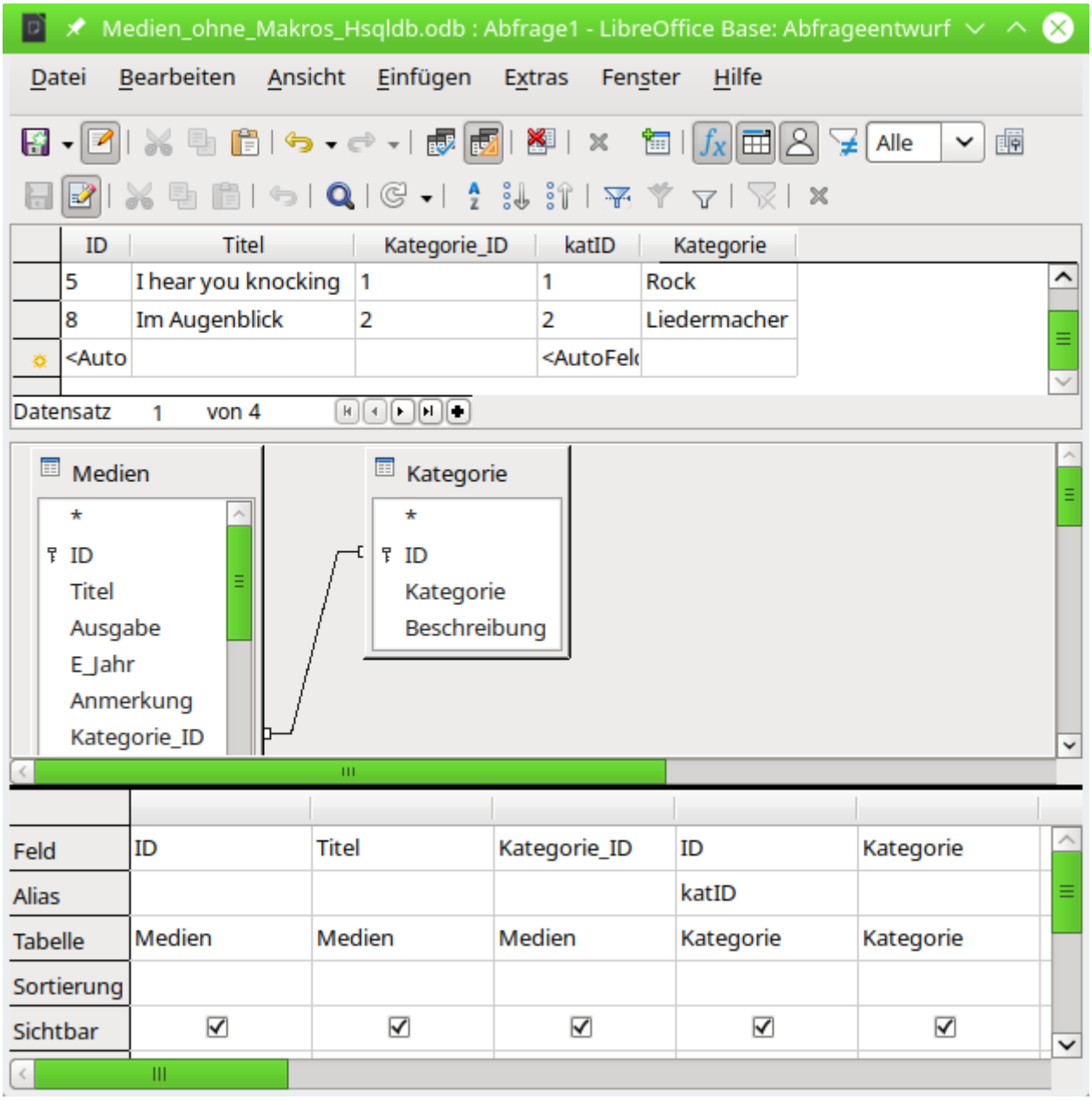
```
006 FROM "Ausleihe"
007 WHERE "Rückgabedatum" IS NULL;
```

So lässt sich im Formular innerhalb eines Tabellenkontrollfeldes all das anzeigen, was ein bestimmter Leser zur Zeit entliehen hat. Auch hier ist die Abfrage über entsprechende Formularkonstruktion (Leser im Hauptformular, Abfrage im Unterformular) zu filtern, so dass nur die tatsächlich entliehenen Medien angezeigt werden. Die Abfrage ist zur Eingabe geeignet, da **der Primärschlüssel in der Abfrage enthalten** ist.

Die Abfrage wird dann nicht mehr editierbar, wenn sie aus mehreren Tabellen besteht und die Tabellen über einen Alias-Namen angesprochen werden. Dabei ist es unerheblich, ob die Primärschlüssel in der Abfrage enthalten sind.

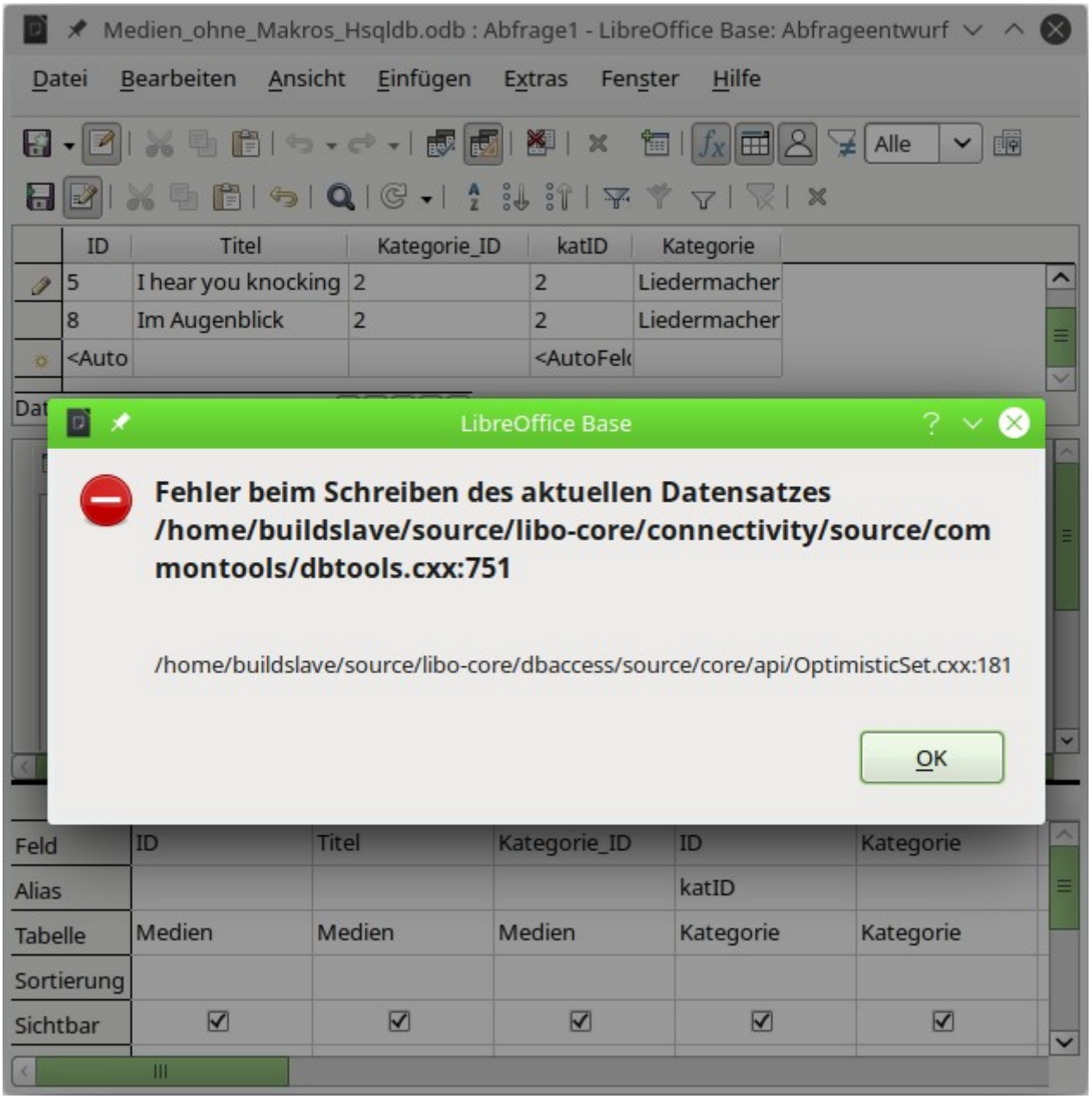
Hinweis

Sind in einer Abfrage gleiche Felder einer Tabelle mehrmals hintereinander vorhanden (z.B. einmal als "Tabelle"."Name" und einmal als "Tabelle".*), so nimmt die GUI eine Veränderung des Inhaltes nur in dem zuletzt erscheinenden Feld wahr. Es sollte daher grundsätzlich vermieden werden, das gleiche Feld mehrmals unverändert in eine Abfrage aufzunehmen. Übersicht und Editierbarkeit leiden darunter.




```
001 SELECT
002    "Medien"."ID",
003    "Medien"."Titel",
004    "Medien"."Kategorie_ID",
005    "Kategorie"."ID" AS "katID"
006    "Kategorie"."Kategorie",
007 FROM "Medien",
008    "Kategorie"
009 WHERE "Medien"."Kategorie_ID" = "Kategorie"."ID";
```

Diese Abfrage bleibt editierbar, da **beide Primärschlüssel enthalten** sind und auf die Tabellen nicht mit einem Alias zugegriffen wird. Auch bei einer Abfrage, die mehrere Tabellen miteinander verknüpft, müssen alle Primärschlüssel vorhanden sein. Außerdem muss eine Verbindung zwischen einem Feld der einen Tabelle und dem Primärschlüssel der anderen Tabelle definiert werden. Dies ist unabhängig davon, ob unter **Extras → Beziehungen** solch eine Verbindung (als Fremdschlüssel zu Primärschlüssel) bereits existiert.



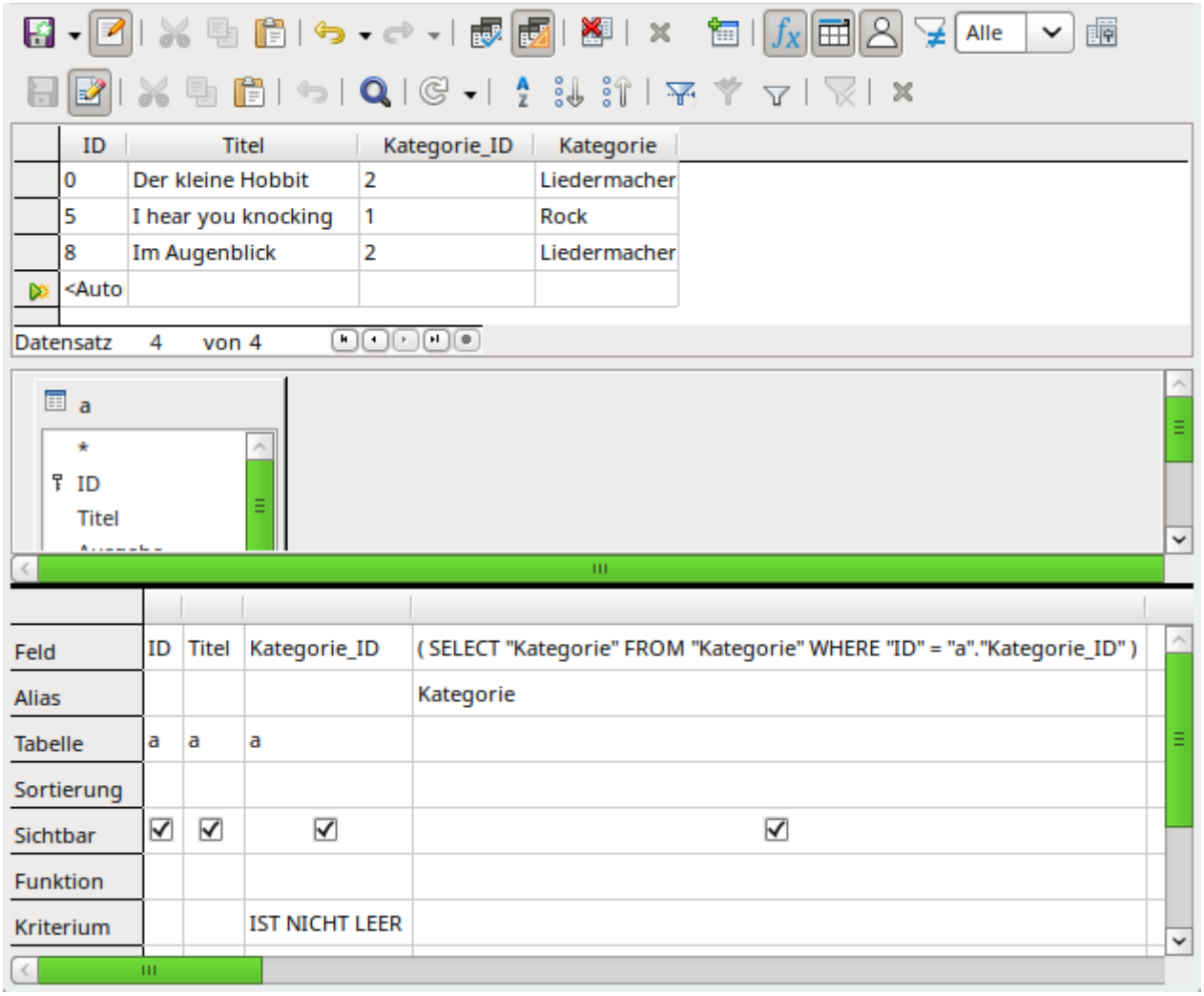
Es ist allerdings nicht möglich, in einer Abfrage, die auf mehreren Tabellen beruht, das Fremdschlüsselfeld der einen Tabelle, das sich auf die andere Tabelle bezieht, zu ändern. Im angezeigten Datensatz wurde versucht, die Kategorie für den Titel 'I hear you knocking' umzustellen. Das Feld "Kategorie_ID" wurde von '1' auf '2' geändert. Die Änderung schien vollzogen, die neue Kategorie erschien auch. Das Speichern war dann aber nicht möglich. Lediglich der erste Satz der Fehlermeldung ist hier aber für den Normalnutzer brauchbar.

Allerdings ist es möglich, den Inhalt von "Kategorie" selbst zu bearbeiten, also z.B. 'Fantasy' zum Begriff 'Fantasie' zu ändern. Das ändert dann allerdings diesen Begriff für die Kategorie, betrifft also alle damit verbundenen Datensätze.

```
001 SELECT
002    "a"."ID",
003    "a"."Titel",
004    "Kategorie"."Kategorie",
005    "Kategorie"."ID" AS "katID"
006 FROM "Medien" AS "a",
007    "Kategorie"
008 WHERE "a"."Kategorie_ID" = "Kategorie"."ID";
```

In dieser Abfrage wird auf die Tabelle "Medien" mit einem **Alias** zugegriffen. Sie wird jetzt **nicht editierbar** sein, weil das "a" für "Medien" auch in Verbindung mit Feldnamen auftaucht ("a"."ID" sowie "a"."Titel".

In dem obigen Beispiel ist das leicht vermeidbar. Wenn aber eine *Korrelierte Unterabfrage* erstellt wird, so muss mit einem Tabellenalias gearbeitet werden. So eine **Abfrage mit Alias** kann nur **dann editierbar** bleiben, wenn sie **lediglich eine Tabelle in der Hauptabfrage** enthält.



In der Design-Ansicht erscheint hier lediglich eine Tabelle. Die Tabelle "Medien" ist mit einem Alias versehen, damit auf die Inhalte des Feldes "Kategorie_ID" mit der korrelierenden Unterabfrage zugegriffen werden kann.

In so einer Abfrage ist es jetzt möglich, das Fremdschlüsselfeld "Kategorie_ID" auf eine andere Kategorie umzustellen. In dem obigen Beispiel wurde das Feld "Kategorie_ID" von '0' auf '2' geändert. Dem "Titel" 'Der kleine Hobbit' wurde so statt der "Kategorie" 'Fantasy' die "Kategorie" 'Liedermacher' zugeordnet.

Allerdings ist es jetzt nicht mehr möglich, einen Wert in dem Feld zu ändern, das seinen Inhalt über die korrelierende Unterabfrage erhält. Die Änderung der "Kategorie" von 'Fantasy' in 'Fantasie' wird erst einmal angezeigt. Die Änderung wird allerdings nicht registriert und ist auch nicht abspeicherbar. In der Tabelle, die die Entwurfsansicht anzeigt, ist das Feld "Kategorie" schließlich nicht enthalten.

Abfragen bei externen Datenbanken (z.B. MySQL/MariaDB) über mehrere Tabellen sind manchmal deswegen nicht editierbar, weil die GUI sie bereits beim Erstellen mit einem Alias versieht: "Datenbankname"."Tabellenname" AS "Tabellenname". Hier kann einfach nachgebessert werden, indem «"Datenbankname"."Tabellenname" AS» entfernt wird. Es reichen in der Regel die Tabellenbezeichnungen.

Verwendung von Parametern in Abfragen

Wird viel mit gleichen Abfragen, aber eventuell unterschiedlichen Werten als Voraussetzung zu diesen Abfragen gearbeitet, so sind Parameterabfragen möglich. Vom Prinzip her funktionieren Parameterabfragen erst einmal genauso wie Abfragen, die in Unterformularen abgelegt werden:

```
001 SELECT
002    "ID",
003    "Leser_ID",
004    "Medien_ID",
005    "Ausleihdatum"
006 FROM "Ausleihe"
007 WHERE "Rückgabedatum" IS NULL
008    AND "Leser_ID"=2;
```

Diese Abfrage zeigt nur die entliehenen Medien des Lesers mit der Nummer '2' an.

```
001 SELECT
002    "ID",
003    "Leser_ID",
004    "Medien_ID",
005    "Ausleihdatum"
006 FROM "Ausleihe"
007 WHERE "Rückgabedatum" IS NULL
008    AND "LeserID" = :Lesernummer;
```

Jetzt erscheint beim Aufrufen der Abfrage ein Eingabefeld. Es fordert zur Eingabe einer Lesernummer auf. Wird hier ein Wert eingegeben, so werden die zur Zeit entliehenen Medien dieses Lesers angezeigt.

```
001 SELECT
002    "Ausleihe"."ID",
003    "Leser"."Nachname"||', '||"Leser"."Vorname",
004    "Ausleihe"."Medien_ID",
005    "Ausleihe"."Ausleihdatum"
006 FROM "Ausleihe", "Leser"
```

```
007 WHERE "Ausleihe"."Rückgabedatum" IS NULL
008    AND "Leser"."ID" = "Ausleihe"."Leser_ID"
009    AND "Leser"."Nachname" LIKE '%' || :Lesername || '%'
010 ORDER BY "Leser"."Nachname"||', '||"Leser"."Vorname" ASC;
```

Diese Abfrage ist noch deutlich komfortabler als die vorhergehende. Jetzt muss nicht eine Nummer des Lesers bekannt sein. Es reicht die Eingabe eines Teils des Nachnamen des Lesers und alle Medien von Lesern, auf die diese Beschreibung zutrifft, werden angezeigt.

Wird

```
009    AND "Leser"."Nachname" LIKE '%' || :Lesername || '%'
```

durch

```
009    AND LOWER("Leser"."Nachname") LIKE '%' || LOWER( :Lesername ) || '%'
```

ersetzt, so ist es auch noch egal, ob die Namen groß oder klein geschrieben sind.

Wird der **Parameter** in der obigen Abfrage **leer** gelassen, so werden bis LO 4.4 **alle Leser** angezeigt, da ein leeres Parameterfeld erst ab der Version LO 4.4 auch tatsächlich nicht als leerer Text, sondern als **NULL** weiter gegeben wird. Soll dies vermieden werden, so muss etwas in die Trickkiste gegriffen werden:

```
009    AND LOWER ("Leser"."Nachname") LIKE '%' ||
          IFNULL( NULLIF ( LOWER (:Lesername), '' ), '§§' ) || '%'
```

Das leere Parameterfeld gibt an die Abfrage einen leeren String, aber nicht NULL weiter. Deshalb muss erst einmal dem leeren Parameterfeld die Eigenschaft NULL mit **NULLIF** zugewiesen werden. Anschließend wird für den Fall, dass eben die Parametereingabe jetzt NULL ergibt, dieser Eingabe ein Wert zugewiesen, der in der Regel in keinem der Datensätze vorkommt. In dem obigen Beispiel ist das **'§§'**. Dieser Wert wird mit der Abfrage entsprechend auch nicht gefunden.

Ab der Version **LO 4.4** muss diese Abfragetechnik etwas angepasst werden:

```
009    AND LOWER ("Leser"."Nachname") LIKE '%' || LOWER (:Lesername) || '%'
```

ergibt zwangsläufig bei einer fehlenden Eingabe für die gesamte Kombination **'%' || LOWER (:Lesername) || '%'** den Wert **NULL**.

Dagegen hilft das Hinzufügen einer weiteren Bedingung, dass bei einem leeren Feld tatsächlich alle Werte aufgezeigt werden:

```
009    AND (LOWER ("Leser"."Nachname") LIKE '%' || LOWER (:Lesername) || '%'
          OR :Lesername IS NULL)
```

Dies sollte in Klammern gesetzt werden. So wird entweder ein Name ausgesucht oder, bei leerem Feld, also **NULL** ab LO 4.4, die zweite Bedingung erfüllt.

Für die interne Firebird-Datenbank muss der zweite Eintrag des Parameters des verbundenen Feldes in den Datentyp **VARCHAR()** umgewandelt werden. Die Länge des Strings muss natürlich wie bei Felddefinitionen mit angegeben werden, hier also **VARCHAR(50)**. Dabei ist es egal, ob der Parameter innerhalb der Abfrage mit einem Text, einer Zahl oder einem Datum verglichen wird. Der Code muss folgendermaßen in Firebird aussehen:

```
009    AND (LOWER ("Leser"."Nachname") LIKE '%' || LOWER (:Lesername) || '%'
          OR CAST( :Lesername AS VARCHAR(50) ) IS NULL)
```

Ein Parameter kann bei Formularen auch von einem Hauptformular an ein Unterformular weitergegeben werden. Es kann allerdings passieren, dass Parameterabfragen in Unterformularen nicht aktualisiert werden, wenn Daten geändert oder neu eingegeben werden.

Manchmal wäre es wünschenswert, Listenfelder in Abhängigkeit von Einstellungen des Hauptformulars zu ändern. So könnte beispielsweise ausgeschlossen werden, dass in einer Bibliothek Medien an Personen entliehen werden, die zur Zeit keine Medien ausleihen dürfen. Leider ist aber eine derartige Listenfelderstellung in Abhängigkeit von der Person über Parameter nicht möglich.

Tipp

Parameterabfragen sind auch eine Möglichkeit, für Berichte Daten vor dem Start des Berichtes zu filtern. Daneben ermöglichen solche Abfragen auch, einem Bericht bestimmte Textbausteine mitzugeben, die sonst nicht in der Abfrage enthalten sind.
Mit
SELECT "Tabelle".*, :Ueberschrift FROM "Tabelle"
wird die Variable «Ueberschrift» abgefragt und kann dann in einem Textfeld des Berichtes über **=Ueberschrift** oder **=:Ueberschrift** ausgelesen werden.

Hinweis

Der Parameter innerhalb einer Abfrage sollte nie genau die gleiche Bezeichnung haben wie ein Feld, das in der gleichen Abfrage auftaucht. In dem Moment können Formulare und Berichte die Parameter von den Feldern nicht mehr unterscheiden. Gerade in Berichten führt dies dazu, dass eventuell die Werte des Parameters statt des Feldes mit dem gleichen Namen angezeigt werden.

Unterabfragen

Unterabfragen, die in Felder eingebaut werden, dürfen immer nur genau einen Datensatz wiedergeben. Das Feld kann schließlich auch nur einen Wert wiedergeben.

```
001 SELECT
002    "ID",
003    "Einnahme",
004    "Ausgabe",
005    ( SELECT
006       SUM( "Einnahme" ) - SUM( "Ausgabe" )
007    FROM "Kasse")
008    AS "Saldo"
009 FROM "Kasse";
```

Diese Abfrage ist eingabefähig (Primärschlüssel vorhanden). Die Unterabfrage liefert nur genau einen Wert, nämlich die Gesamtsumme. Damit lässt sich nach jeder Eingabe der Kassenstand ablesen. Dies ist noch nicht vergleichbar mit der Supermarktkasse aus *Abfragen als Grundlage von Zusatzinformationen in Formularen*. Es fehlt natürlich die Einzelberechnung aus Anzahl * Einzelpreis, aber auch die Berücksichtigung der Rechnungsnummer. Es wird immer die Gesamtsumme ausgegeben. Zumindest die Rechnungsnummer lässt sich über eine Parameterabfrage einbauen:

```
001 SELECT
002    "ID",
003    "Einnahme",
004    "Ausgabe",
005    ( SELECT
006       SUM( "Einnahme" ) - SUM( "Ausgabe" )
007    FROM "Kasse"
008    WHERE "RechnungID" = :Rechnungsnummer)
009    AS "Saldo"
010 FROM "Kasse"
011 WHERE "RechnungID" = :Rechnungsnummer;
```

Bei einer Parameterabfrage muss der Parameter in beiden Abfrageanweisungen gleich sein, wenn er als ein Parameter verstanden werden soll.

Für Unterformulare können diese Parameter mitgegeben werden. Unterformulare erhalten dann statt der Feldbezeichnung die darauf bezogene Parameterbezeichnung. Die Eingabe dieser Verknüpfung ist nur in den Eigenschaften der Unterformulare, nicht über den Assistenten möglich.

Hinweis

Unterformulare, die auf Abfragen beruhen, werden nicht in Bezug auf die Parameter automatisch aktualisiert. Es bietet sich also eher an, den Parameter direkt aus dem darüber liegenden Formular weiter zu geben.

Korrelierte Unterabfrage

Mittels einer noch verfeinerten Abfrage lässt sich innerhalb einer bearbeitbaren Abfrage sogar der laufende Kontostand mitführen:

```
001 SELECT
002    "ID",
003    "Einnahme",
004    "Ausgabe",
005    ( SELECT
006       SUM( "Einnahme" ) - SUM( "Ausgabe" )
       FROM "Kasse"
       WHERE "ID" <= "a"."ID" )
       AS "Saldo"
007 FROM "Kasse" AS "a"
008 ORDER BY "ID" ASC
```

Die Tabelle "Kasse" ist gleichbedeutend mit der Tabelle "a". "a" stellt aber nur den Bezug zu den in diesem Datensatz aktuellen Werten her. Damit ist der aktuelle Wert von "ID" aus der äußeren Abfrage innerhalb der Unterabfrage auswertbar. Auf diese Weise wird in Abhängigkeit von der "ID" der jeweilige Saldo zu dem entsprechenden Zeitpunkt ermittelt, wenn einfach davon ausgegangen wird, dass die "ID" über den Autowert selbständig hoch geschrieben wird.

Hinweis

Soll nach den Inhalten der Unterabfrage mit Hilfe der Filterfunktionen des Abfrageeditors gefiltert werden, so funktioniert dies zur Zeit nur, wenn statt der einfachen Klammern zu Beginn und Ende der Unterabfrage die Klammern doppelt gesetzt werden:
« **((SELECT ….)) AS "Saldo"** »

Abfragen als Bezugstabellen von Abfragen

In einer Abfrage soll für alle Leser, bei denen eine 3. Mahnung zu einem Medium vorliegt, ein Sperrvermerk ausgegeben werden.

```
001 SELECT
002    "Ausleihe"."Leser_ID",
003    '3 Mahnungen - der Leser ist gesperrt' AS "Sperre"
004 FROM
005    (SELECT COUNT( "Datum" ) AS "Anzahl",
006       "Ausleihe_ID"
007    FROM "Mahnung"
008    GROUP BY "Ausleihe_ID")
009    AS "a",
010    "Ausleihe"
011 WHERE "a"."Ausleihe_ID" = "Ausleihe"."ID"
```

```
012    AND "a"."Anzahl" > 2
```

Zuerst wird die **innere Abfrage** konstruiert, auf die sich die äußere Abfrage bezieht. In dieser Abfrage wird die Anzahl der Datumseinträge, gruppiert nach dem Fremdschlüssel "Ausleihe_ID", ermittelt. Dies muss unabhängig von der "Leser_ID" geschehen, da sonst nicht nur 3 Mahnungen bei einem Medium, sondern auch drei Medien mit einer ersten Mahnung zusammengezählt würden. Die innere Abfrage wird mit einem Alias versehen, damit sie mit der "Leser_ID" der äußeren Abfrage in Verbindung gesetzt werden kann.

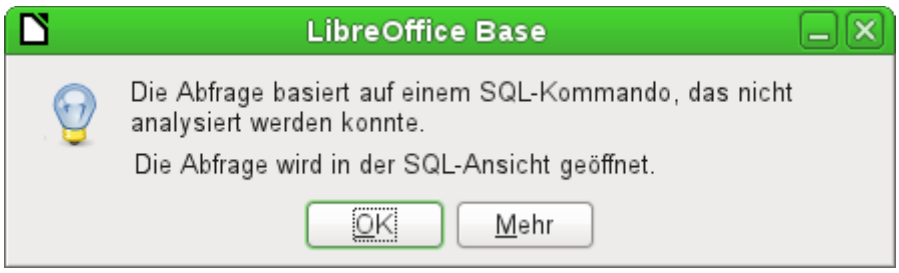
Hinweis

Innere Abfragen dürfen nicht sortiert werden. Die Sortierung erfolgt ausschließlich über die äußere Abfrage. Eine innere Abfrage mit Sortierung wird mit einer Fehlermeldung der HSQLDB quittiert:
`Cannot be in ORDER BY clause in statement [...]`

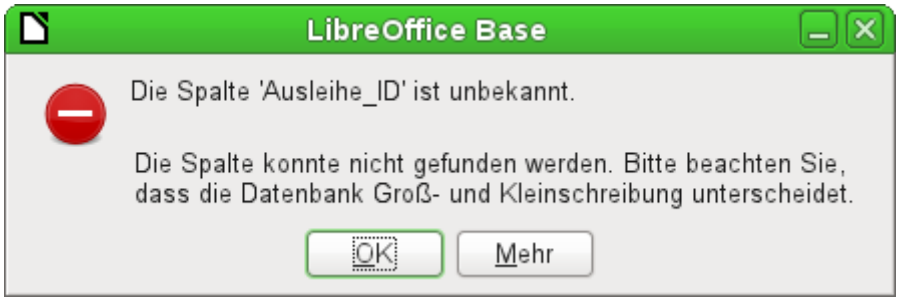
Die **äußere Abfrage** bezieht sich in diesem Fall nur in der Bedingungsformulierung auf die innere Abfrage. Sie zeigt nur dann eine "Leser_ID" und den Text zur "Sperre" an, wenn "Ausleihe"."ID" *und* "a"."Ausleihe_ID" *gleich* sind sowie "a"."Anzahl" > *2* ist.

Prinzipiell stehen der äußeren Abfrage alle Felder der inneren Abfrage zur Verfügung. So ließe sich beispielsweise die Anzahl mit "a"."Anzahl" in der äußeren Abfrage einblenden, damit auch die tatsächliche Mahnzahl erscheint.

Es kann allerdings sein, dass im Abfrageeditor die grafische Benutzeroberfläche nach so einer Konstruktion nicht mehr verfügbar ist. Soll die Abfrage anschließend wieder zum Bearbeiten geöffnet werden, so erscheint die folgende Meldung:



Ist die Abfrage dann in der SQL-Ansicht zum Bearbeiten geöffnet und wird versucht von dort in die grafische Ansicht zu wechseln, so erscheint die Fehlermeldung

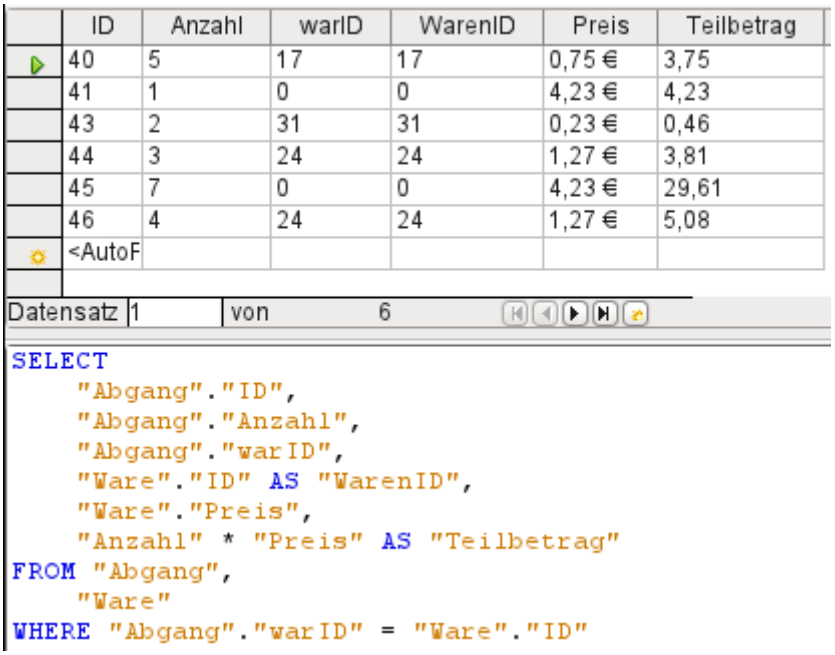


Die grafische Benutzeroberfläche findet also nicht das in der inneren Abfrage enthaltene Feld "Ausleihe_ID", mit dem die Beziehung von innerer und äußerer Abfrage geregelt wird.

Wird die Abfrage allerdings aufgerufen, so wird der entsprechende Inhalt anstandslos wiedergegeben. Es muss also *nicht der SQL-Befehl direkt ausgeführt* werden. Damit stehen auch die Sortier- und Filterfunktionen der grafischen Benutzeroberfläche weiter zur Verfügung.

Die folgenden Screenshots zeigen, wie der unterschiedliche Weg zu einem Abfrageergebnis mit Unterabfragen auch verlaufen kann. Hier soll in der Abfrage einer Rechnungsdatenbank ermittelt werden, was der Kunde an der Kasse letztlich zahlen muss. Die Preise werden multipliziert

mit der Anzahl der entsprechenden Ware zum "Teilbetrag". Außerdem soll auch noch die Summe der Teilbeträge ausgegeben werden. Und all das soll editierbar bleiben, damit die Abfrage als Grundlage für ein Formular dienen kann.



	ID	Anzahl	warID	WarenID	Preis	Teilbetrag
▶	40	5	17	17	0,75 €	3,75
	41	1	0	0	4,23 €	4,23
	43	2	31	31	0,23 €	0,46
	44	3	24	24	1,27 €	3,81
	45	7	0	0	4,23 €	29,61
	46	4	24	24	1,27 €	5,08
✱	<AutoF					

Datensatz 1 von 6

```
SELECT
    "Abgang"."ID",
    "Abgang"."Anzahl",
    "Abgang"."warID",
    "Ware"."ID" AS "WarenID",
    "Ware"."Preis",
    "Anzahl" * "Preis" AS "Teilbetrag"
FROM "Abgang",
    "Ware"
WHERE "Abgang"."warID" = "Ware"."ID"
```

Abbildung 44: Abfrage über zwei Tabellen. Damit die Abfrage editierbar bleibt, muss aus beiden Tabellen der Primärschlüssel in der Abfrage enthalten sein.

Hinweis

Aufgrund des *Bugs 61871* aktualisiert Base leider den Teilbetrag nicht automatisch.

	ID	Anzahl	warID	Preis	Teilbetrag
▶	40	5	17	0,75	3,75
	41	1	0	4,23	4,23
	43	2	31	0,23	0,46
	44	3	24	1,27	3,81
	45	7	0	4,23	29,61
	46	4	24	1,27	5,08
✱	<AutoF				

Datensatz 1 von 6

```
SELECT
    "Abgang"."ID",
    "Abgang"."Anzahl",
    "Abgang"."warID",
    "Ware"."Preis",
    "Anzahl" * "Preis" AS "Teilbetrag"
FROM "Abgang",
        ( SELECT
            "ID",
            "Preis"
        FROM "Ware" )
    AS "Ware"
WHERE "Abgang"."warID" = "Ware"."ID"
```

Abbildung 45: Die Tabelle "Ware" wurde in eine Unterabfrage verschoben. Diese Unterabfrage wird im Tabellenbereich (nach dem Begriff «FROM») erstellt und mit einem Alias versehen. Jetzt ist der Primärschlüssel aus der Tabelle "Ware" in der Abfrage nicht mehr zwingend erforderlich. Die Abfrage bleibt auch so editierbar.

	recID	Summe
▶	0	12,25
	1	34,69

Datensatz 1 von 2

```
SELECT
    "Abgang"."recID",
    SUM("Anzahl" * "Preis") AS "Summe"
FROM "Abgang", "Ware"
WHERE "Abgang"."warID" = "Ware"."ID"
GROUP BY "Abgang"."recID"
```

Abbildung 46: Jetzt soll die Rechnungssumme noch in der Abfrage erscheinen. Bereits die einfache Abfrage der Rechnungssumme ist nicht editierbar, da hier gruppiert und summiert wird.

	ID	Anzahl	warID	Preis	Teilbetrag	Summe
▶	40	5	17	0,75	3,75	12,25
	41	1	0	4,23	4,23	12,25
	43	2	31	0,23	0,46	12,25
	44	3	24	1,27	3,81	12,25
	45	7	0	4,23	29,61	34,69
	46	4	24	1,27	5,08	34,69
✱	<AutoF					

Datensatz 1 von 6

```
SELECT
    "Abgang"."ID",
    "Abgang"."Anzahl",
    "Abgang"."warID",
    "Ware"."Preis",
    "Anzahl" * "Preis" AS "Teilbetrag",
    "Rechnungsbetrag"."Summe"
FROM "Abgang",
        ( SELECT
            "ID",
            "Preis"
        FROM "Ware" )
    AS "Ware",
        ( SELECT
            "Abgang"."recID",
            SUM( "Anzahl" * "Preis" ) AS "Summe"
        FROM "Abgang",
            "Ware"
        WHERE "Abgang"."warID" = "Ware"."ID"
        GROUP BY "Abgang"."recID" )
    AS "Rechnungsbetrag"
WHERE "Abgang"."warID" = "Ware"."ID"
        AND "Abgang"."recID" = "Rechnungsbetrag"."recID"
ORDER BY "Abgang"."recID", "Abgang"."ID"
```

Abbildung 47: Mit der zweiten Unterabfrage wird das scheinbar Unmögliche möglich. Die vorhergehenden Abfrage wird als Unterabfrage in der Tabellendefinition dieser Abfrage (nach «FROM») eingefügt. Im Ergebnis bleibt die gesamte Abfrage so editierbar. Eingaben sind in diesem Fall nur in den Spalten "Anzahl" und "warID" möglich. Dies wird im Formular anschließend berücksichtigt.

Zusammenfassung von Daten mit Abfragen

Sollen Daten in der gesamten Datenbank gesucht werden, so ist dies über die einfachen Formularfunktionen meist nur mit Problemen zu bewältigen. Ein Formular greift schließlich nur auf eine Tabelle zu, und die Suchfunktion bewegt sich nur durch die Datensätze dieses Formulars.

Einfacher wird der Zugriff auf alle Daten mittels Abfragen, die wirklich alle Datensätze abbilden. Im Kapitel *Beziehungsdefinition in der Abfrage* wurde so eine Abfragekonstruktion bereits angedeutet. Diese wird im Folgenden entsprechend der Beispieldatenbank ausgebaut.

```
001 SELECT
002    "Medien"."Titel",
003    "Untertitel"."Untertitel",
004    "Verfasser"."Verfasser"
005 FROM "Medien"
006    LEFT JOIN "Untertitel"
007       ON "Medien"."ID" = "Untertitel"."Medien_ID"
```

```
008    LEFT JOIN "rel_Medien_Verfasser"
009       ON "Medien"."ID" = "rel_Medien_Verfasser"."Medien_ID"
010    LEFT JOIN "Verfasser"
011       ON "rel_Medien_Verfasser"."Verfasser_ID" = "Verfasser"."ID"
```

Hier werden alle "Titel", "Untertitel" und "Verfasser" zusammen angezeigt.

Die Tabelle "Medien" hat insgesamt 9 "Titel". Zu zwei Titeln existieren insgesamt 8 "Untertitel". Beide Tabellen zusammen angezeigt ergäben ohne einen **LEFT JOIN** lediglich 8 Datensätze. Zu jedem "Untertitel" würde der entsprechende "Titel" gesucht und damit wäre die Abfrage zu Ende. "Titel" *ohne* "Untertitel" würden nicht angezeigt.

Jetzt sollen aber alle "Medien" angezeigt werden, auch die *ohne* "Untertitel". "Medien" steht auf der linken Seite der Zuweisung, "Untertitel" auf der rechten Seite. Mit einem **LEFT JOIN** werden alle "Titel" aus "Medien" angezeigt, aber nur die "Untertitel", zu denen auch ein "Titel" existiert. "Medien" wird dadurch zu der Tabelle, die entscheidend für alle anzuzeigenden Datensätze wird. Dies ist bereits aufgrund der Tabellenkonstruktion so vorgesehen (siehe dazu das Kapitel *Tabellen Medienaufnahme*). Da zu 2 der 9 "Titel" "Untertitel" existieren, erscheinen in der Abfrage jetzt 9 + 8 – 2 = 15 Datensätze.

Hinweis

Die normale Verbindung von Tabellen erfolgt, nachdem alle Tabellen durch Komma voneinander getrennt aufgezählt wurden, nach dem Schlüsselwort **WHERE**.
Wird mit einem **LEFT JOIN** oder **RIGHT JOIN** gearbeitet, so wird die Zuweisung direkt nach den beiden Tabellennamen mit **ON** definiert. Die Reihenfolge ist also immer **Tabelle1 LEFT JOIN Tabelle2 ON Tabelle1.Feld1 = Tabelle2.Feld1 LEFT JOIN Tabelle3 ON Tabelle2.Feld1 = Tabelle3.Feld1 ...**

Zu 2 Titeln der Tabelle "Medien" existiert noch keine Verfassereingabe und kein "Untertitel". Gleichzeitig existieren bei einem "Titel" insgesamt 3 "Verfasser". Würde jetzt die Tabelle Verfasser einfach ohne **LEFT JOIN** verbunden, so würden die beiden "Medien" *ohne* "Verfasser" nicht angezeigt. Da aber ein Medium statt einem Verfasser drei Verfasser hat, würde die angezeigte Zahl an Datensätzen bei 15 Datensätzen bleiben.

Erst über die Kette von **LEFT JOIN** wird die Abfrage angewiesen, weiterhin die Tabelle "Medien" als den Maßstab für alles anzuzeigende zu nehmen. Jetzt erscheinen auch wieder die Datensätze, zu denen weder "Untertitel" noch "Verfasser" existieren, also insgesamt 17 Datensätze.

Durch entsprechende Joins wird also der angezeigte Datenbestand in der Regel größer. Durch diesen großen Datenbestand kann schließlich gesucht werden und neben den Titeln werden auch Verfasser und Untertitel erfasst. In der Beispieldatenbank können so alle von den Medien abhängigen Tabellen erfasst werden.

Hinweis

Die Abfrage-GUI ersetzt grundsätzlich **LEFT JOIN** durch **LEFT OUTER JOIN**. Sofern eine Abfrage mit der GUI und einem entsprechenden Join in der HSQLDB erstellt wurde, wird darüber hinaus der Join durch **{ oj … }** in Klammern gesetzt. ~~Firebird~~ kann Abfragen mit dieser Klammerung nicht lesen.

Schnellerer Zugriff auf Abfragen durch Tabellenansichten

Ansichten, in der SQL-Sprache **View**, sind, besonders bei externen Datenbanken, schneller als Abfragen, da sie direkt in der Datenbank verankert sind und vom Server nur das Ergebnis präsentiert wird. Abfragen werden hingegen erst einmal zum Server geschickt und dort dann verarbeitet.

Bezieht sich eine neue Abfrage auf eine andere Abfrage, so sieht das in Base in der SQL-Ansicht so aus, als ob die andere Abfrage eine Tabelle wäre. Wird daraus ein **View** erstellt, so zeigt sich, dass eigentlich mit Unterabfragen (Subselects) gearbeitet wird. Eine Abfrage 2, die sich auf eine andere Abfrage 1 bezieht, kann daher nicht über **SQL-Befehl direkt ausführen** ausgeführt werden, da nur die grafische Benutzeroberfläche, nicht aber die Datenbank selbst die Abfrage 1 kennt.

Auf Abfragen kann von der Datenbank her nicht direkt zugegriffen werden. Dies gilt auch für den Zugriff über Makros. Views hingegen können von Makros wie Tabellen angesprochen werden. Allerdings können in Views keine Datensätze geändert werden. Diesen Komfort bietet nur die Abfrage unter bestimmten Abfragebedingungen.

Tipp

Eine Abfrage, die über **SQL-Befehl direkt ausführen** gestartet wird, hat den Nachteil, dass sie über die GUI nicht mehr sortiert und gefiltert werden kann. Sie kann also nur begrenzt genutzt werden.
Eine Tabellenansicht (*View)* hingegen ist für Base handhabbar wie eine normale Tabelle – mit der Ausnahme, dass keine Änderung der Daten möglich ist. Hier stehen also trotz direktem SQL-Befehl alle Möglichkeiten zur Sortierung und zur Filterung zur Verfügung. Auch bleibt die Formatierung von Spalten in der Ansicht im Gegensatz zu Spalten in der Abfrage beim Schließen der Datenbank erhalten.
Auch zur Nutzung innerhalb von Berichten (siehe Kapitel «Berichte») funktioniert eine Tabellenansicht deutlich besser, da Funktionen und Aliasformulierungen die Interpretation des Codes innerhalb des Berichtes nicht beeinflussen.

Ansichten sind bei manchen Abfragekonstruktionen eine Lösung, um überhaupt ein Ergebnis zu erzielen. Wenn z. B. auf das Ergebnis eines Subselects zugegriffen werden soll, so lässt sich dies nur über den Umweg eines Views erledigen. Entsprechende Beispiele im Kapitel «Datenbank-Aufgaben komplett»: *Zeilennummerierung* und *Gruppieren und Zusammenfassen.*

Hinweis

Ansichten können bei der internen Firebird-Datenbank erst ab Version LO 7.3.1 in der GUI bearbeitet und geändert werden. Vorher war der SQL-Code der Ansicht nicht mehr sichtbar, nachdem die Ansicht erstellt wurde. Um den SQL-Code aller Ansichten in Firebird vor LO 7.3.1 zu erhalten muss der Abfrageeditor im SQL-Modus geöffnet und die folgende Abfrage gestartet werden:

```
001 SELECT RDB$RELATION_NAME, RDB$VIEW_SOURCE
002 FROM RDB$RELATIONS
003 WHERE RDB$VIEW_SOURCE IS NOT NULL
```

Auch die Ansichten in anderen Datenbanken können nicht nachträglich in der GUI bearbeitet werden. Der SQL-Code wird hier nur dann sichtbar, wenn der eingerichtete Nutzer auch eine Berechtigung dazu hat. In MySQL/MariaDB lautet der Code dazu folgendermaßen:

```
001 SELECT TABLE_SCHEMA, TABLE_NAME, VIEW_DEFINITION
002 FROM INFORMATION_SCHEMA.VIEWS
```

Bei PostgreSQL sind je nach verwendetem Treiber Ansichten gar nicht direkt erstellbar. Mit dem JDBC-Treiber ergibt sich folgendes Kommando, um die selbst definierten Ansichten zu erhalten:

```
001 SELECT views.table_catalog, views.table_schema, views.table_name,
    views.view_definition
002 FROM information_schema.views AS views
003 WHERE NOT views.view_definition IS NULL
```

Ansichten passen sich nicht automatisch an, wenn einer Tabelle z.B. ein zusätzliches Feld hinzugefügt wird. In einer Ansicht für eine Tabelle wird **nicht**

```
001 SELECT * FROM "Tabelle"
```

gespeichert. Dort tauchen in der HSQLDB fest die Feldbezeichnungen auf:

```
001 SELECT "ID", "Vorname", "Nachname" … FROM "Tabelle"
```

Deswegen kann auch kein Feld aus einer Tabelle entfernt werden, wenn es in einer Ansicht benötigt wird.

Hinweis

Firebird speichert den Code der Ansicht als

```
001 SELECT * FROM "Tabelle"
```

ab. Wird aber der Ursprungstabelle ein neues Feld hinzugefügt, so wird dieses Feld trotzdem nicht angezeigt. Hier muss die Ansicht zum Bearbeiten neu geöffnet und ohne ersichtliche Änderung wieder abgespeichert werden. Dann berücksichtigt die Ansicht wieder alle aktuellen Felder:
Rechte Maustaste über der Ansicht → Bearbeiten, anschließend Speichern und das Bearbeitungsfenster schließen.

Ansichten können direkt im Tabellenordner erstellt werden. Es ist auch möglich erst eine Abfrage zu erstellen und dann über die rechte Maustaste im Kontextmenü **Als Ansicht erstellen** zu wählen. Dann erscheint ein kleiner Dialog zur Benennung der Ansicht.

Enthält eine Abfrage Felder, die den gleichen Namen haben, so weigert sich die Datenbank, daraus eine Ansicht zu erstellen. Hier muss also auf jeden Fall darauf geachtet werden, dass über ein **Alias** entsprechend gleichlautende Felder in der Ansicht unterschieden werden können. Besonders bei der Kombination mehrerer Tabellen taucht dieses Problem häufig auf:

```
001 SELECT "Tabelle1".*, "Tabelle2".*
002 FROM "Tabelle1", "Tabelle2"
003 WHERE "Tabelle2"."ID" = "Tabelle1"."Tab2ID"
```

Das funktioniert als Ansicht häufig nicht, weil in beiden Tabelle der Primärschlüssel mit "ID" bezeichnet wurde. Hier werden für eine Ansicht sowieso alle Felder aufgelistet, so dass problemlos eine Aliaszuweisung erfolgen kann.

Zeitdifferenzen berechnen

Eine Tabelle hat zwei Zeitangaben: Startzeit und Zielzeit. Die Zeitdifferenz soll berechnet werden.

```
001 SELECT
002    "Startzeit",
003    "Zielzeit",
004    CAST('00:00:'||DATEDIFF('ss',"Startzeit", "Zielzeit") AS TIME)
005       AS "Zeitdifferenz"
006 FROM "Zeitmessung"
```

Hier wird mit dem Trick gearbeitet, dass die HSQLDB auch Zeitangaben umwandeln kann, die von der Textdarstellung her keine Zeitangaben mehr wären. So ergibt die erste Zeitdifferenz erst einmal über die Verbindung von

```
004    CAST('00:00:'||DATEDIFF('ss',"Startzeit", "Zielzeit") AS TIME)
```

den Text

```
004    00:00:3397
```

der anschließend anstandslos in eine korrekte Zeit von 56 Minuten und 37 Sekunden (3397/60 = 56 Rest 37) umgewandelt wird.

Startzeit	Zielzeit	Zeitdifferenz
12:13:34	13:10:11	00:56:37
12:01:23	14:08:13	02:06:50

Leider funktioniert die Darstellung der Zeit in einer Abfrage nur dann korrekt, wenn die Spalte nach Durchführung der Abfrage auf die entsprechende Formatierung umgewandelt wird. Dieses Problem stellt sich in Formularen und Berichten nicht, da dort die Formatierung dauerhaft gespeichert wird. Soll anschließend nicht mehr mit der Zeitdifferenz gerechnet werden, so kann die Zeitdifferenz in einen Text mit der korrekten Schreibweise umgewandelt werden:

```
001 SELECT
002    "Startzeit",
003    "Zielzeit",
004    TO_CHAR(
005       CAST('00:00:'||DATEDIFF('ss',"Startzeit", "Zielzeit") AS TIME),
006       'HH:MI:SS') AS "Zeitdifferenz"
007 FROM "Zeitmessung"
```

Mit der Firebird-Datenbank lässt sich dieses Problem wesentlich eleganter lösen:

```
001 SELECT
002    "Startzeit",
003    "Zielzeit",
004    "Zielzeit" - "Startzeit" AS "Zeitdifferenz"
005 FROM "Zeitmessung"
```

Firebird hat mit der Funktion DATEADD den Vorteil, dass es viele verschiedene Additions- und Subtraktionsverfahren für Datums- und Zeitwerte kennt.

Berichte

Berichte mit dem Report-Designer

Mit Hilfe von Berichten werden Daten so dargestellt, dass sie auch für Personen ohne Datenbankkenntnisse gut lesbar sind. Berichte können

- Daten tabellarisch gut lesbar darstellen,
- zu Daten Diagramme erstellen,
- mit Hilfe von Daten Etikettendruck ermöglichen,
- Serienbriefe wie z. B. Rechnungen, Mahnungen oder auch nur Bestätigungen über einen Vereinsbeitritt oder -austritt erstellen.

Um einen Bericht zu erstellen, muss die Datenbankgrundlage des Berichtes gut vorbereitet sein. Ein Bericht kann nicht, wie ein Formular, Unterberichte und damit zusätzliche Datenquellen aufnehmen. Ein Bericht kann auch nicht, wie im Formular, über Listenfelder andere Daten darstellen als in der zugrundeliegenden Datenquelle vorhanden sind.

Am besten werden Berichte mit Abfragen vorbereitet. Dort sollten alle variablen Inhalte festgeschrieben werden. Es sollte aber, wenn in dem Bericht noch sortiert werden soll, auf jeden Fall eine Abfrage erstellt werden, die das Sortieren zulässt. Dies bedeutet, dass Abfragen im direkten SQL-Modus unter diesen Bedingungen vermieden werden müssen. Muss der Datenbestand über so eine Abfrage zur Verfügung gestellt werden, so lässt sich eine Sortierung erreichen, indem aus der Abfrage eine *Tabellenansicht* erstellt wird. Diese Ansicht ist in der grafischen Benutzeroberfläche von Base immer sortierbar und filterbar.

Vorsicht

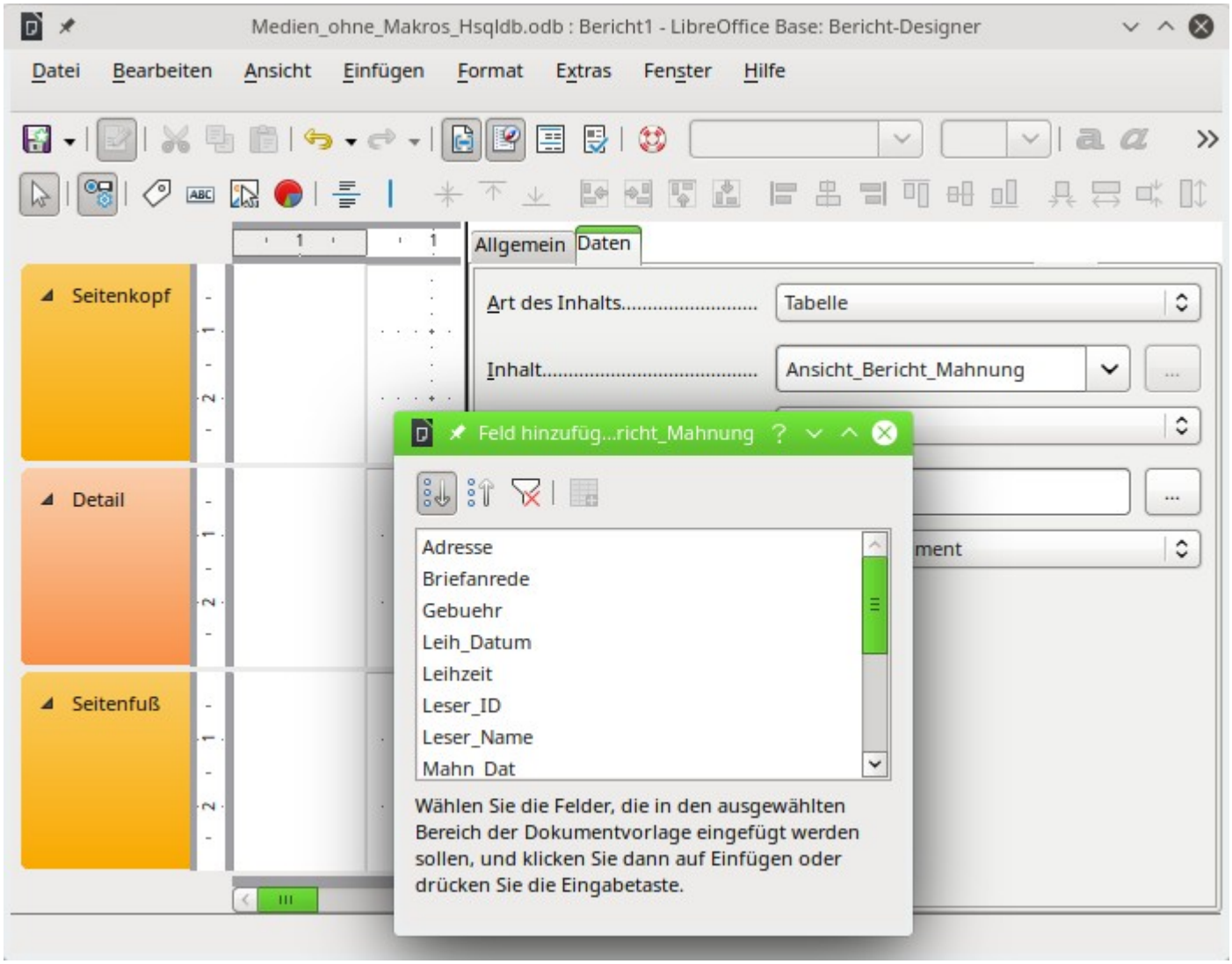
Der Report-Designer ist beim Editieren eines Berichtes mit laufendem Abspeichern zu begleiten. Dazu zählt nicht nur das Abspeichern im Report-Designer selbst nach jedem wichtigen Arbeitsschritt, sondern auch das Abspeichern der gesamten Datenbank.
Je nach Version von LibreOffice kann es beim Editieren auch zu plötzlichen Abstürzen des Report-Designers kommen.
Die Funktionsweise fertiger Berichte ist davon nicht betroffen – auch wenn diese Berichte unter einer anderen Version erstellt wurden, wo eben z. B. das oben genannte Verhalten nicht auftaucht.

Hinweis

Der Report-Designer ist seit LO 4.1 bereits so in LO integriert, dass er nicht mehr sichtbar in den Erweiterungen («Extensions») erscheint. Es sollte auf jeden Fall vermieden werden, neben dem integrierten Report-Designer noch eine nicht zu der LO-Version passende Report-Designer-Erweiterung zu installieren. Bei der Nutzung von LO-Versionen, die von Ubuntu angeboten werden, ist darauf zu achten, dass der Report-Designer auch wirklich mit installiert wird.
Neben dem Report-Designer existiert auch in LibreOffice Base weiterhin ein Assistent zur Berichtserstellung mittels eines Serienbriefverfahrens. Dieser Assistent ist allerdings nicht zugänglich, sobald der Report-Designer installiert wurde.
Der Assistent ist weitgehend selbsterklärend. Er produziert schnell übersichtliche Berichte in vorgefertigten Designs. Seine Berichte sind rein tabellarisch angelegt.
Da mit dem Report-Designer Berichte weiter ausgestaltet werden können, behandelt dieses Kapitel nur diese Berichtsvariante. Auch der Report-Designer kann über einen Assistenten gestartet werden. Hier wird allerdings die Bearbeitung in der Entwurfsansicht erläutert.
Leider tauchen im Report-Designer immer wieder ärgerliche Bugs auf. Am besten dafür einfach den Bugtracker *https://bugs.documentfoundation.org/buglist.cgi?quicksearch=ReportBuilder* aufsuchen. Siehe dazu aber auch das Kapitel *Bugs und Workarounds bei verschiedenen Datenbankverbindungen* in diesem Handbuch.

Die Benutzeroberfläche des Report-Designers

Über **Berichte → Bericht in der Entwurfsansicht erstellen …** wird der Report-Designer aufgerufen.



Der Report-Designer startet in einer dreiteiligen Ansicht. Links ist die vorläufige Einteilung des Berichts in Seitenkopf, Detail und Seitenfuß zu sehen, in der Mitte befinden sich die entsprechenden Bereiche, die Inhalte des Berichtes aufnehmen, und rechts werden die Eigenschaften des Berichtes angezeigt. Die oberen und unteren Seitenränder sind nicht sichtbar. Die Größe dieser Ränder kann über **Format → Seite** eingesehen und geändert werden.

Gleichzeitig wird bereits der Dialog **Feld hinzufügen** angezeigt. Dieser Dialog entspricht dem Dialog aus der Formularerstellung. Er erzeugt Felder mit der entsprechenden dazugehörigen Feldbezeichnung.

Ohne einen Inhalt aus der Datenbank lässt sich ein Bericht nicht sinnvoll nutzen. Deshalb wird zu Beginn direkt der Reiter **Daten** angezeigt. Hier kann der Inhalt des Berichtes eingestellt werden, im obigen Beispiel **Art des Inhalts → Tabelle** und **Inhalt → Ansicht_Bericht_Mahnung**. Solange **SQL-Befehl analysieren → Ja** eingestellt ist, kann der Bericht auch Sortierungen, Gruppierungen und Filterungen vornehmen. Da bereits als Grundlage eine Ansicht (*View*) gewählt wurde, kommt eine Filterung nicht zum Einsatz. Sie wurde bereits in der der Ansicht zugrundeliegenden Abfrage vorgenommen.

Hinweis

Als **Art des Inhaltes** kann eine Tabelle, eine Ansicht (View), eine Abfrage oder direkt SQL-Code angegeben werden. Am besten zu handhaben ist der Report-Designer, wenn er die Daten so weit wie möglich aufbereitet erhält. So können z.B. in Abfragen Berechnungen vorher durchgeführt werden und auch der Bereich der Daten, die im Bericht erscheinen sollen, begrenzt werden.
Vor allem bei früheren LO-Versionen kommt es manchmal zu Problemen, wenn Abfragen aus mehreren Tabellen später gruppiert werden sollen. Solche Probleme können vermieden werden, indem auf eine Ansicht zugegriffen wird. Eine Ansicht erscheint für Programmteile wie den Report-Designer wie eine Tabelle der Datenbank. Feldnamen sind dann unverrückbar vorgegeben. Der anschließende Zugriff mit Sortier- und Gruppierbefehlen ist fehlerfrei möglich.

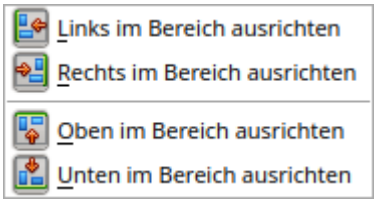
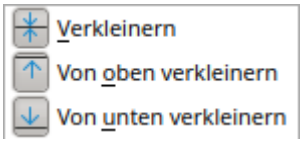
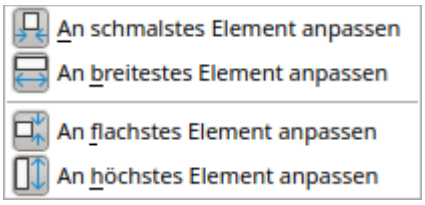
Hinweis

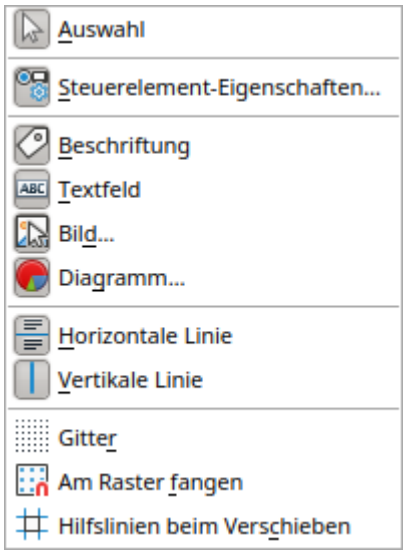
Der Report-Designer kennt nicht alle Datentypen aller möglichen Datenbanken. Enthält ein Datentyp nur Text, so sollte er deshalb durch eine Abfrage in einen bekannten Textdatentyp umgewandelt werden. Dieses Problem hat vor allem FIREBIRD mit dem Datentyp **CLOB**:

```
001  SELECT CAST( ( "CLOB-Feld" ) AS VARCHAR ( 8000 ) ) AS "Text_lang"
002  FROM "Tabelle"
```

Zwei Ausgabeformate für Berichte sind über **Datei → Ausgabeformat** wählbar: **Textdokument**, also ein Writer-Dokument oder **Tabellendokument**, also ein Calc-Dokument. Soll einfach nur eine tabellarische Übersicht ausgegeben werden, so ist das Calc-Dokument für die Berichtserstellung eindeutig vorzuziehen. Es ist wesentlich schneller erstellt und kann anschließend auch besser nachformatiert werden, da weniger Formatvorgaben berücksichtigt werden und Spalten einfach nach der erforderlichen Breite anschließend entsprechend gezogen werden können.

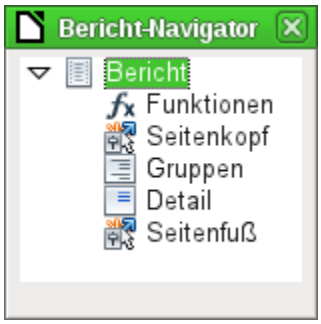
Standardmäßig sucht der Report-Designer als Datenquelle die erste Tabelle der Datenbank aus. So ist auf jeden Fall gewährleistet, dass zumindest ein Ausprobieren der Funktionen möglich ist. Erst nach Auswahl der Datenquelle kann der Bericht mit Feldern beschickt werden.

Schaltflächen inhaltliche Bearbeitung	***Schaltflächen Elementausrichtung***
	  

Schaltflächen inhaltliche Bearbeitung	Schaltflächen Elementausrichtung
	

Der Report-Designer stellt einige zusätzliche Schaltflächen zur Verfügung, so dass in der vorstehenden Tabelle noch einmal die Schaltflächen mit einer entsprechenden Beschriftung abgebildet sind. Die Schaltflächen zur Elementausrichtung werden in diesem Kapitel nicht weiter beschrieben. Sie sind hilfreich beim schnellen Anpassen von Feldern in einem Bereich des Report-Designers. Prinzipiell geht dies alles aber auch über die direkte Bearbeitung der Eigenschaften des jeweiligen Feldes. Zusätzlich gibt es noch die Symbolleiste «Zeichnungsobjekte». Diese Leiste wird standardmäßig nicht angezeigt. Die Zeichnungsobjekte sind auch über **Einfügen → Form** verfügbar.

Wie schon bei den Formularen ist es hilfreich, den entsprechenden Navigator über **Ansicht → Berichtsnavigator** oder die entsprechende Schaltfläche bei Problemen aufzurufen. So kann es zum Beispiel sein, dass durch einen unvorsichtigen Klick beim Start des Report-Designers die Eigenschaften zu den Daten des Berichts verzweifelt gesucht werden. Diese Daten können nur über den Bericht-Navigator erreicht werden:



Ein Klick mit der linken Maustaste auf **Bericht** und die Eigenschaften des Berichtes sind wieder erreichbar.

Der Navigator zeigt zu Beginn neben den sichtbaren Unterteilungen des Dokumentes (Seitenkopf, Gruppen, Detail und Seitenfuß) noch die möglichen Inhalte von Funktionen an. Gruppen ordnen z. B. alle anzumahnenden Medien einer Person zu, so dass nicht viele Einzelmahnungen erstellt werden müssen. Detailbereiche zeigen die zu den Gruppen passenden Datensätze an. Funktionen dienen z. B. zur Berechnung einer Summe einer Rechnung.

Um bei dem oben geöffneten Beispiel sinnvolle Ausgaben zu erhalten, muss der Inhalt der Ansicht gruppiert wiedergegeben werden. Ein Leser soll gebündelt die Anmahnungen für alle seine entliehenen und überzogenen Medien erhalten.

Über **Ansicht → Sortierung und Gruppierung** bzw. den entsprechenden Button startet die Gruppierungsfunktion. Hier können neben Gruppen, die angezeigt werden sollen, auch einfach Sor-

tierungen eingestellt werden. Soll nur sortiert werden, so wird zwar eine Gruppierung ausgewählt, der Gruppenkopf und der Gruppenfuß aber auf auf «Nicht vorhanden» eingestellt.

Zuerst erfolgt die Sortierung nach dem obersten eingetragenen Feld. Gegebenenfalls müssen also die Einträge entsprechend verschoben werden, wenn die gewünschte Sortierung nicht eintritt.

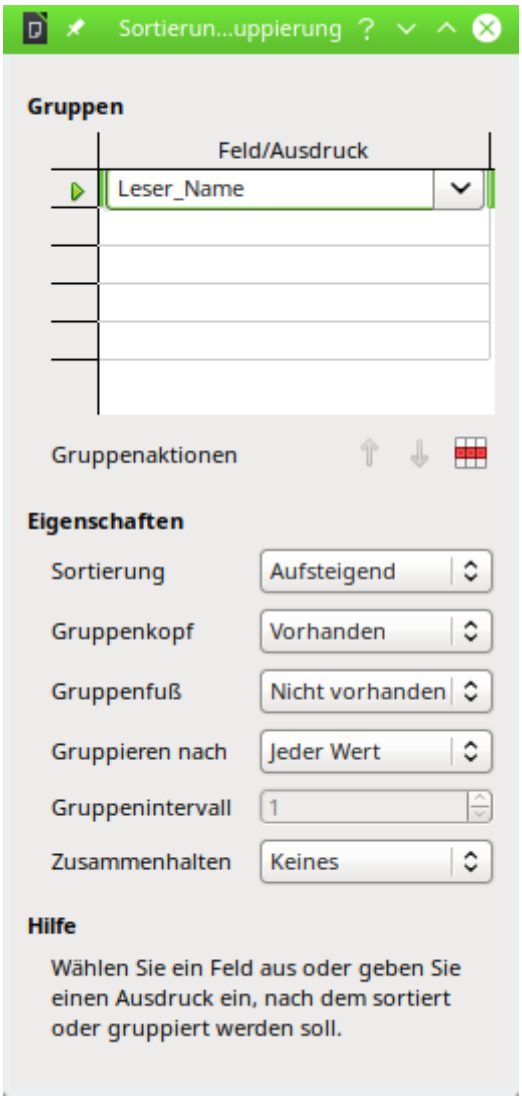


Abbildung 48: Sortierung und Gruppierung

Hier wurde nach dem Feld "Leser_Name" gruppiert und sortiert. In die obere Tabelle können untereinander mehrere Felder eingetragen werden. Soll z. B. zusätzlich nach dem "Leih_Datum" gruppiert und sortiert werden, so ist dies als zweite Zeile anzuwählen.

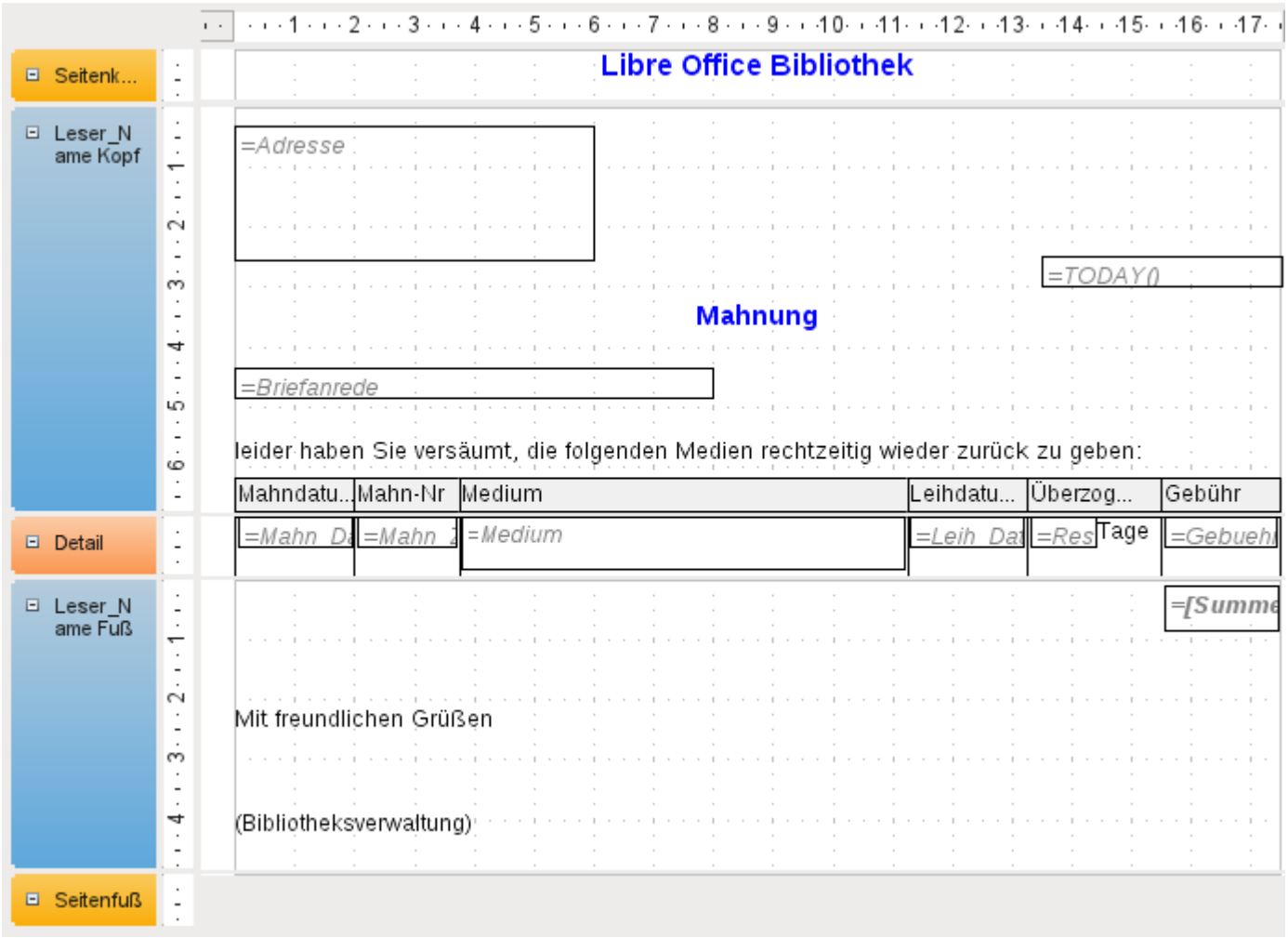
Direkt unter der Tabelle erscheinen verschiedene Gruppenaktionen zur Auswahl: Eine Verschiebung der Gruppe nach oben, eine Verschiebung nach unten oder die komplette Löschung der Gruppe. Da für den geplanten Bericht nur eine Gruppierung notwendig ist, steht in *Abbildung 48* nur mit dem Symbol ganz rechts die **Gruppenaktion → Löschung** zur Verfügung.

Die **Eigenschaft → Sortierung** ist selbsterklärend. Bei der Erstellung des Eintrags hat sich im Report-Designer auf der linken Seite sofort eine neue Einteilung gebildet. Neben der Feldbezeichnung "*Leser_Name*" steht dort noch «*Kopf*». Diese Einteilung nimmt also die Kopfzeile des Berichtes auf. In der Kopfzeile steht nachher z. B. der Name der Person, die eine Mahnung erhalten soll. Ein Gruppenfuß ist hingegen bisher nicht vorhanden. Er könnte z. B. den zu zahlenden Betrag oder den Ort und das aktuelle Datum sowie einen Bereich für die Unterschrift der anmahnenden Person enthalten.

Standardmäßig wird nach jedem Wert gruppiert. Ändert sich also "*Leser_Name*", so entsteht eine neue Gruppe. Alternativ kann hier nach dem Anfangsbuchstaben gruppiert werden. Das würde aber bei einem Mahnverfahren alle Leser-Nachnamen mit gleichem Anfangsbuchstaben zusammenfassen in einer Gruppe. 'Schmidt', 'Schulze' und 'Schulte' erhielten so eine gemeinschaftliche Mahnung. Eine wohl recht sinnlose Aktion an dieser Stelle für dieses Beispiel.

Nur wenn nach Anfangsbuchstaben gruppiert wird, kann noch zusätzlich eingegeben werden, nach wie vielen Werten die nächste Gruppe beginnen soll. Denkbar wäre hier z. B. eine Gruppierung für ein kleines Telefonbüchlein. Je nach Bekanntenkreis reicht da vielleicht eine Gruppierung nach jedem 2. Wert, also A und B in einer Gruppe, dann C und D usw.

Je nach Einstellung kann eine Gruppe entweder mit dem ersten Detail zusammen gehalten werden, oder, sofern möglich, als ganze Gruppe. Standardmäßig ist **Zusammenhalten → Keines** eingestellt. Für ein Mahnverfahren wird vermutlich sowieso die Gruppe so angeordnet, dass für jede Person, die eine Mahnung erhalten soll, eine Seite ausgedruckt wird. Daher ist stattdessen an anderer Stelle zu wählen, dass nach der Gruppe jeweils ein Seitenumbruch erfolgt, bevor der nächste Wert abzuarbeiten ist. Ein Zusammenhalten der ganzen Gruppe bewirkt gegebenenfalls, dass bei zu viel Inhalt auf der ersten Seite der gesamte Inhalt auf die Folgeseite rutscht!



Sind Gruppenkopf und gegebenenfalls Gruppenfuß ausgewählt, so erscheinen diese Elemente als Teile des Berichtnavigators unter dem entsprechenden Feldnamen "*Leser_Name*". Zusätzlich wird auch da wieder die Möglichkeit für Funktionen geboten, die sich nur auf diese Gruppe beschränken.

Das Hinzufügen der Felder läuft über **Ansicht → Feld hinzufügen** wie im Formular. Allerdings sind hier die Beschreibungen und die Inhaltsfelder nicht miteinander gruppiert. Beide können

also unabhängig voneinander verschoben, in der Größe beeinflusst und auf unterschiedliche Einteilungsebenen gezogen werden. Auch das direkte Hinzufügen von Textfeldern (für den Inhalt aus der Datenbank) und Beschriftungsfeldern ist natürlich möglich.

Das obige Bild zeigt den Berichtsentwurf für die Mahnung an. Im Seitenkopf ist fest die Überschrift «Libre Office Bibliothek» als Beschriftungsfeld eingesetzt. Hier könnte auch ein Briefkopf mit Logo stehen, da auch die Einbindung von Grafiken möglich ist. Wenn die Ebene «*Seitenkopf*» heißt, so bedeutet das nicht, dass darüber kein Rand existiert. Dieser wurde in den Seiteneinstellungen bereits festgelegt und liegt oberhalb des Seitenkopfes.

«*Leser_Name Kopf*» ist die Gruppierung und Sortierung, nach der die Daten zusammengefasst werden sollen. In den Feldern, die später Daten aufnehmen, steht hellgrau die Bezeichnung der Datenfelder, die hier ausgelesen werden. So hat die dem Bericht zugrundeliegende Ansicht z. B. ein Feld mit der Bezeichnung Adresse, in dem die komplette Adresse mit Straße und Ort für die anzumahnende Person steht. Um dies in ein Feld zu setzen, sind Absatzumbrüchen in der Abfrage notwendig. Mittels **`CHAR(13)||CHAR(10)`** (Firebird: Statt **`CHAR()`** **`ASCII_CHAR()`** verwenden) wird in einer Abfrage ein Absatz erzeugt. Beispiel:

```
001 SELECT
002 "Anrede"||CHAR(13)||CHAR(10)||"Vorname"||' '||"Nachname"||CHAR(13)||
    CHAR(10)||"Strasse"||' '||"Nr"||CHAR(13)||CHAR(10)||"Postleitzahl"||'
    '||"Ort" AS "Adresse"
003 FROM "Leser"
```

Bei dem Feld «*=TODAY()*» handelt es sich um eine eingebaute Funktion, die das aktuelle Datum an dieser Stelle einliest. Diese Funktion ist über **Einfügen → Datum und Uhrzeit** direkt zu erreichen.

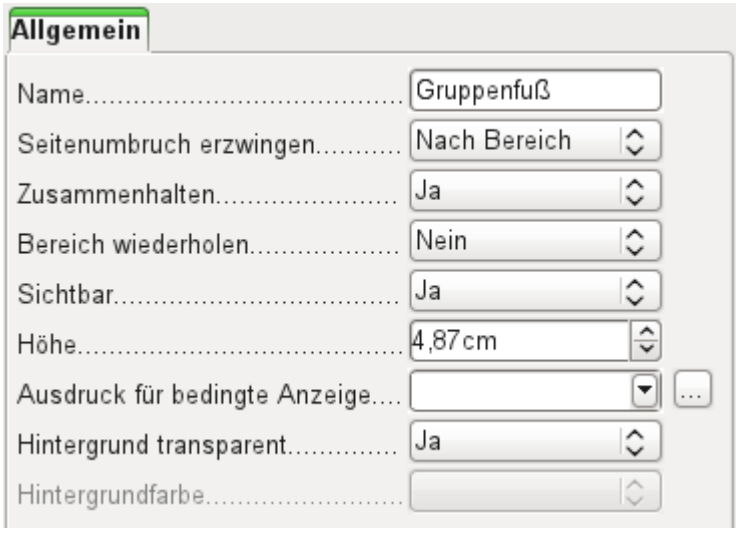
In «*Leser_Name Kopf*» sind außer der Anrede und weiteren Informationen auch die Spaltenköpfe für die anschließende Tabellenansicht untergebracht. Sie sollen ja nur einmal auftauchen, auch wenn mehrere Medien aufgelistet werden.

In den Hintergrund dieser Spaltenköpfe wurde je ein graues Rechteck gelegt. Dieses Rechteck sorgt gleichzeitig für eine entsprechende Umrandung.

Der Detailbereich wird so oft wiederholt, wie unterschiedliche Datensätze mit den gleichen Daten existieren, die in "*Leser_Name*" stehen. Hier werden also alle Medien aufgelistet, die nicht rechtzeitig zurückgegeben wurden. Auch hier liegt im Hintergrund ein Rechteck, um die Inhalte zu umranden. Das Rechteck selbst hat die Füllfarbe «weiß».

Hinweis

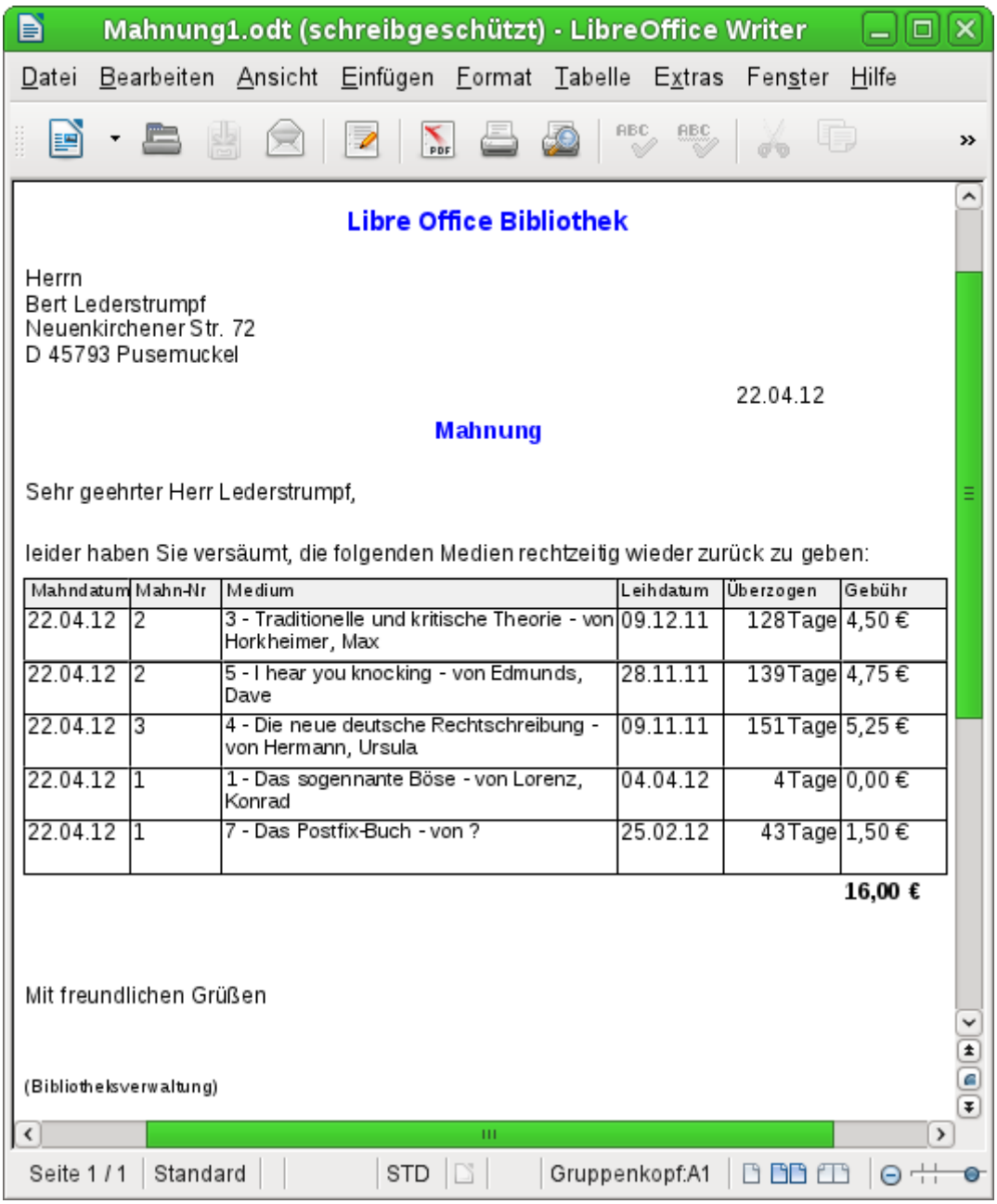
Grundsätzlich gibt es in LO auch die Möglichkeit, horizontale und vertikale Linien hinzu zu fügen. Im Design-Modus werden diese auch angezeigt.
Diese Linien haben den Nachteil, dass sie nur als Haarlinien ausgelegt sind. Sie lassen sich besser nachbilden, indem Rechtecke benutzt werden. Der Hintergrund der Rechtecke wird auf die Farbe Schwarz eingestellt, die Größe wird z. B. mit einer Breite von 17 cm und einer Höhe von 0,03 cm festgelegt. Dann erscheint eine horizontale Linie mit einer Dicke von 0,03 cm mit einer Länge von 17 cm.
Leider hat auch diese Variante einen Nachteil: grafische Elemente lassen sich nicht richtig positionieren, wenn der Bereich über eine Seite hinweg geht.



Der «*Leser_Name Fuß*» schließt schließlich das Briefformat mit einer Grußformel und einem Unterschriftsbereich ab. Der Fuß ist so definiert, dass anschließend ein Seitenumbruch nach dem Bereich erfolgt: **Seitenumbruch erzwingen → Nach Bereich**. Außerdem ist er gegenüber den Standardeinstellungen verändert worden und so eingestellt, dass der Bereich auf jeden Fall zusammen gehalten werden soll. Schließlich würde es reichlich merkwürdig aussehen, wenn bei vielen Mahnungen allein z. B. das Unterschriftsfeld auf die nächste Seite verschoben, die Gebührensumme und der Gruß aber auf der Vorseite präsentiert würde.

Hinweis

Zusammenhalten bezieht sich hier immer auf den Seitenumbruch. Soll der Inhalt eines Datensatzes unabhängig vom Seitenumbruch zusammengehalten werden, so ist dies zur Zeit nur möglich, indem der Datensatz nicht als «Detail» eingelesen wird, sondern als Grundlage für eine Gruppierung genommen wird. Der Bereich «Detail» wird leider auch dann aufgetrennt, wenn **Zusammenhalten → Ja** ausgewählt wird.



Für die Aufsummierung der Gebühren wurde eine interne Funktion benutzt. Hierfür wurde zuerst ein Textfeld ohne Inhalt aufgezogen. Anschließend wurde in den **Eigenschaften → Daten → Datenfeld-Typ → Funktion** gewählt. Anschließend wurde **Datenfeld → Gebuehr** eingestellt. Unter **Funktion** stehen Summe, Minimum und Maximum zur Verfügung. Folgerichtig wurde hier die **Summe** ausgewählt. Schließlich ist im **Geltungsbereich** noch wählbar, ob die Funktion für eine bestimmt Gruppe oder für den ganzen Bericht gelten soll. Die Summe soll bezogen auf den Leser erstellt werden. Also kam hier nur die **Gruppe: Leser_Name** in Frage. Intern weist der Report-Designer dieser Funktion den Namen *SummeGebuehrLeser_Name* zu. Aus dem Namen ist ersichtlich, dass es sich um eine Summierung der «Gebuehr» handelt. Diese Summierung wird für jede Gruppe «Leser_Name» durchgeführt.

So könnte dann eine entsprechende Mahnung aussehen. Im Detailbereich sind hier 5 Medien angegeben, die der Leser entliehen hat. Im Gruppenfuß wird die Summe für die Anmahnung ausgegeben.

Hinweis

Berichte können auch für einzelne Datensätze über mehrere Seiten gehen. Die Bereichsgröße sagt nichts über die Seitengröße aus. Allerdings kann die Ausdehnung des Bereiches Detail über mehr als eine Seite dazu führen, dass die Umbrüche nicht einwandfrei sind. Hier ist der Report-Designer in der Abstandsberechnung noch fehlerhaft. Kommen Gruppierungsbereiche und grafische Elemente hinzu, so entstehen teilweise nicht mehr durchschaubare Bereichsgrößen.

Tipp

Soll sich ein Bereich über mehrere Seiten erstrecken, so muss der Beginn des Bereiches klar auf der ersten Seite definiert sein. Um vertikale Abstände zwischen Textfeldern des Berichtes zu konstruieren, setzt der Report-Designer Tabellenzeilen ein. Passt eine Tabellenzeile nicht mehr auf eine Seite, so wird sie komplett auf die nächste Seite verschoben – mit dem gesamten folgenden Inhalt. Hier hilft es, ein kleines leeres Beschriftungsfeld genau dort zu positionieren, wo der Beginn der neuen Seite sein soll. Der Report-Designer erstellt dort dann die erste Tabellenzeile der neuen Seite. So ist die Positionierung von Textfeldern und Beschriftungsfeldern auch bei größeren Bereichen möglich.

Gliederung eines Berichtes

Die Gliederung eines Berichtes erfolgt über die Zuweisung von Gruppierungen. Zu jeder Gruppierung können Gruppenköpfe und Gruppenfüße angezeigt werden, die dann das zusammenfassende Element der Gliederung aufnehmen. Die Sortierungsreihenfolge wird über **Ansicht → Sortierung und Gruppierung** eingestellt. Sie ist im Editor dadurch sichtbar, dass die führende Sortierung durch den zuerst angezeigten Gruppenkopf dargestellt wird. Ist allerdings **Gruppenkopf → Nicht vorhanden** eingestellt, so muss für die Übersicht zur Sortierung weiter der Sortierungsdialog aufgesucht werden.

Soll ein Bericht mit einer besonderen Startseite versehen werden, z.B. dem Briefkopf einer Firma, so ist dies über **Bearbeiten → Berichtskopf/-fuß einfügen** möglich. Es ist nicht möglich, den Seitenkopf für eine einmalige Anzeige zu nutzen, da die bedingte Anzeige des Seitenkopfes daran scheitert, dass für den Seitenkopf zur Zeit keine unterschiedlichen Bedingungen definierbar sind. Wird der Berichtskopf genutzt, so sollte der Seitenkopf grundsätzlich in den Eigenschaften auf **Sichtbar → Nein** eingestellt werden, da sonst der Berichtskopf beim Ausdruck nicht an erster Position steht.

Grundsätzlich sollte bei Seitenkopf und Seitenfuß klar sein, dass es sich hier um eine Kopf- bzw. Fußzeile handelt, die zusätzlich zu den Randeinstellungen von **Format → Seite** Platz beansprucht. Über **Bearbeiten → Seitenkopf/-fuß löschen** kann der anfänglich automatische eingefügte Seitenkopf bzw. -fuß entfernt werden.

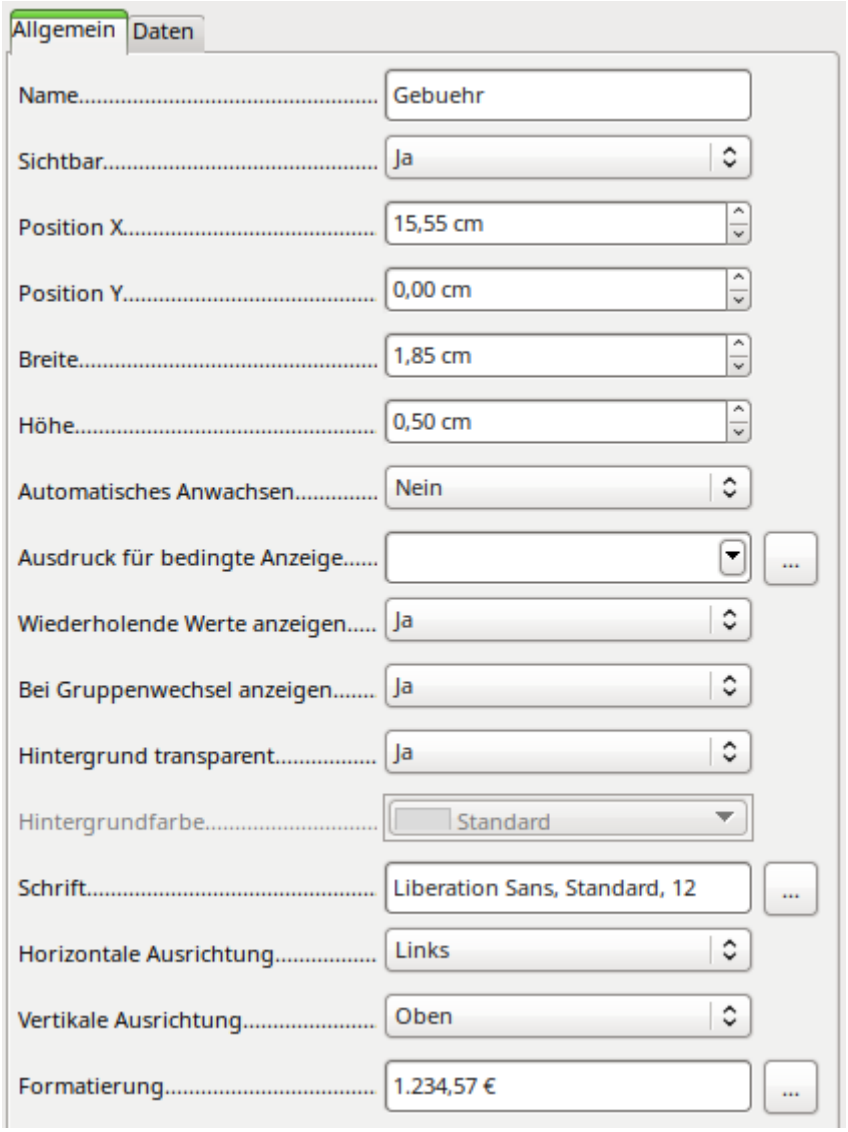
Hinweis

Die Möglichkeit, über das Menü **Bearbeiten** einen **Spaltenkopf/-fuß** einzustellen, ist beständig inaktiv. Die Funktion, einen Bericht mit Spalten zu versehen, wurde anfangs für den Report-Designer geplant, dann aber leider nicht realisiert. Wie so etwas dennoch teilweise nachgebildet werden kann, steht im Kapitel *Zweispaltige Berichte*.

Allgemeine Eigenschaften von Feldern

Zur Darstellung von Daten gibt es lediglich drei unterschiedliche Felder. Neben dem Textfeld, das im Gegensatz zu seiner Namensgebung auch Zahlen und Formatierungen beherbergen

kann, gibt es noch ein Feld, das Bilder aus der Datenbank aufnehmen kann. Das Diagrammfeld stellt eine Zusammenfassung von Daten dar.



Felder werden wie bei den Formularen mit Namen bezeichnet. Standardmäßig ist hier der Name gleich der Feldbezeichnung der zugrundeliegenden Datenquelle.

Ein Feld kann unsichtbar geschaltet werden. Bei Feldern macht dies vielleicht wenig Sinn, bei Gruppenkopf oder Gruppenfuß hingegen schon eher, da hier auch andere Funktionen der Gruppierung erfüllt sein sollen, der Gruppenkopf oder Gruppenfuß aber nicht unbedingt mit Inhalt versehen ist.

Automatisches Anwachsen ist in LO 6.4 neu hinzugekommen. Jetzt können Felder so definiert werden, dass sich die Höhe automatisch an den Inhalt anpasst, wenn mehr Inhalt vorhanden ist, als das vorgesehene Feld fassen kann. Allerdings ist diese Funktion nicht auf die einzelne Zelle sondern auf die gesamte Zeile bezogen. Bei Auswahl des Anwachsens *für das erste Feld* werden alle Felder in der Zeile automatisch auf den entsprechenden Inhalt angepasst.

In dem Report-Designer kann die Ansicht bestimmter Inhalte durch einen **Ausdruck für bedingte Anzeige** unterdrückt werden oder der Wert des Feldes als Grundlage für eine Formatierung von Schrift und Hintergrund genommen werden. Mehr zu diesen Ausdrücken unter *Bedingte Anzeige*.

Wenn **Wiederholende Werte anzeigen** deaktiviert wird, so wird die Anzeige ausgesetzt, wenn direkt davor das Feld mit einem gleichen Dateninhalt bestückt wurde. Dies funktioniert ein-

wandfrei nur bei Datenfeldern, die einen Text beinhalten. Zahlenfelder oder Datumsfelder ignorieren die Deaktivierung, Beschriftungsfelder werden bei einer Deaktivierung komplett ausgeblendet, auch wenn sie nur einmal vorkommen.

Die Funktion **Bei Gruppenwechsel anzeigen** konnte im Bericht nicht nachvollzogen werden.

Ist der Hintergrund nicht als transparent definiert, so kann für das jeweilige Feld eine Hintergrundfarbe definiert werden.

Die weiteren Einträge beziehen sich auf den Inhalt innerhalb der gezogenen Felder. Dies sind im Einzelnen die Schriftart (mit Schriftfarbe, Schriftdicke etc., siehe *Abbildung 49*), die Ausrichtung des Schriftzugs in dem Feld und die Formatierung mit dem entsprechenden Dialog **Zahlenformat** (siehe *Abbildung 50*).

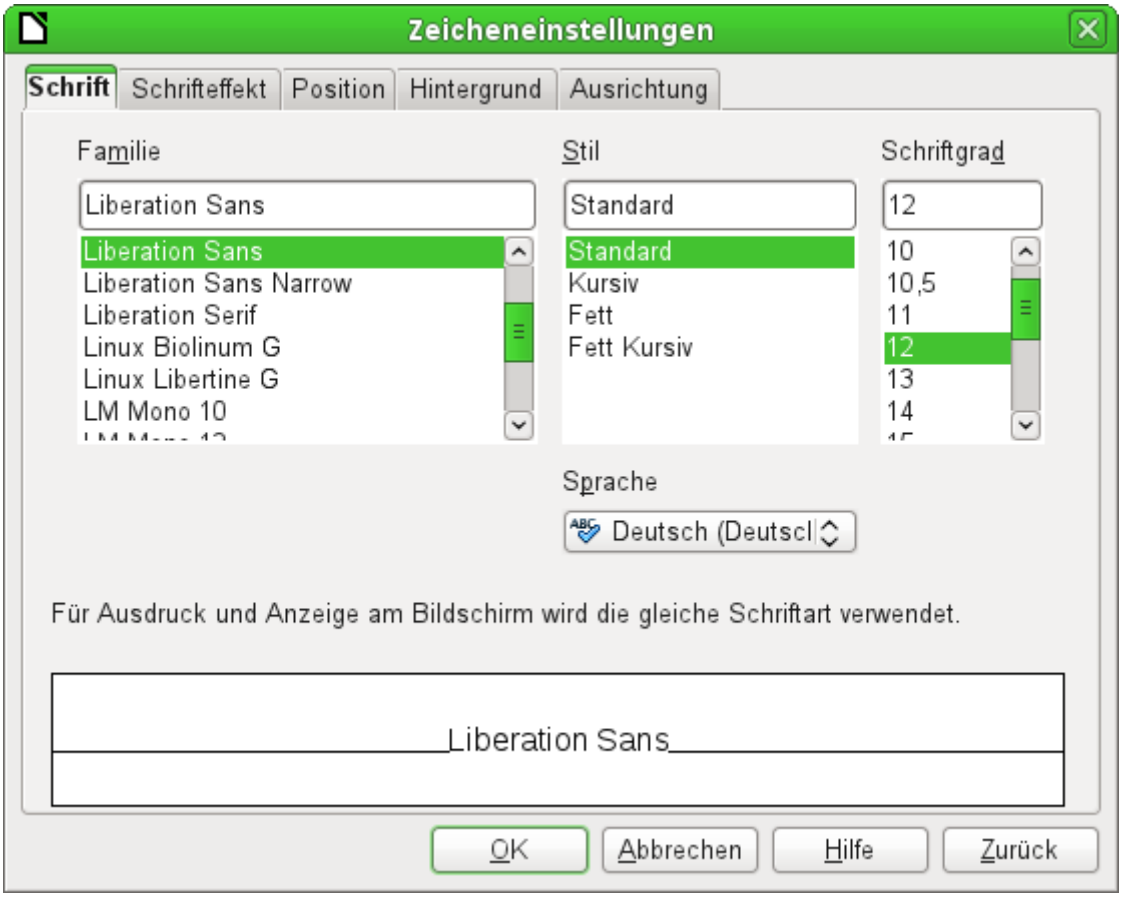


Abbildung 49: Schrift - Zeicheneinstellungen

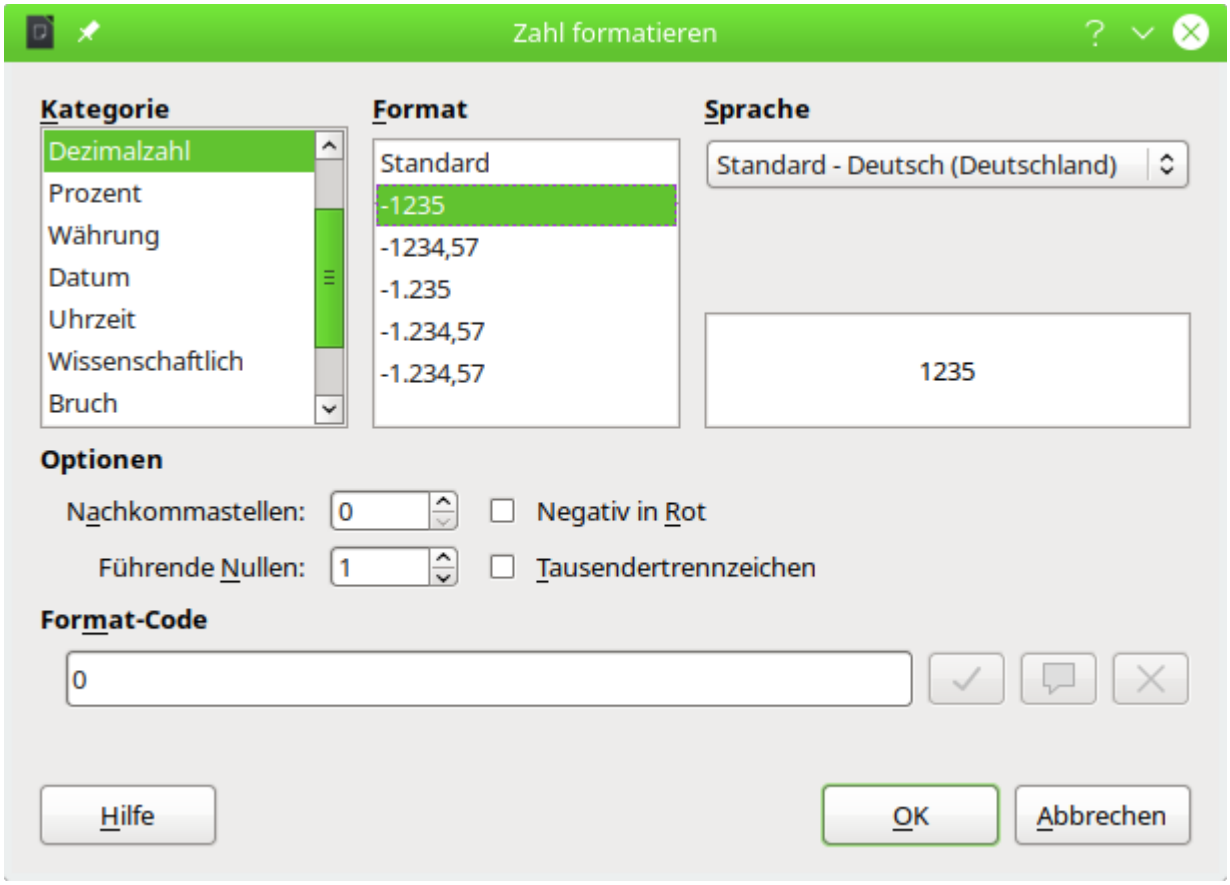


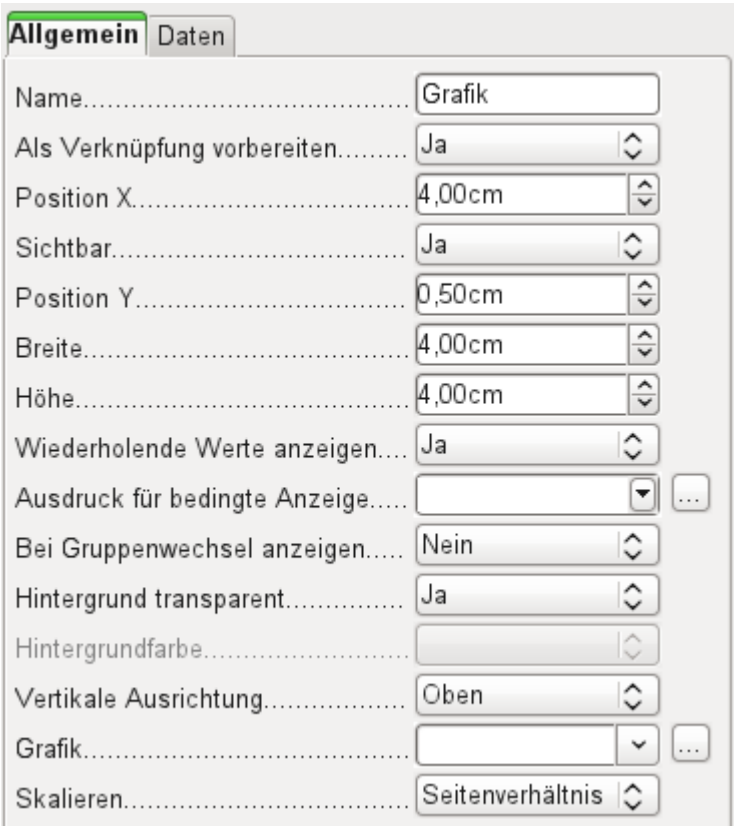
Abbildung 50: Formatierung von Zellen

Ist eine Zelle mit dem Standardformat für Dezimalzahlen belegt, so werden 0-Werte nicht angezeigt. Hier muss der **Format-Code → 0** gesetzt werden.

Tipp

Die Positionierung von Feldern im Report-Designer ist manchmal von (scheinbar) unerklärlichen Fehlermeldungen begleitet, weil ein Element das andere überschneiden würde. Ein Blick auf die eingegebenen Werte in der Maßeinheit Zentimeter widerspricht dem. Ein Blick auf einen ausgeführten Bericht, zum Bearbeiten geöffnet, zeigt allerdings, dass bei der erzeugten Tabellenstruktur etwas nicht stimmt. Teilweise werden zum Ausgleich Tabellenspalten mit 0,04 cm Breite eingesetzt und somit viel zu viele Spalten erzeugt.
Der Report-Designer arbeitet intern alle anderen Maßeinheiten in DTP-Punkte um. Ein solcher Punkt entspricht 1/72 Zoll. Dadurch lässt sich in einem Bericht nur genau positionieren, was sich in diese Punkte umrechnen lässt. Alles andere ergibt sich durch eine Rundung. Ein Bericht, testweise über **Extras → Optionen → Spracheinstellung → Gebietsschema → Englisch (USA)** mit der Maßeinheit Zoll in einer dem Punktschema entsprechenden Seitengröße erstellt, zeigt bei einer Einhaltung der Schritte von ¼ Zoll, dass daraus exakt positionierte Felder mit einer klaren Tabellenstruktur bei der Berichtsausführung werden.
Leider lässt sich beim Report-Designer bisher nicht, wie z. B. dem Writer, die Maßeinheit «Punkt» einstellen. Das würde sicher so einige Positionierungsprobleme lösen.

Besondere Eigenschaften des grafischen Kontrollfeldes



Das grafische Kontrollfeld kann sowohl Grafiken von außerhalb der Datenbank darstellen als auch Grafiken aus der Datenbank auslesen. Leider ist es zur Zeit nicht möglich, eine Grafik dauerhaft in Base zu speichern, um z. B. ein Brieflogo einzulesen. Hierzu muss zwingend die Grafik in dem gesuchten Pfad vorhanden sein, auch wenn die Auswahl anbietet, Bilder ohne Verknüpfung aufzunehmen und das erste Feld mit **Als Verknüpfung vorbereiten** auf eine entsprechende geplante Funktionalität schließen lässt.

Hinweis

Die Dateinamen dürfen keine Sonderzeichen wie **[ ] { } \ < > % "** und Leerzeichen enthalten. Ein Bericht mit diesen Bildern wird nicht erstellt.

Alternativ dazu kann natürlich eine Grafik innerhalb der Datenbank selbst gespeichert werden und so auch intern verfügbar bleiben. Sie muss dann aber über die dem Bericht zugrundeliegende Abfrage in einem der Felder zugänglich sein.

Um eine «*externe Grafik*» aufzunehmen, ist über **Grafik → Button** ... die Grafik zu laden. Um ein Datenbankfeld auszulesen, muss im Register **Daten** das Feld angegeben werden. Dieses Feld kann auch den Pfad zu einer externen Grafik beinhalten. Ist also in der Tabelle ein relativer Pfad zu einer Grafik angegeben, so wird diese Grafik entsprechend auch dargestellt.

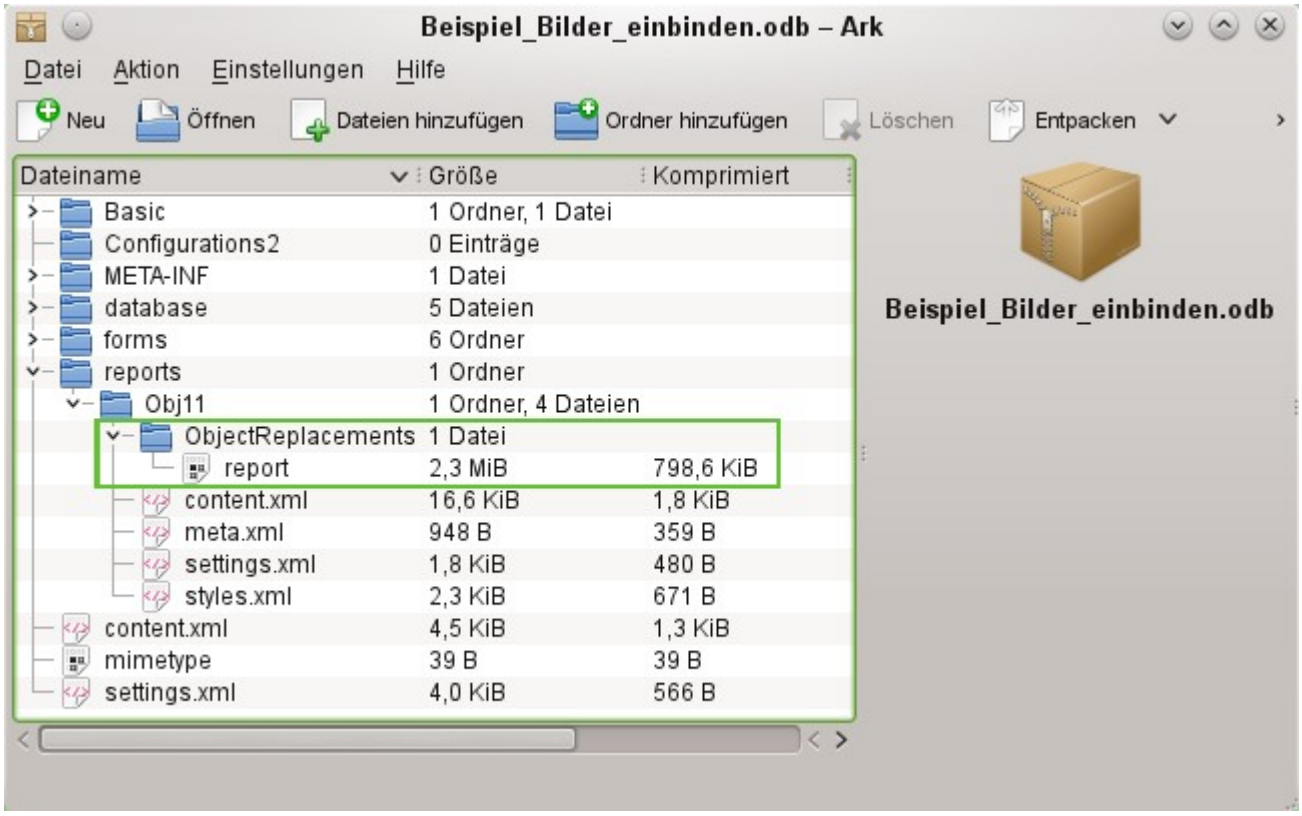
Die Einstellung der vertikalen Ausrichtung scheint im Entwurf nichts zu bewirken. Wird allerdings der Bericht aufgerufen, so erscheint die Grafik entsprechend positioniert.

Beim **Skalieren** kann 'Nein', 'Seitenverhältnis beibehalten' und 'Autom. Größe' gewählt werden. Dies entspricht den Einstellungen im Formular:

- '*Nein*': Das Bild wird nicht an das Kontrollfeld angepasst. Ist das Bild zu groß, so wird ein Ausschnitt des Bildes gezeigt. Das Bild wird nicht verzerrt.
- '*Seitenverhältnis beibehalten*': Das Bild wird in das Kontrollfeld eingepasst, aber nicht verzerrt dargestellt.

- '*Automatische Größe*': Das Bild wird der Größe des Kontrollfeldes angepasst und gegebenenfalls verzerrt dargestellt.

Nach der Bearbeitung von Berichten mit Bildern fällt auf, dass die Datenbankdatei deutlich größer wird. Innerhalb der *.odb-Datei legt Base aus nicht nachvollziehbaren Gründen im Berichtsverzeichnis einen Ordner «ObjectReplacements» an. Dieser Ordner enthält dann eine Datei «report», die für eine entsprechende Vergrößerung der *.odb-Datei sorgt.



Wenn die Datenbankdatei in einem Packprogramm geöffnet wird, ist dieser Ordner mit Inhalt im Verzeichnis «reports» im Unterverzeichnis zu dem jeweiligen Bericht sichtbar. Dieses Verzeichnis kann über das Packprogramm gefahrlos gelöscht werden.

Hinweis

Wenn Berichte nicht wiederholt editiert werden, reicht ein einmaliges Löschen des Verzeichnisses «ObjectReplacements» aus. Der Umfang des Verzeichnisses kann sehr rasch anschwellen. Dies hängt von der Menge und Größe der Dateien ab. Allein eine 2,8 MB *.jpg-Datei vergrößerte so in einem Test eine *.odb-Datei um 11 MB!
Der Bug ist hier gemeldet: *https://bugs.freedesktop.org/show_bug.cgi?id=80320*

Diagramme im Bericht einbinden

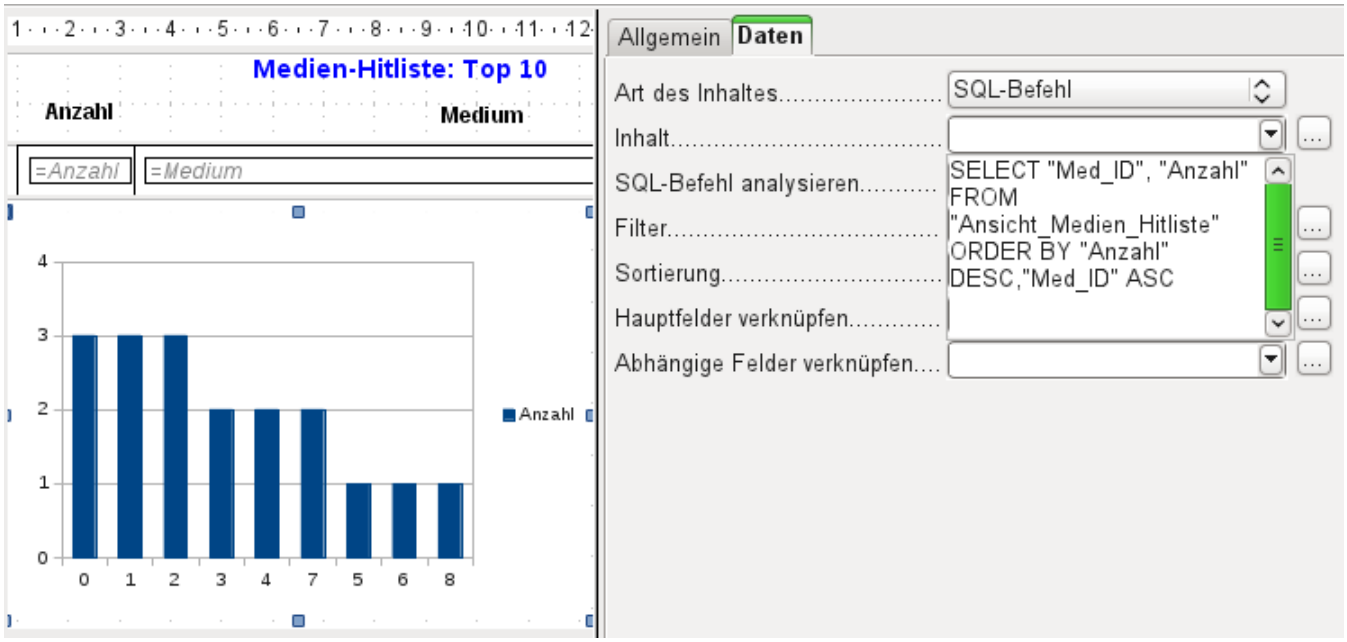
Über das entsprechende Kontrollfeld oder **Einfügen → Bericht-Steuerelemente → Diagramm** lässt sich ein Diagramm dem Bericht hinzufügen. Ein Diagramm ist die einzige Möglichkeit, in einem Bericht Daten wiederzugeben, die nicht aus der Quelle stammen, die im Bericht als Datenquelle angegeben ist. Ein Diagramm ist insofern ein Unterbericht des Berichtes, kann aber auch als eigenständiger Berichtsteil gesehen werden.

Hinweis

Die Funktionsfähigkeit von Diagrammen ist in Berichten nicht beständig gewährleistet. Von LO 5.3 bis LO 7.3.2 funktionierte kein Diagramm im Report-Designer. Noch schlimmer: Alte Diagramme ließen sich nicht einmal entfernen, da die Auswahl der Diagramme dazu führte, dass LO direkt abstürzt.
Seit der Version LO 7.3.3 werden Diagramme wieder angezeigt. Auch die Absturzgefahr beim Bearbeiten von Diagrammen, die eventuell nicht korrekt funktionieren, ist behoben.

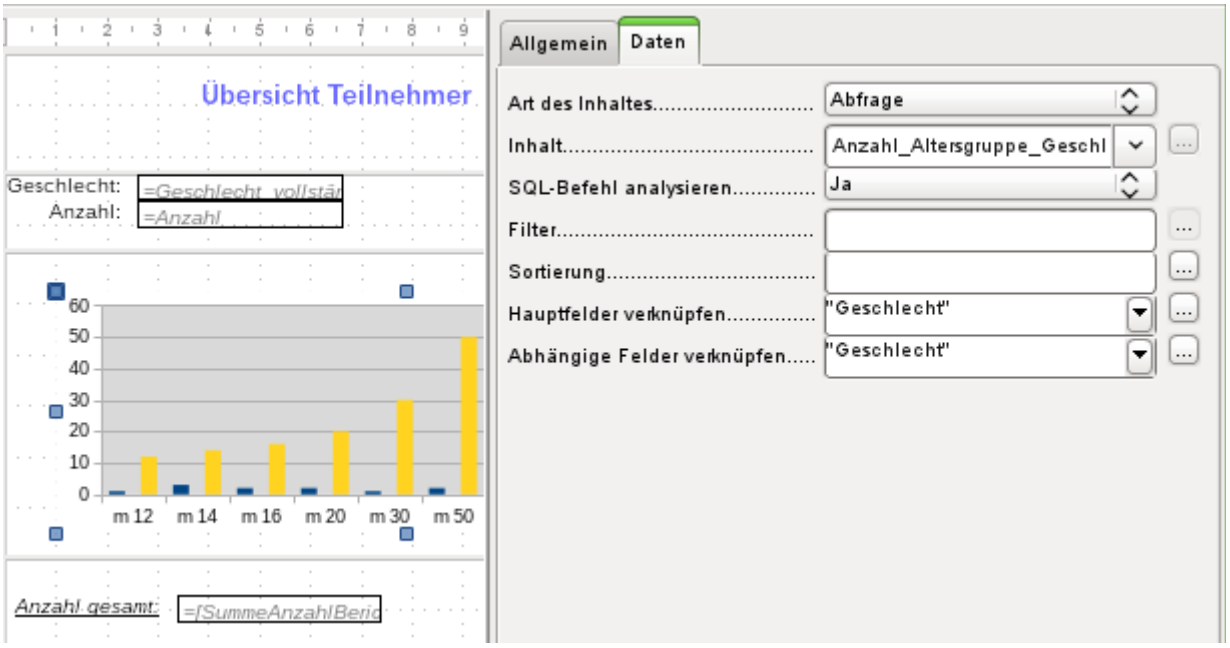
Das Diagramm wird mit der Maus aufgezogen. Bei den allgemeinen Eigenschaften zeigt sich neben bekannten Feldern die Möglichkeit, einen **Diagrammtyp** über den **Button** ... auszuwählen (siehe entsprechende Typen in Calc). Außerdem wird eine maximal für die Vorschau benutzte Anzahl an Datensätzen (**Vorschau für Zeile(n)**) eingestellt, die schon einmal eine Vorstellung davon geben kann, wie das Diagramm letztlich aussieht.

Diagramme können, ebenfalls wie in Calc, entsprechend formatiert werden (Doppelklick auf das Diagramm). Hierzu siehe die Beschreibungen im Handbuch Calc.



Das Diagramm wird im Reiter **Daten** mit den erforderlichen Datenfeldern verbunden. Hier, im Beispielbericht «Medien-Hitliste», soll das Diagramm die Häufigkeit deutlich machen, mit der bestimmte Medien entliehen wurden. Dazu wurde ein SQL-Befehl über den Abfrageeditor wie bei Listenfeldern erstellt und eingebunden. Die erste Spalte wird als die angesehen, mit der die Säulen beschriftet werden sollen, die zweite Spalte liefert dann die Anzahl der Entleihvorgänge, die sich in den Höhen der Säulen widerspiegelt.

In dem obigen Beispiel zeigt das Diagramm erst einmal nur sehr wenig an, da die Test-Entleihvorgänge sich zum Zeitpunkt der SQL-Eingabe in Grenzen hielten.



In den Dateneigenschaften des Diagramms ist hier eine **Abfrage** eingebunden. Dieses Diagramm aus der Datenbank «Beispiel_Sport.odb»[14] zeigt neben den Grundlagen für die Erstellung von Diagrammen in Berichten eine Besonderheit auf: Die Vorschau des Diagramms zeigt mehr Säulen an, als an sich für das Diagramm vorgesehen sind. Dies liegt an der Zusammensetzung der Abfrage, die gleichzeitig mehrere Spalten darstellt, die eben nicht alle im Diagramm selbst dargestellt werden sollen.

Eine Filterung und Sortierung mit den internen Werkzeugen des Report-Designers ist nicht notwendig, weil dies eben schon so weit wie möglich über die Abfrage erledigt wurde.

Tipp

Grundsätzlich sollte die Berichtserstellung von so viel Aufgaben wie möglich entlastet werden. Was vorher durch Abfragen bewerkstelligt werden kann, das muss nachher nicht mehr die im Verhältnis zur Datenbank doch recht träge Ausführung des Berichtes belasten.

Wie bei Hauptformularen und Unterformularen werden jetzt Felder verknüpft. Im eigentlichen Bericht werden tabellarisch die Altersgruppen für männliche und weibliche Teilnehmer des Sportwettkampfs gelistet. Dabei ist nach den Geschlechtern gruppiert worden. In jeder Gruppe erscheint jetzt ein gesondertes Diagramm. Damit das Diagramm nur die zu dem Geschlecht gehörigen Daten übermittelt, sind die Felder "*Geschlecht*" aus dem Bericht und "*Geschlecht*" aus dem Diagramm miteinander verbunden.

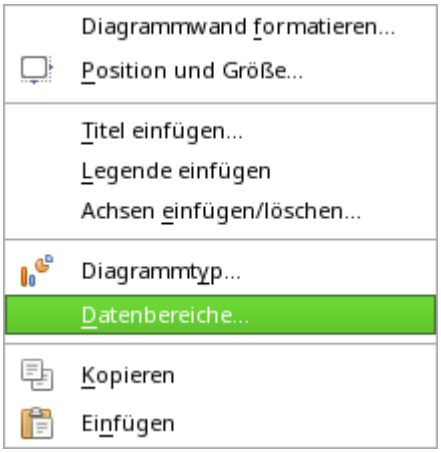
Hinweis

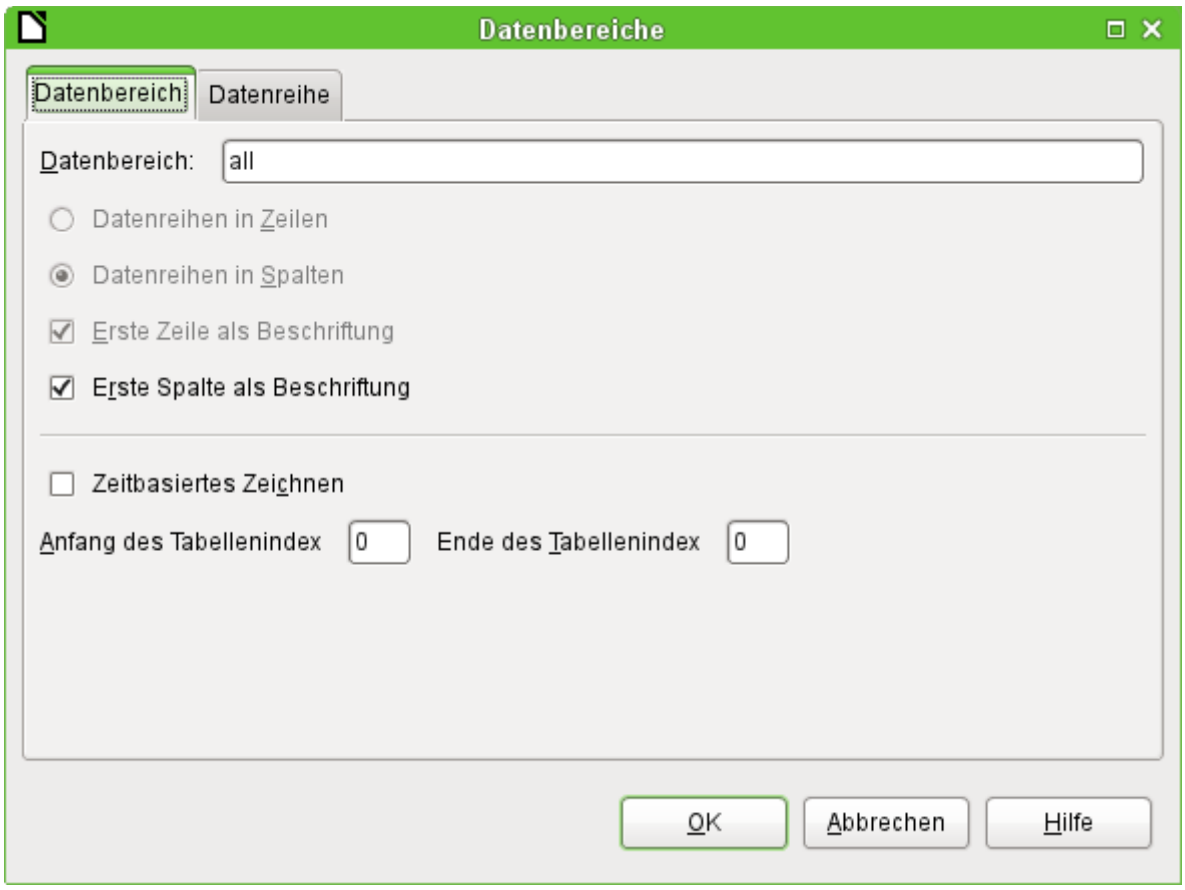
Haben Diagramme die gleiche Datenquelle (Tabelle, Abfrage) wie der Bericht selbst, so funktioniert die Verknüpfung von Hauptfeldern und abhängigen Feldern nicht. Hier muss für eine funktionierende Verknüpfung eine zweite Abfrage gleichen Inhaltes erstellt oder der SQL-Code direkt als Datenquelle für das Diagramm eingegeben werden.

Die x-Achse des Diagramms wird automatisch erst einmal mit der ersten Tabellenspalte der Datenquelle verbunden. Bei mehr als zwei Tabellenspalten stellt die Automatik gleich mehrere Säulen in dem Diagramm dar. Weitere Einstellungen zu dem Diagramm sind zu erreichen, wenn das gesamte Diagramm mit einem Doppelklick markiert wird. Mit einem Klick der rechten

14 Die Datenbank «Beispiel_Sport.odb» ist den Beispieldatenbanken für dieses Handbuch beigefügt.

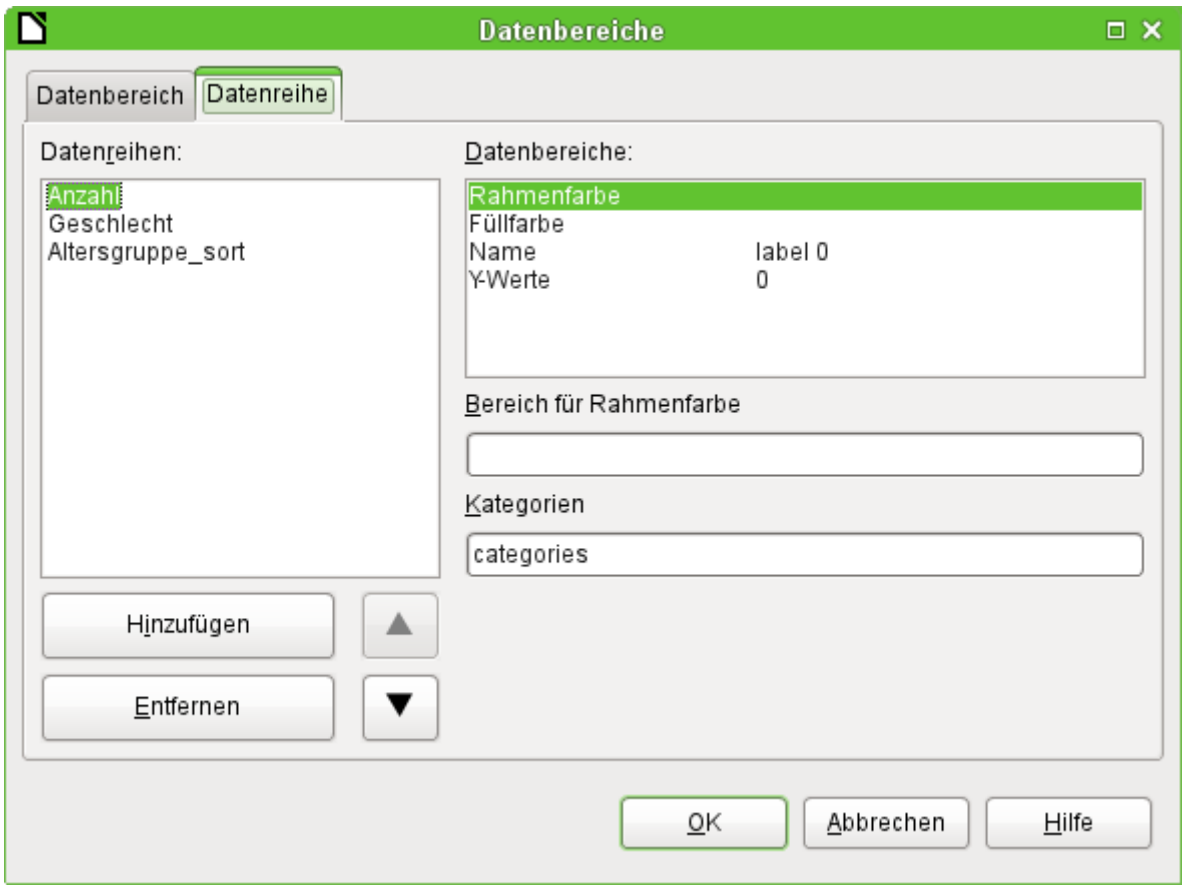
Maustaste öffnet sich über dem Diagramm, je nach markiertem Element, ein Kontextmenü. Hierin enthalten ist immer die Einstellmöglichkeit für die Datenbereiche:





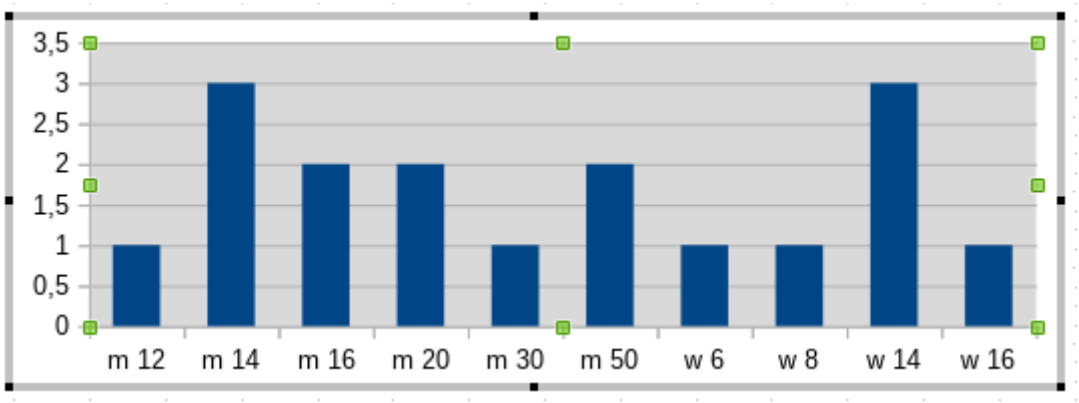
Datenreihen in Spalten ist ausgegraut, also nicht änderbar. Auch eine Änderung des Markierfeldes **Erste Spalte als Beschriftung** ist nicht möglich. An den weiteren Einstellungen des Datenbereichs sollten tunlichst keine Änderungen vorgenommen werden, da dieser Dialog mehr Möglichkeiten offeriert, als tatsächlich für den Report-Designer sinnvoll machbar sind.

Der Reiter **Datenreihen** hingegen verbirgt ein paar Einstellungsmöglichkeiten, die die Standarddarstellung deutlich verändern können. Es werden dort alle Datenreihen angeboten, die neben der ersten Spalte der Abfrage noch verfügbar sind. Datenreihen, die nicht angezeigt werden sollen, können hier entfernt werden.



In dem obigen Beispiel waren zu viele Säulen in dem Diagramm sichtbar. Hier muss also nachgebessert werden. Weder die Datenreihe «Geschlecht» noch die Datenreihe «Altersgruppe_sort», deren Bezeichnungen jeweils mit denen in der zugrundeliegenden Abfrage überein stimmen, ergeben hier einen Sinn. Die Datenreihe «Geschlecht» dient schließlich zur Verbindung des Diagramms mit der Datengrundlage des Berichts und lässt sich numerisch überhaupt nicht darstellen. Die Datenreihe «Altersgruppe_sort» dient nur zur Sortierung der Werte der Abfrage, da sonst Bezeichnungen wie «m 8» nicht am Anfang des Diagramms sondern direkt vor «m 80» einsortiert würden – die Sortierung eines Textes führt eben manchmal zu unerwünschten Ergebnissen.

Nachdem bis auf die Datenreihe «Anzahl» alle Datenreihen entfernt wurden, sieht die Vorschau des Diagramms so aus:



Die Vorschau zeigt hier 10 Säulen auf – die ersten 10 Säulen aus der Abfrage. In der Ausführung werden allerdings anschließend nur die Säulen dargestellt, die zu der entsprechenden Gruppe passen: 'm' für «männlich» und 'w' für «weiblich».

Die y-Achse weist noch eine ungünstige Achsenteilung auf. Schließlich kann es nicht halbe Personen geben. Hier könnte zwar nachgebessert werden. Allerdings wird bei der automatischen Fassung dieser Einstellungen schnell eine Anpassung zu Ganzzahlen erfolgen, wenn die Werte eben nicht, wie in obigen Beispiel, bereits bei einer Anzahl von '3' enden. Dann wäre erneut ein Nachbessern notwendig, wenn vorher die Automatik ausgeschaltet wurde.

Hinweis

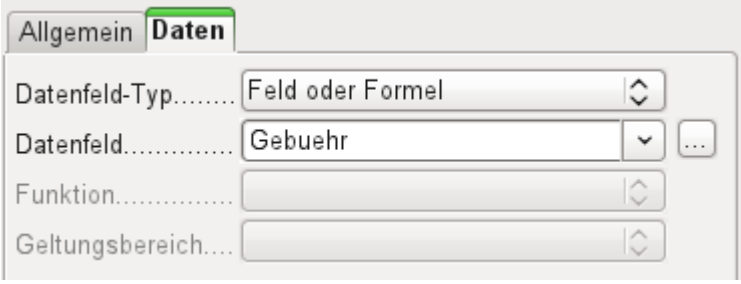
Aufgrund verschiedener Bugs hat die Anzeige eines Diagramms Probleme mit Datumsfeldern und Zeitfeldern. Hier empfiehlt es sich, die Datums- bzw. Zeitangaben in einer Abfrage in Zahlen umzuwandeln. Eine mögliche Abfrage für ein Datumsfeld in der HSQLDB wäre:

```
001 SELECT
002 DATEDIFF( 'dd', '1899-12-30', "Datum" ) AS "DatumInteger",
003 "Betrag"
004 FROM "Verkauf"
```

Anschließend muss allerdings die entsprechende Achse des Diagramms bearbeitet werden. Die automatische Erkennung aus der Datenquelle erstellt hieraus eine Zahl. Diese automatische Erkennung muss abgewählt und stattdessen das Datum angewählt werden.

Alle weiteren Einstellungen sind ähnlich den Möglichkeiten, die LO-Calc für die Einstellung von Diagrammen bietet.

Dateneigenschaften von Feldern



In den Eigenschaften zeigt sich hinter dem Reiter **Daten** standardmäßig nur das Feld der Datenbank, aus dem die Daten für die Felder des Berichtes ausgelesen werden. Allerdings stehen neben dem **Datenfeld-Typ → Feld oder Formel** noch die Typen '*Funktion*', '*Zähler*' und '*Benutzerdefinierte Funktion*' zur Verfügung.

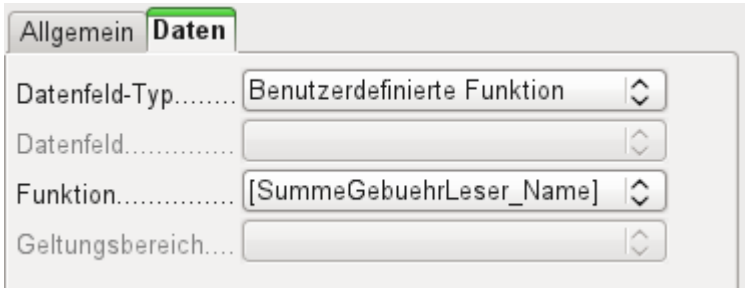
Vorwählbar sind die Funktionen '*Summe*', '*Minimum*' und '*Maximum*'. Ihr Geltungsbereich bezieht sich entweder auf die aktuelle Gruppe oder auf den gesamten Bericht. Bereits diese Funktionen können zu Problemen führen, wenn ein Feld leer, also **NULL**, ist. In solchen Feldern, sofern sie als Zahl formatiert sind, erscheint «N,aN», was wohl so viel wie «kein Zahlenwert» bedeuten soll. Mit leeren Feldern wird keine Rechnung durchgeführt; das Ergebnis ist '0'.

Solche Felder können zwar in der Ansicht durch die folgende Formel im Bereich «Daten» zur Anzeige von 0-Werten umformatiert werden.

```
001 IF([Zahlenfeld];[Zahlenfeld];0)
```

Die Funktion rechnet aber mit dem tatsächlichen Wert, also mit dem Feld, das eben keinen Wert besitzt. Da ist es sinnvoller, wenn gewünscht, die dem Bericht zugrundeliegende Abfrage so zu formulieren, dass statt NULL in Zahlenfeldern '0' wiedergegeben wird.

Der '*Zähler*' zählt einfach nur die Datensätze, die entweder in einer Gruppe oder im ganzen Bericht enthalten sind. Wird der Zähler in den Bereich Detail eingesetzt, so wird durch ihn jeder Datensatz mit einer fortlaufenden Nummer versehen. Dabei kann die Nummerierung auch hier nur die Gruppe oder sämtliche Datensätze des Berichts fortlaufend nummerieren.



Schließlich stehen noch die detaillierten *Benutzerdefinierte Funktionen* zur Verfügung. Es kann passieren, dass der Report-Designer diese Variante selbst wählt, wenn er zwar den Rechenschritt vorgegeben bekommt, aber die Datenquelle aus irgendeinem Grunde nicht richtig interpretieren kann. Dann können allerdings dort auch weitere Einstellungen vorgenommen werden.

Alle Funktionen sind anschließend auch über den *Bericht-Navigator* zur weiteren Bearbeitung verfügbar. Dort kann dann z.B. auch eine Vorausberechnung von Werten eingestellt werden. Über den Bericht-Navigator können auch eigene benutzerdefinierte Funktionen für jede Gruppe oder den gesamten Bericht erstellt werden.

Hinweis

Manchmal kann es bei Berichten auf Grundlage einer Abfrage passieren, dass sich eine vorwählbare Funktion einem Feld in einer Gruppe nicht zuordnen lässt. Stattdessen friert der Report-Designer (und damit LibreOffice insgesamt) ein (*Bug 156186*). Das passiert, wenn die Gruppenbezeichnung einen Unterstrich enthält. Hier hilft es, das entsprechende Feld in der Abfrage mit einem Alias zu versehen, so dass kein Unterstrich vorkommt.

Funktionen im Report-Designer

Der Report-Designer bietet sowohl bei den Daten als auch bei den Bedingungen für eine Anzeige die Nutzung verschiedenen Funktionen an. Reichen die Funktionen nicht, so lassen sich mit einfachen Rechenschritten auch benutzerdefinierte Funktionen erstellen, die vor allem in Gruppenfüßen als Zusammenfassungen Sinn machen.

Formeleingaben

Dem Report-Designer liegt der «Pentaho Report Designer» zugrunde. Ein kleiner Teil der Dokumentation ist unter
http://wiki.pentaho.com/display/Reporting/9.+Report+Designer+Formula+Expressions
zu finden.

Eine weitere Quelle sind die Spezifikationen nach «OpenFormular-Standard»:

http://www.oasis-open.org/committees/download.php/16826/openformula-spec-20060221.html

Teilweise ist auch ein einfacher Blick in Richtung Calc sinnvoll, da dort eine deutsche Beschreibung zumindest einiger Formeln aus dem Report-Designer enthalten ist.

Grundlegend gilt:

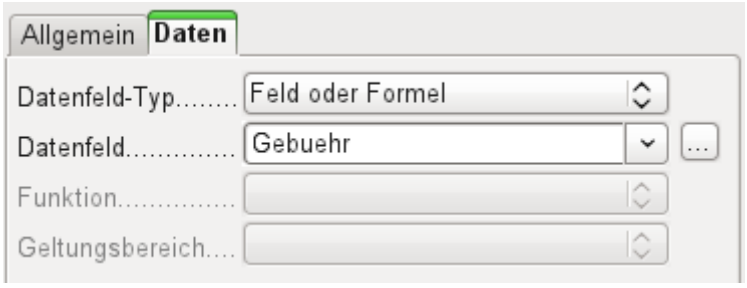
Formeleingaben starten mit einem Gleichheitszeichen	**=**
Bezüge zu Datenfeldern werden in eckige Klammern gesetzt	**[Feldbezeichnung]**
Enthalten die Datenfelder Sonderzeichen, zu denen auch Leertasten gehören, so ist die Feldbezeichnung außerdem noch in doppelte Anführungszeichen zu setzen	**["Diese Feldbezeichnung wäre in doppelte Anführungszeichen zu setzen"]**
Texteingaben haben immer in doppelten Anführungszeichen zu stehen	**"Texteingabe"**
Der Dezimaltrenner ist wie in SQL der Punkt, nicht das Komma	**1.27**
Folgende Operatoren sind möglich	+, -, * (Multiplikation), / (Division), % (dividiert vorher stehende Zahl durch 100), ^ (Potenzierung mit der nachfolgenden Zahl), & (verbindet Texte miteinander),
Folgende Beziehungsdefinitionen sind möglich	**= , <> , < , <= , > , >=**
Runde Klammerungen sind ebenfalls möglich	**( )**
Standardfehlermeldung	**NA** (vermutlich not available, in Calc in NV übersetzt – nicht verfügbar)
Fehlermeldung beim Bericht, wenn ein leeres Feld auftaucht und als Zahl definiert ist.	**N,aN** (erscheint, wenn kein Wert enthalten ist – vielleicht not a number?)

Sämtliche Formeleingaben wirken sich nur auf den aktuell eingelesenen Datensatz aus. Beziehungen zu vorhergehenden oder nachfolgenden Datensätzen sind also nicht möglich, wohl aber z. B. die Berechnung einer Summe zwischen Feldern eines Datensatzes. Enthält z. B. ein Datensatz die Felder "Nettobetrag" und "MWSt", so ergibt die folgende Formel den Bruttobetrag:

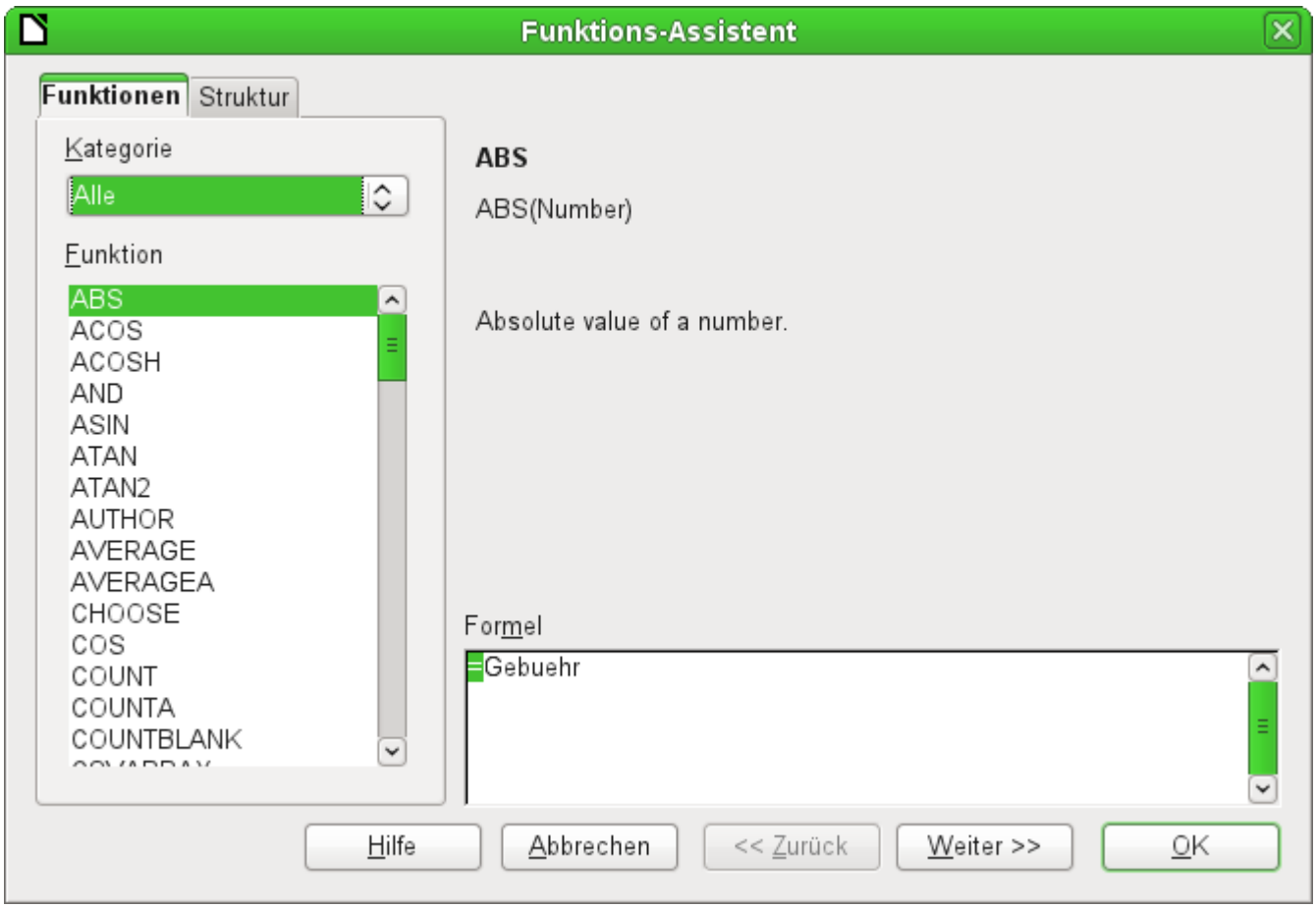
```
001 =[Nettobetrag]+[MWSt]
```

Möglich ist aber auch:

```
001 =SUM([Nettobetrag];[MWSt])
```



Neben dem Datenfeld erscheint der Button ..., sofern eine Formeleingabe möglich ist. Dieser Button startet den Funktionsassistenten.



Im Gegensatz zu den Funktionen in Calc sind hier die Funktionen nicht ins Deutsche übertragen. Die Funktionen sind längst nicht so umfangreich wie die, die in Calc üblich sind. Viele Funktionen haben allerdings ihre Entsprechung in Calc. Dort gibt der Assistent dann auch direkt das Ergebnis der Funktion wieder.

Der Funktions-Assistent funktioniert nicht immer einwandfrei. So werden z. B. Texteingaben nicht mit doppelten Anführungszeichen übernommen. Nur Eingaben mit doppelten Anführungszeichen werden allerdings verarbeitet, so dass die Anführungszeichen später wieder ergänzt werden müssen.

Die folgenden Funktionen stehen zur Verfügung: [**Funktion im Report-Designer** / (*Funktion in Calc)]

Funktion	***Beschreibung***
Datums – und Zeitfunktionen	
DATE *(DATUM)*	Erzeugt aus einer Zahlenangabe für das Jahr, den Monat und den Tag ein gültiges Datum.
DATEDIF *(TAGE | MONATE | JAHRE)*	Gibt die Anzahl an Jahren, Monaten oder Tagen zwischen zwei Datumswerten wieder.
DATEVALUE *(DATWERT)*	Wandelt eine amerikanische Datumseingabe in Textform (Anführungsstriche) in eine Datumsbeschreibung um. Die erzeugte amerikanische Variante kann durch Formatierungseinstellung umgewandelt werden.
DAY *(TAG)*	Gibt den Tag des Monats eines angegebenen Datums wieder. DAY([Datumsfeld])

DAYS *(TAGE)*	Gibt die Zahl der Tage zwischen zwei Datumseingaben wieder.
HOUR *(STUNDE)*	Gibt die Stunde einer angegebenen Zeit im Rahmen von 0 bis 23 wieder. HOUR([DatumZeitFeld]) gibt die Stunde des Feldes wieder.
MINUTE *(MINUTE)*	Gibt die Minuten einer internen Zahl wieder. MINUTE([Zeitfeld]) gibt den Minutenanteil der Zeit wieder.
MONTH *(MONAT)*	Gibt den Monat einer Datumseingabe als Zahl wieder. MONTH([Datumsfeld])
NOW *(JETZT)*	Gibt das aktuelle Datum und die aktuelle Zeit wieder.
SECOND *(SEKUNDE)*	Gibt die Sekunden einer internen Zahl wieder. SECOND(NOW()) gibt den Sekundenanteil der Zeit beim Ausführen des Berichts wieder.
TIME *(ZEIT)*	Zeigt die aktuelle Zeit an.
TIMEVALUE *(ZEITWERT)*	Wandelt die Texteingabe einer Zeit in eine Zeit für Berechnungen um
TODAY *(HEUTE)*	Gibt das aktuelle Datum wieder
WEEKDAY *(WOCHENTAG)*	Gibt den Wochentag einer Datumseingabe als Zahl wieder. Der Tag mit der Nummer 1 ist der Sonntag.
YEAR *(JAHR)*	Gibt das Jahr einer Datumseingabe wieder.
Logische Funktionen	
AND *(UND)*	Gibt TRUE wieder, wenn alle Argumente TRUE sind
FALSE *(FALSCH)*	Definiert den logischen Wert als FALSE
IF *(WENN)*	Wenn eine Bedingung TRUE, dann diesen Wert, ansonsten einen anderen Wert.
IFNA	Wenn der Wert nicht verfügbar ist, dann setze diesen Wert. Ist ähnlich der Anweisung **IFNULL** in SQL. Vor allem bei Funktionen sinnvoll, die Feldwerte addieren sollen, bei denen aber nicht klar ist, ob jedes Feld einen Wert hat. **IFNA([Feldwert];0)** rechnet dann trotzdem z.B. die Summe aus. (ab LO 3.5)
NOT *(NICHT)*	Kehrt den logischen Wert eines Argumentes um
OR *(ODER)*	Gibt TRUE zurück, wenn eine der Bedingungen TRUE ist.
TRUE *(WAHR)*	Definiert den logischen Wert als TRUE
XOR	Nur wenn ein einziger Wert der Verknüpfung TRUE ist, ist auch der logische Wert TRUE.
Rundungsfunktionen	
INT	Rundet zum nächsten Ganzzahlwert ab

Mathematische Funktionen	
ABS *(ABS)*	Gibt den absoluten, nicht negativen Wert einer Zahl wieder.
ACOS *(ARCCOS)*	Berechnet den Arcuscosinus einer Zahl. - Werteingabe zwischen -1 und 1 (ab LO 3.5)
ACOSH *(ARCCOSHYP)*	Berechnet den Areacosinus (inverser hyperbolischer Cosinus) – Werteingabe >= 1 (ab LO 3.5)
ASIN *(ARCSIN)*	Berechnet den Arcussinus einer Zahl. - Werteingabe zwischen -1 und 1 (ab LO 3.5)
ATAN *(ARCTAN)*	Berechnet den Arcustangens einer Zahl. (ab LO 3.5)
ATAN2 *(ARCTAN2)*	Berechnet den Arcustangens einer x-Koordinate und einer y-Koordinate. (ab LO 3.5)
AVERAGE *(MITTELWERT)*	Ermittelt den Mittelwert der angegebenen Werte (taucht im Formularassistenten LO 3.3.4 doppelt auf).
AVERAGEA *(MITTELWERTA)*	Ermittelt den Mittelwert der angegebenen Werte. Text wird als 0 gewertet. (ab LO 3.5)
COS *(COS)*	Winkel im Bogenmaß, dessen Cosinus berechnet werden soll. (ab LO 3.5)
EVEN *(GERADE)*	Rundet eine positive Zahl zum nächsten geraden Integer auf, eine negative Zahl zum nächsten geraden Integer ab.
EXP *(EXP)*	Berechnet die Exponentialfunktion zur Basis 'e' (ab LO 3.5)
LN *(LN)*	Berechnet den natürlichen Logarithmus einer Zahl. (ab LO 3.5)
LOG10 *(LOG10)*	Berechnet den Logarithmus einer Zahl zur Basis '10'. (ab LO 3.5)
MAX *(MAX)*	Gibt den Maximalwert aus einer Reihe von Werten wieder.
MAXA *(MAXA)*	Gibt den Maximalwert aus einer Reihe wieder. Eventuell vorhandener Text wird als '0' gesetzt.
MIN *(MIN)*	Gibt den Minimalwert aus einer Reihe von Werten wieder.
MINA *(MINA)*	Gibt den Minimalwert aus einer Reihe wieder. Eventuell vorhandener Text wird als '0' gesetzt.
MOD *(REST)*	Gibt den Rest einer Division nach Eingabe von Dividend und Divisor wieder
ODD *(UNGERADE)*	Rundet eine positive Zahl zum nächsten ungeraden Integer auf, eine negative Zahl zum nächsten ungeraden Integer ab.
PI *(PI)*	Liefert den Wert der Zahl 'π' (ab LO 3.5)

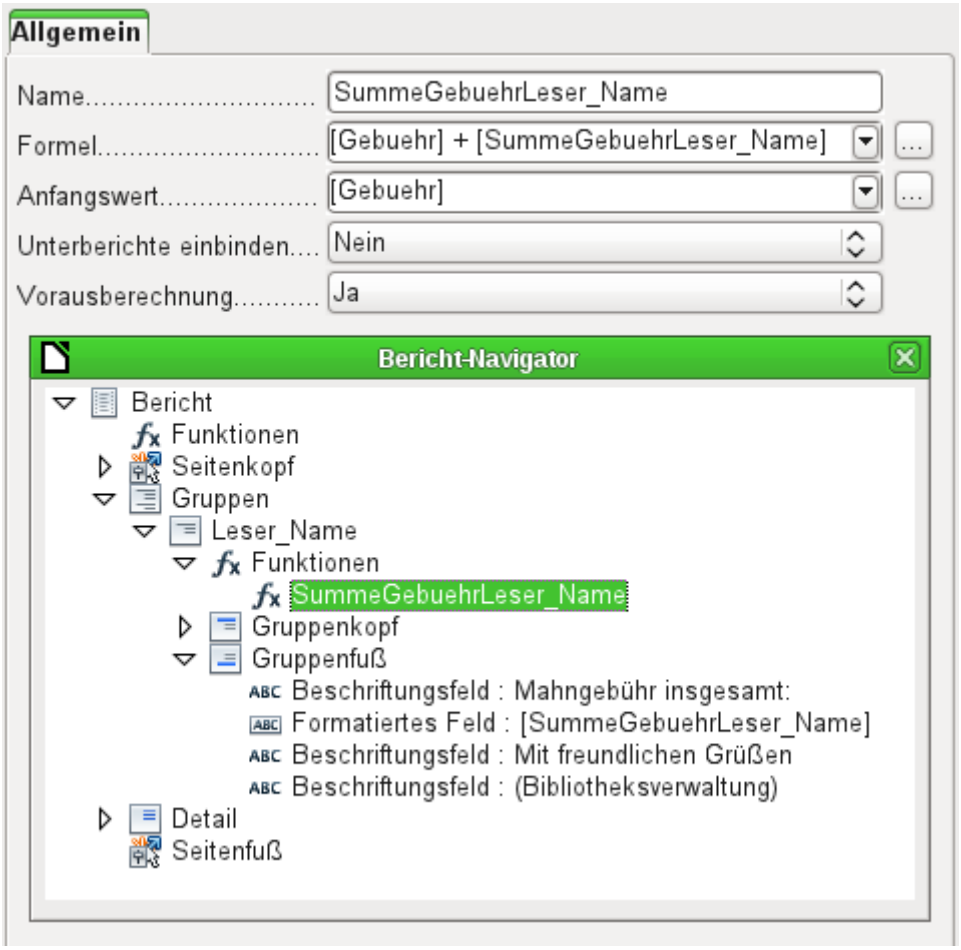
POWER *(POTENZ)*	Liefert aus Basis und Exponent die Potenz. (ab LO 3.5)
SIN *(SIN)*	Berechnet den Sinus einer Zahl. (ab LO 3.5)
SQRT *(WURZEL)*	Berechnet die Quadratwurzel einer Zahl. (ab LO 3.5)
SUM *(SUMME)*	Summiert eine Liste von Zahlenwerten
SUMA	Summiert eine Liste von Zahlenwerten. Text und Ja/Nein-Felder sind erlaubt. Leider endet diese Funktion (noch) mit Fehlermeldung. (ab LO 3.5)
VAR *(VARIANZ)*	Berechnet die Varianz, ausgehend von einer Stichprobe. (ab LO 3.5)
Textfunktionen	
EXACT *(IDENTISCH)*	Zeigt an, ob zwei Texte völlig gleich sind.
FIND *(FINDEN)*	Gibt die Position eines zu suchenden Textes in einem anderen Text an.
LEFT *(LINKS)*	Der Text wird von links aus in der angegebenen Zeichenzahl wiedergegeben.
LEN *(LÄNGE)*	Gibt die Anzahl an Zeichen wieder, die ein Text hat.
LOWER *(KLEIN)*	Gibt den Text in Kleinbuchstaben wieder.
MESSAGE	Formatiert die Werte in dem angegebenen Ausgabeformat. (ab LO 3.5)
MID *(TEIL)*	Gibt Text ab einer Anfangsposition mit einer gegebenen Zeichenlänge wider.
REPLACE *(ERSETZEN)*	In dem Text wird ein Teil durch einen anderen Text ersetzt. Die Startposition und die Länge des zu ersetzenden Textes wird eingegeben.
REPT *(WIEDERHOLEN)*	Wiederholt einen Text die angegebenen Male.
RIGHT *(RECHTS)*	Der Text wird von rechts aus in der angegebenen Zeichenzahl wiedergegeben.
SUBSTITUTE *(WECHSELN)*	Ersetzt in einem vorgegebenen Text bestimmte Textteile durch andere Textteile. Zusätzlich kann angegeben werden, der wievielte Textteil ersetzt werden soll.
T *(T)*	Gibt den Text wieder oder einen leeren Textwert, wenn es sich z. B. um eine Zahl handelt.
TEXT *(TEXT)*	Konvertierung von Zahlen oder Uhrzeiten in Text.
TRIM *(GLÄTTEN)*	Entfernt führende Leerzeichen, Leerzeichen am Textende und reduziert mehrere Leerzeichen hintereinander im Text auf ein Leerzeichen.

UNICHAR *(UNIZEICHEN)*	Wandelt eine Unicode-Nummer in Dezimalschreibweise in ein Unicode-Zeichen um 196 wird zu 'Ä' ('Ä' hat den Hexadezimalwert 00C4, daraus wird bei der Dezimalschreibweise 196 ohne führende Nullen)
UNICODE *(UNICODE)*	Wandelt eine Unicode-Zeichen in eine Unicode-Nummer in Dezimalschreibweise um 'Ä' wird zu 196
UPPER *(GROSS)*	Gibt den Text in Großbuchstaben wieder.
URLENCODE	Wandelt einen vorgegebenen Text in einen Text um, der URL-Konform ist. Wenn keine besonderer Code vorgegeben wurde, passiert dies nach ISO-8859-1.
Informationsfunktionen	
CHOOSE *(Wahl)*	Zuerst wird ein Index angegeben, danach eine Liste von Werten, aus der über den Index der entsprechende Wert ausgesucht wird. **CHOOSE(2;"Apfel";"Birne";"Banane")** gibt **Birne** wieder. **CHOOSE([Altersstufenfeld];"Milch";"Cola";"Bier")** gibt je nach "Altersstufenfeld" aus der Datenquelle wieder, welches Getränk möglich ist.
COUNT *(Anzahl)*	Nur die Felder werden gezählt, die eine Zahl oder einen Datums- bzw. Zeitwert enthalten. **COUNT([Zeit];[Nummer])** gibt **2** aus, wenn beide Felder einen Wert enthalten, also nicht NULL sind, ansonsten 1 oder schließlich 0.
COUNTA *(Anzahl2)*	Berücksichtigt auch die Felder, die einen Text beinhalten. Auch NULL wird gezählt, ebenso boolsche Felder.
COUNTBLANK *(ANZAHLLEEREZELLEN)*	Zählt die leeren Felder in einem Bereich.
HASCHANGED	Überprüft, ob die mit ihrem Namen angegebene Spalte sich geändert hat. Allerdings werden keine Spaltenangaben übernommen.
INDEX *(INDEX)*	Arbeitet mit Bereichen (ab LO 3.5)
ISBLANK *(ISTLEER)*	Prüft, ob das Feld NULL (leer) ist.
ISERR *(ISTFEHL)*	Gibt TRUE zurück, wenn die Eingabe fehlerhaft ist, aber nicht vom Typ NA **ISERR(1/0)** ergibt **TRUE**
ISERROR *(ISTFEHLER)*	Wie ISERR, nur dass auch NA als Meldung TRUE ergibt.
ISEVEN *(ISTGERADE)*	Prüft, ob es sich um eine gerade Zahl handelt.
ISLOGICAL *(ISTLOG)*	Prüft, ob es sich um einen Ja/Nein-Wert handelt. **ISLOGICAL(TRUE())** oder **ISLOGICAL(FALSE())** ergeben **TRUE**, Textwerte wie **ISLOGICAL("TRUE")** ergeben **FALSE**.
ISNA *(ISTNV)*	Prüft, ob der Ausdruck vom Fehlertyp NA ist.
ISNONTEXT *(ISTKTEXT)*	Prüft, ob es sich bei dem Wert nicht um einen Text handelt.

ISNUMBER *(ISTZAHL)*	Prüft, ob es sich um eine Zahl handelt. **ISNUMBER(1)** ergibt **TRUE**, **ISNUMBER("1")** ergibt **FALSE**
ISODD *(ISTUNGERADE)*	Prüft, ob es sich um eine ungerade Zahl handelt.
ISREF *(ISTBEZUG)*	Prüft, ob es sich um einen Bezug handelt. **ISREF([Feldname])** ergibt **TRUE**, **ISREF(1)** ergibt **FALSE**.
ISTEXT *(ISTTEXT)*	Prüft, ob es sich bei dem Wert um einen Text handelt.
NA *(NV)*	Gibt den Fehlercode **NA** wieder.
VALUE	(ab LO 3.5)
Benutzerdefiniert	
CSVARRAY	Wandelt einen CSV-Text in ein Array um. (ab LO 3.5)
CSVTEXT	Wandelt ein Array in einen CSV-Text um. (ab LO 3.5)
NORMALIZEARRAY	(ab LO 3.5)
NULL	Gibt NULL wieder
PARSEDATE	Wandelt Text in ein Datum um. Nutzt das SimpleDateFormat. Benötigt ein Datum als Text sowie die Beschreibung des Datumformates. Beispiel: PARSEDATE("9.10.2012";"dd.MM.yyyy") ergibt die intern verwertbare Zahl für das Datum. (ab LO 3.5)
Dokumentsinformationen	
AUTHOR	Autor, wird ausgelesen aus **Extras → Optionen → LibreOffice → Benutzerdaten**. Der eigentliche Autor wird hier also nicht wiedergegeben, sondern der aktuelle Nutzer der Datenbank.
TITLE	Gibt den Titel des Berichts wieder.

Benutzerdefinierte Funktionen

Benutzerdefinierte Funktionen können z. B. dazu dienen, bestimmte Zwischenergebnisse nach einer Gruppe von Datensätzen wieder zu geben. Im obigen Beispiel ist so etwas über die vordefinierte Funktion der Summe bei der Berechnung der Gebühr im Bereich «Leser_Name_Fuß» automatisch erstellt worden.



Im Bericht-Navigator ist für die Gruppe «*Leser_Name*» eine Funktion verzeichnet. Durch Rechtsklick auf Funktionen können zusätzliche Funktionen hier mit Namen definiert werden.

Die Funktion «*SummeGebuehrLeser_Name*» wird in den Eigenschaften angezeigt. Die **Formel** führt eine Addition des Feldes «*Gebuehr*» mit dem in der Funktion selbst bereits gespeicherten Wert durch. Der **Anfangswert** ergibt sich aus dem Wert des Feldes «*Gebuehr*» beim ersten Durchgang durch die Gruppe. Dieser Wert wird in der Funktion unter dem Funktionsnamen zwischengespeichert und in der Formel wieder benutzt, bis die Schleife beendet ist und der Gruppenfuß geschrieben wird.

Unterberichte einbinden scheint momentan keine Funktion zu haben, es sei denn, dass Diagramme als Unterberichte verstanden werden.

Ist für die Funktion **Vorausberechnung** aktiviert, so kann das Ergebnis auch im Gruppenkopf stehen. Ohne die Aktivierung steht im Gruppenkopf nur der entsprechende Wert des abgefragten ersten Feldes der Gruppe.

Selbstdefinierte Funktionen können auch auf selbstdefinierte Funktionen zurückgreifen. Dabei ist dann aber zu beachten, dass die genutzten Funktionen vorher ausgeführt werden müssen. Wird die Vorausberechnung in dieser Funktion, die auf andere Funktionen zurückgreift, ausgeschaltet, so lässt sich mit der folgenden Kombination im Detail-Bereich der Zensurenschnitt nach jedem Datensatz berechnen.

```
001 [SummeZensurKlasse] / [ZählerKlasse]
```

Wird die Kombination in den Gruppenfuß der Gruppe «*Klasse*» gesetzt, so erscheint dort der Durchschnitt für die gesamte Klasse. Im Gruppenkopf erscheint hingegen als Durchschnitt nur die Zensur des ersten Schülers, sofern die Vorausberechnung für **SummeZensurKlasse** ausgeschaltet wurde. Mit eingeschalteter Vorausberechnung ist auch im Gruppenkopf der korrekte Wert zu erreichen.

Funktionen wie «ZählerKlasse» zeigen ihren Inhalt nur in der Gruppe an. Werden sie im Seitenkopf oder Seitenfuß genutzt, so bleibt der Inhalt leer.

Formeleingabe für ein Feld

Über den Weg **Daten → Datenfeld** können Formeln eingegeben werden, die nur ein einziges Feld im Bereich «Detail» betreffen.

```
001 IF([boolschesFeld];"ja";"nein")
```

schreibt, dort eingegeben, statt WAHR und FALSCH einfach "ja" und "nein".

Ein boolsches Feld hat ja auch die Möglichkeit, neben "ja" und "nein" ein leeres Feld (**NULL**) zu präsentieren. Hier ein Code, der das Ganze gleich in entsprechende Feldform bringt:

```
001 IF([boolschesFeld];"☒";IF(ISBLANK([boolschesFeld]);"-";"☐"))
```

Es kann passieren, dass in einem Feld mit einer Formeleingabe grundsätzlich eine Zahl erscheint. Bei Text ist das dann eine «0». Hier muss nachgebessert werden, indem für das Textfeld vom Standardformat «Zahl» zum Format «Text» gewechselt wird.

Bedingte Anzeige

Gruppenköpfe, Gruppenfüße, Felder – in sämtlichen Untergliederungen befindet sich unter den allgemeinen Eigenschaften das Feld **Ausdruck für bedingte Anzeige**. Formeln, die in dieses Feld geschrieben werden, beeinflussen den Inhalt eines Feldes oder gleich die Anzeige eines ganzen Bereiches. Auch hier steht der Funktions-Assistent zur Verfügung.

```
001 [Feldbezeichnung]="true"
```

sorgt dafür, dass der Inhalt nur dann angezeigt wird, wenn der Inhalt des Feldes «Feldbezeichnung» wahr ist.[15] Dies funktioniert auch bei grafischen Steuerelementen.

Grafische Formen reagieren zur Zeit nicht auf Einträge in der bedingten Anzeige. Soll z. B. eine farbige Trennlinie nach dem 10. Platz einer Wettkampfliste eingezogen werden, so geht dies nicht, indem der grafischen Form über die bedingte Anzeige mitgegeben wird

```
001 [Platz]=10
```

Dieser Befehl wirkt nicht auf die Grafik. Sie erscheint in dem Abschnitt «Detail» dann weiter nach jedem Datensatz.

Dieser Bug ist als *Bug 73707* gemeldet.

Soll lediglich eine rechteckige Form bedingt an dieser Stelle eingeblendet werden, so geht dies aber über ein grafische Steuerelement, das mit einer entsprechenden (eventuell einfarbigen) Grafikdatei angesprochen wird. Bei den allgemeinen Eigenschaften wird **Skalieren → Autom. Größe** gewählt. Dann passt sich die Grafik der Form an und die Bedingung greift.

Sicherer ist es, die bedingte Anzeige an einen *Gruppenfuß* statt an die Grafik zu binden, sofern dieser nicht anderweitig benötigt wird. Die Linie wird im *Gruppenfuß* positioniert. Der *Gruppenfuß* wird mit der Bedingung versehen. Dann erscheint die Linie auch tatsächlich nach dem 10. Platz, wenn sie wie oben formuliert wird. Dadurch, dass der Gruppenfuß mit der Bedingung versehen wird, nimmt die Linie auch nur dann den entsprechenden Platz ein, wenn sie wirklich benötigt wird. Sonst erfolgt durch die bedingte Anzeige die Ausgabe eines Leerraumes statt der Linie.

Hier lassen sich dann auch neben den Formen aus **Einfügen → Formen → Standardformen** z. B. die horizontalen Linien mit dem Gruppenfuß ausblenden.

15 Siehe hierzu die Datenbank «Beispiel_Bericht_bedingte_Einblendung_von_Grafiken.odb», die diesem Handbuch beiliegt.

Hinweis

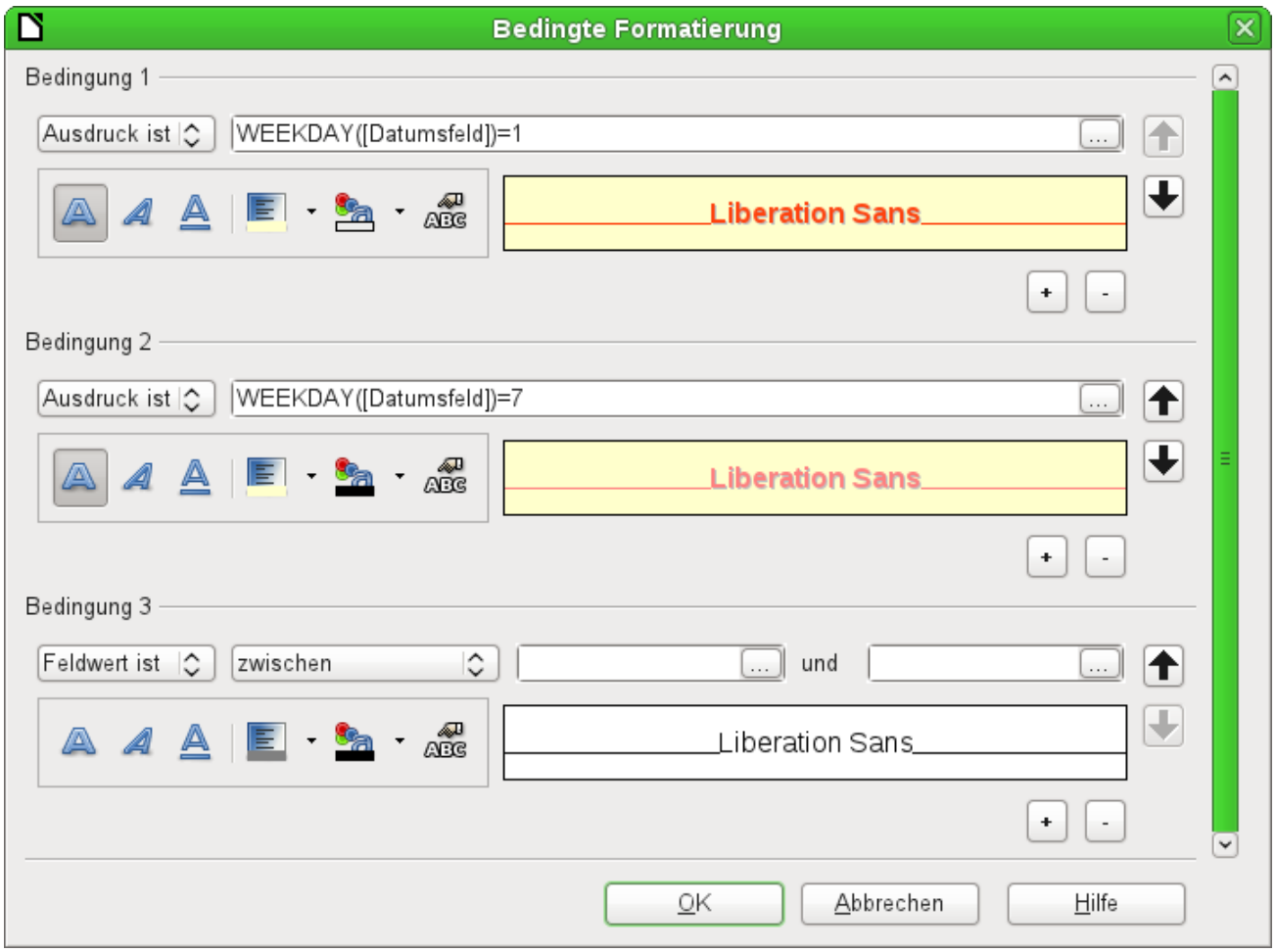
Manchmal scheint es wünschenswert, eine bedingte Anzeige in Abhängigkeit von der Seitenzahl zu erstellen. Die Funktionen PageNumber() und PageCount() können für irgendwelche Formatierungshinweise zur bedingten Anzeige allerdings nicht genutzt werden, da die Werte erst innerhalb des Writer-Dokumentes erstellt werden.
Es scheint momentan überhaupt keine Möglichkeit zu geben, das Erscheinen des Seitenkopfes oder Seitenfußes über die dort mögliche bedingte Anzeige zu steuern.

Bedingte Formatierung

Bei der bedingten Formatierung kann z. B. ein Kalender so formatiert werden, dass die Wochenenden besonders gekennzeichnet werden. Unter **Format → bedingte Formatierung** ist dann einzutragen

```
001 WEEKDAY([Datumsfeld])=1
```

sowie die entsprechende Formatierung für den Sonntag.



Wird in der bedingten Formatierung «Ausdruck ist» gewählt, so kann eine Formel eingegeben werden. Wie auch in Calc üblich können mehrere Bedingungen formuliert werden, die nacheinander abgearbeitet werden. Im obigen Beispiel wird so zuerst der Sonntag abgefragt und dann der Samstag. Zum Schluss könnte dann noch eine Abfrage nach dem Inhalt des Feldes kommen. So könnte z. B. der Inhalt 'Urlaub' mit einer entsprechend anderen Formatierung angezeigt werden.

Hinweis

Der Report-Designer ist ein Addon. Tauchen anscheinend nicht behebbare Fehler auf (Formel wird nicht umgesetzt, zu langer Text wird als ein leeres Feld angezeigt ...), so empfiehlt es sich manchmal, Teile des Berichts zu löschen oder einfach den Bericht neu zu erstellen.

Beispiele für Berichte mit dem Report-Designer

Der Report-Designer birgt einige Tücken in der Anwendung, da bestimmte Funktionen zwar grundsätzlich vorgesehen sind, aber zur Zeit nicht richtig funktionieren. Auch gibt es eine nur sehr geringe Hilfestellung innerhalb der Hilfefunktion von LibreOffice. Deshalb hier einige recht komplexe Beispiele, wie der Report-Designer für verschiedene Berichtstypen genutzt werden kann. Zum Einstieg in den Report-Designer sei hier noch auf das Einführungskapitel dieses Handbuches verwiesen. Dort wird auf mehreren Seiten Schritt für Schritt ein Bericht erstellt.

Rechnungserstellung

Folgende Vorgaben sollen für die Erstellung der Rechnung[16] erfüllt werden:

- Die einzelnen Rechnungspositionen sollen durchnummeriert werden.
- Rechnungen, die mehr als eine Seite umfassen, sollen mit einer Seitennummerierung versehen werden.
- Rechnungen, die mehr als eine Seite umfassen, sollen auf jeder Seite eine Zwischensumme bilden und auf der darauffolgenden Seite diese Zwischensumme als Übertrag darstellen.

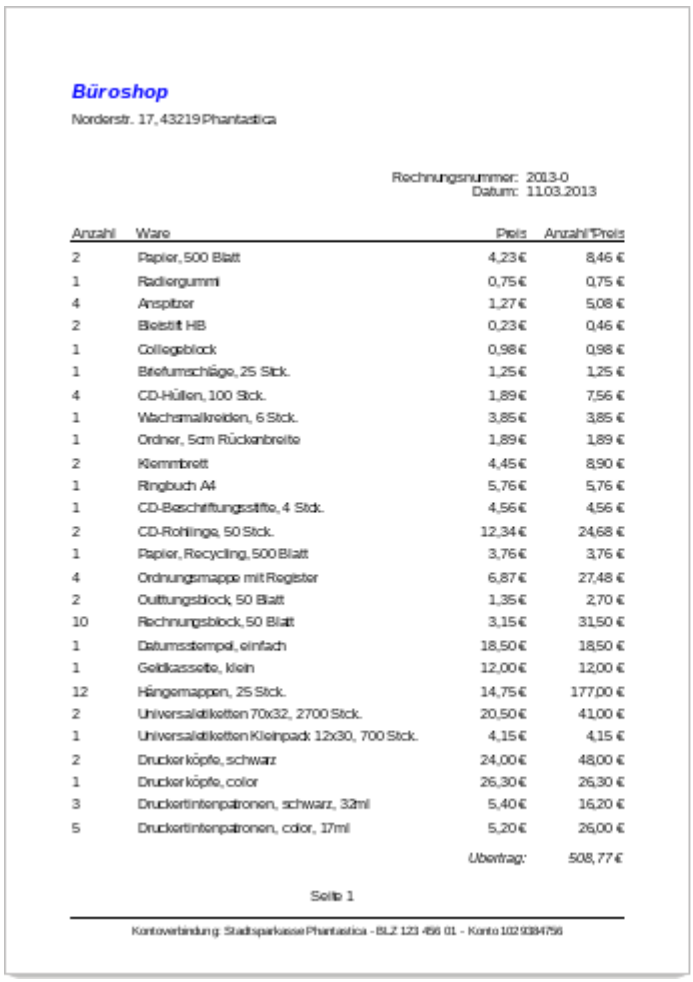
Mehrere aktuelle existierende Bugs scheinen das Verfahren unmöglich zu machen:

- *Bug 51452*: Wird eine Gruppierung auf «Bereich wiederholen» eingestellt, so wird automatisch vor und hinter der Gruppierung ein Seitenumbruch eingefügt.
- *Bug 51453*: Gruppierungen mit neu startender Seitenzählung sind zwar eigentlich vorgesehen, funktionieren aber nicht.
- *Bug 51959*: Ein Gruppenfuß lässt sich nicht wiederholen. Er kann nur am Ende einer Gruppe auftauchen, nicht z. B. am Ende einer jeden Seite. Wird er auf «Bereich wiederholen» gesetzt, so verschwindet er komplett.

Außerdem gibt es noch Schwierigkeiten, Linien in den Bericht einzufügen. Die vorgesehenen horizontalen und vertikalen Linien werden erst ab den Versionen LO 4.0.5 bzw. 4.1.1 angezeigt. Als Ersatz dafür können Rechtecke genutzt werden. Die lassen sich aber nicht richtig positionieren, wenn in einem Bereich ein Seitenumbruch stattfindet.

Der Bericht in einer einfachen Form sollte also so aussehen:

16 Die Datenbank «Beispiel_Bericht_Rechnung.odb» ist den Beispieldatenbanken für dieses Handbuch beigefügt. Eine komplette Beschreibung der Datenbank ist in «Base_Beispiele.pdf» verfügbar.



Büroshop

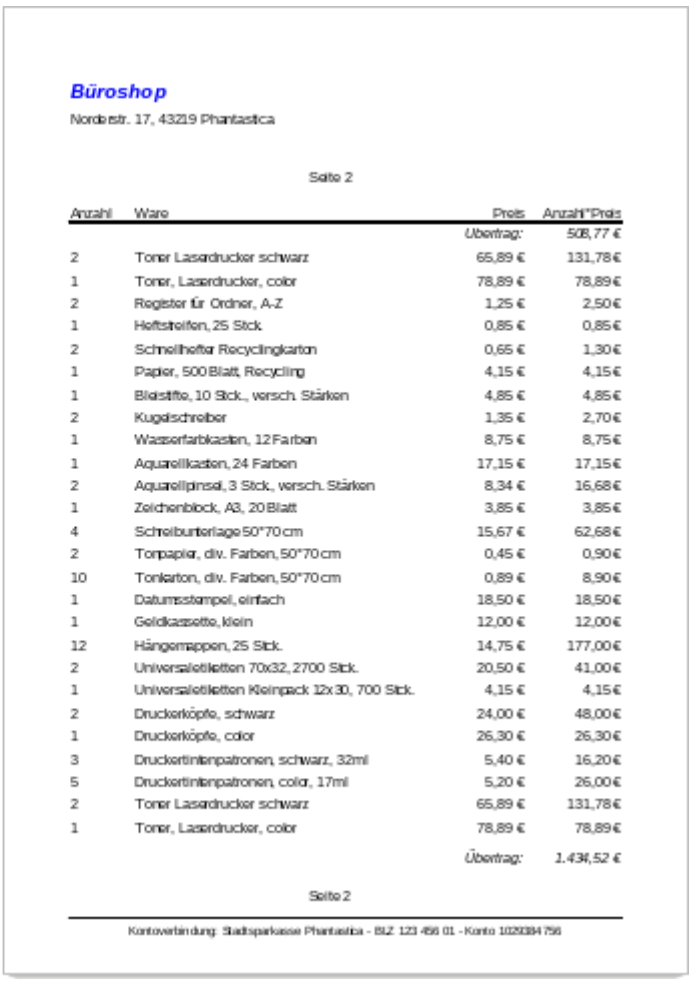
Norderstr. 17, 43219 Phantastica

Rechnungsnummer: 2013-0
Datum: 11.03.2013

Anzahl	Ware	Preis	Anzahl*Preis
2	Papier, 500 Blatt	4,23 €	8,46 €
1	Radiergummi	0,75 €	0,75 €
4	Anspitzer	1,27 €	5,08 €
2	Bleistift HB	0,23 €	0,46 €
1	Collegeblock	0,98 €	0,98 €
1	Briefumschläge, 25 Stck.	1,25 €	1,25 €
4	CD-Hüllen, 100 Stck.	1,89 €	7,56 €
1	Wachsmalkreiden, 6 Stck.	3,85 €	3,85 €
1	Ordner, 5cm Rückenbreite	1,89 €	1,89 €
2	Klemmbrett	4,45 €	8,90 €
1	Ringbuch A4	5,76 €	5,76 €
1	CD-Beschriftungsstifte, 4 Stck.	4,56 €	4,56 €
2	CD-Rohlinge, 50 Stck.	12,34 €	24,68 €
1	Papier, Recycling, 500 Blatt	3,76 €	3,76 €
4	Ordnungsmappe mit Register	6,87 €	27,48 €
2	Quittungsblock, 50 Blatt	1,35 €	2,70 €
10	Rechnungsblock, 50 Blatt	3,15 €	31,50 €
1	Datumsstempel, einfach	18,50 €	18,50 €
1	Geldkassette, klein	12,00 €	12,00 €
12	Hängemappen, 25 Stck.	14,75 €	177,00 €
2	Universaletiketten 70x32, 2700 Stck.	20,50 €	41,00 €
1	Universaletiketten Kleinpack 12x30, 700 Stck.	4,15 €	4,15 €
2	Druckerköpfe, schwarz	24,00 €	48,00 €
1	Druckerköpfe, color	26,30 €	26,30 €
3	Druckertintenpatronen, schwarz, 32ml	5,40 €	16,20 €
5	Druckertintenpatronen, color, 17ml	5,20 €	26,00 €
		Übertrag:	*508,77 €*

Seite 1

Kontoverbindung: Stadtsparkasse Phantastica - BLZ 123 456 01 - Konto 1029384756



Büroshop

Norderstr. 17, 43219 Phantastica

Seite 2

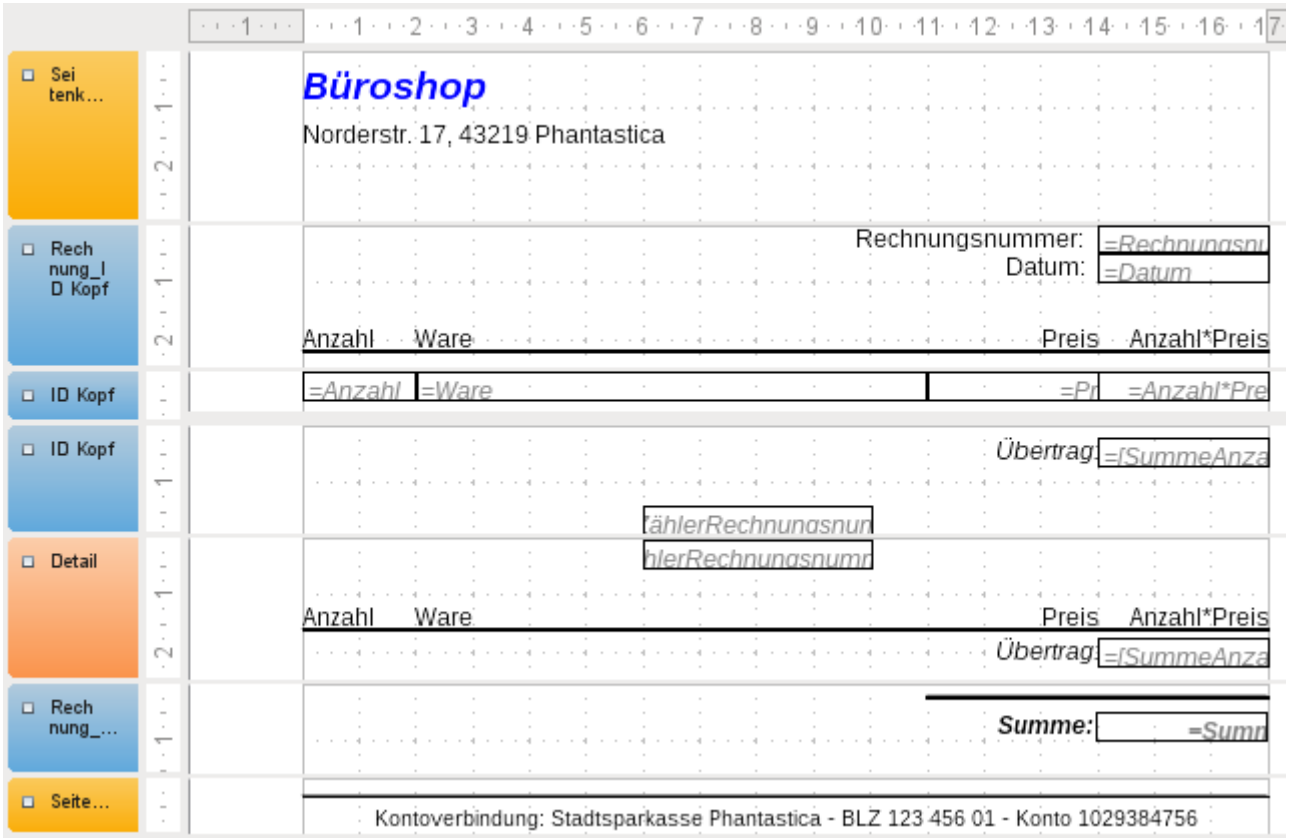
Anzahl	Ware	Preis	Anzahl*Preis
		Übertrag:	*508,77 €*
2	Toner Laserdrucker schwarz	65,89 €	131,78 €
1	Toner, Laserdrucker, color	78,89 €	78,89 €
2	Register für Ordner, A-Z	1,25 €	2,50 €
1	Heftstreifen, 25 Stck.	0,85 €	0,85 €
2	Schnellhefter Recyclingkarton	0,65 €	1,30 €
1	Papier, 500 Blatt, Recycling	4,15 €	4,15 €
1	Bleistifte, 10 Stck., versch. Stärken	4,85 €	4,85 €
2	Kugelschreiber	1,35 €	2,70 €
1	Wasserfarbkasten, 12 Farben	8,75 €	8,75 €
1	Aquarellkasten, 24 Farben	17,15 €	17,15 €
2	Aquarellpinsel, 3 Stck., versch. Stärken	8,34 €	16,68 €
1	Zeichenblock, A3, 20 Blatt	3,85 €	3,85 €
4	Schreibunterlage 50*70 cm	15,67 €	62,68 €
2	Tonpapier, div. Farben, 50*70 cm	0,45 €	0,90 €
10	Tonkarton, div. Farben, 50*70 cm	0,89 €	8,90 €
1	Datumsstempel, einfach	18,50 €	18,50 €
1	Geldkassette, klein	12,00 €	12,00 €
12	Hängemappen, 25 Stck.	14,75 €	177,00 €
2	Universaletiketten 70x32, 2700 Stck.	20,50 €	41,00 €
1	Universaletiketten Kleinpack 12x30, 700 Stck.	4,15 €	4,15 €
2	Druckerköpfe, schwarz	24,00 €	48,00 €
1	Druckerköpfe, color	26,30 €	26,30 €
3	Druckertintenpatronen, schwarz, 32ml	5,40 €	16,20 €
5	Druckertintenpatronen, color, 17ml	5,20 €	26,00 €
2	Toner Laserdrucker schwarz	65,89 €	131,78 €
1	Toner, Laserdrucker, color	78,89 €	78,89 €
		Übertrag:	*1.434,52 €*

Seite 2

Kontoverbindung: Stadtsparkasse Phantastica - BLZ 123 456 01 - Konto 1029384756

Um trotz der oben genannten Einschränkungen eine den Anforderungen entsprechende Rechnungserstellung durchführen zu können, muss genau mit den Seitenmaßen des auszudruckenden Dokumentes gearbeitet werden. In diesem Beispiel wird von einem DIN-A4-Ausgabeformat ausgegangen. Die Gesamthöhe eines Blattes beträgt also 29,7 cm.

Der Bericht muss dazu in mehrere Gruppen unterteilt werden. Zwei Gruppen beziehen sich auf das gleiche Tabellenfeld und enthalten jeweils nur einen entsprechenden Tabellenwert.



Die folgende Tabelle zeigt die Einteilung der Seite in unterschiedliche Bereich des Berichtes:

A	Seitenrand oben (nicht im Screenshot zu sehen, über **Format → Seite** einstellbar)	2,00 cm
B	Seitenkopf (erscheint auf jeder Seite, enthält keine Eingaben der Datenbank, sondern z. B. nur das Firmenlogo sowie die Adresse des Absenders)	3,00 cm
C	Gruppenkopf für die Rechnungsnummer (Nur die Beträge, die zu einer Rechnungsnummer gehören, sollen später auch addiert werden. Der Gruppenkopf erscheint nur zum Beginn der Rechnung.)	2,50 cm
D	Gruppenkopf für die Rechnungsposten (Der Bereich «Detail» wird für andere Inhalte benötigt. Deshalb erfolgt hier eine Gruppierung, die gleichzeitig die Sortierung der Rechnungsinhalte z. B. nach der Eingabe vornimmt. Diese Gruppe besteht immer nur aus einem Wert.)	0,70 cm
E	Gruppenkopf, ebenfalls an Rechnungsposten gebunden (Dieser Bereich wird nur dann angezeigt, wenn so viele Rechnungsposten vorkommen, dass auf jeden Fall eine weitere Rechnungsseite erforderlich ist. Er enthält die Zwischensumme und die Seitennummerierung unten auf der Seite. Nach diesem Bereich erfolgt ein Seitenumbruch.)	2,00 cm
F	Detail-Bereich (Dieser Bereich wird nur dann angezeigt, wenn so viele Rechnungsposten vorkommen, dass auf jeden Fall eine weitere Rechnungsseite erforderlich ist. Er enthält den Übertrag, die Seitennummer oben auf der Seite)	2,50 cm
G	Gruppenfuß für die Rechnungsnummer (Hier erfolgt die Ausgabe des Rechnungsbetrages, ggf. auch mit Angabe der Mehrwertsteuer. Der Gruppenfuß erscheint nur zum Schluss der Rechnung.)	1,60 cm
H	Seitenfuß (z. B. Angabe der Kontoverbindung)	1,00 cm
I	Seitenrand (wie «Seitenrand oben» nicht im Screenshot zu sehen, über **Format → Seite** einstellbar)	1,00 cm

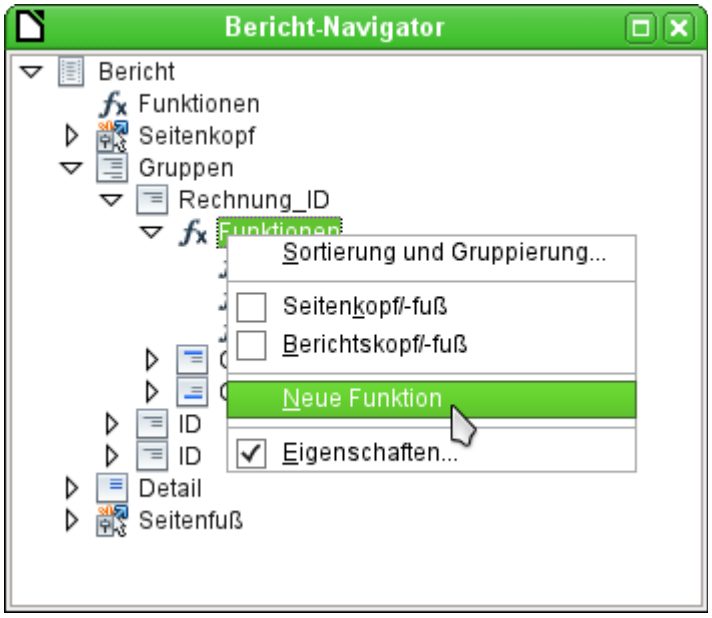
Ein Seitenumbruch soll nur dann erfolgen, wenn zu viele Rechnungsposten vorhanden sind. Für die Rechnungsposten ergibt sich der folgende freie Platz:

	29,70 cm	(DIN A 4)
-	2,00 cm	(Pos. A)
-	3,00 cm	(Pos. B)
-	2,50 cm	(Pos. C)
-	1,60 cm	(Pos. G)
-	1,00 cm	(Pos. H)
-	1,00 cm	(Pos. I)
=	**18,60 cm**	

Der verbleibende freie Platz kann also maximal 18,60 cm / 0,70 cm = 26,57..., abgerundet also 26 Rechnungspositionen erfassen.

Sobald die 27. Rechnungsposition vorkommt, muss auf jeden Fall ein Seitenumbruch erfolgen. Das bedeutet, dass dann auf jeden Fall Gruppenkopf E und der Detail-Bereich angezeigt werden müssen. Es muss also ein Zähler für die Rechnungsnummer (Bereich C) eingerichtet werden. Erreicht dieser Zähler die Nummer 27, so wird der Detail-Bereich (F) angezeigt.

Der Zähler der Rechnungsnummer wird wie folgt definiert:



Über den Berichtsnavigator wird die Gruppe «Rechnung_ID» aufgesucht. Die neu zu gründende Funktion heißt einfach «ZählerRechnungsnummer». Die Formel ist «[ZählerRechnungsnummer] + 1». Der Startwert ist 1. Es werden keine Unterberichte eingebunden (die Funktion gibt es gar nicht ...). Es wird auch nicht vorausberechnet. Für die Vorausberechnung wird ein gesonderter Zähler «ZählerKomplett» erstellt.

Der Gruppenkopf E und der Detailbereich F werden angezeigt, wenn insgesamt mehr als 26 Positionen in der Rechnung stehen und die momentane Rechnungsposition 26 erreicht hat. Der Ausdruck für die bedingte Anzeige ist also in beiden Bereichen

```
001 AND([ZählerRechnungsnummer]=26;[ZählerKomplett]>26)
```

Der Inhalt dieser Bereiche erscheint also nur, wenn mindestens ein 27. Rechnungsposten zu erwarten ist. Gruppenkopf E erscheint auf der ersten Seite. Er ist mit einem Seitenumbruch

nach dem Bereich versehen. Der Inhalt des Bereiches «Detail» wird auf der folgenden Seite angezeigt.

Jetzt ist zu berechnen, wie groß der Anteil des Bereiches ist, der noch auf der ersten Seite angezeigt wird:

	29,70 cm	(DIN A 4)
-	2,00 cm	(Pos. A)
-	3,00 cm	(Pos. B)
-	2,50 cm	(Pos. C)
-	18,20 cm	(Pos. D * 26)
-	1,00 cm	(Pos. H)
-	1,00 cm	(Pos. I)
=	**2,00 cm**	

Der Gruppenfuß fällt aus der ersten Seite raus, insgesamt 26 Rechnungsposten sind enthalten. Der Gruppenkopf E kann also maximal 2 cm auf der ersten Seite anzeigen. In diesen 2 cm muss auf jeden Fall die Zwischensumme und die Seitenzahl untergebracht werden. Um auf jeden Fall einen korrekten Seitenumbruch zu erzeugen, sollte der Bereich also etwas kleiner sein. In dem Beispiel wurde er auf 1,90 cm eingestellt.

Der Bereich «Detail» wird auf der Folgeseite oben angezeigt. Da auf der Folgeseite der Gruppenkopf der Rechnungsnummer (C) nicht mehr erscheint, kann hier der Bereich «Detail» so viel Platz einnehmen wie der Gruppenkopf, nämlich 2,50 cm. Dann starten die darauffolgenden Rechnungsposten wieder mit der gleichen Einstellung wie auf der Vorseite.

Der Übertrag wird jeweils als einfache Summierung der vorhergehenden Posten erreicht.

Über den Berichtsnavigator wird die Gruppe «Rechnungsnummer» aufgesucht. Die neu zu gründende Funktion heißt einfach «SummePreis». Die Formel ist «[Preis] + [SummePreis]». Der Startwert ist [Preis]. Es werden keine Unterberichte eingebunden. Es wird auch nicht vorausberechnet.

Der Übertrag wird in dem Gruppenkopf E und dem Detailbereich F angezeigt. Im Gruppenkopf E ist der Übertrag ganz oben positioniert. Er erscheint auf der ersten Seite unten. Im Detailbereich F ist der Übertrag ganz unten positioniert. Er erscheint auf der zweiten Seite direkt unterhalb der Tabellenköpfe.

Die Seitennummerierung wird ähnlich abgefragt wie die Anzeige des Gruppenkopfes E des Bereiches «Detail».

```
001 IF([ZählerRechnungsnummer]=26;"Seite 1";"")
```

Hiermit wird die Seitennummer auf der ersten Seite erstellt. Weitere IF-Abfragen können dann für die weiteren Seitennummern eingebunden werden.

Die Seitennummer für die Folgeseite wird bei der gleichen Bedingung einfach auf "Seite 2" gesetzt.

Werden die Formeln entsprechend weitgehender formuliert, so können beliebig viele Seiten des Berichtes abgedeckt werden.

Der Ausdruck für die **bedingte Anzeige** wechselt also von

```
001 AND([ZählerRechnungsnummer]=26;[ZählerKomplett]>26)
```

zu

```
001 AND(MOD([ZählerRechnungsnummer];26)=0;
    [ZählerKomplett]>[ZählerRechnungsnummer])
```

Der Gruppenkopf E und der Detailbereich F erscheinen also nur, wenn sich aus der Division des Zählers der Rechnungsnummer mit 26 kein Rest ergibt und die Komplettzahl der Rechnungsposten größer ist als der Zähler der Rechnungsnummer.

Der Ausdruck für die **Seitenzahl** wechselt von

```
001 IF([ZählerRechnungsnummer]=26;"Seite 1";"")
```

zu

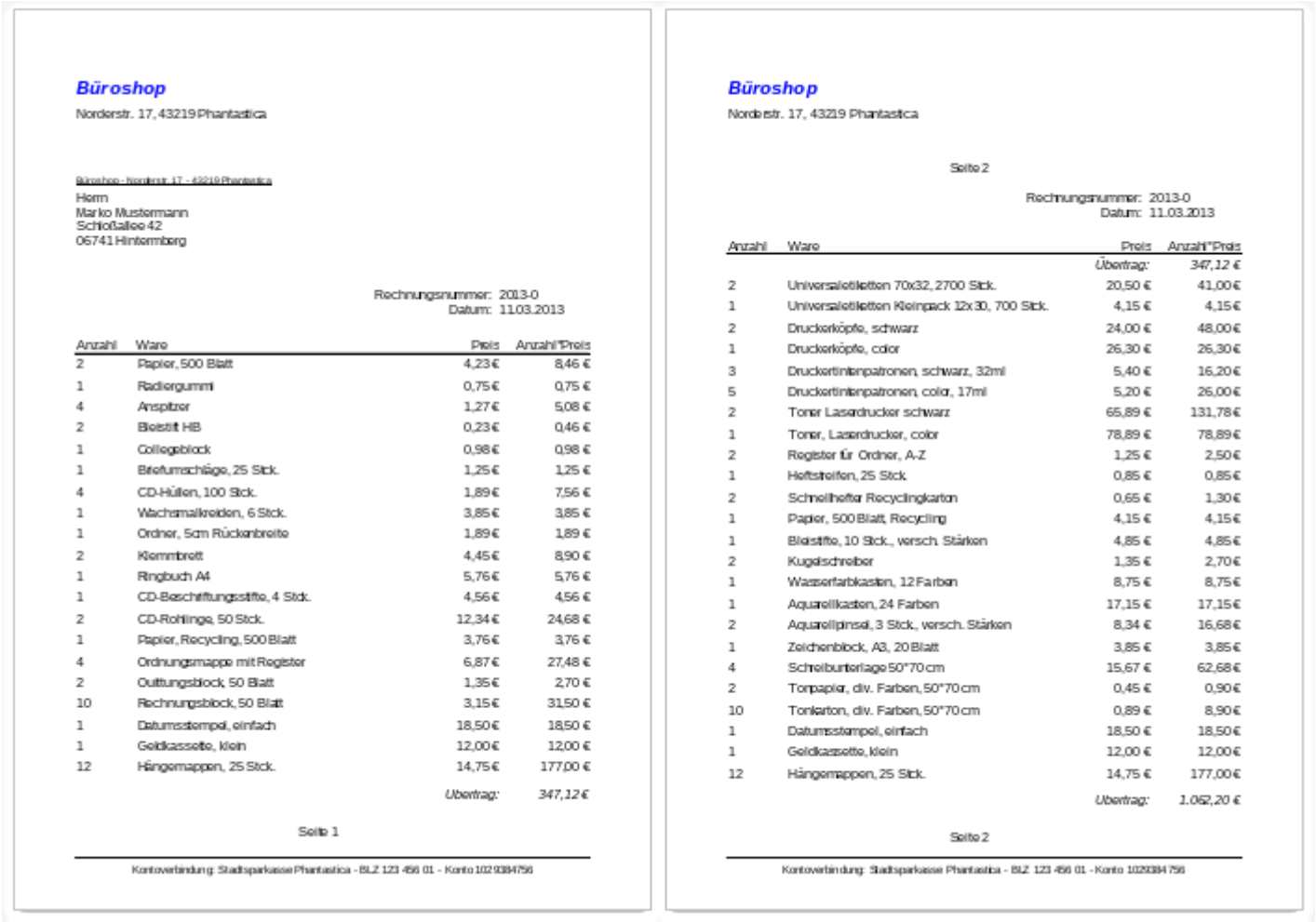
```
001 "Seite "&[ZählerRechnungsnummer]/26
```

für die aktuelle Seite bzw.

```
001 "Seite "&([ZählerRechnungsnummer]/26)+1
```

als Anzeige für die Folgeseite.

Der folgende Berichtsausdruck ist mit diesen Einstellungen noch nicht zu bewerkstelligen:



Büroshop

Norderstr. 17, 43219 Phantastica

Büroshop - Norderstr. 17 - 43219 Phantastica
Herrn
Marko Mustermann
Schloßallee 42
06741 Hintermoberg

Rechnungsnummer: 2013-0
Datum: 11.03.2013

Anzahl	Ware	Preis	Anzahl*Preis
2	Papier, 500 Blatt	4,23 €	8,46 €
1	Radiergummi	0,75 €	0,75 €
4	Anspitzer	1,27 €	5,08 €
2	Bleistift HB	0,23 €	0,46 €
1	Collegeblock	0,98 €	0,98 €
1	Briefumschläge, 25 Stck.	1,25 €	1,25 €
4	CD-Hüllen, 100 Stck.	1,89 €	7,56 €
1	Wachsmalkreiden, 6 Stck.	3,85 €	3,85 €
1	Ordner, 5cm Rückenbreite	1,89 €	1,89 €
2	Klemmbrett	4,45 €	8,90 €
1	Ringbuch A4	5,76 €	5,76 €
1	CD-Beschriftungsstifte, 4 Stck.	4,56 €	4,56 €
2	CD-Rohlinge, 50 Stck.	12,34 €	24,68 €
1	Papier, Recycling, 500 Blatt	3,76 €	3,76 €
4	Ordnungsmappe mit Register	6,87 €	27,48 €
2	Quittungsblock, 50 Blatt	1,35 €	2,70 €
10	Rechnungsblock, 50 Blatt	3,15 €	31,50 €
1	Datumsstempel, einfach	18,50 €	18,50 €
1	Geldkassette, klein	12,00 €	12,00 €
12	Hängemappen, 25 Stck.	14,75 €	177,00 €
		Übertrag:	*347,12 €*

Seite 1

Kontoverbindung: Stadtsparkasse Phantastica - BLZ 123 456 01 - Konto 102 9384 756

Büroshop

Norderstr. 17, 43219 Phantastica

Seite 2

Rechnungsnummer: 2013-0
Datum: 11.03.2013

Anzahl	Ware	Preis	Anzahl*Preis
		Übertrag:	*347,12 €*
2	Universaletiketten 70x32, 2700 Stck.	20,50 €	41,00 €
1	Universaletiketten Kleinpack 12x30, 700 Stck.	4,15 €	4,15 €
2	Druckerköpfe, schwarz	24,00 €	48,00 €
1	Druckerköpfe, color	26,30 €	26,30 €
3	Druckertintenpatronen, schwarz, 32ml	5,40 €	16,20 €
5	Druckertintenpatronen, color, 17ml	5,20 €	26,00 €
2	Toner Laserdrucker schwarz	65,89 €	131,78 €
1	Toner, Laserdrucker, color	78,89 €	78,89 €
2	Register für Ordner, A-Z	1,25 €	2,50 €
1	Heftstreifen, 25 Stck.	0,85 €	0,85 €
2	Schnellhefter Recyclingkarton	0,65 €	1,30 €
1	Papier, 500 Blatt, Recycling	4,15 €	4,15 €
1	Bleistifte, 10 Stck., versch. Stärken	4,85 €	4,85 €
2	Kugelschreiber	1,35 €	2,70 €
1	Wasserfarbkasten, 12 Farben	8,75 €	8,75 €
1	Aquarellkasten, 24 Farben	17,15 €	17,15 €
2	Aquarellpinsel, 3 Stck., versch. Stärken	8,34 €	16,68 €
1	Zeichenblock, A3, 20 Blatt	3,85 €	3,85 €
4	Schreibunterlage 50*70 cm	15,67 €	62,68 €
2	Tonpapier, div. Farben, 50*70 cm	0,45 €	0,90 €
10	Tonkarton, div. Farben, 50*70 cm	0,89 €	8,90 €
1	Datumsstempel, einfach	18,50 €	18,50 €
1	Geldkassette, klein	12,00 €	12,00 €
12	Hängemappen, 25 Stck.	14,75 €	177,00 €
		Übertrag:	*1.062,20 €*

Seite 2

Kontoverbindung: Stadtsparkasse Phantastica - BLZ 123 456 01 - Konto 102 9384 756

Durch das Adressfeld sind auf der ersten Seite der Rechnung weniger Rechnungsposten enthalten als auf der zweiten Seite. Der Bereich «Detail», der auf der zweiten Seite ganz oben abgebildet ist, ist also deutlich kleiner als der Gruppenkopf für die Rechnungsnummer (C).

Um unterschiedliche Mengen an Rechnungsposten auf der ersten Seite und den Folgeseiten zuzulassen, müssen also die Formeln noch einmal angepasst werden.

Die folgende Berechnung sichert ab, dass die entsprechenden Bereiche korrekt angezeigt werden.

```
001 AND
002 (MOD([ZählerRechnungsnummer]-20;24)=0;
003 [ZählerRechnungKomplett]>[ZählerRechnungsnummer])
```

Vom Zähler für die Rechnungsnummer wird die Anzahl der Rechnungsposten auf der ersten Seite subtrahiert. Diese Differenz wird durch die mögliche Anzahl der Rechnungsposten auf der zweiten Seite dividiert. Wenn die Division glatt aufgeht (kein Rest, MOD = 0), dann ist die erste Bedingung für die Anzeige von Gruppenkopf E und der Detailbereich F gegeben. Außerdem muss, wie bereits vorher definiert, der Zähler für die Rechnungsnummer kleiner sein als die insgesamt zu erwartenden Rechnungsposten. Sonst würde ja die Rechnungssumme noch genug Platz auf der entsprechenden Seite haben.

Die mögliche Anzahl der Rechnungsposten auf der zweiten Seite wurde dadurch geringer, dass jetzt auf der zweiten Seite zusätzlich die Rechnungsnummer und das Datum enthalten sind.

Die Seitenzahl wird jetzt etwas einfacher berechnet:

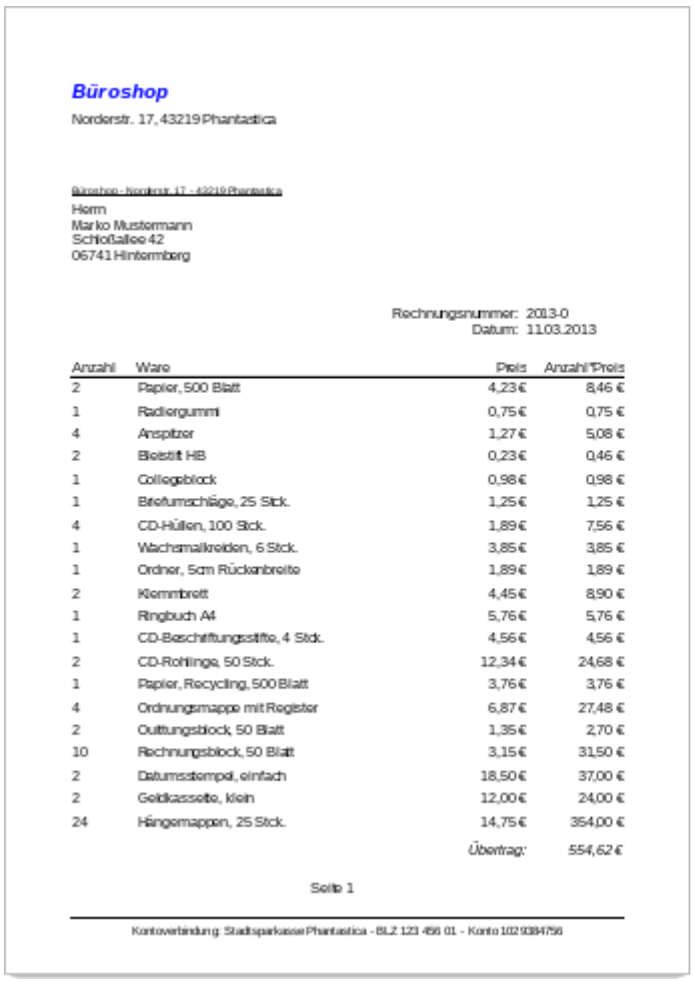
```
 001 "Seite "&INT([ZählerRechnungsnummer]/24)+1
```

INT rundet auf die nächste Ganzzahl ab. Die erste Seite enthält maximal 20 Rechnungsposten. Die Division ergibt für die erste Seite also ein Ergebnis < 1. Abgerundet wird das zu 0. Also muss zu der ermittelten Seitenzahl 1 addiert werden, damit auf der ersten Seite auch 1 erscheint. Entsprechend muss für die zweite Seite 2 addiert werden.

Der Bericht weist in der obigen Form noch einen Schönheitsfehler auf. Ein genauer Blick auf die Rechnungsposten zeigt, dass sich die unteren drei Rechnungsposten gleichen. Dies wurde hier durch einfaches Kopieren der Datensätze erzeugt. Es handelt sich also nicht um die gleichen Datensätze, sondern unterschiedliche Rechnungspositionen, die nacheinander über die Software abgearbeitet wurden. Besser wäre hier, die Waren entsprechend gruppiert anzuzeigen, damit die gleiche Ware nicht mehrmals, sondern nur einmal mit entsprechender Anzahl aufgelistet wird.

Grundsätzlich sollten möglichst viele Berechnungen, Gruppierungen usw. aus dem Report-Designer ausgelagert werden. Deshalb wird für eine entsprechende Gruppierung nicht mit den Gruppen des Report-Designers gearbeitet, sondern mit den Gruppierungsfunktionen des Abfrageeditors. Damit der Report-Designer die Abfrage einwandfrei verarbeiten kann, wird schließlich daraus eine Ansicht gemacht. Der Report-Designer versucht ansonsten, die Abfrage weiter durch seine Gruppierungsvorgaben und Sortierungsvorgaben zu ergänzen, was sehr schnell zu unbrauchbarem Code führt.

Dann ergibt sich die folgende Rechnungsübersicht:



Büroshop

Norderstr. 17, 43219 Phantastica

Büroshop - Norderstr. 17 - 43219 Phantastica

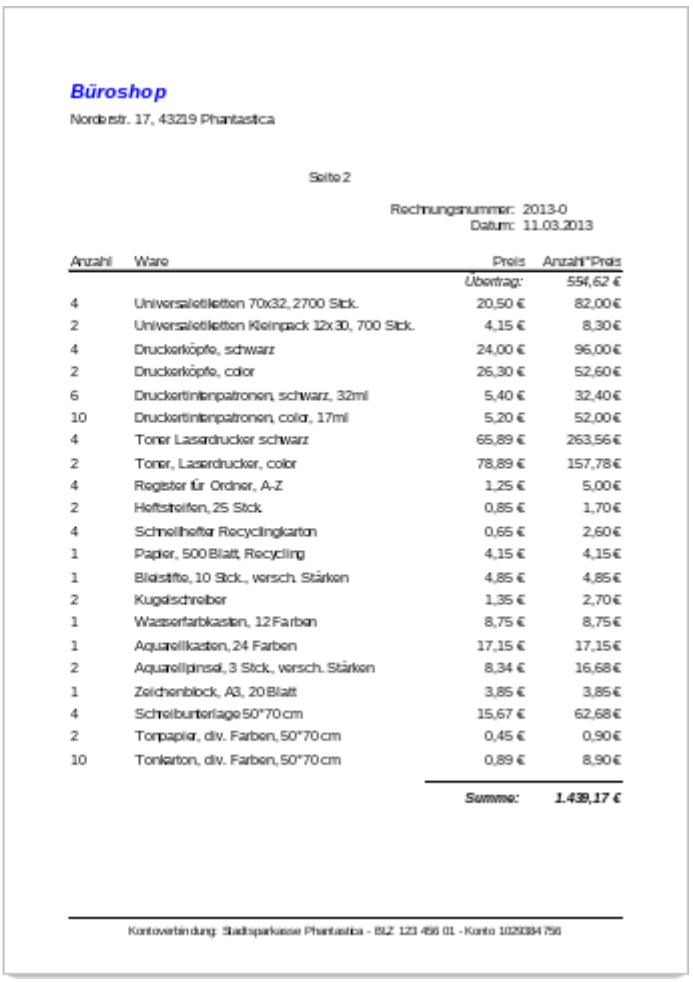
Herrn
Marko Mustermann
Schloßallee 42
06741 Hintermberg

Rechnungsnummer: 2013-0
Datum: 11.03.2013

Anzahl	Ware	Preis	Anzahl*Preis
2	Papier, 500 Blatt	4,23 €	8,46 €
1	Radiergummi	0,75 €	0,75 €
4	Anspitzer	1,27 €	5,08 €
2	Bleistift HB	0,23 €	0,46 €
1	Collegeblock	0,98 €	0,98 €
1	Briefumschläge, 25 Stck.	1,25 €	1,25 €
4	CD-Hüllen, 100 Stck.	1,89 €	7,56 €
1	Wachsmalkreiden, 6 Stck.	3,85 €	3,85 €
1	Ordner, 5cm Rückenbreite	1,89 €	1,89 €
2	Klemmbrett	4,45 €	8,90 €
1	Ringbuch A4	5,76 €	5,76 €
1	CD-Beschriftungsstifte, 4 Stck.	4,56 €	4,56 €
2	CD-Rohlinge, 50 Stck.	12,34 €	24,68 €
1	Papier, Recycling, 500 Blatt	3,76 €	3,76 €
4	Ordnungsmappe mit Register	6,87 €	27,48 €
2	Quittungsblock, 50 Blatt	1,35 €	2,70 €
10	Rechnungsblock, 50 Blatt	3,15 €	31,50 €
2	Datumsstempel, einfach	18,50 €	37,00 €
2	Geldkassette, klein	12,00 €	24,00 €
24	Hängemappen, 25 Stck.	14,75 €	354,00 €
		Übertrag:	*554,62 €*

Seite 1

Kontoverbindung: Stadtsparkasse Phantastica - BLZ 123 456 01 - Konto 1023384756



Büroshop

Norderstr. 17, 43219 Phantastica

Seite 2

Rechnungsnummer: 2013-0
Datum: 11.03.2013

Anzahl	Ware	Preis	Anzahl*Preis
		Übertrag:	*554,62 €*
4	Universaletiketten 70x32, 2700 Stck.	20,50 €	82,00 €
2	Universaletiketten Kleinpack 12x30, 700 Stck.	4,15 €	8,30 €
4	Druckerköpfe, schwarz	24,00 €	96,00 €
2	Druckerköpfe, color	26,30 €	52,60 €
6	Druckertintenpatronen, schwarz, 32ml	5,40 €	32,40 €
10	Druckertintenpatronen, color, 17ml	5,20 €	52,00 €
4	Toner Laserdrucker schwarz	65,89 €	263,56 €
2	Toner, Laserdrucker, color	78,89 €	157,78 €
4	Register für Ordner, A-Z	1,25 €	5,00 €
2	Heftstreifen, 25 Stck.	0,85 €	1,70 €
4	Schnellhefter Recyclingkarton	0,65 €	2,60 €
1	Papier, 500 Blatt, Recycling	4,15 €	4,15 €
1	Bleistifte, 10 Stck., versch. Stärken	4,85 €	4,85 €
2	Kugelschreiber	1,35 €	2,70 €
1	Wasserfarbkasten, 12 Farben	8,75 €	8,75 €
1	Aquarellkasten, 24 Farben	17,15 €	17,15 €
2	Aquarellpinsel, 3 Stck., versch. Stärken	8,34 €	16,68 €
1	Zeichenblock, A3, 20 Blatt	3,85 €	3,85 €
4	Schreibunterlage 50*70 cm	15,67 €	62,68 €
2	Tonpapier, div. Farben, 50*70 cm	0,45 €	0,90 €
10	Tonkarton, div. Farben, 50*70 cm	0,89 €	8,90 €
		Summe:	***1.439,17 €***

Kontoverbindung: Stadtsparkasse Phantastica - BLZ 123 456 01 - Konto 1023384756

Alle Rechnungsposten sind nur einmal enthalten. Aus den ursprünglich auf Seite 1 unten stehenden 12 Hängemappen wurden in der Rechnung jetzt 24 Hängemappen. "Anzahl*Preis" wurde entsprechend angepasst.

Ausdruck von Berichten zum aktuellen Datensatz des Formulars

Gerade bei der Rechnungserstellung wie in dem vorhergehenden Beispiel kann es sinnvoll sein, nach jeder Eingabe der Rechnungsposten einen entsprechenden Ausdruck anzufertigen. In einem Formular soll der Inhalt für die Rechnung bestimmt werden und schließlich der Ausdruck des Einzeldokumentes erfolgen.

Berichte lassen sich nicht über Makros mit einem Filter starten. Allerdings kann die Abfrage, die als Grundlage für den Bericht genutzt wird, vorher gefiltert werden. Dies geht entweder über die Form einer Parameterabfrage

```
001 SELECT * FROM "Rechnung"
002 WHERE "ID" = :ID
```

oder über eine Abfrage, die über eine einzeilige Filtertabelle mit Daten versorgt wird:

```
001 SELECT * FROM "Rechnung"
002 WHERE "ID" = (SELECT "Integer" FROM "Filter" WHERE "ID" = TRUE)
```

Bei der Parameterabfrage muss der Inhalt in ein entsprechendes Dialogfeld nach dem Start des Berichtes eingegeben werden.

Bei der Steuerung über eine Filtertabelle wird der Inhalt der Filtertabelle per Makro geschrieben. Eine separate Eingabe ist also nicht mehr nötig. Diese Möglichkeit ist also für den Nutzer einfacher zu handhaben und soll deshalb im Folgenden beschrieben werden.

Aufbau der Filtertabelle

Die Filtertabelle soll lediglich einen Datensatz enthalten. Deshalb kann das Primärschlüsselfeld ein «Ja/Nein»-Feld sein. Andere Felder der Tabelle werden von der Benennung so gewählt, dass bereits eindeutig ist, was für einen Inhalt sie speichern können. In diesem Beispiel heißt das Feld, das den Primärschlüssel der Tabelle "Rechnung" filtern soll, "Integer", da der Primärschlüssel der Tabelle "Rechnung" eben vom Feldtyp «Integer» ist. Für andere Filterungen können andere Felder zusätzlich eingebaut werden. Der Filter "Integer" kann auch für mehrere Tabellen genutzt werden, da ja vor dem Druck der alte Wert einfach mit dem aktuellen Wert überschrieben wird. Diese Doppelnutzung funktioniert so allerdings nur in einer Einzeldatenbank (Base zusammen mit der internen HSQLDB oder der internen Firebird). In einer Mehrbenutzerdatenbank könnte in dem Moment, in dem die Abfrage des Filters erfolgt, von einem anderen Nutzer der Filterwert in einer normalen Tabelle schon wieder geändert werden. Deswegen wird in Mehrbenutzerdatenbanken die VerbindungsID als Primärschlüssel "ID" genutzt.

Feldname	Feldtyp
ID	Ja/Nein [BOOLEAN]
Integer	Integer [INTEGER]

Diese Tabelle wird zum Start mit einem Datensatz gefüllt. Hierzu muss lediglich das Feld "ID" einmal markiert werden, so dass es den Wert «Ja» (oder in SQL «TRUE») hat.

Aufbau des Makros zum Start des gefilterten Berichtes

Das Formular muss für das Aufrufen eines einzelnen Berichtes an irgendeiner Stelle den Primärschlüssel der Tabelle "Rechnung" enthalten. Dieser Primärschlüsselwert wird ausgelesen und über ein Makro in die Tabelle "Filter" übertragen. Anschließend wird in dem Makro noch der gewünschte Bericht gestartet.

```
001 SUB Filtern_und_Drucken
002    DIM oDoc AS OBJECT
003    DIM oDrawpage AS OBJECT
004    DIM oForm AS OBJECT
005    DIM oFeld AS OBJECT
006    DIM oDatenquelle AS OBJECT
007    DIM oVerbindung AS OBJECT
008    DIM oSQL_Anweisung AS OBJECT
009    DIM stSQL AS STRING
010    oDoc = thisComponent
011    oDrawpage = oDoc.Drawpage
012    oForm = oDrawpage.Forms.getByName("MainForm")
013    oFeld = oForm.getByName("fmtID")
014    oDatenquelle = ThisComponent.Parent.CurrentController
015    If NOT (oDatenquelle.isConnected()) THEN
016       oDatenquelle.connect()
017    END IF
018    oVerbindung = oDatenquelle.ActiveConnection()
019    oSQL_Anweisung = oVerbindung.createStatement()
020    stSql = "UPDATE ""Filter"" SET ""Integer"" = '"+oFeld.GetCurrentValue()+
021       "' WHERE ""ID"" = TRUE"
022    oSQL_Anweisung.executeUpdate(stSql)
023    ThisDatabaseDocument.ReportDocuments.getByName("Rechnung").open
024 END SUB
```

Das Formular hat in diesem Beispiel den Namen "MainForm". Das Primärschlüsselfeld heißt "fmtID". Dieses Schlüsselfeld muss nicht sichtbar sein, um darauf mit dem Makro zugreifen zu können. Der Wert dieses Feldes wird ausgelesen und mit dem UPDATE-Befehl in die Tabelle "Filter" geschrieben. Anschließend wird der Bericht gestartet. Die Ansicht, auf der der Bericht beruht, wurde entsprechend um eine Bedingung erweitert:

```
001 … WHERE "Rechnung_ID" = COALESCE((SELECT "Integer" FROM "Filter" WHERE
    "ID" = TRUE),"Rechnung_ID") …
```

Es wird das Feld "Integer" ausgelesen. Falls dieses Feld keinen Wert enthält, wird stattdessen "Rechnung_ID" = "Rechnung_ID" gesetzt. Das bedeutet, dass alle Datensätze angezeigt werden – nicht nur der Filterdatensatz. So können also gegebenenfalls mit der gleichen Ansicht auch alle gespeicherten Rechnungen ausgedruckt werden.

Wechselnde Einfärbung von Zeilen

Tabellenzeilen in einem Bericht können manchmal beim Lesen dazu führen, dass der Blick um eine Zeile nach unten oder oben rutscht. Dagegen kann eine Einfärbung von mindestens einer Zeile helfen. Im folgenden Beispiel[17] werden die Zeilen einfach wechselseitig mit unterschiedlichen Farben eingefärbt. Der Bericht sieht dann wie folgt aus:

Geburtstagsliste

Kathrin	17.01.84
Sally	15.02.91
Mick	03.03.53
Hanne	13.04.70
Meike	13.04.71
Lara	23.04.85

Grundlage des Berichtes ist eine Abfrage mit Namen und Datumseinträgen. Es wurde aus der ursprünglichen Tabelle eine Abfrage erstellt, damit die Daten nach Monaten und Tagen sortiert sind, also die Geburtstagsreihenfolge im Jahr anzeigen. Dies geht über

```
001 … ORDER BY
002 MONTH("Geburtstag") ASC,
003 DAY("Geburtstag") ASC    (HSQLDB, ~~FIREBIRD~~)
```

Damit der Code sowohl in der HSQLDB als auch in FIREBIRD läuft, muss er folgendermaßen angepasst werden:

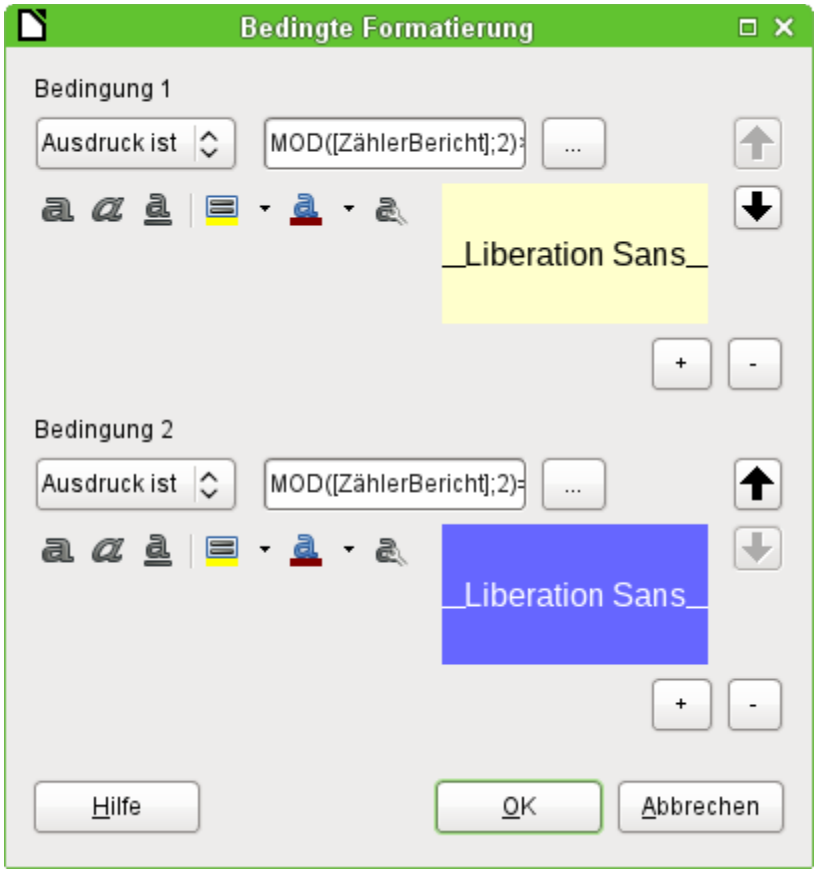
```
001 … ORDER BY
002 EXTRACT(MONTH FROM "Geburtstag") ASC,
003 EXTRACT(DAY FROM "Geburtstag") ASC      (HSQLDB, FIREBIRD)
```

Um eine wechselseitige Einfärbung zu erreichen muss irgendeine Funktion erstellt werden, aus der heraus über einen Wert später die Bedingung für die Einfärbung abgeleitet werden kann. Hier wird ein Textfeld im Bericht aufgezogen und über die **Eigenschaften → Daten → Datenfeld-Typ** ein Zähler definiert.

Der Name der Funktion wird für die bedingte Formatierung benötigt. Der Zähler selbst braucht beim Ausdruck nicht zu erscheinen. Der Name kann direkt aus dem aufgezogenen Feld abgelesen werden. Ist das Feld wieder gelöscht, so ist der Funktionsname weiterhin über **Ansicht → Berichts-Navigator** nachschlagbar.

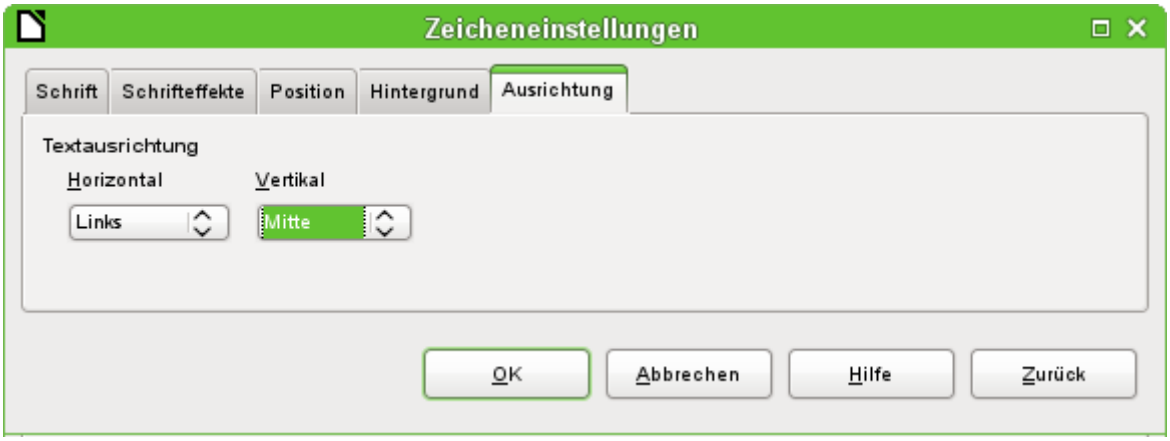
17 Die Datenbank «Beispiel_Bericht_Zeilen_Farbwechsel_Spalten.odb» zu diesem Bericht ist den Beispieldatenbanken für dieses Handbuch beigefügt.

Jetzt muss jedem der Textfelder über den Zähler ein entsprechendes Format zugewiesen werden. Die Bedingung ist jedes Mal ein Ausdruck, der nicht direkt mit dem Feld zusammen hängt. Deshalb wird als **Bedingung 1 → Ausdruck ist → `MOD([ZählerBericht];2)>0`** eingestellt. MOD berechnet den Rest einer Division. Bei allen ungeraden Zahlen ist der Rest größer als 0, bei allen Geraden Zahlen ist der Rest 0. Zeile 1, 3, 5 usw. wird also mit dem entsprechenden Format versehen.



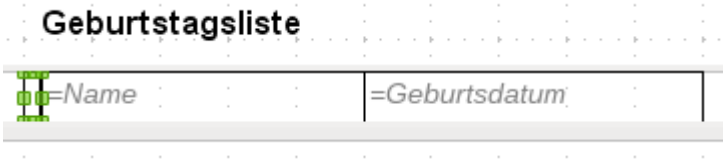
Als zweite Bedingung wird jetzt das Gegenteil von der ersten Bedingung formuliert und eine entsprechende Formatierung zugewiesen. Diese zweite Bedingung könnte auch ausgelassen werden und stattdessen in den Eigenschaften eines jeden Feldes ein entsprechendes Format eingestellt werden. Die bedingte Formatierung der ersten Bedingung würde dann nur die Standardformatierung ersetzen, wenn die erste Bedingung zutrifft.

Da die bedingte Formatierung alle Formate der Standardformatierung überschreibt, muss z. B. eine Ausrichtung der Schrift in den Zeicheneinstellungen der bedingten Formatierung erfolgen:



Hier wird die Schrift vertikal mittig in dem eingefärbten Textfeld dargestellt. Horizontal lassen sich die üblichen Ausrichtungen einstellen, nicht aber eine Einrückung, so dass die Buchstaben nicht am linken Rand des Textfeldes stehen bleiben.

Versuche mit der Verbindung von Leerzeichen mit dem Inhalt in einer Abfrage oder einer Formel führen hier nicht dazu, dass das Textfeld tatsächlich den Text eingerückt darstellt. Die Leerzeichen werden vermutlich einfach abgeschnitten.



Zielführender ist es dagegen, einfach ein Textfeld vor dem eigentlichen Text zu positionieren, das aber nicht mit irgendeinem Datenfeld-Typ verbunden wird. Das Textfeld wird genau wie die anderen enthaltenen Felder mit der entsprechenden bedingten Formatierung versehen, so dass eine einheitlich breite (scheinbare) Einrückung beim Ausdruck erfolgt.

Zweispaltige Berichte

Mit geschickter Abfragetechnik ist es möglich, einen Bericht mit mehreren Spalten zu erstellen, bei dem die folgenden Spalten auch die folgenden Datensätze darstellen:

Geburtstagsliste

Kathrin	17.01.84	Sally	15.02.91
Mick	03.03.53	Hanne	13.04.70
Meike	13.04.71	Lara	23.04.85
Monika	05.06.86	Karl	01.07.67
Paul	11.07.89	Egon	23.07.67
Susanne	02.08.65	Georg	28.08.95
Ysabelle	17.09.89	Maik	28.10.93
John	19.11.97	Erkan	17.12.75
Johann	23.12.91		

Der erste Datensatz wird in der linken Spalte aufgeführt, der zweite Datensatz in der rechten Spalte. Die Daten sind nach der Lage des Geburtstages im Jahr sortiert.

Gerade die Sortierung nach Geburtstagen führt dazu, dass die Abfrage für diesen Bericht recht lang ausfällt. Könnte stattdessen eine Sortierung nur nach dem Primärschlüssel der zugrundeliegenden Tabelle erfolgen, so würde der Inhalt der Abfrage stark abnehmen. Bei dem Sortierkriterium handelt es sich um einen immer wiederkehrenden Textblock, der weiter unten erklärt wird.

Grundlage für diesen Bericht ist die folgende Abfrage: (HSQLDB, ~~FIREBIRD~~ (Codeanpassung für Firebird siehe zum Schluss dieses Kapitels))

```
001 SELECT "T1"."Name" AS "Name1", "T1"."Geburtsdatum" AS "Geburtsdatum1",
    "T2"."Name" AS "Name2", "T2"."Geburtsdatum" AS "Geburtsdatum2"
002 FROM
003    (SELECT "Name", "Geburtsdatum", "ZeilenNr" AS "Zeile" FROM
004       (SELECT "a".*,
005          ( SELECT COUNT( "ID" ) FROM "Geburtstage" WHERE
006          RIGHT( '0' || MONTH( "Geburtsdatum" ), 2 ) ||
007          RIGHT( '0' || DAY( "Geburtsdatum" ), 2 ) || "ID"
008          <= RIGHT( '0' || MONTH( "a"."Geburtsdatum" ), 2 ) ||
009          RIGHT( '0' || DAY( "a"."Geburtsdatum" ), 2 ) || "a"."ID" )
010          AS "ZeilenNr"
011       FROM "Geburtstage" AS "a")
012    WHERE MOD( "ZeilenNr", 2 ) > 0 )
013 AS "T1"
014 LEFT JOIN
015    (SELECT "Name", "Geburtsdatum", "ZeilenNr"-1 AS "Zeile" FROM
016       (SELECT "a".*,
017          ( SELECT COUNT( "ID" ) FROM "Geburtstage" WHERE
018          RIGHT( '0' || MONTH( "Geburtsdatum" ), 2 ) ||
019          RIGHT( '0' || DAY( "Geburtsdatum" ), 2 ) || "ID"
020          <= RIGHT( '0' || MONTH( "a"."Geburtsdatum" ), 2 ) ||
021          RIGHT( '0' || DAY( "a"."Geburtsdatum" ), 2 ) || "a"."ID" )
022          AS "ZeilenNr"
023       FROM "Geburtstage" AS "a")
024    WHERE MOD( "ZeilenNr", 2 ) = 0 )
025 AS "T2"
026 ON "T1"."Zeile" = "T2"."Zeile"
027 ORDER BY "T1"."Zeile"
```

Es werden zwei identische Unterabfragen in der Abfrage dargestellt. Die ersten beiden Spalten beziehen sich auf die Unterabfrage mit dem Alias "T1", die letzten beiden Spalten auf die Unterabfrage mit dem Alias "T2".

Die Unterabfragen stellen neben den Feldern der Tabelle "Geburtstage" noch Felder zur Verfügung, die eine Unterscheidung in Zeilen und damit eine Sortierung ermöglichen. Dazu wird im Kern die Abfragemöglichkeit zur *Zeilennummerierung* genutzt.

```
018 RIGHT( '0' || MONTH( "Geburtsdatum" ), 2 ) || RIGHT( '0' ||
    DAY( "Geburtsdatum" ), 2 ) || "ID"
```

Diese Formulierung dient dazu, eine eindeutige Reihenfolge der Datensätze zu gewährleisten. Da die Beispieldatensätze nach dem Datum sortiert werden sollen, ließe sich ja leicht sagen, dass lediglich das Datum zum Vergleich herangezogen werden sollte. Etwas schwieriger ist es allerdings schon dadurch, dass nicht das Geburtsdatum, sondern die Lage des Datums im Jahr maßgebend sein soll. Zusätzliche Probleme bereiten schließlich gleiche Datumswerte, die eine eindeutige Reihenfolge verhindern. Deshalb wird zur Sortierung neben dem Monat und dem Tag auch noch der Primärschlüssel der Tabelle herangezogen, der schließlich eindeutig ist. Damit nicht der Monat '10' vor den Monat '2' gesetzt wird, wird bei der Zusammenführung des Sortierkriteriums in einen Text über **||** vor jede Monatszahl eine führende '0' positioniert, die anschließend bei zweistelligen Monatszahlen über **RIGHT(… , 2)** wieder entfernt wird.

Mit **SELECT COUNT("ID")** wird die Anzahl der Datensätze ermittelt, deren Kombination aus Monat, Tag und Primärschlüssel kleiner oder gleich der entsprechenden Kombination des aktuellen Datensatzes der Tabelle "Geburtstage" ist. Hier handelt es sich um eine *Korrelierte Unterabfrage*.

Über **MOD("ZeilenNr", 2)** wird aus den so ermittelten Zeilennummerierungen ermittelt, ob es sich um eine gerade oder ungerade Zahl handelt. **MOD** ermittelt den Rest der Division, in dem

genannten Beispiel der Division durch 2. Dadurch gibt die **"ZeilenNr"** wechselweise die Nummern '1' und '0' aus. Hierdurch werden die Abfragen für "T1" und "T2" unterschieden.

In der nächsthöheren Abfrageebene von "T2" wird als "Zeile" "ZeilenNr"-1 definiert. Dadurch werden "T1" und "T2" direkt vergleichbar.

"T1" wird mit "T2" über **LEFT JOIN** verbunden, damit auch bei ungerader Datensatzzahl der Tabelle "Geburtstage" alle Daten dargestellt werden. In der Zusammenführung von "T1" und "T2" kann jetzt auf die Spalten direkt Bezug genommen werden: **"T1"."Zeile" = "T2"."Zeile"**.

Zum Schluss wird der ganze Inhalt noch nach dem Wert aus "Zeile" sortiert, der für "T1" und "T2" gleich ist. Dies könnte auch der Bericht über die Gruppierung direkt übernehmen.

Geburtstagsliste

Januar		Februar	
Kathrin	17.01.84	Sally	15.02.91
März		**April**	
Mick	03.03.53	Hanne	13.04.70
		Meike	13.04.71
		Lara	23.04.85
Mai		**Juni**	
		Monika	05.06.86
Juli		**August**	
Karl	01.07.67	Susanne	02.08.65
Paul	11.07.89	Georg	28.08.95
Egon	23.07.67		
September		**Oktober**	
Ysabelle	17.09.89	Maik	28.10.93
November		**Dezember**	
John	19.11.97	Erkan	17.12.75
		Johann	23.12.91

Wesentlich komplizierter sind von der Zusammensetzung Abfragetechniken, die neben der zweispaltigen Darstellung auch noch zusätzliche Unterteilungen ermöglichen. Hier fallen dann nämlich Leerzeilen mitten im Bericht an, die bei der vorhergehenden zweispaltigen Darstellung höchstens am Schluss vorkommen. Mit einer so erstellten Abfrage kommt der Report-Designer erst einmal nicht zurecht. Deswegen wird stattdessen auf zwei miteinander verbundene Ansichten zurückgegriffen.

Die folgende Ansicht "MonatZ" wird zuerst erstellt:

```
001 SELECT "a"."ID", "a"."Name", "a"."Geburtsdatum",
002    MONTH( "a"."Geburtsdatum" ) AS "MonatZahl",
003       ( SELECT COUNT( "ID" ) FROM "Geburtstage"
004       WHERE MONTH( "Geburtsdatum" ) = MONTH( "a"."Geburtsdatum" )
005          AND RIGHT( '0' || DAY( "Geburtsdatum" ), 2 ) || "ID" <=
006          RIGHT( '0' || DAY( "a"."Geburtsdatum" ), 2 ) || "a"."ID" )
007    AS "MonatZaehler"
008 FROM "Geburtstage" AS "a"
```

Alle Felder der Tabelle "Geburtstage" werden übernommen. Zusätzlich wird der Monat als Zahl dargestellt. Der Tabelle "Geburtstage" wird ein Alias "a" zugewiesen, damit auf die Tabelle mit einer korrelierenden Unterabfrage zugegriffen werden kann.

Die Unterabfrage zählt innerhalb eines Monates alle Datensätze, deren Datumswert einen kleiner oder gleichen Tag vorweist. Bei gleichen Tagen unterscheidet schließlich der Primärschlüssel, welcher Datensatz die niedrigere Nummer erhält. Die Technik ist hier gleich wie bei dem vorhergehenden Beispiel.

Auf die Ansicht "MonatZ" greift die Ansicht "Bericht_Monat_zweispaltig" zu. Diese Ansicht wird hier nur in Ausschnitten wieder gegeben:

```
001 SELECT
002    "Tab1"."Name" AS "Name1",
003    "Tab1"."Geburtsdatum" AS "Geburtsdatum1",
004    1 AS "MonatZahl1",
005    IFNULL("Tab1"."MonatZaehler",999) AS "MonatZaehler1",
006    'Januar' AS "Monat1",
007    "Tab2"."Name" AS "Name2",
008    "Tab2"."Geburtsdatum" AS "Geburtsdatum2",
009    2 AS "MonatZahl2",
010    IFNULL("Tab2"."MonatZaehler",999) AS "MonatZaehler2",
011    'Februar' AS "Monat2"
012 FROM
013    (SELECT * FROM "MonatZ" WHERE "MonatZahl" = 1) AS "Tab1"
014    RIGHT JOIN (SELECT * FROM "MonatZ" WHERE "MonatZahl" = 2) AS "Tab2"
015    ON "Tab1"."MonatZaehler" = "Tab2"."MonatZaehler"
016 UNION
017 SELECT
018    "Tab1"."Name" AS "Name1",
019    "Tab1"."Geburtsdatum" AS "Geburtsdatum1",
020    3 AS "MonatZahl1",
021    IFNULL("Tab1"."MonatZaehler",999) AS "MonatZaehler1",
022    'März' AS "Monat1",
023    "Tab2"."Name" AS "Name2",
024    "Tab2"."Geburtsdatum" AS "Geburtsdatum2",
025    4 AS "MonatZahl2",
026    IFNULL("Tab2"."MonatZaehler",999) AS "MonatZaehler2",
027    'April' AS "Monat2"
028 FROM
029    (SELECT * FROM "MonatZ" WHERE "MonatZahl" = 3) AS "Tab1"
030    LEFT JOIN (SELECT * FROM "MonatZ" WHERE "MonatZahl" = 4) AS "Tab2"
031    ON "Tab1"."MonatZaehler" = "Tab2"."MonatZaehler"
032 UNION
033 ...
034 ORDER BY "MonatZahl1", "MonatZahl2", "MonatZaehler1", "MonatZaehler2"
```

Zuerst werden in der Unterabfrage aus "MonatZ" alle Daten ausgelesen, bei denen die "MonatZahl" **1** ist. Diese Auswahl wird mit dem Alias "Tab1" versehen. Gleichzeitig werden mit dem Alias "Tab2" alle Daten ausgelesen, bei denen die "MonatZahl" 2 ist. Beide Tabellen werden durch einen **RIGHT JOIN** verbunden, so dass alle Datensätze aus "Tab2" und nur die Datensätze aus "Tab1" angezeigt werden, bei denen der "MonatZaehler" mit dem von "Tab2" übereinstimmt.

Die Spalten der Ansicht müssen unterschiedliche Bezeichnungen ausweisen, so dass jede Spalte mit einem Alias versehen ist. Außerdem wird direkt als Spaltenwert für "Tab1" eine **1** als "MonatZahl" sowie **'Januar'** als "Monat1" eingegeben. Diese Einträge erscheinen auch, wenn es keinen Datensatz aus "Tab1", wohl aber noch Datensätze aus "Tab2" gibt. Ist kein "MonatZaehler" vorhanden, so soll dort der Wert **999** eingetragen werden. Da "Tab1" mit "Tab2" über einen **RIGHT JOIN** verbunden ist, kann es ja vorkommen, dass bei weniger Datensätzen in "Tab1" stattdessen leere Felder angezeigt werden. Leere Felder würden aber bei einer späteren Sortierung vor allen Feldern mit Inhalt sortiert, so dass stattdessen ein besonders hoher Wert gewählt wurde.

Bei der Darstellung der Spalten für "Tab2" wird entsprechend vorgegangen. Hier könnte allerdings **IFNULL("Tab2"."MonatZaehler",999)** entfallen, da bei einem **RIGHT JOIN** zugunsten von "Tab2" zwar alle Zeilen aus "Tab2", nicht aber mehr Zeilen aus "Tab1" als aus "Tab2" dargestellt werden.

Genau dieses Problem wird mit der Verbindung zweier Abfragen gelöst. Mit **UNION** werden alle Datensätze aus der ersten Abfrage und alle Datensätze aus der zweiten Abfrage dargestellt. Die Datensätze aus der zweiten Abfrage erscheinen aber nur dann, wenn sie nicht identisch mit einem vorhergehenden Datensatz sind. **UNION** wirkt also wie **DISTINCT**.

Über **UNION** wird die gleiche Abfrage noch einmal gestellt, nur sind "Tab1" und "Tab2" jetzt mit einem **LEFT JOIN** verbunden. Damit erscheinen auf jeden Fall alle Datensätze aus "Tab1" auch wenn in "Tab2" weniger Datensätze vorhanden sind als in "Tab1".

Für die Monate 3 und 4, 5 und 6 usw. werden entsprechend angepasste Abfragen verwendet und wiederum mit **UNION** an die vorhergehenden Abfragen angehängt.

Das Ergebnis der Ansicht wird schließlich nach "MonatZahl1", "MonatZaehler1" und "MonatZaehler2" sortiert. Nach "MonatZahl2" muss nicht sortiert werden, da "MonatZahl1" ja bereits die entsprechende Reihenfolge wiedergibt.

Sollen die Abfragen zur Geburtstagsliste an Firebird angepasst werden, so sind Tage und Monate durch **EXTRACT(DAY FROM "Datum")** bzw. **EXTRACT(MONTH FROM "Datum")** zu ermitteln. Außerdem ist statt **IFNULL()** die Funktion **COALESCE()** erforderlich. Hier braucht allerdings nur der Name ausgetauscht zu werden.

Bugs und Workarounds beim Report-Designer

Der Report-Designer birgt manchmal Fehler, die nicht so ohne weiteres nachvollzogen werden können. Hier einige Fehlerquellen und eventuell nützliche Gegenmaßnahmen.

Der Inhalt eines Feldes aus einer Abfrage erscheint nicht

Eine Datenbank soll den Verkauf von Waren simulieren. In einer Abfrage wird aus der Anzahl einer gekauften Ware und dem Einzelpreis der Gesamtpreis ermittelt.

```
001 SELECT "Verkauf"."Anzahl", "Ware"."Ware", "Ware"."Preis",
002 "Verkauf"."Anzahl"*"Ware"."Preis"
003 FROM "Verkauf", "Ware" WHERE "Verkauf"."Ware_ID" = "Ware"."ID"
```

Diese Abfrage dient als Grundlage des Berichtes. Wird aber das Feld **"Verkauf"."Anzahl"*"Ware"."Preis"** in dem Bericht aufgerufen, so bleibt es ohne Inhalt. Wird dem Feld in der Abfrage hingegen ein Alias zugewiesen, so kann der Report-Designer darauf einwandfrei zugreifen:

```
001 SELECT "Verkauf"."Anzahl", "Ware"."Ware", "Ware"."Preis",
002 "Verkauf"."Anzahl"*"Ware"."Preis" AS "GPreis"
003 FROM "Verkauf", "Ware" WHERE "Verkauf"."Ware_ID" = "Ware"."ID"
```

Das Feld greift jetzt auf **"Gpreis"** zu und stellt den entsprechenden Wert dar.

Ein Bericht lässt sich nicht ausführen

Manchmal kommt es aber vor, dass sich ein Bericht zwar erstellen, aber anschließend nicht ausführen oder nicht einmal speichern lässt. Es erscheint eine Fehlermeldung, die erst einmal wenig aussagekräftig ist:
«Bericht konnte nicht ausgeführt werden. Eine Ausnahme vom Typ com.sun.star.lang.WrappedTargetException wurde entdeckt.»

Hier kann es schon einmal hilfreich sein, sich die entsprechend angeboten zusätzlichen Informationen anzeigen zu lassen. Taucht dort irgendwie ein Bezug zu «SQL» auf, so ist vermutlich der Report-Designer nicht in der Lage, den SQL-Code der Datenquelle korrekt zu interpretieren.

Hier hilft es vielleicht, über den Berichtsnavigator die Eigenschaften des Berichtes aufzusuchen: **Daten → SQL-Befehl analysieren → Nein**. Leider hat diese Problemlösung zur Folge, dass eine einmal eingestellte Gruppierung nicht mehr funktioniert.

Hinweis

Ein besser geeigneter Weg, Problemen mit dem SQL-Code einer Abfrage aus dem Weg zu gehen, ist, statt eines Abfrage als Grundlage für den Bericht eine **Ansicht** zu nutzen. Die wird von der Datenbank so erstellt, dass sie für den Report-Designer wie eine Tabelle erscheint und ohne Probleme zu verarbeiten ist. Hier funktioniert sogar die Sortiervorgabe der Ansicht reibungslos.
Auch Abfragen, die sonst nur in direktem SQL-Code ausführbar und damit für den Report-Designer teilweise nicht interpretierbar sind, können über die Ansicht ihre Inhalte problemlos an den Report-Designer weitergeben.

Datums- und Zeitwerte werden in Diagrammen nicht angezeigt

Leider kam es wiederholt vor, dass Berichte mit Diagrammen im Report-Designer überhaupt nicht angezeigt wurden. Lässt sich ein Bericht mit Diagrammen anzeigen, so kann es vorkommen, dass Achsen mit einer Datums- oder Zeitangabe die Anzeige der Werte für das Diagramm unmöglich machen: *Bug 87012* . Hier hilft die Umwandlung der Datums- und Zeitwerte in einer Abfrage. Aus Datumsangaben werden dadurch Integer-Zahlen, aus Zeitwerten werden dadurch Dezimalzahlen, die die Zeit als Bruchteil eines Tages darstellen.

HSQLDB:

```
001 SELECT
002    DATEDIFF( 'dd', '1899-12-30', "Datum" ) AS "DatumInteger",
003    "Temperatur"
004 FROM "Tabelle"
```

FIREBIRD:

```
001 SELECT
002    "Datum" – DATE '1899-12-30' AS "DatumInteger",
003    "Temperatur"
004 FROM "Tabelle"
```

rechnet den Zeitunterschied in Tagen zu dem internen 0-Wert um. Die Datumszählung beginnt für LO beim 30.12.1899.

HSQLDB:

```
001 SELECT
002    HOUR( "Zeit" ) / 24.00000 + MINUTE( "Zeit" ) / 1440.000000
003    AS "ZeitDezimal",
004    "Temperatur"
005 FROM "Tabelle"
```

FIREBIRD:

```
001 SELECT
002    ("Zeit" – TIME '00:00')/86400.00000 AS "ZeitDezimal",
003    "Temperatur"
004 FROM "Tabelle"
```

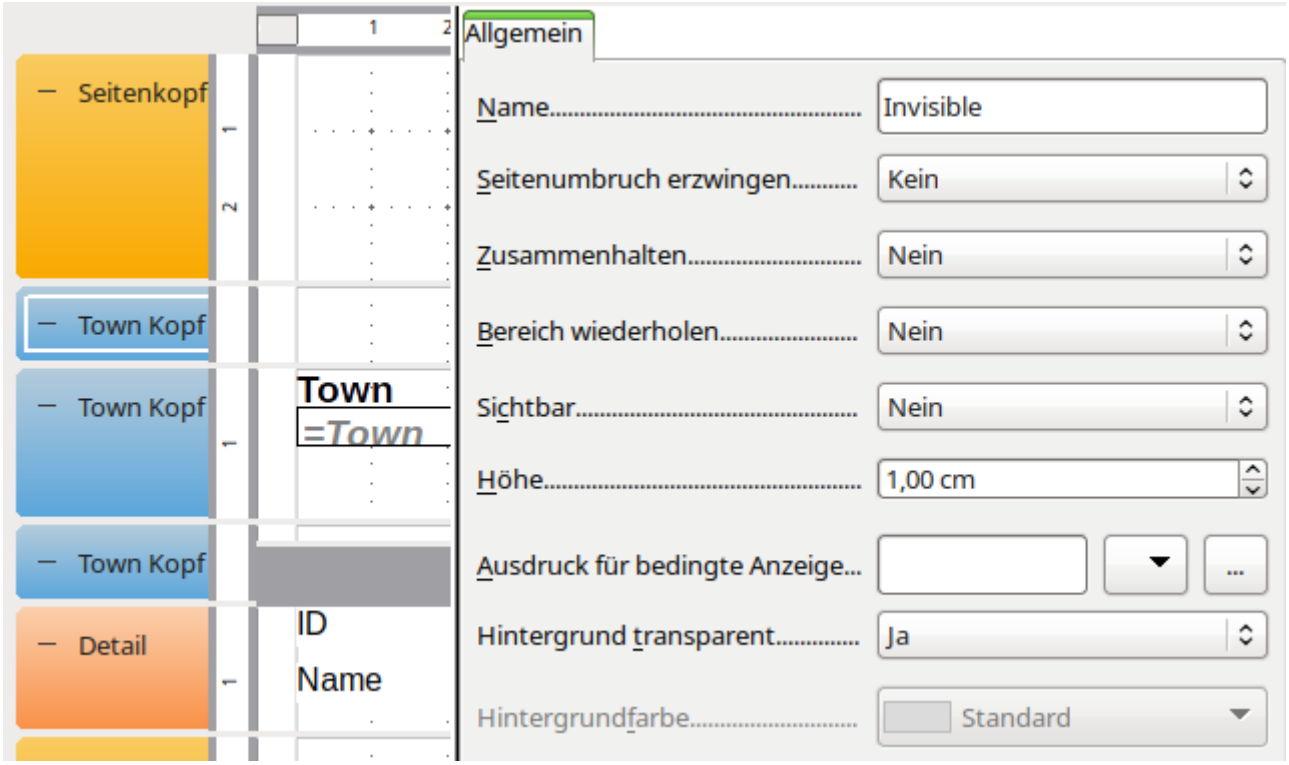
wandelt die Zeitangabe in Bruchteile eines Tages um. Dabei muss darauf geachtet werden, dass die Division durch Dezimalzahlen mit Nachkommastellen erfolgt. Sonst werden nur ganze Tage wiedergegeben.

Die Integer-Angaben für das Datum bzw. die Dezimalzahlen für die Zeit können für die Achsen des Diagramms als Datum bzw. Zeit formatiert werden. Die dargestellten Werte sind korrekt. Das Diagramm wird auch gezeichnet.

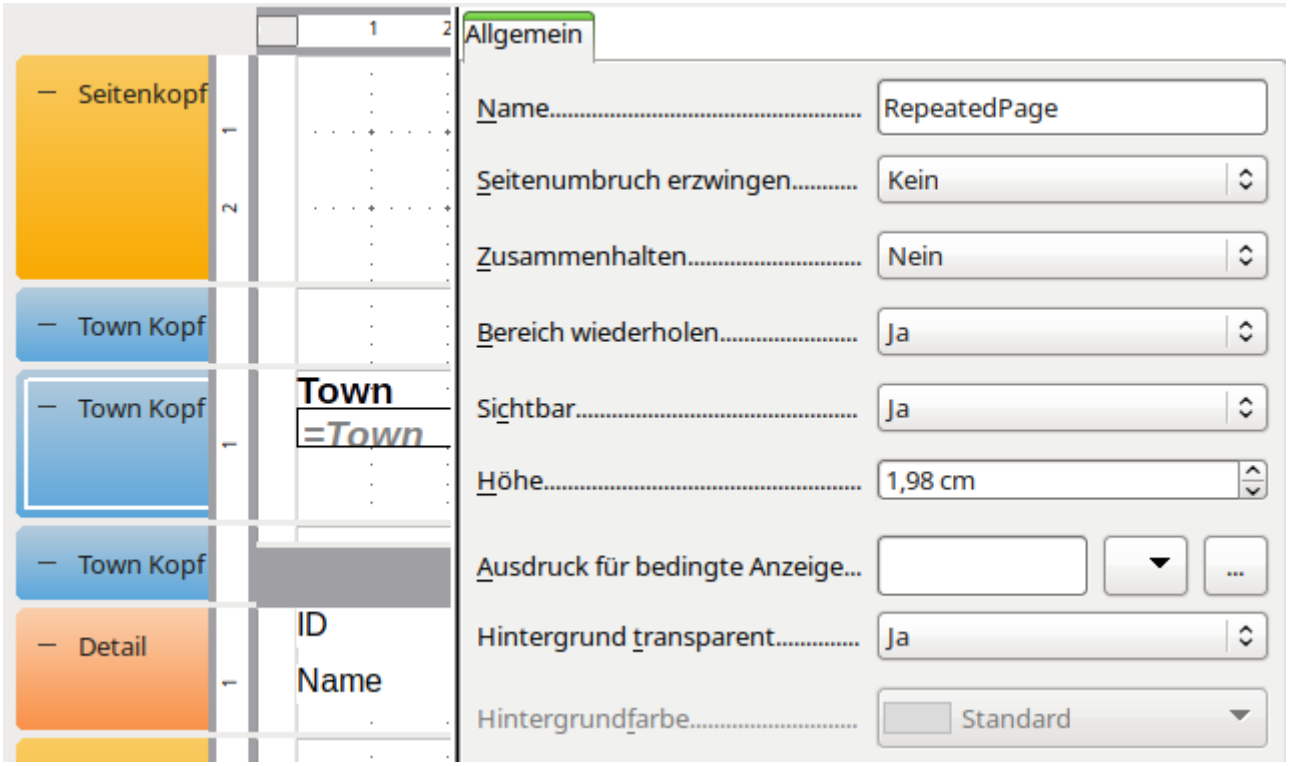
Gruppierungen mit wiederholendem Bereich zeigen nur den ersten Wert

Wird ein Bericht mit einer Gruppierung erstellt, die auf jeder Seite angezeigt werden soll (**Bereich wiederholen → 'Ja'**), dann zeigt anschließend der Bericht den Inhalt des ersten Gruppenbereichs bei jeder Gruppe an: *Bug 82097*. Dies liegt daran, dass der wiederholende Bereich bei der Erstellung des Dokumentes in die Kopfzeile der Seite geschrieben wird. Leider wird aber der Zeitpunkt nicht konkret belegt, wann denn eine neue Seitenvorlage erstellt werden soll, so dass die Kopfzeile jetzt die nächste Gruppe anzeigen könnte.

Wird zu dem gleichen Element eine weitere Gruppe erzeugt, die vor der wiederholenden Gruppe steht, nicht sichtbar ist und den Bereich nicht wiederholen lässt, so lässt sich darüber der Seitenumbruch erzeugen, der auch zu einem Wechsel in der Kopfzeile führen kann.



Hier existieren 3 Gruppen zu dem Feld «Town». Die erste Gruppe ist unsichtbar geschaltet. Alle Eigenschaften sind auf 'Nein' gesetzt. Die Höhe spielt keine Rolle.

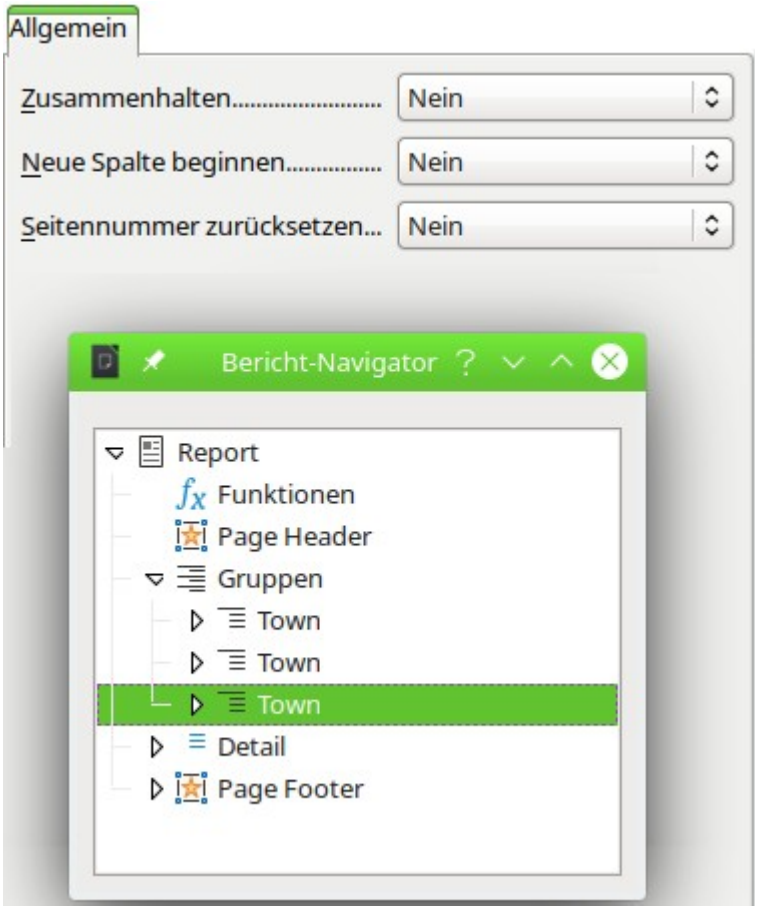


In der zweiten Gruppe steht der Wiederholungsbereich. Die Gruppe ist sichtbar. Dieser Inhalt wird beim Ausführen des Berichtes in der Kopfzeile des Dokumentes angezeigt.

Es ist nicht erforderlich, **Seitenumbruch erzwingen → 'Ja'** einzustellen. Das geschieht automatisch mit der Wiederholung des Bereichs im Hintergrund: *Bug 51452*. Dieser Bug hat auch zur Folge, dass eine Gruppe, die vor der Wiederholungsgruppe im Editor auftaucht, immer alleine auf der vorhergehenden Seite erscheint. Soll dies nicht geschehen, so dürfen vor der Gruppe, die wiederholt werden soll, nur Gruppen stehen, die gar nicht erst angezeigt werden (wie oben die unsichtbare Gruppe, die dem Report-Designer nur hilft, den Änderung des Inhaltes bei der folgenden Gruppe zu erstellen).

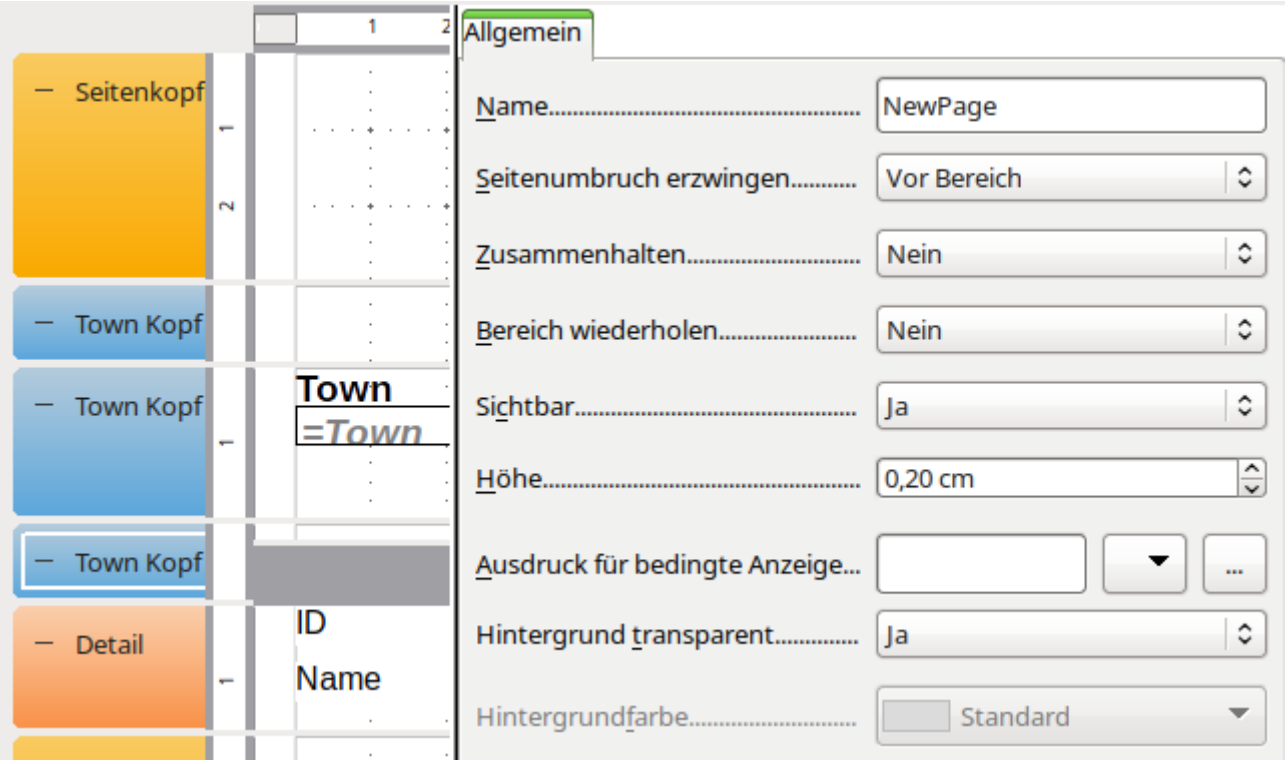
Ein Neustart der Seitenzählung mit der Gruppe ist nicht möglich

Manche Berichte werden ausgedruckt und anschließend an verschiedene Gruppe verteilt, die jeweils nur einen Teil des Berichtes – eben den für ihre Gruppe – erhalten. Hier wäre es gut, wenn der Report-Designer einen Wechsel der Seitenzahlen anbietet: *Bug 51453*. Offiziell tut er das auch – nur funktioniert die Einstellung leider nicht:



Über den Berichtsnavigator ist jeweils eine zusätzliche Einstellung für die Gruppe erreichbar. Weder **Neue Spalte beginnen** (*Bug 52944*)noch **Seitennummer zurücksetzen** bewirken hier irgend etwas.

Für das Setzen der neuen Seitennummer wurde in dem obigen Beispiel eine dritte Gruppe eingebaut:



Diese Gruppe muss sichtbar sein, sollte aber in diesem Fall nur wenig Platz beanspruchen. Sie wird nach dem wiederholten Bereich eingebaut, da sie sonst alleine auf der vorhergehenden Seite stehen würde. Auch wenn hier **Seitenumbruch erzwingen → Vor Bereich** steht, so heißt das nicht, dass der wiederholte Bereich nicht auf der gleichen Seite liegt. Reichlich verwirrend, aber dann etwas klarer, wenn der Aufbau des Berichtes klar wird: Der wiederholte Bereich liegt in der Kopfzeile, der erzwungene Seitenumbruch in der ersten Tabelle ganz oben auf der Seite (aber unterhalb der Kopfzeile).

Der Bereich **Name → 'NewPage'** wird bei der Ausführung des Berichts im Writer zu Tabellen mit der Bezeichnung 'NewPage', 'NewPage0', 'NewPage1' usw. Für jede Gruppe wird also durchnummeriert ein entsprechender Tabellenname erstellt. Hier greift das Makro an, das anhand des Tabellennamens die Tabelle ansteuert und dort dann zu dem Umbruch noch die Änderung der Seitennummer hinzu fügt.

```
001  SUB ReportStart
002     oReport = ThisDatabaseDocument.ReportDocuments.getByName("Report1").open
003     GroupNewPage(oReport)
004  END SUB
```

```
001  SUB GroupNewPage(oReport AS OBJECT)
002     DIM oTables AS OBJECT
003     DIM oTable AS OBJECT
004     DIM inT AS INTEGER
005     oTables = oReport.getTextTables()
006     FOR inT = 0 TO oTables.count() - 1
007        oTable = oTables.getByIndex(inT)
008        IF Left$(oTable.name, 7) = "NewPage" THEN
009           oTable.PageNumberOffset = 1
010        ENDIF
011     NEXT inT
012  END SUB
```

Der Bericht wird aus einem Formular heraus gestartet. Name des Berichts ist hier schlicht «Report1». Beim Start des Berichts wird das entsprechende Dokument direkt als Variable **oReport** zur Weitergabe an die entsprechende Prozedur **GroupNewPage** eingelesen.

Diese Prozedur holt sich aus dem erstellten Writer-Dokument alle Tabellendokumente (Zeile 5). In diesen Tabellendokumenten wird dann nach denen gesucht, die mit dem Namen «NewPage"

beginnen (Zeile 8). Für diese Tabellen wird dann die Seitenzahl für den definierten Seitenumbruch auf 1 gestellt (Zeile 9). So startet jede Gruppe mit einer neuen Seitenzahl 1.

Allerdings ist die Gesamtzahl der Seiten hiervon nicht berührt. Die Gesamtzahl der Seiten für eine Gruppe lässt sich auch in einem Writer-Dokument nicht so ohne weiteres ermitteln. Hier steht immer nur die Gesamtzahl der Seitenzahlen für das ganze Dokument.

Nachträgliche Bearbeitung des Berichtsdokuments

Berichte, die über den Writer erstellt werden, sind Dokumente mit vielen Texttabellen. Die Bearbeitung dieser Dokumente ist zwar möglich, aber doch recht mühselig. Einfacher ist es oft, den Bericht mit Arbeitsanweisungen für die nachträgliche Bearbeitung über Makros zu starten. Damit lassen sich auch Fehler, die der Report-Designer macht, ausgleichen.

Das folgende Makro zeigt einen solchen Zugriff auf die Tabelle eines Berichtes. Es ändert in der Tabelle mit der Bezeichnung «Detail» die Rahmen um die Tabelle und die Zellen. Hierfür gibt es im Bericht sonst keine Einstellung. Es ist lediglich möglich, mit viel Mühe horizontale und vertikale Linien einzufügen. Aber Achtung: Um alle Zellen werden mit dem Makro Rahmen gezogen. Wenn der Detail-Bereich z.B. höher ist als die darin enthaltenen Felder für den Text, dann gibt es entsprechend viele Linien.

```
001 SUB TableBorder
002    DIM oReport AS OBJECT
003    DIM oTables AS OBJECT
004    DIM oTable AS OBJECT
005    DIM inT AS INTEGER
006    DIM inI AS INTEGER
007    oReport = ThisDatabaseDocument.ReportDocuments.getByName("NameDesBerichts").open
008    oTables = oReport.getTextTables()
009    FOR inT = 0 TO oTables.count() - 1
010       oTable = oTables.getByIndex(inT)
011       IF Left$(oTable.Name, 6) = "Detail" THEN
012          oBorder = oTable.Tableborder
013          oBorderline = oBorder.TopLine
014          oBorderline.outerlinewidth = 1
015          oBorderline.innerlinewidth = 0
016          oBorderline.linedistance = 0
017          oBorderline.color = 0
018          oBorder.Topline = oBorderline
019          oBorder.Bottomline = oBorderline
020          oBorder.Leftline = oBorderline
021          oBorder.Rightline = oBorderline
022          oBorder.Horizontalline = oBorderline
023          oBorder.Verticalline = oBorderline
024          oBorder.Distance = 1
025          oTable.Tableborder = oBorder
026       ENDIF
027    NEXT inT
028 END SUB
```

Die Tabellen haben dabei die gleichen Namen wie die Bereiche im Report-Designer. Gegebenenfalls kann einfach ein Bericht erstellt werden und der Tabellenname daraus ausgelesen werden. An dieser Stelle stehen alle Möglichkeiten für **oTables** zur Verfügung, die unter *https://api.libreoffice.org/docs/idl/ref/servicecom_1_1sun_1_1star_1_1text_1_1TextTable.html* verzeichnet sind.

Andere Formen der Berichtserstellung

Der Report-Designer ist zur Zeit die einzige Möglichkeit, über die grafische Benutzeroberfläche direkt Berichte zu gestalten. Nur wenn der Report-Designer nicht installiert ist, kommt eine ältere Möglichkeit der Berichtserstellung zutage.

Das alte Berichtsmodul

Bevor der Report-Designer entwickelt wurde, ist ein anderes Modul für die Erstellung von Berichten entwickelt worden. Dieses Modul kann Berichte gruppiert darstellen, ist aber nur zu tabellarischen Berichten in der Lage. Ist der Report-Designer nicht mit LO installiert worden, so steht dieses Berichtsmodul weiterhin für die Erstellung von Berichten zur Verfügung. Ein Aufruf der so erstellten Berichte ist auch mit installiertem Report-Designer weiterhin möglich.

Die BaseReportExtension

Unter *https://extensions.libreoffice.org/extensions/basereportextension* stellt Georg Mößlacher eine Berichtsmöglichkeit zur Verfügung, die er seit Jahren selbst nutzt. Diese Berichtsmöglichkeit benötigt eine Writer-Vorlagendatei (Endung: .ott). Diese Vorlagendatei wird mit mindestens einer zweizeiligen Tabelle an beliebiger Stelle versehen, die über den Tabellennamen aus der Extension heraus gefunden und mit Inhalten gefüllt wird. Die erste Zeile der Tabelle enthält die Spaltenbeschriftungen, in den nachfolgenden Zeilen wird der Inhalt eingefügt. Fehlende Tabellenzeilen werden über die Extension hinzugefügt.

Mit der BaseReportExtension kann auf den Inhalt geöffneter Formulare Bezug genommen werden. Auch lässt sich der Inhalt der Ausgabe über eine Input-Box filtern. Die Extension ist damit vor allem für so etwas wie die Erstellung von Rechnungen gut geeignet. Die Extension ist allerdings nicht in der Lage, Berichte wie im Report-Designer zu gruppieren.

Weitere Berichtsmöglichkeiten

Weitere Möglichkeiten, einen Bericht zu erstellen, sind im Kapitel «Makros» > «Datenbankaufgaben mit Makros erweitert» > «Drucken aus Base heraus» beschrieben. Auch die diesem Handbuch beigefügten Base-Beispiele enthalten Möglichkeiten des Seriendrucks und des Rechnungsdrucks in Tabellenform mit dem Writer incl. der Erstellung eines Feldes, das beim Seitenwechsel einen Übertrag ermittelt .

Datenbank-Anbindung

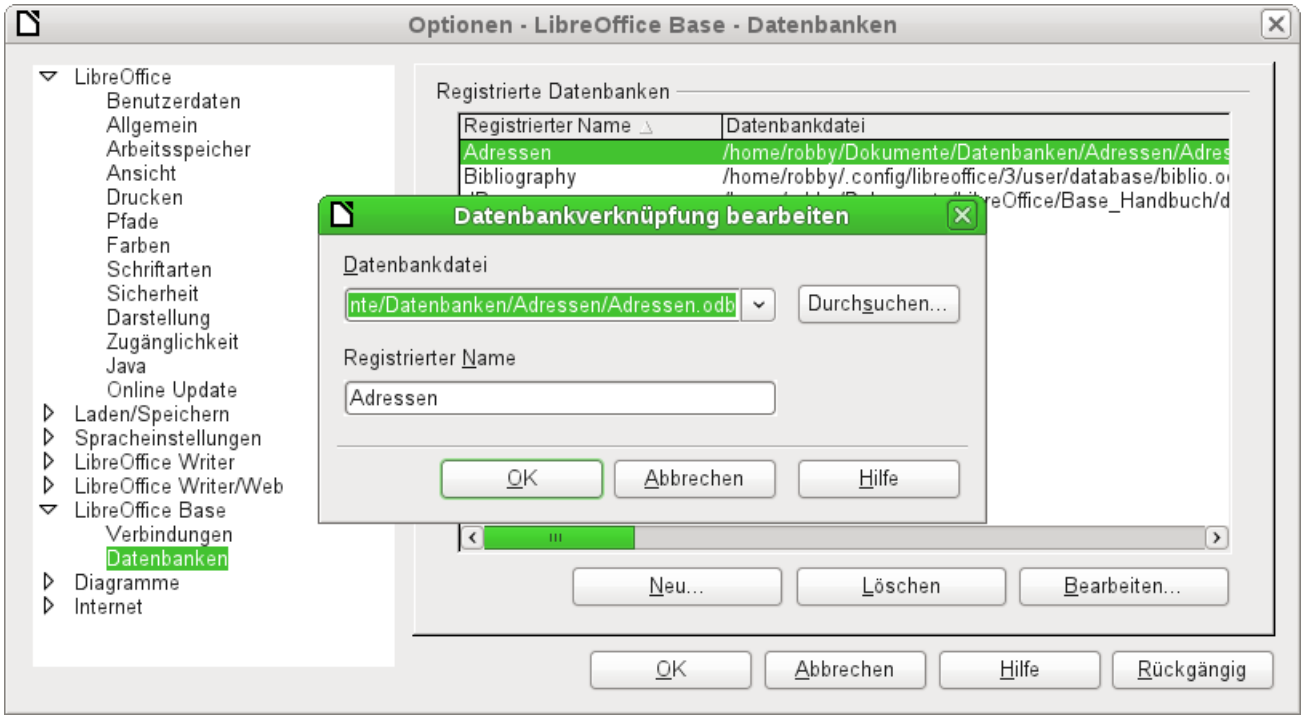
Allgemeines zur Datenbank-Anbindung

Base lässt sich in LibreOffice Writer und LibreOffice Calc auf verschiedene Weise als Datenquelle nutzen. Die Nutzung von Base ist dabei nicht unbedingt an eine Anmeldung der Datenbank in den Einstellungen von LibreOffice gebunden. So interagieren externe Formulare auch direkt mit Base, sofern dort als Datenquelle der Weg zur Datenbank angegeben wird.

Anmeldung der Datenbank

Für viele Funktionen wie z.B. die Nutzung beim Etikettendruck, die Nutzung von Daten für Serienbriefe usw. ist die Anmeldung einer Datenbank in den Einstellungen von LibreOffice notwendig.

Über **Extras → Optionen → LibreOffice Base → Datenbanken → Neu** lässt sich eine Datenbank für den weitergehenden Gebrauch in anderen LibreOffice-Komponenten anmelden.



Die Datenbank wird über den Dateibrowser gesucht und auf ähnliche Weise mit LibreOffice verknüpft wie sonst bei einem einfachen Formular. Der Datenbank selbst wird allerdings ein eingängigerer Name gegeben, der ohne weiteres anders lauten kann als der Name der eigentlichen Datenbankdatei. Die Namensbezeichnung ist so etwas wie ein Alias-Name, was ja auch bei Abfragen in Datenbanken vergeben werden kann.

Hinweis

Wird eine Datenbankverknüpfung bearbeitet, so lässt die Verknüpfung sich anschließend zumindest bis einschließlich LO 7.3 nur mit verändertem registrierten Namen speichern: *Bug 149195*. Für den Dialog ist der ursprüngliche registrierte Name bereits vergeben.
Um eine Abspeicherung mit gleichem Namen wie ursprünglich zu erreichen muss also der Dialog erneut aufgerufen werden. Jetzt wird wieder der ursprüngliche Name gewählt und die Abspeicherung mit ursprünglichem Namen gelingt.

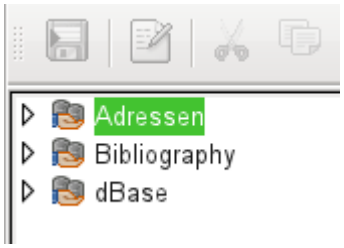
Datenquellenbrowser

Über den Datenquellenbrowser im Writer oder in Calc erhalten sie Zugriff auf Tabellen und Abfragen der registrierten Datenbanken unter dem dort angegebenen registrierten Namen. Der Browser öffnet sich über **Ansicht → Datenquellen** oder nach dem Drücken der Tastenkombination *Strg+Umschalt+F4* oder über das entsprechende, in der Grundeinstellung nicht eingeblendete, Symbol in der Standard-Symbolleiste.

Hinweis

Die folgende Beschreibung des Datenquellenbrowser richtet sich nach den Funktionen im **Writer** aus. Für **Calc** siehe *Datenbanknutzung in Calc.*

Auf der linken Seite im Datenquellen-Browser sind die angemeldeten Datenquellen zu sehen. Die Datenquelle "Bibliography" ist standardmäßig bei LibreOffice mit dabei. Die weiteren Datenquellen unterscheiden sich je nach angemeldetem Namen.

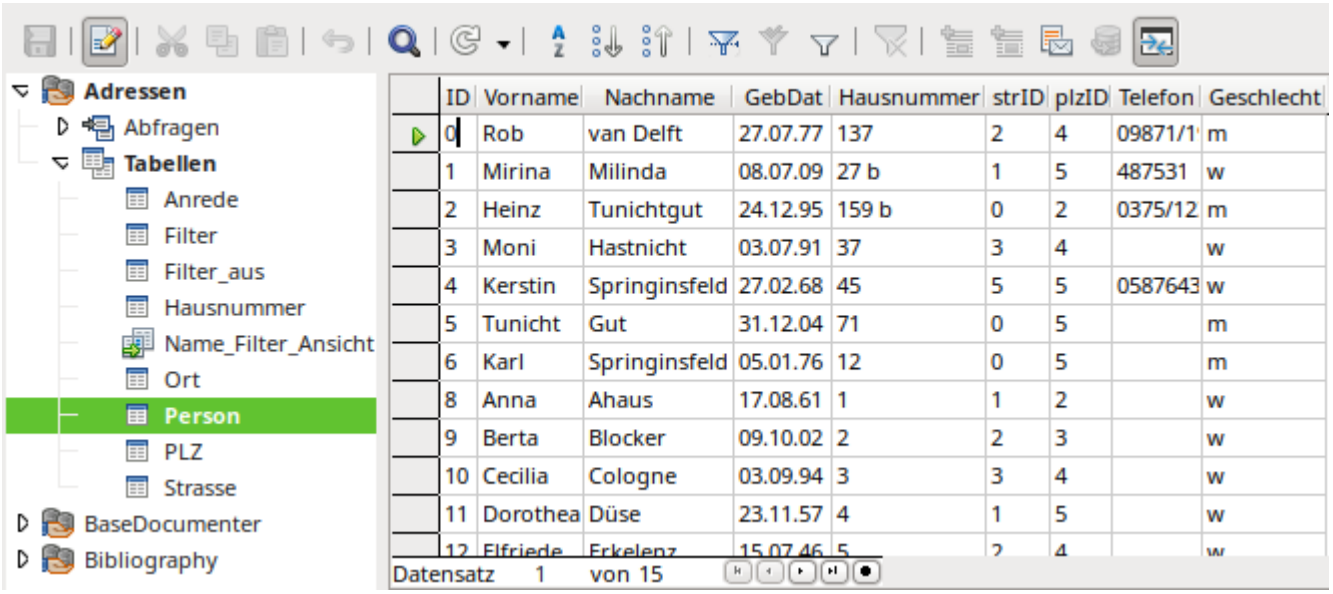


Durch einen Klick auf das ⊳ - Zeichen vor den Datenbankbezeichnungen öffnet sich die Datenbank und zeigt an, dass sich darin ein Unterordner für Abfragen und ein Unterordner für Tabellen befindet. Die weiteren Unterordner der Datenbank werden nicht zur Verfügung gestellt. Interne Formulare sowie Berichte sind also nur über die Datenbank selbst verfügbar.

Erst beim Klick auf den Ordner Tabellen erfolgt der tatsächliche Datenbankzugriff. Bei passwortgeschützten Datenbanken erfolgt zu diesem Zeitpunkt die Passwortabfrage.

Rechts von dem Verzeichnisbaum zeigt sich die angewählte Tabelle. Sie kann wie in Base selbst bearbeitet werden. Bei stärker ausgeprägten relationalen Datenbanken ist allerdings die Eingabe in die Tabelle mit Vorsicht zu bewerkstelligen, da die Tabellen ja über Fremdschlüssel miteinander verknüpft sind. Allein die unten abgebildete Datenbank zeigt eine separate Tabelle für die Straßennamen, eine für Postleitzahlen und noch eine für den Ort.

Für den richtigen Blick auf die Daten ohne entsprechende Editiermöglichkeit eignen sich daher Abfragen oder entsprechende Ansichten (Views) besser.



Von den Zeichen in der Symbolleiste sind viele bereits aus der Eingabe von Daten in Tabellen bekannt. Neu ist insbesondere der letzte Abschnitt: **Daten in Text**, **Daten in Felder**, **Seriendruck**, **Aktuelle Dokument-Datenquelle**, **Explorer ein/aus**.

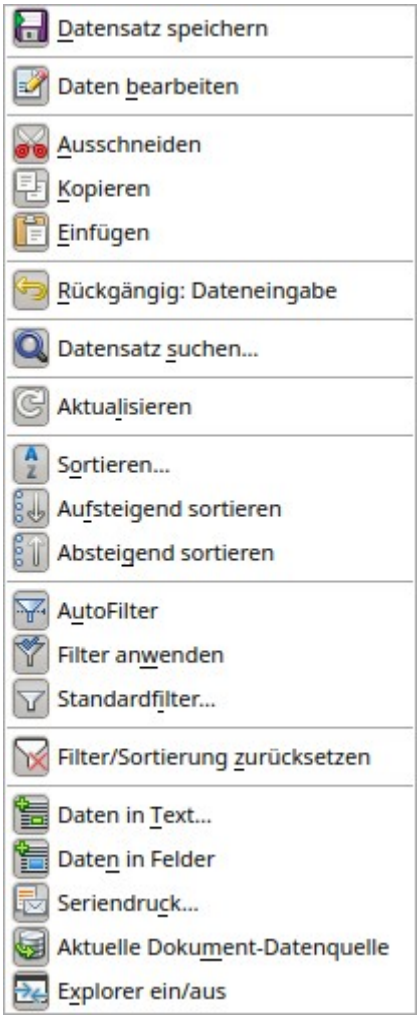


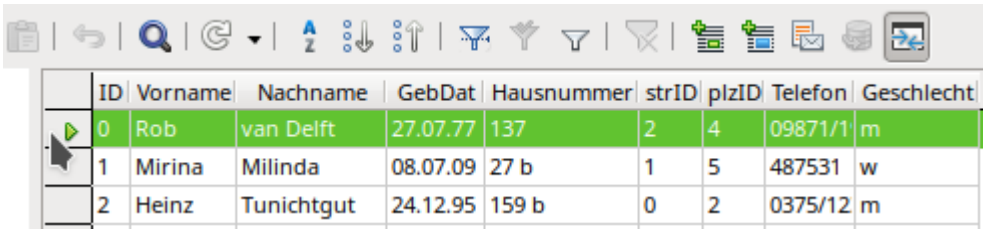
Abbildung 51: Symbolleiste Datenquellenbrowser

Hinweis

Der **Datenquellenbrowser** eignet sich auch für die Dateneingabe. Dies sollte bei ~~Firebird~~ so lange **nicht zur Eingabe und Änderung von Daten** genutzt werden, wie diese Datenbank noch das Abspeichern in der *.odb-Datei erfordert. Sonst scheinen die Daten geändert, werden aber nicht gesichert.

Daten in Text

Die Funktion **Daten in Text** steht zur Verfügung, sobald ein Datensatz markiert wurde.



ID	Vorname	Nachname	GebDat	Hausnummer	strID	plzID	Telefon	Geschlecht
0	Rob	van Delft	27.07.77	137	2	4	09871/1	m
1	Mirina	Milinda	08.07.09	27 b	1	5	487531	w
2	Heinz	Tunichtgut	24.12.95	159 b	0	2	0375/12	m

Abbildung 52: Markierung eines Datensatzes

Wird jetzt **Daten in Text** angewählt, so erscheint ein Assistent, der die erforderlichen Formatierungen vornimmt:

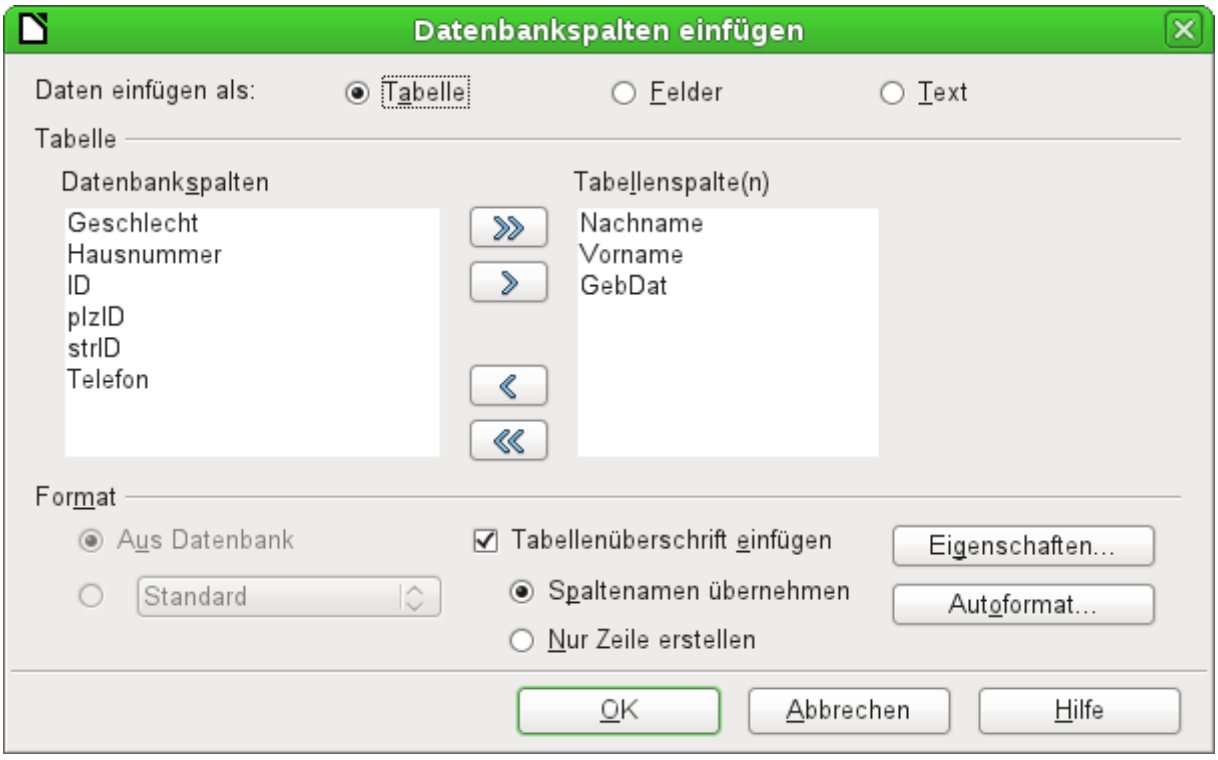


*Abbildung 53: **Daten Einfügen als → Tabelle***

Für das Einfügen von Daten in den Text stehen die Möglichkeiten der Darstellung in Tabellenform, als einzelne Felder oder als Gesamttext zur Verfügung.

Die obige Abbildung zeigt den Aufbau **Daten einfügen als → Tabelle**. Bei Zahlenfeldern und Datumsfeldern kann das Format der Datenbank durch ein eigenes gewähltes Format geändert werden. Ansonsten erfolgt die Formatierung mit der Auswahl der Tabellenfelder. Die Reihenfolge der Felder wird über die Pfeiltasten festgelegt.

Sobald Tabellenspalten gewählt sind, wird der Button Eigenschaften für die Tabellen aktiviert. Hier können die üblichen Tabelleneigenschaften des Writer eingestellt werden (Breite der Tabelle, Spaltenbreite ...).

Das Markierfeld gibt darüber Auskunft, ob eine Tabellenüberschrift gewünscht ist. Ist dieses Markierfeld nicht angewählt, so wird für die Überschrift keine separate Zeile eingefügt.

Die so eventuell gewählte Zeile für die Tabellenüberschrift kann entweder die Spaltennamen übernehmen oder nur die Zeile erstellen und den Platz für die Überschrift zum späteren Editieren schaffen.

Über den Button Autoformat werden einige vorformatierte Tabellenansichten angeboten. Bis auf das Vorschlagsformat «*Standard*» können alle hier enthaltenen Formatierungen umbenannt werden.

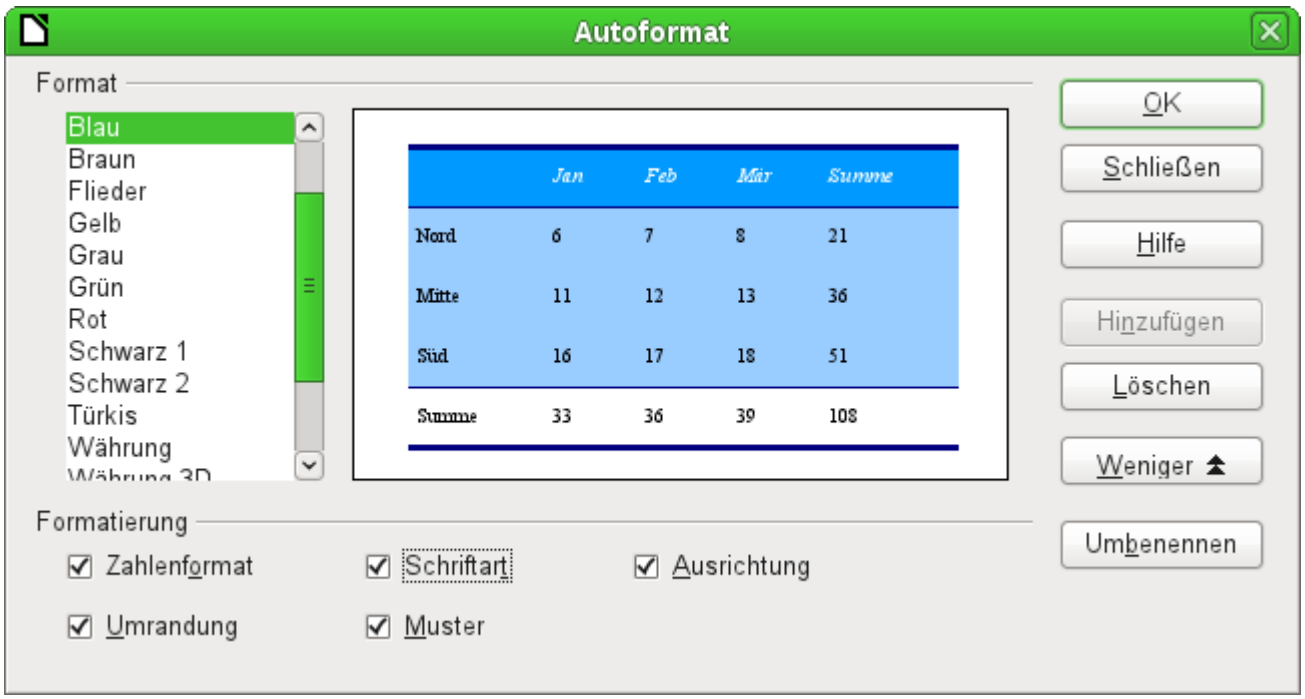


Abbildung 54: Über Autoformat *stehen Tabellenformate zur Auswahl*

Um dem Autoformat-Vorschlag für Tabellen eine Tabelle hinzuzufügen, muss eine Tabelle erstellt worden sein. Diese wird dann markiert und kann über den Button Hinzufügen in die Tabellenliste aufgenommen werden.

Die Tabelle wird schließlich mit der Anzahl der markierten Zeilen erstellt.

Über **Daten einfügen als Felder** wird die Möglichkeit geboten, in einem kleinen Editor die verschiedenen Tabellenfelder hintereinander in einem Text zu positionieren. Dem erstellten Text kann außerdem eine Absatzvorlage zugewiesen werden. Auch hier ist die Formatierung von Datums- und Zahlenwerten separat wählbar, kann aber auch direkt aus den Tabelleneinstellungen der Datenbank gelesen werden.

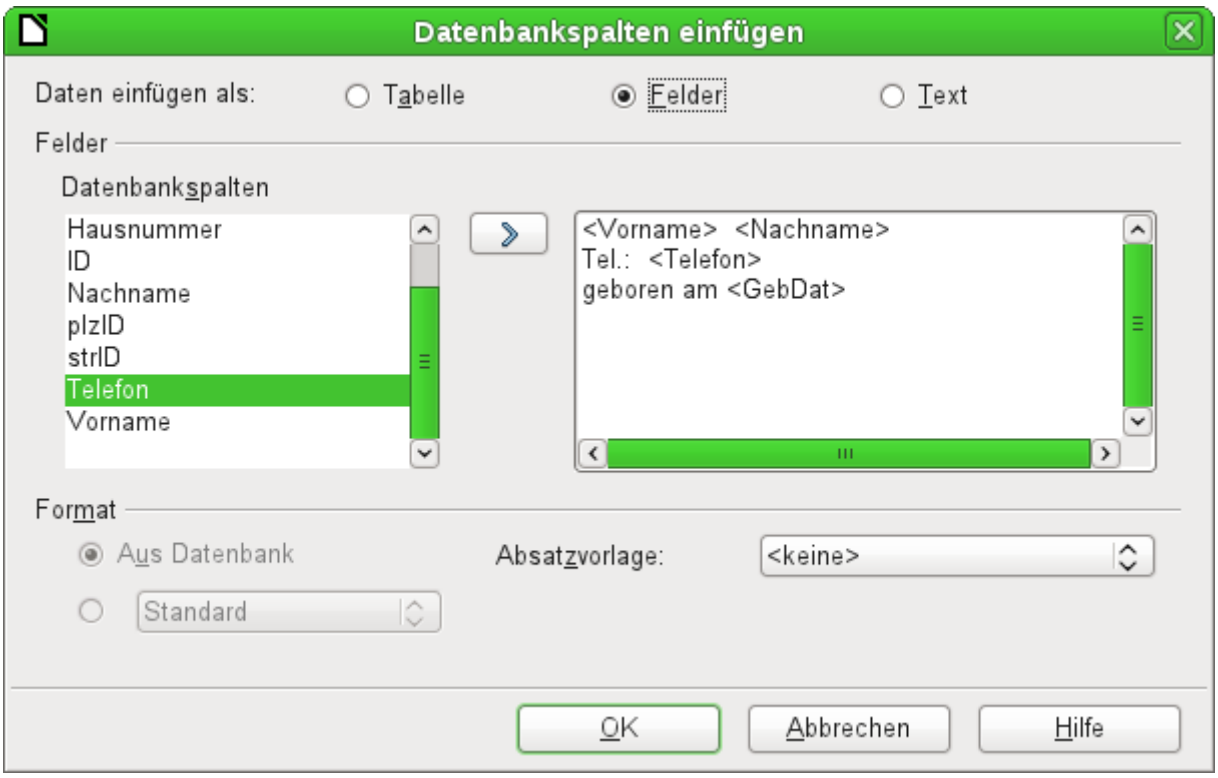


Abbildung 55: ***Daten einfügen als → Felder*** *entspricht auch dem Fenster für* ***Daten einfügen als → Text***

Die auf diese Weise in den Text eingefügten Felder können nachher auch einzeln gelöscht oder als Serienbrieffelder weiter genutzt werden.

Wird **Daten einfügen als → Text** gewählt, so unterscheidet sich das gegenüber den Feldern nur in sofern, als bei den Feldern weiterhin die Datenbankverknüpfung bestehen bleibt. Bei der Einfügung als Text wird lediglich der direkte Inhalt der jeweiligen Felder übernommen, nicht aber die Verknüpfung zur Datenbank selbst. Daher unterscheidet sich auch das Fenster für diese Funktion nicht von dem Fenster der vorhergehenden.

Das Ergebnis der beiden Funktionen sieht im Vergleich so aus:

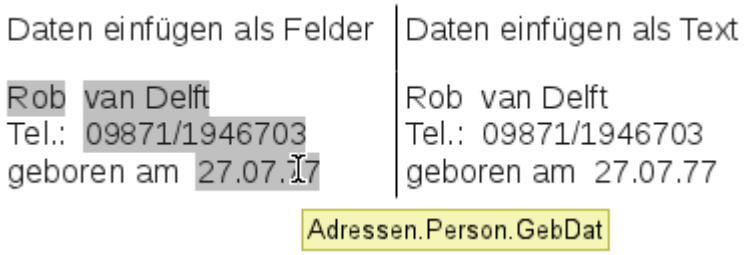


Abbildung 56: Vergleich: Daten als Felder - Daten als Text

Die Felder sind grau hinterlegt. Wird mit einer Maus über die Felder gefahren, so zeigt sich, dass die Felder weiterhin eine Verbindung zur Datenbank "*Adressen*", zur dortigen Tabelle "*Person*" und zum, in dieser Tabelle befindlichen, Feld "*GebDat*" haben.

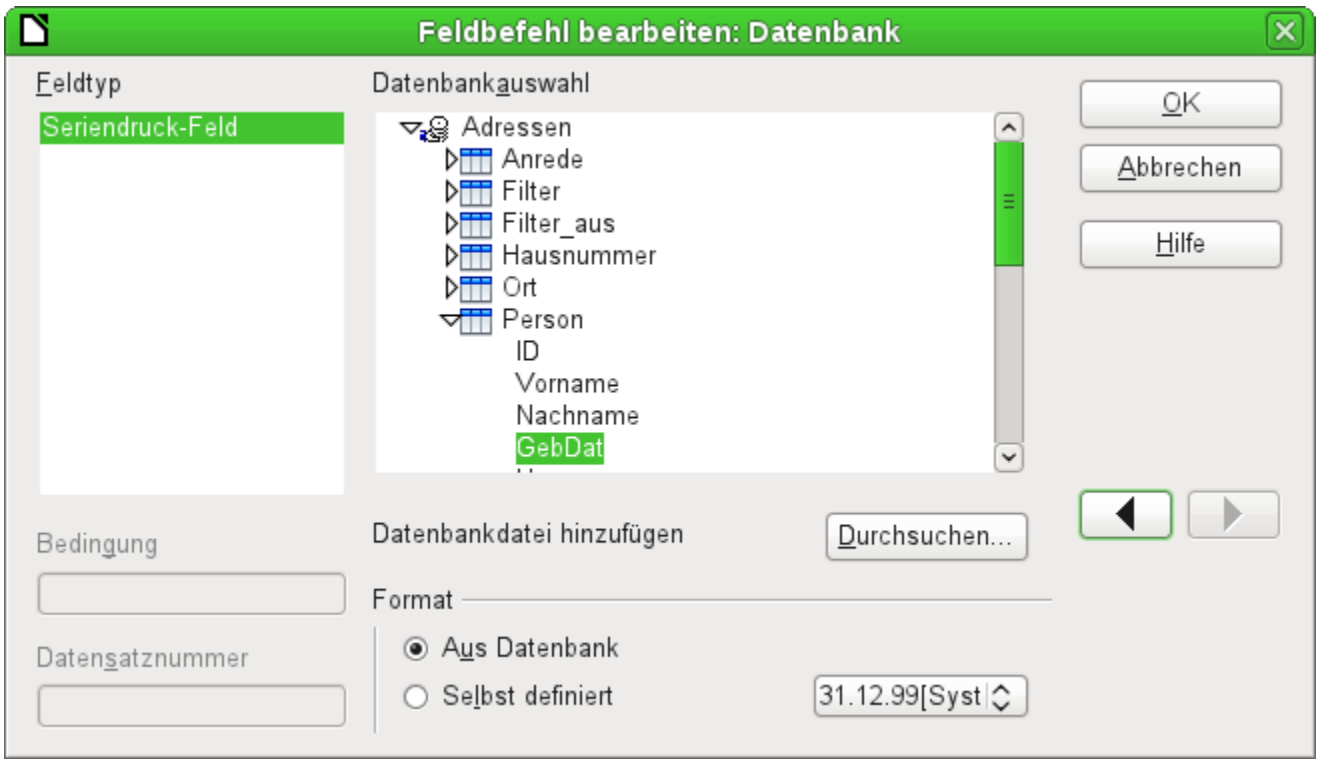


Abbildung 57: Doppelklick auf ein eingefügtes Feld zeigt die Eigenschaften des Seriendruckfeldes

Ein Doppelklick z.B. auf das Feld "*GebDat*" öffnet schließlich folgende Übersicht. Hier wird dann klar, welches Feld letztlich über die Funktion **Daten einfügen als → Felder** erstellt wurde. Es ist der gleiche Feldtyp, der auch über **Einfügen → Feldbefehl → Weitere Feldbefehle → Datenbank** erzeugt wird.

Einfacher lässt sich so ein Feld übrigens erzeugen, wenn im Datenquellenbrowser der Spaltenkopf der Tabelle markiert und mit der Maus in das Dokument gezogen wird. So kann direkt ein Serienbrief erstellt werden.

Daten in Felder

Über **Daten einfügen als → Felder** im vorherigen Abschnitt wurden in einem Writer-Dokument Seriendruckfelder erzeugt. Wird jetzt im Datenquellen-Browser ein anderer Datensatz markiert und dann Daten in Felder ausgewählt, so werden die vorherigen dargestellten Daten durch die neuen Daten ersetzt:

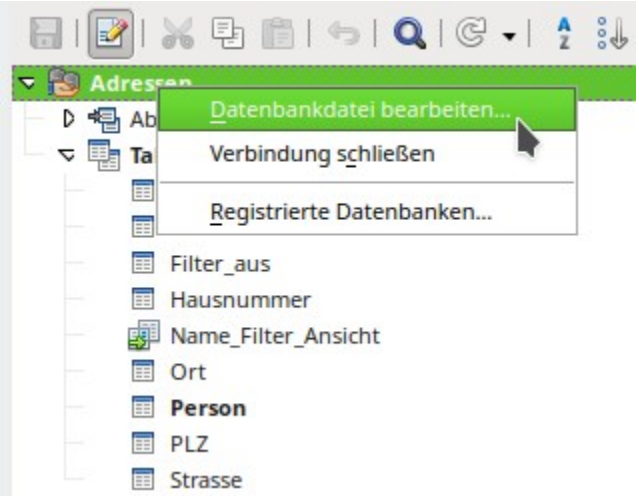
Daten einfügen als Felder	Daten einfügen als Text
Mirina Milinda Tel.: 487531 geboren am 08.07.09	Rob van Delft Tel.: 09871/1946703 geboren am 27.07.77

Hier wurde ein anderer Datensatz markiert. Während bei Betätigung von Daten in Felder die vorher eingefügten Felder auf den neuen Datensatz umgeschrieben werden, bleibt der über **Daten einfügen als → Text** erstellte Text bestehen.

Seriendruck

Mit dem Button Seriendruck wird der Seriendruck-Assistent gestartet. Da für einen Serienbrief im obigen Beispiel die Daten aus verschiedenen Tabelle zusammengefasst werden müssen,

muss zuerst einmal die Datenbank gestartet werden. In der Datenbank ist dann eine neue Abfrage zu erstellen, die die Daten entsprechend zur Verfügung stellt.



Erfolgt der Start der Datenbank durch einen rechten Mausklick auf die Datenbank oder eine der Tabellen oder Abfragen der Datenbank, so wird die Ansicht im Datenquellenbrowser anschließend sofort aktualisiert. Anschließend kann dann der Serienbrief-Assistent über den entsprechenden Button aufgerufen werden.

Aktuelle Dokument-Datenquelle

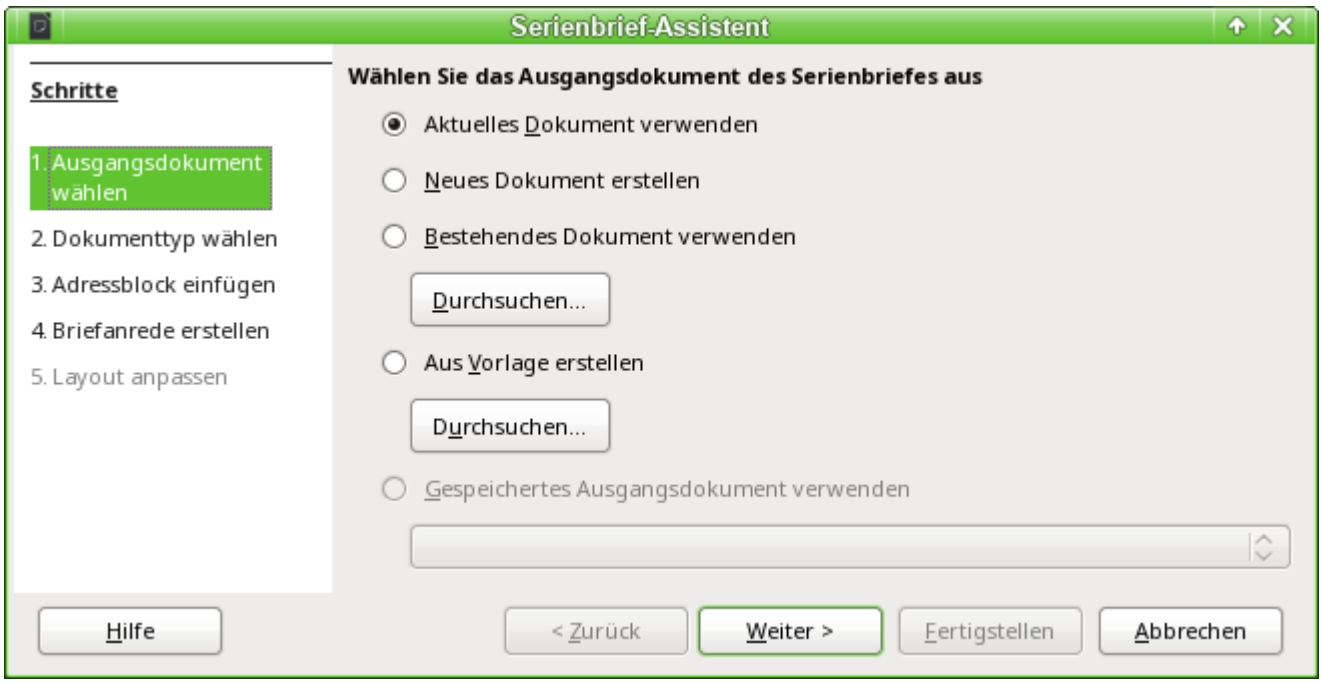
Ein Klick auf den Button Aktuelle Dokument-Datenquelle öffnet direkt die Ansicht auf die Tabelle, die die Grundlage für die Daten bildet, die in das Dokument eingefügt wurden. Im obigen Beispiel zeigt sich also sofort die Tabelle "*Person*" aus der Datenbank "*Adressen*".

Explorer ein/aus

Mit Betätigung des Buttons Explorer ein/aus wird die Ansicht auf den Verzeichnisbaum links von der Tabellenansicht ein- und ausgeschaltet. Dies dient dazu, gegebenenfalls mehr Platz für eine Sicht auf die Daten zu erhalten. Um auf eine andere Tabelle zugreifen zu können, muss der Explorer eingeschaltet werden.

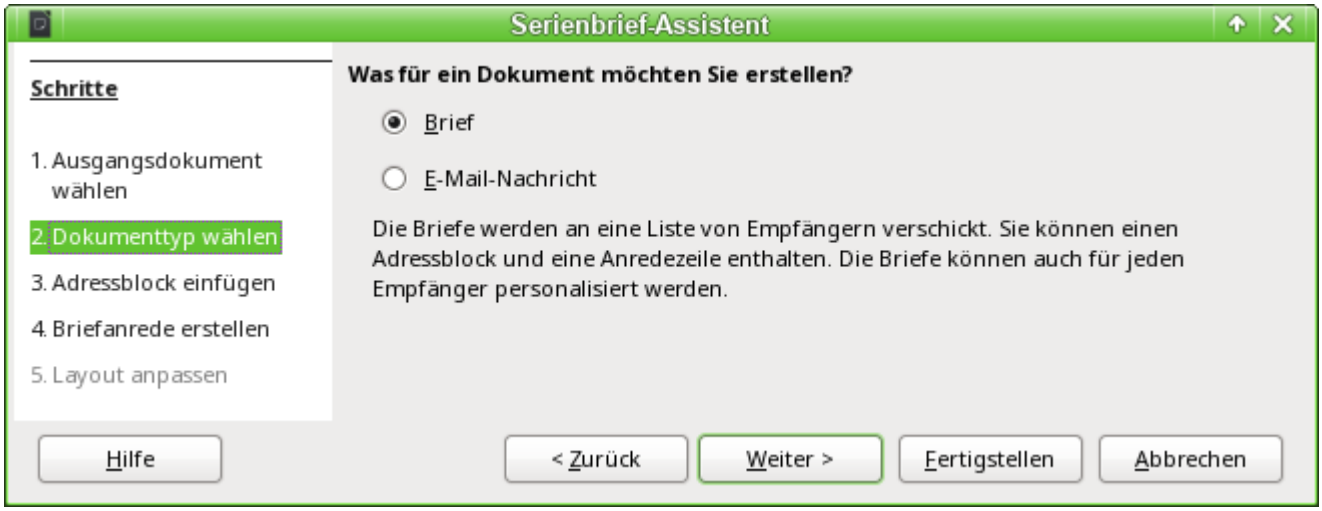
Serienbrieferstellung

Über den Datenbankbrowser ist auch der Serienbriefassistent zugänglich. Mit diesem Assistenten wird kleinschrittig das Adressfeld und die Anrede anhand einer Datenquelle nachvollzogen. Prinzipiell ist die Erstellung von solchen Feldern natürlich auch ohne den Assistenten möglich. Hier wird einmal beispielhaft der gesamte Assistent durchgearbeitet.

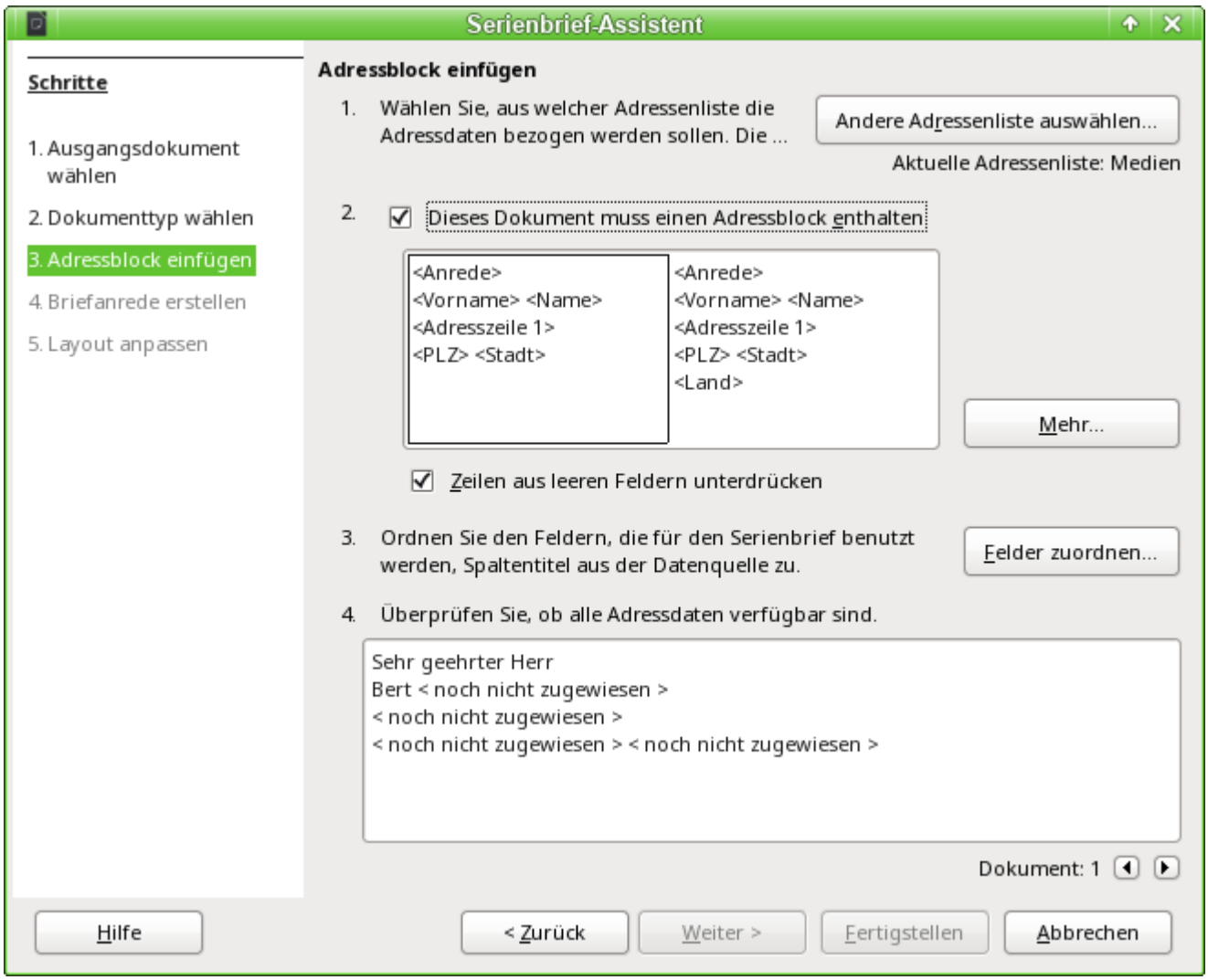


Das **Ausgangsdokument** des Serienbriefs ist das Dokument, mit dem die **Datenbankfelder verknüpft** werden.

Das **Serienbriefdokument** hingegen ist das, in dem nachher die Daten der verschiedenen Personen stehen, die den Serienbrief erhalten haben. Im Serienbriefdokument ist **keine Verknüpfung** zur Datenquelle mehr vorhanden. Dies entspricht der Einstellung aus «*Daten einfügen als Text*'.



Mit dem Serienbrief-Assistenten können entweder Briefe oder E-Mails entsprechend mit Daten aus der Datenbank versorgt werden.



Das Einfügen des Adressblocks erfordert die umfangreichsten Einstellungen. Als Adressenliste wird erst einmal die aktuell gewählte Abfrage oder Tabelle der aktuell gewählten Datenbank vorgeschlagen.

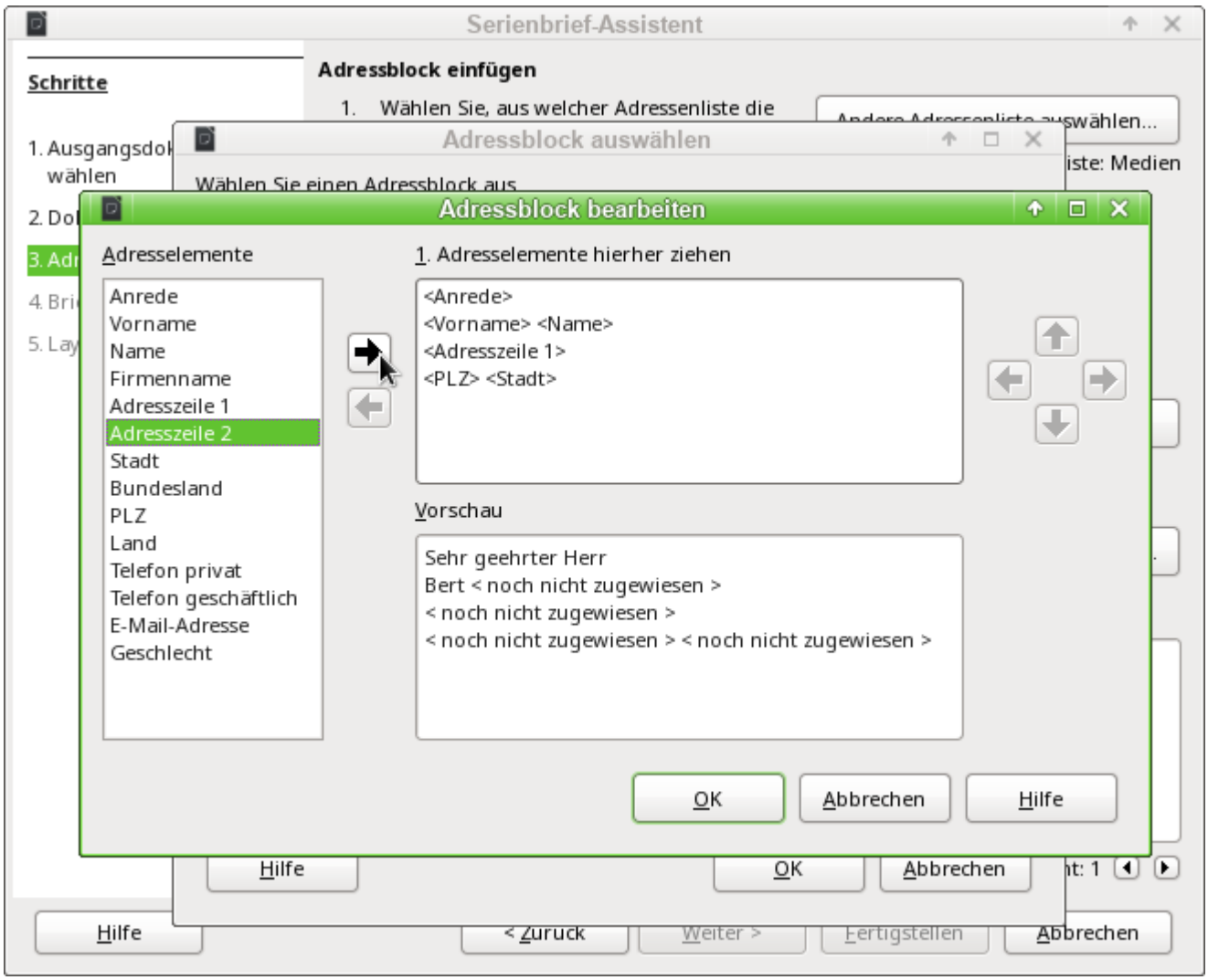
Unter Punkt 2 wird das prinzipielle Aussehen des Adressblocks festgelegt. Dieser Adressblock kann entsprechend über den Button Mehr angepasst werden. Siehe hierzu die nächste Abbildung.

Punkt 3 dient dazu, den entsprechend angedeuteten Feldern aus dem Adressblock die richtigen Felder aus der Datenbank zuzuweisen. Der Assistent erkennt zunächst einmal nur die Felder aus der Datenbank, die genau so geschrieben wurden wie die Felder des Assistenten.

Unter Punkt 4 werden die Adressen angezeigt. Mit den Pfeiltasten können verschiedene Adressen aus der Datenbank abgerufen werden. Bei der abgebildeten Adresse fallen 2 Elemente auf, die einer Nachbearbeitung bedürfen:

- Eine Anrede fehlt
- Bis auf den Vornamen sind alle anderen Felder < *noch nicht zugewiesen* > , da die Benennung in der Datenbank anders ist als die Feldbenennungen, mit denen der Assistent zuerst einmal arbeitet.

Um diese Fehler zu bearbeiten, muss zuerst der Adressblock aus Punkt 2 nachbearbeitet werden.

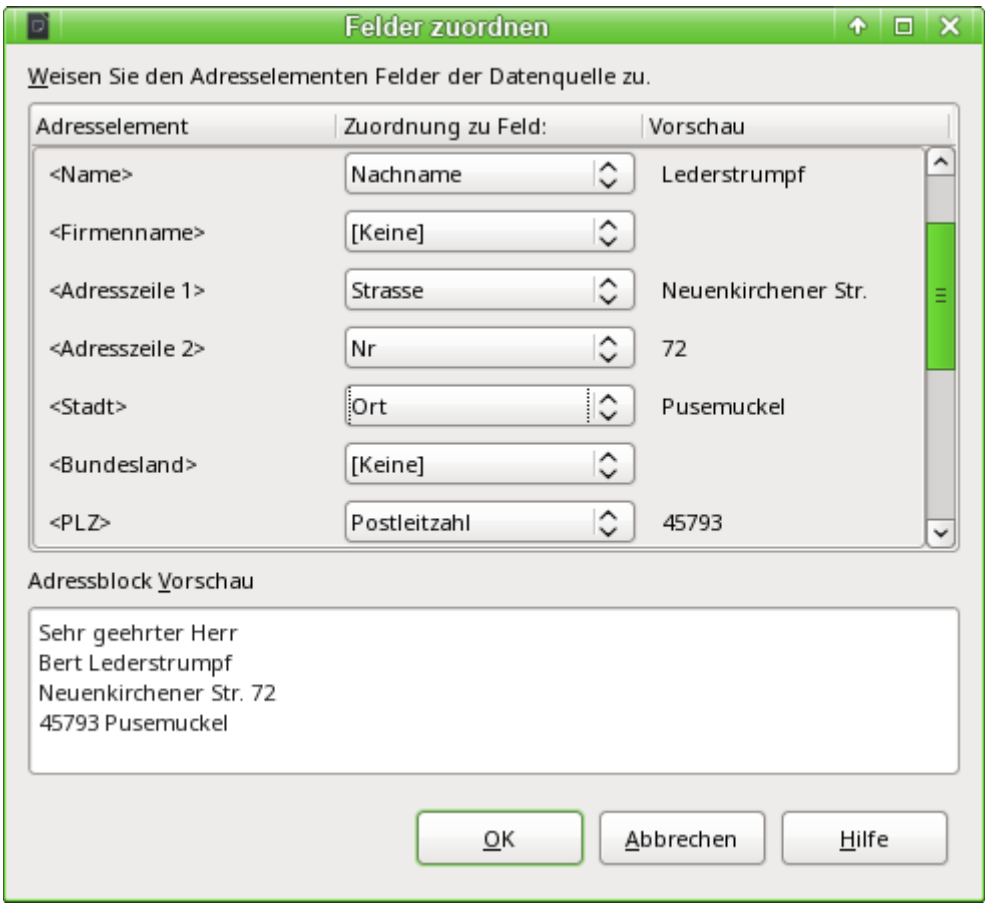


Im Hintergrund ist zu erkennen, dass bei der Nachbearbeitung zuerst einmal wieder eine erweitere Auswahl von Adressblöcken präsentiert wird. Der passendste Adressblock wurde hier ausgewählt und entsprechend bearbeitet.

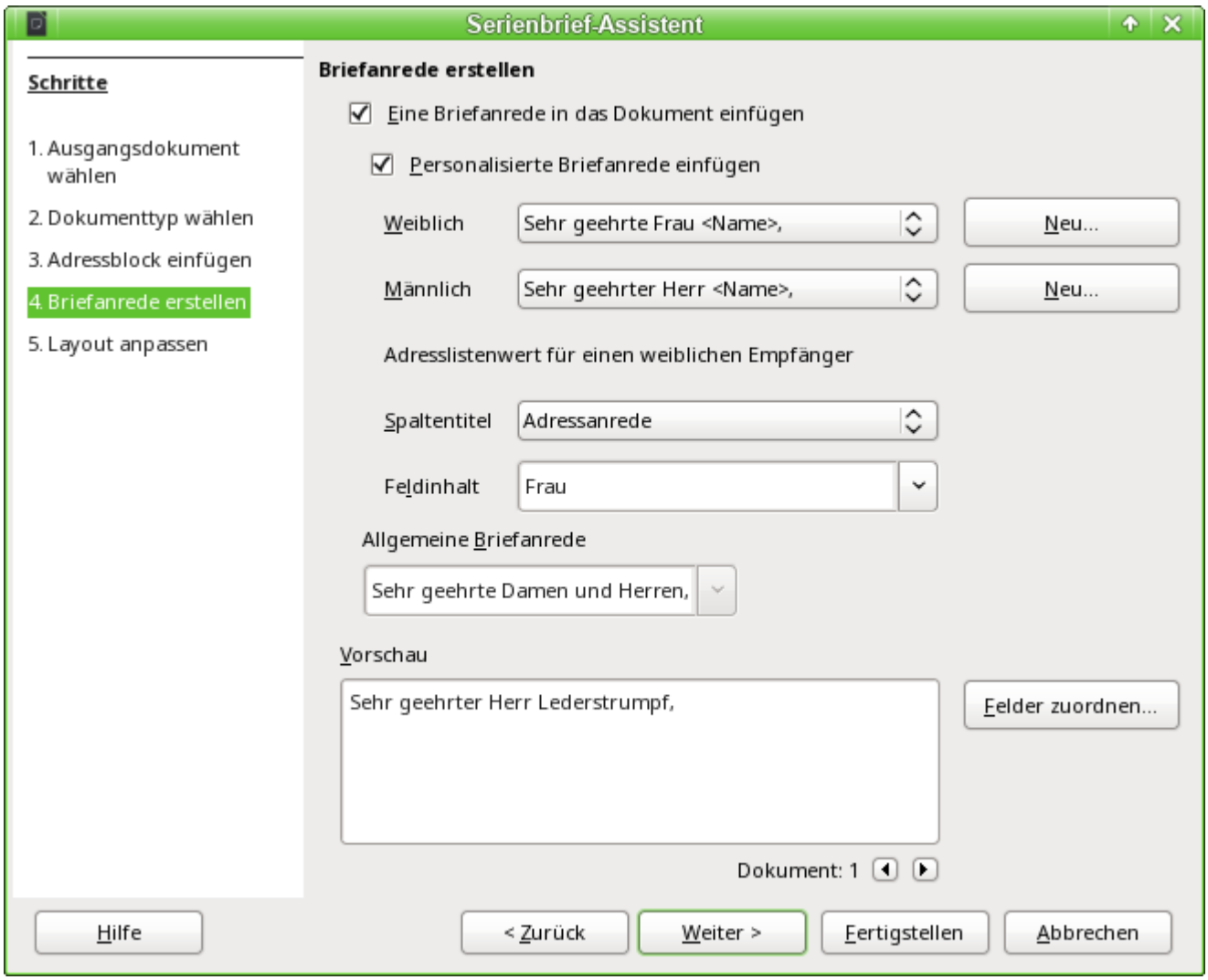
Mit den Pfeiltasten werden Felder in den Adressblock übernommen bzw. aus dem Adressblock entfernt. Der Adressblock kann nicht direkt editiert werden. Stattdessen wird die Anordnung der Felder über die auf der rechten Seite sichtbaren Pfeiltasten hergestellt.

Für die Adressanrede wurde einfach das Element «*Anrede*» eingefügt. Bis auf den Vornamen müssen jetzt noch alle anderen Felder entsprechend zugewiesen werden.

In unserem Fall muss auch das Element «*Anrede*» anders zugewiesen werden, da ein entsprechendes Feld mit etwas anderem Inhalt in der Abfrage existiert und hier aufgrund der Namensgleichheit als "Adressanrede" benutzt wird. Die "Adressanrede" für Rob lautet in der Abfrage der Datenbank «Herrn», die "Anrede" «Herr» für eine gegebenenfalls gesondert zu erstellende Anredezeile zum Beginn des Briefinhaltes.

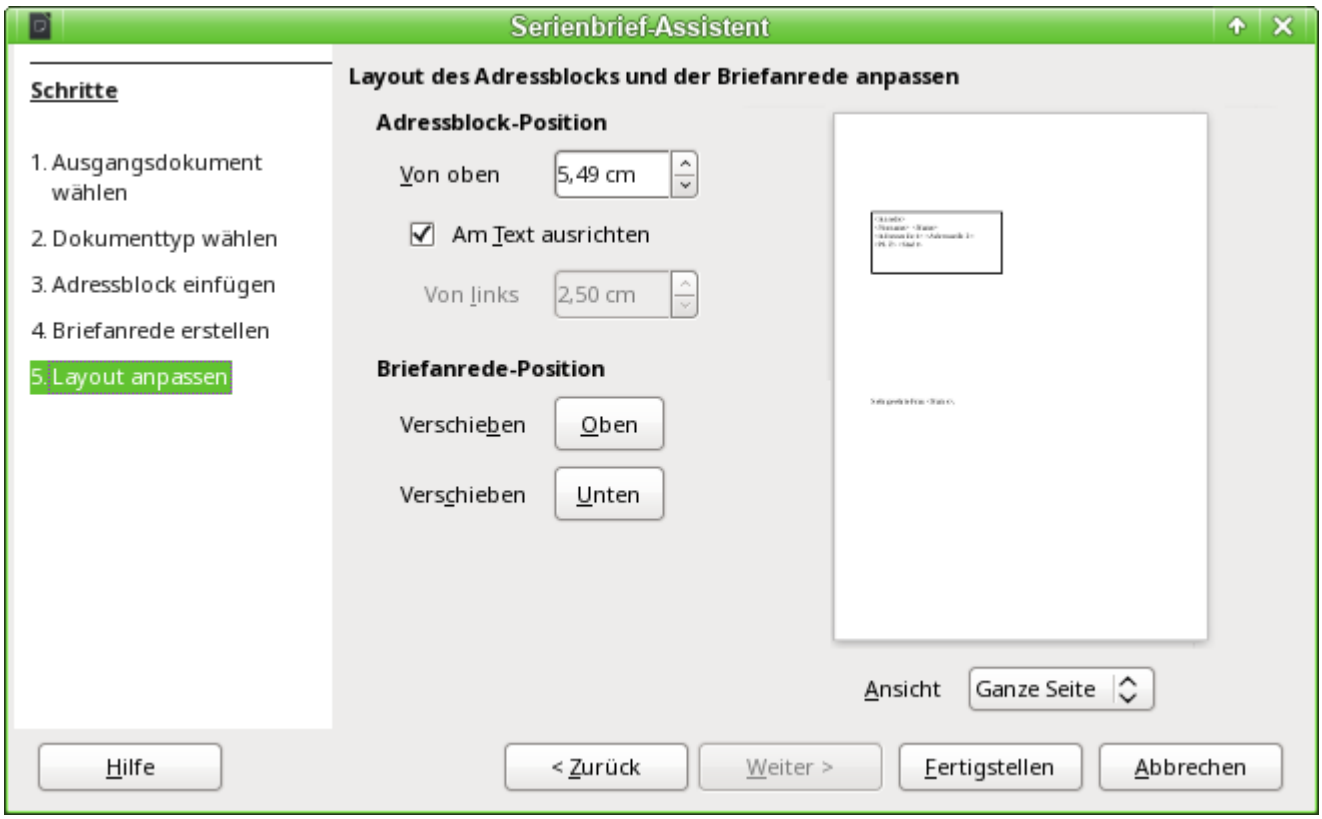


Hier wurden die Adresselemente aus dem Serienbriefassistenten erfolgreich entsprechenden Elementen aus der Abfrage der Datenbank zugeordnet. Als Vorschau dient weiterhin der erste Datensatz der Abfrage.



Die wesentlichen Datenbankeinstellungen werden mit dem 4. Schritt eigentlich schon beendet. Hier geht es lediglich darum, aus welchem Feld das Geschlecht des Adressaten abgelesen werden soll. Dies war bereits so benannt, dass nur noch der Feldinhalt für den weiblichen Empfänger gekennzeichnet werden musste.

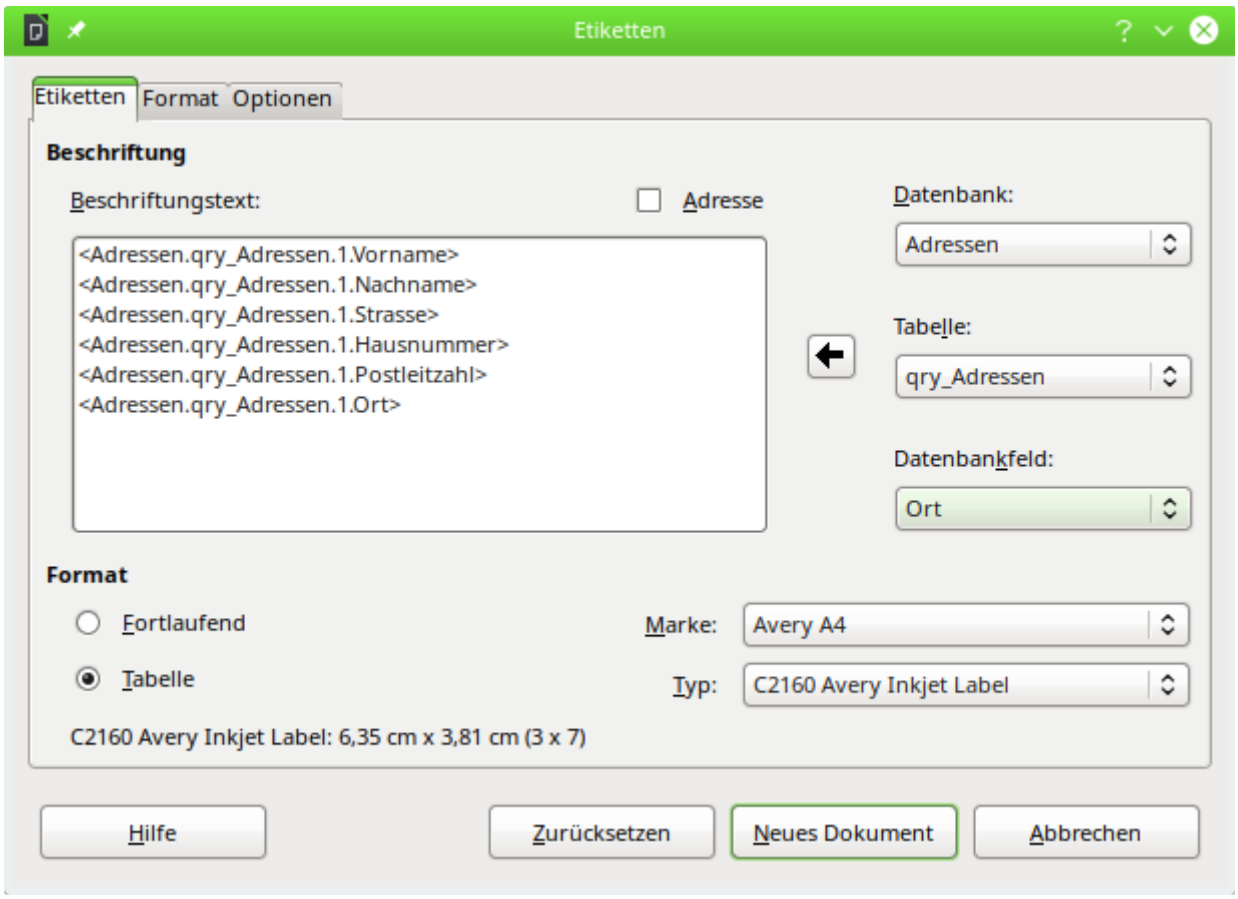
Drei verschiedene Briefanreden werden so erzeugt. Alle mit einem 'w' versehenen Datensätze starten mit '*Sehr geehrte Frau ...*', alle mit 'm' versehenen Datensätze mit '*Sehr geehrter Herr ...*' . Ist kein Geschlecht angegeben, so wird schließlich '*Sehr geehrte Damen und Herren*' gewählt.



In der Regel ist das Dokument erst einmal eine Rohform, so dass es für den Gebrauch im Writer noch weiter bearbeitet werden kann.

Erstellung von Etiketten

Über **Dateien → Neu → Etiketten** wird der Assistent zur Erstellung von Etiketten gestartet. Es öffnet sich ein Fenster, das alle Formatierungs- und Inhaltsfragen zu den Etiketten abfragt, bevor die Etiketten selbst erstellt werden. Die Einstellungen dieses Fensters werden in den persönlichen Einstellungen des Nutzers gespeichert.



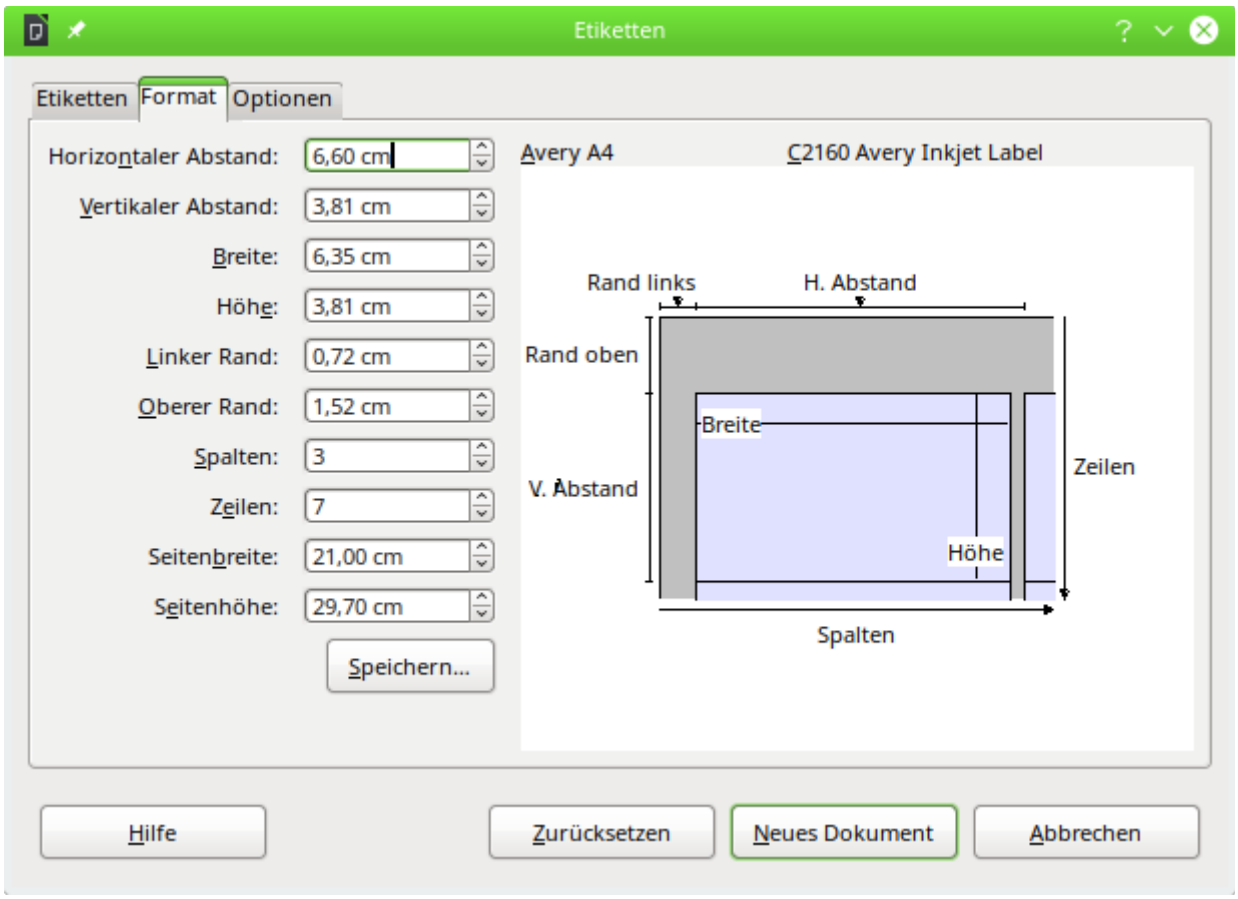
Die Grundeinstellungen zum Inhalt werden im Reiter **Etiketten** erstellt. Wird hier statt des Datenbankinhaltes über das Markierfeld **Adresse** gewählt, so wird jedes Etikett mit dem gleichen Inhalt ausgefüllt, der aus den Einstellungen in **Extras → Optionen → LibreOffice → Benutzerdaten** gelesen wird.

Als Beispieldatenbank dient hier wieder die **Datenbank → Adressen**. Auch wenn über dem nächsten Selektionsfeld der Begriff «*Tabelle*» steht, werden hier **Tabellen und Abfragen** gelistet, wie sie im Datenquellenbrowser verfügbar sind.

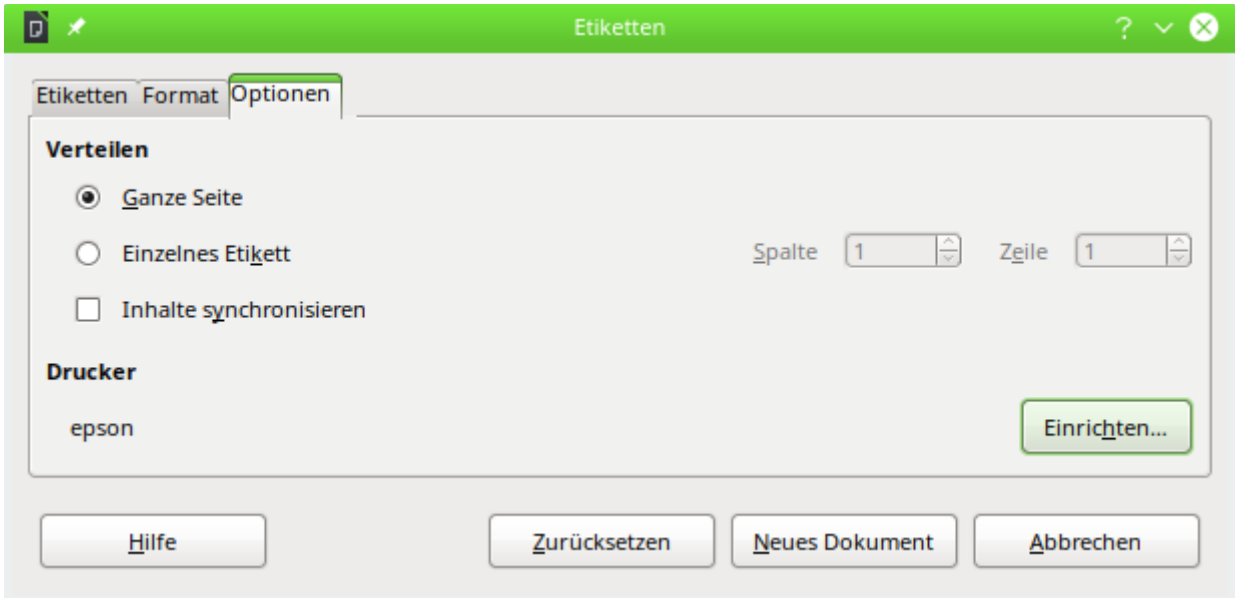
Über die Pfeiltasten werden die einzelnen Datenbankfelder in den Editor eingefügt. Die Bezeichnung für das Datenbankfeld "*Vorname*" wird hier zu *<Adressen.qry_Adressen.1.Vorname>*. Die Reihenfolge ist also *<Datenbank.Tabelle.1.Datenbankfeld>*.

In dem Editor kann mit der Tastatur gearbeitet werden. So ist es z. B. möglich, zu Beginn einen Zeilenumbruch einzufügen, damit die Etiketten nicht direkt an der oberen Kante beschriftet werden, sondern der Inhalt möglichst komplett und gut sichtbar gedruckt werden kann.

Das Format kann im Reiter **Etiketten** vorgewählt werden. Hier sind viele Etikettenmarken eingearbeitet, so dass meist keine weiteren Einstellungen im Reiter **Format** notwendig sind.



Unter dem Reiter **Format** lässt sich eine genaue Einstellung der Etikettengröße vornehmen. Die Einstellungen haben nur eine Bedeutung, wenn die Marke und der Typ nicht bekannt sind. Wichtig wäre hier bei Etiketten der Breite 7,00 cm, die Seitenbreite geringfügig größer als 3*7,00 cm = 21,00 cm zu stellen. Erst dann werden 3 Etiketten nebeneinander auf einer Seite ausgegeben, wenn gleichzeitig der linke Rand 0 cm und der horizontale Abstand 7,00 cm ist.



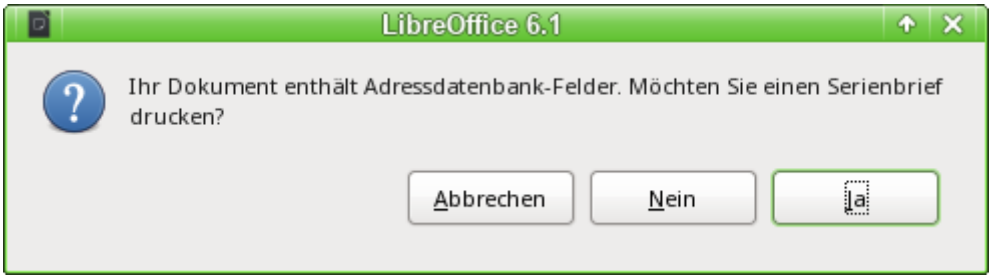
Unter dem Reiter **Zusätze** kann schließlich noch eingestellt werden, ob nur ein einzelnes Etikett oder eine ganze fortlaufende Seite erstellt werden soll. Die fortlaufende Seite wird dann, beginnend mit dem ersten Datensatz, mit Daten aus der Datenbank bestückt. Sind mehr Datensätze

vorhanden als auf eine Seite passen, so wird automatisch das nächste Blatt mit den fortlaufenden Daten versehen.

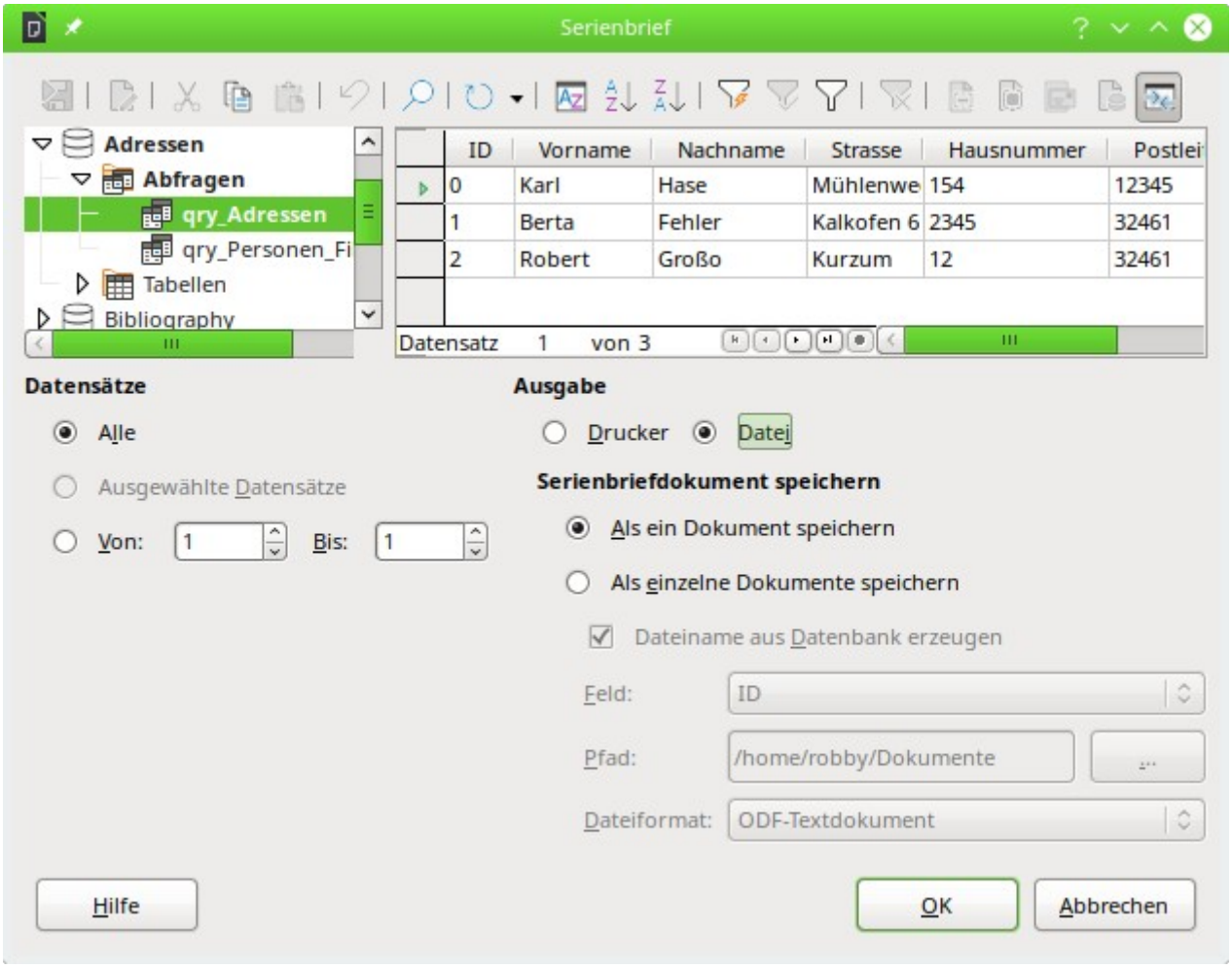
Mit dem Markierfeld **Inhalte synchronisieren** werden alle Etiketten miteinander verbunden, so dass die anschließenden Änderungen im Layout einer Etikette auf die anderen Etiketten übernommen werden. Um den editierten Inhalt zu übertragen, muss lediglich die eingeblendete Schaltfläche mit dem Aufdruck Synchronisieren betätigt werden.

Über den Button Neues Dokument wird schließlich ein Dokument mit Serienbrieffeldern erzeugt.

Mit dem Aufruf des Druckvorganges erscheint die folgende Nachfrage:



Erst wenn diese Nachfrage mit Ja beantwortet wird, werden die Adressdatenbank-Felder auch mit einem entsprechenden Inhalt gefüllt.



Ist die Abfrage ausgesucht und sind die entsprechenden Datensätze ausgewählt (hier: **Alle**), so kann mit dem Druck begonnen werden. Ratsam vor allem bei ersten Tests ist **Ausgabe → Datei**. So wird der Inhalt als ein Dokument gespeichert wird. Die Speicherung in mehrere Dokumente

dient hier nicht dem Etikettendruck, sondern dem Druck von Briefen an unterschiedliche Personen, die so separat nachbearbeitet werden können.

Serienbriefe und Etiketten direkt erstellen

Neben den Assistenten steht natürlich der Weg offen, durch möglichst direkte Interaktion Serienbriefe und Etiketten selbst zu erstellen.

Serienbrieffelder mit der Maus erstellen

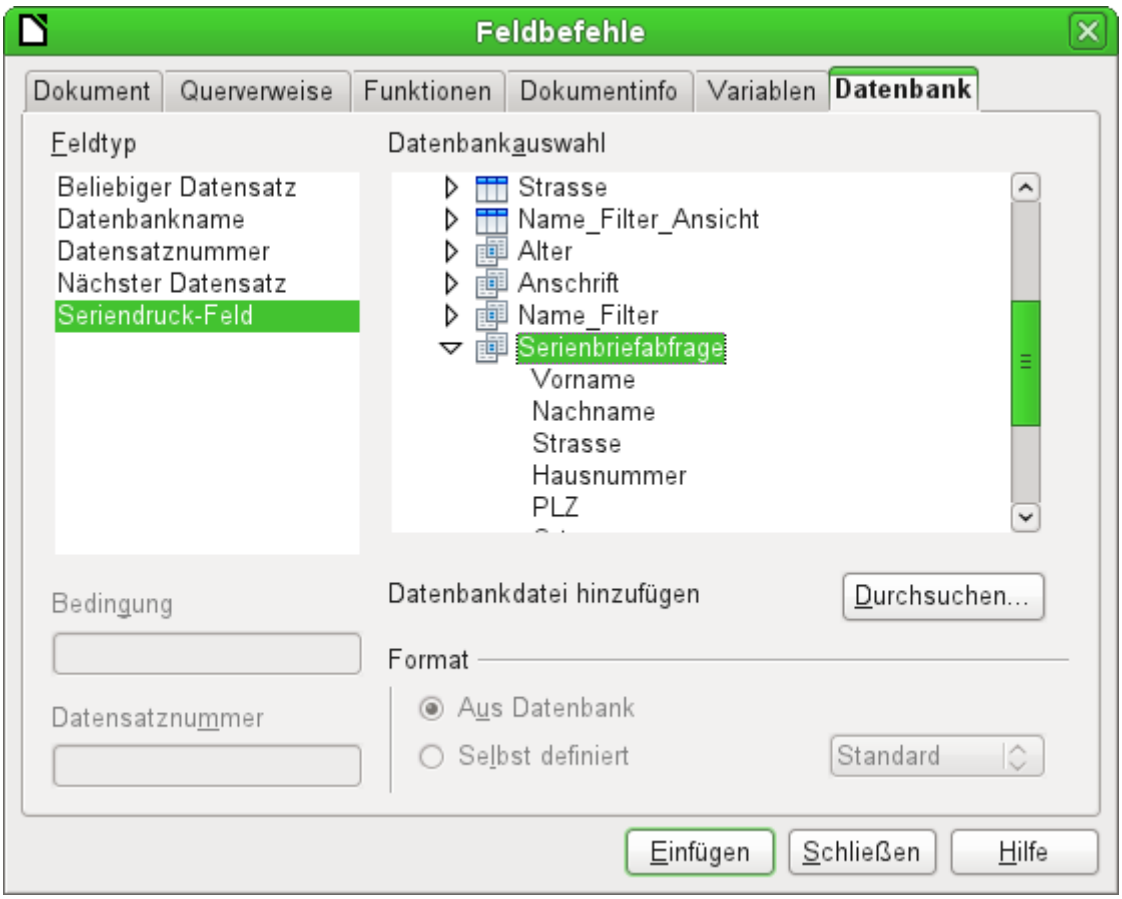
Serienbrief-Felder können mit der Maus aus dem Datenbankbrowser heraus erstellt werden:

Der Tabellenkopf wird mit der linken Maustaste markiert. Die Maustaste wird gedrückt gehalten und der Cursor in das Textdokument gezogen. Der Cursor verändert sein Symbol zu einem Einfügesymbol. In dem Textdokument wird schließlich das Serienbrieffeld eingefügt, das hier in der kompletten Beschreibung über **Ansicht → Feldname** sichtbar gemacht wurde.

Auch aus einer nicht angemeldeten Datenbank heraus können Serienbrieffelder mit der Maus direkt erstellt werden. Hierzu muss die Datenbank neben dem Writer-Dokument geöffnet sein. Ist eine Tabelle oder Abfrage ausgewählt, so wird mit der linken Maustaste der jeweilige Tabellenkopf markiert und in das Writer-Dokument hinüber gezogen. Der Ablauf entspricht also dem im vorherigen Abschnitt geschilderten Vorgehen.

Serienbrieffelder über Feldbefehle erstellen

Serienbrief-Felder können über **Einfügen → Feldbefehl → Weitere Feldbefehle → Datenbank** eingefügt werden.



Hier stehen in der gewählten Datenbank alle Tabellen und Abfragen zur Verfügung. Über den Button Einfügen können nacheinander die verschiedenen Felder direkt in den Text an der momentanen Cursorposition eingebunden werden.

Soll, wie im Serienbrief, eine Anrede erstellt werden, so kann dies mit Hilfe eines versteckten Absatzes oder versteckten Textes funktionieren: **Einfügen → Feldbefehl → Andere → Funktionen → Versteckter Absatz**. Bei beiden Varianten ist zu beachten, dass die Bedingung, die formuliert wird, **nicht** erfüllt sein darf, damit der Absatz erscheint.

Damit die Formulierung '*Sehr geehrte Frau <Nachname>,*' nur dann erscheint, wenn die Person weiblich ist, reicht als Bedingung

```
001 [Adressen.Serienbriefabfrage.Geschlecht] != "w".
```

Nur kann es jetzt noch passieren, dass kein Nachname existiert. Unter diesen Umständen sollte dort stattdessen '*Sehr geehrte Damen und Herren,*' erscheinen. Also ist diese Bedingung hinzuzufügen. Insgesamt ergibt das die Bedingung:

```
001 [Adressen.Serienbriefabfrage.Geschlecht] != "w"
002 OR NOT [Adressen.Serienbriefabfrage.Nachname]
```

Damit ist schließlich ausgeschlossen, dass der Absatz erscheint, wenn die Person nicht weiblich ist oder kein Nachname eingetragen wurde.

Entsprechend kann mit den Eintragungen für das männliche Geschlecht und für fehlende Eintragungen für die beiden verbleibenden Anreden verfahren werden.

Auf die gleiche Art und Weise kann natürlich auch eine Anrede im Adressfeld erstellt werden, soweit eben das Geschlecht angegeben wurde.

Weitere Informationen hierzu in der Hilfestellung unter dem Titel «*Versteckter Text*» bzw. «*Bedingter Text*».

Noch einfacher wäre es natürlich für Personen mit Datenbankkenntnissen, die gesamte Anrede bereits in der Abfrage zu hinterlegen. Dies ist über eine *Korrelierte Unterabfrage* (Kapitel «Abfragen») möglich.

Für Etiketten besonders interessant ist der Feldtyp «Nächster Datensatz». Wird dieser Feldtyp am Schluss einer Etikette gewählt, so wird die nächste Etikette mit dem darauffolgenden Datensatz gefüllt. Typische Etiketten für den fortlaufenden Etikettendruck sehen also wie im folgenden Bild aus, wenn über **Ansicht → Feldnamen** die entsprechenden Bezeichnungen sichtbar gemacht werden:

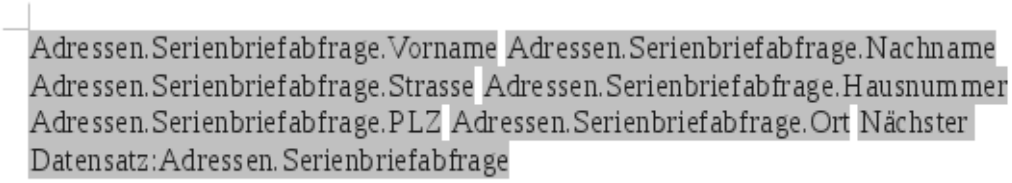


Abbildung 58: Feldbefehle für Etiketten mit fortlaufendem Inhalt

Bei dem letzten Etikett auf der Seite ist allerdings darauf zu achten, dass hier der nächste Datensatz schon automatisch mit dem Seitenumbruch aufgerufen wird. Dort darf also der Feldtyp «Nächster Datensatz» nicht erscheinen. Sonst fehlt ein Datensatz, weil ein doppelter Datensatzsprung enthalten ist.

Tipp

Das Erstellen von Serienbriefen ist auch direkt im Formular einer Datenbank möglich. Voraussetzung ist lediglich, dass die Datenbank in LibreOffice registriert wird.
Beim Erstellen der Serienbriefe sollte allerdings darauf geachtet werden, dass **Ansicht → Drucklayout** gewählt wird. Dadurch ist sichergestellt, dass die Elemente anschließend auch an der korrekten Position auf dem Blatt erscheinen. Wird ein so erstelltes Formular gedruckt, so erscheint die übliche Serienbriefabfrage.
Serienbriefe dieser Art haben den Vorteil, dass neben der *.odb-Datei keine weiteren Dateien für den Druck existieren müssen.

Externe Formulare

Sollen einfache Formulareigenschaften unter LibreOffice in anderen Programmteilen wie Writer und Calc genutzt werden, so reicht es aus, über **Ansicht → Symbolleisten → Formularentwurf** die Formularentwurfsleiste anzeigen zu lassen, den «*Formularnavigator*» zu öffnen und ein Formular zu gründen oder, wie im Kapitel «Formulare» beschrieben, eine *Formulargründung über ein Formularfeld*. Auch die direkte Auswahl von Elementen über **Formular → Entwurfsmodus** ist seit LO 6.0 möglich. Bei der Erstellung des Formulars erscheint unter **Formular-Eigenschaften → Daten** ein etwas anderer Aufbau als bei Formularen direkt in der Datenbankdatei *.odb:

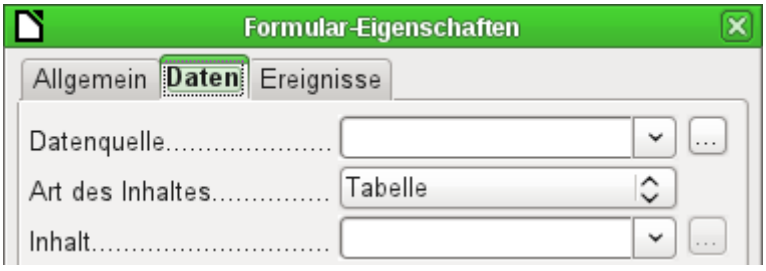


Abbildung 59: Formular mit einer externen ...

Abbildung 60: ... und einer internen Datenquelle.

Lediglich die **Datenquelle** muss bei externen Formularen zusätzlich ausgewählt werden. Geschieht dies mit dem Button ... rechts von dem Listenfeld zur Datenquelle, so wird der Dateibrowser geöffnet. Eine beliebige *.odb-Datei kann ausgewählt werden. Anschließend steht in dem Feld zur Datenquelle ein Link, der mit der Bezeichnung «file:///» beginnt.

Wird stattdessen nur im Listenfeld gesucht, so stehen dort die bereits in LibreOffice registrierten Datenbanken unter dem registrierten Namen zur Verfügung.

Die Formulare werden im Weiteren dann genau so erstellt wie unter Base selbst. Zur Formularerstellung gilt daher das Gleiche wie unter Base.

Die so erstellten Formulare werden standardmäßig bei jedem neuen Öffnen der Datei im Bearbeitungsmodus und nicht, wie in Base, schreibgeschützt geöffnet. Um eine versehentliche Änderung des Formulars zu vermeiden kann über **Datei → Eigenschaften → Sicherheit** die Datei schreibgeschützt geöffnet werden. Das Formular kann hier sogar mit einem Passwort gegen Veränderungen geschützt werden. Auf Betriebssystemebene lässt sich letztlich auch die ganze Datei als schreibgeschützt deklarieren. Dann sind immer noch die Eingaben in die Formularfelder möglich, aber nicht mehr das Verschieben der Felder oder eine Texteingabe zwischen den Feldern.

Tipp

Das Erstellen von Formularen kann auch im Schnellverfahren durch Ziehen und Ablegen erfolgen. Dafür ist die Datenbank zu öffnen, die Tabelle bzw. Abfrage aufzusuchen und der jeweilige Tabellenkopf zu markieren.
Für den Writer gilt: Mit der linken Maustaste markieren, *Shift*- und *Strg*-Taste gedrückt halten, so dass ein Verknüpfungssymbol als Mauscursor erscheint, in das Writerdokument ziehen.
Bei Calc-Dateien muss ohne Zuhilfenahme zusätzlicher Tasten in das Calc-Dokument gezogen werden. Dort erscheint das Kopiersymbol als Mauscursor.
In beiden Fällen wird ein Eingabefeld sowie ein dazugehöriges Beschriftungsfeld mit der Bezeichnung des Feldnamens erzeugt. Beim ersten Einfügen wird gleichzeitig die Verbindung zur Datenquelle erstellt. So kann direkt nach dem Ziehen und Ablegen mit der Eingabe von Daten in diesem Formular begonnen werden.

Tipp

Sind bereits in Base Formulare erstellt worden, so können sie aus Base heraus als externe Formulare abgespeichert werden. Es muss dann lediglich noch die Verbindung zur Datenbank erstellt werden und die Formulare funktionieren außerhalb der *.odb-Datei. Sollten Makros in den Formularen genutzt worden sein, so ist hier natürlich gegebenenfalls noch eine Anpassung notwendig.

Vorsicht

Die Nutzung von externen Formularen kann in Zusammenhang mit Firebird zu Datenverlusten führen. Die Firebird Datenbank benötigt auch in LO 7.1 immer noch die Sicherung der Daten über das Speicher-Symbol der Datenbankdatei.
Nur mittels Makro kann diese Sicherung auch bei externen Formularen erfolgen. Dabei sollte das Makro an die Eigenschaft **Nach der Datensatzaktion** gebunden werden:

```
001  SUB DataSave(oEvent AS OBJECT)
002     oEvent.Source.activeConnection.Parent.flush
003  END SUB
```

Vorteil externer Formulare

Base muss nicht erst geöffnet werden, um mit der Datenbank zu arbeiten. Im Hintergrund ist also nicht immer ein weiteres Fenster der Datenbankoberfläche geöffnet.

Bei einer fertigen Datenbank können anderen Nutzern der Datenbank anschließend verbesserte Formulare problemlos zugesandt werden. So können sie während der Entwicklung weiterer Formulare die Datenbank weiter nutzen und müssen nicht, für Außenstehende kompliziert, Formulare aus einer Datenbank in eine andere kopieren.

Formulare zu einer Datenbank können je nach Nutzer der Datenbank unterschiedlich sein. Nutzer, die keine Datenkorrekturen und Neueingaben tätigen, können von anderen Nutzern den jeweils aktuellen Datenbestand zugesandt bekommen und einfach die *.odb-Datei austauschen, um einen aktuellen Bestand zu haben. Dies könnte z.B. bei einer Datenbank für Vereine sinnvoll sein, wo alle Vorstandsmitglieder die Datenbank erhalten, aber nur eine Person Daten letztlich bearbeitet, die anderen aber einen Blick auf die Adressen ihrer jeweiligen Abteilung haben.

Über externe Formulare ist der Zugriff und die Bearbeitung von Inhalten mehrerer Datenbanken möglich. Die unterschiedlichen Datenbanken brauchen nur im jeweiligen Neben- oder Unterformular selbst angegeben zu werden. Auch der Zugriff auf mehrere externe Datenbanken wie z.B. eine Datenbanksammlung unter MySQL/MariaDB ist damit möglich.

Tipp

Externe Formulare sollten schreibgeschützt geöffnet werden. Sonst erfolgt beim Schließen die missverständliche Nachfrage, ob das Dokument gespeichert werden soll. Deshalb: **Datei → Eigenschaften → Sicherheit → Datei schreibgeschützt öffnen**. Der Schreibschutz des Betriebssystems erzeugt hingegen immer die Meldung, dass das Dokument schreibgeschützt ist und ob es denn bearbeitet werden soll.

Nachteil externer Formulare

Andere Nutzer müssen immer Formulare und Base in der gleichen Verzeichnisstruktur installieren. Nur so kann der einwandfreie Kontakt zur Datenbank hergestellt werden. Sind die Datenbanken angemeldet, so muss allerdings lediglich der Anmeldename übereinstimmen.

Nur die Formulare sind extern erstellbar, nicht aber Abfragen und Berichte. Ein einfacher Blick auf eine Abfrage muss also über ein Formular erfolgen. Als Darstellung eignet sich da sicher das Tabellenkontrollfeld ganz gut. Ein Bericht hingegen erfordert die Öffnung der Datenbank. Dies kann auch über Makros mit einem *Druck von Berichten aus einem externen Formular heraus* erfolgen. Alternativ dazu kann er zumindest teilweise mit Hilfe von Serienbrieffeldern nachgestellt werden. Zu weiteren Druckmöglichkeiten siehe auch das entsprechende Beispieldatenbankpaket «Serienbrief»[18].

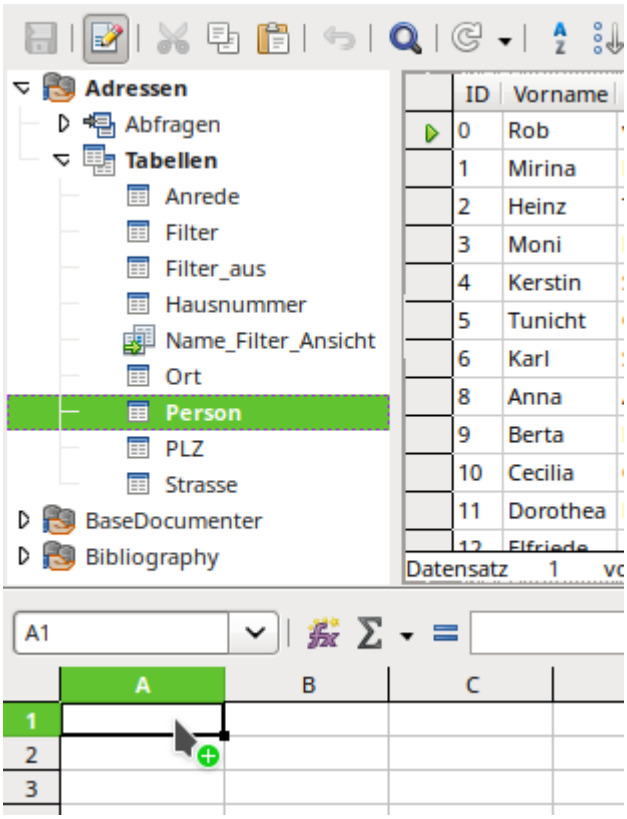
18 «Serienbrief.zip» ist den Beispieldatenbanken für dieses Handbuch beigefügt.

Datenbanknutzung in Calc

Daten können in Calc zu Berechnungszwecken genutzt werden. Dazu ist es notwendig, die Daten zuerst in einem Tabellenblatt von Calc verfügbar zu machen.

Daten in Calc einfügen

Daten lassen sich aus der Datenbank auf verschiedene Weisen in Calc einfügen:



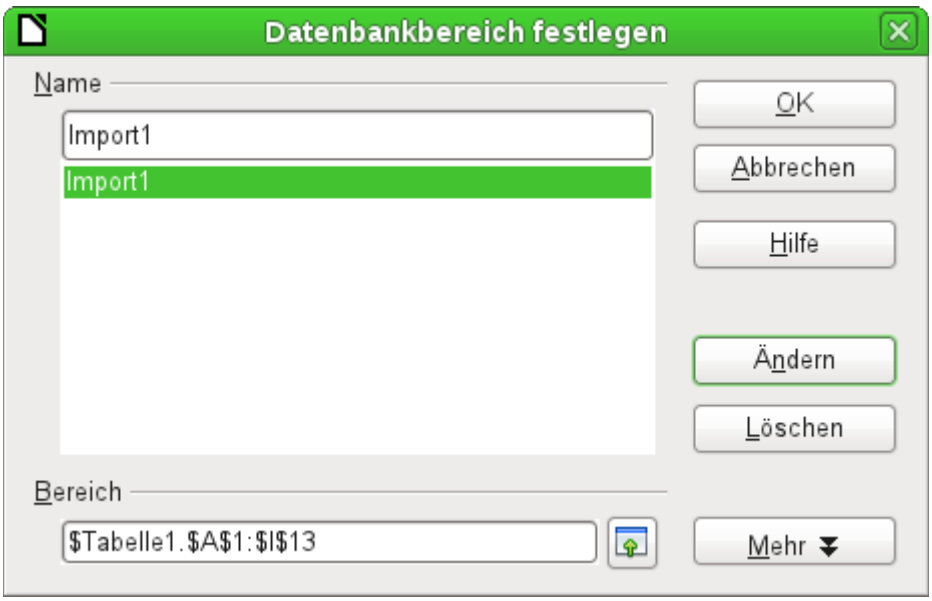
Die Tabelle wird ausgewählt, mit der linken Maustaste markiert und in ein Tabellenblatt von Calc hereingezogen. Mit dem Cursor wird die linke obere Ecke der Tabelle festgelegt. Die Tabelle wird mit Feldbezeichnungen erstellt. Der Datenquellen-Browser bietet bei dieser Aktion nicht erst Daten in Text oder Daten in Felder an.

Die so nach Calc hereingezogenen Daten weisen die folgenden Eigenschaften auf:

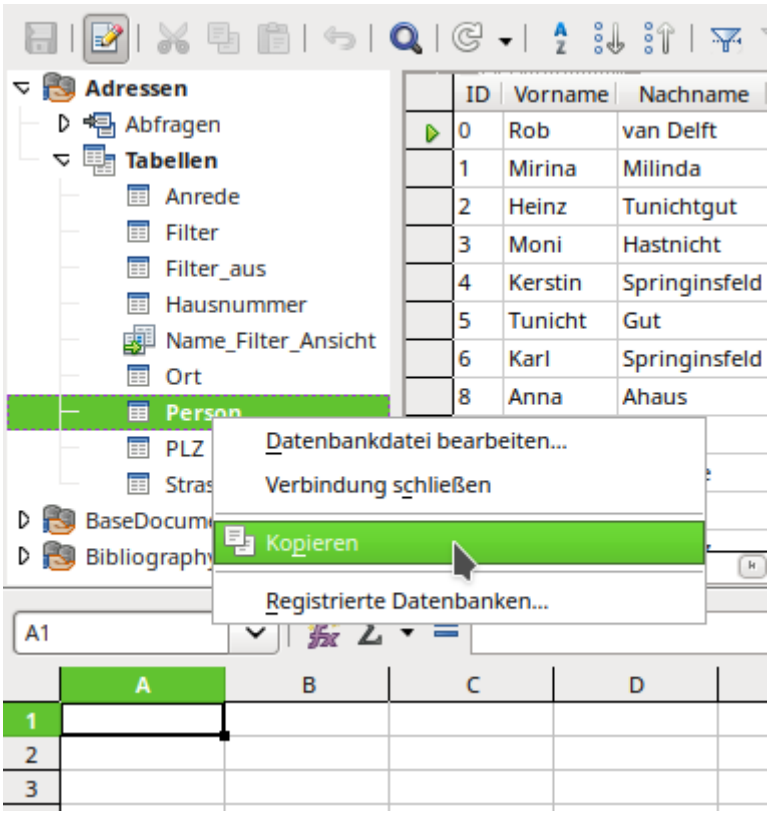
Nicht nur die Daten werden importiert, sondern auch die Eigenschaften der Felder werden ausgelesen und beim Import beachtet. Felder wie z.B. die Hausnummern, die als Textfelder deklariert wurden, werden auch als Text formatiert in Calc eingefügt.

Der Import wird direkt als Bereich eingefügt. Ihm wird standardmäßig der Name Import1 zugewiesen. Über diesen Bereich können die Daten später angesprochen werden. Über **Daten → Bereich aktualisieren** wird der Bereich gegebenenfalls einem neuen Datenstand aus der Datenbank heraus angepasst.

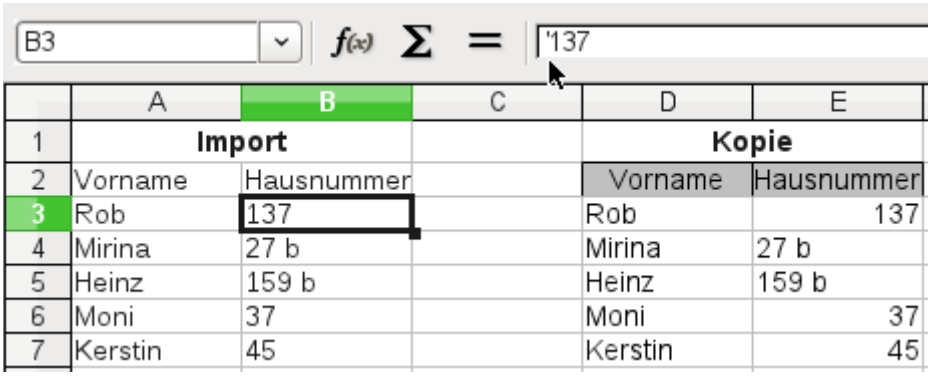
Werden einzelne Datenzeilen oder die komplette Tabelle in der Datenansicht markiert, so kann der Bereich auch über einen Klick auf Daten in Text eingefügt und auch aktualisiert werden. Hierbei werden dann nur die markierten Zeilen der Tabelle neu geschrieben. Alle anderen Zeilen werden aus der Tabelle in Calc entfernt.



Die hereingezogenen Daten sind nicht weiter formatiert als es den Eigenschaften der Datenbankfelder entspricht. Der Bereich ist angegeben, wird aber beim Neueinlesen gegebenenfalls um zusätzlich hinzugekommene Datensätze erweitert.



Über das Kontextmenü der Tabelle kann eine Kopie der Daten erfolgen. Hier wird allerdings kein Importbereich erzeugt, sondern eine Kopie. Die Eigenschaften der Datenfelder werden nicht ausgelesen, sondern von Calc analysiert. Außerdem werden die Feldbezeichnungen als Tabellenköpfe vorformatiert.



	A	B	C	D	E
1	**Import**			**Kopie**	
2	Vorname	Hausnummer		Vorname	Hausnummer
3	Rob	137		Rob	137
4	Mirina	27 b		Mirina	27 b
5	Heinz	159 b		Heinz	159 b
6	Moni	37		Moni	37
7	Kerstin	45		Kerstin	45

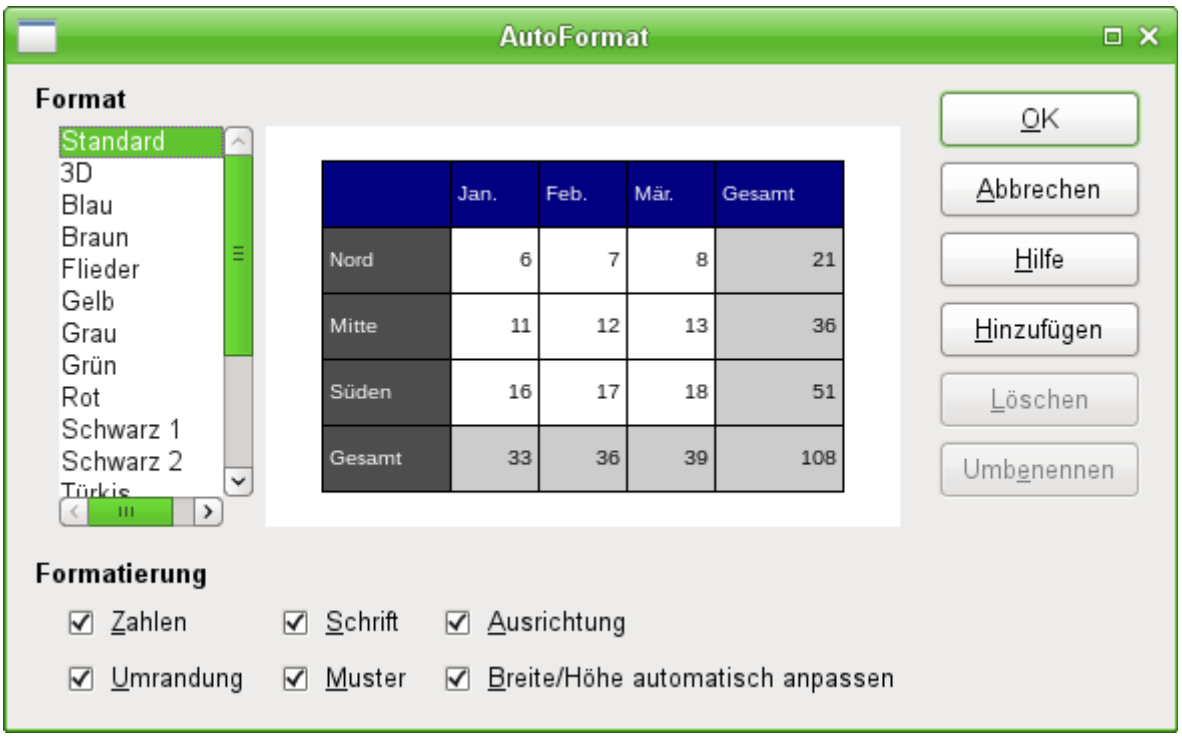
Die Unterschiede zeigen sich besonders bei Feldern, die in der Datenbank als Text formatiert sind. Beim Import macht Calc daraus Textfelder und setzt Zahlen, die es sonst auch als Zahlen interpretieren würde, ein Hochkomma **('137)** voran. Mit diesen Zahlen kann also nicht direkt weiter gerechnet werden. Dies wäre aber auch in Base nicht möglich, da es sich ja tatsächlich um Text handelt.

Bei einem eventuellen Export wird das Hochkomma allerdings wieder entfernt, so dass die Daten letztlich in der gleichen Art bestehen bleiben.

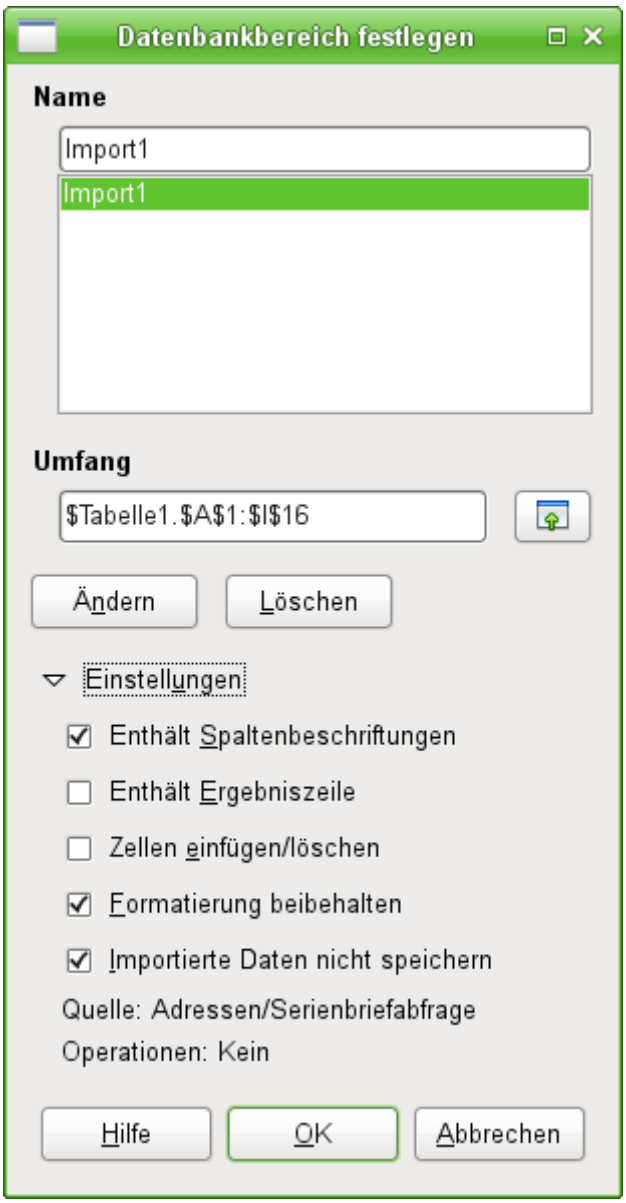
Hinweis

Nur der Import zeigt Daten, die tatsächlich in der Datenbank abgespeichert sind. Wird in einem Feld der Datenbank eine Dezimalzahl über die Sprachauswahl mit dem Dezimaltrenner «.» formatiert, so wird beim *Import* weiter die Dezimalzahl erkannt. Bei der *Kopie* hingegen wird aus der Dezimalzahl Text. Wird eine Dezimalzahl in Base mit 2 Nachkommastellen angezeigt, obwohl das Feld in der Datenbank tatsächlich 3 Nachkommastellen enthält, so werden bei der *Kopie* nur die sichtbaren 2 Nachkommastellen kopiert.
Wird aus einer Base-Datei direkt per *Drag- and Drop* eine Tabelle in Calc gezogen, so ist dies ein *Import*. Kopieren über die *Zwischenablage* erzeugt eine *Kopie*.

Importierte Daten können natürlich im Nachhinein formatiert werden. Hierzu wird der importierte Bereich komplett markiert und dann über **Format → AutoFormat Vorlagen** weiter formatiert:



Diese Formatierung bleibt erhalten, wenn unter **Daten → Bereich festlegen** die folgenden Angaben gemacht werden:



Die **Einstellungen** müssen aufgeklappt werden. Dann sollte **Formatierung beibehalten** ausgewählt werden. Zusätzlich ist noch sinnvoll, **Importierte Daten nicht speichern** zu wählen, wenn eine automatische Abfrage der Daten bei jedem Programmstart erfolgen soll. Dabei werden im übrigen auch Daten aufgenommen, die neu hinzugefügt werden. In sofern ist der Eintrag in **Umfang** nur der Startbereich. Es erfolgt automatisch eine Erweiterung des Bereiches nach unten.

Tipp

Der Import von Daten nach Calc überschreibt den vorherigen Inhalt – und auch die eventuell bereits erledigten Formatierungen, die anschließend direkt für einzelne Felder erfolgt sind. Werden beständig in die gleiche Tabelle Daten exportiert, so empfiehlt es sich, ein separates Tabellenblatt für den Datenimport zu nutzen. Die Daten werden dann über den Bezug **`Tabellenname.Feldbezeichnung`** in einem anderen Tabellenblatt ausgelesen. Die Felder in diesem Tabellenblatt können beliebig vorformatiert werden, ohne dass das Format überschrieben wird.

Tipp

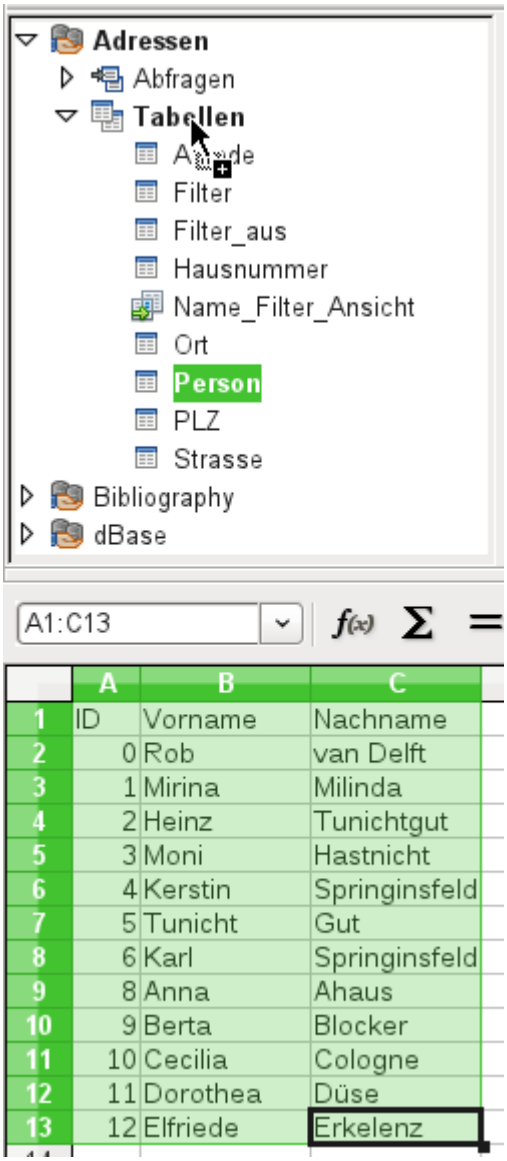
Daten können auch direkt aus der Datenbank über die Zwischenablage oder über Ziehen und Ablegen mit der Maus kopiert werden. Wird eine Tabelle oder Abfrage in ein Calc-Blatt gezogen, so wird der gesamte Inhalt eingefügt. Wird die Tabelle oder Abfrage geöffnet und ein oder mehrere Datensätze markiert, so werden nur diese Datensätze zusammen mit den Feldnamen beim Ziehen kopiert.

Vorsicht

Kopien über die Zwischenablage werden in Calc so übernommen, wie sie in der Datenquelle sichtbar sind. Tabellenköpfe werden als Feldbezeichner hervorgehoben, Zahlen mit der Anzahl an Nachkommastellen übernommen, die in Base **angezeigt** werden. Kopien über Drag-and-Drop transportieren den tatsächlichen Inhalt der Datenquelle, also auch z.B. Nachkommastellen von Zahlen, die zur Zeit in der Datenquelle nicht zu sehen, wohl aber vorhanden sind. (*Bug 112023*)

Daten aus Calc in eine Datenbank exportieren

Die Daten werden in dem Tabellenblatt von Calc markiert. Die linke Maustaste wird gedrückt und auf den Tabellenbereich der gewünschten Datenbank im Datenbankbrowser gezogen.



	A	B	C
1	ID	Vorname	Nachname
2	0	Rob	van Delft
3	1	Mirina	Milinda
4	2	Heinz	Tunichtgut
5	3	Moni	Hastnicht
6	4	Kerstin	Springinsfeld
7	5	Tunicht	Gut
8	6	Karl	Springinsfeld
9	8	Anna	Ahaus
10	9	Berta	Blocker
11	10	Cecilia	Cologne
12	11	Dorothea	Düse
13	12	Elfriede	Erkelenz

Der Cursor verändert sein Symbol und deutet an, dass etwas hinzugefügt werden kann.

Es öffnet sich das erste Fenster des Importassistenten. Die weiteren Schritte mit dem Assistenten sind im Kapitel «Tabellen» im Abschnitt *Import von Daten aus anderen Datenquellen* beschrieben.

Hinweis

Der Transport der Daten aus einer Calc-Tabelle in eine Datenbank läuft nicht in der gleichen Weise ab wie der Transport von Daten aus einer Datenbank zu einer anderen Datenbank.
Der oben beschriebene Weg führt z.B. dazu, dass *Zeilenumbrüche* in Feldern der Tabelle nicht in die Tabelle der Datenbank übernommen werden (*Bug 117436*). Wird stattdessen die *Calc-Datei als Datenquelle für eine Datenbank* genommen, so kann von dieser Datenbankdatei aus zu einer internen HSQLDB-Datenbankdatei der Tabelleninhalt mit Zeilenumbruch kopiert werden.

Daten von einer Datenbank zu einer anderen konvertieren

Im Explorer des Datenquellenbrowsers können Tabellen von einer Datenbank zur anderen kopiert werden, indem die Quelltabelle mit der linken Maustaste markiert wird, die Taste dann gedrückt gehalten wird und über dem Tabellencontainer der Zieldatenbank losgelassen wird. Es erscheint dann der Dialog zum Kopieren von Tabellen.

Auf diese Weise können beispielsweise Datenbanken, die sonst nur gelesen werden können (Datenquelle z.B. ein Adressbuch aus einem Mailprogramm oder eine Tabellenkalkulationstabelle), als Grundlage für eine Datenbank genutzt werden, die anschließend auch schreibend auf die Daten zugreift. Auch können beim Wechsel eines Datenbankprogramms (z.B. von PostgreSQL zu MySQL) die Daten direkt kopiert werden.

Hinweis

Beim Kopieren von Daten einer Datenbank in eine andere müssen die Felder der aufnehmenden Datenbank groß genug sein, um die Daten der abgebenden Datenbank aufzunehmen. Wurden z. B. Bilder in einer HSQLDB gespeichert (Datentyp **LONGVARBINARY**, 2^{31} – 1 Byte), so passen diese Bilder nicht unbedingt in ein Feld des Type **BLOB** von MARIADB/MYSQL (2^{16} – 1 Byte). Zu große Bilder in der HSQLDB führen dann zu einem Abbruch des Imports in MARIADB/MYSQL.

Wird gewünscht, dass die neue Datenbank andere Relationen aufweisen soll als die alte Datenbank, so kann dies durch entsprechende Abfragen in der Datenbank realisiert werden. Wer darin nicht so firm ist, kann stattdessen Calc nutzen. Hier werden die Daten in ein Tabellenblatt gezogen und anschließend mit Hilfe von Calc für den Import in die Zieldatenbank vorbereitet.

Für einen möglichst sauberen Import in eine neue Datenbank sollten die Tabellen der neuen Datenbank vorher erstellt werden. So können Formatierungsprobleme und Probleme bei der Erstellung von Primärschlüsseln rechtzeitig erkannt werden.

Daten über die Zwischenablage in eine Tabelle einfügen

Sind Daten in Tabellenform vorhanden, so können diese über die Zwischenablage und den Assistenten in Base eingefügt werden.

Wird in Base mit einem rechten Mausklick auf die Zieltabelle der Einfügevorgang begonnen, so erscheinen in dem **Kontextmenü** der Maus unter **Kopieren** die Befehle **Einfügen** und **Inhalte einfügen …**. Wird hier **Einfügen** gewählt, so ist beim Importassistenten die entsprechende Tabelle sowie das Anhängen von Daten bereits vorgewählt. **Inhalte einfügen …** schaltet hier lediglich eine Abfrage für einen Importfilter vor. Hier steht 'HTML' und 'RTF' zur Verfügung.

Wird stattdessen nur in den Tabellencontainer mit der rechten Maustaste geklickt, so erscheint der Importassistent mit der entsprechenden Wahl einer neuen Tabelle.

Datenimport aus PDF-Formularen

Wer Daten aus verschiedenen Quellen von außen her importieren will, sollte am besten eine Formularstruktur wählen, die bei der Eingabe der Daten nicht weiter verändert werden kann. Mit Hilfe des Writers lassen sich solche PDF-Formulare erstellen, im Internet verteilen und entsprechend z.B. als E-Mail-Anhang ausgefüllt zurückschicken. Fehlt schließlich nur noch der möglichst einfache Import in die Base-Datenbank. Das Beispiel[19] soll so eine Importmöglichkeit aufzeigen.

19 Die Datenbank «Beispiel_PDFFormular_Import.odb» zu diesem Bericht ist den Beispieldatenbanken für dieses Handbuch beigefügt.

Erstellen eines PDF-Formulars

Ein PDF-Formular wird als externes Formular ohne Datenbankanbindung erstellt. Über **Formular → Entwurfsmodus** können die notwendigen Elemente für das Formular eingefügt werden.

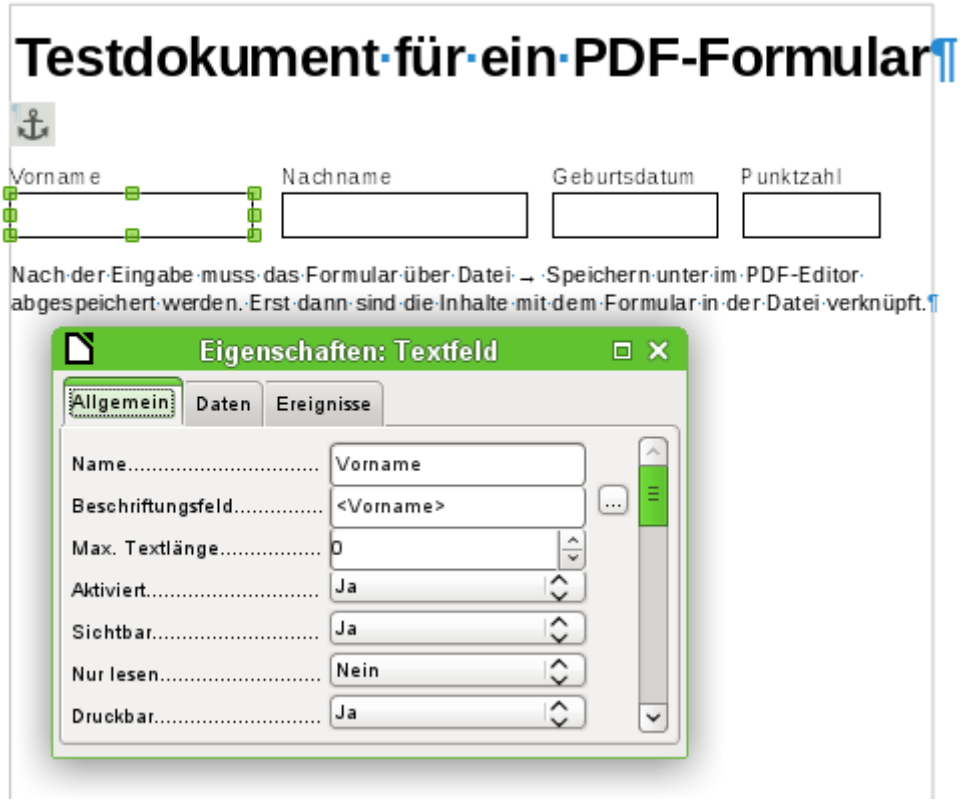
Leider werden im PDF-Formular keine Unterschiede zwischen Zahlenfeldern, Datumsfeldern und Textfeldern gemacht. Für das beigefügte Beispiel reicht es also völlig aus, bei jedem Eingabefeld ein Textfeld aufzuziehen. Weitere Feldformatierungen des Writer-Formulars gehen bei dem Export in ein PDF-Formular zwangsläufig verloren.

Grundsätzlich ermöglichen PDF-Formulare nur die folgenden Formularfelder:

- Schaltfläche
- Textfeld
- Markierfeld
- Optionsfeld
- Kombinationsfeld und
- Listenfeld

Hinweis

Die Eingabemöglichkeit in Textfeldern kann *in Abhängigkeit von dem PDF-Eingabeprogramm* unterschiedlich sein. Wird über **Formular → Weitere Felder → Numerisches Feld** ein numerisches Feld eingefügt, so richten sich viele PDF-Eingabeprogramme über den erzeugten JavaScript-Code nach den dortigen Einstellungen für die Dezimalstellen. Auch wird nur ein Punkt als Dezimaltrenner akzeptiert.
Beim **Datumsfeld** werden leider nur Codes erzeugt, die nicht dem deutschsprachigen Datum entsprechen. Dort wird dann eine Datumseingabe wie 14/07/23 als korrektes Datum erkannt. Auch zeigen manche Betrachter ein aufklappbares Feld mit Datumsauswahl an.



Das Testformular enthält insgesamt 4 Textfelder. In **Eigenschaften: Textfeld → Allgemein → Name** soll für den folgenden Importmodus die jeweilige Bezeichnung gewählt werden, die auch in der Tabelle der Datenbank als Feldname vorgesehen ist. Dadurch kann einwandfrei Feldname und Feldinhalt zugeordnet werden.

Hilfetexte werden zwar beim Auslesen der Daten mit angezeigt, erscheinen allerdings wohl nicht in jedem *.pdf-Viewer.

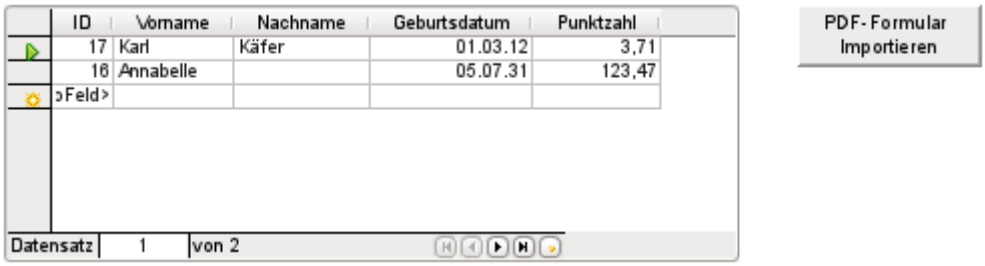
Hinweis

Der Acrobat Reader hat mit der voreingestellten Standard-Schriftart der Felder Probleme. Hier sollte eine weit verbreitete Schriftart wie Arial gewählt werden. Ist die in den Eingabefeldern definierte Schriftart nicht auf dem System vorhanden, so weigert sich der Acrobat Reader, eine Eingabe in den Feldern zu ermöglichen.
Der Fehler liegt auch darin begründet, dass LibreOffice die Schriftarten, die in den Formularfeldern benutzt werden, nicht mit exportiert.

Damit das Formular die Daten anschließend auch enthält, ist nach der Dateneingabe über **Datei → Speichern unter** im PDF-Viewer eine Speicherung der Datei vorzunehmen. Dieser Befehl kann von Viewer zu Viewer unterschiedlich sein. Ohne diese Maßnahme zeigt der Viewer die Daten nach dem Öffnen des Formular auf dem eigenen Rechner zwar an, liest sie aber wohl aus den temporären Dateien des Viewers, nicht aus der *.pdf-Datei direkt. Das Formular bleibt dann beim Transport von einem Rechner auf einen anderen leer.

Auslesen der Daten aus dem PDF-Formular

Das Formular der Base-Datenbank sieht recht einfach aus. Es ist mit der Tabelle verbunden und zeigt die gerade eingelesenen Daten an. Dabei sind die jüngsten Einträge in dem Tabellenkontrollfeld oben zu sehen.



Das Makro zum Einlesen der Daten wird unter **Eigenschaften : Schaltfläche → Ereignisse → Aktion ausführen** eingetragen.

Zum Auslesen der Daten wird hier auf das OpenSource-Programm «pdftk» zurückgegriffen. Das Programm ist auf jeden Fall frei für Linux und Windows erhältlich. Linux-Distributionen haben meist bereits ein Paket in den Repositories. Windows-User finden das Programm hier: *https://www.pdflabs.com/tools/pdftk-the-pdf-toolkit/* .

Unter Linux geht das Auslesen der Daten mit dem folgenden Befehl:

```
001 for i in *.pdf ; do pdftk "$i" dump_data_fields_utf8 >> auswertung.txt ; done
```

Der Befehl liest aus der im gleichen Verzeichnis liegenden *.pdf-Datei die Daten in eine Textdatei mit der Bezeichnung «auswertung.txt».

Die mit «pdftk» ausgelesenen Daten werden in eine Textdatei geschrieben, die wie folgt aussieht:

```
---
FieldType: Text
FieldName: Vorname
FieldFlags: 0
FieldValue: Karl
FieldJustification: Left
---
FieldType: Text
FieldName: Nachname
FieldFlags: 0
FieldValue: Käfer
FieldJustification: Left
---
FieldType: Text
FieldName: Geburtsdatum
FieldNameAlt: Datum mit mindestens zweistelliger Jahreszahl
FieldFlags: 0
FieldValue: 1.3.12
FieldJustification: Left
---
FieldType: Text
FieldName: Punktzahl
FieldNameAlt: Dezimalzahl, 2 Nachkommastellen
FieldFlags: 0
FieldValue: 3,71
FieldJustification: Left

```

Für jedes Feld sind in der Textdatei 5 Zeilen enthalten. Für die Auswertung im Makro sind die Zeilen «FieldName» (Feldname auch in der Zieltabelle), «FieldValue» (Inhalt des Feldes nach Abspeicherung der *.pdf-Datei) sowie «FieldJustification» (letzte Zeile, die einen Feldeintrag beendet) von Bedeutung.

Der gesamte Import wird über Makros geregelt. Hier muss das PDF-Formular im gleichen Pfad wie die Datenbank abgelegt werden. Die Daten werden in die Textdatei ausgelesen und aus dieser Textdatei wieder eingelesen. Dies geschieht so oft, wie *.pdf-Dateien mit Formulardaten in dem Verzeichnis liegen. Alte Dateien sollten also tunlichst aus dem Verzeichnis entfernt werden, denn die Funktion überprüft nicht auf Duplikate.

```
001 SUB PDF_Form_Import(oEvent AS OBJECT)
002     DIM inNumber AS INTEGER
003     DIM stRow AS STRING
004     DIM i AS INTEGER
005     DIM k AS INTEGER
006     DIM oDatasource AS OBJECT
007     DIM oConnection AS OBJECT
008     DIM oSQL_Command AS OBJECT
009     DIM oResult AS OBJECT
010     DIM stSql AS STRING
011     DIM oDB AS OBJECT
012     DIM oFileAccess AS OBJECT
013     DIM inFields AS INTEGER
014     DIM stFieldName AS STRING
015     DIM stFieldValue AS STRING
016     DIM stFieldType AS STRING
017     DIM stDir AS STRING
018     DIM stDir2 AS STRING
019     DIM stPDFForm AS STRING
020     DIM stFile AS STRING
021     DIM stTable AS STRING
022     DIM inNull AS INTEGER
023     DIM aFiles()
024     DIM aNull()
025     DIM stCommand AS STRING
026     DIM stParameter AS STRING
027     DIM oShell AS OBJECT
```

Nach der Deklaration der Variablen wird die Anzahl der Felder angegeben, die das PDF-Formular enthält. Die Zählung beginnt hier mit 0. Beim Wert '3' sind also insgesamt 4 Felder in dem Formular vorhanden. Mit Hilfe dieser Zählung wird ermittelt, wann alle Daten für einen Daten-

satz ausgelesen wurden, so dass die Daten in die Tabelle der Datenbank übergeben werden können.

```
028     inFields = 3
029     stTable = "Name"
030     oDatasource = ThisComponent.Parent.CurrentController
031     If NOT (oDatasource.isConnected()) THEN
032        oDatasource.connect()
033     END IF
034     oConnection = oDatasource.ActiveConnection()
035     oSQL_Command = oConnection.createStatement()
```

Die Datenbankverbindung wurde ermittelt. Anschließend wird der Pfad zu der bestehenden Datenbankdatei im Dateisystem ausgelesen. Mit Hilfe dieses Pfades wird in dem Array **aFiles** der Inhalt des Verzeichnisses ermittelt. Für jede der Dateien des Verzeichnisses wird in einer Schleife nachgesehen, ob die Dateiendung «.pdf» lautet. Groß- und Kleinschreibung sind hier egal, da das Ergebnis der Suche über **LCase** direkt in Kleinschreibung umgesetzt wird.

```
036     oDB = ThisComponent.Parent
037     stDir = Left(oDB.Location,Len(oDB.Location)-Len(ConvertToURL(oDB.Title))+8)
038     oFileAccess = createUnoService("com.sun.star.ucb.SimpleFileAccess")
039     aFiles = oFileAccess.getFolderContents(stDir, False)
040     FOR k = 0 TO uBound(aFiles())
041        IF LCase(Right(aFiles(k),4)) = ".pdf" THEN
042           stDir2 = ConvertFromUrl(stDir)
043           stPDFForm = ConvertFromUrl(aFiles(k))
```

Um das Kommando für das Auslesen betriebssystemspezifisch adressieren zu können, muss der ursprünglich mit **file:///** beginnende Pfadname dem jeweiligen System angepasst werden. Das Startkommando für das Programm «pdftk» ist abhängig vom Betriebssystem entweder mit dem Zusatz «.exe», vielleicht sogar mit dem gesamten Pfad zum Programm wie z.B. «C:\Program Files (x86)\pdftk\pdftk.exe», oder eben ohne Zusatz zu versehen. Über **GetGuiType** wird der Systemtyp ermittelt, wobei '1' für Windows, '3' für MAC und '4' für Unix/Linux steht. Im folgenden wird nur zwischen Windows und allen anderen Systemen unterschieden.

Im Anschluss daran gibt der Befehl **Shell()** die Startinformationen für **pdftk** an die Konsole weiter. Über die Angabe von **True** wird dabei geregelt, dass LibreOffice wartet, bis der Shell-Prozess beendet wird.

```
044           IF GetGuiType = 1 THEN '()
045              stCommand = "pdftk.exe"
046           ELSE
047              stCommand = "pdftk"
048           END IF
049           stParameter = stPDFForm & " dump_data_fields_utf8 output "
050              & stDir2 & "PDF_Form_Data.txt"
051           Shell(stCommand,0,stParameter,True)
052           stFile = stDir & "PDF_Form_Data.txt"
053           i = -1
054           inNumber = FreeFile
```

Mit der Funktion **FreeFile** wird ermittelt, welchen freien Datenkanal das Betriebssystem als nächstes zur Verfügung stellen wird. Dieser Datenkanal wird als eine Integerziffer ausgelesen und direkt zur Öffnung der gerade erstellten PDF-Datendatei genutzt. Die Datei wird über INPUT ausgelesen. Dies ist aus Sicht von LibreOffice gesehen. Daten von außerhalb werden in das Programm LibreOffice eingelesen.

```
055           OPEN stFile FOR INPUT AS inNumber
056              DO WHILE NOT Eof(inNumber)
057                 LINE INPUT #inNumber, stRow
```

Zeile für Zeile wird die PDF-Datendatei jetzt ausgelesen. Taucht der Begriff «FieldName: » auf, so entspricht der folgende Inhalt der Zeile der Bezeichnung des Feldes im PDF-Formular und durch die Definition des Formulars auch der Bezeichnung des Feldes in der Datenbanktabelle, in die der Inhalt geschrieben werden soll.

Alle Feldnamen werden direkt für den späteren SQL-Befehl zusammengefasst. Das bedeutet, dass die Feldnamen mit doppelten Anführungszeichen versehen und durch Kommas voneinander getrennt werden.

Zu jedem Feldnamen wird außerdem durch eine Abfrage ermittelt, um welchen Feldtyp der Tabelle es sich hierbei handelt (HSQLDB, SQL-Code für Firebird siehe Anhang). Datumswerte und Dezimalzahlen müssen anders weiter gegeben werden als Texte.

```
058                IF instr(stRow, "FieldName: ") THEN
059                   IF stFieldName = "" THEN
060                      stFieldName = """"+mid(stRow,12)+""""
061                   ELSE
062                      stFieldName = stFieldName & ","" + mid(stRow,12)+""""
063                   END IF
064                   stSql = "SELECT TYPE_NAME FROM INFORMATION_SCHEMA.SYSTEM_COLUMNS
                         WHERE TABLE_NAME = '" + stTable + "' AND
                         COLUMN_NAME = '" + mid(stRow,12) + "'"
065                   oResult = oSQL_Command.executeQuery(stSql)
066                   WHILE oResult.next
067                      stFieldType = oResult.getString(1)
068                   WEND
069                END IF
```

Hinweis

Der SQL-Code für FIREBIRD ist hier leider wesentlich komplizierter, da FIREBIRD in den Systemtabellen die Feldtypen nicht benennt, sondern nur numerisch aufführt:

```
064  stSql = "SELECT TRIM(CASE ""b"".RDB$FIELD_TYPE||'|'||
        COALESCE(""b"".RDB$FIELD_SUB_TYPE,0) "+ _
        "WHEN '7|0' THEN 'SMALLINT' "+ _
        "WHEN '8|0' THEN 'INTEGER' "+ _
        "WHEN '8|1' THEN 'NUMERIC' "+ _
        "WHEN '8|2' THEN 'DECIMAL' "+ _
        "WHEN '10|0' THEN 'FLOAT' "+ _
        "WHEN '12|0' THEN 'DATE' "+ _
        "WHEN '13|0' THEN 'TIME' "+ _
        "WHEN '14|0' THEN 'CHAR' "+ _
        "WHEN '16|0' THEN 'BIGINT' "+ _
        "WHEN '35|0' THEN 'TIMESTAMP' "+ _
        "WHEN '37|0' THEN 'VARCHAR' "+ _
        "WHEN '261|0' THEN 'BLOB' "+ _
        "WHEN '261|1' THEN 'BLOB Text' "+ _
        "WHEN '261|2' THEN 'BLOB BLR' "+ _
        "WHEN '261|3' THEN 'BLOB ACL' "+ _
        "END) AS ""SQL_Datentyp"" "+ _
        "FROM RDB$RELATION_FIELDS AS ""a"", RDB$FIELDS AS ""b"" "+ _
        "WHERE ""a"".RDB$FIELD_SOURCE = ""b"".RDB$FIELD_NAME "+ _
        "AND ""a"".RDB$RELATION_NAME = '" + stTable + "' "+ _
        "AND ""a"".RDB$FIELD_NAME = '" + mid(stRow,12) + "'"
```

Wie bei den Feldnamen wird auch bei den Werten verfahren. Sie dürfen allerdings nicht in doppelten Anführungszeichen weitergegeben werden, sondern müssen für den SQL-Code entsprechend vorbereitet werden. So muss Schrift in einfache Anführungszeichen gesetzt werden, Datumsangaben SQL-konform umgewandelt werden usw. Dies geschieht durch die extra ausgelagerte Funktion **SQL_Value**.

```
070                IF instr(stRow, "FieldValue: ") THEN
071                   IF stFieldValue = "" THEN
072                      stFieldValue = SQL_Value(mid(stRow,13), stFieldType)
073                   ELSE
074                      stFieldValue = stFieldValue & "," &
075                         SQL_Value(mid(stRow,13), stFieldType)
076                   END IF
077                END IF
```

Taucht der Begriff «FieldJustification:» auf, so ist das Ende einer Kombination von Feldbezeichnung und Feldwert erreicht. Der Zähler **i**, der anschließend mit der vorher angegebenen Anzahl der Felder **inFields** verglichen werden soll, wird um '1' heraufgesetzt.

Wenn schließlich **i** und **inFields** gleich ist, wird der SQL-Befehl zusammengestellt. Allerdings soll vermieden werden, dass leere Datensätze bei leeren Formularen entstehen. Deshalb wird vorher nachgesehen, ob die Werte aller Felder **NULL** sind. Ist dies der Fall, so wird kein SQL-Befehl ausgelöst. Ansonsten wird der Datensatz in die Tabelle "Name" eingefügt. Anschließend werden die Variablen wieder auf ihre Standardwerte zurückgesetzt und das nächste PDF-Formular kann ausgelesen werden.

```
078                IF instr(stRow, "FieldJustification:") THEN
079                   i = i + 1
080                END IF
081                IF i = inFields THEN
082                   aNull = Split(stFieldValue,",")
083                   FOR n = 0 TO Ubound(aNull())
084                      IF aNull(n) = "NULL" THEN inNull = inNull + 1
085                   NEXT
086                   IF inNull < inFields THEN
087                      stSql ="INSERT INTO """ + stTable + """(" + stFieldName + ")"
088                      stSql = stSql + "VALUES (" + stFieldValue + ")"
089                      oSQL_Command.executeUpdate(stSql)
090                   END IF
091                   stFieldName = ""
092                   stFieldValue = ""
093                   stFieldType = ""
094                   i = -1
095                   inNull = 0
096                END IF
097             LOOP
098          CLOSE inNumber
```

Zum Schluss der Prozedur bleibt schließlich eine Datei «PDF_Form_Data.txt» übrig. Diese Datei wird gelöscht. Anschließend wird das Formular neu geladen, damit die eingelesenen Daten direkt angezeigt werden können.

```
099          Kill(stFile)
100       END IF
101    NEXT
102    oEvent.Source.Model.Parent.reload()
103 END SUB
```

Enthält ein Text ein einfaches Anführungszeichen «'», so wird es beim Einfügen in SQL als Ende des Textes angesehen. Der SQL-Code für den Insert-Befehl scheitert, wenn anschließend noch Text folgt, der nicht in einfachen Anführungszeichen steht. Um dies zu vermeiden muss jedes einfache Anführungszeichen innerhalb des Textes mit einem weiteren einfachen Anführungszeichen maskiert werde. Das erledigt die Funktion **String_to_SQL**.

```
001 FUNCTION String_to_SQL(st AS STRING) AS STRING
002    IF InStr(st,"'") THEN
003       st = Join(Split(st,"'"),"''")
004    END IF
005    String_to_SQL = st
006 END FUNCTION
```

Die Datumsangabe in einem PDF-Formular wird als Text ausgelesen. Sie kann nicht vorher auf korrekte Eingabe kontrolliert werden.

In der deutschen Schreibweise werden Tag, Monat und Jahr voneinander häufig durch einen Punkt getrennt. Dabei kann die Tagesangabe und die Monatsangabe einstellig oder zweistellig, die Jahresangabe zweistellig oder vierstellig sein.

Für den SQL-Code muss die Datumsangabe beginnend mit der vierstelligen Jahreszahl in dem folgenden international üblichen erweiterten ISO-Format Format geschrieben werden: YYYY-MM-DD. Es muss also gegebenenfalls eine Umwandlung des eingegebenen Datums erfolgen.

Die Datumsangabe wird in den Tagesanteil, den Monatsanteil und den Jahresanteil aufgesplittet. Die Tagesangabe und die Monatsangabe wird mit einer führenden «0» versehen und anschließend von rechts aus auf zwei Zeichen begrenzt. Damit ist die Angabe auf jeden Fall zweistellig.

Ist die Jahresangabe bereits vierstellig (größer als 1000), so wird der Wert nicht geändert. Ansonsten wird bei einer Jahresangabe größer als 30 davon ausgegangen, dass es sich um eine Angabe handelt, die im letzten Jahrhundert liegt und um 1900 erhöht werden muss. Alle anderen Jahresangaben werden als Angaben für das aktuelle Jahrhundert angesehen.

```
001 FUNCTION Date_to_SQLDate(st AS STRING) AS STRING
002    DIM stDay AS STRING
003    DIM stMonth AS STRING
004    DIM stDate AS STRING
005    DIM inYear AS INTEGER
006    stDay = Right("0" & Day(CDate(st)), 2)
007    stMonth = Right("0" & Month(CDate(st)), 2)
008    inYear = Year(CDate(st))
009    IF inYear = 0 THEN
010       inYear = Year(Now())
011    END IF
012    IF inYear > 1000 THEN
013    ELSEIF inYear > 30 THEN
014       inYear = 1900 + inYear
015    ELSE
016       inYear = 2000 + inYear
017    END IF
018    stDate = inYear & "-" & stMonth & "-" & stDay
019    Date_to_SQLDate = stDate
020 END FUNCTION
```

Die Funktion **SQL_Value** fasst die vorhergehenden Funktionen sowie die **NULL**-Werte zusammen und gibt entsprechend vorformatierte Werte für die Eingabe in die Datenbank an die aufrufende Prozedur zurück.

Für leere Felder wird NULL zurück gegeben. Damit bleibt das Feld auch in der Tabelle anschließend leer.

```
001 FUNCTION SQL_Value(st AS STRING, stType AS STRING) AS STRING
002    DIM stValue AS STRING
003    IF st = "" THEN
004       SQL_Value = "NULL"
```

Handelt es sich bei dem Feldtyp um ein Datumsfeld und ist der Inhalt als Datum zu erkennen, so soll der Inhalt in ein SQL-Datumsformat umgewandelt werden. Ist der Inhalt nicht als Datum erkennbar, so soll das Feld leer bleiben.

```
005    ELSEIF stType = "DATE" THEN
006       IF isDate(st) THEN
007          SQL_Value = "'" & Date_to_SQLDate(st) & "'"
008       ELSE
009          SQL_Value = "NULL"
010       END IF
```

Handelt es sich bei dem Feldtyp um ein Dezimalfeld, so kann es Nachkommastellen geben. Der Dezimaltrenner in Basic und auch in SQL ist allerdings ein Punkt. Entsprechend müssen Zahlen umgewandelt werden, sofern sie ein Komma enthalten. Das Feld kann nur Zahlen aufnehmen, so dass andere Zeichen wie z.B. Maßeinheiten entfernt werden müssen. Dies geschieht mit der Funktion **Val()**. Leider gibt diese Funktion aber den Wert mit dem Dezimaltrenner der eingestellten Sprache zurück, so dass hier anschließend noch das Komma erneut durch einen Punkt ersetzt werden muss.

```
011    ELSEIF stType = "DECIMAL" THEN
012       stValue = Str(Val(Join(Split(st,","),".")))
013       SQL_Value = Join(Split(stValue,","),".")
```

Alle anderen Inhalte werden als Text behandelt. Einfache Anführungszeichen werden mit einem weiteren einfachen Anführungszeichen maskiert und der Gesamtbegriff wiederum in einfachen Anführungszeichen gefasst weitergegeben.

```
014     ELSE
015        SQL_Value = "'" & String_to_SQL(st) & "'"
016     END IF
017  END FUNCTION
```

Zu weiteren Details beim Aufbau von Makros sollte das separate Kapitel dieses Handbuches zu Rate gezogen werden. Dieses Beispiel sollte lediglich aufzeigen, dass es auch möglich ist, Daten aus PDF-Formularen nach Base zu übertragen, ohne die Werte über die Zwischenablage aus jedem Feld in die Datenbank kopieren zu müssen. Der Aufbau der obigen Prozedur ist dabei recht allgemein gehalten und müsste sicher den jeweiligen Bedürfnissen angepasst werden.

Datenbank-Aufgaben

Allgemeines zu Datenbankaufgaben

Hier werden einige Lösungen für Problemstellungen vorgestellt, die im Laufe der Zeit viele Datenbankuser beschäftigen werden. Anfragen dazu kamen vor allem aus den Mailinglisten, insbesondere *users@de.libreoffice.org*,
sowie aus den Foren *http://de.openoffice.info/viewforum.php?f=8*
und *http://www.libreoffice-forum.de/viewforum.php?f=10* .

Datenfilterung

Die Datenfilterung mittels der GUI ist bereits bei der Dateneingabe in Tabellen beschrieben. Hier soll eine Lösung aufgezeigt werden, die bei vielen Nutzern gefragt ist: Mittels Listenfeldern werden Inhalte von Tabellenfeldern ausgesucht, die dann im darunterliegenden Formularteil herausgefiltert erscheinen und bearbeitet werden können.

Grundlage für diese Filterung ist neben einer bearbeitbaren Abfrage (siehe das Kapitel *Eingabemöglichkeit in Abfragen*) eine weitere Tabelle, in der die zu filternden Daten abgespeichert werden. Die Abfrage zeigt aus der ihr zugrundeliegenden Tabelle nur die Datensätze an, die dem eingegebenen Filterwert entsprechen. Ist kein Filterwert angegeben, so zeigt die Abfrage alle Datensätze an.

Für das folgenden Beispiel wird von einer Tabelle **"Medien"** ausgegangen, die unter anderem die folgenden Felder beinhaltet: **"ID"** (Primärschlüssel), **"Titel"**, **"Kategorie"**.

Zuerst wird eine Tabelle **"Filter"** benötigt. Diese Tabelle erhält einen Primärschlüssel und 2 Filterfelder (das kann natürlich beliebig erweitert werden): **"ID"** (Primärschlüssel), **"Filter_1"**, **"Filter_2"**. Da die Felder der Tabelle **"Medien"**, die gefiltert werden sollen, vom Typ **VARCHAR** sind, haben auch die Felder **"Filter_1"** und **"Filter_2"** diesen Typ. **"ID"** kann ein **Ja/Nein**-Feld sein. Die Tabelle **"Filter"** wird sowieso nur einen Datensatz abspeichern.

Feldname	***Feldtyp***
ID	Ja/Nein [BOOLEAN]
Filter_1	Text [VARCHAR]
Filter_2	Text [VARCHAR]

Tipp

Wird keine Einbenutzer-Datenbank wie Base mit der internen HSQLDB genutzt, so würde so eine Filtertabelle erst einmal die Filterung auch bei anderen Nutzern erzeugen. Hier könnte einfach der Nutzername direkt als Primärschlüssel der Filtertabelle genutzt werden. Dann wäre das Feld ID kein Ja/Nein-Feld, sondern ein VARCHAR-Feld.
Über

```
001 SELECT CURRENT_USER From "Medien"
```

würde in diesem Falle der aktuelle Nutzer abgefragt. Bei der internen Datenbank ist das immer 'SA'. Entsprechend darf bei der Filterung dann allerdings nur der Datensatz ausgelesen werden, bei dem

```
001 "ID" = CURRENT_USER
```

ist. Unterscheiden sich die Nutzernamen nicht, so wäre mit **CURRENT_CONNECTION** der Integer-Wert der aktuellen Verbindung (FIREBIRD) nutzbar.

Natürlich kann auch nach Feldern gefiltert werden, die in der Tabelle "Medien" nur über einen Fremdschlüssel vertreten sind. Dann müssen die entsprechenden Felder in der Tabelle "Filter" natürlich dem Typ des Fremdschlüssels entsprechen, in der Regel also «Integer» sein.

Folgende Abfrageform bleibt sicher editierbar:

```
001 SELECT * FROM "Medien"
```

Alle Datensätze der Tabelle **"Medien"** werden angezeigt, auch der Primärschlüssel.

```
001 SELECT * FROM "Medien"
002 WHERE "Titel" = COALESCE(
003    ( SELECT "Filter_1" FROM "Filter" WHERE "ID" = TRUE), "Titel" )
```

Ist das Feld **"Filter_1"** nicht **NULL**, so werden die Datensätze angezeigt, bei denen der **"Titel"** gleich dem **"Filter_1"** ist. Wenn das Feld **"Filter_1"** **NULL** ist wird stattdessen der Wert des Feldes **"Titel"** genommen. Da **"Titel"** gleich **"Titel"** ist, werden so alle Datensätze angezeigt – sollte angenommen werden, trifft aber nicht zu, wenn im Feld **"Titel"** irgendwo ein leeres Feld **'NULL'** enthalten ist. Das bedeutet, dass die Datensätze nie angezeigt werden, die keinen Titeleintrag haben. Hier muss in der Abfrage nachgebessert werden.

```
001 SELECT * ,
002    COALESCE( "Titel", '' ) AS "T"
003 FROM "Medien"
004 WHERE "T" = COALESCE(
005    ( SELECT "Filter_1" FROM "Filter"  WHERE "ID" = TRUE), "T" )
```

Diese Variante würde zum Ziel führen. Statt **"Titel"** direkt zu filtern, wird ein Feld gefiltert, das den Alias-Namen **"T"** erhält. Dieses Feld ist zwar weiter ohne Inhalt, aber eben nicht **NULL**. In der Bedingung wird nur auf dieses Feld **"T"** Bezug genommen. Alle Datensätze werden angezeigt, auch wenn **"Titel"** **NULL** sein sollte.

Leider spielt hier die GUI nicht mit. Der Befehl ist nur direkt über SQL absetzbar. Um ihn mit der GUI editierbar zu machen, ist weitere Handarbeit erforderlich:

```
001 SELECT "Medien".* ,
002    COALESCE( "Medien"."Titel", '' ) AS "T"
003 FROM "Medien"
004 WHERE "T" = COALESCE(
005    ( SELECT "Filter_1" FROM "Filter"  WHERE "ID" = TRUE), "T" )
```

Wenn jetzt der Tabellenbezug zu den Feldern hergestellt ist, ist die Abfrage auch in der GUI editierbar.

Zum Testen kann jetzt einfach ein Titel in **"Filter"."Filter_1"** eingegeben werden. Als **"Filter"."ID"** wird der Wert **'0'** gesetzt. Der Datensatz wird abgespeichert und die Filterung kann nachvollzogen werden. Wird **"Filter"."Filter_1"** wieder geleert, so macht die GUI daraus **NULL**. Ein erneuter Test ergibt, dass jetzt wieder alle Medien angezeigt werden. Bevor ein Formular erstellt und getestet wird, sollte auf jeden Fall ein Datensatz, aber wirklich nur einer, mit einem Primärschlüssel in der Tabelle **"Filter"** stehen. Nur ein Datensatz darf es sein, da Unterabfragen wie oben gezeigt nur einen Wert wiedergeben dürfen.

Die Abfrage wird jetzt erweitert, um auch ein 2. Feld zu filtern:

```
001 SELECT "Medien".* ,
002    COALESCE( "Medien"."Titel", '' ) AS "T",
003    COALESCE( "Medien"."Kategorie", '' ) AS "K"
004 FROM "Medien"
005 WHERE "T" = COALESCE(
006    ( SELECT "Filter_1" FROM "Filter" WHERE "ID" = TRUE), "T" )
007    AND "K" = COALESCE(
008    ( SELECT "Filter_2" FROM "Filter" WHERE "ID" = TRUE), "K" )
```

Damit ist die Erstellung der editierbaren Abfrage abgeschlossen. Jetzt wird noch die Grundlage für die beiden Listenfelder als Abfrage zusammengestellt:

```
001 SELECT DISTINCT "Titel", "Titel"
002 FROM "Medien" ORDER BY "Titel" ASC
```

Das Listenfeld soll sowohl die **"Titel"** anzeigen als auch die **"Titel"** an die dem Formular zugrundeliegende Tabelle **"Filter"** in das Feld **"Filter_1"** weitergeben. Dabei sollen keine doppelten Werte angezeigt werden (Anordnung «**DISTINCT**») . Und das Ganze soll natürlich richtig sortiert erscheinen. Dabei ist die Abfrage an die Standardeinstellung der Listenfelder

angepasst, die dem gebundenen Feld eine '1' zuweist. Wird stattdessen unter **Eigenschaften Listenfeld → Daten → Gebundenes Feld** eine '0' zugewiesen, so braucht nur einmal das Feld "Titel" abgefragt werden.

Eine entsprechende Abfrage wird dann auch für das Feld **"Kategorie"** erstellt, die ihre Daten in der Tabelle **"Filter"** in das Feld **"Filter_2"** schreiben soll.

Handelt es sich bei einem der Felder um ein Fremdschlüsselfeld, so ist die Abfrage entsprechend so anzupassen, dass der Fremdschlüssel an die zugrundeliegende Tabelle "Filter" weitergegeben wird.

Das Formular besteht aus zwei Teilformularen. Formular 1 ist das Formular, dem die Tabelle **"Filter"** zugrunde liegt. Formular 2 ist das Formular, dem die Abfrage zugrunde liegt. Formular 1 hat **keine Navigationsleiste** und den Zyklus «**Aktueller Datensatz**». Die Eigenschaft «**Daten hinzufügen**» ist außerdem auf «**Nein**» gestellt. Der erste und einzige Datensatz existiert ja bereits.

Formular 1 enthält 2 Listenfelder mit entsprechenden Überschriften. Listenfeld 1 soll Werte für **"Filter_1"** liefern und wird mit der Abfrage für das Feld **"Titel"** versorgt. Listenfeld 2 soll Werte für **"Filter_2"** weitergeben und beruht auf der Abfrage für das Feld **"Kategorie"**.

Formular 2 enthält ein Tabellenkontrollfeld, in dem alle Felder aus der Abfrage aufgelistet sein können – mit Ausnahme der Felder **"T"** und **"K"**. Mit den Feldern wäre der Betrieb auch möglich – sie würden aber wegen der doppelten Feldinhalte nur verwirren. Außerdem enthält das Formular 2 noch einen Button, der die Eigenschaft «**Formular aktualisieren**» hat. Zusätzlich kann noch eine Navigationsleiste eingebaut werden, damit nicht bei jedem Formularwechsel der Bildschirm aufflackert, weil die Navigationsleiste in einem Formular ein-, in dem anderen ausgestellt ist.

Wenn das Formular fertiggestellt ist, geht es zur Testphase. Wird ein Listenfeld geändert, so reicht die Betätigung des Buttons aus dem Formular 2 aus, um zuerst diesen Wert zu speichern und dann das Formular 2 zu aktualisieren. Das Formular 2 bezieht sich jetzt auf den Wert, den das Listenfeld angibt. Die Filterung kann über die Wahl des im Listenfeld enthaltenen leeren Feldes rückgängig gemacht werden.

Datensuche

Der Hauptunterschied zwischen der Suche von Daten und der Filterung von Daten liegt in der Abfragetechnik. Schließlich soll zu frei eingegebenen Begriffen ein Ergebnis geliefert werden, das diese Begriffe auch nur teilweise beinhaltet.

Suche mit LIKE

Die Tabelle für die Suchinhalte kann die gleiche sein, in die bereits die Filterwerte eingetragen werden. Die Tabelle **"Filter"** wird einfach ergänzt um ein Feld mit der Bezeichnung **"Suchbegriff"**. So kann gegebenenfalls auf die gleiche Tabelle zugegriffen werden und in Formularen gleichzeitig gefiltert und gesucht werden. **"Suchbegriff"** hat die Feldeigenschaft **VARCHAR**.

Das Formular wird wie bei der Filterung aufgebaut. Statt eines Listenfeldes muss für den Suchbegriff ein Texteingabefeld erstellt werden, zusätzlich vielleicht auch ein Beschriftungsfeld mit dem Titel «Suche». Das Feld für den Suchbegriff kann alleine in dem Formular stehen oder zusammen mit den Feldern für die Filterung, wenn eben beide Funktionen gewünscht sind.

Der Unterschied zwischen Filterung und Suche liegt in der Abfragetechnik. Während die Filterung bereits von einem Begriff ausgeht, den es in der zugrundeliegenden Tabelle gibt (schließlich baut das Listenfeld auf den Tabelleninhalten auf) geht die Suche von einer beliebigen Eingabe aus.

```
001 SELECT * FROM "Medien"
002 WHERE "Titel" = ( SELECT "Suchbegriff" FROM "Filter" WHERE "ID" = TRUE)
```

Diese Abfrage würde in der Regel ins Leere führen. Das hat mehrere Gründe:

- Selten weiß jemand bei der Eingabe des Suchbegriffs den kompletten Titel fehlerfrei auswendig. Damit würde der Titel nicht angezeigt. Um das Buch «Per Anhalter durch die Galaxis» zu finden müsste es ausreichen, in das Suchfeld 'Anhalter' einzugeben, vielleicht auch nur 'Anh'.
- Ist das Feld "Suchbegriff" leer, so würde überhaupt kein Datensatz angezeigt. Die Abfrage gäbe **NULL** zurück und **NULL** kann in einer Bedingung nur mittels **IS NULL** erscheinen.
- Selbst wenn dies ignoriert würde, so würde die Abfrage dazu führen, dass alle die Datensätze angezeigt würden, die keine Eingabe im Feld **"Titel"** haben.

Die letzten beiden Bedingungen könnten erfüllt werden, indem wie bei der Filterung vorgegangen würde:

```
001 SELECT * FROM "Medien"
002 WHERE "Titel" = COALESCE(
003    ( SELECT "Suchbegriff" FROM "Filter" WHERE "ID" = TRUE), "Titel" )
```

Mit den entsprechenden Verfeinerungen aus der Filterung (was ist mit Titeln, die **NULL** sind?) würde das zum entsprechenden Ergebnis führen. Nur würde die erste Bedingung nicht erfüllt. Die Suche lebt ja schließlich davon, dass nur Bruchstücke geliefert werden. Die Abfragetechnik der Wahl müsste daher über den Begriff «**LIKE**» gehen:

```
001 SELECT * FROM "Medien"
002 WHERE "Titel" LIKE
003    ( SELECT '%' || "Suchbegriff" || '%' FROM "Filter" WHERE "ID" = TRUE)
```

oder besser:

```
001 SELECT * FROM "Medien"
002 WHERE "Titel" LIKE COALESCE(
003    ( SELECT '%' || "Suchbegriff" || '%' FROM "Filter" WHERE "ID" = TRUE),
004    "Titel" )
```

LIKE, gekoppelt mit '**%**', bedeutet ja, dass alle Datensätze gezeigt werden, die an irgendeiner Stelle den gesuchten Begriff stehen haben. '**%**' steht als Joker für beliebig viele Zeichen vor und hinter dem Suchbegriff. Verschiedene Baustellen bleiben nach dieser Abfrageversion:

- Besonders beliebt ist ja, in Suchformularen alles klein zu schreiben. Wie bekomme ich mit 'anhalter' statt 'Anhalter' auch noch ein Ergebnis?
- Welche anderen Schreibgewohnheiten gibt es noch, die vielleicht zu berücksichtigen wären?
- Wie sieht es mit Feldern aus, die nicht als Textfelder formatiert sind? Lassen sich auch Datumsanzeigen oder Zahlen mit dem gleichen Feld suchen?
- Und was ist, wenn, wie bei dem Filter, ausgeschlossen werden muss, dass **NULL**-Werte in dem Feld verhindern, dass alle Datensätze angezeigt werden?

Die folgende Variante deckt ein paar mehr Möglichkeiten ab:

```
001 SELECT * FROM "Medien"
002 WHERE LOWER("Titel") LIKE COALESCE(
003    ( SELECT '%' || LOWER("Suchbegriff") || '%'
004       FROM "Filter" WHERE "ID" = TRUE),
005    LOWER("Titel") )
```

Die Bedingung ändert den Suchbegriff und den Feldinhalt auf Kleinschreibweise. Damit werden auch ganze Sätze vergleichbar.

```
001 SELECT * FROM "Medien"
002 WHERE LOWER("Titel") LIKE COALESCE(
003    ( SELECT '%' || LOWER("Suchbegriff") || '%'
004       FROM "Filter" WHERE "ID" = TRUE), LOWER("Titel") )
```

```
005     OR LOWER("Kategorie") LIKE ( SELECT '%' || LOWER("Suchbegriff") || '%'
006        FROM "Filter" WHERE "ID" = TRUE)
```

Die **COALESCE**-Funktion muss nur einmal vorkommen, da bei dem **"Suchbegriff" NULL** ja dann **LOWER("Titel") LIKE LOWER("Titel")** abgefragt wird. Und da der Titel ein Feld sein soll, das nicht **NULL** sein darf, werden so auf jeden Fall alle Datensätze angezeigt. Für entsprechend viele Felder wird dieser Code natürlich entsprechend lang. Schöner geht so etwas mittels Makro, das dann den Code in einer Schleife über alle Felder erstellt.

Aber funktioniert der Code auch mit Feldern, die keine Textfelder sind? Obwohl die Bedingung **LIKE** ja eigentlich auf Texte zugeschnitten ist, brauchen Zahlen, Datums- oder Zeitangaben keine Umwandlung, um damit zusammen zu arbeiten. Allerdings können hierbei die Textumwandlungen unterbleiben. Nur wird natürlich ein Zeitfeld auf eine Mischung aus Text und Zahlen nicht mit einer Fundstelle reagieren können – es sei denn die Abfrage wird ausgeweitet, so dass der eine Suchbegriff an jeder Leerstelle unterteilt wird. Dies bläht allerdings die Abfrage noch wieder deutlich auf.

Tipp

Die Abfragen, die zur Filterung und zum Durchsuchen von Daten genutzt werden, lassen sich auch direkt in ein Formular einbauen.
Die gesamten obigen Bedingungen sind bei den Formular-Eigenschaften in der Zeile Filter eintragbar. Aus
SELECT * FROM "Medien" WHERE "Titel" = COALESCE((SELECT "Suchbegriff" FROM "Filter" WHERE "ID" = TRUE), "Titel")
wird dann ein Formular, das als Inhalt die Tabelle "Medien" nutzt.
Unter «Filter» steht dann
("Medien"."Titel" = COALESCE((SELECT "Suchbegriff" FROM "Filter" WHERE "ID" = TRUE), "Medien"."Titel"))
In den Filtereingaben ist darauf zu achten, dass die Bedingung in Klammern gesetzt wird und jeweils mit der Angabe "Tabelle"."Feld" arbeitet.
Vorteil dieser Variante ist, dass der Filter bei geöffnetem Formular ein- und wieder ausgeschaltet werden kann.

Suche mit LOCATE oder POSITION

Die Suche mit **LIKE** ist in der Regel völlig ausreichend für Datenbanken mit Feldern, die Text in überschaubarem Maße enthalten. Was aber, wenn der Inhalt über Memo-Felder eingegeben wird, also ohne weiteres auch einmal mehrere Seiten Text enthalten kann? Dann geht die Suche erst einmal los, wo denn nun der Text zu finden ist.

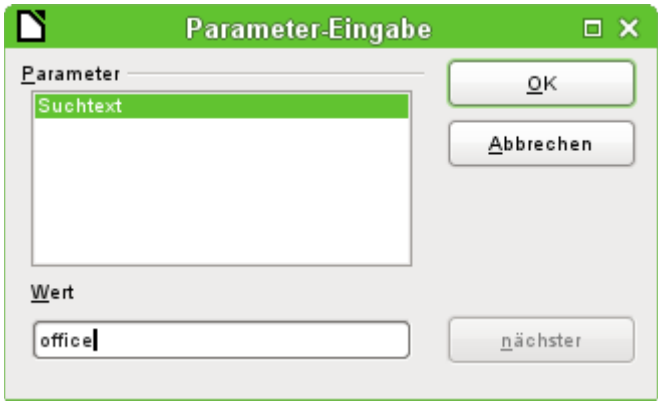
Um Text genau zu finden, gibt es in der HSQLDB die Funktion **LOCATE**. **LOCATE** erwartet einen Suchbegriff sowie den Text, der durchsucht werden soll, als Parameter. Zusätzlich *kann* noch angegeben werden, ab welcher Position gesucht werden soll. Kurz also: **LOCATE(Suchbegriff, Textfeld aus der Datenbank, Startposition der Suche)**.

In FIREBIRD gibt es die Funktion **LOCATE nicht**. Hier muss auf **POSITION** zurückgegriffen werden: **POSITION(Suchbegriff, Textfeld aus der Datenbank, Startposition der Suche)**. Die Startposition kann hier eingegeben werden, muss es aber nicht.

Auch die im weiteren verwendete Funktion **SUBSTRING** muss für FIREBIRD in einer andern Syntax geschrieben werden. Statt **SUBSTRING(Text,Startporition[,Länge])** ist dort **SUBSTRING(Text FROM Startposition [FOR Länge])** zu verwenden.

Für die folgende Erklärung wird eine Tabelle genutzt, die den Namen "Tabelle" hat. Der Primärschlüssel heißt "ID" und muss lediglich einzigartig sein. Zusätzlich gibt es noch ein Feld

"Memo", das als Feld des Typs **Memo (LONGVARCHAR)** erstellt wurde. In dem Feld "Memo" sind ein paar Absätze dieses Handbuchs gespeichert.[20]



Die Beispielabfragen sind als Parameterabfragen angelegt. Der einzugebende Suchtext ist jeweils 'office'.

	ID	Memo
▶	3	Jeder, der sich in das Modul Base von LibreOffice einarbeiten und tiefer einsteigen will, findet hier
	4	Dieses Buch, wie auch die anderen LibreOffice-Handbücher, das eingebaute Hilfesystem und die
	5	LibreOffice besitzt ein umfangreiches Hilfesystem.

Datensatz 1 von 3

```
SELECT "ID", "Memo" FROM "Tabelle"
WHERE LOWER ( "Memo" ) LIKE '%' || LOWER ( :Suchtext ) || '%'
```

Zuerst ein Zugriff über **LIKE**. **LIKE** kann nur in der Bedingung stehen. Wird der Suchtext irgendwo gefunden, dann wird der entsprechende Datensatz angezeigt. Durch den Vergleich von der Kleinschreibung des Feldinhaltes über **LOWER("Memo")** mit der Kleinschreibung des Suchtextes über **LOWER(:Suchtext)** werden die Inhalte unabhängig von der Schreibweise gefunden. Je länger der Text in dem Memo-Feld ist, desto schwerer wird es, den Begriff dann tatsächlich zu sehen.

	ID	Memo	Position
▶	1	Dieses Dokument unterliegt dem Copyright © 2014. Die Beitragenden sind unten aufgeführt. Sie	0
	2	Alles, was an eine Mailingliste geschickt wird, inklusive der E-Mail-Adresse und	0
	3	Jeder, der sich in das Modul Base von LibreOffice einarbeiten und tiefer einsteigen will, findet hier	44
	4	Dieses Buch, wie auch die anderen LibreOffice-Handbücher, das eingebaute Hilfesystem und die	40
	5	LibreOffice besitzt ein umfangreiches Hilfesystem.	6

Datensatz 1 von 5

```
SELECT "ID", "Memo", LOCATE( LOWER ( :Suchtext ), LOWER ( "Memo" ) ) AS "Position"
FROM "Tabelle"
```

LOCATE gibt genauer wieder, an welcher Stelle sich der Suchbegriff befindet. In Datensatz 1 und 2 ist der Suchbegriff nicht vorhanden. **LOCATE** gibt als Position hier '0' aus. Leicht nachzählen lässt sich das Ergebnis am Datensatz 5: Mit dem 6. Buchstaben beginnt hier die Textfolge 'Office'.

Natürlich wäre es auch möglich, das entsprechende Ergebnis wie bei **LIKE** auch über **LOCATE** zu erhalten:

```
001 SELECT "ID", "Memo"
002 FROM "Tabelle"
003 WHERE LOCATE(LOWER(:Suchtext),LOWER("Memo")) > 0
```

20 Die Screenshots zu diesem Kapitel entstammen der Datenbank «Beispiel_Autotext_Suchmarkierung_Rechtschreibung.odb», die dem Handbuch beiliegt.

Das Auffinden der Position allein ist im obigen Beispiel auch mit einem Blick auf das Feld "Memo" schon recht einfach. Komplizierter wird es aber, wenn der Inhalt eben nicht gerade, wie hier zur Demonstration, in den ersten 70 Zeichen enthalten ist. Dann wird es sinnvoll, Textstücke mit dem Inhalt direkt zu finden.

ID	Memo	Position	Treffer
1	Dieses Dokument	0	**keine Fundstelle**
2	Alles, was an eine	0	**keine Fundstelle**
3	Jeder, der sich in das	44	von LibreOffice einarbeiten und tiefer einsteigen will, findet hier die
4	Dieses Buch, wie auch	40	LibreOffice-Handbücher, das eingebaute Hilfesystem und die Benutzer-
5	LibreOffice besitzt ein	6	LibreOffice besitzt ein umfangreiches Hilfesystem.

Datensatz 1 von 5

```
SELECT "ID", "Memo", "Position",
    CASE
        WHEN "Position" = 0 THEN '**keine Fundstelle**'
        WHEN "Position" < 10 THEN SUBSTRING ( "Memo", 1 )
        ELSE SUBSTRING ( "Memo", LOCATE( ' ', "Memo", "Position" - 10 ) + 1 )
    END AS "Treffer"
FROM
    ( SELECT "ID", "Memo", LOCATE( LOWER ( :Suchtext ), LOWER ( "Memo" ) ) AS "Position"
    FROM "Tabelle" )
```

In der Spalte «Treffer» wird das Suchergebnis genauer dargestellt. Die vorherige Abfrage ist einfach als Basis für diese Abfrage genommen worden. Dies bewirkt, dass in der äußeren Abfrage nicht jedes Mal **LOCATE(LOWER(:Suchtext),LOWER("Memo"))** sondern einfach "Position" eingegeben werden kann. Vom Prinzip her ist dieses Verfahren nicht anders, als wenn die vorhergehende Abfrage gespeichert würde und diese Abfrage als Quellenangabe auf die vorherige Abfrage zugreift.

"Position" = 0 bedeutet, dass kein Suchergebnis vorhanden ist. In dem Fall also die Ausgabe **'**keine Fundstelle**'**.

"Position" < 10 bedeutet, dass sich der Suchbegriff direkt am Anfang des Textes befindet. 10 Zeichen können leicht überblickt werden. Es wird also der gesamte Text wiedergegeben. Hier könnte also auch statt **SUBSTRING("Memo",1)** direkt **"Memo"** stehen.

Für alle anderen Treffer wird ab 10 Zeichen vor der Position des Treffers nach einer Leerstelle **' '** gesucht. Der Text soll nicht mitten in einem Wort starten, sondern nach so einer Leerstelle beginnen. Über **SUBSTRING("Memo",LOCATE(' ',"Memo","Position"-10)+1)** wird erreicht, dass der Text mit dem Beginn eines Wortes startet, das maximal 10 Zeichen vor dem Begriff 'office' erscheint.

In der Praxis dürften hier mehr Zeichen erforderlich sein, da doch sehr viele Worte die Anzahl von 10 Zeichen übersteigen und selbst der Suchbegriff ja in einem Wort liegen kann, das noch 10 Zeichen vor dem eigentlich Begriff hat. 'LibreOffice' wird bei Suchbegriff 'office' so noch dargestellt, da das 'O' an der 6. Stelle steht. Stellen wir uns aber z.B. den Begriff 'hand' vor, so würde im 4. Datensatz bereits das Aus für die Darstellung stehen. 'LibreOffice-Handbücher' hat, von 'hand' aus nach links gezählt, 12 Zeichen. Wird aber höchsten 10 Zeichen nach links gesucht, so wird als erstes Leerzeichen das Zeichen hinter dem Komma gefunden. Die Darstellung in «Treffer» würde mit 'das eingebaute Hilfesystem ...' beginnen.

	ID	Memo	Position	Treffer
▶	1	Dieses Dokument unterliegt dem	0	**keine Fundstelle**
	2	Alles, was an eine Mailingliste	0	**keine Fundstelle**
	3	Jeder, der sich in das Modul Base von	44	von LibreOffice einarbeit
	4	Dieses Buch, wie auch die anderen	40	LibreOffice-Handbücher, d
	5	LibreOffice besitzt ein umfangreiches	6	LibreOffice besitzt ein u

Datensatz 1 von 5

```
SELECT "ID", "Memo", "Position",
    CASE
        WHEN "Position" = 0 THEN '**keine Fundstelle**'
        WHEN "Position" < 10 THEN SUBSTRING ( "Memo", 1 , 25 )
        ELSE SUBSTRING ( "Memo", LOCATE( ' ', "Memo", "Position" - 10 ) + 1 , 25 )
    END AS "Treffer"
FROM
    ( SELECT "ID", "Memo", LOCATE( LOWER ( :Suchtext ), LOWER ( "Memo" ) ) AS "Position"
    FROM "Tabelle" )
```

Die Abfragetechnik ist gegenüber der vorhergehenden Abfrage gleich geblieben. Lediglich die Länge des auszugebenden Treffers ist beschränkt worden. In diesem Falle erfolgte die Beschränkung hart auf 25 Zeichen. Die Funktion **SUBSTRING** erfordert als erstes die Angabe des zu durchsuchenden Textes, als zweites dann die Startposition der Ausgabe und als drittes optional die Länge des auszugebenden Textes. Natürlich hier auch reichlich kurz gehalten, aber eben nur zu Demonstrationszwecken. Vorteil der Verkürzung ist natürlich ein deutlich geringerer Speicherverbrauch bei entsprechend großen Datenmengen und ein direkter Blick auf die Fundstelle. Sichtbarer Nachteil dieser Form der Verkürzung ist aber, dass der Schnitt rigoros nach dem 25. Zeichen gemacht wird – ohne Rücksicht auf einen Wortbeginn.

	ID	Memo	Position	Treffer
▶	1	Dieses Dokument unterliegt dem	0	**keine Fundstelle**
	2	Alles, was an eine Mailingliste	0	**keine Fundstelle**
	3	Jeder, der sich in das Modul Base von	44	von LibreOffice einarbeiten und
	4	Dieses Buch, wie auch die anderen	40	LibreOffice-Handbücher, das
	5	LibreOffice besitzt ein umfangreiches	6	LibreOffice besitzt ein umfangreiches

Datensatz 1 von 5

```
SELECT "ID", "Memo", "Position",
    CASE
        WHEN "Position" = 0 THEN '**keine Fundstelle**'
        WHEN "Position" < 10 THEN SUBSTRING ( "Memo", 1, LOCATE( ' ', "Memo", 25 ) )
        ELSE SUBSTRING ( "Memo", LOCATE( ' ', "Memo", "Position" - 10 ) + 1,
            ( LOCATE( ' ', "Memo", "Position" + 20 ) -
                ( LOCATE( ' ', "Memo", "Position" - 10 ) + 1 ) )
            )
        END AS "Treffer"
FROM
    ( SELECT "ID", "Memo", LOCATE( LOWER ( :Suchtext ), LOWER ( "Memo" ) ) AS "Position"
    FROM "Tabelle" )
```

Hier wird ab dem 25. Zeichen in dem darzustellenden «Treffer» nach dem nächsten Leerzeichen gesucht. Der auszugebende Inhalt wird dann durch die gefundene Position begrenzt.

Recht einfach gestaltet sich dies noch, wenn der Treffer am Anfang liegt. Hier gibt **LOCATE(' ","Memo",25)** genau die Position vom Anfang des gesamten Textes an wieder. Sie entspricht, da der Text von Anfang an ausgegeben werden soll, auch genau der Länge des auszugebenden Begriffes.

Die Suche der dem Suchbegriff folgenden Leerzeichen ist auch bei einem weiter hinten liegenden Treffer nicht weiter kompliziert. Die Suche beginnt einfach an der Position des Treffers. Hinzugezählt werden noch 20 Zeichen, die auf jeden Fall folgen sollen. Danach wird das nächste Leerzeichen ausgemacht: **LOCATE(' ","Memo","Position"+20)**. Hiermit ist aber nur die Position im Gesamtfeld ausgemacht. Die ermittelte Position gibt also auf keinen Fall die Länge des auszugebenden Textes wieder. Von dem ermittelten Positionswert muss hingegen der Positionswert abgezogen werden, bei dem die Ausgabe des Treffers starten soll. Dies wurde durch

LOCATE(' ","Memo","Position"-10)+1 vorher bereits einmal abgefragt. Erst so kann dann die korrekte Länge des Textes dargestellt werden.

ID	Memo	Position01	Treffer01	Position02	Treffer02	Position03
1	Dieses	0	**keine Fundstelle**	0	**keine Fundstelle**	0
2	Alles, was	0	**keine Fundstelle**	0	**keine Fundstelle**	0
3	Jeder, der	44	von LibreOffice	312	einer OfficeSuite oder ein	0
4	Dieses	40	LibreOffice-Handbücher,	0	**keine Fundstelle**	0
5	LibreOffice	6	LibreOffice besitzt ein	123	Sie LibreOffice Hilfe aus dem	223

Datensatz 1 von 5

```
SELECT "ID", "Memo", "Position01", "Treffer01", "Position02",
    CASE
        WHEN "Position02" = 0 THEN '**keine Fundstelle**'
        WHEN "Position02" < 10 THEN SUBSTRING ( "Memo", 1, LOCATE( ' ', "Memo", 25 ) )
        ELSE SUBSTRING ( "Memo", LOCATE( ' ', "Memo", "Position02" - 10 ) + 1,
            ( LOCATE( ' ', "Memo", "Position02" + 20 ) -
                ( LOCATE( ' ', "Memo", "Position02" - 10 ) + 1 ) )
            )
    END AS "Treffer02",
    CASE
        WHEN "Position02" = 0 THEN 0
        ELSE LOCATE( LOWER ( :Suchtext ), LOWER ( "Memo" ), "Position02" + 1 )
    END AS "Position03"
FROM
    ( SELECT "ID", "Memo", "Position01",
        CASE
            WHEN "Position01" = 0 THEN '**keine Fundstelle**'
            WHEN "Position01" < 10 THEN SUBSTRING ( "Memo", 1, LOCATE( ' ', "Memo", 25 ) )
            ELSE SUBSTRING ( "Memo", LOCATE( ' ', "Memo", "Position01" - 10 ) + 1,
                ( LOCATE( ' ', "Memo", "Position01" + 20 ) -
                    ( LOCATE( ' ', "Memo", "Position01" - 10 ) + 1 ) )
                )
        END AS "Treffer01",
        CASE
            WHEN "Position01" = 0 THEN 0
            ELSE LOCATE( LOWER ( :Suchtext ), LOWER ( "Memo" ), "Position01" + 1 )
        END AS "Position02"
    FROM
        ( SELECT "ID", "Memo", LOCATE( LOWER ( :Suchtext ), LOWER ( "Memo" ) ) "Position01"
        FROM "Tabelle" )
    )
```

Mit der gleichen Technik können mehrere Abfragen hintereinander geschachtelt erfolgen. Die vorhergehende Abfrage ist jetzt die Datenquelle dieser Abfrage. Sie ist komplett unterhalb des Begriffes **FROM** in Klammern eingefügt worden. Lediglich die Felder wurden etwas umbenannt, da ja jetzt mehrere Positionen und Treffer angegeben werden. Außerdem wurde die nächste Position einer Fundstelle über **LOCATE(LOWER(:Suchtext),LOWER("Memo"),"Position01"+1)** ermittelt. Es wird also mit dem Suchen der nächsten Stelle einfach eine Stelle hinter dem vorhergehenden Treffer gestartet.

Die äußerste Abfrage stellt die entsprechenden Felder der anderen beiden Abfragen dar und fügt zusätzlich «Treffer02» auf die gleiche Weise hinzu, mit der vorher «Treffer01» ermittelt wurde. Außerdem wird in der äußeren Abfrage schon ermittelt, ob es vielleicht noch weitere Treffer gibt. Die entsprechende Position wird in «Position03» ausgegeben. Lediglich Datensatz 5 hat noch weitere Positionen mit Treffern vorzuweisen und könnte also in einer weiteren Nachfrage noch weitere Treffer ermöglichen.

Die Staffelung der Abfragen ist hier beliebig weit möglich. Allerdings werden sie mit jeder weiteren äußeren Abfrage natürlich für das System immer belastender. Hier sind entsprechende Tests notwendig, was denn nun sinnvoll und was realistisch machbar ist. Wie mit Hilfe von Makros über ein Formular sogar eine Abfragetechnik machbar ist, die gleichzeitig alle Fundstellen im Text markiert, ist im Kapitel «Makros» erklärt.

Bilder und Dokumente mit Base verarbeiten

Base-Formulare bieten für die Verarbeitung von Bildern grafische Kontrollfelder an. Nur über diese grafischen Kontrollfelder ist ohne Einsatz von Makros möglich, Bilder in die Datenbank einzulesen. Das grafische Kontrollfeld kann aber auch dazu genutzt werden, nur die Verknüpfung zu Bildern außerhalb der Datenbankdatei zu ermöglichen.[21]

Bilder in die Datenbank einlesen

Die Datenbank benötigt eine Tabelle, die mindestens die folgenden Voraussetzungen erfüllt:

Feldname	***Feldtyp***	***Beschreibung***
ID	Integer	Die ID ist Primärschlüssel dieser Tabelle.
Bild	Bild	Nimmt das Bild als Binärdatenstrom auf.

Hinweis

Bei Verwendung der internen Firebird Datenbank sollte nicht der Feldtyp **`Bild`** sondern der Feldtyp **`BLOB`** gewählt werden. Nur mit diesem Feldtyp werden Bilder auch im Formular angezeigt.

Beim Primärschlüssel ist natürlich nicht unbedingt der Integer-Feldtyp bestimmend. Ein Primärschlüssel ist aber unabdingbar. Andere Felder, die zumindest Informationen zu dem Bild enthalten, sollten noch hinzugefügt werden.

Daten, die irgendwann in das Bild-Feld eingetragen werden, sind in den Tabellen nicht lesbar. Dort erscheint als Anzeige nur **`<OBJECT>`**. Entsprechend sind Bilder auch nicht direkt in die Tabelle eingebbar. Sie müssen über ein Formular und dort mit Hilfe des grafischen Kontrollfeldes eingegeben werden. Das grafische Kontrollfeld öffnet beim Mausklick auf das Feld eine Dateiauswahl. Es zeigt hinterher das Bild an, das über die Dateiauswahl in die Datenbank eingelesen wurde.

Bilder, die direkt in die Datenbank eingefügt werden, sollten möglichst klein sein. Da Base ohne den Einsatz von Makros auch keine Möglichkeit bietet, die Bilder in Originalgröße wieder aus der Datenbank heraus zu befördern, macht es aus dieser Warte erst einmal Sinn, als Maßstab für die Größe z.B. einen möglichen Ausdruck im Bericht anzusehen. Originalbilder im Megapixelbereich sind hier völlig unnötig und blähen die Datenbank stark auf. Bereits nach wenigen Bildern meldet bei der internen HSQLDB Base eine **`Java.NullPointerException`** und kann den Datensatz nicht mehr speichern. Auch wenn die Bilder nicht ganz so groß sind, kann es irgendwann passieren, dass die Datenbankdatei nicht mehr bedienbar wird.

Bilder sollten außerdem möglichst nicht in Tabellen integriert werden, die als Suchgrundlage gedacht werden. Wird z.B. in einer Datenbank zur Personenverwaltung auch das Passbild mit abgespeichert, so ist es besser in einer separaten Tabelle über einen Fremdschlüssel mit der Haupttabelle verbunden. Die Suche in der Haupttabelle kann dann deutlich schneller erfolgen, da die Tabelle selbst nicht so viel Speicher beansprucht.

Bilder und Dokumente verknüpfen

Mit einer entsprechend durchdachten Ordnerstruktur ist es günstiger, direkt auf die Dateien von außerhalb zuzugreifen. Dateien außerhalb der Datenbank können beliebig groß sein, ohne dass die Funktionen der Datenbank selbst beeinflusst werden. Leider bedeutet dies aber auch, dass eine Umbenennung von Ordnern auf dem eigenen Rechner oder im Internet dazu führt, dass der Zugriff auf die entsprechenden Dateien verloren geht.

21 «Externe_Dateien.zip» ist als gepacktes Verzeichnis den zusätzlichen Beispieldatenbanken beigefügt.

Hinweis

Sollen die Bilder später über den Report-Designer ausgelesen werden, so dürfen die Dateinamen keine Sonderzeichen wie **[ ] { } \ < > % "** und Leerzeichen enthalten. Ein Bericht mit diesen Bildern wird nicht erstellt.

Um Bilder nicht in eine Datenbankdatei einzulesen, sondern nur zu verknüpfen, bedarf es nur einer kleinen Änderung gegenüber der vorhergehenden Tabelle:

Feldname	***Feldtyp***	***Beschreibung***
ID	Integer	Die ID ist Primärschlüssel dieser Tabelle.
Bild	Text	Nimmt den Pfad zu dem Bild auf.

Wird einfach statt des Feldtyps **Bild** der Feldtyp **Text** gewählt, so wird über das grafische Kontrollfeld der Pfad zu dem Bild eingetragen. Das Bild kann über das Kontrollfeld genauso betrachtet werden wie ein Bild, das in die Datenbank eingefügt wurde.

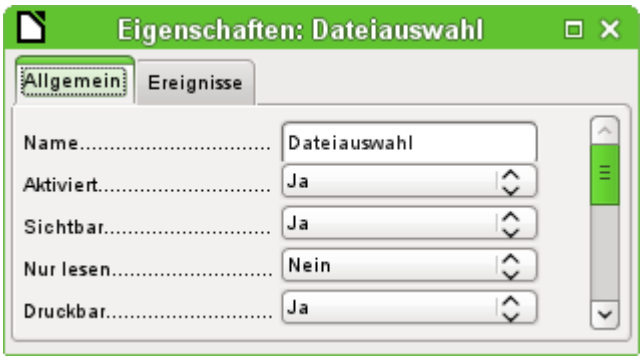
Hinweis

Seit LO 5.0 können, abhängig von der verwendeten Benutzeroberfläche, auch andere Dateien über das grafische Kontrollfeld eingebunden werden. Hier funktionierte das mit gtk3 allerdings erst ab der Version LO 6.3. **Für diese Versionen ist der Einsatz einer Dateiauswahl also weitgehend überflüssig.**
Bei *.pdf-Dateien wird die erste Seite in dem grafischen Kontrollfeld als Bild angezeigt.

Bei einem Bild kann über den Pfad wenigstens noch der Inhalt auf dem grafischen Kontrollfeld gelesen werden. Bei einem Dokument (mit Ausnahme des *.pdf-Dokumentes) kann aber keine Anzeige erfolgen, selbst wenn der Pfad in der Tabelle verzeichnet ist. Zuerst ist deshalb die Tabelle etwas zu erweitern, damit wenigstens ein geringfügiges Maß an Information zu dem Dokument sichtbar wird.

Feldname	***Feldtyp***	***Beschreibung***
ID	Integer	Die ID ist Primärschlüssel dieser Tabelle.
Beschreibung	Text	Beschreibung des Dokumentes, Suchbegriffe ...
Datei	Text	Nimmt den Pfad zu dem Bild auf.

Damit der Pfad zur Datei sichtbar wird, muss in dem Formular ein Dateiauswahlfeld mit eingebaut werden.



Ein Dateiauswahlfeld hat in seinen Eigenschaften keinen Reiter für Daten, ist also auch nicht mit irgendeinem Feld der dem Formular zugrundeliegenden Tabelle verbunden.

Dokumente mit absoluter Pfadangabe verknüpfen

Über das Dateiauswahlfeld kann zwar der Pfad angezeigt, aber nicht gespeichert werden. Hierfür ist eine gesonderte Prozedur erforderlich, die über **Ereignisse → Text modifiziert** ausgelöst wird:

```
001 SUB Pfad_Einlesen(oEvent AS OBJECT)
002    DIM oForm AS OBJECT
003    DIM oFeld AS OBJECT
004    DIM oFeld2 AS OBJECT
005    DIM stUrl AS STRING
006    oFeld = oEvent.Source.Model
007    oForm = oFeld.Parent
008    oFeld2 = oForm.getByName("GraphischesFeld")
009    IF oFeld.Text <> "" THEN
010       stUrl = ConvertToUrl(oFeld.Text)
011       oFeld2.BoundField.updateString(stUrl)
012    END IF
013 END SUB
```

Das auslösende Ereignis wird beim Aufruf der Prozedur mitgeliefert und hilft dabei, das Formular und auch das Feld zu finden, in dem der Pfad gespeichert werden soll. Mit **oEvent AS OBJECT** ist der Zugriff vor allem dann einfacher, wenn ein anderer User ein gleichlautendes Makro in irgendeinem Unterformular verwenden will. Das Dateiauswahlfeld ist von dort aus über **oEvent.Source.Model** zu erreichen. Das Formular ist **Parent** (Elternteil) zu dem Dateiauswahlfeld. Der Name des Formulars spielt also keine Rolle. Vom Formular aus geht dann der Zugriff auf das Feld mit dem Namen «GraphischesFeld». Über dieses Feld werden die Dateipfade der Bilder abgespeichert. Jetzt wird in das Feld die URL der ausgesuchten Datei geschrieben. Damit die URL betriebssystemunabhängig verwendet werden kann, wird der in dem Dateiauswahlfeld stehende Text vorher über **ConvertToUrl** in eine allgemeingültige URL-Schreibweise überführt.

In der Tabelle der Datenbank steht jetzt ein Pfad mit absoluter Schreibweise: **file:///...** .

Werden allerdings über ein graphisches Kontrollfeld Pfadeingaben eingelesen, so erfolgt die Aufnahme in relativer Pfadangabe. Dies ist von Vorteil, wenn die Datenbank in ein anderes System übertragen werden soll. So können z.B. in einem Unterverzeichnis der Datenbankdatei alle Bilder liegen und die Verbindung dazu wird auch auf anderen Rechner gefunden. Um dies mit der Dateiauswahl zu bewerkstelligen, müsste nachgebessert werden. Die Prozedur für diesen Zweck sieht wesentlich umfangreicher aus, da ein Vergleich der Pfadeingabe mit der aktuellen Speicherposition der Datenbankdatei nötig ist.

Dokumente mit relativer Pfadangabe verknüpfen

Das Dateiauswahlfeld liefert zuerst einmal nur den absoluten Pfad, der zwar für das Aufrufen der Datei gut nutzbar ist, aber eben den Transport der Datenbankdatei zusammen mit den Dokumenten erschwert. Die folgende Prozedur ist, wie im vorhergehenden Abschnitt, über **Ereignisse → Text modifiziert** an das Dateiauswahlfeld gebunden.[22]

```
001 SUB SaveFilePath(oEvent AS OBJECT)
002    DIM oField AS OBJECT
003    DIM oForm AS OBJECT
004    DIM oDoc AS OBJECT
005    DIM stUrlNew AS STRING
006    DIM stUrl AS STRING
007    DIM i AS INTEGER, ina AS INTEGER, inb AS INTEGER, inx AS INTEGER, iny AS INTEGER
008    oField = oEvent.Source.Model
009    oForm = oField.Parent
```

Nach der Deklaration der Variablen und dem Aufsuchen des Feldes und des Formulars wird erst einmal geklärt, ob das Feld überhaupt einen Inhalt aufweisen kann. Ohne Inhalt braucht auch nichts weiter eingelesen zu werden.

22 Dieses Makro ist in der Datenbank «Beispiel_Formular_Eingabekontrolle.odb» enthalten. Bei Verwendung des grafischen Kontrollfeldes ist der Einsatz des Makros nicht mehr nötig.

```
010     IF oField.Text <> "" THEN
011        stUrl = ConvertToUrl(oField.Text)
```

Bei dem Dateiauswahlfeld fehlt **file:///** am Anfang. Ansonsten ist dieser Pfad absolut. Der Pfad der Dateiauswahl wird jetzt mit dem Pfad der geöffneten Base-Datei verglichen.

```
012        oDoc = ThisComponent
013        a = split(oDoc.Parent.Url,"/")
014        b = split(stUrl,"/")
015        ina = UBound(a()) - 1
```

Beide Pfade werden zu Arrays aufgesplittet. Der Trenner ist der Frontslash. Der Dateiname der Base-Datei wird von der URL des Base-Dokumentes abgetrennt, da er bei der Pfadermittlung nicht mitgezählt werden darf.

Die Größe der Arrays wird verglichen und damit die maximale Anzahl der gleichen Elemente festgestellt.

```
016        inb = UBound(b())
017        IF ina > inb THEN
018           inx = inb
019        ELSE
020           inx = ina
021        END IF
```

Die beiden Arrays werden je Element miteinander verglichen. Die gleichen Elemente werden gezählt. Der Zähler wird um 1 erhöht, damit die gleichen Elemente anschließend beim Auslesen der Arrays nicht mehr berücksichtigt werden.

```
022        FOR i = 0 TO inx
023           IF a(i) = b(i) THEN
024              iny = i + 1
025           ELSE
026              EXIT FOR
027           END IF
028        NEXT
```

Für jedes Element, das nach dem Vergleich in der URL der Base-Datei liegt, wird ein Schritt aufwärts benötigt: **../**

```
029        FOR i = iny TO ina
030           stUrlNew = stUrlNew & "../"
031        NEXT
```

An die Aufwärtsschritte wird der verbleibende Pfad mit der ausgewählten Datei angehängt. Auch diese Schleife startet mit dem ersten ermittelten Unterschied der beiden Arrays.

```
032        FOR i = iny TO inb
033           stUrlNew = stUrlNew & b(i) & "/"
034        NEXT
```

Die Pfadangabe ist um einen Frontslash zu groß und wird daher um diesen Frontslash gekürzt. Anschließend wird der relative Pfad in das Zielfeld des Formulars übertragen. In dem aufrufenden Feld zur Dateiauswahl wird in den Zusatzinformationen (**Tag**) der Name des Tabellenfeldes der Datenquelle aufgeführt. So kann über das Tabellenfeld der Datenquelle direkt die neue URL in das Formular geschrieben werden – und dies unabhängig davon, ob das Formularfeld ein Textfeld ist, sich in einem Tabellenkontrollfeld befindet oder eventuell gar nicht in dem Formular sichtbar ist, sondern nur zur Datenquelle des Formulars gehört.

```
035        stUrlNew = Left(stUrlNew, len(stUrlNew) - 1)
036        oForm.updateString(oForm.findColumn(oField.Tag), stUrlNew)
037     END IF
038  END SUB
```

Verknüpfte Bilder und Dokumente anzeigen

Verknüpfte Bilder können direkt in dem kleinen graphischen Kontrollfeld angezeigt werden. Besser wäre aber ein größere Anzeige, um auch Details erkennen zu können.

Dokumente lassen sich standardmäßig in Base überhaupt nicht anzeigen.

Um dennoch eine Anzeige zu ermöglichen, bedarf es wieder einer Makrolösung. Dieses Makro wird über einen Button in dem Formular gestartet, in dem das graphische Kontrollfeld liegt.[23]

```
001 SUB Betrachten(oEvent AS OBJECT)
002    DIM oDoc AS OBJECT
003    DIM oForm AS OBJECT
004    DIM oFeld AS OBJECT
005    DIM oShell AS OBJECT
006    DIM stUrl AS STRING
007    DIM stFeld AS STRING
008    DIM arUrl_Start()
009    oDoc = thisComponent
010    oForm = oEvent.Source.Model.Parent
011    oFeld = oForm.getByName("GraphischesFeld")
012    stUrl = oFeld.BoundField.getString
```

Das graphische Kontrollfeld im Formular wird aufgesucht. Da in der Tabelle nicht das Bild selbst, sondern nur der Pfad als Text gespeichert wird, wird hier über **getString** dieser Text ausgelesen.

Anschließend wird der Pfad zu der Datenbankdatei ermittelt. Mit **oDoc.Parent** wird die *.odb-Datei erreicht. Sie ist der Container für die Formulare. Über **oDoc.Parent.Url** wird schließlich die gesamte URL incl. Dateinamen ausgelesen. Der Dateiname ist auch zu sehen in **oDoc.Parent.Title**. Der Text wird jetzt mit der Funktion **split** aufgetrennt, wobei als Trenner der Dateiname, umgewandelt in Url-Schreibweise, benutzt wird. Die Auftrennung gibt so nur als erstes und einziges Element des Arrays den Pfad zur *.odb-Datei wieder.

```
013    arUrl_Start = split(oDoc.Parent.Url,right(convertToUrl(oDoc.Parent.Title),
014       len(convertToUrl(oDoc.Parent.Title))-8))
015    oShell = createUnoService("com.sun.star.system.SystemShellExecute")
016    stFeld = convertToUrl(arUrl_Start(0) + stUrl)
017    oShell.execute(stFeld,,0)
018 END SUB
```

Externe Programme können über das **Struct com.sun.star.system.SystemShellExecute** gestartet werden. Dem externen Programm wird hier nur der Pfad zur Datei mitgegeben, der aus dem Pfad zur Datenbankdatei und dem intern gespeicherten relativen Pfad von der Datenbankdatei aus zusammengesetzt wurde. Die grafische Benutzeroberfläche des Betriebssystems entscheidet jetzt darüber, mit welchem Programm die entsprechende Datei zu öffnen ist.

Mit dem Kommando **oShell.execute** werden 3 Parameter übergeben. Als erstes wird eine ausführbare Datei oder der Pfad zu einer Datei aufgeführt, die im System mit einem Programm verbunden sind. Als zweites werden Parameter aufgeführt, mit denen das Programm gestartet werden soll. Als drittes wird über eine Ziffer mitgeteilt, wie mit Fehlermeldungen des Systems bei missglückter Ausführung umzugehen ist. Hier stehen 0 (Standard), 1 (keine Meldung anzeigen) und 2 (nur das Öffnen von absoluten URLs erlauben) zur Verfügung.

Dokumente in die Datenbank einlesen

Beim Einlesen der Dokumente sollten folgende Bedingungen immer im Auge behalten werden:[24]

- Je größer die Dokumente sind, desto schwerfälliger wird die Datenbank. Bei entsprechend großen Dokumenten, vor allem Bildern, ist deshalb eine externe Datenbank der internen Datenbank vorzuziehen.
- In Dokumenten kann ebenso wenig wie in Bildern recherchiert werden. Sie werden als Binärdaten gespeichert und können in einem Feld gespeichert werden, das einem Bildfeld entspricht..

23 Es ist auch möglich, so ein Ereignis durch eine Kombination van z.B. Strg + Maustaste auszulösen. Siehe dazu die Datenbank "Beispiel_Formular_Eingabekontrolle.odb"

24 Die Datenbank «Beispiel_Dokumente_einlesen_auslesen.odb» liegt diesem Handbuch bei.

- Dokumente, die in die Datenbank eingelesen werden, können nur über den Makroweg auch wieder ausgelesen werden. Ein SQL-Weg ist für die interne Datenbank nicht erreichbar.

Die folgenden Makros zum Ein-und Auslesen bauen dabei auf eine Tabelle auf, die neben der Datei im Binärformat eine Beschreibung der Datei und den ursprünglichen Dateinamen enthalten soll. Schließlich wird der Dateiname ja nicht mit abgespeichert und sollte beim Auslesen der Datei aber darauf Schlüsse zulassen, um welchen Dateityp es sich denn handelt. Nur dann kann die Datei auch zum Lesen durch andere Programme einwandfrei erkannt werden.

Die Tabelle enthält die folgenden Felder:

Feldname	***Feldtyp***	***Beschreibung***
ID	Integer	Die ID ist Primärschlüssel dieser Tabelle.
Beschreibung	Text	Beschreibung des Dokumentes, Suchbegriffe ...
Datei	Bild FIREBIRD: BLOB	Nimmt den binären Inhalt des Bildes oder der Datei auf.
DateiName	Text	Soll die Bezeichnung der Datei mit Dateiendung abspeichern. Wichtig für das spätere Auslesen.

Das Formular zum Einlesen und wieder Ausgeben der Dateien sieht so aus:

Solange sich Bilddateien in der Datenbank befinden, können diese Dateien auch in dem graphischen Kontrollfeld angesehen werden. Alle anderen Dateien mit Ausnahme der ersten Seiten von *.pdf-Dateien werden in dem Kontrollfeld nicht sichtbar.

Das folgende Makro für das Einlesen der Dateien wird über **Eigenschaften: Dateiauswahl → Ereignisse → Text modifiziert** ausgelöst.

```
001 SUB DateiEinlesen_mitName(oEvent AS OBJECT)
002    DIM oForm AS OBJECT
003    DIM oFeld AS OBJECT
004    DIM oFeld2 AS OBJECT
005    DIM oFeld3 AS OBJECT
006    DIM oStream AS OBJECT
007    DIM oSimpleFileAccess AS OBJECT
008    DIM stUrl AS STRING
009    DIM stName AS STRING
010    oFeld = oEvent.Source.Model
011    oForm = oFeld.Parent
012    oFeld2 = oForm.getByName("DateiName")
```

```
013     oFeld3 = oForm.getByName("GraphischesFeld")
014     IF oFeld.Text <> "" THEN
015        stUrl = ConvertToUrl(oFeld.Text)
016        ar = split(stUrl,"/")
017        stName = ar(UBound(ar))
018        oFeld2.BoundField.updateString(stName)
019        oSimpleFileAccess = createUnoService("com.sun.star.ucb.SimpleFileAccess")
020        oStream = oSimpleFileAccess.openFileRead(stUrl)
021        oFeld3.BoundField.updateBinaryStream(oStream, oStream.getLength())
022     END IF
023  END SUB
```

Da das Makro über das auslösende Ereignis die Position der anderen Formularfelder ermittelt, muss nicht besonders überprüft werden, ob die Felder nun in einem Formular oder Unterformular liegen. Alle Felder müssen lediglich im gleichen Formular positioniert sein.

Das Feld «DateiName» speichert den Namen der Datei ab, die ausgesucht wird. Bei Bildern muss dieser Name ohne ein zusätzliches Makro händisch eingegeben werden. Hier wird stattdessen der Dateiname aus der URL ermittelt und automatisch beim Einlesen der Datei mit eingefügt.

Das Feld «GraphischesFeld» speichert die Daten in dem gemeinsamen Feld für Bilder und Dateien ab.

Aus dem Dateiauswahlfeld wird über **oFeld.Text** der Pfad komplett mit Dateinamen ausgelesen. Damit die URL-Schreibweise nicht an systemspezifischen Bedingungen scheitert, wird der ausgelesene Text mit **ConvertToUrl** in eine allgemeingültige URL umgewandelt. Die so erstellte allgemeingültige URL wird in ein Array aufgeteilt. Der Trenner ist das /. Letztes Element dieser Pfadangabe ist der Dateiname. **Ubound(ar)** gibt die Nummer für das letzte Element an. Daher kann der Dateiname über **ar(Ubound(ar))** direkt ausgelesen und anschließend als String an das Feld übergeben werden.

Um die Datei selbst einzulesen, muss der **UnoService com.sun.star.ucb.SimpleFileAccess** bemüht werden. Über diesen Service kann der Inhalt der Datei als Datenstrom ausgelesen werden. Anschließend wird der so in dem Objekt **oStream** zwischengespeicherte Inhalt wiederum als Datenstrom in das Feld eingefügt, das mit dem Feld "Datei" der Tabelle verbunden ist. Dabei muss neben dem Objekt **oStream** auch die Länge des Datenstromes als Parameter angegeben werden.

Die Daten sind jetzt wie eine normale Eingabe in die Formularfelder eingefügt. Wird das Formular einfach geschlossen, so sind die Daten noch nicht abgespeichert. Die Speicherung erfolgt erst bei Betätigung des Speicherbuttons in der Navigationsleiste oder automatisch bei der Navigation zum nächsten Datensatz.

Bildnamen ermitteln

Bei dem obigen Verfahren wurde kurz erwähnt, dass der Name der Datei bei der Eingabe über das graphische Kontrollfeld so nicht ermittelt werden kann. Hier jetzt kurz ein Makro zur Ermittlung des Dateinamens, das zum obigen Formular passt. Der Dateiname lässt sich nicht sicher durch ein Ereignis ermitteln, das mit dem grafischen Kontrollfeld direkt verbunden ist. Deswegen wird das Makro über **Formular-Eigenschaften → Ereignisse → Vor der Datensatzaktion** gestartet.

```
001  SUB BildnamenAuslesen(oEvent AS OBJECT)
002     oForm = oEvent.Source
003     IF InStr(oForm.ImplementationName, "ODatabaseForm") THEN
004        oFeld = oForm.getByName("GraphischesFeld")
005        oFeld2 = oForm.getByName("DateiName")
006        IF oFeld.ImageUrl <> "" THEN
007           stUrl = ConvertToUrl(oFeld.ImageUrl)
008           ar = split(stUrl,"/")
009           stName = ar(UBound(ar))
010           oFeld2.BoundField.updateString(stName)
011        END IF
```

```
012     END IF
013 END SUB
```

Vor der Datensatzaktion werden zwei Implementationen mit unterschiedlichem Implementationsnamen ausgeführt. Das Formular ist am einfachsten über die Implementation erreichbar, das in seinem Namen als **ODatabaseForm** bezeichnet wird.

In dem graphischen Kontrollfeld ist die URL der Datenquelle über die **ImageUrl** erreichbar. Diese URL wird ausgelesen, der Dateinamen wie in der vorhergehenden Prozedur «DateiEinlesen_mitName» ausgelesen und in das Feld «DateiName» übertragen.

Bildnamen aus dem Speicher entfernen

Wird nach dem Ablauf des obigen Makros zum nächsten Datensatz gewechselt, so ist der Pfad zu dem ursprünglichen Bild weiter vorhanden. Würde jetzt eine allgemeine Datei über das Dateiauswahlfeld eingelesen, so würde der Name der Datei ohne das folgende Makro einfach durch den Namen der zuletzt eingelesenen Bilddatei überschrieben.

Der Pfad kann leider nicht mit dem vorhergehenden Makro entfernt werden, da das Einlesen der Bilddatei erst beim Abspeichern erfolgt. Eine Entfernung des Pfades vor der Abspeicherung löscht das Bild.

Das Makro wird über **Formular-Eigenschaften → Ereignisse → Nach der Datensatzaktion** gestartet.

```
001 SUB BildnamenZuruecksetzen(oEvent AS OBJECT)
002    oForm = oEvent.Source
003    IF InStr(oForm.ImplementationName, "ODatabaseForm") THEN
004       oFeld = oForm.getByName("GraphischesFeld")
005       IF oFeld.ImageUrl <> "" THEN
006          oFeld.ImageUrl = ""
007       END IF
008    END IF
009 END SUB
```

Es wird, wie in der Prozedur «BildnamenAuslesen», auf das graphische Kontrollfeld zugegriffen. Existiert dort noch ein Eintrag in der **ImageUrl**, dann wird dieser geleert.

Bilder und Dokumente auslesen und anzeigen

Für Dateien wie für die Bilder in Originalgröße gilt, dass der Button Datei mit externem Programm betrachten ausgelöst werden muss. Dann werden die Dateien in das temporäre Verzeichnis ausgelesen und anschließend über das mit der Endung verknüpfte Programm des Betriebssystems angezeigt.

Das Makro wird über **Eigenschaften: Schaltfläche → Ereignisse → Aktion ausführen** gestartet.

```
001 SUB DateiAuslesen_mitName(oEvent AS OBJECT)
002    DIM oForm AS OBJECT
003    DIM oFeld AS OBJECT
004    DIM oFeld2 AS OBJECT
005    DIM oStream AS OBJECT
006    DIM oShell AS OBJECT
007    DIM oPath AS OBJECT
008    DIM oSimpleFileAccess AS OBJECT
009    DIM stName AS STRING
010    DIM stPfad AS STRING
011    DIM stFeld AS STRING
012    oForm = oEvent.Source.Model.Parent
013    oFeld = oForm.getByName("GraphischesFeld")
014    oFeld2 = oForm.getByName("DateiName")
015    stName = oFeld2.Text
016    IF stName = "" THEN
017       stName = "Datei"
018    END IF
```

```
019     oStream = oFeld.BoundField.getBinaryStream
020     oPath = createUnoService("com.sun.star.util.PathSettings")
021     stPfad = oPath.Temp & "/" & stName
022     oSimpleFileAccess = createUnoService("com.sun.star.ucb.SimpleFileAccess")
023     oSimpleFileAccess.writeFile(stPfad, oStream)
024     oShell = createUnoService("com.sun.star.system.SystemShellExecute")
025     stFeld = convertToUrl(stPfad)
026     oShell.execute(stFeld,,0)
027 END SUB
```

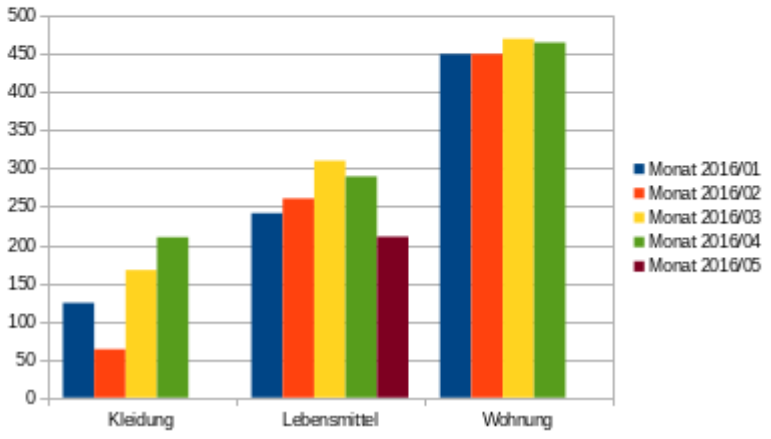
Die Lage der anderen betroffenen Felder im Formular wird abhängig von dem auslösenden Button ermittelt. Fehlt ein Dateiname, so wird der ausgelesenen Datei einfach der Name «Datei» zugeschrieben.

Der Inhalt des Formularfeldes «GraphischesFeld» entspricht dem Inhalt, der in dem Feld "Datei" der Tabelle liegt. Er wird als Datenstrom ausgelesen. Als Pfad für die ausgelesene Datei wird der Pfad zum temporären Verzeichnis genutzt, der in LibreOffice über **Extras → Optionen → LibreOffice → Pfade** eingestellt werden kann. Soll also die Datei anschließend noch anderweitig verwendet und nicht nur angezeigt werden, so ist sie aus diesem Pfad heraus auch kopierbar. Innerhalb des Makros wird die Datei direkt nach dem erfolgreichen Auslesen mit dem Programm geöffnet, das in der grafischen Benutzeroberfläche des Betriebssystems mit dem Dateityp verbunden ist.

Diagramme in Formulare einbinden

Das Einbinden von Diagrammen in Formulare ist von der GUI her nicht vorgesehen. Mit einem Umweg über Writer, etwas Makrohilfe und allgemein verarbeitbaren Ansichten lässt sich so ein Diagramm dennoch in ein Formular integrieren und darüber hinaus beständig an die momentane Datenlage anpassen.

ID	Kategorie	Datum	Betrag
1	Lebensmittel	15.01.16	241,51 €
2	Lebensmittel	15.02.16	260,75 €
3	Kleidung	15.01.16	124,00 €
4	Wohnung	15.01.16	450,00 €
5	Kleidung	15.02.16	63,20 €
6	Wohnung	15.02.16	450,00 €
7	Lebensmittel	15.03.16	310,23 €
8	Kleidung	15.03.16	167,00 €
9	Wohnung	15.03.16	470,00 €
10	Lebensmittel	15.04.16	289,50 €
11	Kleidung	15.04.16	210,00 €
12	Wohnung	15.04.16	465,00 €
13	Lebensmittel	15.05.16	210,76 €



Als Datenbank für die Diagrammdarstellung wurden beispielhaft Daten aus einer Haushaltsführung und Daten aus einer Temperaturmessung über einen kürzeren Zeitraum verwandt.[25] Während sich die Haushaltsführung für verschiedene Diagrammtypen eignet, ist die Messung eines Temperaturverlaufes auf ein XY-Diagramm zugeschnitten.

25 Die Datenbank «Beispiel_Baseformular_mit_Diagramm» ist den Beispieldatenbanken für dieses Handbuch beigefügt.

Diagramme aus dem Writer importieren

Zuerst wird ein neues Writer-Dokument erstellt: **Datei → Neu → Textdokument**. Anschließend wird über die Symbolleiste oder über **Einfügen → Objekt → Diagramm** ein Diagramm erstellt. Standardmäßig wird hier zuerst einmal ein Säulendiagramm in das Dokument eingefügt.

Das Diagramm wird kopiert und in ein Formular der Datenbank eingefügt. Alle weiteren Einstellungen des Diagramms können auch im Base-Formular über das Kontextmenü vorgenommen werden.

Daten für das Diagramm werden später automatisch eingelesen. Es ist also nicht notwendig, hier irgendeine Voreinstellung vorzunehmen.

Abfragen erstellen und als Ansichten speichern

Für die Darstellung der verschiedenen Diagrammtypen werden Zeilenbeschreibungen (**RowDescriptions**), Spaltenbeschreibungen (**ColumnDescriptions**) und Daten (**Data**) benötigt. Um Probleme bei der Darstellung von XY-Diagrammen zu umgehen, wird zusätzlich eine Information zum Diagrammtyp in den Abfragen dargestellt.

Abfrage für ein Säulendiagramm

Würde von vornherein sichergestellt, dass nur ein Eintrag pro Monat erstellt würde und das Diagramm nur nach allen Einträgen für den Monat aktualisiert werden soll, so könnte für ein **Säulendiagramm** einfach die Tabelle direkt abgefragt werden:

```
001 SELECT "Kategorie", "Datum", "Betrag", 'BC' AS "Type" FROM "Kategorien"
```

Die Kategorien stellen die Zeilenbeschriftungen dar. Die Zeilenbeschriftungen erscheinen unter den Säulen des Diagramms. Für jeden Monat wird eine neue Säule dargestellt. Die Monatsbezeichnungen erscheinen beim Säulendiagramm als Spaltenbeschriftungen rechts vom Diagramm in der Legende. Der Betrag gibt als Dezimalzahl die Höhe der jeweiligen Säule an. Die Information zum Diagrammtypen wird später nur für das XY-Diagramm separat ausgewertet. 'BC' steht hier schlicht für «bar chart» (Säulendiagramm).

Damit auch mehrere Einträge pro Monat angefertigt werden können, sind die Einträge für alle gleichen Kategorien innerhalb eines Monats zusammen zu fassen. Die Darstellung des Monats soll in dem Diagramm z.B. als «Monat 2016/02» erfolgen. Außerdem sollen die Einträge in dem Diagramm auch dann aktualisiert werden, wenn für eine Kategorie eventuell noch gar kein Monatseintrag existiert.

Um all dies zu bewerkstelligen, wird zuerst einmal eine Kombination aller Kategorien mit allen in der Tabelle "Kategorien" vorhandenen Monaten erstellt:

```
001 SELECT
002    "a"."Kategorie",
003    "b"."Monat"
004 FROM "Kategorien" AS "a",
005    ( SELECT 'Monat ' || YEAR( "Datum" ) || '/' || RIGHT( '0'
006       || MONTH( "Datum" ), 2 ) AS "Monat" FROM "Kategorien" ) AS "b"
007 ORDER BY "Kategorie", "Monat"
```

Diese Konstruktion kann allerdings den Datenbestand deutlich aufblähen. Hat die Tabelle "Kategorien" 5 Einträge, so werden bei dieser Abfrage 5 Einträge zu "Kategorie" mit 5 Einträgen zu der Monatsdarstellung kombiniert. Es entstehen zwangsläufig 25 Einträgen, von denen dann gewiss viele Einträge mehrfach vorkommen.

Hinweis

Für Firebird muss statt **MONTH("Datum") EXTRACT(MONTH FROM "Datum")** usw. eingefügt werden. Firebird kennt die hier verwendeten Kurzformen nicht.

Mit Hilfe des Zusatzes **DISTINCT** kann die Ausgabe der Einträge eingeschränkt werden. Dies sollte möglichst schon bei den Datenquellen für die Abfrage ansetzen, damit nicht unnötig erst Daten geladen und wieder aus dem Speicher entfernt werden. Deshalb wird auf die Tabelle "Kategorien" zweimal mit einer Abfrage und dem Zusatz **DISTINCT** zugegriffen:

```
001 SELECT
002    "a"."Kategorie",
003    "b"."Monat"
004 FROM ( SELECT DISTINCT "Kategorie" FROM "Kategorien" ) AS "a",
005    ( SELECT DISTINCT 'Monat ' || YEAR( "Datum" ) || '/' || RIGHT( '0'
006       || MONTH( "Datum" ), 2 ) AS "Monat" FROM "Kategorien" ) AS "b"
007 ORDER BY "Kategorie", "Monat"
```

Mit Hilfe einer korrelierenden Unterabfrage wird jetzt zu jeder Kombination von "Monat" und "Kategorie" der entsprechende "Betrag" aufsummiert. So werden auch mehrere Einträge im Monat bei der Darstellung im Diagramm berücksichtigt:

```
001 SELECT
002    "a"."Kategorie" AS "RowDescription",
003    "b"."Monat" AS "ColumnDescription",
004    ( SELECT SUM( "Betrag" ) FROM "Kategorien" WHERE 'Monat ' ||
005       YEAR( "Datum" ) || '/' || RIGHT( '0' || MONTH( "Datum" ), 2 ) =
006       "b"."Monat" AND "Kategorie" = "a"."Kategorie" ) AS "Data",
007    'BC' AS "Type"
008 FROM ( SELECT DISTINCT "Kategorie" FROM "Kategorien" ) AS "a",
009    ( SELECT DISTINCT 'Monat ' || YEAR( "Datum" ) || '/' || RIGHT( '0'
010       || MONTH( "Datum" ), 2 ) AS "Monat" FROM "Kategorien" ) AS "b"
011 ORDER BY "Kategorie", "Monat"
```

Diese Abfrage wird abgespeichert, mit der rechten Maustaste angeklickt und über **Abfrage → Als Ansicht erstellen** unter der Bezeichnung «Chart_Kategorie» als Ansicht gespeichert.

Abfrage für ein Kreisdiagramm

Für die Darstellung eines **Kreisdiagramms** ist es erforderlich, dass statt der einzelnen Monatsdarstellungen die Summe über alle Monate erstellt wird. Hier wird der Einfachheit halber direkt auf die eben erstellte Ansicht zugegriffen:

```
001 SELECT
002    "RowDescription",
003    '' AS "ColumnDescription",
004    SUM( "Data" ) AS "Data",
005    'CC' AS "Type"
006 FROM "Chart_Kategorie"
007 GROUP BY "RowDescription"
```

Nach der "Kategorie", hier unter dem Alias "RowDescription", wird die Abfrage gruppiert. Für das Feld "Data" wird die Summe des Feldes "Data" aus der Ansicht "Chart_Kategorie", bezogen auf diese Gruppierung, erstellt.

Eine Spaltenbeschreibung ist nicht erforderlich und wird mit einem leeren Text versehen. Als Typ des Diagramms wird 'CC' für «circle chart» (Kreisdiagramm) angegeben.

Die Abfrage wird abgespeichert, mit der rechten Maustaste angeklickt und über **Abfrage → Als Ansicht erstellen** unter der Bezeichnung «Chart_Kategorie_Circle» als Ansicht gespeichert.

Abfrage für ein XY-Diagramm

Daten für ein **XY-Diagramm** liegen in der Regel in anderer Weise vor als für Säulendiagramme oder Kreisdiagramme. Statt einen Wert in einer Zeile zu speichern, werden hier Wertepaare gespeichert. Damit dennoch auf die Daten mit der gleichen Makrokonstruktion zugegriffen werden kann, ist eine besondere Abfragekonstruktion vorgesehen, die wieder in der ersten Spalte

die Zeilenbeschreibung, in der zweiten Spalte die Spaltenbeschreibung, in der dritten Spalte die Daten und in der vierten Spalte den Diagrammtyp wiedergeben kann.

```
001 SELECT
002    "Zeit" AS "RowDescription",
003    '' AS "ColumnDescription",
004    "Zeit" AS "Data",
005    'XY' AS "Type"
006 FROM "Temperaturverlauf"
007 UNION
008 SELECT
009    "Zeit" AS "RowDescription",
010    'Temperatur [°C]' AS "ColumnDescription",
011    "Temperatur" AS "Data",
012    'XY' AS "Type"
013 FROM "Temperaturverlauf"
014 ORDER BY "RowDescription" ASC, "ColumnDescription" ASC
```

Zuerst werden alle Zeiten ausgelesen. Das Ergebnis dieser Abfrage wird mit der Folgeabfrage über **UNION** kombiniert, die jetzt alle Temperaturdaten ausliest. Leider funktioniert diese Kombination nicht wie gewünscht, da die Zeit einen anderen Datentypen hat als die Temperatur. Es kommt noch hinzu, dass mit dem Datentypen für die Zeit leider auch keine kontinuierliche Darstellung in einem XY-Diagramm möglich ist. Aus der Zeit muss durch Umformung ein Dezimalwert erstellt werden. Dies erfolgt, indem der Tag als Grundmaß angesehen wird. Stunden, Minuten und Sekunden werden als Bruchteile des Tages errechnet und addiert.

```
001 SELECT
002    HOUR( "Zeit" ) / 24.00000 + MINUTE( "Zeit" ) / 1440.00000 +
003       SECOND( "Zeit" ) / 86400.00000 AS "RowDescription",
004    '' AS "ColumnDescription",
005    HOUR( "Zeit" ) / 24.00000 + MINUTE( "Zeit" ) / 1440.00000 +
006       SECOND( "Zeit" ) / 86400.00000 AS "Data",
007    'XY' AS "Type"
008 FROM "Temperaturverlauf"
009 UNION
010 SELECT
011    HOUR( "Zeit" ) / 24.00000 + MINUTE( "Zeit" ) / 1440.00000 +
012       SECOND( "Zeit" ) / 86400.00000 AS "RowDescription",
013    'Temperatur [°C]' AS "ColumnDescription",
014    "Temperatur" AS "Data",
015    'XY' AS "Type"
016 FROM "Temperaturverlauf"
017 ORDER BY "RowDescription" ASC, "ColumnDescription" ASC
```

Hinweis

Für Firebird muss statt **HOUR("Zeit") / 24.00000 + MINUTE("Zeit") / 1440.00000 +SECOND("Zeit") / 86400.00000** einfach **("Zeit" – TIME '00:00')/86400.00000** eingefügt werden. Firebird kennt die hier verwendeten Kurzformen nicht, kann aber Datumswerte und Zeitwerte voneinander subtrahieren.

Die Zeilenbeschriftung "RowDescription" wird für das XY-Diagramm gar nicht benötigt. Hier stehen schließlich die Daten der X-Achse. Diese Spalte wird über die zu einer Dezimalzahl umformatierten Inhalte dazu genutzt, die Zeilen in der korrekten Reihenfolge zu sortieren.

Das Diagramm stellt lediglich die Werte der Y-Achse als 'Temperatur [°C]' dar. Deswegen wird für die Zeilen, die jetzt die Zeitangaben als Daten wiedergeben, eine leere Spaltenbeschriftung "ColumnDescription" ausgegeben.

Der gesamte Inhalt wird nach den Zeiten in "RowDescription" und den Einträgen in "ColumnDescription" sortiert, so dass die Werte für X- und Y-Achse direkt aufeinander folgen und durch gleiche "RowDescription"-Werte einander zugeordnet werden können.

Diagramme über ein Makro anpassen

Um dem Makro mitteilen zu können, welche Ansicht in dem jeweiligen Diagramm dargestellt werden soll, wird in den Formularen ein **verstecktes Steuerelement** eingebaut. Über den Formularnavigator wird das Formular, das als Auslöser des Diagramms genutzt werden soll, mit der rechten Maustaste angeklickt. Im Kontextmenü wird **Neu → Verstecktes Steuerelement** aufgerufen.

Das Element wird über das Makro mit dem Namen 'Chart' gesucht. Entsprechend wird das versteckte Steuerelement erst einmal umbenannt: **rechte Maustaste → Kontextmenü → Umbenennen**. Anschließend wird in dem Steuerelement über r**echte Maustaste → Kontextmenü → Eigenschaften → Allgemein → Zusatzinformation** der Name für die Ansicht vermerkt.

Jetzt ist nur noch notwendig, das folgende Makro mit den Ereignissen des Formulars **Beim Laden** bzw. **Nach der Datensatzaktion** zu verknüpfen.

```
001 SUB ChangeData(oEvent AS OBJECT)
002    DIM oDiag AS OBJECT
003    DIM oDatasource AS OBJECT
004    DIM oConnection AS OBJECT
005    DIM oSQL_Command AS OBJECT
006    DIM oResult AS OBJECT
007    DIM oForm AS OBJECT
008    DIM oHiddenControl AS OBJECT
009    DIM stSql AS STRING
010    DIM stRow AS STRING
011    DIM stType AS STRING
012    DIM i AS INTEGER
013    DIM k AS INTEGER
014    DIM x AS INTEGER
015    DIM n AS INTEGER
016    DIM aNewData(0)
017    DIM aNewRowDescription(0)
018    DIM aTmp() AS DOUBLE
019    DIM aNewColumnDescription()
020    DIM aType()
021    DIM arView()
022    DIM arDiag()
023    oForm = oEvent.Source
024    oHiddenControl = oForm.getByName("Chart")
025    arView = Split(oHiddenControl.Tag,",")
026    arDiag = Split(oHiddenControl.HiddenValue,",")
027    FOR n = LBound(arView()) TO UBound(arView())
028       stView = oHiddenControl.Tag
029       stSql = "SELECT * FROM """+arView(n)+""""
030       oDatasource = ThisComponent.Parent.CurrentController
031       IF NOT (oDatasource.isConnected()) THEN
032          oDatasource.connect()
033       END IF
034       oConnection = oDatasource.ActiveConnection()
035       oSQL_Command = oConnection.createStatement()
036       oResult = oSQL_Command.executeQuery(stSql)
037       i = 0
038       k = 0
039       x = 0
040       WHILE oResult.next
041          stRow = oResult.getString(1)
042          stType = oResult.getString(4)
043          IF aNewRowDescription(i) = stRow THEN
044             ReDim Preserve aNewColumnDescription(k)
```

```
045             ReDim Preserve aTmp(k)
046          ELSE
047             IF x > 0 THEN
048                i = i + 1
049             ELSE
050                x = 1
051             END IF
052             ReDim Preserve aNewRowDescription(i)
053             ReDim Preserve aNewData(i)
054             k = 0
055             ReDim Preserve aNewColumnDescription(k)
056             ReDim aTmp(k)
057          END IF
058          aNewRowDescription(i) = stRow
059          aNewColumnDescription(k) = oResult.getString(2)
060          aTmp(k) = oResult.getDouble(3)
061          aNewData(i) = aTmp()
062          k = k + 1
063       WEND
064       oDiag = thisComponent.EmbeddedObjects.getByName(arDiag(n))
065       oXCOEO = oDiag.ExtendedControlOverEmbeddedObject
066       oXCOEO.changeState(4)
067       IF stType <> "XY" THEN
068          oDiag.model.Data.setData(aNewData)
069       END IF
070       oDiag.model.DataProvider.setData(aNewData)
071       oDiag.model.DataProvider.setRowDescriptions(aNewRowDescription)
072       oDiag.model.DataProvider.setColumnDescriptions(aNewColumnDescription)
073       oDiag.Component.setmodified(true)
074       oDiag.Component.update()
075       oXCOEO.changeState(1)
076    NEXT
077 END SUB
```

Nach der Deklaration der Variablen wird zuerst aus den Zusatzinformationen (**Tag**) des versteckten Kontrollfeldes der Name für die Ansicht ausgelesen. Da hier mehrere Namen für mehrere Diagramme gespeichert werden können werden die Namen durch Komma getrennt und über **Split** in ein Array eingelesen. Gleiches gilt für die Namen der Diagramme, die im **HiddenValue** in der entsprechenden Reihenfolge eingetragen sind. Anschließend läuft eine Schleife über das erste Array ab, das ja genau so viele Elemente enthält wie das zweite Array. Der erforderliche SQL-Code zum Auslesen der gesamten Ansicht wird formuliert, die Verbindung zur Datenbank, falls erforderlich, hergestellt und der SQL-Code an die Datenbank weitergeleitet. In **oResult** wird das Ergebnis der Abfrage gespeichert.

Vor dem Start der Schleife zur Ermittlung des Inhaltes aus **oResult** werden noch einige Zahlenvariablen auf '0' gesetzt, die im Laufe der Schleife verändert werden sollen.

Innerhalb der Schleife werden die Inhalte der verschiedenen Spalten der Ansicht ausgelesen. Die Inhalte werden in unterschiedliche Arrays zur Weitergabe an den DataProvider abgespeichert. Hier werden wieder Daten (**Data**), Zeilenbeschriftungen (**RowDescriptions**) und Spaltenbeschriftungen (**ColumnDescriptions**) unterschieden.

Mit **oResult.getString(1)** wird das Ergebnis zum jeweiligen Datensatz aus der ersten Spalte als Text ausgelesen. Hier handelt es sich um Zeilenbeschriftungen, die eben einfach Text sind. Solange die Abfrage nacheinander gleiche Zeilenbeschriftungen liefert, werden alle weitere Inhalte dem gleichen Datenbestand zugeordnet. Um dies zu gewährleisten, erfolgt zuerst eine Abfrage, ob der entsprechende Eintrag von **stRow** bereits als letzter Eintrag von **aNewRowDescriptions**() vorhanden ist.

Ist dies nicht der Fall, wie z.B. direkt beim Einlesen des ersten Datensatzes, dann erfolgt das Vorgehen, das unter **ELSE** beschrieben ist. Nur wenn der Zähler **x** größer als '0' ist, wird der Zähler für **i** heraufgesetzt. Dies soll vermeiden, dass der erste Eintrag des zu erzeugenden Arrays später leer ist. Für alle späteren Eintritte in diese Schleife wird allerdings dann **x** auf '1'

gesetzt, so dass **i** heraufgesetzt wird und die Arrays mit einem weiteren Datensatz beschrieben werden können.

Redim Preserve sichert den bisherigen Inhalt eines Arrays und eröffnet gleichzeitig die Möglichkeit, einen zusätzlichen Eintrag ans Ende des Arrays anzufügen.

aNewData und **aNewRowDescriptions** werden immer zusammen abgespeichert. Deswegen haben die Arrays den gemeinsamen Zähler **i**. **aNewColumnDescriptions** sowie die in einem temporären Array **aTmp** gespeicherten Dateninhalte werden bei jedem neuen Durchgang durch die **WHILE**-**NEXT**–Schleife mit einem zusätzlichen Datensatz versehen. Auch sie haben deshalb einen gemeinsamen Zähler, hier **k**. Dieser Zähler wird bei jeder Änderung von **aNewRowDescriptions** neu mit '0' gestartet. Dabei wird das temporäre Array nicht gesichert, da es zwischenzeitig in dem Array **aNewData** abgespeichert wurde.

Die Inhalte für das Array **aTmp** werden immer als Dezimalzahlen aus der Abfrage ausgelesen. Daher der Eintrag **oResult.getDouble(3)**.

Damit ein Diagramm im Formular kontinuierlich geändert werden kann, muss es zuerst einmal aktiviert werden. Das Diagramm selbst ist dem Formular sonst völlig unbekannt. Das Diagramm wird über den Namen ausgesucht. Anschließend wird der Controller für das eingebettete Diagramm angesprochen.Zuerst wird mit '4' die UI aktiviert. Die möglichen Parameter sind unter **com.sun.star.embed.EmbedStates**: LOADED = 0, RUNNING = 1, ACTIVE = 2, INPLACE_ACTIVE = 3, UI_ACTIVE = 4.

Nachdem alle Daten ausgelesen wurden, werden diese in den **DataProvider** übertragen. Mit **oDiag.model.Data.setData(aNewData)** werden nicht nur die Daten neu geschrieben, sondern z.B. auch die Anzahl der Säulen in einem Spaltendiagramm aktualisiert. Dieser Eintrag führt allerdings bei einem XY-Diagramm dazu, dass hier die Grundstruktur des Diagramms zerstört und der erste Dateneintrag nicht als X-Achsenwert gesehen wird. Deshalb ist dieser Eintrag für XY-Diagramme ausgeschlossen.

Nach der Änderung wird der Status des Diagramms auf '1' gesetzt, da ansonsten das Diagramm die ganze Zeit im Bearbeitungsmodus erscheint.

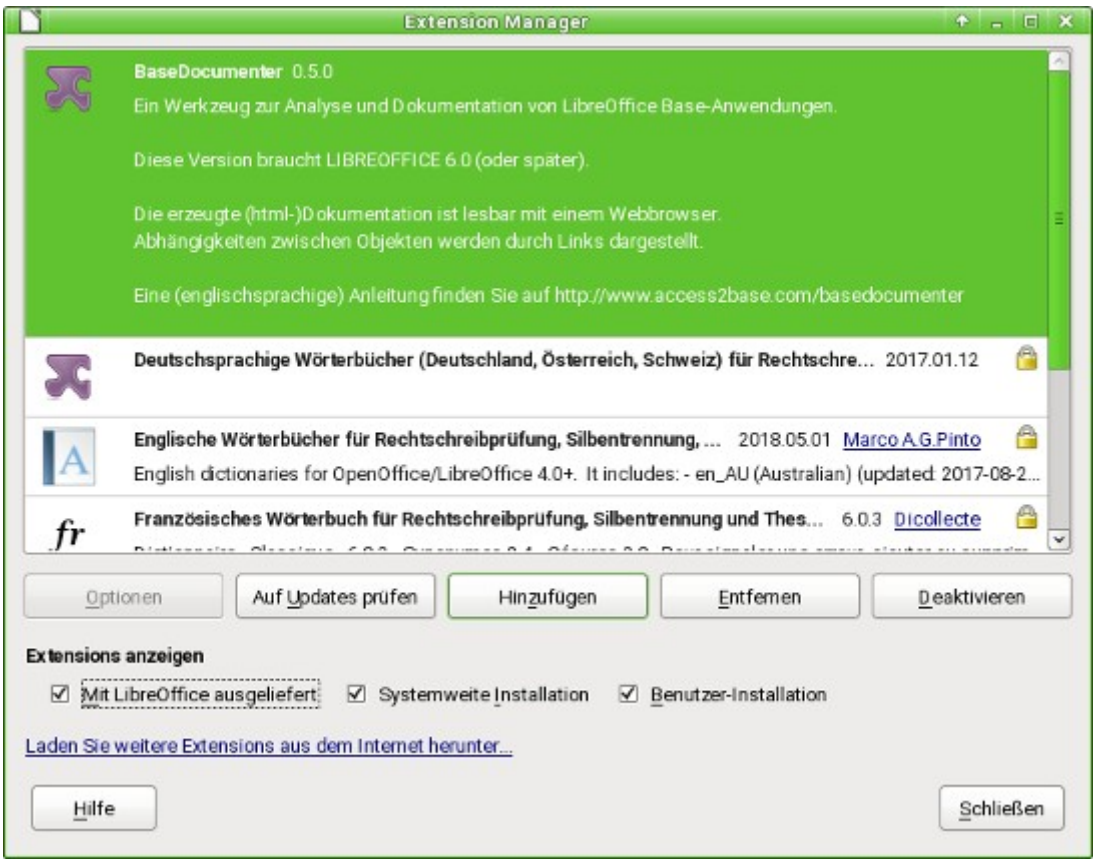
Übersicht über die Datenbank: BaseDocumenter – Extension

Wem geht es nicht oft so, dass irgendwann der Überblick über die ganzen Details einer Datenbank nützlich wären. Hier hilft die Erweiterung «BaseDocumenter»[26] von Jean-Pierre Ludure, die eine Übersicht in Form von HTML-Dateien erzeugt. Diese Erweiterung erscheint dann als zusätzliches Menü in jedem Base-Fenster.

Hinweis

Diese Erweiterung funktioniert zur Zeit (LO 7.2) nur mit der internen HSQLDB, nicht mit der internen ~~FIREBIRD~~ Datenbank.

26 Siehe: *http://www.access2base.com/basedocumenter/basedocumenter.html*

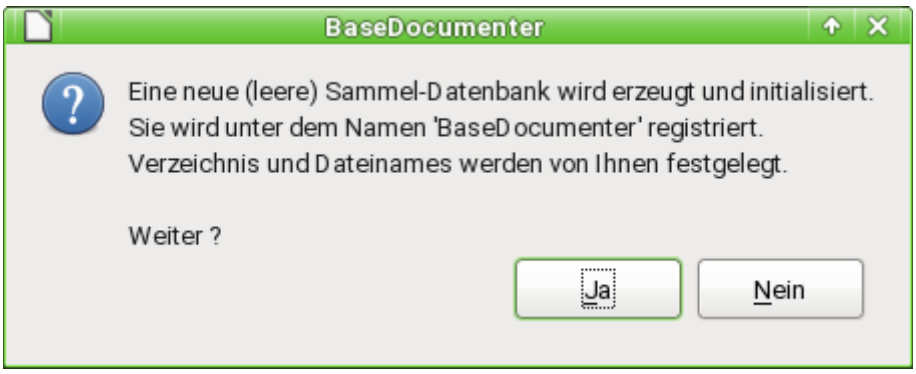


Zuerst wird über **Extras → Extension Manager** der BaseDocumenter hinzugefügt. Der BaseDocumenter benötigt mindestens eine LO-Version 6.0.

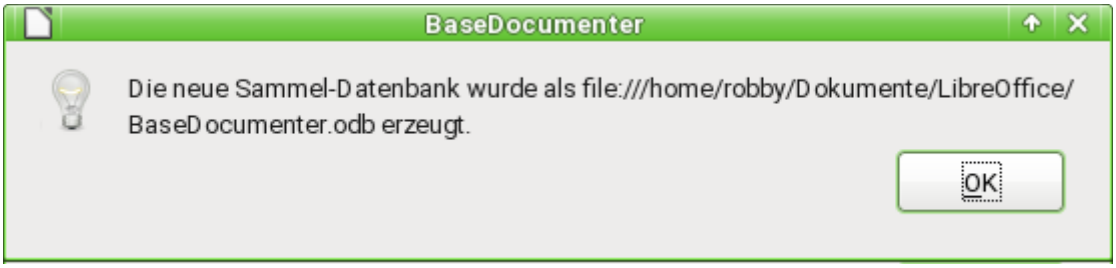
Die abzuspeichernden Informationen für die Datenbanken benötigen ein Verzeichnis, in dem sie abgelegt werden können. Dieses Verzeichnis muss zuerst erstellt werden.

In diesem Verzeichnis sollte außerdem css-Dateien («Cascading Style Sheets»), Javascripte, Logos und Vorlagen für die Darstellung der HTML-Dateien abgespeichert werden. Ein erstes Beispiel steht hier zum Download bereit: *http://www.access2base.com/basedocumenter/_download/Templates.zip* . Diese *.zip-Datei muss entpackt und anschließend komplett in das erstellte Verzeichnis kopiert werden.

BaseDocumenter → Neue Sammel-DB erstellt eine Datenbank, in der die notwendigen Informationen gespeichert werden. Diese Datenbank hat für den normalen Nutzer keine weitere Bedeutung. Sie braucht anschließend auch nicht mit **Öffne Sammel-DB** aufgesucht zu werden.

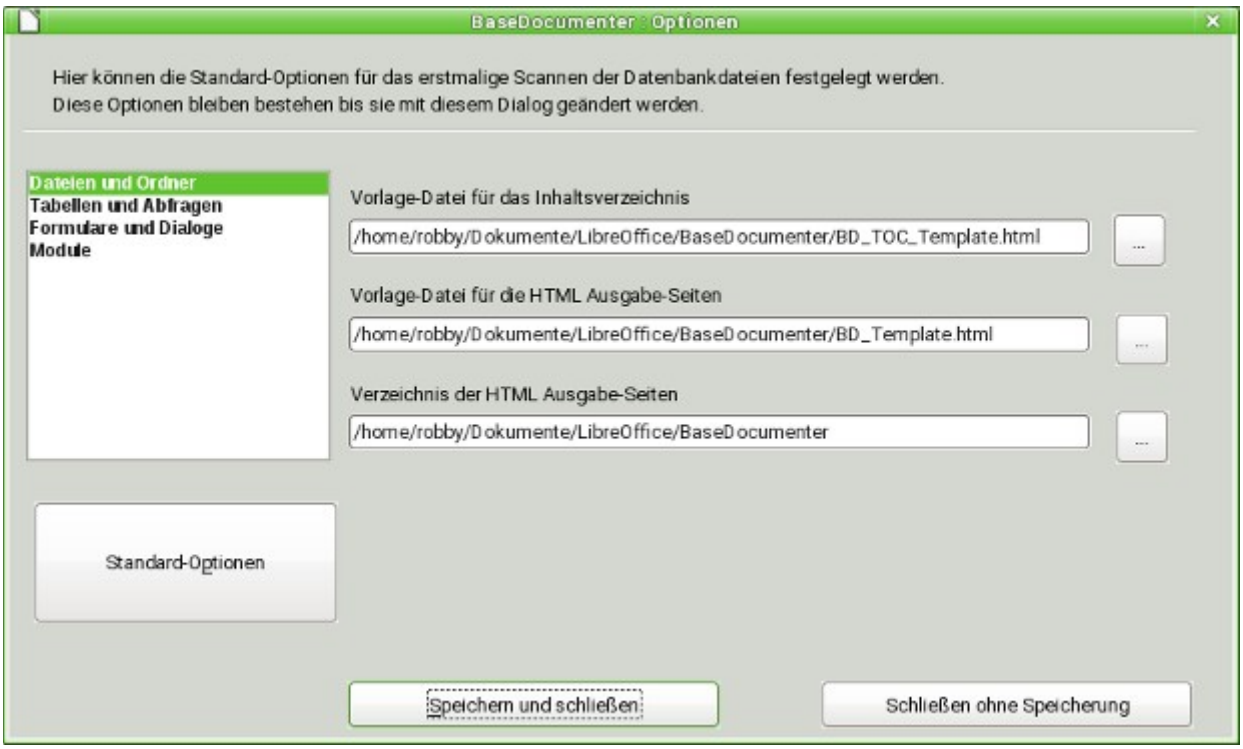


Die Sammeldatenbank wird erstellt und als «BaseDocumenter» in **Extras → Optionen → LibreOffice Base → Datenbanken** registriert.



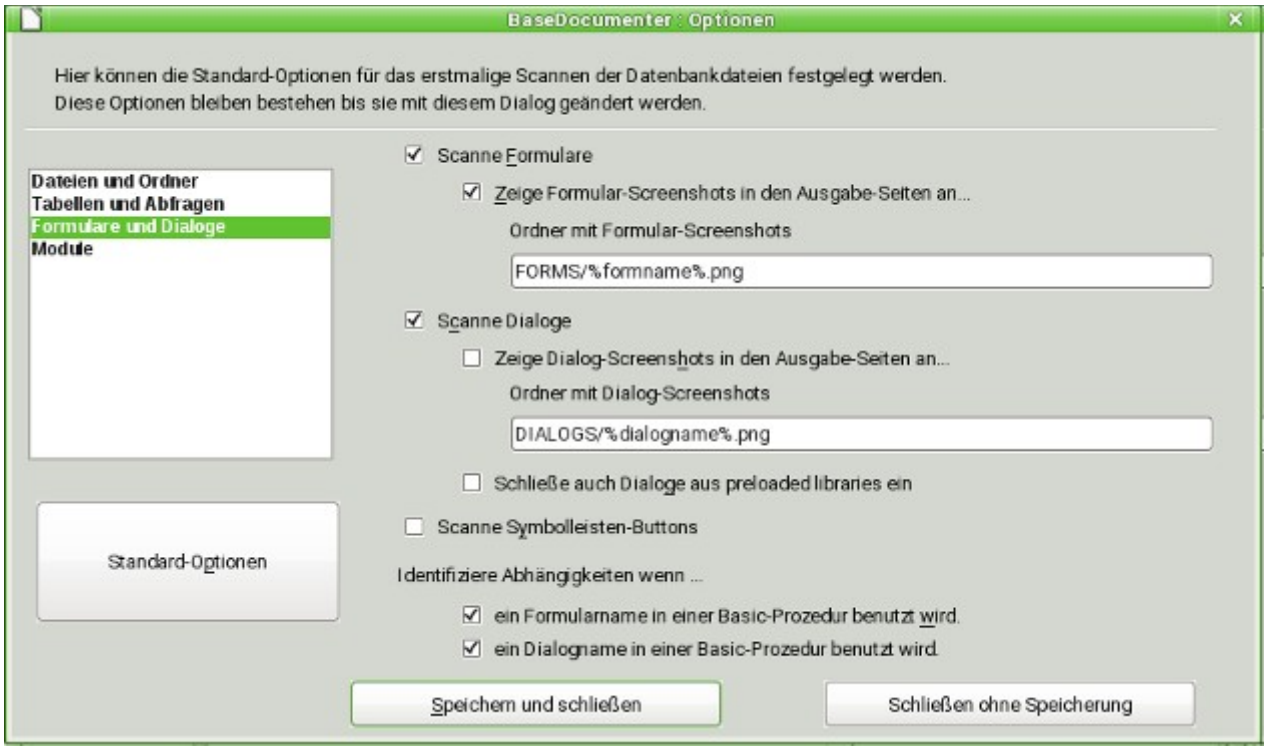
Die in dieser Sammeldatenbank erstellten Tabellen dürfen nicht geändert werden. Es könnten aber sehr wohl z.B. Abfragen hinzugefügt werden. Für den Normalgebrauch ist dies aber nicht notwendig.

BaseDocumenter → Optionen dient zuerst einmal zur Einstellung der Pfade für die gerade abgespeicherten Vorlagedateien und die abzuspeichernden Informationen der Datenbank. In diesem Dialog kann auch detailliert ausgewählt werden, welche Elemente in den HTML-Dateien dargestellt werden sollen. Hier ist es auch möglich, Screenshots von Formularen und eventuell erstellten Dialogen anfertigen zu lassen.



Die Pfade werden auf das vorher erstellte Verzeichnis und die darin befindlichen Vorlagen für das Inhaltsverzeichnis und der HTML-Ausgabe der einzelnen Dateien eingestellt.

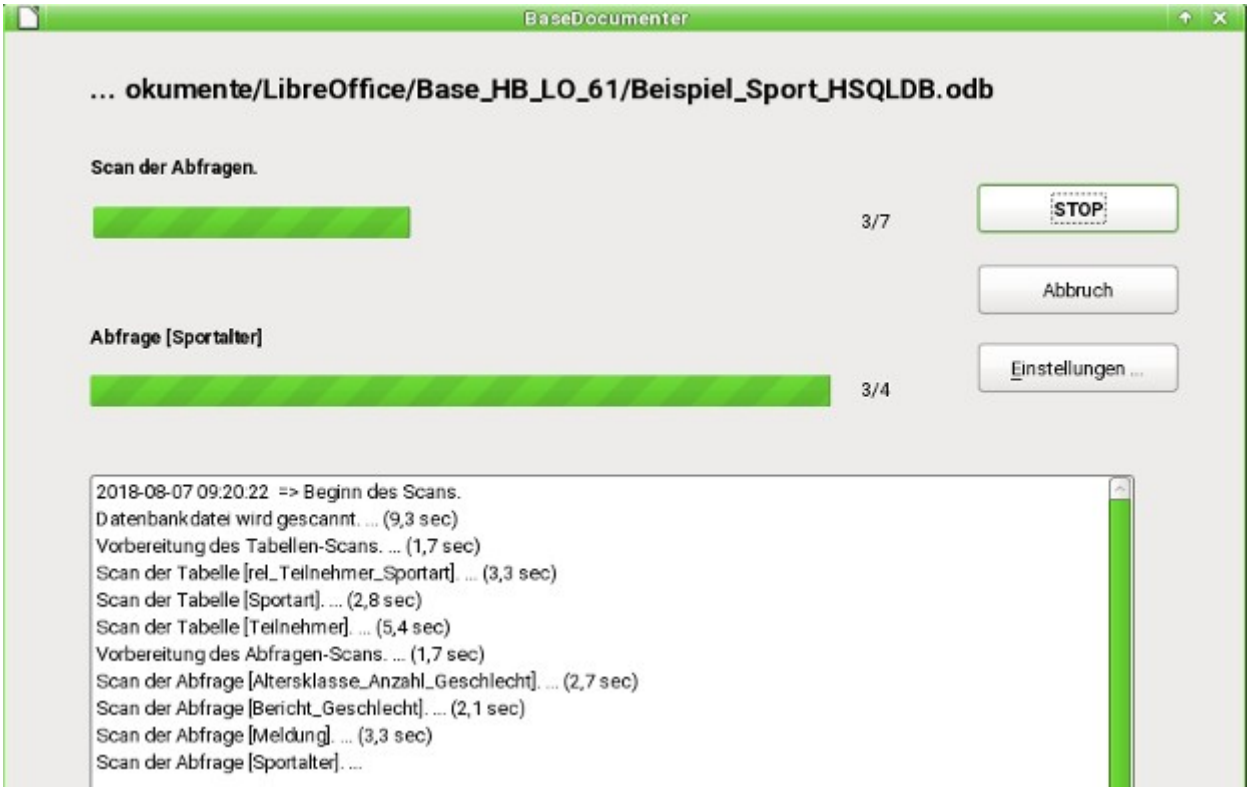
Ohne die Angabe eines Verzeichnisses für die Ausgabe-Seiten lässt sich ein Export nicht starten. Ohne die Vorlage-Dateien sind die erzeugten HTML-Dateien sehr unübersichtlich.



Bei den Formularen und Dialogen lässt sich die Einbindung von Screenshots einstellen. Die Screenshots selbst müssen allerdings durch den Nutzer angefertigt werden. Sie werden für Formulare in dem Unterordner **FORMS/«Formularname».png** abgelegt.

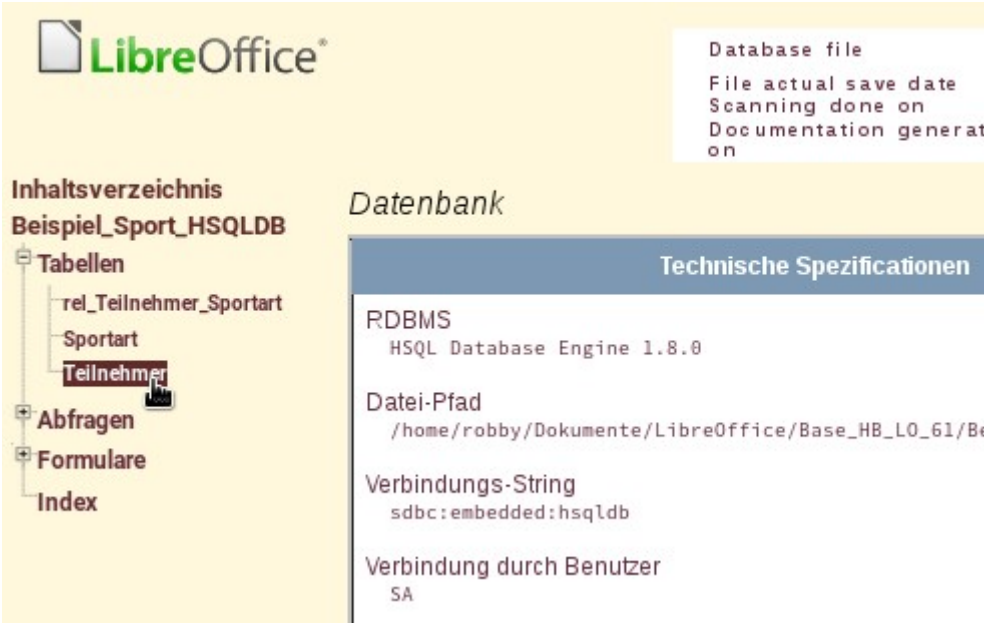
Wenn alle oben genannten Einstellungen erledigt sind kann die erste Datenbank dokumentiert werden. Der Aufruf erfolgt hier immer direkt in der geöffneten Datenbank, bei der für die Dokumentation die Ausführung von Makros unterbunden werden sollte.

Ein recht großer Dialog erscheint, von dem aus der Scan der Datei gestartet wird.



Nach dem Scan liegt in dem vorher erstellten Verzeichnis für die HTML-Dateien eine Datei «TOC.html». Diese Datei kann über den Browser aufgerufen werden.

Sind die Formatdateien installiert, so erscheint ein Inhaltsverzeichnis, das in dem obigen Bild erst einmal nur den Scan für die aktuelle Datei aufweist. Direkt über das Inhaltsverzeichnis sind hier u.a. die Tabelle (3), Abfragen (4) und Formulare(1) zu erreichen.



Wird der Name der Datenbank angeklickt, so erscheint zuerst eine Übersicht über die Datenbank mit technischen Spezifikationen usw. Daneben ist bei installierten Formatdateien ein Auswahlmenü zu sehen, mit dem z.B. durch die Tabellen navigiert werden kann.

Der BaseDocumenter zeigt sämtliche Beziehungen der Tabellen zu Abfragen und Formularen auf. Hier kann genau recherchiert werden, an welcher Stelle die Tabelle in der Datenbank benötigt wird. Das aktuelle Beispiel ist über *https://www.familiegrosskopf.de/robert/base_documenter/TOC.html* komplett einzusehen.

Codeschnipsel

Die Codeschnipsel erwachsen aus Anfragen innerhalb von Mailinglisten. Bestimmte Problemstellungen tauchen dort auf, deren Lösungen vielleicht gut innerhalb der eigenen Datenbankentwürfe genutzt werden können.

Aktuelles Alter ermitteln

Aus einem Datum soll mittels Abfrage das aktuelle Alter ermittelt werden. Siehe hierzu die Funktionen im Anhang zu diesem Base-Handbuch.

```
001 SELECT DATEDIFF('yy',"Geburtsdatum",CURDATE()) AS "Alter" FROM "Person"
```

Hinweis

Für Firebird muss der Code entsprechend angepasst werden. **CURDATE()** als Kurzform ist hier unbekannt. Es muss schon **CURRENT_DATE** sein. Und statt **'yy'** muss an dieser Stelle ein String ohne Hochkommata angegeben werden: **year**.
Auch die Kurzform **DAYOFYEAR** kennt Firebird nicht. Hier muss wieder **EXTRACT(YEARDAY FROM "Geburtsdatum")** usw. geschrieben werden.

Die Abfrage gibt das Alter als Jahresdifferenz aus. Das Alter eines Kindes, das am 31.12.2011 geboren ist, wird am 1.1.2012 mit 1 Jahr angegeben. Es muss also die Lage des Tages im Jahr berücksichtigt werden. Dies ist mit der Funktion **'DAYOFYEAR()'** ermittelbar. Mittels einer Funktion wird der Vergleich durchgeführt.

```
001 SELECT CASEWHEN
002 ( DAYOFYEAR("Geburtsdatum") > DAYOFYEAR(CURDATE()) ,
003 DATEDIFF ('yy',"Geburtsdatum",CURDATE())-1,
004 DATEDIFF ('yy',"Geburtsdatum",CURDATE()))
005 AS "Alter" FROM "Person"
```

Jetzt wird das aktuelle Alter in Jahren ausgegeben.

Über **'CASEWHEN'** könnte dann auch in einem weiteren Feld der Text **'Heute Geburtstag'** ausgegeben werden, wenn **DAYOFYEAR("Geburtsdatum") = DAYOFYEAR(CURDATE())**.

Spitzfindig könnte jetzt der Einwand kommen: «Wie steht es mit Schaltjahren?». Für Personen, die nach dem 28. Februar geboren wurden, kann es zu Abweichungen um einen Tag kommen. Für den Hausgebrauch nicht weiter schlimm, aber wo bleibt der Ehrgeiz, es möglichst doch genau zu machen?

Mit

```
001 SELECT CASEWHEN (
002 (MONTH("Geburtsdatum") > MONTH(CURDATE())) OR
003    ((MONTH("Geburtsdatum") = MONTH(CURDATE()))
004    AND (DAY("Geburtsdatum") > DAY(CURDATE()))) ,
005 DATEDIFF('yy',"Geburtsdatum",CURDATE())-1,
006 DATEDIFF('yy',"Geburtsdatum",CURDATE()))
007 AS "Alter" FROM "Person"
```

wird das Ziel erreicht. Solange der Monat des Geburtsdatums größer ist als der aktuelle Monat wird auf jeden Fall von der Jahresdifferenz 1 Jahr abgezogen. Ebenfalls 1 Jahr abgezogen wird, wenn zwar der Monat gleich ist, der Tag im Monat des Geburtsdatums aber größer ist als der Tag im aktuellen Monat. Leider ist diese Eingabe für die GUI nicht verständlich. Erst **'SQL-Kommando direkt ausführen'** lässt die Abfrage erfolgreich absetzen. So ist also unsere Abfrage nicht mehr editierbar. Die Abfrage soll aber weiter editierbar sein; also gilt es die GUI zu überlisten:

```
001 SELECT CASE
002 WHEN MONTH("Geburtsdatum") > MONTH(CURDATE())
003 THEN DATEDIFF('yy',"Geburtsdatum",CURDATE())-1
004 WHEN (MONTH("Geburtsdatum") = MONTH(CURDATE())
005    AND DAY("Geburtsdatum") > DAY(CURDATE()))
006 THEN DATEDIFF('yy',"Geburtsdatum",CURDATE())-1
007 ELSE DATEDIFF('yy',"Geburtsdatum",CURDATE())
008 END
009 AS "Alter" FROM "Person"
```

Auf diese Formulierung reagiert die GUI nicht mit einer Fehlermeldung. Das Alter wird jetzt auch in Schaltjahren genau ausgegeben und die Abfrage bleibt editierbar.

Geburtstage in den nächsten Tagen anzeigen

Mit Hilfe von einigen kleinen Berechnungskniffen lässt sich auch aus einer Tabelle z.B. auslesen, wer in den nächsten 8 Tagen Geburtstag hat.

```
001 SELECT *
002 FROM "Tabelle"
003 WHERE
004    DAYOFYEAR("Datum") BETWEEN DAYOFYEAR(CURDATE()) AND
005       DAYOFYEAR(CURDATE()) + 7
006    OR DAYOFYEAR("Datum") < 7 -
007       DAYOFYEAR(CAST(YEAR(CURDATE())||'-12-31' AS DATE)) +
008       DAYOFYEAR(CURDATE())
```

Die Abfrage zeigt alle Datensätze an, deren Datumseintrag zwischen dem momentanen Jahrestag und den folgenden 7 Tagen liegt.

Damit auch am Jahresende entsprechend 8 Tage angezeigt werden müssen die Anfangstage des Jahres grundsätzlich auch überprüft werden. Diese Überprüfung findet aber nur für die Jahrestage statt, die höchstens den Wert 7 abzüglich des Jahrestages des letzten Datumswertes des aktuellen Jahres (meist 365) zuzüglich des Jahrestages des aktuellen Datumswertes. Liegt das aktuelle Datum mehr als 7 Tage vom Jahresende entfernt, so ist der Gesamtwert also < 1. Kein Eintrag in der Tabelle hat so ein Datum. Diese Teilbedingung wird dann nicht erfüllt.

In der oben stehenden Formulierung werden bei Schaltjahren noch Unstimmigkeiten hervortreten, da sich dann ab dem 29.2. die Jahrestagzählung verschiebt. Der Code, um dieses zu berücksichtigen, gestaltet sich um einiges umfangreicher:

```
001 SELECT *
002 FROM "Tabelle"
003 WHERE
004       CASE
005          WHEN
006             DAYOFYEAR(CAST(YEAR("Datum")||'-12-31' AS DATE)) = 366
007             AND DAYOFYEAR("Datum") > 60 THEN DAYOFYEAR("Datum") - 1
008          ELSE
009             DAYOFYEAR("Datum")
010       END
011    BETWEEN
012       CASE
013          WHEN
014             DAYOFYEAR(CAST(YEAR(CURDATE())||'-12-31' AS DATE)) = 366
015             AND DAYOFYEAR(CURDATE()) > 60 THEN DAYOFYEAR(CURDATE()) - 1
016          ELSE
017             DAYOFYEAR(CURDATE())
018       END
019    AND
020       CASE
021          WHEN
022             DAYOFYEAR(CAST(YEAR(CURDATE())||'-12-31' AS DATE)) = 366
023             AND DAYOFYEAR(CURDATE()) > 60 THEN DAYOFYEAR(CURDATE()) + 6
024          ELSE
025             DAYOFYEAR(CURDATE()) + 7
026       END
027    OR DAYOFYEAR("Datum") < 7 -
028       DAYOFYEAR(CAST(YEAR(CURDATE())||'-12-31' AS DATE)) +
029       DAYOFYEAR(CURDATE())
```

Schaltjahre sind daran zu erkennen, dass das gesamte Jahr 366 und nicht 365 Tage hat. Dies wird für die entsprechenden Unterscheidungen genutzt.

Zum einen muss jeder Datumswert darauf überprüft werden, ob er in einem Schaltjahr liegt und außerdem für die korrekte Zählung der 60. Tag im Jahr ist (31 Tage im Januar und 29 Tage im Februar). Für diesen Fall werden alle darauffolgende DAYOFYEAR – Werte für das Datum um 1 herabgestuft. Dann trifft ein 1.3. eines Schaltjahres wieder exakt auf einen 1.3. eines normalen Jahres.

Zum anderen muss auch beim aktuellen Jahr (CURDATE()) überprüft werden, ob es sich um ein Schaltjahr handelt. Auch hier wird entsprechend die Tageszahl um 1 herabgestuft.

Auch eine Anzeige von Terminen für die nächsten 8 Tage ist so noch nicht einwandfrei möglich, da das Jahr in der Abfrage noch ausgeklammert ist. Dies wäre aber über eine einfache zusätzliche Bedingung **`YEAR("Datum") = YEAR(CURDATE())`** für das aktuelle bzw. **`YEAR("Datum") = YEAR(CURDATE()) + 1`** für das zukünftige Jahr zu erfüllen.

Tage zu Datumswerten addieren

Bei der Ausleihe von Medien möchte vielleicht der Entleiher genau wissen, an welchem Tag denn die Rückgabe des Mediums erfolgen soll. Leider bietet die interne HSQLDB dazu nicht die Funktion **`DATEADD()`** an, wie dies in vielen der externen Datenbanken und auch bei Firebird der Fall ist.

Hinweis

Für Firebird ist diese Vorgehensweise zum Addieren von Tagen zu Datumswerten überflüssig. Mit

```
001  SELECT
002     "Leih_Datum",
003     DATEADD(DAY, 14, "LeihDatum") AS "Rueckgabedatum"
004  FROM "Ausleihe"
```

werden zum Ausleihdatum 14 Tage addiert. Der Befehl funktioniert allerdings nur bei direkter Ausführung des SQL-Kommandos.

Für die interne HSQLDB lässt sich für einen begrenzten Zeitraum der folgende Umweg nutzen:

Zuerst wird eine Tabelle mit dem Datumsverlauf über den gewünschten Zeitraum erstellt. Hierzu wird einfach Calc geöffnet, in das Feld A1 die Bezeichnung "ID" und in das Feld B1 die Bezeichnung "Datum" geschrieben. In Feld A2 wird eine 1 eingetragen, in Feld B2 das gewünschte Startdatum, z.B. 1.1.15. A2 und B2 werden markiert und weiter nach unten gezogen, so dass daraus eine fortlaufende Nummer und ein fortlaufendes Datum entstehen.

Anschließend wird diese Tabelle mit den Spaltenüberschriften zusammen markiert und in Base eingefügt: **rechte Maustaste → Einfügen → Tabellenname → Datum**. Bei den Optionen wird **Definition und Daten** sowie **Erste Zeile als Spaltennamen verwenden** angeklickt. Alle Spalten werden übernommen. Anschließend ist nur noch darauf zu achten, dass bei den Typformatierungen das Feld "ID" dem Typ **`Integer [INTEGER]`** zugeordnet werden soll und dem Datumsfeld auch der Typ **`Datum [DATE]`** zugeordnet wird. Ein Primärschlüssel ist nicht erforderlich, da die Daten später nicht geändert werden sollen. Dadurch, dass eben der Primärschlüssel nicht definiert wird, wird die Tabelle gleichzeitig schreibgeschützt.

Tipp

Auch über Abfragetechniken ist die Erstellung so einer Übersicht möglich. Diese Übersicht kann, wenn eine Filtertabelle genutzt wird, im Startdatum und ihrem Umfang an Datumswerten sogar gesteuert werden:

```
001  SELECT DISTINCT CAST
002     ( "Y"."Nr" + (SELECT "Jahr" FROM "Filter" WHERE "ID" =
003        True) - 1 || '-' ||
004     CASEWHEN( "M"."Nr" < 10, '0' || "M"."Nr", '' || "M"."Nr" )
005     || '-' ||
006     CASEWHEN( "D"."Nr" < 10, '0' || "D"."Nr", '' || "D"."Nr" )
007     AS DATE ) AS "Datum"
008  FROM "NrBis31" AS "D", "NrBis31" AS "M", "NrBis31" AS "Y"
009  WHERE "Y"."Nr" <= (SELECT "Jahre" FROM "Filter" WHERE "ID" =
010     True)
011  AND "M"."Nr" <= 12 AND "D"."Nr" <= 31
```

Diese Ansicht greift auf eine Tabelle zu, die lediglich die Nummern von 1 bis 31 erfasst und schreibgeschützt ist. In einer weiteren Filtertabelle wird das Startjahr und der Umfang in Jahren festgelegt, den die Ansicht umfassen soll. Das Datum wird dadurch zusammengestellt, dass ein Datumsausdruck aus Jahr, Monat und Tag als Text erstellt und anschließend in ein Datum umgewandelt wird. Die HSQLDB akzeptiert alle Tage bis zum 31. eines Monats, auch z.B. den 31.02.2015. Aus dem 31.02.2015 wird allerdings entsprechend der 3.03.2015 wiedergegeben. Deshalb muss bei der Erstellung der Ansicht über **DISTINCT** ausgeschlossen werden, dass Datumswerte doppelt vorkommen. Hierauf greift die folgende Ansicht zu:

```
001  SELECT "a"."Datum",
002     (SELECT COUNT(*) FROM "Ansicht_Datum" WHERE "Datum" <=
003        "a"."Datum") AS "lfdNr"
004  FROM "Ansicht_Datum" AS "a"
```

Über eine *Zeilennummerierung* wird den Datumswerten eine Nummer hinzugefügt. Da aus Ansichten sowieso keine Daten gelöscht werden können, muss hier natürlich nicht extra ein Schreibschutz bemüht werden.

Mit einer Abfrage kann jetzt zu einem bestimmten Datum ermittelt werden, wie z.B. das Datum in 14 Tagen lauten wird:

```
001 SELECT "a"."Leih_Datum",
002    (SELECT "Datum" FROM "Datum" WHERE "ID" =
003       (SELECT "ID" FROM "Datum" WHERE "Datum" = "a"."Leih_Datum")+14)
004       AS "Rueckgabedatum"
005 FROM "Ausleihe" AS "a"
```

In der ersten Spalte wird das Ausleihdatum ermittelt. Auf diese Spalte greift eine korrelierende Unterabfrage zu, die wieder in zwei Abfragen geschachtelt ist. Es wird über **SELECT "ID" FROM "Datum"** der Wert des Feldes "ID" ermittelt, der dem Ausleihdatum entspricht. Zu diesem Wert wird 14 addiert. Dieser Wert wird dem Feld "ID" bei der äußeren Unterabfrage zugewiesen. Zu dieser neuen "ID" wird nachgesehen, welches Datum im Datumsfeld des Datensatzes steht.

Leider wird bei der Anzeige in der Abfrage der Datumstyp nicht automatisch erkannt, so dass hier entsprechend formatiert werden muss. In einem Formular ist die entsprechende Anzeige aber dauerhaft speicherbar, so dass auch bei jeder Abfrage entsprechend ein Datumswert ausgegeben wird.

Eine Direktvariante zur Ermittlung des Datumswertes ist sogar auf kürzerem Wege möglich. Hierbei wird der Startwert genutzt, ab dem die interne Tageszählung von Base beginnt:

```
001 SELECT "Leih_Datum",
002    DATEDIFF( 'dd', '1899-12-30', "Leih_Datum" ) + 14 AS "Rueckgabedatum"
003 FROM "Tabelle"
```

Der ermittelte Zahlenwert kann im Formular als Datum über ein formatiertes Feld dargestellt werden. Es ist allerdings nur mit großem Aufwand möglich, ihn einfach für weiteren Abfragecode in ein SQL-Datum zu übertragen.

Zeiten zu Zeitstempeln addieren

In MySQL gibt es die Funktion **TIMESTAMPADD**(). Eine vergleichbare Funktion existiert in der HSQLDB nicht. Aber auch hier kann über den internen Zahlenwert, den so ein Zeitstempel einnimmt, mittels eines formatierten Feldes im Formular auch die Addition oder Subtraktion einer Zeit dargestellt werden.

Hinweis

Für Firebird ist diese Vorgehensweise zum Addieren von Zeiten zu Zeitstempeln überflüssig. Mit

```
001  SELECT "Zeitstempel",
002     DATEADD( MINUTE, 14, "Zeitstempel" )
003  FROM "Tabelle"
```

werden zum Zeitstempel 14 Minuten addiert. Der Befehl funktioniert allerdings nur bei direkter Ausführung des SQL-Kommandos.

Im Gegensatz zur Addition von Tagen zu einem Datum tritt bei den Zeiten allerdings ein Problem auf, das anfangs vielleicht gar nicht auffällt:

```
001 SELECT "DatumZeit"
002    DATEDIFF( 'ss', '1899-12-30', "DatumZeit" ) / 86400.0000000000 + 36/24
003       AS "DatumZeit+36Stunden"
004 FROM "Tabelle"
```

Für die neue Zeitberechnung wird der Unterschied zur Startzeit '0' des Systems genommen. Das ist, wie bei der Datumsberechnung bereits angewandt, der Datumsstempel vom 30.12.1899.

Hinweis

Das Nulldatum am 30.12.1899 ist vermutlich deshalb entstanden, weil das Jahr 1900 im Gegensatz zur üblichen 4-Jahres-Rechnung kein Schaltjahr war. So ist der Tag '1' der internen Rechnung auf den 31.12.1899 vorverlegt worden und eben nicht der 1.1.1900.

Der Unterschied wird hier allerdings in Sekunden ausgedrückt. Die interne Zahl rechnet allerdings die Tage als Stellen vor dem Komma, die Stunden, Minuten und Sekunden als Stellen nach dem Komma. Da in einen Tag 60*60*24 Sekunden passen, muss die entsprechend ermittelte Sekundenzahl durch 86400 geteilt werden, um schließlich vor dem Komma mit den entsprechenden Tagen und hinter dem Komma mit den Bruchteilen rechnen zu können. Damit die interne Hsqldb überhaupt Nachkommastellen bei dieser Berechnung ausgibt, müssen diese auch in der Rechnung vorkommen. Statt durch 86400 zu teilen, wurde deshalb durch 86400,0000000000 geteilt. Dezimalstellen erscheinen in der Abfrage durch einen Punkt getrennt. Das Ergebnis hat letztlich also 10 Dezimalstellen hinter dem Komma.

Zu diesem Ergebnis wird die Stundenzahl als Bruchteil eines Tages hinzugezählt. Die berechnete Ziffer lässt sich bei entsprechender Formatierung in der Anfrage darstellen. Dort wird die Formatierung aber leider nicht gespeichert. Innerhalb eines Formulars mit formatierbarem Feld oder innerhalb eines Berichts kann sie aber dauerhaft mit entsprechender Formatierung übernommen werden.

Sollen Minuten oder Sekunden addiert werden, so ist hier auch immer darauf zu achten, dass die Angaben der Minuten und Sekunden als Tagesbruchteile angegeben werden.

Liegt das Datum in den Monaten November, Dezember, Januar usw., dann fällt bei der Rechnung erst einmal nichts auf. Die Darstellung ist stimmig, zum Zeitstempel von 20.01.2015 13:00:00 eine Zeit von 36 Stunden addiert ergibt den neuen (nur dargestellten) Stempel 22.01.2015 01:00:00. Anders verhält es sich bei 20.04.2015 13:00:00. Da wird anschließend der 22.04.2015 00:00:00 ausgegeben. Das liegt an der Sommerzeit, die der Berechnung hier in die Quere kommt. Die gerade bei der Zeitumstellung «verlorene» oder

«gewonnene» Stunde lässt sich bei einem Übergang nicht berücksichtigen. Innerhalb einer Zeitzone kann allerdings auf verschiedene Weise eine entsprechende Berechnung mit «korrektem» Ergebnis durchgeführt werden. Hier die dafür einfachere Variante:

```
001 SELECT "DatumZeit"
002    DATEDIFF( 'dd', '1899-12-30', "DatumZeit" ) +
003       HOUR( "DatumZeit" ) / 24.0000000000 +
004       MINUTE( "DatumZeit" ) / 1440.0000000000 +
005       SECOND( "DatumZeit" ) / 86400.0000000000 +
006       36/24
007          AS "DatumZeit+36Stunden"
008 FROM "Tabelle"
```

Statt die Stunden, Minuten und Sekunden aus dem Ursprungsdatum zu beziehen, werden sie hier im Verhältnis zum aktuellen Datum dargestellt. Am 20.05.2015 steht die Uhr auf 13:00 Uhr, würde aber ohne die Sommerzeit 12:00 Uhr anzeigen. Die Funktion **HOUR** berücksichtigt die Sommerzeit und gibt 13 Stunden als Stundenanteil aus. Damit kann dann der Stundenanteil korrekt zum Tagesanteil addiert werden. Auf gleiche Art und Weise geschieht dies mit Hilfe des Minutenanteils und des Sekundenanteils. Anschließend werden die zu addierenden Stunden wieder als Tagesbruchteil addiert und das Ganze mit Hilfe der Zellenformatierung als berechneter Zeitstempel ausgegeben.

Zwei Dinge sollten aber bei diesen Berechnungen nie aus den Augen verloren werden:

1. Bei Übergängen von Winterzeit zu Sommerzeit lassen sich die Stundenwerte nicht korrekt berechnen. Hier könnte mit Hilfe einer Zusatztabelle nachgeholfen werden, die Beginn und Ende der Sommerzeiten aufnimmt und dann mit einer Stundenkorrektur einspringt. Ein etwas aufwändiges Verfahren.
2. Die Ausgabe der Zeitangabe ist nur über formatierbare Felder zu erreichen. Das Ergebnis ist ein Dezimalwert, kein Zeitstempelwert, der z.B. wiederum in einer Datenbank direkt gespeichert werden könnte. Hier müsste entweder innerhalb des Formular kopiert werden oder über aufwändige Abfragen aus dem Dezimalwert zu einem Zeitstempelwert portiert werden. Der Knackpunkt bei dieser Portierung ist der Datumswert, da es eben Schaltjahre und Monate mit unterschiedlicher Tagesanzahl gibt.

Laufenden Kontostand nach Kategorien ermitteln

Statt eines Haushaltsbuches wird eine Datenbank im PC die leidigen Aufsummierungen von Ausgaben für Lebensmittel, Kleidung, Mobilität usw. erleichtern. Ein Großteil dieser Angaben sollte natürlich auf Anhieb in der Datenbank sichtbar sein. Dabei wird in dem Beispiel davon ausgegangen, dass Einnahmen und Ausgaben in einem Feld "Betrag" mit Vorzeichen abgespeichert werden. Prinzipiell lässt sich das Ganze natürlich auf getrennte Felder und eine Summierung hierüber erweitern.

```
001 SELECT "ID", "Betrag",
002    ( SELECT SUM( "Betrag" ) FROM "Kasse" WHERE "ID" <= "a"."ID" )
          AS "Saldo"
003 FROM "Kasse" AS "a" ORDER BY "ID" ASC
```

Mit dieser Abfrage wird bei jedem neuen Datensatz direkt ausgerechnet, welcher Kontostand jetzt erreicht wurde. Dabei bleibt die Abfrage editierbar, da das Feld "Saldo" durch eine korrelierende Unterabfrage erstellt wurde. Die Abfrage gibt den Kontostand in Abhängigkeit von dem automatisch erzeugten Primärschlüssel "ID" an. Kontostände werden aber eigentlich täglich ermittelt. Es muss also eine Datumsabfrage her.

```
001 SELECT "ID", "Datum", "Betrag",
002    ( SELECT SUM( "Betrag" ) FROM "Kasse" WHERE "Datum" <= "a"."Datum"
003       ) AS "Saldo"
004 FROM "Kasse" AS "a" ORDER BY "Datum", "ID" ASC
```

Die Ausgabe erfolgt jetzt nach dem Datum sortiert und nach dem Datum summiert. Bleibt noch die Kategorie, nach der entsprechende Salden für die einzelnen Kategorien dargestellt werden sollen.

```
001 SELECT "ID", "Datum", "Betrag", "Konto_ID",
002    ( SELECT "Konto" FROM "Konto" WHERE "ID" = "a"."Konto_ID" )
003       AS "Kontoname",
004    ( SELECT SUM( "Betrag" ) FROM "Kasse" WHERE "Datum" <= "a"."Datum"
005       AND "Konto_ID" = "a"."Konto_ID" ) AS "Saldo",
006    ( SELECT SUM( "Betrag" ) FROM "Kasse" WHERE "Datum" <= "a"."Datum"
007       ) AS "Saldo_gesamt"
008 FROM "Kasse" AS "a" ORDER BY "Datum", "ID" ASC
```

Hiermit wird eine editierbare Abfrage erzeugt, in der neben den Eingabefeldern (Datum, Betrag, Konto_ID) der Kontoname, der jeweilige Kontostand und der Saldo insgesamt erscheinen. Da sich die korrelierenden Unterabfragen teilweise auch auf vorhergehende Eingaben stützen ("Datum" <= "a"."Datum") werden nur Neueingaben reibungslos dargestellt. Änderungen eines vorhergehenden Datensatzes machen sich zuerst einmal nur in diesem Datensatz bemerkbar. Die Abfrage muss aktualisiert werden, damit auch die davon abhängigen späteren Berechnungen neu durchgeführt werden.

Zeilennummerierung

Automatisch hoch zählende Felder sind etwas feines. Nur sagen sie nicht unbedingt etwas darüber aus, wie viele Datensätze denn nun in der Datenbank oder dem Abfrageergebnis wirklich vorhanden sind. Häufig werden Datensätze gelöscht und manch ein User versucht verzweifelt dahinter zu kommen, welche Nummer denn nun nicht mehr vorhanden ist, damit die laufenden Nummern wieder stimmen.

```
001 SELECT "ID",
002    ( SELECT COUNT( "ID" ) FROM "Tabelle" WHERE "ID" <= "a"."ID" )
003       AS "lfdNr."
004 FROM "Tabelle" AS "a"
```

Das Feld "ID" wird ausgelesen, im zweiten Feld wird durch eine korrelierende Unterabfrage festgestellt, wie viele Feldinhalte von "ID" kleiner oder gleich dem aktuellen Feldinhalt sind. Daraus wird dann die laufende Zeilennummer erstellt.

Hinweis

Mit Firebird ist es möglich, hier eine eingebaute Funktion zu nutzen.

```
001 SELECT "ID",
002    ROW_NUMBER() OVER (ORDER BY "ID") AS "lfdNr."
003 FROM "Tabelle" AS "a"
```

Wird einer Abfrage eine Bedingung hinzugefügt oder beruht eine Abfrage aus einer Verbindung von mehreren Tabellen, so ist diese Bedingung bzw. die Verbindung von mehreren Tabellen auch in der korrelierenden Unterabfrage hinzuzufügen. Andernfalls stimmt die Zählung nicht:

```
001 SELECT "Name",
002    ( SELECT COUNT( "ID" ) FROM "Name" WHERE "GebDat" > '1995-12-31'
          AND "ID" <= "a"."ID" ) AS "lfdNr."
003 FROM "Name" AS "a" WHERE "GebDat" > '1995-12-31'
```

Die gesamte Bedingung der äußeren Abfrage muss in der korrelierenden Unterabfrage wiederholt werden. Hinzu kommt in der korrelierenden Unterabfrage noch die Bedingung, mit der sich die korrelierende Unterabfrage auf den aktuellen Datensatz bezieht.

Auch eine Nummerierung für entsprechende Gruppierungen ist möglich:

```
001 SELECT "ID",
002    ( SELECT COUNT( "ID" ) FROM "Tabelle" WHERE "ID" <= "a"."ID"
```

```
          AND "RechnungsID" = "a"."RechnungsID" ) AS "lfdNr."
003 FROM "Tabelle" AS "a"
```

Hier gibt es in einer Tabelle verschiedene Rechnungsnummern ("RechnungsID"). Für jede Rechnungsnummer wird separat die Anzahl der Felder "ID" aufsteigend nach der Sortierung des Feldes "ID" wiedergegeben. Das erzeugt für jede Rechnung die Nummerierung von 1 an aufwärts.

Soll die aktuelle Sortierung der Abfrage mit der Zeilennummer übereinstimmen, so ist die Art der Sortierung entsprechend abzubilden. Dabei muss die Sortierung allerdings einen eindeutigen Wert ergeben. Sonst liegen 2 Nummern auf dem gleichen Wert.

```
001 SELECT "Name",
002    ( SELECT COUNT( "ID" ) FROM "Tabelle" WHERE "Name" <= "a"."Name" )
003       AS "lfdNr."
004 FROM "Tabelle" AS "a"
005 ORDER BY "Name"
```

Nur wenn ein eindeutigen Index für das Feld "Name" definiert wurde, ist hier eine eindeutige Sortierung und auch eine klare Nummerierung zu erwarten. Tauchen aber zwei gleiche Namen auf, so erhalten beide die gleiche Nummer.

```
001 SELECT "Name",
002    ( SELECT COUNT( "ID" ) FROM "Tabelle"
003       WHERE "Name"||"ID" <= "a"."Name"||"a"."ID" ) AS "lfdNr."
004 FROM "Tabelle" AS "a"
005 ORDER BY "Name"||"ID"
```

Hier wird der Inhalt des Feldes "Name" mit dem Inhalt des Feldes "ID" zusammengefügt. Die Sortierung ist jetzt eindeutig, die laufende Nummer entsprechend auch. Bei genauerer Betrachtung fällt allerdings auf, dass die Sortierung der "ID" natürlich jetzt nicht der Zahlensortierung folgt. Stattdessen wird z.B. die Zahl '9' nach der Zahl '10' einsortiert, da 9 größer als 1 ist. Wer dem noch abhelfen will muss entsprechend mit Leerstellen oder Nullen auffüllen:

```
001 SELECT "Name",
002    ( SELECT COUNT( "ID" ) FROM "Tabelle"
003       WHERE "Name"||RIGHT('000000000'||"ID",10) <= "a"."Name"||
004       RIGHT('000000000'||"a"."ID",10) ) AS "lfdNr."
005 FROM "Tabelle" AS "a"
006 ORDER BY "Name"||RIGHT('000000000'||"ID",10)
```

Das Feld "ID" ist in diesem Beispiel ein INTEGER-Feld. INTEGER-Zahlen haben maximal 10 Stellen. Der Wert des Feldes wird mit 9 führenden Nullen aufgefüllt. Anschließend werden die 10 rechts stehenden Zeichen für die weitere Verarbeitung genutzt. '0000000009' ist nun kleiner als '0000000010'. Die Sortierung ist eindeutig und erfolgt bei gleichem Namen schließlich in gewohnter Reihenfolge der Schlüsselnummerierung.

Sollen Zeilen für eine Abfrage nummeriert werden, die sich nicht nur auf eine Tabelle bezieht oder mit einer zusätzlichen Bedingung versehen ist, so muss in der korrelierenden Unterabfrage die komplette Bedingung mit allen betroffenen Tabellen enthalten sein. Nur dann kann die Unterabfrage die gleichen Daten verarbeiten wie die äußere Abfrage.

Gleiche Werte können natürlich genutzt werden, wenn z.B. die Platzierung bei einem Wettkampf wiedergegeben werden soll, da hier gleiche Wettkampfergebnisse auch zu einer gleichen Platzierung führen. Damit die Platzierung allerdings so wiedergegeben wird, dass bei einer gleichen Platzierung der nachfolgende Wert ausgelassen wird, ist die Abfrage etwas anders zu konstruieren:

```
001 SELECT "ID",
002    ( SELECT COUNT( "ID" ) + 1 FROM "Tabelle" WHERE "Zeit" < "a"."Zeit" )
003       AS "Platz"
004 FROM "Tabelle" AS "a"
```

Es werden alle Einträge ausgewertet, die in dem Feld "Zeit" einen kleineren Eintrag haben. Damit werden alle Sportler erfasst, die vor dem aktuellen Sportler ins Ziel gekommen sind. Zu diesem Wert wird der Wert 1 addiert. Damit ist der Platz des aktuellen Sportlers bestimmt. Ist

dieser zeitgleich mit einem anderen Sportler, so ist auch der Platz gleich. Damit sind Platzierungen wie 1. Platz, 2. Platz, 2. Platz, 4. Platz usw. möglich.

Schwieriger wird es, wenn neben der Platzierung auch eine Zeilennummerierung erfolgen soll. Dies kann z.B. sinnvoll sein, um mehrere Datensätze in einer Zeile zusammen zu fassen.

```
001 SELECT "ID",
002    ( SELECT COUNT( "ID" ) + 1 FROM "Tabelle" WHERE "Zeit" < "a"."Zeit" )
003       AS "Platz",
004    CASE WHEN
005       ( SELECT COUNT( "ID" ) + 1 FROM "Tabelle"
006       WHERE "Zeit" = "a"."Zeit" ) = 1
007    THEN
008       ( SELECT COUNT( "ID" ) + 1 FROM "Tabelle"
009       WHERE "Zeit" < "a"."Zeit" )
010    ELSE (SELECT
011       (SELECT COUNT( "ID" ) + 1 FROM "Tabelle" WHERE "Zeit" < "a"."Zeit")
012       + COUNT( "ID" ) FROM "Tabelle"
013       WHERE "Zeit" = "a"."Zeit" "ID" < "a"."ID" )
       END AS "Zeilennummer"
014 FROM "Tabelle" AS "a"
```

Die zweite Spalte gibt weiterhin die Platzierung wieder. In der 3. Spalte wird zuerst nachgefragt, ob auch wirklich nur eine Person mit der gleichen Zeit durchs Ziel gekommen ist. Wenn dies erfüllt ist, wird die Platzierung auf jeden Fall direkt als Zeilennummer übernommen. Wenn dies nicht erfüllt ist, wird zu der Platzierung ein weiterer Wert addiert. Bei gleicher Zeit (**`"Zeit" = "a"."Zeit"`**) wird dann mindestens 1 addiert, wenn es eine weitere Person mit dem Primärschlüssel ID gibt, deren Primärschlüssel kleiner ist als der aktuelle Primärschlüssel des aktuellen Datensatzes (**`"ID" < "a"."ID"`**). Diese Abfrage gibt also solange identische Werte zur Platzierung heraus, wie keine zweite Person mit der gleichen Zeit existiert. Existiert eine zweite Person mit der gleichen Zeit, so wird nach der ID entschieden, welche Person die geringere Zeilennummer enthält.

Diese Zeilensortierung entspricht übrigens der, die die Datenbanken anwenden. Wird z.B. eine Reihe Datensätze nach dem Namen sortiert, so erfolgt die Sortierung bei gleichen Datensätzen nicht nach dem Zufallsprinzip, sondern aufsteigend nach dem Primärschlüssel, der ja schließlich eindeutig ist. Es lässt sich auf diese Weise also über die Nummerierung eine Sortierung der Datensätze abbilden.

Die Zeilennummerierung ist auch eine gute Voraussetzung, um einzelne Datensätze als einen Datensatz zusammen zu fassen. Wird eine Abfrage zur Zeilennummerierung als Ansicht erstellt, so kann darauf mit einer weiteren Abfrage problemlos zugegriffen werden. Als einfaches Beispiel hier noch einmal die erste Abfrage zur Nummerierung, nur um ein Feld ergänzt:

```
001 SELECT "ID", "Name",
002    ( SELECT COUNT( "ID" ) FROM "Tabelle" WHERE "ID"   <= "a"."ID" )
003       AS "lfdNr."
004 FROM "Tabelle" AS "a"
```

Aus dieser Abfrage wird jetzt die Ansicht '*Ansicht1*' erstellt. Die Abfrage, mit der z.B. die ersten 3 Namen zusammen in einer Zeile erscheinen können, lautet:

```
001 SELECT "Name" AS "Name_1",
002    ( SELECT "Name" FROM "Ansicht1" WHERE "lfdNr." = 2 ) AS "Name_2",
003    ( SELECT "Name" FROM "Ansicht1" WHERE "lfdNr." = 3 ) AS "Name_3"
004 FROM "Ansicht1" WHERE "lfdNr." = 1
```

Auf diese Art und Weise können mehrere Datensätze nebeneinander als Felder dargestellt werden. Allerdings läuft diese Nummerierung einfach vom ersten bis zum letzten Datensatz durch.

Sollen alle Personen zu einem Nachnamen zugeordnet werden, so ließe sich das folgendermaßen realisieren:

```
001 SELECT "ID", "Name", "Nachname",
```

```
002    ( SELECT COUNT( "ID" ) FROM "Tabelle" WHERE "ID" <= "a"."ID"
003       AND "Nachname" = "a"."Nachname") AS "lfdNr."
004 FROM "Tabelle" AS "a"
```

Jetzt kann über die erstellte Ansicht eine entsprechende Familienzusammenstellung erfolgen:

```
001 SELECT "Nachname", "Name" AS "Name_1",
002    ( SELECT "Name" FROM "Ansicht1" WHERE "lfdNr." = 2
003       AND "Nachname" = "a"."Nachname") AS "Name_2",
004    ( SELECT "Name" FROM "Ansicht1" WHERE "lfdNr." = 3
005       AND "Nachname" = "a"."Nachname") AS "Name_3"
006 FROM "Ansicht1" AS "a"
007 WHERE "lfdNr." = 1
```

In einem Adressbuch ließen sich so alle Personen einer Familie ("Nachnamen") zusammenfassen, damit jede Adresse nur einmal für ein Anschreiben berücksichtigt würde, aber alle Personen, an die das Anschreiben gehen soll, aufgeführt würden.

Da es sich um keine fortwährende Schleifenfunktion handelt, ist hier allerdings Vorsicht geboten. Schließlich wird die Grenze der parallel als Felder angezeigten Datensätze durch die Abfrage im obigen Beispiel z.B. auf 3 begrenzt. Diese Grenze wurde willkürlich gesetzt. Weitere Namen tauchen nicht auf, auch wenn die Nummerierung der "*lfdNr.*" größer als 3 ist.

In seltenen Fällen ist so eine Grenze aber auch klar nachvollziehbar. Soll z.B. ein Kalender erstellt werden, so können mit diesem Verfahren die Zeilen die Wochen des Jahres darstellen, die Spalten die Wochentage. Da im ursprünglichen Kalender nur das Datum über den Inhalt entscheidet, werden durch die Zeilennummerierung immer die Tage einer Woche durchnummeriert und nach Wochen im Jahr als Datensatz später ausgegeben. Spalte 1 gibt dann Montag wieder, Spalte 2 Dienstag usw. Die Unterabfrage endet also jeweils bei der "*lfdNr.*" = *7*. Damit lassen sich dann im Bericht alle sieben Wochentage nebeneinander anzeigen und eine entsprechende Kalenderübersicht erstellen.

Zeilenumbruch durch eine Abfrage erreichen

Manchmal ist es sinnvoll, durch eine Abfrage verschiedene Felder zusammenzufassen und mit einem Zeilenumbruch zu trennen. So ist es z.B. einfacher eine Adresse in einen Bericht komplett einzulesen.

Der Zeilenumbruch innerhalb einer Abfrage erfolgt durch **`Char(13)`**. Beispiel:

```
001 SELECT "Vorname"||' '||"Nachname"||Char(13)||"Straße"||Char(13)||"Ort"
002 FROM "Tabelle"
```

Dies erzeugt nachher:

```
Vorname Nachname
Straße
Ort
```

Hinweis

Für Firebird muss statt **`CHAR(13)`** **`ASCII_CHAR(13)`** eingefügt werden. Firebird kennt die hier verwendete Kurzform nicht.

Mit so einer Abfrage, zusammen mit einer Nummerierung jeweils bis zur Nummer 3, lassen sich auch dreispaltige Etikettendrucke von Adressetiketten über Berichte realisieren. Eine Nummerierung ist in diesem Zusammenhang nötig, damit drei Adressen nebeneinander in einem Datensatz erscheinen. Nur so sind sie auch nebeneinander im Bericht einlesbar.

Je nach Betriebssystem kann es auch notwendig sein, zusätzlich zu Char(13) auch Char(10) in den Code mit aufzunehmen:

```
001 SELECT "Vorname"||' '||"Nachname"||Char(13)||Char(10)||
002    "Straße"||Char(13)||Char(10)||"Ort"
```

```
003 FROM "Tabelle"
```

Solche Zeilenumbrüche werden allerdings nicht in der Abfrage angezeigt. Die Steuerzeichen werden in einem VARCHAR-Feld nicht umgesetzt. Entsprechend rückt der Text dort ohne Leerzeichen aneinander. Soll der Zeilenumbruch in der Abfrage angezeigt werden, so muss der zusammengefügte Text von VARCHAR nach LONGVARCHAR umgewandelt werden:

```
001 SELECT CAST("Vorname"||' '||"Nachname"||Char(13)||Char(10)||
002    "Straße"||Char(13)||Char(10)||"Ort" AS LONGVARCHAR) AS "Adresse"
003 FROM "Tabelle"
```

Damit wird dann die Adresse auch in einer Abfrage mehrzeilig unter dem Alias "Adresse" dargestellt.

Gruppieren und Zusammenfassen

Für andere Datenbanken, auch neuere Versionen der HSQLDB, ist der Befehl **'Group_Concat()'** verfügbar. Mit ihm können einzelne Felder einer Datensatzgruppe zusammengefasst werden. So ist es z.B. möglich, in einer Tabelle Vornamen und Nachnamen zu speichern und anschließend die Daten so darzustellen, dass in einem Feld die Nachnamen als Familiennamen erscheinen und in dem 2. Feld alle Vornamen hintereinander, durch z.B. Komma getrennt, aufgeführt werden.

Hinweis

Für Firebird ist diese Vorgehensweise zum Gruppieren überflüssig. Mit

```
001 SELECT "Nachname", LIST( "Vorname", ', ' ) AS "Vornamen"
002 FROM "Tabelle" GROUP BY "Nachname"
```

werden die Vornamen, gruppiert nach den Nachnamen, zusammengefasst.

Dieses Beispiel entspricht in vielen Teilen dem der Zeilennummerierung. Die Gruppierung zu einem gemeinsamen Feld stellt hier eine Ergänzung dar.

Nachname	***Vorname***
Müller	Karin
Schneider	Gerd
Müller	Egon
Schneider	Volker
Müller	Monika
Müller	Rita

Wird nach der Abfrage zu:

Nachname	***Vornamen***
Müller	Karin, Egon, Monika, Rita
Schneider	Gerd, Volker

Dieses Verfahren kann in Grenzen auch in der HSQLDB nachgestellt werden. Das folgende Beispiel bezieht sich auf eine Tabelle "Name" mit den Feldern "ID", "Vorname" und "Nachname". Folgende Abfrage wird zuerst an die Tabelle gestellt und als Ansicht "Ansicht_Gruppe" gespeichert:

```
001 SELECT "Nachname", "Vorname",
002    ( SELECT COUNT( "ID" ) FROM "Name" WHERE "ID" <= "a"."ID"
003       AND "Nachname" = "a"."Nachname" ) AS "GruppenNr"
004 FROM "Name" AS "a"
```

Im Kapitel «Abfragen» ist nachzulesen, wie diese Abfrage über die *Korrelierte Unterabfrage* auf Feldinhalte in der gleichen Abfragezeile zugreift. Dadurch wird eine aufsteigende Nummerierung, gruppiert nach den "*Nachnamen*", erzeugt. Diese Nummerierung wird in der folgenden Abfrage benötigt, so dass in dem Beispiel maximal 5 Vornamen aufgeführt werden.

```
001 SELECT DISTINCT "Nachname",
002    ( SELECT "Vorname" FROM "Ansicht_Gruppe"
003       WHERE "Nachname" = "a"."Nachname" AND "GruppenNr" = 1 ) ||
004       COALESCE( ( SELECT ', ' || "Vorname" FROM "Ansicht_Gruppe"
005       WHERE "Nachname" = "a"."Nachname" AND "GruppenNr" = 2 ), '' ) ||
006       COALESCE( ( SELECT ', ' || "Vorname" FROM "Ansicht_Gruppe"
007       WHERE "Nachname" = "a"."Nachname" AND "GruppenNr" = 3 ), '' ) ||
008       COALESCE( ( SELECT ', ' || "Vorname" FROM "Ansicht_Gruppe"
009       WHERE "Nachname" = "a"."Nachname" AND "GruppenNr" = 4 ), '' ) ||
010       COALESCE( ( SELECT ', ' || "Vorname" FROM "Ansicht_Gruppe"
011       WHERE "Nachname" = "a"."Nachname" AND "GruppenNr" = 5 ), '' )
012 AS "Vornamen"
013 FROM "Ansicht_Gruppe" AS "a"
```

Durch Unterabfragen werden nacheinander die Vornamen zu Gruppenmitglied 1, 2 usw. abgefragt und zusammengefasst. Ab der 2. Unterabfrage muss abgesichert werden, dass **'NULL'**-Werte nicht die Zusammenfassung auf **'NULL'** setzen. Deshalb wird bei einem Ergebnis von **'NULL'** stattdessen **''** angezeigt.

Mehrere Werte in einem Feld speichern

Sollen mit dem Datensatz einer Tabelle mehrere Werte der gleichen Art verbunden werden, so wird dies normalerweise durch eine n:m-Beziehung über 3 Tabellen gelöst. Bei einer bereits bestehenden Datenbank und einem nicht zu großen Umfang der Werte können aber auch mehrere Werte in einem Feld abgespeichert und wieder abgerufen werden. Dies soll hier an einem Terminkalender dargestellt werden.[27]

Neben den üblichen Feldern für einen Terminkalender erscheint in der Tabelle "Termine" ein Feld "Personen". Aus diesem Feld "Personen" soll erkennbar sein, welche Personen von einem Termin betroffen sind. Jeder Person wird dabei ein bestimmter Wert zugeordnet, der gleichzeitig einem bestimmten Bit-Wert entspricht.

Person	***Wert (Integer)***	***Wert (Bit)***
Müller	1	0000 0001
Schneider	2	0000 0010
Meier	4	0000 0100
Schulze	8	0000 1000
Achenbach	16	0001 0000
Sorge	32	0010 0000

Wird in dem Feld "Personen" '1' abgespeichert, so betrifft der Termin nur die Person '1' - 'Müller'. Wird '3' eingetragen, so sind die Personen '1' - 'Müller' und '2' - 'Schneider' von dem Termin betroffen. In das Feld "Personen" wird also immer die Summe der Integer-Werte eingetragen. Über die folgende Abfrage kann dann ermittelt werden, welche von den Personen betroffen ist:

```
001 SELECT "Termine".*,
002    BITAND( "Personen", 1 ) "Müller",
003    BITAND( "Personen", 2 ) "Schneider",
004    BITAND( "Personen", 4 ) "Meier",
005    BITAND( "Personen", 8 ) "Schulze",
```

27 Die Datenbank «Beispiel_Arrayfeld.odb» ist den Beispieldatenbanken für dieses Handbuch beigefügt.

```
006    BITAND( "Personen", 16 ) "Achenbach",
007    BITAND( "Personen", 32 ) "Sorge"
008 FROM "Termine"
```

Während bei der HSQLDB die Funktion den Namen **BITAND** trägt, wird die gleiche Berechnung unter FIREBIRD mit **BIN_AND** durchgeführt. Die Abfrage ergibt die jeweiligen Personenwerte oder stattdessen 0, wenn der entsprechende Wert nicht in der abgespeicherten Integer-Zahl enthalten ist.

In einem Formular lassen sich diese unterschiedlichen Werte am besten mit Markierfeldern auslesen. Den Markierfeldern wird dabei unter **Eigenschaften → Allgemein → Titel** der jeweilige Name der Person gegeben, unter **Eigenschaften → Daten → Referenzwert (ein)** der Integer-Wert zugeschrieben. Bei **Referenzwert (aus)** kann '0' stehen. In dem Formular sind dann die jeweiligen Felder markiert, wenn das Formular mit der Abfrage und die Felder mit dem entsprechenden Namen verbunden sind.

Das Ändern der Werte soll jetzt natürlich auch über das Formular möglich sein. Dafür muss ein bisschen mit einem Makro nachgeholfen werden:

```
001 SUB PersonenSpeichern(oEvent AS OBJECT)
002    DIM aFields()
003    DIM oForm AS OBJECT
004    DIM inValue AS INTEGER
005    DIM i AS INTEGER
006    oForm = oEvent.Source.Model.Parent
007    aFields = Array("Check1","Check2","Check3","Check4","Check5","Check6")
008    inValue = 0
009    FOR i = LBound(aFields()) TO UBound(aFields())
010       IF oForm.getByName(aFields(i)).State = 1 THEN
011          inValue = inValue + CInt(oForm.getByName(aFields(i)).refValue)
012       END IF
013    NEXT
014    oForm.UpdateInt(oForm.findColumn("Personen"),inValue)
015 END SUB
```

Das Formular wird aus dem auslösenden Ereignis (**Eigenschaften → Ereignisse → Status geändert**) ausgelesen. Sämtliche Markierfelder, die zusammen mit ihrem Wert in dem Feld "Personen" vertreten sein sollen, sind in dem Array aufgelistet. Aus jedem Feld, bei dem der Status auf «ausgewählt» steht (**State = 1**) wird der Referenzwert ausgelesen. Dieser Referenzwert ist als Text gespeichert, so dass er für das Makro mit **Cint** in einen Integer-Wert umgewandelt werden muss. Alle Werte werden addiert und dann in das Feld "Personen" der dem Formular zugrundeliegenden Abfrage übertragen.

Damit lassen sich beliebige Personenzusammenstellungen in einem Feld speichern und auch wieder auslesen. Über Abfragen lassen sich alle Termine zusammenstellen, die eine Person hat, ohne dass dabei auch noch die anderen Termine stören, von denen die Person gar nicht betroffen ist.

Makros

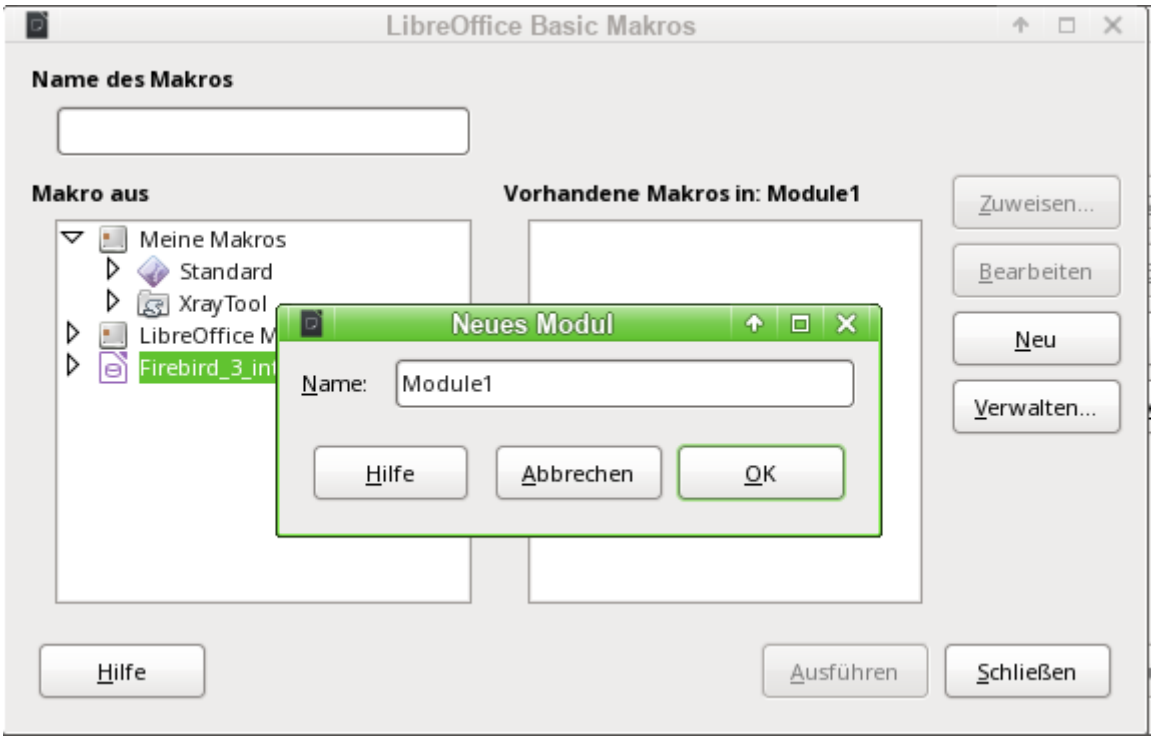
Allgemeines zu Makros

Prinzipiell kommt eine Datenbank unter Base ohne Makros aus. Irgendwann kann aber das Bedürfnis kommen,

- bestimmte Handlungsschritte zu vereinfachen (Wechsel von einem Formular zum anderen, Aktualisierung von Daten nach Eingabe in einem Formular ...),
- Fehleingaben besser abzusichern,
- häufigere Aufgaben zu automatisieren oder auch
- bestimmte SQL-Anweisungen einfacher aufzurufen als mit dem separaten SQL-Editor.

Es ist natürlich jedem selbst überlassen, wie intensiv er/sie Makros in Base nutzen will. Makros können zwar die Bedienbarkeit verbessern, sind aber auch immer mit geringen, bei ungünstiger Programmierung auch stärkeren Geschwindigkeitseinbußen des Programms verbunden. Es ist immer besser, zuerst einmal die Möglichkeiten der Datenbank und die vorgesehenen Einstellmöglichkeiten in Formularen auszureizen, bevor mit Makros zusätzliche Funktionen bereitgestellt werden. Makros sollten deshalb auch immer wieder mit größeren Datenbanken getestet werden, um ihren Einfluss auf die Verarbeitungsgeschwindigkeit abschätzen zu können.

Makros werden über den Weg **Extras → Makros → Makros verwalten → LibreOffice Basic...** erstellt. Es erscheint ein Fenster, das den Zugriff auf alle Makros ermöglicht. Makros für Base werden meistens in dem Bereich gespeichert, der dem Dateinamen der Base-Datei entspricht.



Über den Button Neu im Fenster «LibreOffice Basic Makros» wird ein zweites Fenster geöffnet. Hier wird lediglich nach der Bezeichnung für das Modul (Ordner, in dem das Makro abgelegt wird) gefragt. Der Name kann gegebenenfalls auch noch später geändert werden.

Sobald dies bestätigt wird, erscheint der Makro-Editor und auf seiner Eingabefläche wird bereits der Start und das Ende für eine Prozedur angegeben:

```
001  REM  *****  BASIC  *****
002
003  Sub Main
004
005  End Sub
```

Um Makros, die dort eingegeben wurden, nutzen zu können, sind folgende Schritte notwendig:

- Unter **Extras → Optionen → Sicherheit → Makrosicherheit** ist der **Sicherheitslevel → Mittel** zu wählen. Gegebenenfalls kann auch zusätzlich unter **Vertrauenswürdige Quellen → Vertrauenswürdige Speicherorte** der Pfad angegeben werden, in dem eigene Dateien mit Makros liegen, um spätere Nachfragen nach der Aktivierung von Makros zu vermeiden.
- Die Datenbankdatei muss nach der Erstellung des ersten Makro-Moduls einmal geschlossen und anschließend wieder geöffnet werden.

Einige Grundprinzipien zur Nutzung des Basic-Codes in LibreOffice:

- Zeilen haben keine Zeilenendzeichen. Zeilen enden mit einem festen Zeilenumbruch.
- Zwischen Groß- und Kleinschreibung wird bei Funktionen, reservierten Ausdrücken usw. nicht unterschieden. So ist z.B. die Bezeichnung «String» gleichbedeutend mit «STRING» oder auch «string» oder eben allen anderen entsprechenden Schreibweisen. Groß- und Kleinschreibung dienen nur der besseren Lesbarkeit.
- Eigentlich wird zwischen Prozeduren (beginnend mit **SUB**) und Funktionen (beginnend mit **FUNCTION**) unterschieden. Prozeduren sind ursprünglich Programmabschnitte ohne Rückgabewert, Funktionen können Werte zurückgeben, die anschließend weiter ausgewertet werden können. Inzwischen ist diese Unterscheidung weitgehend irrelevant; man spricht allgemein von Methoden oder Routinen – mit oder ohne Rückgabewert. Auch eine Prozedur kann einen Rückgabewert mit festem Variablentyp (außer «Variant») erhalten; der wird einfach in der Definition zusätzlich festgelegt:

  ```
  SUB myProcedure AS INTEGER
  END SUB
  ```

Zu weiteren Details siehe auch das Handbuch «Erste Schritte Makros mit LibreOffice».

Hinweis

Makros in diesem Kapitel sind entsprechend den Vorgaben aus dem Makro-Editor von LibreOffice eingefärbt:

```
Makro-Bezeichner
Makro-Kommentar
Makro-Operator
Makro-Reservierter-Ausdruck
Makro-Zahl
Makro-Zeichenkette
```

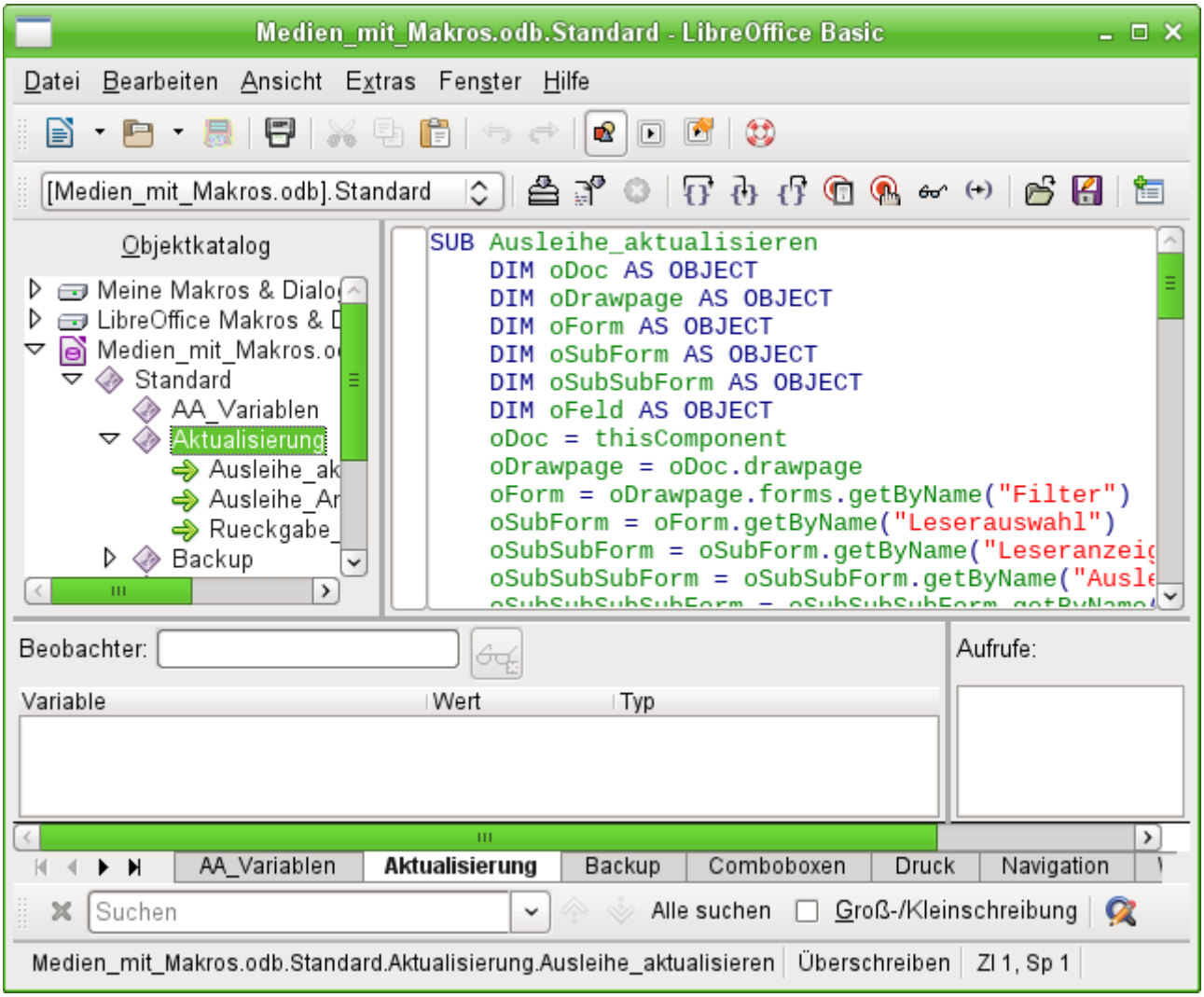
Hinweis

Bezeichner können frei gewählt werden, sofern sie nicht einem reservierten Ausdruck entsprechen. Viele Makros sind in dieser Anleitung mit an die deutsche Sprache angelehnten Bezeichnern versehen. Dies führte bei der englischsprachigen Übersetzung allerdings zu zusätzlichen Problemen. Deshalb sind die Bezeichner in neueren Makros an die englische Sprache angelehnt.

Hinweis

Die hier aufgezeigten Makros sind **nahezu ausschließlich innerhalb der Base-Datei gespeichert** und dort auch getestet. So kann z.B. der Kontakt zu einer Datenbank mit **ThisDatabaseDocument** nur innerhalb einer Base-Datei hergestellt werden. Sollen die Makros außerhalb der Datei unter **Meine Makros und Dialoge** gespeichert werden, so kann eventuell statt **ThisDatabaseDocument** einfach **ThisComponent** zum Ziel führen. Es kann aber auch sein, dass dann bestimmte Methoden einfach nicht zur Verfügung stehen.

Der Makro-Editor



Der Objektkatalog auf der linken Seite zeigt alle zur Zeit verfügbaren Bibliotheken und darin Module an, die über ein Ereignis aufgerufen werden können. **Meine Makros & Dialoge** ist für alle Dokumente eines Benutzers verfügbar. **LibreOffice Makros & Dialoge** sind für alle Benutzer des Rechners und auch anderer Rechner nutzbar, da sie standardmäßig mit LibreOffice installiert werden. Hinzu kommen noch die Bibliotheken, die in dem jeweiligen Dokument, hier **Medien_mit_Makros.odb**, abgespeichert sind.

Prinzipiell ist es zwar möglich, aus allen verfügbaren Bibliotheken die Module und die darin liegenden Makros zu nutzen. Für eine sinnvolle Nutzung empfiehlt es sich aber nicht, Makros aus anderen Dokumenten zu nutzen, da diese eben nur bei Öffnung des entsprechenden Dokumentes verfügbar sind. Ebenso ist es nicht empfehlenswert, Bibliotheken aus «Meine Makros & Dialoge» einzubinden, wenn die Datenbankdatei auch an andere Nutzer weitergegeben werden soll. Ausnahmen können hier Erweiterungen («Extensions») sein, die dann mit der Datenbankdatei weiter gegeben werden.

In dem Eingabebereich wird aus dem Modul **Aktualisierung** die Prozedur **Ausleihe_aktualisieren** angezeigt. Eingegebene Zeilen enden mit einem Return. Groß- und Kleinschreibung sowie Einrückung des Codes sind in Basic beliebig. Lediglich der Verweis auf Zeichenketten, z.B. "Filter", muss genau der Schreibweise in dem dort gemeinten Formular entsprechen.

Makros können schrittweise für Testzwecke durchlaufen werden. Entsprechende Veränderungen der Variablen werden im Beobachter angezeigt.

Benennung von Modulen, Dialogen und Bibliotheken

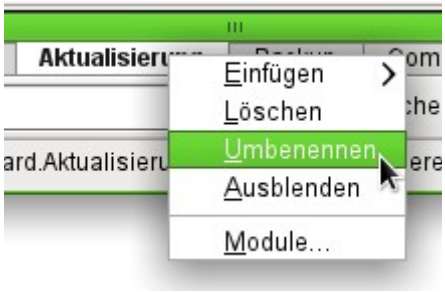
Die Benennung von Modulen, Dialogen und Bibliotheken sollte erfolgen, bevor irgendein Makro in die Datenbank eingebunden wird. Sie definieren schließlich den Pfad, in dem das auslösende Ereignis nach dem Makro sucht.

Innerhalb einer Bibliothek kann auf alle Makros der verschiedenen Module zugegriffen werden. Sollen Makros anderer Bibliotheken genutzt werden, so müssen diese extra geladen werden:

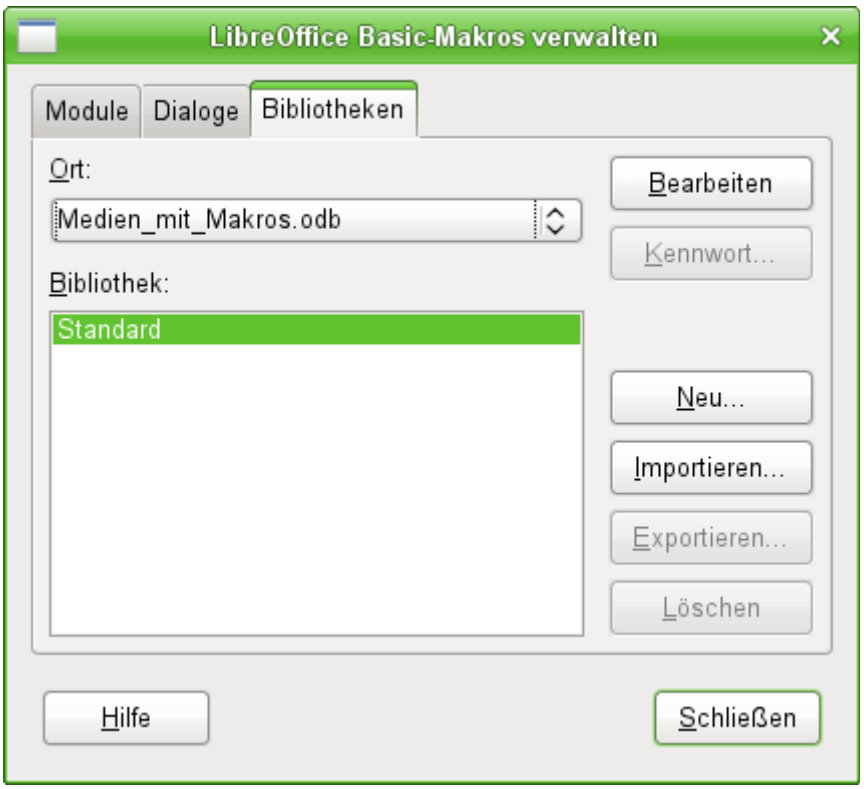
```
001 GlobalScope.BasicLibraries.LoadLibrary("Tools")
```

lädt die Bibliothek «Tools», die eine Bibliothek von LibreOffice Makros ist.

Über **Extras → Makros verwalten → LibreOffice Basic → Verwalten** kann der obige Dialog aufgerufen werden. Hier können neue Module und Dialoge erstellt und mit einem Namen versehen werden. Die Namen können allerdings nicht hier, sondern nur in dem Makroeditor selbst verändert werden.



Im Makroeditor wird mit einem rechten Mausklick auf die Reiter mit der Modulbezeichnung direkt oberhalb der Suchleiste ein Kontextmenü geöffnet, das u.a. die Änderung des Modulnamens ermöglicht.



Neue Bibliotheken können innerhalb der Base-Datei angelegt werden. Die Bezeichnung «Standard» der ersten erstellten Bibliothek lässt sich nicht ändern. Die Namen der weiteren Bibliotheken sind frei wählbar, anschließend aber auch nicht änderbar. In eine Bibliothek können Makros aus anderen Bibliotheken importiert werden. Sollte also der dringende Wunsch bestehen, eine andere Bibliotheksbezeichnung zu erreichen, so müsste eine neue Bibliothek mit diesem Namen erstellt werden und sämtlicher Inhalt der alten Bibliothek in die neue Bibliothek exportiert werden. Dann kann anschließend die alte Bibliothek gelöscht werden.

Tipp

Bei der Bibliothek «Standard» ist es nicht möglich, ein Kennwort zu setzen. Sollen Makros vor den Blicken des normalen Nutzers verborgen bleiben, so muss dafür eine neue Bibliothek erstellt werden. Diese lässt sich dann mit einem Kennwort schützen.

Makros in Base

Makros benutzen

Der «direkte Weg» über **Extras → Makros → Makros ausführen** ist zwar auch möglich, aber bei Base-Makros nicht üblich. Ein Makro wird in der Regel einem Ereignis zugeordnet und durch dieses Ereignis gestartet.

- Ereignisse eines Formular
- Bearbeitung einer Datenquelle innerhalb des Formulars
- Wechsel zwischen verschiedenen Kontrollfeldern
- Reaktionen auf Maßnahmen innerhalb eines Kontrollfelds

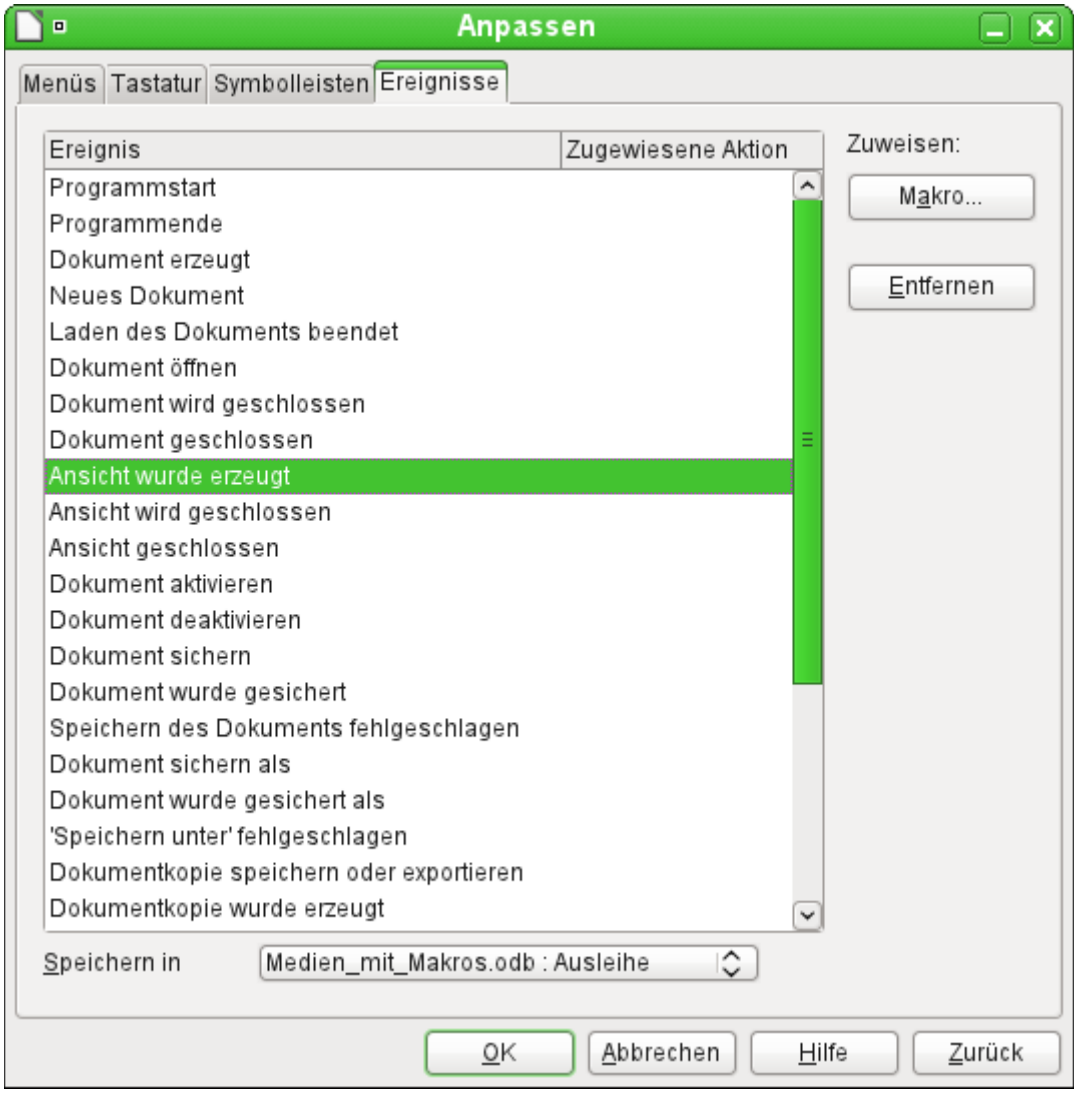
Der «direkte Weg» ist vor allem dann nicht möglich – auch nicht zu Testzwecken –, wenn eines der Objekte **`thisComponent`** (siehe den Abschnitt *Zugriff auf das Formular*) oder **`oEvent`** (siehe den Abschnitt *Zugriff auf Elemente eines Formulars*) benutzt wird.

Makros zuweisen

Damit ein Makro durch ein Ereignis gestartet werden kann, muss es zunächst definiert werden (siehe den einleitenden Abschnitt *Allgemeines zu Makros*). Dann kann es einem Ereignis zugewiesen werden. Dafür gibt es vor allem zwei Stellen.

Ereignisse eines Formulars beim Öffnen oder Schließen des Fensters

Maßnahmen, die beim Öffnen oder Schließen eines Formulardokuments erledigt werden sollen, werden so registriert:



- Rufen Sie im Formularentwurf über **Extras → Anpassen** das Register **Ereignisse** auf.
- Wählen Sie das passende Ereignis aus. Bestimmte Makros lassen sich nur starten, wenn **Ansicht wurde erzeugt** gewählt ist. Andere Makros wie z.B. das Erzeugen eines Vollbild-Formulars kann über **Dokument öffnen** gestartet werden.
- Suchen Sie über die Schaltfläche Makro das dafür definierte Makro und bestätigen Sie diese Auswahl.
- Unter **Speichern in** ist das Formular anzugeben (hier: «Medien_mit_Makros : Ausleihe)».

Dann kann diese Zuweisung mit OK bestätigt werden.

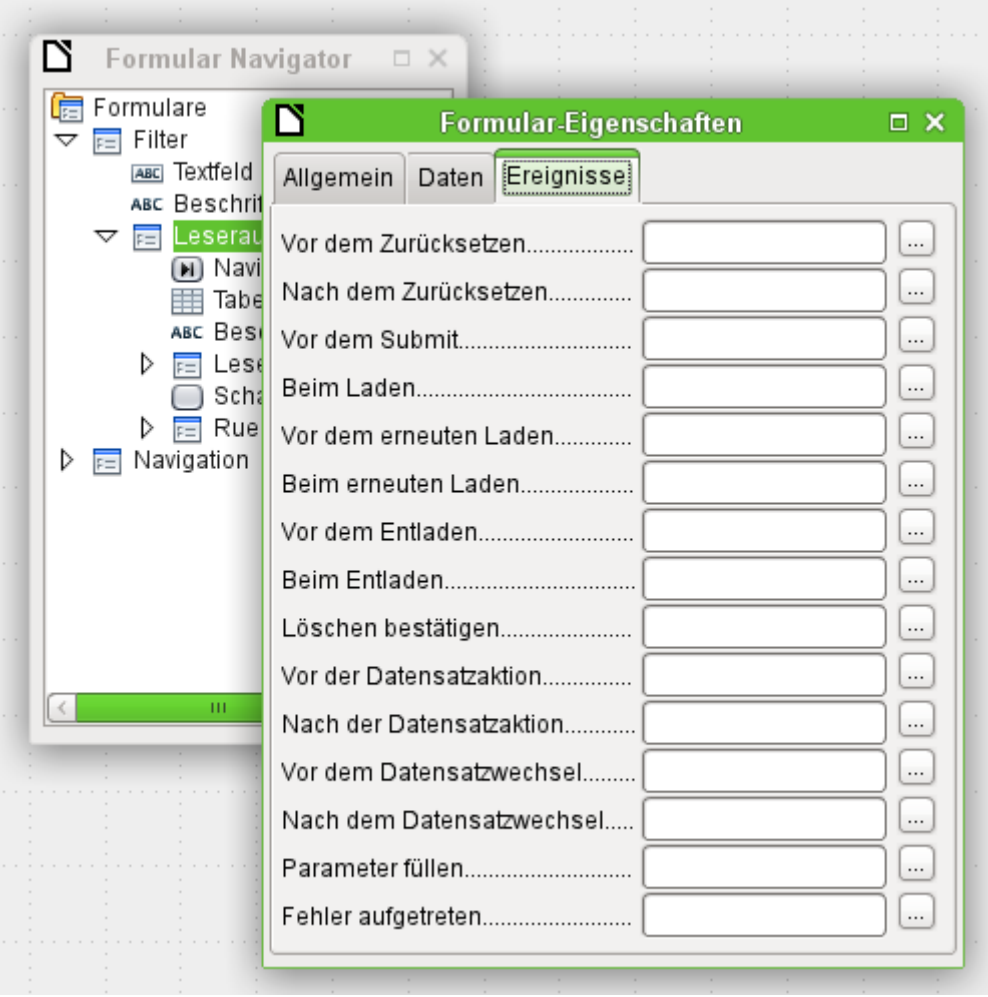
Die Einbindung von Makros in das Datenbankdokument erfolgt auf dem gleichen Weg. Nur stehen hier teilweise andere Ereignisse zur Wahl.

Vorsicht

Beim Starten eines Datenbankdokumentes kann, wenn die Makrounterstützung nicht aktiviert wurde, die Nachfrage kommen, ob Makros ausgeführt werden. Wird in einem Datenbankdokument das Ereignis **Ansicht wurde erzeugt** gewählt, so wird dieses Ereignis nicht ausgeführt, auch wenn die Frage nach dem Ausführen der Makros bejaht wurde.
Das Ereignis **Dokument öffnen** wird hingegen ausgeführt. Soll also auf jeden Fall beim Start des Datenbankdokumentes ein Makro ausgeführt werden, so sollte dieses Ereignis gewählt werden.

Ereignisse eines Formulars bei geöffnetem Fenster

Nachdem das Fenster für die gesamten Inhalte des Formulars (Formulardokument) geöffnet wurde, kann auf die einzelnen Elemente des Formulardokuments zugegriffen werden. Hierzu gehören auch die dem Formular zugeordneten Formularelemente.



Die Formularelemente können, wie in obigem Bild, über den Formularnavigator angesteuert werden. Sie sind genauso gut über jedes einzelne Kontrollfeld der Formularoberfläche über das Kontextmenü des Kontrollfeldes zu erreichen.

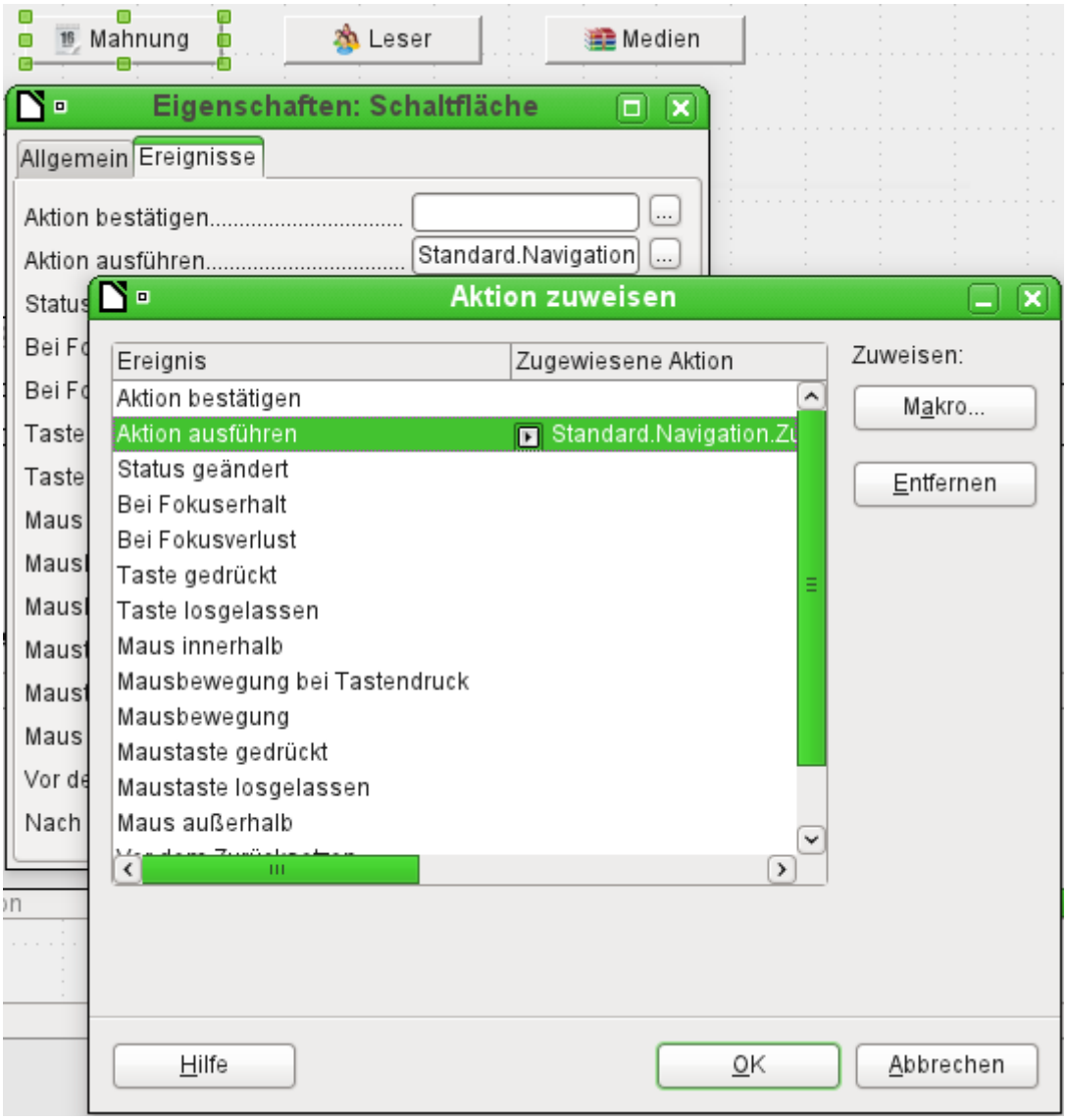
Die unter **Formular-Eigenschaften → Ereignisse** aufgezeigten Ereignisse finden statt während das Formularfenster geöffnet ist. Sie können für jedes Formular oder Unterformular des Formularfensters separat ausgewählt werden.

Hinweis

Der Gebrauch des Begriffes «Formular» ist bei Base leider nicht eindeutig. Der Begriff wird zum einen für das Fenster benutzt, das zur Eingabe von Daten geöffnet wird. Zum anderen wird der Begriff für das Element genutzt, das mit diesem Fenster eine bestimmte Datenquelle (Tabelle oder Abfrage) verbindet.
Es können auf einem Formularfenster sehr wohl mehrere Formulare mit unterschiedlichen Datenquelle untergebracht sein. Im Formularnavigator steht zuerst immer der Begriff «Formulare», dem dann in einem einfachen Formular lediglich ein Formular untergeordnet wird.
Das Formularfenster wird in dieser Dokumentation zur besseren Unterscheidung deshalb oft auch als «Formulardokument» bezeichnet.

Ereignisse innerhalb eines Formulars

Alle anderen Makros werden bei den Eigenschaften von Teilformularen und Kontrollfeldern über das Register **Ereignisse** registriert.



- Öffnen Sie (sofern noch nicht geschehen) das Fenster mit den Eigenschaften des Kontrollfelds.
- Wählen Sie im Register **Ereignisse** das passende Ereignis aus.

- Um die Datenquelle zu bearbeiten, gibt es vor allem die Ereignisse, die sich auf *Datensatz* oder *Aktualisieren* oder *Zurücksetzen* beziehen.
- Zu Schaltflächen oder einer Auswahl bei Listen- oder Optionsfeldern gehört in erster Linie das Ereignis *Aktion ausführen*.
- Alle anderen Ereignisse hängen vom Kontrollfeld und der gewünschten Maßnahme ab.

- Durch einen Klick auf den rechts stehenden Button ... wird das Fenster «Aktion zuweisen» geöffnet.
- Über die Schaltfläche Makro wird das dafür definierte Makro ausgewählt.

Über mehrfaches OK wird diese Zuweisung bestätigt.

Bestandteile von Makros

In diesem Abschnitt sollen einige Teile der Makro-Sprache erläutert werden, die in Base – vor allem bei Formularen – immer wieder benutzt werden. (Soweit möglich und sinnvoll, werden dabei die Beispiele der folgenden Abschnitte benutzt.)

Der «Rahmen» eines Makros

Die Definition eines Makros beginnt mit dem Typ des Makros – **SUB** oder **FUNCTION** – und endet mit **END SUB** bzw. **END FUNCTION**. Einem Makro, das einem Ereignis zugewiesen wird, können Argumente (Werte) übergeben werden; sinnvoll ist aber nur das Argument **oEvent**. Alle anderen Routinen, die von einem solchen Makro aufgerufen werden, können abhängig vom Zweck mit oder ohne Rückgabewert definiert werden und beliebig mit Argumenten versehen werden.

```
001  SUB Ausleihe_aktualisieren
002  END SUB
```

```
001  SUB Zu_Formular_von_Formular(oEvent AS OBJECT)
002  END SUB
```

```
001  FUNCTION Loeschen_bestaetigen(oEvent AS OBJECT) AS BOOLEAN
002     Loeschen_bestaetigen = FALSE
003  END FUNCTION
```

Es ist hilfreich, diesen Rahmen sofort zu schreiben und den Inhalt anschließend einzufügen. Bitte vergessen Sie nicht, Kommentare zur Bedeutung des Makros nach dem Grundsatz «so viel wie nötig, so wenig wie möglich» einzufügen. Außerdem unterscheidet Basic nicht zwischen Groß- und Kleinschreibung. In der Praxis werden feststehende Begriffe wie **SUB** vorzugsweise groß geschrieben, während andere Bezeichner Groß- und Kleinbuchstaben mischen.

Variablen definieren

Im nächsten Schritt werden – am Anfang der Routine – mit der **DIM**-Anweisung die Variablen, die innerhalb der Routine vorkommen, mit dem jeweiligen Datentyp definiert. Basic selbst verlangt das nicht, sondern akzeptiert, dass während des Programmablaufs neue Variablen auftreten. Der Programmcode ist aber «sicherer», wenn die Variablen und vor allem die Datentypen festgelegt sind. Viele Programmierer verpflichten sich selbst dazu, indem sie Basic über **Option Explicit** gleich zu Beginn eines Moduls mitteilen: Erzeuge nicht automatisch irgendwelche Variablen, sondern nutze nur die, die ich auch vorher definiert habe.

```
001  DIM oDoc AS OBJECT
002  DIM oDrawpage AS OBJECT
003  DIM oForm AS OBJECT
004  DIM sName AS STRING
005  DIM bOKEnabled AS BOOLEAN, iCounter AS INTEGER, dBirthday AS DATE
```

Die Deklaration von Variablen ist in Einzelzeilen oder zusammengefasst in einer Zeile, getrennt durch ein Komma, möglich.

Für die Namensvergabe stehen nur Buchstaben (A–Z oder a–z), Ziffern und der Unterstrich '_' zur Verfügung, aber keine Umlaute oder Sonderzeichen. (Unter Umständen ist das Leerzeichen zulässig. Sie sollten aber besser darauf verzichten.) Das erste Zeichen muss ein Buchstabe sein.

Üblich ist es, durch den ersten Buchstaben den Datentyp deutlich zu machen.[28] Dann erkennt man auch mitten im Code den Typ der Variablen. Außerdem sind «sprechende Bezeichner» zu empfehlen, sodass die Bedeutung der Variablen schon durch den Namen erkannt werden kann.

Die Liste der möglichen Datentypen in Star-Basic steht im Anhang des Handbuches. An verschiedenen Stellen sind Unterschiede zwischen der Datenbank, von Basic und der LibreOffice-API zu beachten. Darauf wird bei den Beispielen hingewiesen.

Arrays definieren

Gerade für Datenbanken ist die Sammlung von mehreren Variablen in einem Datensatz von Bedeutung. Werden mehrere Variablen zusammen in einer gemeinsamen Variablen gespeichert, so wird dies als ein Array bezeichnet. Ein Array muss definiert werden, bevor Daten in das Array geschrieben werden können.

```
001 DIM arDaten()
```

erzeugt eine leeres Array.

```
002 arDaten = array("Lisa","Schmidt")
```

So wird ein Array auf eine bestimmte Größe von 2 Elementen festgelegt und gleichzeitig mit Daten versehen.

Über

```
003 print arDaten(0), arDaten(1)
```

werden die beiden definierten Inhalte auf dem Bildschirm ausgegeben. Die **Zählung** für die Felder in **Arrays** beginnt hier mit **0**.

```
001 DIM arDaten(2)
002 arDaten(0) = "Lisa"
003 arDaten(1) = "Schmidt"
004 arDaten(2) = "Köln"
```

Dies erstellt ein Array, in dem 3 Elemente beliebigen Typs gespeichert werden können, also z.B. ein Datensatz mit den Variablen «Lisa» «Schmidt» «Köln». Mehr passt leider in dieses Array nicht hinein. Sollen mehr Elemente gespeichert werden, so muss das Array vergrößert werden. Wird während der Laufzeit eines Makros allerdings die Größe eines Arrays einfach nur neu definiert, so ist das Array anschließend leer wie eben ein neues Array.

```
005 ReDIM Preserve arDaten(3)
006 arDaten(3) = "18.07.2003"
```

Durch den Zusatz **Preserve** werden die vorherigen Daten beibehalten, das Array also tatsächlich zusätzlich um die Datumseingabe, hier in Form eines Textes, erweitert.

Das oben aufgeführte Array kann leider nur einen einzelnen Satz an Daten speichern. Sollen stattdessen, wie in einer Tabelle einer Datenbank, mehrere Datensätze gespeichert werden, so muss dem Array zu Beginn eine zusätzliche Dimension hinzugefügt werden.

```
001 DIM arDaten(2,1)
002 arDaten(0,0) = "Lisa"
003 arDaten(1,0) = "Schmidt"
004 arDaten(2,0) = "Köln"
005 arDaten(0,1) = "Egon"
006 arDaten(1,1) = "Müller"
007 arDaten(2,1) = "Hamburg"
```

28 Die Kennzeichnung sollte eventuell noch verfeinert werden, da mit nur einem Buchstaben zwischen dem Datentyp «Double» und dem Datentyp «Date» bzw. «Single» und «String» nicht unterschieden werden kann.

Auch hier gilt bei einer Erweiterung über das vorher definierte Maß hinaus, dass der Zusatz **Preserve** die vorher eingegebenen Daten mit übernimmt.

Zugriff auf das Formular

Das Formular liegt in dem momentan aktiven Dokument. Der Bereich, der dargestellt wird, wird als **drawpage** bezeichnet. Der Behälter, in dem alle Formulare aufbewahrt werden, heißt **forms** – im Formularnavigator ist dies sozusagen der oberste Begriff, an den dann sämtliche Formulare angehängt werden. Die o.g. Variablen erhalten auf diesem Weg ihre Werte:

```
001  oDoc = thisComponent
002  oDrawpage = oDoc.drawpage
003  oForm = oDrawpage.forms.getByName("Filter")
```

Das Formular, auf das zugegriffen werden soll, ist hier mit dem Namen «Filter» versehen. Dies ist der Name, der auch im Formularnavigator in der obersten Ebene sichtbar ist. (Standardmäßig erhält das erste Formular den Namen «MainForm».) Unterformulare liegen – hierarchisch angeordnet – innerhalb eines Formulars und können Schritt für Schritt erreicht werden:

```
004  DIM oSubForm AS OBJECT
005  DIM oSubSubForm AS OBJECT
006  oSubForm = oForm.getByName("Leserauswahl")
007  oSubSubForm = oSubForm.getByName("Leseranzeige")
```

Anstelle der Variablen in den «Zwischenstufen» kann man auch direkt zu einem bestimmten Formular gelangen. Ein Objekt der Zwischenstufen, das mehr als einmal verwendet wird, sollte selbständig deklariert und zugewiesen werden. (Im folgenden Beispiel wird **oSubForm** nicht mehr benutzt.)

```
006  oForm = thisComponent.drawpage.forms.getByName("Filter")
007  oSubSubForm = oForm.getByName("Leserauswahl").getByName("Leseranzeige")
```

Hinweis

Sofern ein Name ausschließlich aus Buchstaben und Ziffern besteht (keine Umlaute, keine Leer- oder Sonderzeichen), kann der Name in der Zuweisung auch direkt verwendet werden:

```
006  oForm = thisComponent.drawpage.forms.Filter
007  oSubSubForm = oForm.Leserauswahl.Leseranzeige
```

Anders als bei Basic sonst üblich, ist bei solchen Namen auf Groß- und Kleinschreibung genau zu achten.

Einen anderen Zugang zum Formular ermöglicht das auslösende Ereignis für das Makro.

Startet ein Makro über ein Ereignis des Formulars, wie z. B. **Formular-Eigenschaften → Vor der Datensatzaktion**, so wird das Formular selbst folgendermaßen erreicht:

```
001  SUB MakrobeispielBerechne(oEvent AS OBJECT)
002     oForm = oEvent.Source
003     ...
004  END SUB
```

Startet ein Makro über ein Ereignis eines Formularfeldes, wie z. B. Eigenschaften: **Textfeld → Bei Fokusverlust**, so kann sowohl das Formularfeld als auch das Formular ermittelt werden:

```
001  SUB MakrobeispielBerechne(oEvent AS OBJECT)
002     oFeld = oEvent.Source.Model
003     oForm = oFeld.Parent
004     ...
005  END SUB
```

Die Zugriffe über das Ereignis haben den Vorteil, dass kein Gedanke darüber verschwendet werden muss, ob es sich bei dem Formular um ein Hauptformular oder Unterformular handelt. Auch interessiert der Name des Formulars für die Funktionsweise des Makros nicht.

Zugriff auf Elemente eines Formulars

In gleicher Weise kann man auf die Elemente eines Formulars zugreifen: Deklarieren Sie eine entsprechende Variable als **object** und suchen Sie das betreffende Kontrollfeld innerhalb des Formulars:

```
001 DIM btnOK AS OBJECT  ' Button »OK»
002 btnOK = oSubSubForm.getByName("Schaltfläche 1")   ' aus dem Formular Leseranzeige
```

Dieser Weg funktioniert immer dann, wenn bekannt ist, mit welchem Element das Makro arbeiten soll. Wenn aber im ersten Schritt zu prüfen ist, welches Ereignis das Makro gestartet hat, ist der o.g. Weg über **oEvent** sinnvoll. Dann wird die Variable innerhalb des Makro-"Rahmens" deklariert und beim Start des Makros zugewiesen. Die Eigenschaft **Source** liefert immer dasjenige Element, das das Makro gestartet hat; die Eigenschaft **Model** beschreibt das Kontrollfeld im Einzelnen:

```
001 SUB Auswahl_bestaetigen(oEvent AS OBJECT)
002    DIM btnOK AS OBJECT
003    btnOK = oEvent.Source.Model
004 END
```

Mit dem Objekt, das man auf diesem Weg erhält, werden die weiteren angestrebten Maßnahmen ausgeführt.

Bitte beachten Sie, dass auch Unterformulare als Bestandteile eines Formulars gelten.

Zugriff auf die Datenbank

Normalerweise wird der Zugriff auf die Datenbank über Formulare, Abfragen, Berichte oder die Serienbrief-Funktion geregelt, wie es in allen vorhergehenden Kapiteln beschrieben wurde. Wenn diese Möglichkeiten nicht genügen, kann ein Makro auch gezielt die Datenbank ansprechen, wofür es mehrere Wege gibt.

Die Verbindung zur Datenbank

Das einfachste Verfahren benutzt dieselbe Verbindung wie das Formular, wobei **oForm** wie oben bestimmt wird:

```
001 DIM oConnection AS OBJECT
002 oConnection = oForm.activeConnection()
```

Oder man holt die Datenquelle, also die Datenbank, durch das Dokument und benutzt die vorhandene Verbindung auch für das Makro:

```
001 DIM oDatasource AS OBJECT
002 DIM oConnection AS OBJECT
003 oDatasource = thisComponent.Parent.dataSource
004 oConnection = oDatasource.getConnection("","")
```

Ein weiterer Weg stellt sicher, dass bei Bedarf die Verbindung zur Datenbank hergestellt wird:

```
001 DIM oDatasource AS OBJECT
002 DIM oConnection AS OBJECT
003 oDatasource = thisComponent.Parent.CurrentController
004 IF NOT (oDatasource.isConnected()) THEN oDatasource.connect()
005 oConnection = oDatasource.ActiveConnection()
```

Die **IF**-Bedingung bezieht sich hier nur auf eine Zeile. Deshalb ist **END IF** nicht erforderlich.

Wenn das Makro durch die Benutzeroberfläche – nicht aus einem Formulardokument heraus – gestartet werden soll, ist folgende Variante geeignet. Dazu muss das Makro innerhalb der Base-Datei gespeichert werden.

```
001 DIM oDatasource AS OBJECT
002 DIM oConnection AS OBJECT
003 oDatasource = thisDatabaseDocument.CurrentController
004 IF NOT (oDatasource.isConnected()) THEN oDatasource.connect()
005 oConnection = oDatasource.ActiveConnection()
```

Der Zugriff auf Datenbanken außerhalb der aktuellen Datenbank ist folgendermaßen möglich:

```
001  DIM oDatabaseContext AS OBJECT
002  DIM oDatasource AS OBJECT
003  DIM oConnection AS OBJECT
004  oDatabaseContext = createUnoService("com.sun.star.sdb.DatabaseContext")
005  oDatasource = oDatabaseContext.getByName("angemeldeter Name der Datenbank in LO")
006  oConnection = oDatasource.GetConnection("","")
```

Auch die Verbindung zu nicht in LO angemeldete Datenbanken ist möglich. Hier muss dann lediglich statt des angemeldeten Namens der Pfad zur Datenbank mit «file:///..../Datenbank.odb» angegeben werden.

Ergänzende Hinweise zur Datenbankverbindung stehen im Abschnitt *Verbindung mit Datenbanken erzeugen*.

SQL-Befehle

Die Arbeit mit der Datenbank erfolgt über SQL-Befehle. Ein solcher muss also erstellt und an die Datenbank geschickt werden; je nach Art des Befehls wird das Ergebnis ausgewertet und weiter verarbeitet. Mit der Anweisung **createStatement** wird das Objekt dafür erzeugt:

```
001  DIM oSQL_Statement AS OBJECT    ' das Objekt, das den SQL-Befehl ausführt
002  DIM stSql AS STRING             ' Text des eigentlichen SQL-Befehls
003  DIM oResult AS OBJECT           ' Ergebnis für executeQuery
004  DIM iResult AS INTEGER          ' Ergebnis für executeUpdate
005  oSQL_Statement = oConnection.createStatement()
```

Um *Daten abzufragen,* wird mit dem Befehl die Methode **executeQuery** aufgerufen und ausgeführt; das Ergebnis wird anschließend ausgewertet. Tabellennamen und Feldnamen werden üblicherweise in doppelte Anführungszeichen gesetzt. Diese müssen im Makro durch weitere doppelte Anführungszeichen maskiert werden, damit sie im Befehl erscheinen.

```
006  stSql = "SELECT * FROM ""Tabelle1"""
007  oResult = oSQL_Statement.executeQuery(stSql)
```

Um *Daten zu ändern* – also für **INSERT**, **UPDATE** oder **DELETE** – oder um die *Struktur der Datenbank* zu beeinflussen, wird mit dem Befehl die Methode **executeUpdate** aufgerufen und ausgeführt. Je nach Art des Befehls und der Datenbank erhält man kein nutzbares Ergebnis (ausgedrückt durch die Zahl 0) oder die Anzahl der bearbeiteten Datensätze.

```
008  stSql = "DROP TABLE ""Suchtmp"" IF EXISTS"
009  iResult = oSQL_Statement.executeUpdate(stSql)
```

Der Vollständigkeit halber sei noch ein Spezialfall erwähnt: Wenn **oSQL_Statement** unterschiedlich für **SELECT** oder für andere Zwecke benutzt wird, steht die Methode **execute** zur Verfügung. Diese benutzen wir nicht; wir verweisen dazu auf die API-Referenz.

Hinweis

SQL-Befehle, die so abgesandt werden, entsprechen nicht genau dem, was z. B. bei den Abfragen über direkte SQL-Ausführung erreicht wird. Eine Abfrage wie
… "Name" LIKE '%*%'
gibt nicht nur die Namen mit einem '*' wieder, da intern aus dem '*' ein '%' erstellt wird. Um wirklich das gleiche Verhalten mit direkter SQL-Ausführung zu erhalten, muss
oSQL_Statement.EscapeProcessing = False
nach der Erstellung von oSQL_Statement und vor der Ausführung des Codes eingefügt werden:

```
001  oSQL_Statement = oConnection.createStatement()
002  oSQL_Statement.EscapeProcessing = False
```

Vorbereitete SQL-Befehle mit Parametern

In allen Fällen, in denen manuelle Eingaben der Benutzer in einen SQL-Befehl übernommen werden, ist es einfacher und sicherer, den Befehl nicht als lange Zeichenkette zu erstellen, sondern ihn vorzubereiten und mit Parametern zu benutzen. Das vereinfacht die Formatierung von

Zahlen, Datumsangaben und auch Zeichenketten (die ständigen doppelten Anführungszeichen entfallen) und verhindert Datenverlust durch böswillige Eingaben.

Bei diesem Verfahren wird zunächst das Objekt für einen bestimmten SQL-Befehl erstellt und vorbereitet:

```
001 DIM oSQL_Statement AS OBJECT      ' das Objekt, das den SQL-Befehl ausführt
002 DIM stSql AS STRING               ' Text des eigentlichen SQL-Befehls
003 stSql = "UPDATE ""Verfasser"" " _
004    & "SET ""Nachname"" = ?, ""Vorname"" = ?" _
005    & "WHERE ""ID"" = ?"
006 oSQL_Statement = oConnection.prepareStatement(stSql)
```

Das Objekt wird mit **prepareStatement** erzeugt, wobei der SQL-Befehl bereits bekannt sein muss. Jedes Fragezeichen markiert eine Stelle, an der später – vor der Ausführung des Befehls – ein konkreter Wert eingetragen wird. Durch das «Vorbereiten» des Befehls stellt sich die Datenbank darauf ein, welche Art von Angaben – in diesem Fall zwei Zeichenketten und eine Zahl – vorgesehen ist. Die verschiedenen Stellen werden durch die Position (ab 1 gezählt) unterschieden.

Anschließend werden mit passenden Anweisungen die Werte übergeben und danach der SQL-Befehl ausgeführt. Die Werte werden hier aus Kontrollfeldern des Formulars übernommen, können aber auch aus anderen Makro-Elementen stammen oder im Klartext angegeben werden:

```
007 oSQL_Statement.setString(1, oTextfeld1.Text)    ' Text für den Nachnamen
008 oSQL_Statement.setString(2, oTextfeld2.Text)    ' Text für den Vornamen
009 oSQL_Statement.setLong(3, oZahlenfeld1.Value)   ' Wert für die betreffende ID
010 iResult = oSQL_Statement.executeUpdate
```

Die vollständige Liste der Zuweisungen findet sich im Abschnitt *Parameter für vorbereitete SQL-Befehle*.

Es ist auch möglich, direkt mehrere Datensätze über einen vorbereiteten SQL-Befehl einzufügen:

```
011 stSql = "UPDATE ""Person"" " _
012    & "SET ""Name"" = ?" _
013    & "WHERE ""ID"" = ?"
014 oSQL_Statement = oConnection.prepareStatement(stSql)
015 oSQL_Statement.setString(1, "Bill")
016 oSQL_Statement.setLong(2, 1)
017 oSQL_Statement.addBatch()
018 oSQL_Statement.setString(1, "Michaela")
019 oSQL_Statement.setLong(2, 2)
020 oSQL_Statement.addBatch()
021 oSQL_Statement.executeBatch()
```

Mit **clearBatch** könnte der gesamte Inhalt vor der Ausführung auch wieder zurückgenommen werden.

Wer sich weiter über die Vorteile dieses Verfahrens informieren möchte, findet hier Erläuterungen:

- *SQL-Injection (Wikipedia)* (http://de.wikipedia.org/wiki/SQL-Injection)
- *Why use PreparedStatement (Java JDBC)* (http://javarevisited.blogspot.de/2012/03/why-use-preparedstatement-in-java-jdbc.html)
- *SQL-Befehle (Einführung in SQL)* (http://de.wikibooks.org/wiki/Einführung_in_SQL:_SQL-Befehle#Hinweis_f.C3.BCr_Programmierer:_Parameter_benutzen.21)

Datensätze lesen und benutzen

Es gibt – abhängig vom Zweck – mehrere Wege, um Informationen aus einer Datenbank in ein Makro zu übernehmen und weiter zu verarbeiten.

Bitte beachten Sie: Wenn hier von einem «Formular» gesprochen wird, kann es sich auch um ein Unterformular handeln. Es geht dann immer um dasjenige (Teil-) Formular, das mit einer bestimmten Datenmenge verbunden ist.

Mithilfe des Formulars

Der aktuelle Datensatz und seine Daten stehen immer über das Formular zur Verfügung, das die betreffende Datenmenge (Tabelle, Abfrage, Ansicht (*View*)) anzeigt. Dafür gibt es mehrere Methoden, die mit **get** und dem Datentyp bezeichnet sind, beispielsweise diese:

```
001 DIM ID AS LONG
002 DIM sName AS STRING
003 DIM dValue AS CURRENCY
004 DIM dEintritt AS NEW com.sun.star.util.Date
005 ID = oForm.getLong(1)
006 sName = oForm.getString(2)
007 dValue = oForm.getDouble(4)
008 dEintritt = oForm.getDate(7)
```

Bei allen diesen Methoden ist jeweils die Nummer der Spalte in der Datenmenge anzugeben – gezählt ab 1.

Hinweis

Bei allen Methoden, die mit **Datenbanken** arbeiten, wird **ab 1** gezählt. Das gilt sowohl für Spalten als auch für Zeilen.

Soll anstelle der Spaltennummern mit den Spaltennamen der zugrundeliegenden Datenmenge (Tabelle, Abfrage, Ansicht (*View*)) gearbeitet werden, so kann die Spaltennummer über die Methode **findColumn** ermittelt werden. Hier ein Beispiel zum Auffinden der Spalte "Name":

```
001 DIM sName AS STRING
002 DIM nName AS STRING
003 nName = oForm.findColumn("Name")
004 sName = oForm.getString(nName)
```

Das Ergebnis ist immer ein Wert des Typs der Methode, wobei die folgenden Sonderfälle zu beachten sind.

- Es gibt keine Methode für Daten des Typs **Decimal**, **Currency** o.ä., also für kaufmännisch exakte Berechnungen. Da Basic automatisch die passende Konvertierung vornimmt, kann ersatzweise **getDouble** verwendet werden.
- Bei **getBoolean** ist zu beachten, wie in der Datenbank «Wahr» und «Falsch» definiert sind. Die «üblichen» Definitionen (logische Werte, 1 als «Wahr») werden richtig verarbeitet.
- Datumsangaben können nicht nur mit dem Datentyp **DATE** definiert werden, sondern auch (wie oben) als **util.Date**. Das erleichtert u.a. Lesen und Ändern von Jahr, Monat, Tag.
- Bei ganzen Zahlen sind Unterschiede der Datentypen zu beachten. Im obigen Beispiel wird **getLong** verwendet; auch die Basic-Variable ID muss den Datentyp **Long** erhalten, da dieser vom Umfang her mit **Integer** aus der Datenbank übereinstimmt.

Die vollständige Liste dieser Methoden findet sich im Abschnitt *Datenzeilen bearbeiten*.

Tipp

Sollen Werte aus einem Formular für direkte Weiterverarbeitung in SQL genutzt werden (z.B. für die Eingabe der Daten in eine andere Tabelle), so ist es wesentlich einfacher, nicht nach dem Typ der Felder zu fragen.
Das folgende Makro, an **Eigenschaften: Schaltfläche → Ereignisse → Aktion ausführen** gekoppelt, liest das erste Feld des Formulars aus – unabhängig von dem für die Weiterverarbeitung in Basic erforderlichen Typ.

```
001  SUB WerteAuslesen(oEvent AS OBJECT)
002     DIM oForm AS OBJECT
003     DIM stFeld1 AS STRING
004     oForm = oEvent.Source.Model.Parent
005     stFeld1 = oForm.getString(1)
006  END SUB
```

Werden die Felder über **getString()** ausgelesen, so werden die Formatierungen beibehalten, die für eine Weiterverarbeitung in SQL notwendig sind. Ein Datum, das in einem deutschsprachigen Formular als '08.03.15' dargestellt wird, wird so im Format '2015-03-08' ausgelesen und kann direkt in SQL weiter verarbeitet werden.
Die Auslesung in dem dem Typ entsprechenden Format ist nur erforderlich, wenn im Makro Werte weiter verarbeitet, z.B. mit ihnen gerechnet werden soll.

Hinweis

Grundsätzlich können alle Felder mit **getString** ausgelesen werden. Um sie in Makros weiter zu verarbeiten, müssen die erhaltenen Strings dann gegebenenfalls anschließend in die jeweiligen Variablen umgewandelt werden.
Das Auslesen mit **getString** hat den Vorteil, dass auch leere Felder einwandfrei ermittelt werden können. Werden die (passenden) Variablen stattdessen z.B. über **getInt** ausgelesen, so ergibt selbst ein leeres Feld '0'. Dies täuscht also vor, dass in dem Feld eben diese Zahl enthalten war. So etwas ist besonders lästig, wenn eben auch der Wert '0' selbst vorkommt – bei der internen HSQLDB z. B. beim automatisch hoch zählenden Schlüsselwert als Startwert.
Alternativ muss immer mit **wasNull** nachgesehen werden, ob denn der Inhalt vielleicht leer gewesen ist.

Ergebnis einer Abfrage

In gleicher Weise kann die Ergebnismenge einer Abfrage benutzt werden. Im Abschnitt *SQL-Befehle* steht die Variable **oResult** für diese Ergebnismenge, die üblicherweise so oder ähnlich ausgelesen wird:

```
001  WHILE oResult.next         ' einen Datensatz nach dem anderen verarbeiten
002     rem übernimm die benötigten Werte in Variablen
003     stVar = oResult.getString(1)
004     inVar = oResult.getLong(2)
005     boVar = oResult.getBoolean(3)
006     rem mach etwas mit diesen Werten
007  WEND
```

Je nach Art des SQL-Befehls, dem erwarteten Ergebnis und dem Zweck kann vor allem die **WHILE**-Schleife verkürzt werden oder sogar entfallen. Aber grundsätzlich wird eine Ergebnismenge immer nach diesem Schema ausgewertet.

Soll nur der erste Datensatz ausgewertet werden, so wird mit

```
001  oResult.next
```

zuerst die Zeile auf diesen Datensatz bewegt und dann mit

```
002  stVar = oResult.getString(1)
```

z.B. der Inhalt des ersten Datenfeldes gelesen. Die Schleife entfällt hier.

```
003  IF wasNull THEN
```

```
004     ...
005  END IF
```

Mit dieser Nachfrage würde die gerade getätigte Abfrage zu **stVar** darauf überprüft, ob sie nach SQL-Standard **NULL** gewesen ist.

Die Abfrage zu dem obigen Beispiel hat in der ersten Spalte einen Text, in der zweiten Spalte einen Integer-Zahlenwert (**Integer** aus der Datenbank entspricht **Long** in Basic) und in der dritten Spalte ein Ja/Nein-Feld. Die Felder werden durch den entsprechenden Indexwert angesprochen. Der Index für die Felder beginnt hier, im Gegensatz zu der sonstigen Zählung bei Arrays, mit dem Wert '1'.

In dem so erstellten Ergebnis ist allerdings keine Navigation möglich. Nur einzelne Schritte zum nächsten Datensatz sind erlaubt. Um innerhalb der Datensätze navigieren zu können, muss der **ResultSetType** bei der Erstellung der Abfrage bekannt sein. Hierauf wird über

```
001  oSQL_Anweisung.ResultSetType = 1004
```

oder

```
001  oSQL_Anweisung.ResultSetType = 1005
```

zugegriffen. Der Typ **1004 - SCROLL_INTENSIVE** erlaubt eine beliebige Navigation. Allerdings bleibt eine Änderung an den Originaldaten während des Auslesens unbemerkt. Der Typ **1005 – SCROLL_SENSITIVE** berücksichtigt zusätzlich gegebenenfalls Änderungen an den Originaldaten, die das Abfrageergebnis beeinflussen könnten.

Soll zusätzlich in dem Ergebnissatz eine Änderung der Daten ermöglicht werden, so muss die **ResultSetConcurrency** vorher definiert werden. Die Update-Möglichkeit wird über

```
001  oSQL_Anweisung.ResultSetConcurrency = 1008
```

hergestellt. Der Typ **1007 - READ_ONLY** ist hier die Standardeinstellung.

Die Anzahl der Zeilen, die die Ergebnismenge enthält, kann nur nach Wahl der entsprechenden Typen so bestimmt werden:

```
001  DIM iResult AS LONG
002  IF oResult.last THEN           ' gehe zum letzten Datensatz, sofern möglich
003     iResult = oResult.getRow    ' die laufende Nummer ist die Anzahl
004  ELSE
005     iResult = 0
006  END IF
```

Hinweis

Sollen viele Daten über die Schleife **WHILE oResult.next** ausgelesen werden, so macht sich hier ein **Geschwindigkeitsunterschied** bei verschiedenen Datenbanken deutlich bemerkbar. Die interne Firebird Datenbank arbeitet hier deutlich schneller als die interne Hsqldb. Selbst bei gleichen Datenbanken kann es abhängig vom Treiber zu deutlichen Unterschieden kommen. Bei der MariaDB mit direkter Verbindung ist die Geschwindigkeit auf dem Level der Firebird Datenbank. Mit der JDBC-Verbindung hingegen ist das Auslesen so langsam wie mit der Hsqldb.
Dieses Verhalten kann schon bei einer Abfrage getestet werden, wenn zum Ausführen die direkte SQL-Verbindung genutzt wird. Das Scrollen zum letzten Datensatz braucht bei vielen Zeilen deutlich länger mit der internen HSQLDB und MariaDB über die JDBC-Verbindung als mit der internen Firebird Datenbank und MariaDB mit der direkten Verbindung.

Mithilfe eines Kontrollfelds

Wenn ein Kontrollfeld mit einer Datenmenge verbunden ist, kann der Wert auch direkt ausgelesen werden, wie es im nächsten Abschnitt beschrieben wird. Das ist aber teilweise mit Problemen verbunden. Sicherer ist – neben dem Verfahren *Mithilfe des Formulars* – der folgende Weg, der für verschiedene Kontrollfelder gezeigt wird:

```
001  sValue = oTextField.BoundField.Text        ' Beispiel für ein Textfeld
```

```
002  nValue = oNumericField.BoundField.Value    ' Beispiel für ein numerisches Feld
003  dValue = oDateField.BoundField.Date         ' Beispiel für ein Datumsfeld
```

BoundField stellt dabei die Verbindung her zwischen dem (sichtbaren) Kontrollfeld und dem eigentlichen Inhalt der Datenmenge.

Datensätze wechseln und bestimmte Datensätze ansteuern

Im vorletzten Beispiel wurde mit der Methode **Next** von einer Zeile der Ergebnismenge zur nächsten gegangen. In gleicher Weise gibt es weitere Maßnahmen und Prüfungen, und zwar sowohl für die Daten eines Formulars – angedeutet durch die Variable **oForm** – als auch für eine Ergebnismenge. Beispielsweise kann man beim Verfahren *Automatisches Aktualisieren von Formularen* den vorher aktuellen Datensatz wieder markieren:

```
001  DIM loRow AS LONG
002  loRow = oForm.getRow()    ' notiere die aktuelle Zeilennummer
003  oForm.reload()            ' lade die Datenmenge neu
004  oForm.absolute(loRow)     ' gehe wieder zu der notierten Zeilennummer
```

Im Abschnitt *In einer Datenmenge navigieren* stehen alle dazu passenden Methoden.

Die Methode mit **getRow()** funktioniert aber nur dann einwandfrei, wenn es sich bei dem aktuellen Datensatz nicht um einen neuen Datensatz handelt. Deswegen hier eine weitere Möglichkeit der Navigation über den **Bookmark**-Befehl. Damit lässt sich der Cursor auch dann sicher positionieren, wenn bei einer neuen Zeile **getRow() 0** ermittelt.

```
001  DIM var AS VARIANT
002  var = oForm.getBookmark()
003  loRow = oForm.getRow()
004  oForm.reload()
005  IF loRow = 0 THEN
006     oForm.MoveToInsertRow()
007  ELSE
008     oForm.MoveToBookmark(var)
009  END IF
```

Bei einem neuen Datensatz wird die Zeile mit '0' angegeben. In diesem Fall wird dann der Cursor nach dem Neuladen mit **MoveToInsertRow** auf diese Position gesetzt. Mit **MoveToBookmark** würde hier der Cursor auf die letzte Zeile mit Inhalt gesetzt.

Datensätze bearbeiten – neu anlegen, ändern, löschen

Um Datensätze zu bearbeiten, müssen mehrere Teile zusammenpassen: Eine Information muss vom Anwender in das Kontrollfeld gebracht werden; das geschieht durch die Tastatureingabe. Anschließend muss die Datenmenge «dahinter» diese Änderung zur Kenntnis nehmen; das geschieht durch das Verlassen eines Feldes und den Wechsel zum nächsten Feld. Und schließlich muss die Datenbank selbst die Änderung erfahren; das erfolgt durch den Wechsel von einem Datensatz zu einem anderen.

Bei der Arbeit mit einem Makro müssen ebenfalls diese Teilschritte beachtet werden. Wenn einer fehlt oder falsch ausgeführt wird, gehen Änderungen verloren und «landen» nicht in der Datenbank. In erster Linie muss die Änderung nicht in der Anzeige des Kontrollfelds erscheinen, sondern in der Datenmenge. Es ist deshalb sinnlos, die Eigenschaft **Text** des Kontrollfelds zu ändern. Diese Eigenschaft dient nur zur Anzeige des Wertes.

Bitte beachten Sie, dass nur Datenmengen vom Typ «Tabelle» problemlos geändert werden können. Bei anderen Datenmengen ist dies nur unter besonderen Bedingungen möglich.

Inhalt eines Kontrollfelds ändern

Wenn es um die Änderung eines einzelnen Wertes geht, wird das über die Eigenschaft **BoundField** des Kontrollfelds mit einer passenden Methode erledigt. Anschließend muss nur noch die Änderung an die Datenbank weitergegeben werden. Beispiel für ein Datumsfeld, in das das aktuelle Datum eingetragen werden soll:

```
001  DIM unoDate AS NEW com.sun.star.util.Date
```

```
002  unoDate.Year = Year(Date)
003  unoDate.Month = Month(Date)
004  unoDate.Day = Day(Date)
005  oDateField = oForm.getByName("Datum")
006  oDateField.BoundField.updateDate( unoDate )
007  oForm.updateRow()       ' Weitergabe der Änderung an die Datenbank
```

Für **BoundField** wird diejenige der **updateXxx**-Methoden aufgerufen, die zum Datentyp des Feldes passt – hier geht es um einen **Date**-Wert. Als Argument wird der gewünschte Wert übergeben – hier das aktuelle Datum, konvertiert in die vom Makro benötigte Schreibweise. Die entsprechende Erstellung des Datums kann auch durch die Formel **CDateToUnoDate** erreicht werden:

```
001  oDateField = oForm.getByName("Datum")
002  oDateField.BoundField.updateDate( CDateToUnoDate(NOW()) )
003  oForm.updateRow()       ' Weitergabe der Änderung an die Datenbank
```

Zeile einer Datenmenge ändern

Wenn mehrere Werte in einer Zeile geändert werden sollen, ist der vorstehende Weg ungeeignet. Zum einen müsste für jeden Wert ein Kontrollfeld existieren, was oft nicht gewünscht oder sinnvoll ist. Zum anderen muss man sich für jedes dieser Felder ein Objekt «holen». Der einfache und direkte Weg geht über das Formular, beispielsweise so:

```
001  DIM unoDate AS NEW com.sun.star.util.Date
002  unoDate.Year = Year(Date)
003  unoDate.Month = Month(Date)
004  unoDate.Day = Day(Date)
005  oForm.updateDate(3, unoDate )
006  oForm.updateString(4, "ein Text")
007  oForm.updateDouble(6, 3.14)
008  oForm.updateInt(7, 16)
009  oForm.updateRow()
```

Für jede Spalte der Datenmenge wird die zum Datentyp passende **updateXxx**-Methode aufgerufen. Als Argumente werden die Nummer der Spalte (ab 1 gezählt) und der jeweils gewünschte Wert übergeben. Anschließend muss nur noch die Änderung an die Datenbank weitergegeben werden.

Zeilen anlegen, ändern, löschen

Die genannten **Änderungen** beziehen sich immer auf die aktuelle Zeile der Datenmenge des Formulars. Unter Umständen muss vorher eine der Methoden aus *In einer Datenmenge navigieren* aufgerufen werden. Es werden also folgende Maßnahmen benötigt:

1. Wähle den aktuellen Datensatz.
2. Ändere die gewünschten Werte, wie im vorigen Abschnitt beschrieben.
3. Bestätige die Änderungen mit folgendem Befehl:
   `oForm.updateRow()`
4. Als Sonderfall ist es auch möglich, die Änderungen zu verwerfen und den vorherigen Zustand wiederherzustellen:
   `oForm.cancelRowUpdates()`

Für einen **neuen Datensatz** gibt es eine spezielle Methode (vergleichbar mit dem Wechsel in eine neue Zeile im Tabellenkontrollfeld). Es werden also folgende Maßnahmen benötigt:

1. Bereite einen neuen Datensatz vor:
   `oForm.moveToInsertRow()`
2. Trage alle vorgesehenen und benötigten Werte ein. Dies geht ebenfalls mit den **updateXxx**-Methoden, wie im vorigen Abschnitt beschrieben.
3. Bestätige die Neuaufnahme mit folgendem Befehl:
   `oForm.insertRow()`
4. Die Neuaufnahme kann nicht einfach rückgängig gemacht werden. Stattdessen ist die soeben neu angelegte Zeile wieder zu löschen.

Für das **Löschen** eines Datensatzes gibt es einen einfachen Befehl; es sind also folgende Maßnahmen nötig:

1. Wähle – wie für eine Änderung – den gewünschten Datensatz und mache ihn zum aktuellen.
2. Bestätige die Löschung mit folgendem Befehl:
   `oForm.deleteRow()`

Tipp

Damit eine Änderung in die Datenbank übernommen wird, ist sie durch **updateRow** bzw. **insertRow** ausdrücklich zu bestätigen. Während beim Betätigen des Speicher-Buttons die passende Funktion automatisch ermittelt wird, muss vor dem Abspeichern ermittelt werden, ob der Datensatz neu ist (**Insert**) oder ein bestehender Datensatz bearbeitet wurde (**Update**).

```
001  IF oForm.isNew THEN
002     oForm.insertRow()
003  ELSE
004     oForm.updateRow()
005  END IF
```

Kontrollfelder prüfen und ändern

Neben dem Inhalt, der aus der Datenmenge kommt, können viele weitere Informationen zu einem Kontrollfeld gelesen, verarbeitet und geändert werden. Das betrifft vor allem die Eigenschaften, die im Kapitel «Formulare» aufgeführt werden. Eine Übersicht steht im Abschnitt *Eigenschaften bei Formularen und Kontrollfeldern*.

In mehreren Beispielen des Abschnitts *Bedienbarkeit verbessern* wird die Zusatzinformation eines Feldes benutzt:

```
001  SUB Main(oEvent AS OBJECT)
002     DIM stTag AS STRING
003     stTag = oEvent.Source.Model.Tag
```

Die Eigenschaft **Text** kann – wie im vorigen Abschnitt erläutert – nur dann sinnvoll geändert werden, wenn das Feld nicht mit einer Datenmenge verbunden ist. Aber andere Eigenschaften, die «eigentlich» bei der Formulardefinition festgelegt werden, können zur Laufzeit angepasst werden. Beispielsweise kann in einem Beschriftungsfeld die Textfarbe gewechselt werden, wenn statt einer Meldung ein Hinweis oder eine Warnung angezeigt werden soll:

```
001  SUB showWarning(oField AS OBJECT, iType AS INTEGER)
002     SELECT CASE iType
003        CASE 1
004           oField.TextColor = RGB(0,0,255)    ' 1 = blau
005        CASE 2
006           oField.TextColor = RGB(255,0,0)    ' 2 = rot
007        CASE ELSE
008           oField.TextColor = RGB(0,255,0)    ' 0 = grün (weder 1 noch 2)
009     END SELECT
010  END SUB
```

Englische Bezeichner in Makros

Während der Formular-Designer in der deutschen Version auch deutsche Bezeichnungen für die Eigenschaften und den Datenzugriff verwendet, müssen in Basic englische Begriffe verwendet werden. Diese sind in den folgenden Übersichten aufgeführt.

Eigenschaften, die üblicherweise nur in der Formular-Definition festgelegt werden, stehen nicht in den Übersichten. Gleiches gilt für Methoden (Funktionen und Prozeduren), die nur selten verwendet werden oder für die kompliziertere Erklärungen nötig wären.

Die Übersichten nennen folgende Angaben:

- *Name* Bezeichnung der Eigenschaft oder Methode im Makro-Code
- *Datentyp* Einer der Datentypen von Basic
  Bei Funktionen ist der Typ des Rückgabewerts angegeben; bei Prozeduren entfällt diese Angabe.
- *L/S* Hinweis darauf, wie der Wert der Eigenschaft verwendet wird:
  - *L* nur Lesen
  - *S* nur Schreiben (Ändern)
  - *(L)* Lesen möglich, aber für weitere Verarbeitung ungeeignet
  - *(S)* Schreiben möglich, aber nicht sinnvoll
  - *L+S* geeignet für Lesen und Schreiben

Weitere Informationen finden sich vor allem in der *API-Referenz* mit Suche nach der englischen Bezeichnung des Kontrollfelds. Gut geeignet, um herauszufinden, welche Eigenschaften und Methoden denn eigentlich bei einem Element zur Verfügung stehen, ist auch das Tool *Xray* .

```
001  SUB Main(oEvent AS OBJECT)
002     Xray(oEvent)
003  END SUB
```

Hiermit wird die Erweiterung Xray aus dem Aufruf heraus gestartet.

Eigenschaften bei Formularen und Kontrollfeldern

Das «Modell» eines Kontrollfelds beschreibt seine Eigenschaften. Je nach Situation kann der Wert einer Eigenschaft nur gelesen und nur geändert werden. Die Reihenfolge orientiert sich an den Aufstellungen «Eigenschaften der Kontrollfelder» im Kapitel «Formular».

Wenn mit

```
001  oFeld = oForm.getByName("Name des Kontrollfeldes")
```

auf ein Kontrollfeld zugegriffen wird, so werden die Eigenschaften einfach durch ein Anhängen an dieses Objekt mit einem Punkt als Verbinder angesprochen:

```
001  oFeld.FontHeight = 16
```

definiert also z. B. die Schriftgröße in 16 Punkten.

Schrift

In jedem Kontrollfeld, das Text anzeigt, können die Eigenschaften der Schrift angepasst werden.

Name	***Daten-typ***	***L/S***	***Eigenschaft***
FontName	string	L+S	Schriftart.
FontHeight	single	L+S	Schriftgröße.
FontWeight	single	L+S	Schriftstärke.
FontSlant	integer	L+S	Art der Schrägstellung.
FontUnderline	integer	L+S	Art der Unterstreichung.
FontStrikeout	integer	L+S	Art des Durchstreichens.

Formular

Englische Bezeichnung: *Form*

Name	Daten-typ	L/S	Eigenschaft
ApplyFilter	boolean	L+S	Filter aktiviert.
Filter	string	L+S	Aktueller Filter für die Datensätze.
FetchSize	long	L+S	Anzahl der Datensätze, die «am Stück» geladen werden.
Row	long	L	Nummer der aktuellen Zeile.
RowCount	long	L	Anzahl der Datensätze. Entspricht der Anzeige der Gesamtdatensätze in der Navigationsleiste. Da nicht direkt alle Datensätze über FetchSize in den Cache gelesen werden steht hier z.B. '41*', obwohl die Tabelle deutlich mehr Datensätze hat. RowCount gibt dann leider auch nur '41' aus.

Einheitlich für alle Arten von Kontrollfeld

Englische Bezeichnung: *Control* – siehe auch *FormComponent*

Name	Datentyp	L/S	Eigenschaft
Name	string	L+(S)	Bezeichnung für das Feld.
Enabled	boolean	L+S	Aktiviert: Feld kann ausgewählt werden.
EnableVisible	boolean	L+S	Sichtbar: Feld wird dargestellt.
ReadOnly	boolean	L+S	Nur lesen: Inhalt kann nicht geändert werden.
TabStop	boolean	L+S	Feld ist in der Tabulator-Reihenfolge erreichbar.
Align	integer	L+S	Horizontale Ausrichtung: 0 = links, 1 = zentriert, 2 = rechts
BackgroundColor	long	L+S	Hintergrundfarbe.
Tag	string	L+S	Zusatzinformation.
HelpText	string	L+S	Hilfetext als «Tooltip».

Einheitlich für viele Arten von Kontrollfeld

Name	Daten-typ	L/S	Eigenschaft
Text	string	(L+S)	Inhalt des Feldes aus der Anzeige. Bei Textfeldern nach dem Lesen auch zur weiteren Verarbeitung geeignet, andernfalls nur in Ausnahmefällen.
Spin	boolean	L+S	Drehfeld eingeblendet (bei formatierten Feldern).
TextColor	long	L+S	Textfarbe.
DataField	string	L	Name des Feldes aus der Datenmenge
BoundField	object	L	Objekt, das die Verbindung zur Datenmenge herstellt und vor allem dem Zugriff auf den Feldinhalt dient.

Textfeld – weitere Angaben

Englische Bezeichnung: *TextField*

Name	Daten-typ	L/S	Eigenschaft
String	string	L+S	Inhalt des Feldes aus der Anzeige.
MaxTextLen	integer	L+S	Maximale Textlänge.

Name	Daten-typ	L/S	Eigenschaft
DefaultText	string	L+S	Standardtext.
MultiLine	boolean	L+S	Mehrzeilig oder einzeilig.
EchoChar	(integer)	L+S	Zeichen für Kennwörter (Passwort-Eingabe verstecken).

Numerisches Feld

Englische Bezeichnung: *NumericField*

Name	Daten-typ	L/S	Eigenschaft
ValueMin	double	L+S	Minimalwert zur Eingabe.
ValueMax	double	L+S	Maximalwert zur Eingabe.
Value	double	L+(S)	Aktueller Wert nicht für Werte aus der Datenmenge verwenden.
ValueStep	double	L+S	Intervall bei Verwendung mit Mausrad oder Drehfeld.
DefaultValue	double	L+S	Standardwert.
DecimalAccuracy	integer	L+S	Nachkommastellen.
ShowThousandsSeparator	boolean	L+S	Tausender-Trennzeichen anzeigen.

Datumsfeld

Englische Bezeichnung: *DateField*

Datumswerte werden als Datentyp **`long`** definiert und im ISO-Format YYYYMMDD angezeigt, also 20120304 für den 04.03.2012. Zur Verwendung dieses Typs zusammen mit **`getDate`** und **`updateDate`** sowie dem Typ **`com.sun.star.util.Date`** verweisen wir auf die Beispiele.

Name	Daten-typ	Datentyp ab LO 4.1.1	L/S	Eigenschaft
DateMin	long	com.sun.star.util.Date	L+S	Minimalwert zur Eingabe.
DateMax	long	com.sun.star.util.Date	L+S	Maximalwert zur Eingabe.
Date	long	com.sun.star.util.Date	L+(S)	Aktueller Wert nicht für Werte aus der Datenmenge verwenden.
DateFormat	integer		L+S	Datumsformat nach Festlegung des Betriebssystems: 0 = kurze Datumsangabe (einfach) 1 = kurze Datumsangabe `tt.mm.jj` (Jahr zweistellig) 2 = kurze Datumsangabe `tt.mm.jjjj` (Jahr vierstellig) 3 = lange Datumsangabe (mit Wochentag und Monatsnamen) Weitere Möglichkeiten sind der Formulardefinition oder der *API-Referenz* zu entnehmen.

Name	Daten-typ	Datentyp ab LO 4.1.1	L/S	Eigenschaft
DefaultDate	long	com.sun.star.util.Date	L+S	Standardwert.
DropDown	boolean		L+S	Aufklappbaren Monatskalender anzeigen.

Zeitfeld

Englische Bezeichnung: *TimeField*

Auch Zeitwerte werden als Datentyp **`long`** definiert.

Name	Daten-typ	Datentyp ab LO 4.1.1	L/S	Eigenschaft
TimeMin	long	com.sun.star.util.Time	L+S	Minimalwert zur Eingabe.
TimeMax	long	com.sun.star.util.Time	L+S	Maximalwert zur Eingabe.
Time	long	com.sun.star.util.Time	L+(S)	Aktueller Wert nicht für Werte aus der Datenmenge verwenden.
TimeFormat	integer		L+S	Zeitformat: 0 = kurz als `hh:mm` (Stunde, Minute, 24 Stunden) 1 = lang als `hh:mm:ss` (dazu Sekunden, 24 Stunden) 2 = kurz als `hh:mm` (12 Stunden AM/PM) 3 = lang als `hh:mm:ss` (12 Stunden AM/PM) 4 = als kurze Angabe einer Dauer 5 = als lange Angabe einer Dauer
DefaultTime	long	com.sun.star.util.Time	L+S	Standardwert.

Währungsfeld

Englische Bezeichnung: *CurrencyField*

Ein Währungsfeld ist ein numerisches Feld mit den folgenden zusätzlichen Möglichkeiten.

Name	Datentyp	L/S	Eigenschaft
CurrencySymbol	string	L+S	Währungssymbol (nur zur Anzeige).
PrependCurrencySymbol	boolean	L+S	Anzeige des Symbols vor der Zahl.

Formatiertes Feld

Englische Bezeichnung: *FormattedControl*

Ein formatiertes Feld wird wahlweise für Zahlen, Währungen oder Datum/Zeit verwendet. Sehr viele der bisher genannten Eigenschaften gibt es auch hier, aber mit anderer Bezeichnung.

Name	Daten-typ	L/S	Eigenschaft
CurrentValue	variant	L	Aktueller Wert des Inhalts; der konkrete Datentyp hängt vom Inhalt des Feldes und dem For-
EffectiveValue		L+(S)	

Name	Daten-typ	L/S	Eigenschaft
			mat ab.
EffectiveMin	double	L+S	Minimalwert zur Eingabe.
EffectiveMax	double	L+S	Maximalwert zur Eingabe.
EffectiveDefault	variant	L+S	Standardwert.
FormatKey	long	L+(S)	Format für Anzeige und Eingabe. Es gibt kein einfaches Verfahren, das Format durch ein Makro zu ändern.
EnforceFormat	boolean	L+S	Formatüberprüfung: Bereits während der Eingabe sind nur zulässige Zeichen und Kombinationen möglich.

Listenfeld

Englische Bezeichnung: *ListBox*

Der Lese- und Schreibzugriff auf den Wert, der hinter der ausgewählten Zeile steht, ist etwas umständlich, aber möglich.

Name	Datentyp	L/S	Eigenschaft
ListSource	array of string	L+S	Datenquelle: Herkunft der Listeneinträge oder Name der Datenmenge, die die Einträge liefert.
ListSourceType	integer	L+S	Art der Datenquelle: 0 = Werteliste 1 = Tabelle 2 = Abfrage 3 = Ergebnismenge eines SQL-Befehls 4 = Ergebnis eines Datenbank-Befehls 5 = Feldnamen einer Datenbank-Tabelle
StringItemList	array of string	L	Listeneinträge, die zur Auswahl zur Verfügung stehen.
ItemCount	integer	L	Anzahl der vorhandenen Listeneinträge.
ValueItemList	array of string	L	Liste der Werte, die über das Formular an die Tabelle weitergegeben werden.
DropDown	boolean	L+S	Aufklappbar.
LineCount	integer	L+S	Anzahl der angezeigten Zeilen im aufgeklappten Zustand.
MultiSelection	boolean	L+S	Mehrfachselektion vorgesehen.
SelectedItems	array of integer	L+S	Liste der ausgewählten Einträge, und zwar als Liste der Positionen in der Liste aller Einträge.

Das (erste) ausgewählte Element aus dem Listenfeld erhält man auf diesem Weg:

```
001  oControl = oForm.getByName("Name des Listenfelds")
002  sEintrag = oControl.ValueItemList( oControl.SelectedItems(0) )
```

Hinweis

Seit LO 4.1 wird direkt der Wert ermittelt, der bei einem Listenfeld an die Datenbank weitergegeben wird.

```
001  oControl = oForm.getByName("Name des Listenfelds")
002  iD = oControl.getCurrentValue()
```

Mit `getCurrentValue()` wird also immer der Wert ausgegeben, der auch tatsächlich in der Tabelle der Datenbank abgespeichert wird. Dies ist beim Listenfeld von dem hiermit verknüpften gebundenen Feld (`BoundField`) abhängig.
Bis einschließlich LO 4.0 wurde hier immer der angezeigte Inhalt, nicht aber der an die darunterliegende Tabelle weitergegebene Wert wiedergegeben.

Soll für die Einschränkung einer Auswahlmöglichkeit die Abfrage für ein Listenfeld ausgetauscht werden, so ist dabei zu beachten, dass es sich bei dem Eintrag um ein «array of string» handelt:

```
001  SUB Listenfeldfilter
002     DIM stSql(0) AS STRING
003     DIM oDoc AS OBJECT
004     DIM oDrawpage AS OBJECT
005     DIM oForm AS OBJECT
006     DIM oFeld AS OBJECT
007     oDoc = thisComponent
008     oDrawpage = oDoc.drawpage
009     oForm = oDrawpage.forms.getByName("MainForm")
010     oFeld = oForm.getByname("Listenfeld")
011     stSql(0) = "SELECT ""Name"", ""ID"" FROM ""Filter_Name"" ORDER BY ""Name"""
012     oFeld.ListSource = stSql
013     oFeld.refresh
014  END SUB
```

Hinweis

Soll der gerade geänderte Wert eines Listenfeldes ausgelesen werden, der noch nicht im Formular abgespeichert ist, so geht dies über die Listenposition:

```
001  SUB Kontofilter_Feldstart(oEvent AS OBJECT)
002     DIM oFeld AS OBJECT
003     DIM inID AS INTEGER
004     oFeld = oEvent.Source.Model
005     inID = oFeld.ValueItemList(oEvent.Selected)
006     ...
007  END SUB
```

Statt **`oEvent.Selected`** kann hier natürlich auch **`oFeld.SelectedItemPos`** stehen.

Kombinationsfeld

Englische Bezeichnung: *ComboBox*

Trotz ähnlicher Funktionalität wie beim Listenfeld weichen die Eigenschaften teilweise ab.

Hier verweisen wir ergänzend auf das Beispiel *Kombinationsfelder als Listenfelder mit Eingabemöglichkeit*.

Name	***Datentyp***	***L/S***	***Eigenschaft***
Autocomplete	boolean	L+S	Automatisch füllen.
StringItemList	array of string	L+S	Listeneinträge, die zur Auswahl zur Verfügung stehen.
ItemCount	integer	L	Anzahl der vorhandenen Listeneinträge.
DropDown	boolean	L+S	Aufklappbar.

Name	Datentyp	L/S	Eigenschaft
LineCount	integer	L+S	Anzahl der angezeigten Zeilen im aufgeklappten Zustand.
Text	string	L+S	Aktuell angezeigter Text.
DefaultText	string	L+S	Standardeintrag.
ListSource	string	L+S	Name der Datenquelle, die die Listeneinträge liefert.
ListSourceType	integer	L+S	Art der Datenquelle; gleiche Möglichkeiten wie beim Listenfeld (nur die Auswahl «Werteliste» wird ignoriert).

Markierfeld, Optionsfeld

Englische Bezeichnungen: *CheckBox* (Markierfeld) bzw. *RadioButton* (Optionsfeld; auch «Option Button» möglich)

Name	Datentyp	L/S	Eigenschaft
Label	string	L+S	Titel (Beschriftung)
State	short	L+S	Status 0 = nicht ausgewählt 1 = ausgewählt 2 = unbestimmt
MultiLine	boolean	L+S	Wortumbruch (bei zu langem Text).
RefValue	string	L+S	Referenzwert

Maskiertes Feld

Englische Bezeichnung: *PatternField*

Neben den Eigenschaften für «einfache» Textfelder sind folgende interessant.

Name	Datentyp	L/S	Eigenschaft
EditMask	string	L+S	Eingabemaske.
LiteralMask	string	L+S	Zeichenmaske.
StrictFormat	boolean	L+S	Formatüberprüfung bereits während der Eingabe.

Tabellenkontrollfeld

Englische Bezeichnung: *GridControl*

Name	Datentyp	L/S	Eigenschaft
Count	long	L	Anzahl der Spalten.
ElementNames	array of string	L	Liste der Spaltennamen.
HasNavigationBar	boolean	L+S	Navigationsleiste vorhanden.
RowHeight	long	L+S	Zeilenhöhe.

Beschriftungsfeld

Englische Bezeichnung: *FixedText* – auch *Label* ist üblich

Name	Datentyp	L/S	Eigenschaft
Label	string	L+S	Der angezeigte Text.

Name	Datentyp	L/S	Eigenschaft
MultiLine	boolean	L+S	Wortumbruch (bei zu langem Text).

Gruppierungsrahmen

Englische Bezeichnung: *GroupBox*

Keine Eigenschaft dieses Kontrollfelds wird üblicherweise durch Makros bearbeitet. Wichtig ist der Status der einzelnen Optionsfelder.

Schaltfläche

Englische Bezeichnungen: *CommandButton* – für die grafische Schaltfläche *ImageButton*

Name	Datentyp	L/S	Eigenschaft
Label	string	L+S	Titel – Text der Beschriftung.
State	short	L+S	Standardstatus «ausgewählt» bei «Umschalten».
MultiLine	boolean	L+S	Wortumbruch (bei zu langem Text).
DefaultButton	boolean	L+S	Standardschaltfläche

Navigationsleiste

Englische Bezeichnung: *NavigationBar*

Weitere Eigenschaften und Methoden, die mit der Navigation zusammenhängen – z.B. Filter und das Ändern des Datensatzzeigers –, werden über das Formular geregelt.

Name	Daten-typ	L/S	Eigenschaft
IconSize	short	L+S	Symbolgröße.
ShowPosition	boolean	L+S	Positionierung anzeigen und eingeben.
ShowNavigation	boolean	L+S	Navigation ermöglichen.
ShowRecordActions	boolean	L+S	Datensatzaktionen ermöglichen.
ShowFilterSort	boolean	L+S	Filter und Sortierung ermöglichen.

Methoden bei Formularen und Kontrollfeldern

Die Datentypen der Parameter werden durch Kürzel angedeutet:

- c Nummer der Spalte des gewünschten Feldes in der Datenmenge – ab 1 gezählt
- n numerischer Wert – je nach Situation als ganze Zahl oder als Dezimalzahl
- s Zeichenkette (String); die maximale Länge ergibt sich aus der Tabellendefinition
- b *boolean* (Wahrheitswert) – *true* (wahr) oder *false* (falsch)
- d Datumswert

In einer Datenmenge navigieren

Diese Methoden gelten sowohl für ein Formular als auch für die Ergebnismenge einer Abfrage.

Mit «Cursor» ist in den Beschreibungen der Datensatzzeiger gemeint.

Name	Datentyp	Beschreibung
Prüfungen für die Position des Cursors		
isBeforeFirst	boolean	Der Cursor steht vor der ersten Zeile, wenn der Cursor nach dem Einlesen noch nicht gesetzt wurde.

Name	Datentyp	Beschreibung
isFirst	boolean	Gibt an, ob der Cursor auf der ersten Zeile steht.
isLast	boolean	Gibt an, ob der Cursor auf der letzten Zeile steht.
isAfterLast	boolean	Der Cursor steht hinter der letzten Zeile, wenn er von der letzten Zeile aus mit *next* weiter gesetzt wurde.
getRow	long	Nummer der aktuellen Zeile
Setzen des Cursors Beim Datentyp boolean steht das Ergebnis «true» dafür, dass das Navigieren erfolgreich war.		
beforeFirst	–	Wechselt vor die erste Zeile.
first	boolean	Wechselt zur ersten Zeile.
previous	boolean	Geht um eine Zeile zurück.
next	boolean	Geht um eine Zeile vorwärts.
last	boolean	Wechselt zur letzten Zeile.
afterLast	–	Wechselt hinter die letzte Zeile.
absolute(n)	boolean	Geht zu der Zeile mit der angegebenen Nummer.
relative(n)	boolean	Geht um eine bestimmte Anzahl von Zeilen weiter: bei positivem Wert von n vorwärts, andernfalls zurück.
Maßnahmen zum Status der aktuellen Zeile		
refreshRow	–	Liest die Werte der aktuellen Zeile neu ein. Nach dem Abspeichern der Zeile sind das auch die aktuellen Werte.
rowInserted	boolean	Gibt an, ob es sich um eine neue Zeile handelt.
rowUpdated	boolean	Gibt an, ob die aktuelle Zeile geändert wurde.
rowDeleted	boolean	Gibt an, ob die aktuelle Zeile gelöscht wurde.

Datenzeilen bearbeiten

Die Methoden zum Lesen stehen bei jedem Formular und bei einer Ergebnismenge zur Verfügung. Die Methoden zum Ändern und Speichern gibt es nur bei einer Datenmenge, die geändert werden kann (in der Regel also nur bei Tabellen, nicht bei Abfragen).

Name	Datentyp	Beschreibung
Maßnahmen für die ganze Zeile		
insertRow	–	Speichert eine neue Zeile.
updateRow	–	Bestätigt Änderungen der aktuellen Zeile.
deleteRow	–	Löscht die aktuelle Zeile.
cancelRowUpdates	–	Macht Änderungen der aktuellen Zeile rückgängig.
moveToInsertRow	–	Wechselt den Cursor in die Zeile für einen neuen Datensatz.
moveToCurrentRow	–	Kehrt nach der Eingabe eines neuen Datensatzes zurück zur vorherigen Zeile.

Name	Datentyp	Beschreibung
Werte lesen		

Name	***Datentyp***	***Beschreibung***
getString(c)	string	Liefert den Inhalt der Spalte als Zeichenkette. Kann auch zum Auslesen aller anderen Spalten genutzt werden und hat den Vorteil, dass leere Felder direkt bestimmt werden können. Strings können immer noch später in andere Datentypen umgewandelt werden.
getBoolean(c)	boolean	Liefert den Inhalt der Spalte als Wahrheitswert.
getByte(c)	byte	Liefert den Inhalt der Spalte als einzelnes Byte.
getShort(c)	short	Liefert den Inhalt der Spalte als ganze Zahl.
getInt(c)	integer	
getLong(c)	long	
getFloat(c)	float	Liefert den Inhalt der Spalte als Dezimalzahl von einfacher Genauigkeit.
getDouble(c)	double	Liefert den Inhalt der Spalte als Dezimalzahl von doppelter Genauigkeit. – Wegen der automatischen Konvertierung durch Basic ist dies auch für decimal- und currency-Werte geeignet.
getBytes(c)	array of bytes	Liefert den Inhalt der Spalte als Folge einzelner Bytes.
getDate(c)	Date	Liefert den Inhalt der Spalte als Datumswert.
getTime(c)	Time	Liefert den Inhalt der Spalte als Zeitwert.
getTimestamp(c)	DateTime	Liefert den Inhalt der Spalte als Zeitstempel (Datum und Zeit).
In Basic selbst werden Datums- und Zeitwerte einheitlich mit dem Datentyp DATE verarbeitet. Für den Zugriff auf die Datenmenge gibt es verschiedene Datentypen: com.sun.star.util.Date für ein Datum, com.sun.star.util.Time für eine Zeit, com.sun.star.util.DateTime für einen Zeitstempel.		
getBinaryStream(c)	Object	Liest den Inhalt eines Binärfeldes (z.B. Bild) aus. So ein Inhalt könnte als Datei gespeichert werden.
wasNull	boolean	Gibt an, ob der Wert der zuletzt gelesenen Spalte NULL war. Beim Auslesen z.B. mit getInt wird sonst für ein leeres Feld grundsätzlich '0' weitergegeben.
Werte speichern		
updateNull(c)	–	Setzt den Inhalt der Spalte c auf NULL.
updateBoolean(c,b)	–	Setzt den Inhalt der Spalte c auf den Wahrheitswert b.
updateByte(c,x)	–	Speichert in Spalte c das angegebene Byte x.
updateShort(c,n)	–	Speichert in Spalte c die angegebene ganze Zahl n.
updateInt(c,n)	–	
updateLong(c,n)	–	
updateFloat(c,n)	–	Speichert in Spalte c die angegebene Dezimalzahl n.
updateDouble(c,n)	–	
updateString(c,s)	–	Speichert in Spalte c die angegebene Zeichenkette s.

Name	Datentyp	Beschreibung
updateBytes(c,x)	–	Speichert in Spalte c das angegebene Byte-Array x.
updateDate(c,d)	–	Speichert in Spalte c das angegebene Datum d.
updateTime(c,d)	–	Speichert in Spalte c den angegebenen Zeitwert d.
updateTimestamp(c,d)	–	Speichert in Spalte c den angegeb. Zeitstempel d.

Einzelne Werte bearbeiten

Mit diesen Methoden wird über **BoundField** aus einem Kontrollfeld der Inhalt der betreffenden Spalte gelesen oder geändert. Diese Methoden entsprechen fast vollständig denen im vorigen Abschnitt; die Angabe der Spalte entfällt.

Name	Datentyp	Beschreibung
Werte lesen		
getString	string	Liefert den Inhalt der Spalte als Zeichenkette.
getBoolean	boolean	Liefert den Inhalt der Spalte als Wahrheitswert.
getByte	byte	Liefert den Inhalt der Spalte als einzelnes Byte.
getShort	short	Liefert den Inhalt der Spalte als ganze Zahl.
getInt	integer	
getLong	long	
getFloat	float	Liefert den Inhalt der Spalte als Dezimalzahl von einfacher Genauigkeit.
getDouble	double	Liefert den Inhalt der Spalte als Dezimalzahl von doppelter Genauigkeit. – Wegen der automatischen Konvertierung durch Basic ist dies auch für decimal- und currency-Werte geeignet.
getBytes	array of bytes	Liefert den Inhalt der Spalte als Folge einzelner Bytes.
getDate	Date	Liefert den Inhalt der Spalte als Datumswert.
getTime	Time	Liefert den Inhalt der Spalte als Zeitwert.
getTimestamp	DateTime	Liefert den Inhalt der Spalte als Zeitstempel (Datum und Zeit).
In Basic selbst werden Datums- und Zeitwerte einheitlich mit dem Datentyp DATE verarbeitet. Für den Zugriff auf die Datenmenge gibt es verschiedene Datentypen: com.sun.star.util.Date für ein Datum, com.sun.star.util.Time für eine Zeit, com.sun.star.util.DateTime für einen Zeitstempel.		
getBinaryStream(c)	Object	Liest den Inhalt eines Binärfeldes (z.B. Bild) aus. So ein Inhalt könnte als Datei gespeichert werden.
wasNull	boolean	Gibt an, ob der Wert der zuletzt gelesenen Spalte NULL war.
Werte speichern		
updateNull	–	Setzt den Inhalt der Spalte auf NULL.
updateBoolean(b)	–	Setzt den Inhalt der Spalte auf den Wahrheitswert b.
updateByte(x)	–	Speichert in der Spalte das angegebene Byte x.
updateShort(n)	–	Speichert in der Spalte die angegebene ganze Zahl n.

Name	Datentyp	Beschreibung
updateInt(n)	–	
updateLong(n)	–	
updateFloat(n)	–	Speichert in der Spalte die angegebene Dezimalzahl n.
updateDouble(n)	–	
updateString(s)	–	Speichert in der Spalte die angegebene Zeichenkette s.
updateBytes(x)	–	Speichert in der Spalte das angegebene Byte-Array x.
updateDate(d)	–	Speichert in der Spalte das angegebene Datum d.
updateTime(d)	–	Speichert in der Spalte den angegebenen Zeitwert d.
updateTimestamp(d)	–	Speichert in der Spalte den angegebenen Zeitstempel d.

Parameter für vorbereitete SQL-Befehle

Die Methoden, mit denen die Werte einem vorbereiteten SQL-Befehl – siehe *Vorbereitete SQL-Befehle mit Parametern* – übergeben werden, sind ähnlich denen der vorigen Abschnitte. Der erste Parameter – mit i bezeichnet – nennt seine Nummer (Position) innerhalb des SQL-Befehls.

Name	Datentyp	Beschreibung
setNull(i, n)	–	Setzt den Inhalt der Spalte auf NULL n bezeichnet den SQL-Datentyp gemäß *API-Referenz*.
setBoolean(i, b)	–	Fügt den angegebenen Wahrheitswert b in den SQL-Befehl ein.
setByte(i, x)	–	Fügt das angegebene Byte x in den SQL-Befehl ein.
setShort(i, n)	–	Fügt die angegebene ganze Zahl n in den SQL-Befehl ein.
setInt(i, n)		
setLong(i, n)		
setFloat(i, n)	–	Fügt die angegebene Dezimalzahl n in den SQL-Befehl ein.
setDouble(i, n)		
setString(i, s)	–	Fügt die angegebene Zeichenkette s in den SQL-Befehl ein.
setBytes(i, x)	–	Fügt das angegebene Byte-Array x in den SQL-Befehl ein.
setDate(i, d)	–	Fügt das angegebene Datum d in den SQL-Befehl ein.
setTime(i, d)	–	Fügt den angegebenen Zeitwert d in den SQL-Befehl ein.
setTimestamp(i, d)	–	Fügt den angegebenen Zeitstempel d in den SQL-Befehl ein.
clearParameters	–	Entfernt die bisherigen Werte aller Parameter eines SQL-Befehls.

Arbeit mit UNO-Befehlen in Formularen

Über den Makrorekorder können die Befehle ausgelesen werden, die z. B. mit den Buttons aus der Navigationsleiste der Formulare verbunden sind. Diese Befehle haben häufig eine umfassendere Funktion als die Funktionen, die sonst für Makros vorgesehen sind.

```
001  SUB FormularNeuLadenKontrollfelderAktualisieren
002     DIM oDocument AS OBJECT
003     DIM oDispatcher AS OBJECT
004     DIM Array()
```

```
005     oDocument = ThisComponent.CurrentController.Frame
006     oDispatcher = createUnoService("com.sun.star.frame.DispatchHelper")
007     oDispatcher.executeDispatch(oDocument, ".uno:Refresh", "", 0, Array())
008 END SUB
```

Über den **Dispatcher** wird das Formular neu geladen und die Formularfelder aktualisiert. Würde nur das Formular über **oForm.reload()** neu eingelesen, nachdem aus einem anderen Formular heraus der Inhalt eines Listenfeldes geändert wurde, so würde die Änderung in dem Formular nicht angezeigt. Hier müsste auch noch jedes einzelne Feld mit **oFeld.refresh()** neu eingelesen werden.

Auch das Sichern eines Datensatz ist über **.uno:RecSave** einfacher als mittels der direkten Ansprache des Formulars. Bei der direkten Ansprache des Formulars muss erst geklärt werden, ob es sich um einen neuen Datensatz handelt, für den dann ein **Insert** durchgeführt wird, oder ob es sich um einen bestehenden Datensatz handelt, der dann ein **Update** erfordert.

Eine Übersicht über verschiedene UNO-Befehle befindet sich im Anhang des Handbuches. UNO-Befehle können auch z. B. über **Extras → Anpassen → Symbolleisten → Beschreibung** ermittelt werden.

Bedienbarkeit verbessern

Als erste Kategorie werden verschiedene Möglichkeiten vorgestellt, die zur Verbesserung der Bedienbarkeit von Base-Formularen dienen. Sofern nicht anders erwähnt, sind diese Makros Bestandteil der **Beispieldatenbank** «Medien_mit_Makros.odb».

Automatisches Aktualisieren von Formularen

Oft wird in einem Formular etwas geändert und in einem zweiten, auf der gleichen Seite liegenden Formular, soll die Änderung anschließend erscheinen. Hier hilft bereits ein kleiner Codeschnipsel, um das betreffende Anzeigeformular zu aktualisieren.

```
001 SUB Aktualisieren
```

Zuerst wird einmal das Makro benannt. Die Standardbezeichnung für ein Makro ist **SUB**. Dies kann groß oder klein geschrieben sein, Mit **SUB** wird eine Prozedur durchgeführt, die nach außen in der Regel keinen Wert zurück gibt. Weiter unten wird im Gegensatz dazu einmal eine Funktion beschrieben, die im Unterschied dazu Rückgabewerte erzeugt.

Das Makro hat jetzt den Namen «*Aktualisieren*». Um sicher zu gehen, dass keine Variablen von außen eingeschleust werden, gehen viele Programmierer so weit, dass sie Basic über **Option Explicit** gleich zu Beginn mitteilen: Erzeuge nicht automatisch irgendwelche Variablen, sondern nutze nur die, die ich auch vorher definiert habe.

Deshalb werden jetzt standardmäßig erst einmal die Variablen deklariert. Bei allen hier deklarierten Variablen handelt es sich um Objekte (nicht z.B. Zahlen oder Texte), so dass der Zusatz **AS OBJECT** hinter der Deklaration steht. Um später noch zu erkennen, welchen Typ eine Variable hat, ist vor die Variablenbezeichnung ein «o» gesetzt worden. Prinzipiell ist aber die Variablenbezeichnung nahezu völlig frei wählbar.

```
002     DIM oDoc AS OBJECT
003     DIM oDrawpage AS OBJECT
004     DIM oForm AS OBJECT
```

Das Formular liegt in dem momentan aktiven Dokument. Der Behälter, in dem alle Formulare aufbewahrt werden, wird als **Drawpage** bezeichnet. Im Formularnavigator ist dies sozusagen der oberste Begriff, an den dann sämtliche Formulare angehängt werden.

Das Formular, auf das zugegriffen werden soll, ist hier mit den Namen "Anzeige" versehen. Dies ist der Name, der auch im Formularnavigator sichtbar ist. So hat z.B. das erste Formular standardmäßig erst einmal den Namen "MainForm".

```
005     oDoc = thisComponent
```

```
006     oDrawpage = oDoc.Drawpage
007     oForm = oDrawpage.forms.getByName("Anzeige")
```

Nachdem das Formular jetzt ansprechbar ist und der Punkt, an dem es angesprochen wurde, in der Variablen **oForm** gespeichert wurde, wird es jetzt mit dem Befehl **reload()** neu geladen.

```
008     oForm.reload()
009 END SUB
```

Die Prozedur hat mit **SUB** begonnen. Sie wird mit **END SUB** beendet.

Dieses Makro kann jetzt z.B. ausgelöst werden, wenn die Abspeicherung in einem anderen Formular erfolgt. Wird z.B. in einem Kassenformular an einer Stelle die Anzahl der Gegenstände und (über Barcodescanner) die Nummer eingegeben, so kann in einem anderen Formular im gleichen geöffneten Fenster hierdurch der Kassenstand, die Bezeichnung der Ware usw. nach dem Abspeichern sichtbar gemacht werden.

Filtern von Datensätzen

Der Filter selbst funktioniert ja schon ganz ordentlich in einer weiter oben beschriebenen Variante im Kapitel «Datenfilterung». Die untenstehende Variante ersetzt den Abspeicherbutton und liest die Listenfelder neu ein, so dass ein gewählter Filter aus einem Listenfeld die Auswahl in dem anderen Listenfeld einschränken kann.[29]

```
001 SUB Filter
002     DIM oDoc AS OBJECT
003     DIM oDrawpage AS OBJECT
004     DIM oForm1 AS OBJECT
005     DIM oForm2 AS OBJECT
006     DIM oFeldList1 AS OBJECT
007     DIM oFeldList2 AS OBJECT
008     oDoc = thisComponent
009     oDrawpage = oDoc.drawpage
```

Zuerst werden die Variablen definiert und auf das Gesamtformular zugegriffen. Das Gesamtformular besteht aus den Formularen "Filter" und "Anzeige". Die Listenfelder befinden sich in dem Formular "Filter" und sind mit dem Namen "Liste_1" und "Liste_2" versehen.

```
010     oForm1 = oDrawpage.forms.getByName("Filter")
011     oForm2 = oDrawpage.forms.getByName("Anzeige")
012     oFeldList1 = oForm1.getByName("Liste_1")
013     oFeldList2 = oForm1.getByName("Liste_2")
```

Zuerst wird der Inhalt der Listenfelder an das darunterliegende Formular mit **commit()** weitergegeben. Die Weitergabe ist notwendig, da ansonsten die Änderung eines Listenfeldes bei der Speicherung nicht berücksichtigt wird. Genau genommen müsste der **commit()** nur auf dem Listenfeld ausgeführt werden, das gerade betätigt wurde. Danach wird der Datensatz mit **updateRow()** abgespeichert. Es existiert ja in unserer Filtertabelle prinzipiell nur ein Datensatz, und der wird zu Beginn einmal geschrieben. Dieser Datensatz wird also laufend durch ein Update-Kommando überschrieben.

```
014     oFeldList1.commit()
015     oFeldList2.commit()
016     oForm1.updateRow()
```

Die Listenfelder sollen einander beeinflussen. Wird in einem Listenfeld z.B. eingegrenzt, dass an Medien nur CDs angezeigt werden sollen, so muss das andere Listenfeld bei den Autoren nicht noch sämtliche Buchautoren auflisten. Eine Auswahl im 2. Listenfeld hätte dann allzu häufig ein leeres Filterergebnis zur Folge. Daher müssen die Listenfelder jetzt neu eingelesen werden. Genau genommen müsste der **refresh()** nur auf dem Listenfeld ausgeführt werden, das gerade nicht betätigt wurde.

Anschließend wird das Formular2, das den gefilterten Inhalt anzeigen soll, neu geladen.

29 Siehe zu diesem Abschnitt auch die Datenbank «Beispiel_Suchen_und_Filtern.odb», die diesem Handbuch beiliegt.

```
017     oFeldList1.refresh()
018     oFeldList2.refresh()
019     oForm2.reload()
020 END SUB
```

Soll mit diesem Verfahren ein Listenfeld von der Anzeige her beeinflusst werden, so kann das Listenfeld mit Hilfe verschiedener Abfragen bestückt werden.

Die einfachste Variante ist, dass sich die Listenfelder mit ihrem Inhalt aus dem Filterergebnis versorgen. Dann bestimmt der eine Filter, aus welchen Datenbestand anschließend weiter gefiltert werden kann.

```
001 SELECT
002    "Feld_1" || ' - ' || "Anzahl" AS "Anzeige",
003    "Feld_1"
004 FROM
005    ( SELECT COUNT( "ID" ) AS "Anzahl", "Feld_1" FROM "Suchtabelle"
          GROUP BY "Feld_1" )
006 ORDER BY "Feld_1"
```

Es wird der Feldinhalt und die Trefferzahl angezeigt. Um die Trefferzahl zu errechnen, wird eine Unterabfrage gestellt. Dies ist notwendig, da sonst nur die Trefferzahl ohne weitere Information aus dem Feld in der Listbox angezeigt würde.

Das Makro erzeugt durch dieses Vorgehen ganz schnell Listboxen, die nur noch mit einem Wert gefüllt sind. Steht eine Listbox nicht auf NULL, so wird sie schließlich bei der Filterung bereits berücksichtigt. Nach Betätigung der 2. Listbox stehen also bei beiden Listboxen nur noch die leeren Felder und jeweils 1 angezeigter Wert zur Verfügung. Dies mag für eine eingrenzende Suche erst einmal praktisch erscheinen. Was aber, wenn z.B. in einer Bibliothek die Zuordnung zur Systematik klar war, aber nicht eindeutig, ob es sich um ein Buch, eine CD oder eine DVD handelt? Wurde einmal die Systematik angewählt und dann die 2. Listbox auf CD gestellt so muss, um auch die Bücher zu sehen, die 2. Listbox erst einmal wieder auf NULL gestellt werden, um dann auch die Bücher anwählen zu können. Praktischer wäre, wenn die 2. Listbox direkt die verschiedenen Medienarten anzeigen würde, die zu der Systematik zur Verfügung stehen – natürlich mit den entsprechenden Trefferquoten.

Um dies zu erreichen, wurde die folgende Abfrage konstruiert, die jetzt nicht mehr direkt aus dem Filterergebnis gespeist wird. Die Zahlen für die Treffer müssen anders ermittelt werden.

```
001 SELECT
002    COALESCE( "Feld_1" || ' - ' || "Anzahl", 'leer - ' || "Anzahl" )
          AS "Anzeige",
003    "Feld_1"
004 FROM
005    ( SELECT COUNT( "ID" ) AS "Anzahl", "Feld_1" FROM "Tabelle"
006    WHERE "ID" IN
007       ( SELECT "Tabelle"."ID" FROM "Filter", "Tabelle"
008          WHERE "Tabelle"."Feld_2" = COALESCE( "Filter"."Filter_2",
009          "Tabelle"."Feld_2" ) )
010    GROUP BY "Feld_1"
011    )
012 ORDER BY "Feld_1"
```

Diese doch sehr verschachtelte Abfrage kann auch unterteilt werden. In der Praxis bietet es sich häufig an, die Unterabfrage in einer Tabellenansicht (**'VIEW'**) zu erstellen. Das Listenfeld bekommt seinen Inhalt dann über eine Abfrage, die sich auf diesen **'VIEW'** bezieht.

Die Abfrage im Einzelnen:

Die Abfrage stellt 2 Spalten dar. Die erste Spalte enthält die Ansicht, die die Person sieht, die das Formular vor sich hat. In der Ansicht werden die Inhalte des Feldes und, mit einem Bindestrich abgesetzt, die Treffer zu diesem Feldinhalt gezeigt. Die zweite Spalte gibt ihren Inhalt an die zugrundeliegende Tabelle des Formulars weiter. Hier steht nur der Inhalt des Feldes. Die Lis-

tenfelder beziehen ihre Inhalte dabei aus der Abfrage, die als Filterergebnis im Formular dargestellt wird. Nur diese Felder stehen schließlich zur weiteren Filterung zur Verfügung.

Als Tabelle, aus der diese Informationen gezogen werden, liegt eine Abfrage vor. In dieser Abfrage werden die Primärschlüsselfelder gezählt (**SELECT COUNT("ID") AS "Anzahl"**). Dies geschieht gruppiert nach der Bezeichnung, die in dem Feld steht (**GROUP BY "Feld_1"**). Als zweite Spalte stellt diese Abfrage das Feld selbst als Begriff zur Verfügung. Diese Abfrage wiederum basiert auf einer weiteren Unterabfrage:

```
001 SELECT "Tabelle"."ID"
002 FROM "Filter", "Tabelle"
003 WHERE "Tabelle"."Feld_2" = COALESCE( "Filter"."Filter_2",
        "Tabelle"."Feld_2" )
```

Diese Unterabfrage bezieht sich jetzt auf das andere zu filternde Feld. Prinzipiell muss das andere zu filternde Feld auch zu den Primärschlüsselnummern passen. Sollten noch mehrere weitere Filter existieren, so ist diese Unterabfrage zu erweitern:

```
001 SELECT "Tabelle"."ID"
002 FROM "Filter", "Tabelle"
003 WHERE "Tabelle"."Feld_2" = COALESCE( "Filter"."Filter_2",
          "Tabelle"."Feld_2" )
004    AND
005    "Tabelle"."Feld_3" = COALESCE( "Filter"."Filter_3",
          "Tabelle"."Feld_3" )
```

Alle weiteren zu filternden Felder beeinflussen, was letztlich in dem Listenfeld des ersten Feldes, "Feld_1", angezeigt wird.

Zum Schluss wird die gesamte Abfrage nur noch nach dem zugrundeliegenden Feld sortiert.

Wie letztlich die Abfrage aussieht, die dem anzuzeigenden Formular zugrunde liegt, ist im Kapitel «Datenfilterung» nachzulesen.

Mit dem folgenden Makro kann über das Listenfeld gesteuert werden, welches Listenfeld abgespeichert werden muss und welches neu eingelesen werden muss.

Die Variablen für das Array werden in den Eigenschaften des Listenfeldes unter Zusatzinformationen abgelegt. Die erste Variable enthält dort immer den Namen des Listenfeldes selbst, die weiteren Variablen die Namen aller anderen Listenfelder, getrennt durch Kommata.

```
001 SUB Filter_Zusatzinfo(oEvent AS OBJECT)
002    DIM oDoc AS OBJECT
003    DIM oDrawpage AS OBJECT
004    DIM oForm1 AS OBJECT
005    DIM oForm2 AS OBJECT
006    DIM stTag AS String
007    stTag = oEvent.Source.Model.Tag
```

Ein Array (Ansammlung von Daten, die hier über Zahlenverbindungen abgerufen werden können) wird gegründet und mit den Feldnamen der Listenfelder gefüllt. Der erste Name ist der Name des Listenfelds, das mit der Aktion (Event) verbunden ist.

```
008    aList() = Split(stTag, ",")
009    oDoc = thisComponent
010    oDrawpage = oDoc.drawpage
011    oForm1 = oDrawpage.forms.getByName("Filter")
012    oForm2 = oDrawpage.forms.getByName("Anzeige")
```

Das Array wird von seiner Untergrenze (**LBound()**) bis zu seiner Obergrenze (**UBound()**) in einer Schleife durchlaufen. Alle Werte, die in den Zusatzinformationen durch Komma getrennt erschienen, werden jetzt nacheinander weitergegeben.

```
013    FOR i = LBound(aList()) TO Ubound(aList())
014       IF i = 0 THEN
```

Das auslösende Listenfeld muss abgespeichert werden. Es hat die Variable **aList(0)**. Zuerst wird die Information des Listenfeldes auf die zugrundeliegende Tabelle übertragen, dann wird der Datensatz gespeichert.

```
015             oForm1.getByName(aList(i)).commit()
016             oForm1.updateRow()
017         ELSE
```

Die anderen Listenfelder müssen neu eingelesen werden, da sie ja in Abhängigkeit vom ersten Listenfeld jetzt andere Werte abbilden.

```
018             oForm1.getByName(aList(i)).refresh()
019         END IF
020     NEXT
021     oForm2.reload()
022 END SUB
```

Die Abfragen für dieses besser nutzbare Makro sind natürlich die gleichen wie in diesem Abschnitt zuvor bereits vorgestellt.

Daten über den Formularfilter filtern

Alternativ zu dieser Vorgehensweise ist es auch möglich, die Filterfunktion des Formulars direkt zu bearbeiten.

```
001 SUB FilterSetzen
002     DIM oDoc AS OBJECT
003     DIM oForm AS OBJECT
004     DIM oFeld  AS OBJECT
005     DIM stFilter As String
006     oForm = thisComponent.Drawpage.Forms.getByName("MainForm")
007     oFeld = oForm.getByName("Filter")
008     stFilter = oFeld.Text
009     oForm.filter = " UPPER(""Name"") LIKE '%'||'" + UCase(stFilter) + "'||'%'"
010     oForm.ApplyFilter = TRUE
011     oForm.reload()
012 End Sub
```

Das Feld wird im Formular aufgesucht, der Inhalt ausgelesen. Der Filter wird entsprechend gesetzt. Die Filterung wird angeschaltet und das Formular neu geladen.

```
001 SUB FilterEntfernen
002     DIM oForm AS OBJECT
003     oForm = thisComponent.Drawpage.Forms.getByName("MainForm")
004     oForm.ApplyFilter = False
005     oForm.reload()
006 END SUB
```

Die Beendigung des Filters kann natürlich auch über die Navigationsleiste erfolgen. In diesem Fall wird einfach ein weiteres Makro genutzt.

Über diese Filterfunktion kann ein Formular auch direkt mit einem Filter z. B. für nur einen Datensatz gestartet werden. Aus dem startenden Formular wird ein Wert (z.B. ein Primärschlüssel für den aktuellen Datensatz) ausgelesen und an das Zielformular als Filterwert weiter gegeben.

Filterdialog über einen Button starten

Wird ein Formular geöffnet, so stehen über die Navigationsleiste verschiedene Filtermöglichkeiten zur Verfügung. Während in Tabellen, Abfaregn und über das Formularkontrollelement ein Filterdialog zur Verfügung steht, ist dieser über die Navigationsleiste des Formularfensters durch den Formularbasierten Filter ersetzt worden. Auch über eine einfache Schaltfläche lässt sich dieser Dialog nicht starten. Hier kann zur Zeit nur mit einem Makro nachgeholfen werden.

```
001 SUB FilterDefault(oEvent AS OBJECT)
002     oForm = oEvent.Source.Model.Parent
```

```
003     oController = thisComponent.getCurrentController()
004     oFormController = oController.getFormController(oForm)
005     oFormController.FormOperations.execute(16)
006 END SUB
```

Das Makro wird von der Schaltfläche aus über **Ereignisse → Aktion ausführen** gestartet. Über das auslösende Ereignis wird das zugrundeliegende Formular ermittelt. Anschließend wird über den Controller die Kontrolle über die Elemente des Formulars übernommen, die dort prinzipiell zur Verfügung stehen. Zu den möglichen FormOperations gehört mit der Short-Variablen 16 der Dialog zum Setzen eines Filters.

Der Filterdialog kann über den Button gestartet werden und die Filterundg wird mit OK direkt vollzogen.

Die folgenden Variablen stehen über die com::sun::star::form::runtime::FormFeature Constant Group Reference zur Verfügung:

Nr.	***Befehl***	***Bedeutung***

1	MoveAbsolute
	`005 DIM v as new com.sun.star.beans.NamedValue`
	`006 v.Name = "Position"`
	`007 v.Value = 5`
	`008 oFormController.FormOperations.executeWith Arguments(1, Array(v))`
		Gehe zu dem Datensatz, der über ein Array angegeben werden muss

2	TotalRecord
	`005 oState = oFormController.FormOperations.getState(2)`
	`006 Msgbox "Gesamtzahl der Datensätze: "& cStr(oState.State)`
		Zeige die Anzahl der gesamten Datensätze an – leider auch hier nur wie in der Navigationsleiste mit '41 *'

3	MoveToFirst	Gehe zum ersten Datensatz
4	MoveToPrevious	Gehe zum vorhergehenden Datensatz
5	MoveToNext	Gehe zum nächsten Datensatz
6	MoveToLast	Gehe zum letzten Datensatz
7	MoveToInsertRow	Gehe zum Einfügen eines neuen Datensatzes
8	SaveRecordChanges	Speichere die Datensatzänderung
9	UndoRecordChanges	Mache die Änderungen rückgän-

		gig
10	DeleteRecord	Lösche den Datensatz
11	ReloadForm	Lies das Formular neu ein
12	SortAscending	Sortiere aufsteigend
13	SortDescending	Sortiere absteigend
14	InteractiveSort	Öffne Dialog zum Sortieren
15	AutoFilter	Automatischer Filter, vorher in ein gewünschtes Feld klicken.
16	InteractiveFilter	Öffne Dialog zum Filtern
17	ToggleApplyFilter	Stelle den Filter ein oder aus
18	RemoveFilterAndSort	Nimm Filterung und Sortierung zurück
19	RefreshCurrentControl	Lies den Inhalt des Listenfeldes oder Kombinationsfeldes neu ein.

Durch Datensätze mit der Bildlaufleiste scrollen

Die Bildlaufleiste lässt sich nur über Makros nutzen. Das folgende Beispiel[30] zeigt auf, wie mit so einer Bildlaufleiste durch Datensätze gescrollt werden kann. Die Bildlaufleiste kann dann statt der Navigationsleiste zur Navigation durch die Datensätze genutzt werden.

```
001 GLOBAL loPos AS LONG
```

Die Position des aktuellen Datensatzes wird als globale Variable gespeichert, damit sie aus allen Prozeduren gelesen und in allen Prozeduren geändert werden kann.

```
001 SUB MaxRow(oEvent AS OBJECT)
002    'Auslösen durch das Formular "Beim Laden" und "Nach der Datensatzaktion"
003    DIM oForm AS OBJECT
004    DIM oScrollField AS OBJECT
005    DIM loMax AS LONG
006    oForm = oEvent.Source
007    oScrollField = oForm.getByName("Bildlaufleiste")
008    loPos = oForm.getRow
009    oForm.last
010    loMax = oForm.getRow
011    oForm.absolute(loPos)
012    oScrollField.ScrollValueMax = loMax
013    oScrollField.ScrollValue = loPos
014 END SUB
```

Die Gesamtzahl der Datensätze kann nicht über die Funktion **RowCount** des Formulars ermittelt werden, da diese Funktion nur die Datensätze zählt, die bereits in den Cache geladen wurden. Deswegen wird hier zuerst die aktuelle Zeilennummer des Formulars ausgelesen, dann ans Ende der einzulesenden Daten gesprungen und die dortige Zeilennummer als Maximalwert ermittelt. Anschließend muss wieder auf die ursprüngliche Zeile mit **oForm.absolute()** zurückgesprungen werden.

Der Bildlaufleiste wird der ermittelte maximale Wert als **ScrollValueMax** und die aktuelle Position als **loPos** mitgeteilt. Geschieht das letzte nicht, so stimmt die Position innerhalb der Bildlaufleiste nicht unbedingt mit dem aktuellen Datensatz überein.

```
001 SUB Navigation(oEvent AS OBJECT)
002    'Auslösen durch das Formular "Nach dem Datensatzwechsel"
003    'Synchronisiert die Scrollstellung mit der Position des Datensatzes
```

30 Die Beispieldatenbank «Beispiel_Datensatz_scrollbar.odb» ist den Beispieldatenbanken für dieses Handbuch beigefügt.

```
004     DIM oForm AS OBJECT
005     DIM oScrollField AS OBJECT
006     oForm = oEvent.Source
007     oScrollField = oForm.getByName("Bildlaufleiste")
008     loPos = oForm.getRow
009     IF loPos = 0 THEN
010        'Bei einem neuen Datensatz wird über getRow '0' ermittelt.
011        'In dem Datensatzanzeiger soll stattdessen die maximale Zahl an Zeilen
012        ''RowCount' um '1' erhöht werden
013        loPos = oForm.RowCount + 1
014     END IF
015     oScrollField.ScrollValue = loPos
016  END SUB
```

Wird durch die Datensätze navigiert, so muss die Anzeige der Bildlaufleiste und die Anzeige in der Navigationsleiste immer übereinstimmen. Deshalb wird nach dem Datensatzwechsel immer die aktuelle Zeilennummer ermittelt. Für die aktuelle Zeilennummer wird '0' ausgegeben, wenn der Cursor zur Neuaufnahme eines Datensatzes über die letzte Zeile hinaus geht. In diesem Fall soll aber die Bildlaufleiste nicht auf die Startposition zurückspringen sondern wie die Navigationsleiste um '1' oberhalb des bisherigen maximalen Wertes positioniert werden.

```
001  SUB FormScroll(oEvent AS OBJECT)
002     'Auslösen durch die Bildlaufleiste "Beim Justieren"
003     DIM oForm AS OBJECT
004     DIM oScrollAction AS OBJECT
005     oScrollAction = oEvent.Source
006     oForm = oScrollAction.Model.Parent
007     loPos = oScrollAction.getValue()
008     oForm.absolute(loPos)
009  END SUB
```

In dieser Prozedur wird aus der Bildlaufleiste ein neuer Wert für die Zeile des Formulars ermittelt. Über **`getValue()`** wird der Wert aus der Bildlaufleiste ausgelesen und dem Formular über **`oForm.absolute(loPos)`** zugewiesen.

Daten aus Textfeldern auf SQL-Tauglichkeit vorbereiten

Beim Speichern von Daten über einen SQL-Befehl können vor allem Hochkommata **`(')`** Probleme bereiten, wie sie z.B. in Namensbezeichnungen wie O'Connor vorkommen können. Dies liegt daran, dass Texteingaben in Daten in **`''`** eingeschlossen sind. Hier muss eine Funktion eingreifen und die Daten entsprechend vorbereiten.

```
001  FUNCTION String_to_SQL(st AS STRING)
002     IF InStr(st,"'") THEN
003        st = Replace(st,"'","''")
004     END IF
005     String_to_SQL = st
006  END FUNCTION
```

Es handelt sich hier um eine Funktion. Eine Funktion nimmt einen Wert auf und liefert anschließend auch einen Gegenwert zurück.

Der übergebende Text wird zuerst einmal daraufhin untersucht, ob er ein Hochkomma enthält. Ist dies der Fall, so wird so wird das eine Hochkomma durch zwei Hochkommata ersetzt. Der SQL-Code wird so maskiert.

Die Funktion übergibt ihr Ergebnis durch den folgenden Aufruf:

```
001  stTextneu = String_to_SQL(stTextalt)
```

Es wird also einfach nur die Variable stTextalt überarbeitet und der entsprechende Wert wieder in der Variablen stTextneu gespeichert. Dabei müssen die Variablen gar nicht unterschiedlichen heißen. Der Aufruf geht praktischer direkt mit:

```
001  stText = String_to_SQL(stText)
```

Diese Funktion wird in den nachfolgenden Makros immer wieder benötigt, damit Hochkommata auch über SQL abgespeichert werden können.

Beliebige SQL-Kommandos speichern und bei Bedarf ausführen

Über das Abfragemodul können nur Abfragen gestartet werden. SQL-Code, der auch verändernd auf Daten wirkt, ist dort nicht ausführbar, sofern nicht der Datenbanktreiber das, wie bei dem direkten Treiber für MySQL, zulässt. Der über **Extras → SQL** zur Verfügung stehende Editor lässt zwar die Ausführung von Code zu, bietet aber keine Speichermöglichkeit. Laufend wiederkehrende Befehle müssen so umständlich irgendwo separat abgespeichert werden und können dann in den Editor über die Zwischenablage kopiert werden. Das folgende Makro schafft hier Abhilfe.[31]

Der für eine Operation wie **UPDATE**, **DELETE** oder **INSERT** gedachte Code wird in einer Tabelle abgelegt, deren erstes Feld den Primärschlüssel **"ID"** und deren zweites Feld den SQL-Code enthält. Der jeweils aktuelle Datensatz wird in einem Formular ausgewählt und dann über einen Button ausgeführt.

```
001 SUB ChangeData(oEvent AS OBJECT)
002    DIM oConnection AS OBJECT
003    DIM oForm AS OBJECT
004    DIM stSql AS STRING
005    DIM oSql_Statement AS OBJECT
006    DIM inValue AS INTEGER
007    oForm = oEvent.Source.Model.Parent
008    oConnection = oForm.activeConnection()
009    stSQL = oForm.getString(2)
010    inValue = MsgBox("Soll der SQL-Code" & CHR(13) & stSQL & CHR(13) &
          "ausgeführt werden?", 20, "SQL-Code ausführen")
011    IF inValue = 6 THEN
012       oSQL_Statement = oConnection.createStatement()
013       oSQL_Statement.execute(stSql)
014    END IF
015 END SUB
```

Nach der Definition der Variablen wird das aktuelle Formular über das auslösende Ereignis des Buttons ermittelt. Die Verbindung zur Datenbank wird aus dem Formular ausgelesen. Als zweites Feld steht in der Tabelle der gewünschte SQL-Code.

Zur Sicherheit wird dieser SQL-Code zuerst in einer Messagebox noch einmal dargestellt. Wird hier nicht mit Ja bestätigt, so wird der Code nicht ausgeführt. Diese Bestätigung wird über den Rückgabewert der Messagebox ermittelt (**6** entspricht Ja). Anschließend wird zuerst die Verbindung für das Statement hergestellt und dann das Statement ausgeführt.

Hinweis

Zur Ausführung eines SQL-Kommados stehen die folgenden Methoden zur Verfügung[32]:
executeQuery(stSql) erwartet eine Rückgabe eines Wertes
executeUpdate(stSql) für **INSERT**, **UPDATE** oder **DELETE** erwartet keine Rückgabe
execute(stSql) kann beliebig viele Rückgaben verarbeiten
Das ist in Zusammenhang mit der Firebird Datenbank wichtig, weil die zur Zeit (LO 7.4) so reagiert, dass z. B. Bei **ALTER**-Anweisungen eine (unverständliche) Rückmeldung erfolgt und damit die obige Prozedur scheinbar einen Fehler erzeugt, wenn sie mit **executeUpdate(stSql)** abläuft.

Werte in einem Formular vorausberechnen

Werte, die über die Datenbankfunktionen berechnet werden können, werden in der Datenbank nicht extra gespeichert. Die Berechnung erfolgt allerdings nicht während der Eingabe im Formular, sondern erst nachdem der Datensatz abgespeichert ist. Solange das Formular aus

31 Die Beispieldatenbank «Beispiel_InsertUpdateDelete_SQL.odb» ist den Beispieldatenbanken für dieses Handbuch beigefügt.
32 *https://api.libreoffice.org/docs/idl/ref/interfacecom_1_1sun_1_1star_1_1sdbc_1_1XStatement.html*

einem Tabellenkontrollfeld besteht, mag das nicht so viel ausmachen. Schließlich kann direkt nach der Eingabe ein berechneter Wert ausgelesen werden. Bei Formularen mit einzelnen Feldern bleibt der vorherige Datensatz aber nicht unbedingt sichtbar. Hier bietet es sich an, die Werte, die sonst in der Datenbank berechnet werden, direkt in entsprechenden Feldern anzuzeigen.[33]

Die folgenden drei Makros zeigen, wie so etwas vom Prinzip her ablaufen kann. Beide Makros sind mit dem Verlassen bestimmter Felder gekoppelt. Dabei ist auch berücksichtigt, dass hinterher eventuell Werte in einem bereits bestehenden Feld geändert werden.

```
001 SUB Berechnung_ohne_MWSt(oEvent AS OBJECT)
002    DIM oForm AS OBJECT
003    DIM oFeld AS OBJECT
004    DIM oFeld2 AS OBJECT
005    oFeld = oEvent.Source.Model
006    oForm = oFeld.Parent
007    oFeld2 = oForm.getByName("Preis_ohne_MWSt")
008    oFeld2.BoundField.UpdateDouble(oFeld.getCurrentValue / 1.19)
009    IF NOT IsEmpty(oForm.getByName("Anzahl").getCurrentValue()) THEN
010       Berechnung_gesamt2(oForm.getByName("Anzahl"))
011    END IF
012 END SUB
```

Ist in einem Feld «Preis» ein Wert eingegeben, so wird beim Verlassen des Feldes das Makro ausgelöst. Im gleichen Formular wie das Feld «Preis» liegt das Feld «Preis_ohne_MWSt». Für dieses Feld wird mit **BoundField.UpdateDouble** der berechnete Preis ohne Mehrwertsteuer festgelegt. Das Datenfeld dazu entstammt einer Abfrage, bei der vom Prinzip her die gleiche Berechnung, allerdings bei bereits gespeicherten Daten, durchgeführt wird. Auf diese Art und Weise wird der berechnete Wert sowohl während der Eingabe als auch später während der Navigation durch die Datensätze sichtbar, ohne abgespeichert zu werden.

Ist bereits im Feld «Anzahl» ein Wert enthalten, so wird eine Folgerechnung auch für die damit verbundenen Felder durchgeführt.

```
001 SUB Berechnung_gesamt(oEvent AS OBJECT)
002    oFeld = oEvent.Source.Model
003    Berechnung_gesamt2(oFeld)
004 END SUB
```

Diese kurze Prozedur dient nur dazu, die Auslösung der Folgeprozedur vom Verlassen des Formularfeldes «Anzahl» weiter zu geben. Die Angabe könnte genauso gut mit Hilfe der Bestimmung des Feldes über die Drawpage in der Folgeprozedur integriert werden.

```
001 SUB Berechnung_gesamt2(oFeld AS OBJECT)
002    DIM oForm AS OBJECT
003    DIM oFeld2 AS OBJECT
004    DIM oFeld3 AS OBJECT
005    DIM oFeld4 AS OBJECT
006    oForm = oFeld.Parent
007    oFeld2 = oForm.getByName("Preis")
008    oFeld3 = oForm.getByName("Preis_gesamt_mit_MWSt")
009    oFeld4 = oForm.getByName("MWSt_gesamt")
010    oFeld3.BoundField.UpdateDouble(oFeld.getCurrentValue * oFeld2.getCurrentValue)
011    oFeld4.BoundField.UpdateDouble(oFeld.getCurrentValue * oFeld2.getCurrentValue -
          oFeld.getCurrentValue * oFeld2.getCurrentValue / 1.19)
012 END SUB
```

Diese Prozedur ist lediglich eine Prozedur, bei der mehrere Felder berücksichtigt werden sollen. Die Prozedur wird aus einem Feld «Anzahl» gestartet, das die Anzahl bestimmter gekaufter Waren vorgeben soll. Mit Hilfe dieses Feldes und des Feldes «Preis» wird jetzt der «Preis_gesamt_mit_MWSt» und die «MWSt_gesamt» berechnet und in die entsprechenden Felder übertragen.

Nachteil in den Prozeduren und auch bei Abfragen: Der Steuersatz wird hier fest einprogrammiert. Besser wäre eine entsprechende Angabe dazu in Verbindung mit dem Preis, da ja Steuer-

33 Siehe hierzu die Beispieldatenbank «Beispiel_Direktberechnung_im Formular.odb»

sätze unterschiedlich sein können und auch nicht immer konstant sind. In dem Fall müsste eben der Mehrwertsteuersatz aus einem Feld des Formulars ausgelesen werden.

Die aktuelle Office-Version ermitteln

Mit der Version 4.1 sind Änderungen bei Listenfeldern und Datumswerten vorgenommen worden, die es erforderlich machen, vor der Ausführung eines Makros für diesen Bereich zu erkunden, welche Office-Version denn nun verwendet wird. Dazu dient der folgende Code:

```
001 FUNCTION OfficeVersion()
002    DIM aSettings, aConfigProvider
003    DIM aParams2(0) AS NEW com.sun.star.beans.PropertyValue
004    DIM sProvider$, sAccess$
005    sProvider = "com.sun.star.configuration.ConfigurationProvider"
006    sAccess = "com.sun.star.configuration.ConfigurationAccess"
007    aConfigProvider = createUnoService(sProvider)
008    aParams2(0).Name = "nodepath"
009    aParams2(0).Value = "/org.openoffice.Setup/Product"
010    aSettings = aConfigProvider.createInstanceWithArguments(sAccess, aParams2())
011    OfficeVersion() = array(aSettings.ooName,aSettings.ooSetupVersionAboutBox)
012 END FUNCTION
```

Diese Funktion gibt ein Array wieder, das als ersten Wert z.B. "LibreOffice" und als zweiten Wert die detaillierte Version, z.B. "4.1.5.2" ausgibt.

Wert von Listenfeldern ermitteln

Mit LibreOffice 4.1 wird der Wert, den Listenfelder an die Datenbank weitergeben, über «CurrentValue» ermittelt. In Vorversionen, auch OpenOffice oder AOO, ist dies nicht der Fall. Die folgende Funktion soll dem Rechnung tragen. Die ermittelte LO-Version muss daraufhin untersucht werden, ob sie nach der Version 4.0 entstanden ist.

```
001 FUNCTION ID_Ermittlung(oFeld AS OBJECT) AS INTEGER
002    a() = OfficeVersion()
003    IF a(0) = "LibreOffice" AND ((LEFT(a(1),1) = 4 AND RIGHT(LEFT(a(1),3),1) > 0)
          OR LEFT(a(1),1) > 4) THEN
004       stInhalt = oFeld.currentValue
005    ELSE
```

Vor LO 4.1 wird der Wert, der weiter gegeben wird, aus der Werteliste des Listenfeldes ausgelesen. Der sichtbar ausgewählte Datensatz ist SelectedItems(0). '0', weil auch mehrere Werte in einem Listenfeld ausgewählt werden könnten.

```
006       stInhalt = oFeld.ValueItemList(oFeld.SelectedItems(0))
007    END IF
008    IF IsEmpty(stInhalt) THEN
```

Mit -1 wird ein Zahlenwert weiter gegeben, der nicht als AutoWert verwendet wird, also in vielen Tabellen nicht als Fremdschlüssel existiert.

```
009       ID_Ermittlung = -1
010    ELSE
011       ID_Ermittlung = Cint(stInhalt)
```

Der Text wird in eine Integer-Variable umgewandelt.

```
012    END IF
013 END FUNCTION
```

Die Funktion gibt den Wert als Integer wieder. Meist werden für Primärschlüssel ja automatisch hoch zählende Integer-Werte verwendet. Für eine Verwendung von Fremdschlüsseln, die diesem Kriterium nicht entsprechen, muss die Ausgabe der Variablen entsprechend angepasst werden.

Der angezeigte Wert eines Listenfeldes lässt sich weiterhin über die Ansicht des Feldes ermitteln:

```
001 SUB Listenfeldanzeige
002    DIM oDoc AS OBJECT
003    DIM oForm AS OBJECT
004    DIM oListbox AS OBJECT
005    DIM oController AS OBJECT
006    DIM oView AS OBJECT
007    oDoc = thisComponent
008    oForm = oDoc.Drawpage.Forms(0)
009    oListbox = oForm.getByName("Listenfeld")
010    oController = oDoc.getCurrentController()
011    oView = oController.getControl(oListbox)
012    print "Angezeigter Inhalt: " & oView.SelectedItem
013 END SUB
```

Es wird über den Controller auf die Ansicht des Formulars zugegriffen. Damit wird ermittelt, was auf der sichtbaren Oberfläche tatsächlich erscheint. Der ausgewählte Wert ist der **SelectedItem**.

Auch der direkte Weg über die Liste der zur Anzeige zur Verfügung stehenden Inhalte ist möglich:

```
001 SUB ListView
002    oField = thisComponent.Drawpage.Forms(0).getByName("Listfield")
003    stView = oField.StringItemList(oField.SelectedItems(0))
004    msgbox stView
005 END SUB
```

Die Position des ausgewählten Werts steht in **oField.SelectedItems(0)**, da es sich bei Datenbanken um ein Listenfeld handelt, das keine Mehrfachselektion erlaubt. Mit der Position wird in der Liste der zur Anzeige zur Verfügung stehenden Inhalte ermittelt, welcher Inhalt an dieser Stelle steht. Beide Listen sind Arrays, so dass sie die entsprechende Zahlenangabe für die Position, beginnend mit 0, benötigen.

Listenfelder durch Eingabe von Anfangsbuchstaben einschränken

Manchmal kann es vorkommen, dass der Inhalt für Listenfelder unübersichtlich groß wird. Damit eine Suche schneller zum Erfolg führt, wäre es sinnvoll, hier den Inhalt des Listenfeldes nach Eingabe eines oder mehrerer Buchstaben einzugrenzen. Das Listenfeld selbst wird mit einem SQL-Befehl versehen, der eine Filterung unterschiedlicher Listenfeldabfragen vereinfacht[34]. Hier könnte z.B. stehen:

```
001 SELECT Tab".* FROM (SELECT "Name" AS "Field", "ID", "Name" AS "Sort" FROM
    "Tabelle") AS "Tab" ORDER BY "Field"
```

Dadurch wird klar, an welcher Stelle der Code später durch die Filterung ersetzt werden muss. Die Bedingungen der inneren Abfrage bleiben unberührt. Soll zuerst nur ein begrenzter Teil des gesamten Inhaltes angezeigt werden, so kann auch ein **LIMIT** für die Abfrage gesetzt werden. Durch die Zuweisung eines Alias in der Unterabfrage kann auf die Ermittlung von entsprechenden Feldnamen verzichtet werden. «Field» steht hier immer für das Feld, das gefiltert werden soll, «Sort» immer für die Sortierung. Das vereinfacht den Code für das Makro erheblich.

Das folgende Makro ist dafür an **Eigenschaften: Listenfeld → Ereignisse → Taste losgelassen** gekoppelt.

```
001 GLOBAL stListStart AS STRING
002 GLOBAL lZeit AS LONG
```

Zuerst werden globale Variablen erstellt. Diese Variablen sind notwendig, damit nicht nur nach einem Buchstaben, sondern nach dem Betätigen weiterer Tasten schließlich auch nach einer Buchstabenkombination gesucht werden kann.

In der globalen Variablen **stListStart** werden die Buchstaben in der eingegebenen Reihenfolge gespeichert.

34 Die Datenbank «Beispiel_Suchen_Filtern.odb» ist in den zusätzlichen Datenbanken mit besonderer Beschreibung jeder einzelnen Datenbank enthalten.

Die globale Variable **lZeit** wird mit der aktuellen Zeit in Sekunden versorgt. Bei einer längeren Pause zwischen den Tastatureingaben soll die Variable **stListStart** wieder zurückgesetzt werden können. Deswegen wird jeweils der Zeitunterschied zur vorhergehenden Eingabe abgefragt.

```
001 SUB ListFilter(oEvent AS OBJECT)
002    oFeld = oEvent.Source.Model
003    IF oEvent.KeyCode < 538 OR oEvent.KeyCode = 1283 OR oEvent.KeyCode = 1284
004       OR oEvent.KeyCode = 1281 THEN
```

Das Makro wird durch einen Tastendruck ausgelöst. Eine Taste hat innerhalb der API einen bestimmten Zahlencode, der unter *com>sun>star>awt>Key* nachgeschlagen werden kann. Sonderzeichen wie das «ä», «ö» und «ü» haben den **KeyCode** 0, alle anderen Schriftzeichen und Zahlen haben einen **KeyCode** kleiner als 538. Den **KeyCode** 1283 belegt ← (Rücktaste). Wird dieser Code mit ausgelesen, so können auch Korrekturen durchgeführt werden. Mit dem **KeyCode** 1284 wird auch die Leertaste in die möglichen Zeichen aufgenommen. Hinter dem **KeyCode** 1281 steckt *ESC*. Die Taste soll zum Zurücksetzen des Listenfeldes dienen.

Die Abfrage des **KeyCode** ist hier wichtig, da auch der Schritt mit der Tabulatortaste auf das Auswahlfeld natürlich das Makro auslöst. Der **KeyCode** für die Tabulatortaste liegt allerdings bei 1282, so dass der weitere Code der Prozedur hier nicht ausgeführt wird.

```
005       DIM stSql(0) AS STRING
```

Der SQL-Code für das Listenfeld wird in einem Array gespeichert. Das Array hat im Falle des SQL-Codes aber nur ein Datenfeld. Deshalb ist das Array direkt auf **stSql(0)** begrenzt.

Entsprechend muss auch beim Auslesen des SQL-Codes aus dem Listenfeld darauf geachtet werden, dass der SQL-Code nicht direkt als Text zugänglich ist. Stattdessen ist der Code in einem Array als einziger Eintrag vorhanden: **oFeld.ListSource(0)**.

Der SQL-Code wird nach der Deklaration der Variablen für die weitere Verwendung aufgesplittet. Der Code wird direkt nach der Klammer für die Unterabfrage aufgetrennt. Die Unterabfrage hat hier den Alias **AS "Tab"**.

```
006       DIM stText AS STRING
007       DIM stFeld AS STRING
008       DIM stQuery AS STRING
009       DIM ar()
010       ar() = Split(oFeld.ListSource(0), " AS ""Tab""")
```

Für die Filterung der inneren Abfrage wird eine Bedingung erforderlich. An die bestehende Abfareg wird also zusammen mit dem Alias **WHERE** angehängt:

```
011       stQuery = ar(0) & " AS ""Tab"" WHERE"
012       IF oEvent.KeyCode = 1281 THEN  'Taste ESC
013          stListStart = "" 'Bei ESC wieder den gesamten Inhalt anzeigen
014       ELSEIF lZeit > 0 AND Timer() - lZeit < 5 THEN
015          stListStart = stListStart & oEvent.KeyChar
016       ELSE
017          stListStart = oEvent.KeyChar
018       END IF
019       lZeit = Timer()
```

Zuerst wird die Bedingung abgefragt, ob *ESC* gedrückt wurde. Hier kann nicht der **KeyChar** an die Abfrage weiter gegeben werden. Stattdessen muss ein leerer String weitergegeben werden, der die Suchergebnisse auf alle Daten ausdehnt. Ist bereits einmal eine Zeit in der globalen Variablen abgespeichert worden und beträgt die Distanz zu dieser Zeit zum Zeitpunkt der Eingabe weniger als 5 Sekunden, so wird der eingegebene Buchstabe an die vorher eingegebenen Buchstaben angehängt. Anderenfalls wird der eingegebene Buchstabe als einzige (neue) Eingabe verstanden. Das Listenfeld wird dann einfach neu nach dem entsprechenden Buchstaben gefiltert. Anschließend wird die aktuelle Zeit wieder in der globalen Variablen **lZeit** gespeichert.

```
020       stText = LCase( stListStart & "%")
021       stSql(0) = stQuery+"LOWER(""Field"") LIKE '"+stText+"' ORDER BY ""Sort"""
022       oFeld.ListSource = stSql
```

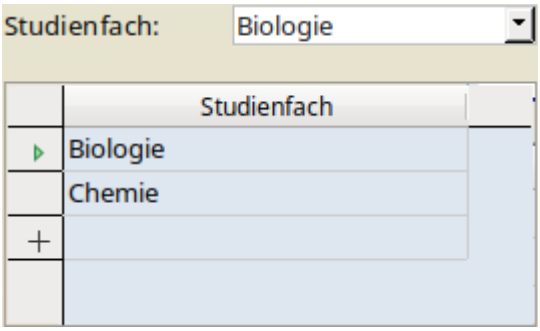
```
023         oFeld.refresh
024       END IF
025   END SUB
```

Der SQL-Code wird schließlich zusammengefügt. Die Kleinschreibweise des Feldinhaltes wird mit der Kleinschreibweise des eingegebenen Buchstabens verglichen. Der Code wird dem Listenfeld hinzugefügt und das Listenfeld aufgefrischt, so dass nur noch der gefilterte Inhalt nachgeschlagen werden kann.

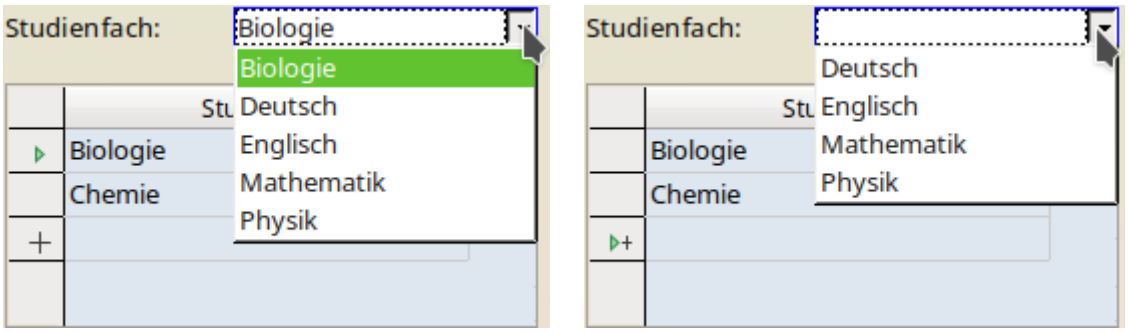
Soll eventuell nicht nach den Anfangsbuchstaben sondern ganz allgemein im angezeigten Inhalt gesucht werden, so muss lediglich die Variable «stText» von **`LCase( stListStart & "%")`** zu **`Lcase("%" & stListStart & "%")`** umgewandelt werden.

Listenfelder mit eingeschränkter Auswahl

Manchmal darf von einem Listenfeld nicht mehrmals der gleiche Wert ausgewählt werden. So sollte z. B. eine Person nicht doppelt einer Gruppe zugewiesen werden oder, wie im folgenden Screenshot, ein Studienfach mehrmals in der Liste der Studienfächer auftauchen.



In dem Screenshot wird das untere Tabellenkontrollfeld durch das darüberstehende Listenfeld mit Inhalten versehen. Ist das Listenfeld in dem Tabellenkontrollfeld eingebaut, so kann ein «Studienfach» mehrfach ausgewählt werden. Die Tabellenkonstruktion ist aber zur Sicherheit bereits so erstellt, dass der eingefügte Wert zusammen mit einem Wert des darüber liegenden Hauptformulars den Primärschlüssel bildet. Wird jetzt «Biologie» noch einmal gewählt, so hängt die Eingabe. Keine Rückmeldung, nur über *ESC* geht es wieder aus dieser Situation heraus.



Das Listenfeld zeigt jetzt nur die möglichen Einträge an. Das ist sowohl der aktuelle Datensatz als auch die anderen Fächer, die noch nicht ausgewählt wurden. Das Tabellenkontrollfeld kann nur zum Löschen von Datensätzen genutzt werden.

Wird ein neuer Datensatz bearbeitet, so stehen nur die zusätzlich verfügbaren Inhalte zur Auswahl. Die Fächer «Biologie» und «Chemie» erscheinen hier nicht.

Eine solche Konstruktion ist nicht innerhalb eines Tabellenkontrollfeldes möglich. Die Listenfelder werden hier nicht gleichbleibend aktualisiert, so dass zwischendurch die aktuell ausgewählten Fächer verschwinden würden, obwohl bereits ein entsprechender Fremdschlüssel abgespeichert wurde.

Der in dem Listenfeld stehende SQL-Code für die Anzeige der Liste sieht so aus:

```
001 SELECT "Tab".* FROM
002    (SELECT "Studienfach" AS "Field",
003    "ID" AS "Key",
004    "Studienfach" AS "Sort" FROM "tbl_Studienfach") AS "Tab"
005 ORDER BY "Sort"
```

In der Unterabfrage wird über ein Alias dem anzuzeigenden Inhalt, dem weiter zu gebendem Schlüsselwert und der beabsichtigten Sortierung ein fester Begriff zugeordnet. So kann auch bei komplizierterem Code über z. B. mehrere Tabellen immer Bezug auf diese Schlüsselpositionen genommen werden: **"Field"**, **"Key"** und **"Sort"**. Der eigentliche Code für das Listenfeld kann über den Alias für die Unterabfrage **AS "Tab"** von den über Makros hinzugefügten Bedingungen abgetrennt werden.

In den Zusatzinformationen zu dem Listenfeld steht ein Code, der in der Prozedur «Listenfeldfilter» zur Anzeige des zu dem aktuellen Datensatz passenden Listeninhaltes benötigt wird:

```
001 SELECT "Studienfach_ID" FROM "tbl_LehrerIn_Studienfach" WHERE "L_ID"
```

Die folgende Prozedur wird über **Formulareigenschaften → Ereignisse → Nach dem Datensatzwechsel** bei jedem neuen Datensatz wieder aufgerufen.

```
001 SUB Listenfeldfilter(oEvent AS OBJECT)
002    DIM oListField AS OBJECT
003    DIM oForm AS OBJECT
004    DIM stID AS STRING
005    DIM stIDCurrent AS STRING
006    DIM stNotIn AS STRING
007    DIM stSql(0) AS STRING
008    oForm = oEvent.Source
009    IF hasUnoInterfaces(oForm, "com.sun.star.form.XForm" ) THEN
010       IF NOT oForm.Parent.isBeforeFirst() AND NOT oForm.Parent.isAfterLast() THEN
011          stID = oForm.Parent.getByName("ID").BoundField.getString
012          IF stID <> "" THEN
013             oListField = oForm.getByname("lboStart")
014             stIDCurrent = oListField.BoundField.getString
015             IF stIDCurrent = "" THEN stIDCurrent = "-1"
016             stNotIn = oListField.Tag
017             stSource = oListField.ListSource(0)
018             ar() = Split(stSource, " AS ""Tab""")
019             stSource = ar(0) & " AS ""Tab"" WHERE ( ""Key"" NOT IN
                   ( " & stNotIn & " = '" & stID & "' )
                   OR ""Key"" = '" & stIDCurrent & "' ) ORDER BY ""Sort"""
020             stSql(0) = stSource
021             oListField.ListSource = stSql
022             oListField.refresh
023          END IF
024       END IF
025    END IF
026 END SUB
```

Der Code für die Listenfelder ist so vorgegeben, dass nur die unbekannten Werte nachgeliefert werden müssen. Dabei ist der Code für das Listenfeld nach dem Format SELECT * FROM (SELECT ...) erstellt.Die folgende Aliassetzungen werden vorgenommen: SELECT "Name" AS "Field", "ID" AS "Key", "Name" AS "Sort". Der Ergänzungscode beginnt mit AS "Tab". Der Alias ist notwendig, da bei machen Datenbanken (z. B. MariaDB) eine Unterabfrage nicht ohne Alias angenommen wird.

In Zeile 11 wird auf ein fest im übergeordneten Formular befindlichen Feld Bezug genommen. Hier wird extra die Bezeichnung der Formularfeldes genutzt, da der entsprechende Eintrag ja nicht unbedingt auch in der Datenquelle für das übergeordnete Formular mit dem Namen Verzeichnet ist. Nur wenn dieser Verbindungseintrag von Hauptformular zu Unterformular vorhanden ist wird der weitere Code des Makros ausgeführt.

Auch in Zeile 13 ist wieder ein Feld mit dem Feldnamen vorgegeben. Dies ist das Listenfeld, das zur Anzeige des Inhaltes dient. Es soll also immer die Bezeichnung «lboStart» erhalten.

Ist noch kein Eintrag für den aktuellen Datensatz erfolgt, so ist der aktuelle Wert des Listenfeldes leer. Hier wird dann stattdessen der Wert '-1' angenommen, der standardmäßig bei automatisch hoch zählenden Feldern nicht vorkommt.

Die Variable stNotIn speichert den Inhalt aus den Zusatzinformationen des Listenfeldes (Zeile 16). Anschließend wird der SQL-Code des Listenfeldes ausgelesen und so in ein Array eingelesen, dass das erste Element des Arrays den Startcode des Listenfeldes wiedergibt (Zeilen 17 und 18).

Schließlich wird der Code mit den entsprechenden Variablen neu zusammengefügt, so dass nur die Inhalte angeboten werden, die noch nicht ausgewählt wurden. Zusätzlich wird lediglich der Inhalt aufgenommen, der im aktuellen Datensatz bei einer vorhergehenden Auswahl bereits abgespeichert wurde. Ohne diese Ergänzung wäre das Listenfeld bei bestehenden Datensätzen ja grundsätzlich leer.

Der zusammengesetzte Inhalt sieht dann bei z. B. «ID = 1» und «stIDCurrent = 3» für die oben genannte Abfrage so aus:

```
001 SELECT "Tab".* FROM
002    (SELECT "Studienfach" AS "Field",
003    "ID" AS "Key",
004    "Studienfach" AS "Sort" FROM "tbl_Studienfach") AS "Tab"
005    WHERE "Key" NOT IN
006       (SELECT "Studienfach_ID" FROM "tbl_LehrerIn_Studienfach"
          WHERE "L_ID" = '1')
007       OR "Key" = '3'
008 ORDER BY "Sort"
```

Das Listenfeld sollte lediglich eine Prozedur erhalten, die zum Abspeichern des Inhaltes führt. Sie ist im Listenfeld über **Eigenschaften → Listenfeld → Ereignisse → Modifiziert** verknüpft.

```
001 SUB ListenfeldSpeichern(oEvent AS OBJECT)
002    oListField = oEvent.Source.Model
003    oForm = oListField.Parent
004    oListField.commit
005    IF oForm.IsNew THEN
006       oForm.insertRow
007    ELSE
008       oForm.updateRow
009    END IF
010    IF oForm.isLast THEN
011       oForm.moveToInsertRow
012    ELSE
013       oForm.next
014    END IF
015 END SUB
```

Über das auslösende Listenfeld wird auf das Formular zugegriffen. Der Inhalt des Listenfeldes wird in das Formular übertragen und abhängig davon, ob der Datensatz schon existiert oder neu ist abgespeichert.

Ab Zeile 10 wird lediglich der nächste Datensatz in dem Unterformular angesteuert. Solange Daten vorhanden sind kann dies mit «Next» geschehen. Steht der Datensatzzeiger aber auf dem letzten Datensatz, dann wird als nächster Datensatz ein neuer Datensatz eingefügt.

So kann das Listenfeld in dem obigen Beispiel durch mehrfache Betätigung ohne zusätzliche Bewegung der Maus dazu genutzt werden, alle erforderlichen Studienfächer auszuwählen.

Listenfelder zur Mehrfachauswahl nutzen

Die Nutzung der Mehrfachauswahl von Listenfeldern zeigt das folgende Formular[35]:

Die Mehrfachauswahl funktioniert zusammen mit Base nur über Makros. In dem obigen Formular werden zuerst die Termindaten eingetragen und abgespeichert, damit für das weitere Verfahren die Nummer des Primärschlüssels der Tabelle "Termine" über das Makro ausgelesen werden kann. Die Inhalte des darunter stehenden Listenfeldes werden beim Klick auf die entsprechenden Werte in Verbindung mit *Ctrl* oder *Alt* markiert und direkt gespeichert.

Die zugrunde liegende Datenbank hat den folgenden Aufbau:

Über die Makros muss die Tabelle "rel_Termine_Personen" beschrieben werden. Nach einem Wechsel des Datensatzes muss außerdem aus dieser Tabelle ausgelesen werden, auf welche Datensätze das Listenfeld einzustellen ist. Das Listenfeld kann dabei nicht mit dieser Tabelle verbunden werden, da in ein Datenbankfeld nur ein Wert eingetragen bzw. ausgelesen werden kann.

35 Die Datenbank «Beispiel_Listenfeld_Mehrfachauswahl.odb» ist den Beispieldatenbanken für dieses Handbuch beigefügt.

Die Prozedur «AuswahlSpeichern» wird in den Eigenschaften des Listenfeldes mit **Ereignisse → Modifiziert** verbunden.

```
001 SUB AuswahlSpeichern(oEvent AS OBJECT)
002    DIM oConnection AS OBJECT
003    DIM oSQL_Statement AS OBJECT
004    DIM oField AS OBJECT
005    DIM oForm AS OBJECT
006    DIM stID AS STRING
007    DIM i AS INTEGER
008    DIM k AS INTEGER
009    oField = oEvent.Source.Model
010    oForm = oField.Parent
011    stID = oForm.getString(oForm.FindColumn("ID")
```

Obwohl die im Feld "ID" gespeicherte Variable eine Integer-Variable ist, wird sie hier als String ausgelesen. Eine String-Variable kann besser für die folgende Bedingung als leer erkannt werden, da die Integer-Variable durch die vorherige Definition als Integer den Wert '0' annimmt, wenn in "ID" gar kein Wert steht.

```
012    IF stID <> "" THEN
013       oConnection = oForm.activeConnection()
014       oSQL_Statement = oConnection.createStatement()
```

Die Verbindung zur Datenbank wird über die Formularverbindung geholt, Das SQL-Statement wird vorbereitet. Danach werden einfach alle bisherigen Datensätze mit Personen, die zu dem Termin gehören, gelöscht.

```
015       FOR k = LBound(oField.ValueItemList()) TO UBound(oField.ValueItemList())
016          stSql = "DELETE FROM ""rel_Termine_Personen"" WHERE ""Termine_ID"" =
                '"+stID+"'"
017          oSQL_Statement.executeUpdate(stSql)
018       NEXT
```

Anschließend werden alle ausgewählten Werte neu in die Tabelle "rel_Termine_Personen" geschrieben

```
019       FOR i = LBound(oField.SelectedValues()) TO UBound(oField.SelectedValues())
020          stSql = "INSERT INTO ""rel_Termine_Personen"" (""Termine_ID"",
                ""Personen_ID"") VALUES ('"+stID+"','"+oField.SelectedValues(i)+"')"
021          oSQL_Statement.executeUpdate(stSql)
022       NEXT
023    END IF
024 END SUB
```

Die Prozedur «AuswahlErstellen» wird in den Eigenschaften des Formulars mit **Ereignisse → Nach dem Datensatzwechsel** verbunden.

```
001 SUB AuswahlErstellen(oEvent AS OBJECT)
002    DIM aFields()
003    DIM aValues()
004    DIM oConnection AS OBJECT
005    DIM oSQL_Statement AS OBJECT
006    DIM oResult AS OBJECT
007    DIM oForm AS OBJECT
008    DIM oField AS OBJECT
009    DIM stSql AS STRING
010    DIM stID AS STRING
011    DIM i AS INTEGER
012    DIM k AS INTEGER
013    oForm = oEvent.Source
014    oConnection = oForm.activeConnection()
015    oSQL_Statement = oConnection.createStatement()
016    aFields = Array("ListePersonen")
```

Die Listenfelder im Formular werden in ein Array geschrieben. Dies hat den Vorteil, dass gleich mehrere Listenfelder mit entsprechenden Inhalten gefüllt werden können.

```
017    stID = oForm.getString(oForm.FindColumn("ID")
018    IF stID <> "" THEN
```

Eine Schleife über alle Listenfelder wird gestartet. In diesem Fall läuft die Schleife ein einziges Mal durch.

```
019         FOR i = LBound(aFields()) TO UBound(aFields())
```

Der SQL-Code für die bestehende Auswahl der Personen zu dem betreffenden Termin wird erstellt.

```
020             stSql = "SELECT ""Personen_ID"" FROM ""rel_Termine_Personen"" WHERE
                   ""Termine_ID"" = '"+stID+"'"
```

Die Abfrage wird gestartet und das Ergebnis in ein Array lesen.

```
021             oResult = oSQL_Statement.executeQuery(stSql)
022             k = 0
023             WHILE oResult.Next
024                ReDim Preserve aValues(k)
025                aValues(k) = oResult.getString(1)
026                k = k + 1
027             WEND
```

Das Ergebnisarray wird in die Feldeigenschaften übernommen, so dass die Werte angezeigt werden. Dabei ist zu beachten, dass die Eigenschaft **`selectedValues()`** erst in LO-Versionen ab LO 4.1 und nicht unter AOO sowie älteren LO-Versionen bekannt ist. Hier müsste also gegebenenfalls nachgebessert werden.

```
028             oField = oForm.GetByName(aFields(i))
029             oField.selectedValues() = aValues()
030             NEXT
031       END IF
032   END SUB
```

Datumswert aus einem Formularwert in eine Datumsvariable umwandeln

```
001   FUNCTION Datumswert(oFeld AS OBJECT) AS DATE
002       a() = OfficeVersion()
003       IF a(0) = "LibreOffice" AND (LEFT(a(1),1) = 4 AND RIGHT(LEFT(a(1),3),1) > 0)
             OR LEFT(a(1),1) > 4 THEN
```

Hier werden alle Versionen ab 4.1 durch die oben vorgestellte Funktion «OfficeVersion()» abgefangen. Dazu wird die Version in ihre Bestandteile aufgesplittet. Die Hauptversion und die erste Unterversion werden abgefragt. Das funktioniert vorerst bis zur LO-Version 9 einwandfrei.

```
004          DIM stMonat AS STRING
005          DIM stTag AS STRING
006          stMonat = Right(Str(0) & Str(oFeld.CurrentValue.Month),2)
007          stTag =  Right(Str(0) & Str(oFeld.CurrentValue.Day),2)
008          Datumswert = CDateFromIso(oFeld.CurrentValue.Year & stMonat & stTag)
009       ELSE
010          Datumswert = CDateFromIso(oFeld.CurrentValue)
011       END IF
012   END FUNCTION
```

Das Datum wird seit LO 4.1.2 als Array im Formularfeld gespeichert. Mit dem aktuellen Wert kann also nicht auf das Datum selbst zugegriffen werden. Entsprechend ist es neu aus den Werten für Tag, Monat und Jahr zusammen zu setzen, damit anschließend in Makros damit weiter gearbeitet werden kann.

Eingabemöglichkeiten in Feldern einschränken

Numerische Felder, formatierbare Felder und Datumsfelder lassen es zu, dass mit den Pfeiltasten die Werte geändert werden können. In einem Forum kam die Nachfrage, ob das nicht irgendwie unterbunden werden könnte. Folgendes Makro verhindert nicht die Änderung, macht sie aber direkt wieder rückgängig. Es muss an das Ereignis **Taste gedrückt** gebunden werden.

```
001  SUB KeyTest(oEvent AS OBJECT)
002     IF oEvent.KeyCode = 1025 OR oEvent.KeyCode = 1024 THEN
003        oEvent.Source.Model.reset()
004     ELSE
005        oEvent.Source.Model.commit
006     END IF
007  END SUB
```

Die beiden Pfeiltasten werden abgefangen und das Feld auf den Ausgangswert zurückgestellt. Bei der Betätigung jeder anderen Taste wird der Wert direkt festgeschrieben und ist nicht mehr über **reset()** erreichbar. Andernfalls würden alle nicht abgespeicherten Eingaben des Feldes mit den Pfeiltasten wieder aufgehoben.

Suchen von Datensätzen

Ohne Makro funktioniert das Suchen von Datensätzen auch. Hier ist aber die entsprechende Abfrage äußerst unübersichtlich zu erstellen. Da könnte eine Schleife mittels Makro Abhilfe schaffen.

Die folgende Variante liest die Felder einer Tabelle aus, gründet dann intern eine Abfrage und schreibt daraus schließlich eine Liste der Primärschlüsselnummern der durchsuchten Tabelle auf, auf die der Suchbegriff zutrifft. Für die folgende Beschreibung existiert eine Tabelle "Suchtmp", die aus einem per Autowert erstellten Primärschlüsselfeld "ID" und einem Feld "Nr." besteht, in das die aus der zu durchsuchenden Tabelle gefundenen Primärschlüssel eingetragen werden. Der Tabellenname wird dabei der Prozedur am Anfang als Variable mitgegeben.

Um ein entsprechendes Ergebnis zu bekommen, muss die Tabelle natürlich nicht die Fremdschlüssel, sondern entsprechende Feldinhalte in Textform enthalten. Dafür ist gegebenenfalls eine Tabellenansicht (*View*) zu erstellen, auf die das Makro auch zugreifen kann.[36]

```
001  SUB Suche(stTabelle AS STRING)
002     DIM oDatenquelle AS OBJECT
003     DIM oVerbindung AS OBJECT
004     DIM oSQL_Anweisung AS OBJECT
005     DIM stSql AS STRING
006     DIM oAbfrageergebnis AS OBJECT
007     DIM oDoc AS OBJECT
008     DIM oDrawpage AS OBJECT
009     DIM oForm AS OBJECT
010     DIM oForm2 AS OBJECT
011     DIM oFeld AS OBJECT
012     DIM stInhalt AS STRING
013     DIM arInhalt() AS STRING
014     DIM inI AS INTEGER
015     DIM inK AS INTEGER
016     oDoc = thisComponent
017     oDrawpage = oDoc.drawpage
018     oForm = oDrawpage.forms.getByName("Suchform")
019     oFeld = oForm.getByName("Suchtext")
020     stInhalt = oFeld.getCurrentValue()
021     stInhalt = LCase(stInhalt)
```

Der Inhalt des Suchtext-Feldes wird hier von vornherein in Kleinbuchstaben umgewandelt, damit die anschließende Suchfunktion nur die Kleinschreibweisen miteinander vergleicht.

```
022     oDatenquelle = ThisComponent.Parent.DataSource
023     oVerbindung = oDatenquelle.GetConnection("","")
024     oSQL_Anweisung = oVerbindung.createStatement()
```

Zuerst wird einmal geklärt, ob überhaupt ein Suchbegriff eingegeben wurde. Ist das Feld leer, so wird davon ausgegangen, dass keine Suche vorgenommen wird. Alle Datensätze sollen angezeigt werden; eine weitere Abfrage erübrigt sich.

36 Siehe zu diesem Abschnitt auch die Datenbank «Beispiel_Suchen_und_Filtern.odb», die diesem Handbuch beiliegt.

Ist ein Suchbegriff eingegeben worden, so werden die Spaltennamen der zu durchsuchenden Tabelle ausgelesen, um auf die Felder mit einer Abfrage zugreifen zu können.

```
025     IF stInhalt <> "" THEN
026        stInhalt = String_to_SQL(stInhalt)
027        stSql ="SELECT ""COLUMN_NAME"" FROM ""INFORMATION_SCHEMA"".""SYSTEM_COLUMNS""
              WHERE ""TABLE_NAME"" = '" + stTabelle + "' ORDER BY ""ORDINAL_POSITION"""
028        oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
```

Hinweis

Der SQL-Code müsste für Firebird hier angepasst werden:

```
027 stSql = "SELECT RDB$FIELD_NAME FROM RDB$RELATION_FIELDS WHERE
        RDB$RELATION_NAME = '" + stTabelle + "' ORDER BY
        RDB$Field_POSITION"
```

Auf die Doppelung der doppelten Anführungszeichen kann hier verzichtet werden, da die Bezeichnungen sowieso keine Sonderzeichen enthalten und nur aus Großbuchstaben zusammengesetzt sind.
Leider ist der folgende SQL-Code dieses Makros für Firebird so nicht geeignet, da Firebird nicht in der Lage ist, aus selektierten Daten eine neue Tabelle zu erstellen. (~~Firebird~~)

Hinweis

SQL-Formulierungen müssen in Makros wie normale Zeichenketten zuerst einmal in doppelten Anführungsstrichen gesetzt werden. Feldbezeichnungen und Tabellenbezeichnungen stehen innerhalb der SQL-Formulierungen in der Regel bereits in doppelten Anführungsstrichen. Damit letztlich ein Code entsteht, der auch diese Anführungsstriche weitergibt, müssen für Feldbezeichnungen und Tabellenbezeichnungen diese Anführungsstriche verdoppelt werden.
Aus `stSql = "SELECT ""Name"" FROM ""Tabelle"";"`
wird, wenn es mit dem Befehl `msgbox stSql` auf dem Bildschirm angezeigt wird,
`SELECT "Name" FROM "Tabelle";`

Der Zähler des Arrays, in das die Feldnamen geschrieben werden, wird zuerst auf 0 gesetzt. Dann wird begonnen die Abfrage auszulesen. Da die Größe des Arrays unbekannt ist, muss immer wieder nachjustiert werden. Deshalb beginnt die Schleife damit, über **`ReDim Preserve arInhalt(inI)`** die Größe des Arrays festzulegen und den vorherigen Inhalt dabei zu sichern. Anschließend werden die Felder ausgelesen und der Zähler des Arrays um 1 heraufgesetzt. Damit kann dann das Array neu dimensioniert werden und wieder ein weiterer Wert abgespeichert werden.

```
029        InI = 0
030        WHILE oAbfrageergebnis.next
031           ReDim Preserve arInhalt(inI)
032           arInhalt(inI) = oAbfrageergebnis.getString(1)
033           inI = inI + 1
034        WEND
035        stSql = "DROP TABLE ""Suchtmp"" IF EXISTS"
036        oSQL_Anweisung.executeUpdate (stSql)
```

Jetzt wird die Abfrage in einer Schleife zusammengestellt, die anschließend an die zu Beginn angegebene Tabelle gestellt wird. Dabei werden alle Schreibweisen untersucht, da auch der Inhalt des Feldes in der Abfrage auf Kleinbuchstaben umgewandelt wird.

Die Abfrage wird direkt so gestellt, dass die Ergebniswerte in der Tabelle "Suchtmp" landen. Dabei wird davon ausgegangen, dass der Primärschlüssel an der ersten Position der Tabelle steht (**`arInhalt(0)`**).

```
037        stSql = "SELECT """+arInhalt(0)+"""INTO ""Suchtmp"" FROM """+stTabelle+"""
              WHERE "
038        FOR inK = 0 TO (inI - 1)
039           stSql = stSql+"LOWER("""+arInhalt(inK)+""") LIKE '%"+stInhalt+"%'"
```

```
040             IF inK < (inI - 1) THEN
041                stSql = stSql+" OR "
042             END IF
043          NEXT
044          oSQL_Anweisung.executeQuery(stSql)
045       ELSE
046          stSql = "DELETE FROM ""Suchtmp"""
047          oSQL_Anweisung.executeUpdate (stSql)
048       END IF
```

Das Anzeigeformular muss neu geladen werden. Es hat als Datenquelle eine Abfrage, in diesem Beispiel "Suchabfrage"

```
049       oForm2 = oDrawpage.forms.getByName("Anzeige")
050       oForm2.reload()
051    End Sub
```

Damit wurde eine Tabelle erstellt, die nun in einer Abfrage ausgewertet werden soll. Die Abfrage ist dabei möglichst so zu fassen, dass sie anschließend noch editiert werden kann. Im Folgenden also ein Abfragecode:

```
001 SELECT * FROM "Suchtabelle"
002 WHERE "Nr." IN ( SELECT "Nr." FROM "Suchtmp" )
003    OR "Nr." =
004       CASE WHEN ( SELECT COUNT( "Nr." ) FROM "Suchtmp" ) > 0
          THEN '0' ELSE "Nr." END
```

Alle Elemente der **"Suchtabelle"** werden dargestellt. Auch der Primärschlüssel. Keine andere Tabelle taucht in der direkten Abfrage auf; somit ist auch kein Primärschlüssel einer anderen Tabelle nötig, damit das Abfrageergebnis weiterhin editiert werden kann.

Der Primärschlüssel ist in dieser Beispieltabelle unter dem Titel **"Nr."** abgespeichert. Durch das Makro wird genau dieses Feld ausgelesen. Es wird jetzt also zuerst nachgesehen, ob der Inhalt des Feldes **"Nr."** in der Tabelle **"Suchtmp"** vorkommt. Bei der Verknüpfung mit **'IN'** werden ohne weiteres auch mehrere Werte erwartet. Die Unterabfrage darf also auch mehrere Datensätze liefern.

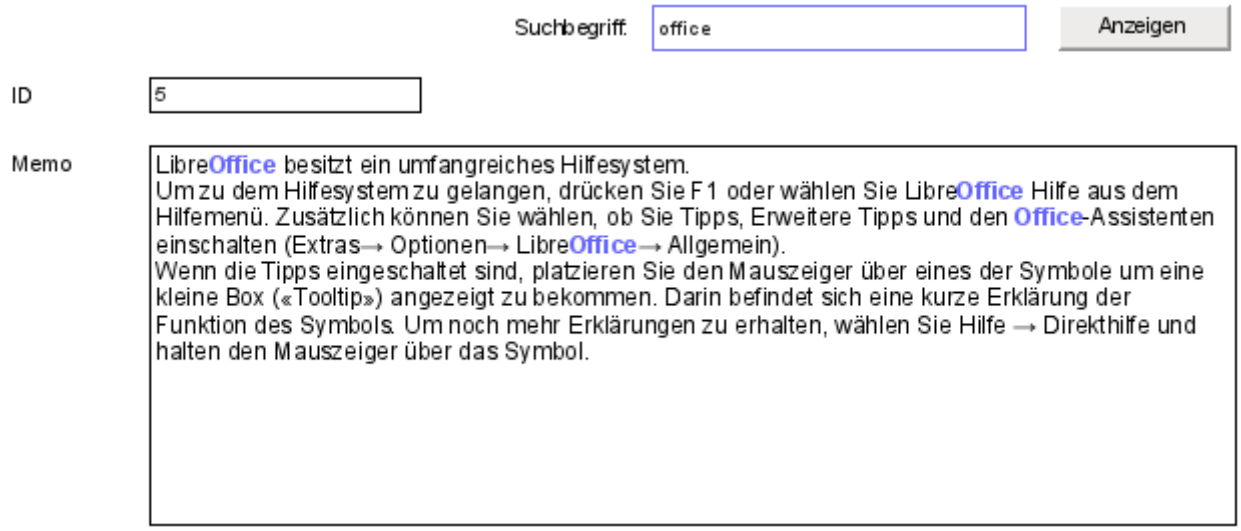
Bei größeren Datenmengen wird der Abgleich von Werten über die Verknüpfung IN aber zusehends langsamer. Es bietet sich also nicht an, für eine leere Eingabe in das Suchfeld einfach alle Primärschlüsselfelder der **"Suchtabelle"** in die Tabelle **"Suchtmp"** zu übertragen und dann auf die gleiche Art die Daten anzusehen. Stattdessen erfolgt bei einer leeren Eingabe eine Leerung der Tabelle **"Suchtmp"**, so dass gar keine Datensätze mehr vorhanden sind. Hierauf zielt der zweite Bedingungsteil:

```
003    OR "Nr." =
004       CASE WHEN ( SELECT COUNT( "Nr." ) FROM "Suchtmp" ) > 0
          THEN '-1' ELSE "Nr." END
```

Wenn in der Tabelle **"Suchtmp"** ein Datensatz gefunden wird, so ist das Ergebnis der ersten Abfrage größer als 0. Für diesen Fall gilt: **"Nr." = '-1'** (hier steht am Besten ein Zahlenwert, der als Primärschlüssel nicht vorkommt, also z.B. **'-1'**). Ergibt die Abfrage genau 0 (Dies ist der Fall wenn keine Datensätze da sind), dann gilt **"Nr." = "Nr."**. Es wird also jeder Datensatz dargestellt, der eine **"Nr."** hat. Da **"Nr."** der Primärschlüssel ist, gilt dies also für alle Datensätze.

Suchen in Formularen und Ergebnisse farbig hervorheben

Bei größeren Inhalten eines Textfeldes ist oft unklar, an welcher Stelle denn nun die Suche den Treffer zu verzeichnen hat. Da wäre es doch gut, wenn das Formular den entsprechenden Treffer auch markieren könnte. So sollte das dann im Formular aussehen:



Um so ein Formular zum Laufen zu bringen, bedarf es ein paar zusätzlicher Griffe in die Trickkiste.[37]

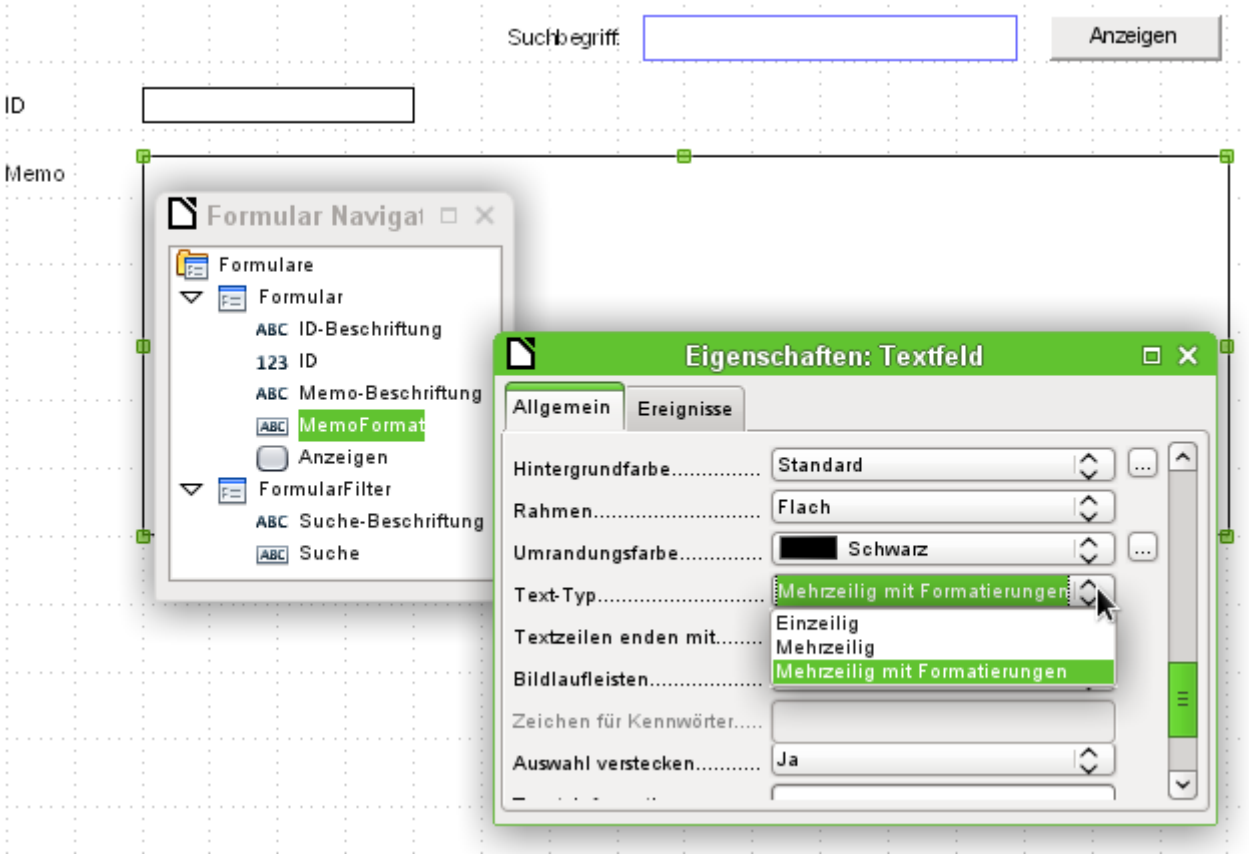
Die Funktionsweise eines solchen Suchfeldes wurde bereits bei den Abfragetechniken erklärt: Es wird eine Filtertabelle erstellt. Über ein Formular wird nur der aktuelle Wert des einzigen Datensatzes in dieser Tabelle neu geschrieben. Das Hauptformular wird über eine Abfrage mit dem entsprechenden Inhalt versorgt. Die Abfrage sieht im obigen Fall so aus:

```
001 SELECT "ID", "Memo"
002 FROM "Tabelle"
003 WHERE LOWER ( "Memo" ) LIKE '%' || LOWER (
004    ( SELECT "Suchtext" FROM "Filter" WHERE "ID" = TRUE ) ) || '%'
```

Wird ein Suchtext eingetragen, so werden nur die Datensätze der Tabelle "Tabelle" angezeigt, bei denen der Text im Feld "Memo" vorkommt. Dabei ist Groß- und Kleinschreibung egal.

Wird kein Suchtext eingetragen, so werden alle Datensätze der Tabelle "Tabelle" angezeigt. Da der Primärschlüssel dieser Tabelle auch in der Abfrage enthalten ist, ist die Abfrage außerdem editierbar.

37 Siehe zu diesem Abschnitt auch die Datenbank «Beispiel_Autotext_Suchmarkierung_Rechtschreibung.odb», die dem Handbuch beiliegt.



In dem Formular ist neben dem Feld «ID» für den Primärschlüsseleintrag nur ein Feld «MemoFormat», das über **Eigenschaften → Allgemein → Text-Typ → Mehrzeilig mit Formatierungen** so eingestellt ist, dass es überhaupt Farben innerhalb von schwarzem Text darstellen kann. Die genaue Betrachtung der Eigenschaften des Textfeldes zeigt, dass der Reiter **Daten** fehlt. Daten lassen sich über ein Feld, das zusätzlich formatierbar ist, nicht eingeben. Das ist wohl dadurch begründet, dass die Datenbank selbst auch solche Formatierungen nicht speichert. Und trotzdem ist es durch den entsprechenden Makroeinsatz möglich, Text in dieses Feld hinein zu bekommen, ihn zu markieren und bei Änderungen auch wieder aus dem Feld hinaus in die Datenbank zu befördern.

Die Prozedur «InhaltUebertragen» dient dazu, den Inhalt aus dem Datenbankfeld "Memo" in das formatierbare Textfeld «MemoFormat» zu übertragen und so zu formatieren, dass bei einem entsprechenden Eintrag im Suchfeld der dazugehörige Begriff hervorgehoben wird.

Die Prozedur ist an das folgende Ereignis gebunden: **Formular → Ereignisse → Nach dem Datensatzwechsel**

```
001 Sub InhaltUebertragen(oEvent AS OBJECT)
002     DIM inMemo AS INTEGER
003     DIM oFeld AS OBJECT
004     DIM stSuchtext AS STRING
005     DIM oCursor AS OBJECT
006     DIM inSuch AS INTEGER
007     DIM inSuchAlt AS INTEGER
008     DIM inLen AS INTEGER
009     oForm = oEvent.Source
010     inMemo = oForm.findColumn("Memo")
011     oFeld = oForm.getByName("MemoFormat")
012     oFeld.Text = oForm.getString(inMemo)
```

Zuerst werden die Variablen definiert. Anschließend wird über das Formular das Tabellenfeld "Memo" gesucht und aus diesem Feld über **getString**() der entsprechende Text des Feldes "Memo" der Tabelle "Tabelle" ausgelesen. Der entsprechende Feldinhalt wird in das Feld übertragen, das sich formatieren lässt, aber keine Verbindung zur Datenbank hat: «MemoFormat».

Bei Tests ist es zuerst vorgekommen, dass sich das Formular zwar öffnete, aber leider die Formularleiste am unteren Rand des Formulars nicht mehr aufgebaut wurde. Deswegen erfolgt hier ein sehr kurzer Wartebefehl von 5/1000 Sekunden. Danach wird aus dem parallel zum «Formular» liegenden «FormularFilter» der angezeigte Inhalt als Suchtext ausgelesen.

```
013     Wait 5
014     stSuchtext = oForm.Parent.getByName("FormularFilter").getByName("Suche").Text
```

Um Textteile formatieren zu können muss ein (nicht sichtbarer) **TextCursor** in dem Feld erstellt werden, das den Text enthält. Die Darstellung des Textes in der Standardversion hat eine serifenbetonte Schriftart in 12-Punkt-Größe, die in anderen Formularteilen nicht unbedingt vorkommt und über das Formularfeld auch nicht direkt abwählbar ist. In dieser Prozedur wird direkt zu Beginn der Text einmal auf die gewünschte Darstellungsart eingestellt. Erfolgt dies nicht schon zu Beginn, so wird wegen der unterschiedlichen Formatierungen der oberer Textrand in dem Feld erst einmal angeschnitten. Die erste Zeile war in Versuchen nur zu 2/3 lesbar.

Damit der Cursor (wieder nicht sichtbar) den Text markiert, wird er zuerst an den Anfang gesetzt und mit dem Zusatz **true** weiterbewegt zum Endpunkt, der ebenfalls den Zusatz **true** hat. Dann erfolgt die Zuweisung der notwendigen Eigenschaften wie Schriftgröße, Schriftstil, Schriftfarbe oder auch Schriftdicke. Anschließend wird der Cursor wieder zur Startposition gesetzt.

```
015     oCursor = oFeld.createTextCursor()
016     oCursor.gotoStart(true)
017     oCursor.gotoEnd(true)
018     oCursor.CharHeight = 10
019     oCursor.CharFontName = "Arial, Helvetica, Tahoma"
020     oCursor.CharColor = RGB(0,0,0)
021     oCursor.CharWeight = 100.000000   'com::sun::star::awt::FontWeight
022     oCursor.gotoStart(false)
```

Enthält das Feld Text und ist ein Eintrag zum Suchen vorhanden, so wird jetzt der Text nach dem Suchbegriff durchsucht. Die äußere Schleife fragt erst einmal nur nach der Bedingung, die nächste Schleife klärt noch einmal, ob der Suchtext denn tatsächlich in dem Text enthalten ist, der in «MemoFormat» steht. Diese Einstellung könnte auch unterlassen werden, da die Abfrage, auf der das Formular basiert, nur solchen Text anzeigt, auf den diese Bedingung zutrifft.

```
023     IF oFeld.Text <> "" AND stSuchtext <> "" THEN
024        IF inStr(oFeld.Text, stSuchtext) THEN
025           inSuch = 1
026           inSuchAlt = 0
027           inLen = Len(stSuchtext)
```

Der Text wird nach dem Suchtext durchsucht. Dies erfolgt in einer Schleife, die dann endet, wenn keine weitere Trefferposition mehr angezeigt wird. **InStr**() liefert dabei die Fundstelle des ersten Zeichens des Suchtextes, in der aufgezeigten Fassung unabhängig von Groß- und Kleinschreibung. Die Schleife wird dadurch gesteuert, dass der Suchbeginn **inSuch** bei jedem Schleifenende in der Summe um 1 erhöht wird (erste Schleifenzeile -1, letzte Schleifenzeile +2). Bei jedem Durchgang wird der Cursor mit **oCursor.goRight(Position, false)** zuerst ohne zu markieren an die Startstelle gesetzt, dann um die Länge des Suchtextes weiter mit der Markierungsaufforderung nach rechts gesetzt. Dann wird die gewünschte Formatierung (blau, etwas dicker) vorgenommen und der Cursor wieder für den nächsten Start an den Startpunkt der Markierung zurückgesetzt.

```
028           DO WHILE inStr(inSuch, oFeld.Text, stSuchtext) > 0
029              inSuch = inStr(inSuch, oFeld.Text, stSuchtext) - 1
030              oCursor.goRight(inSuch-inSuchAlt,false)
031              oCursor.goRight(inLen,true)
032              oCursor.CharColor = RGB(102,102,255)
033              oCursor.CharWeight = 110.000000
034              oCursor.goLeft(inLen,false)
035              inSuchAlt = inSuch
036              inSuch = inSuch + 2
037           LOOP
038        END IF
```

```
039     END IF
040  End Sub
```

Die Prozedur «InhaltSchreiben» dient dazu, den Inhalt aus dem formatierbaren Textfeld «MemoFormat» in die Datenbank zu übertragen. Dies erfolgt in dieser Fassung unabhängig davon, ob eine Änderung vorgenommen wurde.

Die Prozedur ist an das folgende Ereignis gebunden: **Formular → Ereignisse → Vor dem Datensatzwechsel**

```
001  Sub InhaltSchreiben(oEvent AS OBJECT)
002     DIM oForm AS OBJECT
003     DIM inMemo AS INTEGER
004     DIM loID AS LONG
005     DIM oFeld AS OBJECT
006     DIM stMemo AS STRING
007     oForm = oEvent.Source
008     IF InStr(oForm.ImplementationName, "ODatabaseForm") THEN
```

Das auslösende Ereignis ist doppelt belegt. Nur der Implementationsname, der auf **`ODatabaseForm`** endet, gibt den richtigen Zugriff auf den Datensatz.

```
009        IF NOT oForm.isBeforeFirst() AND NOT oForm.isAfterLast() THEN
```

Beim Einlesen, auch beim Reload des Formulars, steht der Cursor vor dem ersten Datensatz. Würde jetzt ein Schreibversuch unternommen, dann erscheint die Meldung «ungüliger Cursorstatus».

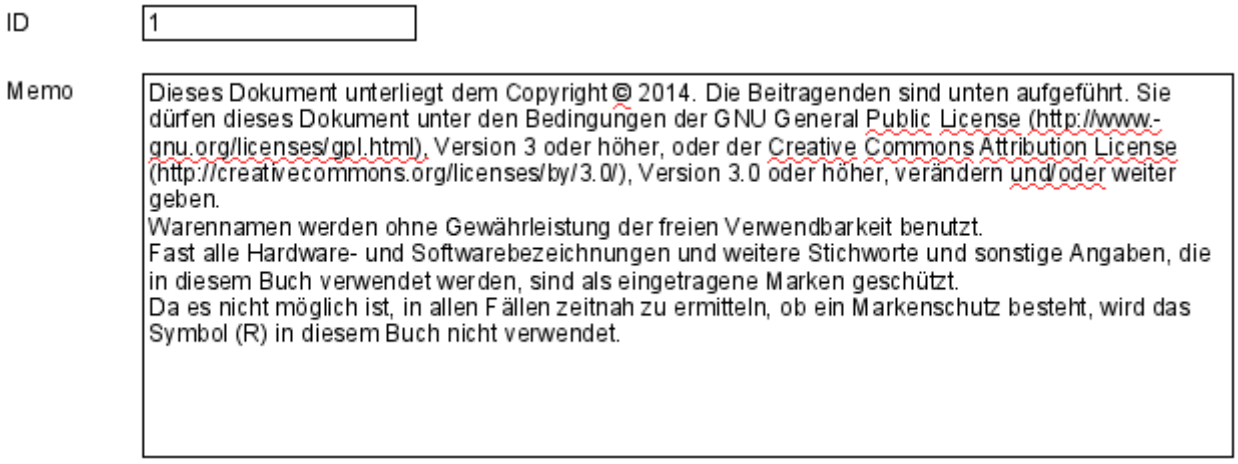
```
010           inMemo = oForm.findColumn("Memo")
011           loID = oForm.findColumn("ID")
012           oFeld = oForm.getByName("MemoFormat")
013           stMemo = oFeld.Text
014           IF stMemo <> "" THEN
015              oForm.updateString(inMemo,stMemo)
016           END IF
017           IF stMemo <> "" AND oForm.getString(loID) <> "" THEN
018              oForm.UpdateRow()
019           END IF
020        END IF
021     END IF
022  End Sub
```

Das Tabellenfeld "Memo" wird aus der Datenquelle des Formulars herausgesucht. Ebenso das Feld "ID". Befindet sich im Feld «MemoFormat» Text, so wird er mit **`oForm.updateString`**() in das Feld "Memo" der Datenquelle übertragen. Nur wenn bereits ein Eintrag im Feld "ID" existiert, also der Primärschlüssel belegt ist, erfolgt ein Update. Ansonsten wird ja sowieso ein neuer Datensatz über die Formularfunktionen eingefügt, da das Formular die Änderung entsprechend bemerkt und eine Abspeicherung selbständig vornimmt.

Rechtschreibkontrolle während der Eingabe

Auf **mehrzeilige Textfelder mit Formatierungen** greift auch dieses Makro zu. Entsprechend muss auch, wie bei dem vorherigen Kapitel, der Inhalt bei jedem Datensatzwechsel zuerst geschrieben und danach der Inhalt des neuen Datensatzes in das Formularfeld geladen werden. Die Prozeduren «InhaltUebertragen» und «InhaltSchreiben» unterscheiden sich höchstens in dem Punkt, dass die Suchfunktion ausgeklammert werden kann.[38]

38 Siehe auch hierzu: «Beispiel_Autotext_Suchmarkierung_Rechtschreibung.odb»



Die Rechtschreibkontrolle wird in dem obigen Formular dadurch ausgelöst, dass in dem Formularfeld entweder eine Leertaste oder ein Return betätigt wird. Sie läuft also nach der Beendigung eines Wortes jedes Mal ab und könnte gegebenenfalls auch noch mit dem Fokusverlust des Formularfeldes gekoppelt werden, damit auch das letzte Wort sicher überprüft wird.

Die Prozedur ist an das folgende Ereignis gebunden: **Formular → Ereignisse → Taste losgelassen**

```
001 SUB MarkierungFehlerDirekt(oEvent AS OBJECT)
002    GlobalScope.BasicLibraries.LoadLibrary("Tools")
```

Es wird die Funktion **RTrimStr** zum Entfernen von Satzzeichen am Ende vor Worten benötigt. Sonst werden alle Worte, denen ein Komma, Punkt oder irgendein anderes Satzzeichen folgt, als falsch angesehen. Mit **LTrimChar** müssen außerdem Klammern zum Beginn des Wortes entfernt werden.

```
003    DIM aProp() AS NEW com.sun.star.beans.PropertyValue
004    DIM oLinuSvcMgr AS OBJECT
005    DIM oSpellChk AS OBJECT
006    DIM oFeld AS OBJECT
007    DIM arText()
008    DIM stWort AS STRING
009    DIM inlenWort AS INTEGER
010    DIM ink AS INTEGER
011    DIM i AS INTEGER
012    DIM oCursor AS OBJECT
013    DIM stText AS OBJECT
014    oLinguSvcMgr = createUnoService("com.sun.star.linguistic2.LinguServiceManager")
015    IF NOT IsNull(oLinguSvcMgr) THEN
016       oSpellChk = oLinguSvcMgr.getSpellChecker()
017    END IF
```

Zuerst werden alle Variablen deklariert. Danach wird auf das Rechtschreibüberprüfungsmodul **SpellChecker** zugegriffen. Mit diesem Modul werden anschließend die einzelnen Worte auf ihre Richtigkeit hin überprüft.

```
018    oFeld = oEvent.Source.Model
019    ink = 0
020    IF oEvent.KeyCode = 1280 OR oEvent.KeyCode = 1284 THEN
```

Das Ereignis, das das Makro auslöst, ist ein Tastendruck. Zu dem Ereignis wird ein Code für jede Taste mitgeliefert, der **KeyCode**. Der **KeyCode** für die ↵ (Eingabetaste) ist 1280, der für die Leertaste ist 1284. Wie viele andere Informationen sind diese Informationen einfach durch das Tool «Xray» gewonnen worden. Wird also eine Leertaste oder die ↵ (Eingabetaste) betätigt, so wird die Rechtschreibung überprüft. Sie startet also zu jedem Wortende. Lediglich die Überprüfung für das letzte Wort ist so nicht automatisch möglich.

Bei jedem Durchlauf werden alle Worte des Textes überprüft. Die Überprüfung einzelner Worte könnte eventuell auch möglich sein, bedeutet aber erheblich mehr Aufwand.

Der Text wird also in Worte aufgesplittet. Trenner ist hier das Leerzeichen. Vorher müssen allerdings noch Trennungen an Zeilenumbrüchen erzeugt werden, die sonst später als ein Wort wahrgenommen werden.

```
021         stText = Join(Split(oFeld.Text,CHR(10))," ")
022         stText = Join(Split(stText,CHR(13))," ")
023         arText = Split(RTrim(stText)," ")
024         FOR i = LBound(arText) TO Ubound(arText)
025            stWort = arText(i)
026            inlenWort = len(stWort)
027            stWort = Trim( RtrimStr( RtrimStr( RtrimStr( RtrimStr( RtrimStr(
                  RtrimStr(stWort,","), "."),"?"),"!"),"."),")"))
028            stWort = LTrimChar(stWort,"(")
```

Das einzelne Wort wird ausgelesen, seine ungekürzte Länge ist notwendig für die folgenden Bearbeitungsschritte. Nur so kann die Position des Wortes innerhalb des gesamten Textes bestimmt werden, die auch für die gezielte Markierung von Schreibfehlern gebraucht wird.

Mit **Trim** werden Leerzeichen entfernt, mit der Funktion **RTrimStr** Kommas und Satzzeichen am Ende des Textes, mit der Funktion **LTrimChar** Zeichen am Anfang des Textes.

```
029            IF stWort <> "" THEN
030               oCursor = oFeld.createTextCursor()
031               oCursor.gotoStart(false)
032               oCursor.goRight(ink,false)
033               oCursor.goRight(inLenWort,true)
034               If Not oSpellChk.isValid(stWort, "de", aProp()) Then
035                  oCursor.CharUnderline = 9
036                  oCursor.CharUnderlineHasColor = True
037                  oCursor.CharUnderlineColor = RGB(255,51,51)
038               ELSE
039                  oCursor.CharUnderline = 0
040               END IF
041            END IF
042            ink = ink + inLenWort + 1
043         NEXT
044      END IF
045   END SUB
```

Hat das Wort einen Inhalt, so wird zuerst einmal ein Textcursor erstellt. Der Textcursor wird ohne Markierung an den Start des Textes in dem Eingabefeld gesetzt. Dann geht es, immer noch ohne Markierung, um den Betrag nach rechts im Text vorwärts, der in der Variablen **ink** gespeichert ist. Diese Variable ist am Anfang 0, nach Durchlaufen der ersten Schleife dann so groß wie das vorhergehende Wort lang war +1 für das angehängte Leerzeichen. Dann wird der Cursor mit Markierung um die Länge des aktuellen Wortes weiter gesetzt. Erfolgt jetzt eine Änderung der Buchstabeneigenschaften, so betrifft diese nur den markierten Bereich.

Der **Spellchecker** startet. Als Variablen müssen das Wort und der Landescode übergeben werden. Ohne Landescode ist alles richtig. Das Array bleibt in der Regel leer.

Ist das Wort nicht in den Lexika eingetragen, so wird es mit einer roten Wellenlinie versehen. Die Wellenlinie entspricht hier der **'9'**. Ist das Wort eingetragen, so wird statt einer Wellenlinie keine Linie (**'0'**) gezeichnet. Dieser Schritt ist notwendig, weil sonst ein einmal als falsch erkanntes Wort bei einer Korrektur auch weiterhin mit der roten Wellenlinie gekennzeichnet würde. Eine rote Wellenlinie würde nie aufgehoben, da es keine entgegengesetzte Formatierung gibt.

Kombinationsfelder als Listenfelder mit Eingabemöglichkeit

Aus Kombinationsfeldern und Tabellenfeldern aus dem Formular kann direkt eine Tabelle mit einem Datensatz versehen und der entsprechende Primärschlüssel in eine andere Tabelle eingetragen werden.[39]

39 Die Beispieldatenbank «Beispiel_Combobox_Listfeld.odb» zum Einsatz von Kombinationsfeldern statt Listenfeldern ist den Beispieldatenbanken für dieses Handbuch beigefügt.

Das Modul «Comboboxen» macht aus den Formularfeldern zur Eingabe und Auswahl von Werten (Kombinationsfelder) Listenfelder mit Eingabemöglichkeiten. Dazu werden neben den Kombinationsfeldern im Formular die jeweils an die zugrundeliegende Tabelle zu übergebenden Schlüsselfeldwerte in den Tabellenspalten abgelegt, die dem Formular zugrunde liegen. Die Schlüssel aus den Tabellenspalten werden beim Start des Formulars ausgelesen und das Kombinationsfeld auf den entsprechenden Inhalt eingestellt. Wird der Inhalt des Kombinationsfeldes geändert, so wird er neu abgespeichert und der neue Primärschlüssel zum Abspeichern in der Haupttabelle in das entsprechende numerische Fremdschlüsselfeld übertragen.

Werden statt der Tabellen entsprechend konstruierte eingabefähige Abfragen erstellt, so kann der Text, den die Kombinationsfelder darstellen sollen, direkt aus den Abfragen ermittelt werden. Ein Makro ist dann für diesen Arbeitsschritt nicht notwendig.

Voraussetzung für die Funktionsweise des Makros ist, dass alle Primärschlüssel der Tabellen, die in den Kombinationsfeldern als Datenquellen auftauchen, mit einem automatisch hochzählenden Autowert versehen sind. Außerdem ist als Bezeichnung hier vorausgesetzt, dass die Primärschlüssel den Namen "ID" tragen.

Textanzeige im Kombinationsfeld

Diese Prozedur soll Text in den Kombinationsfeldern nach den Werten der (unsichtbaren) Fremdschlüssel-Felder aus dem Hauptformular einstellen. Dabei werden gegebenenfalls auch Listenfelder berücksichtigt, die sich auf 2 unterschiedliche Tabellen beziehen. Dies kann z.B. dann sein, wenn bei einer Ortsangabe die Postleitzahl vom Ort abgetrennt wurde. Dann wird die Postleitzahl aus einer Tabelle ausgelesen, in der auch ein Fremdschlüssel für den Ort liegt. Im Listenfeld werden Postleitzahl und Ort zusammen angezeigt.

```
001  SUB TextAnzeigen(oEvent AS OBJECT)
```

Dieses Makro sollte an das folgende Ereignis des Formulars gebunden werden: 'Nach dem Datensatzwechsel'

Das Makro wird direkt aus dem Formular angesprochen. Über das auslösende Ereignis werden die gesamten notwendigen Variablen für das Makro ermittelt.

Die Variablen werden deklariert. Einige Variablen sind in einem separaten Modul bereits global deklariert und werden hier nicht noch einmal erwähnt.

```
002     DIM oForm AS OBJECT
003     DIM oFeld AS OBJECT
004     DIM oFeldList AS OBJECT
005     DIM stAbfrage AS STRING
006     DIM stFeldWert AS STRING
007     DIM stFeldID AS STRING
008     DIM inCom AS INTEGER
009     oForm = oEvent.Source
```

Das Formular startet das Ereignis. Es ist die Quelle für das das Makro auslösende Ereignis.

In dem Formular befindet sich ein verstecktes Kontrollelement, aus dem hervorgeht, wie die verschiedenen Kombinationsfelder in diesem Formular heißen. Nacheinander werden dann in dem Makro die Kombinationsfelder abgearbeitet.

```
010     aComboboxen() = Split(oForm.getByName("Comboboxen").Tag,",")
011     FOR inCom = LBound(aComboboxen) TO Ubound(aComboboxen)
            ...
        NEXT inCom
```

Aus den Zusatzinformationen («Tag») des versteckten Kontrollelementes wird die Bezeichnung der Kombinationsfelder ermittelt. Sie sind dort durch Kommas voneinander getrennt aufgeschrieben. Die Namen der Felder werden in ein Array geschrieben und nacheinander in einer Schleife abgearbeitet. Die Schleife endet mit der Bezeichnung **NEXT …** .

Das Kombinationsfeld, das jetzt statt eines Listenfeldes existiert, wird anschließend als **oFeldList** bezeichnet. Der Fremdschlüssel wird über die Bezeichnung des Tabellenfeldes, die

in den Zusatzinformationen des Kombinationsfeldes steht, aus der Tabellenspalte des Formulars ermittelt.

```
012         oFeldList = oForm.getByName(Trim(aComboboxen(inCom)))
013         stFeldID = oForm.getString(oForm.findColumn(oFeldList.Tag))
014         oFeldList.Refresh()
```

Das Kombinationsfeld wird mit **Refresh()** neu eingelesen. Es kann ja sein, dass sich der Inhalt des Feldes durch Neueingaben geändert hat. Diese müssen schließlich verfügbar gemacht werden.

Die Abfrage, die zur Ermittlung des anzuzeigenden Inhaltes des Kombinationsfeldes notwendig ist, wird aus der Abfrage des Kombinationsfeldes und dem ermittelten Wert des Fremdschlüssels erstellt. Damit der SQL-Code brauchbar wird, wird zuerst eine eventuelle Sortieranweisung entfernt. Anschließend wird nachgesehen, ob bereits eine Beziehungsdefinition (beginnend mit **WHERE**) existiert. Da die **InStr()**-Funktion standardmäßig keinen Unterschied zwischen Groß- und Kleinschreibung macht, werden hier gleich alle Schreibweisen abgedeckt. Existiert eine Beziehungsdefinition, so enthält die Abfrage Felder aus zwei unterschiedlichen Tabellen. Es muss jetzt die Tabelle herausgesucht werden, aus der der Fremdschlüssel für die Verbindung zur Verfügung gestellt wird. Das Makro funktioniert hier nur mit Hilfe der Information, dass der Primärschlüssel einer jeden Tabelle "ID" heißt.

Existiert keine Beziehungsdefinition, so beruht die Abfrage nur auf einer Tabelle. Die Tabelleninformation kann entfallen, die Bedingung direkt mit dem Fremdschlüsselwert zusammen formuliert werden.

```
015         IF stFeldID <> "" THEN
016            stAbfrage = oFeldList.ListSource
017            IF InStr(stAbfrage,"order by") > 0 THEN
018               stSql = Left(stAbfrage, InStr(stAbfrage,"order by")-1)
019            ELSE
020               stSql = stAbfrage
021            END IF
022            IF InStr(stSql,"where") THEN
023               st = Right(stSql, Len(stSql)-InStr(stSql,"where")-4)
024               IF InStr(Left(st, InStr(st,"=")),"""ID""") THEN
025                  a() = Split(Right(st, Len(st)-InStr(st,"=")-1),".")
026               ELSE
027                  a() = Split(Left(st, InStr(st,"=")-1),".")
028               END IF
029               stSql = stSql + "AND "+a(0)+"""ID""" = "+stFeldID
030            ELSE
031               stSql = stSql + "WHERE ""ID"" = "+stFeldID
032            END IF
```

Jedes Feld und jeder Tabellenname muss bereits in der SQL-Eingabe mit doppelten Anführungsstrichen oben versehen werden. Da bereits Anführungsstriche einfacher Art in Basic als die Einführung zu Text interpretiert werden, sind diese bei der Weitergabe des Codes nicht mehr sichtbar. Erst bei einer Doppelung der Anführungsstriche wird ein Element mit einfachen Anführungsstrichen weitergegeben. **""ID""** bedeutet also, dass in der Abfrage auf das Feld **"ID"** (mit einfachen Anführungsstrichen für die SQL-Verbindung) zugegriffen wird.

Die in der Variablen **stSql** abgespeicherte Abfrage wird jetzt ausgeführt und das Ergebnis dieser Abfrage in der Variablen **oAbfrageergebnis** gespeichert.

```
033            oDatenquelle = ThisComponent.Parent.CurrentController
034            IF NOT (oDatenquelle.isConnected()) Then
035               oDatenquelle.connect()
036            End IF
037            oVerbindung = oDatenquelle.ActiveConnection()
038            oSQL_Anweisung = oVerbindung.createStatement()
039            oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
```

Das Abfrageergebnis wird über eine Schleife ausgelesen. Hier könnten, wie in einer Abfrage aus der GUI, mehrere Felder und Datensätze dargestellt werden. Von der Konstruktion der Abfrage her wird aber nur ein Ergebnis erwartet. Dieses Ergebnis wird in der ersten Spalte **(1)** der

Abfrage zu finden sein. Es ist der Datensatz, der den anzuzeigenden Inhalt des Kombinationsfeldes wiedergibt. Der Inhalt ist ein Textinhalt (**getString()**), deshalb hier **oAbfrageergebnis.getString(1)**.

```
040          WHILE oAbfrageergebnis.next
041             stFeldWert = oAbfrageergebnis.getString(1)
042          WEND
```

Das Kombinationsfeld muss jetzt auf den aus der Abfrage sich ergebenden Textwert eingestellt werden.

```
043          oFeldList.Text = stFeldWert
044       ELSE
```

Falls überhaupt kein Wert in dem Feld für den Fremdschlüssel **oFeld** vorhanden ist, ist auch die ganze Abfrage nicht gelaufen. Das Kombinationsfeld wird jetzt auf eine leere Anzeige eingestellt.

```
045          oFeldList.Text = ""
046       END IF
047    NEXT inCom
048 END SUB
```

Diese Prozedur erledigt jetzt also den Kontakt von dem in der Datenquelle des Formulars abgelegten Fremdschlüssel zu dem Kombinationsfeld. Für die Anzeige der richtigen Werte im Kombinationsfeld würde das ausreichen. Eine Abspeicherung von neuen Werten hingegen benötigt eine weitere Prozedur.

Fremdschlüsselwert vom Kombinationsfeld zum numerischen Feld übertragen

Wird nun ein neuer Wert ausgewählt oder neu in das Kombinationsfeld eingegeben (nur wegen dieser Eigenschaft wurde ja das Makro konstruiert), so muss der entsprechende Primärschlüssel als Fremdschlüssel in die dem Formular zugrundeliegende Tabelle eingetragen werden.

```
001 SUB TextAuswahlWertSpeichern(oEvent AS OBJECT)
```

Dieses Makro sollte an das folgende Ereignis des Formulars gebunden werden: 'Vor der Datensatzaktion'.

Nach Deklaration der Variablen (hier nicht weiter aufgeführt) wird zuerst differenziert, bei welchem Ereignis genau das Makro überhaupt ablaufen soll. Vor der Datensatzaktion werden zwei Implementationen nacheinander aufgerufen. Für das Makro selbst ist es wichtig, das Formularobjekt zu erhalten. Das geht prinzipiell über beide Implementationen, aber eben auf unterschiedliche Weise. Es wird hier die Implementation mit dem Namen **"ODatabaseForm"** herausgefiltert.

```
002    IF InStr(oEvent.Source.ImplementationName,"ODatabaseForm") THEN
          ...
       END IF
    END SUB
```

In diese Schleife wird der gleiche Start wie bei der Prozedur **TextAnzeigen** eingebaut:

```
003       oForm = oEvent.Source
004       aComboboxen() = Split(oForm.getByName("Comboboxen").Tag,",")
005       FOR inCom = LBound(aComboboxen) TO Ubound(aComboboxen)
             ...
          NEXT inCom
```

Das Feld **oFeldList** zeigt den Text an. Es kann in einem Tabellenkontrollfeld liegen. Dann kann nicht direkt vom Formular auf das Feld zugegriffen werden. In den Zusatzinformationen des versteckten Kontrollfeldes «Comboboxen» ist für diesen Fall der Pfad zum Kombinationsfeld über «Tabellenkontrollfeld>Kombinationsfeld» eingetragen. Durch Aufsplittung dieses Eintrages wird ermittelt, wie das Kombinationsfeld anzusprechen ist.

```
006          a() = Split(Trim(aComboboxen(inCom)),">")
007          IF Ubound(a) > 0 THEN
008             oFeldList = oForm.getByName(a(0)).getByName(a(1))
009          ELSE
```

```
010             oFeldList = oForm.getByName(a(0))
011          END IF
```

Anschließend wird die Abfrage aus dem Kombinationsfeld ausgelesen und in ihre Einzelteile zerlegt. Bei einfachen Kombinationsfeldern wären die notwendigen Informationen lediglich der Feldname und der Tabellenname:

```
001 SELECT "Feld" FROM "Tabelle"
```

Dies könnte gegebenenfalls noch durch eine Sortierung erweitert sein. Sobald zwei Felder in dem Kombinationsfeld zusammen dargestellt werden, muss aber bereits bei den Feldern zur Trennung entsprechend mehr Aufwand getrieben werden:

```
001 SELECT "Feld1"||' '||"Feld2" FROM "Tabelle"
```

Diese Abfrage fasst zwei Felder zusammen und setzt dazwischen eine Leertaste ein. Da der Trenner eine Leertaste ist, wird in dem Makro nach so einem Trenner gesucht und danach der Text in zwei Teile gesplittet. Das funktioniert natürlich nur dann einwandfrei, wenn "Feld1" nicht bereits Text enthalten soll, der eine Leertaste erlaubt. Sonst wird z.B. aus dem Vornamen «Anne Marie» und dem Nachnamen «Müller» durch das Makro der Vorname «Anne» und der Nachname «Marie Müller». Für solch einen Fall sollte ein passender Trenner eingesetzt werden, der dann auch vom Makro gefunden werden kann. Bei Namen ist dies z. B. ein Komma: «Nachname, Vorname».

Noch komplizierter wird es, wenn die beiden enthaltenen Felder aus zwei verschiedenen Tabellen stammen:

```
001 SELECT "Tabelle1"."Feld1"||' > '||"Tabelle2"."Feld2"
002 FROM "Tabelle1", "Tabelle2"
003 WHERE "Tabelle1"."ID" = "Tabelle2"."FremdID"
004 ORDER BY "Tabelle1"."Feld1"||' > '||"Tabelle2"."Feld2" ASC
```

Hier müssen die Felder voneinander getrennt, die Tabellenzuordnungen zu den Feldern erfasst und die Fremdschlüsselzuweisung ermittelt werden.

```
012          stAbfrage = oFeldList.ListSource
013          aFelder() = Split(stAbfrage, """")
014          stInhalt = ""
015          FOR i=LBound(aFelder)+1 TO UBound(aFelder)
```

Der Inhalt der Abfrage wird von Ballast befreit. Die Teile werden anschließend über eine nicht übliche Zeichenkombination zu einem Array wieder zusammengefügt.«FROM» trennt die sichtbare Feldanzeige von der Tabellenbezeichnung. «WHERE» trennt die Beziehungsdefinition von der Tabellenbezeichnung. Joins werden nicht unterstützt.

```
016             IF Trim(UCASE(aFelder(i))) = "ORDER BY" THEN
017                EXIT FOR
018             ELSEIF Trim(UCASE(aFelder(i))) = "FROM" THEN
019                stInhalt = stInhalt+" §§ "
020             ELSEIF Trim(UCASE(aFelder(i))) = "WHERE" THEN
021                stInhalt = stInhalt+" §§ "
022             ELSE
023                stInhalt = stInhalt+Trim(aFelder(i))
024             END IF
025          NEXT i
026          aInhalt() = Split(stInhalt, " §§ ")
```

Die sichtbare Feldanzeige wird gegebenenfalls in Inhalte aus verschiedenen Feldern aufgeteilt:

```
027          aErster() = Split(aInhalt(0),"||")
028          IF UBound(aErster) > 0 THEN
029             IF UBound(aInhalt) > 1 THEN
```

Der erste Teil enthält mindestens 2 Felder. Die Felder haben zu Beginn eine Tabellenbezeichnung. Der zweite Teil enthält zwei Tabellenbezeichnungen, die aber schon aus dem ersten Teil ermittelt werden können. Der dritte Teil enthält eine Beziehung über einen Fremdschlüssel mit «=» getrennt:

```
030                aTest() = Split(aErster(0),".")
031                NameTabelle1 = aTest(0)
032                NameTabellenFeld1 = aTest(1)
033                Erase aTest
034                Feldtrenner = Join(Split(aErster(1),"'"),"")
035                aTest() = Split(aErster(2),".")
036                NameTabelle2 = aTest(0)
037                NameTabellenFeld2 = aTest(1)
038                Erase aTest
039                aTest() = Split(aInhalt(2),"=")
040                aTest1() = Split(aTest(0),".")
041                IF aTest1(1) <> "ID" THEN
042                   NameTab12ID = aTest1(1)
043                   IF aTest1(0) = NameTabelle1 THEN
044                   Position = 2
045                   ELSE
046                      Position = 1
047                   END IF
048                ELSE
049                   Erase aTest1
050                   aTest1() = Split(aTest(1),".")
051                   NameTab12ID = aTest1(1)
052                   IF aTest1(0) = NameTabelle1 THEN
053                      Position = 2
054                   ELSE
055                      Position = 1
056                   END IF
057                END IF
058             ELSE
```

Der erste Teil enthält zwei Feldbezeichnungen ohne Tabellenbezeichnung mit Trenner, der zweite Teil enthält die Tabellenbezeichnung. Ein dritter Teil ist nicht vorhanden:

```
059                NameTabellenFeld1 = aErster(0)
060                Feldtrenner = Join(Split(aErster(1),"'"),"")
061                NameTabellenFeld2 = aErster(2)
062                NameTabelle1 = aInhalt(1)
063             END IF
064          ELSE
```

Es existiert nur ein Feld, das aus einer Tabelle stammt:

```
065             NameTabellenFeld1 = aErster(0)
066             NameTabelle1 = aInhalt(1)
067          END IF
```

Die maximale Zeichenlänge, die eine Eingabe haben darf, wird im Folgenden mit der Funktion **Spaltengroesse** ermittelt. Das Kombinationsfeld kann hier mit seiner Beschränkung nicht sicher weiterhelfen, da ja ermöglicht werden soll, gleichzeitig 2 Felder in einem Kombinationsfeld einzutragen.

```
068          LaengeFeld1 = Spaltengroesse(NameTabelle1,NameTabellenFeld1)
069          IF NameTabellenFeld2 <> "" THEN
070             IF NameTabelle2 <> "" THEN
071                LaengeFeld2 = Spaltengroesse(NameTabelle2,NameTabellenFeld2)
072             ELSE
073                LaengeFeld2 = Spaltengroesse(NameTabelle1,NameTabellenFeld2)
074             END IF
075          ELSE
076             LaengeFeld2 = 0
077          END IF
```

Der Inhalt des Kombinationsfeldes wird ausgelesen:

```
078          stInhalt = oFeldList.getCurrentValue()
```

Der angezeigte Inhalt des Kombinationsfeldes wird ausgelesen. Leertasten und nicht druckbare Zeichen am Anfang und Ende der Eingabe werden gegebenenfalls entfernt.

```
079          stInhalt = Trim(stInhalt)
080          IF stInhalt <> "" THEN
081             IF NameTabellenFeld2 <> "" THEN
```

Wenn ein zweites Tabellenfeld existiert, muss der Inhalt des Kombinationsfeldes gesplittet werden. Um zu wissen, an welcher Stelle die Aufteilung vorgenommen werden soll, ist der Feldtrenner von Bedeutung, der der Funktion als Variable mitgegeben wird. Ein Leerzeichen aus dem Feldtrenner wird bei der Funktion «Split» nicht direkt erkannt. Deswegen wird das ASCII-Zeichen noch einmal in den entsprechenden Feldtrenner umgewandelt.

```
082                IF ASC(Feldtrenner) = 32 THEN
083                   Feldtrenner = " "
084                END IF
085                a_stTeile = Split(stInhalt, Feldtrenner, 2)
```

Der letzte Parameter weist darauf hin, dass maximal 2 Teile erzeugt werden.

Abhängig davon, welcher Eintrag mit dem Feld 1 und welcher mit dem Feld 2 zusammenhängt, wird jetzt der Inhalt des Kombinationsfeldes den einzelnen Variablen zugewiesen. «Position = 2» wird hier als Zeichen dafür genommen, dass an zweiter Position der Inhalt für das Feld 2 steht. Das ist auch dann der Fall, wenn beide Felder aus einer Tabelle stammen.

```
086                IF Position = 2 OR Position = 0 THEN
087                   stInhalt = Trim(a_stTeile(0))
088                   IF UBound(a_stTeile()) > 0 THEN
089                      stInhaltFeld2 = Trim(a_stTeile(1))
090                   ELSE
091                      stInhaltFeld2 = ""
092                   END IF
093                   stInhaltFeld2 = Trim(a_stTeile(1))
094                ELSE
095                   stInhaltFeld2 = Trim(a_stTeile(0))
096                   IF UBound(a_stTeile()) > 0 THEN
097                      stInhalt = Trim(a_stTeile(1))
098                   ELSE
099                      stInhalt = ""
100                   END IF
101                   stInhalt = Trim(a_stTeile(1))
102                END IF
103             END IF
```

Es kann passieren, dass bei zwei voneinander zu trennenden Inhalten die Größeneinstellung des Kombinationsfeldes (Textlänge) nicht zu einem der abzuspeichernden Tabellenfelder passt. Bei Kombinationsfeldern für nur ein Feld wird dies in der Regel durch Einstellungen im Formularkontrollfeld erledigt. Hier muss hingegen ein eventueller Fehler abgefangen werden. Es wird darauf hingewiesen, wie lang der Inhalt für das jeweilige Feld sein darf.

```
104             IF (LaengeFeld1 > 0 AND Len(stInhalt) > LaengeFeld1) OR
                   (LaengeFeld2 > 0 AND Len(stInhaltFeld2) > LaengeFeld2) THEN
```

Wenn die Feldlänge des 1. oder 2. Teiles zu groß ist, wird erst einmal ein Standardtext in je einer Variablen abgespeichert. **CHR(13)** fügt hier einen Zeilenumbruch hinzu.

```
105                stmsgbox1 = "Das Feld " + NameTabellenFeld1 + " darf höchstens " +
                      LaengeFeld1 + "Zeichen lang sein." + CHR(13)
106                stmsgbox2 = "Das Feld " + NameTabellenFeld2 + " darf höchstens " +
                      LaengeFeld2 + "Zeichen lang sein." + CHR(13)
```

Sind beide Feldinhalte zu lang, so wird der Text mit beiden Feldinhalten ausgegeben.

```
107                IF (LaengeFeld1 > 0 AND Len(stInhalt) > LaengeFeld1) AND
                      (LaengeFeld2 > 0 AND Len(stInhaltFeld2) > LaengeFeld2) THEN
108                   msgbox ("Der eingegebene Text ist zu lang." + CHR(13) +
                         stmsgbox1 + stmsgbox2 + "Bitte den Text kürzen.",
                         64,"Fehlerhafte Eingabe")
```

Die Anzeige erfolgt mit der Funktion **msgbox()**. Sie erwartet zuerst einen Text, dann optional einen Zahlenwert (der zu einer entsprechenden Darstellungsform gehört) und schließlich optio-

nal einen Text als Überschrift über dem Fenster. Das Fenster hat hier also die Überschrift "Fehlerhafte Eingabe", die '64' fügt das Informationssymbol hinzu.

Im Folgenden werden alle auftretenden weiteren Fälle zu großer Textlänge abgearbeitet.

```
109                ELSEIF (LaengeFeld1 > 0 AND Len(stInhalt) > LaengeFeld1) THEN
110                   msgbox ("Der eingegebene Text ist zu lang." + CHR(13) +
                         stmsgbox1 + "Bitte den Text kürzen.",64,"Fehlerhafte Eingabe")
111                ELSE
112                   msgbox ("Der eingegebene Text ist zu lang." + CHR(13) +
                         stmsgbox2 + "Bitte den Text kürzen.",64,"Fehlerhafte Eingabe")
113                END IF
114             ELSE
```

Liegt kein zu langer Text vor, so kann die Funktion weiter durchlaufen. Ansonsten endet sie hier.

Jetzt werden die Inhaltseingaben so maskiert, dass eventuell vorhandene Hochkommata keine Fehlermeldung erzeugen.

```
115                stInhalt = String_to_SQL(stInhalt)
116                IF stInhaltFeld2 <> "" THEN
117                   stInhaltFeld2 = String_to_SQL(stInhaltFeld2)
118                END IF
```

Zuerst werden Variablen vorbelegt, die anschließend per Abfrage geändert werden können. Die Variablen **inID1** und **inID2** sollen den Inhalt der Primärschlüsselfelder der beiden Tabellen speichern. Da bei einer Abfrage, die kein Ergebnis wiedergibt, durch Basic einer Integer-Variablen 0 zugewiesen wird, dies aber für das Abfrageergebnis auch bedeuten könnte, dass der ermittelte Primärschlüssel eben den Wert 0 hat, wird die Variable auf jeweils -1 voreingestellt. Diesen Wert nimmt ein Autowert-Feld bei der HSQLDB nicht automatisch an.

Anschließend wird die Datenbankverbindung erzeugt, soweit sie nicht schon besteht.

```
119                inID1 = -1
120                inID2 = -1
121                oDatenquelle = ThisComponent.Parent.CurrentController
122                If NOT (oDatenquelle.isConnected()) Then
123                   oDatenquelle.connect()
124                End If
125                oVerbindung = oDatenquelle.ActiveConnection()
126                oSQL_Anweisung = oVerbindung.createStatement()
127                IF NameTabellenFeld2 <> "" AND NOT IsEmpty(stInhaltFeld2) AND
                      NameTabelle2 <> "" THEN
```

Wenn ein zweites Tabellenfeld existiert, muss zuerst die zweite Abhängigkeit geklärt werden. Zuerst wird überprüft, ob für den zweiten Wert in der Tabelle 2 bereits ein Eintrag existiert. Existiert dieser Eintrag nicht, so wird er eingefügt.

Beispiel: Die Tabelle 2 ist die Tabelle "Ort". In ihr werden also Orte abgespeichert. Ist z.B. ein Eintrag für den Ort 'Rheine' vorhanden, so wird der entsprechende Primärschlüsseleintrag ausgelesen. Ist der Eintrag 'Rheine' nicht vorhanden, wird er eingefügt und anschließend der beim Einfügen erzeugte Primärschlüsselwert festgestellt.

```
128                   stSql = "SELECT ""ID"" FROM """ + NameTabelle2 + """ WHERE """ +
                         NameTabellenFeld2 + """='" + stInhaltFeld2 + "'"
129                   oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
130                   WHILE oAbfrageergebnis.next
131                      inID2 = oAbfrageergebnis.getInt(1)
132                   WEND
133                   IF inID2 = -1 THEN
134                      stSql = "INSERT INTO """ + NameTabelle2 + """ (""" +
                            NameTabellenFeld2 + """) VALUES ('" + stInhaltFeld2 + "') "
135                      oSQL_Anweisung.executeUpdate(stSql)
136                      stSql = "CALL IDENTITY()"
```

Ist der Inhalt in der entsprechenden Tabelle nicht vorhanden, so wird er eingefügt. Der dabei entstehende Primärschlüsselwert wird anschließend ausgelesen. Ist der Inhalt bereits vorhanden, so wird der Primärschlüsselwert durch die vorangehende Abfrage ermittelt. Die Funktion geht hier von automatisch erzeugten Primärschlüsselfeldern (**IDENTITY**) aus.

Hinweis

Die Funktion **CALL IDENTITY()** ist in ~~Firebird~~ unbekannt. Firebird hat hierfür eigentlich die Funktion **RETURNING** vorgesehen, die direkt an den SQL-Befehl angehängt wird und der Ausgabe den Primärschlüsselwert mitgibt. Leider funktioniert dies unter LO zur Zeit nicht. Stattdessen muss über eine separate Abfrage der gerade erzeugte Primärschlüsselwert ermittelt werden.

```
137                     oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
138                     WHILE oAbfrageergebnis.next
139                           inID2 = oAbfrageergebnis.getInt(1)
140                     WEND
141                  END IF
```

Der Primärschlüssel aus dem zweiten Wert wird in der Variablen **'inID2'** zwischengespeichert. Jetzt wird überprüft, ob eventuell dieser Schlüsselwert bereits in der Tabelle 1 zusammen mit dem Eintrag aus dem ersten Feld vorhanden ist. Ist diese Kombination nicht vorhanden, so wird sie neu eingefügt.

Beispiel: Für den Ort 'Rheine' aus der Tabelle 2 können in der Tabelle 1 mehrere Postleitzahlen verfügbar sein. Ist die Kombination '48431' und 'Rheine' vorhanden, so wird nur der Primärschlüssel aus der Tabelle 1 ausgelesen, in der die Postleitzahlen und der Fremdschlüssel aus der Tabelle 2 gespeichert wurden.

```
142                  stSql = "SELECT ""ID"" FROM """ + NameTabelle1 + """ WHERE """ +
                        NameTab12ID + """='" + inID2 + "' AND """ +
                        NameTabellenFeld1 + """ = '" + stInhalt + "'"
143                  oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
144                  WHILE oAbfrageergebnis.next
145                     inID1 = oAbfrageergebnis.getInt(1)
146                  WEND
```

War der Inhalt der ersten Tabelle noch nicht vorhanden, so wird der Inhalt neu abgespeichert (**INSERT**).

Beispiel: Existiert bereits die Postleitzahl '48429' in Kombination mit dem Fremdschlüssel aus der Tabelle 2 "Ort", so wird auf jeden Fall ein neuer Datensatz erzeugt, wenn jetzt die Postleitzahl '48431' auftaucht. Der vorhergehenden Datensatz wird also nicht auf die neue Postleitzahl geändert. Schließlich sind durch die n:1-Verknüpfung der Tabellen mehrere Postleitzahlen für einen Ort ermöglicht worden.

```
147                  IF inID1 = -1 THEN
148                     stSql = "INSERT INTO """ + NameTabelle1 + """ (""" +
                           NameTabellenFeld1 + """,""" + NameTab12ID + """)
                           VALUES ('" + stInhalt + "','" + inID2 + "') "
149                     oSQL_Anweisung.executeUpdate(stSql)
```

Der Primärschlüssel der ersten Tabelle muss schließlich wieder ausgelesen werden, damit er in die dem Formular zugrundeliegende Tabelle übertragen werden kann.

```
150                     stSql = "CALL IDENTITY()"
151                     oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
152                     WHILE oAbfrageergebnis.next
153                        inID1 = oAbfrageergebnis.getInt(1)
154                     WEND
155                  END IF
156               END IF
```

Für den Fall, dass beide in dem Kombinationsfeld zugrundeliegenden Felder in einer Tabelle gespeichert sind (z. B. Nachname, Vorname in der Tabelle Name) muss eine andere Abfrage erfolgen:

```
157               IF NameTabellenFeld2 <> "" AND NameTabelle2 = "" THEN
158                  stSql = "SELECT ""ID"" FROM """ + NameTabelle1 + """ WHERE """ +
                        NameTabellenFeld1 + """='" + stInhalt + "' AND """ +
                        NameTabellenFeld2 + """='" + stInhaltFeld2 + "'"
159                  oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
```

```
160                     WHILE oAbfrageergebnis.next
161                        inID1 = oAbfrageergebnis.getInt(1)
162                     WEND
163                     IF inID1 = -1 THEN
```

Wenn eine zweite Tabelle nicht existiert:

```
164                        stSql = "INSERT INTO """ + NameTabelle1 + """ (""" +
                              NameTabellenFeld1 + """,""" + NameTabellenFeld2 + """)
                              VALUES ('" + stInhalt + "','" + stInhaltFeld2 + "') "
165                        oSQL_Anweisung.executeUpdate(stSql)
```

Anschließend wird das Primärschlüsselfeld wieder ausgelesen.

```
166                        stSql = "CALL IDENTITY()"
167                        oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
168                        WHILE oAbfrageergebnis.next
169                              inID1 = oAbfrageergebnis.getInt(1)
170                        WEND
171                     END IF
172                  END IF
173                  IF NameTabellenFeld2 = "" THEN
```

Jetzt wird der Fall geklärt, der der einfachste ist: Das 2. Tabellenfeld existiert nicht und der Eintrag ist noch nicht in der Tabelle vorhanden. In das Kombinationsfeld ist also ein einzelner neuer Wert eingetragen worden.

```
174                     stSql = "SELECT ""ID"" FROM """ + NameTabelle1 + """ WHERE """ +
                           NameTabellenFeld1 + """='" + stInhalt + "'"
175                     oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
176                     WHILE oAbfrageergebnis.next
177                        inID1 = oAbfrageergebnis.getInt(1)
178                     WEND
179                     IF inID1 = -1 THEN
```

Wenn ein zweites Tabellenfeld nicht existiert, wird der Inhalt neu eingefügt ...

```
180                        stSql = "INSERT INTO """ + NameTabelle1 + """ (""" +
                              NameTabellenFeld1 + """) VALUES ('" + stInhalt + "') "
181                        oSQL_Anweisung.executeUpdate(stSql)
```

... und die entsprechende ID direkt wieder ausgelesen. (HSQLDB, ~~FIREBIRD~~)

```
182                        stSql = "CALL IDENTITY()"
183                        oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
184                        WHILE oAbfrageergebnis.next
185                              inID1 = oAbfrageergebnis.getInt(1)
186                        WEND
187                     END IF
188                  END IF
```

Der Wert des Primärschlüsselfeldes muss ermittelt werden, damit er in die Haupttabelle des Formulars übertragen werden kann.

Anschließend wird der aus all diesen Schleifen ermittelte Primärschlüsselwert in das Feld der Haupttabelle und die darunterliegende Datenbank übertragen. Mit **`findColumn`** wird das mit dem Formularfeld verbundene Tabellenfeld erreicht. Mit **`updateLong`** wird eine Integer-Zahl (siehe *Datentypen in StarBasic*) diesem Feld zugewiesen.

```
189                  oForm.updateLong(oForm.findColumn(oFeldList.Tag),inID1)
190               END IF
191            ELSE
```

Ist kein Primärschlüsselwert einzutragen, weil auch kein Eintrag in dem Kombinationsfeld erfolgte oder dieser Eintrag gelöscht wurde, so ist auch der Inhalt des Feldes zu löschen. Mit **`updateNULL()`** wird das Feld mit dem datenbankspezifischen Ausdruck für ein leeres Feld, **`NULL`**, versehen.

```
192               oForm.updateNULL(oForm.findColumn(oFeldList.Tag),NULL)
193            END IF
194        NEXT inCom
```

```
195     END IF
196  END SUB
```

Kontrollfunktion für die Zeichenlänge der Kombinationsfelder

Die folgende Funktion soll die Zeichenlänge der jeweiligen Tabellenspalten ermitteln, damit zu lange Eingaben nicht einfach gekürzt werden. Der Typ **FUNCTION** wurde hier wegen der Rückgabewerte gewählt.

```
001  FUNCTION Spaltengroesse(Tabellenname AS STRING, Feldname AS STRING) AS INTEGER
002     oDatenquelle = ThisComponent.Parent.CurrentController
003     If NOT (oDatenquelle.isConnected()) Then
004        oDatenquelle.connect()
005     End If
006     oVerbindung = oDatenquelle.ActiveConnection()
007     oSQL_Anweisung = oVerbindung.createStatement()
008     stSql = "SELECT ""COLUMN_SIZE"" FROM ""INFORMATION_SCHEMA"".""SYSTEM_COLUMNS""
           WHERE ""TABLE_NAME"" = '" + Tabellenname + "' AND ""COLUMN_NAME"" = '"
           + Feldname + "'"
009     oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
010     WHILE oAbfrageergebnis.next
011        i = oAbfrageergebnis.getInt(1)
012     WEND
013     Spaltengroesse = i
014  END FUNCTION
```

Hinweis

Für Firebird muss der SQL-Code angepasst werden:

```
008  stSql = "SELECT B.RDB$FIELD_LENGTH
           FROM RDB$RELATION_FIELDS AS A, RDB$FIELDS AS B
           WHERE A.RDB$FIELD_SOURCE = B.RDB$FIELD_NAME
           AND A.RDB$RELATION_NAME = '" + Tabellenname + "'
           AND A.RDB$FIELD_NAME = '" + Feldname + "'"
```

Datensatzaktion erzeugen

Dieses Makro sollte an das **Ereignis → Bei Fokuserhalt** des Listenfeldes gebunden werden. Es ist notwendig, damit auf jeden Fall bei einer Änderung des Listenfeldinhaltes die Speicherung abläuft. Ohne dieses Makro wird keine Änderung in der Tabelle erzeugt, die für Base wahrnehmbar ist, da die Combobox mit dem Formular nicht verbunden ist.

Dieses Makro stellt direkt die Eigenschaft des Formulars um.

```
001  SUB Datensatzaktion_erzeugen(oEvent AS OBJECT)
002     DIM oForm AS OBJECT
003     oForm = oEvent.Source.Model.Parent
004     oForm.IsModified = TRUE
005  END SUB
```

Bei Formularen, die bereits ihren Inhalt auch für die Kombinationsfelder aus Abfragen erhalten, ist dieses Makro nicht notwendig. Änderungen in den Kombinationsfeldern werden direkt registriert.

Navigation von einem Formular zum anderen

Ein Formular soll über ein entsprechendes Ereignis geöffnet werden.

Im Formularkontrollfeld wird unter **Formular-Eigenschaften → Zusatzinformationen** (Tag) der Name des Formulars eintragen. Hier können auch weitere Informationen eingetragen werden, die über den Befehl **`Split()`** anschließend voneinander getrennt werden.

```
001 SUB Zu_Formular_von_Formular(oEvent AS OBJECT)
002    DIM stTag AS String
003    stTag = oEvent.Source.Model.Tag
004    aForm() = Split(stTag, ",")
```

Das Array wird gegründet und mit den Formularnamen gefüllt, in diesem Fall zuerst in dem zu öffnenden Formular und als zweites dem aktuellen Formular, dass nach dem Öffnen des anderen geschlossen werden soll. Existiert ein zweiter Eintrag nicht, so wird durch das Makro nur ein neues Formular geöffnet.

```
005    ThisDatabaseDocument.FormDocuments.getByName( Trim(aForm(0)) ).open
006    IF Ubound(aForm()) > 0 THEN
007       ThisDatabaseDocument.FormDocuments.getByName( Trim(aForm(1)) ).close
008    END IF
009 END SUB
```

Soll grundsätzlich das aktuelle Formular geschlossen werden, so braucht nur in den Zusatzinformationen des Buttons für das folgende Makro das Zielformular angegeben zu werden:

```
001 SUB Zu_Formular_von_Formular(oEvent AS OBJECT)
002    DIM stZiel AS String
003    aFormStart() = Split(thisComponent.Title, thisComponent.UntitledPrefix)
004    stZiel = oEvent.Source.Model.Tag
005    ThisDatabaseDocument.FormDocuments.getByName( Trim(stZiel) ).open
006    ThisDatabaseDocument.FormDocuments.getByName( Trim(aFormStart(1)) ).close
007 END SUB
```

Soll stattdessen nur beim Schließen ein anderes Formular geöffnet werden, weil z.B. ein Hauptformular existiert und alle anderen Formulare von diesem aus über entsprechende Buttons angesteuert werden, so ist das folgende Makro einfach an das Formular unter **Extras → Anpassen → Ereignisse → Dokument wird geschlossen** anzubinden:

```
001 SUB Hauptformular_oeffnen
002    ThisDatabaseDocument.FormDocuments.getByName( "Hauptformular" ).open
003 END SUB
```

Wenn die Formulardokumente innerhalb der *.odb-Datei in Verzeichnissen sortiert sind, so muss das Makro für den Formularwechsel etwas umfangreicher sein:

```
001 SUB Zu_Formular_von_Formular_mit_Ordner(oEvent AS OBJECT)
002    REM Das zu öffenende Formular wird als erstes angegeben.
003    REM Liegt ein Formular in einem Ordner, so ist die Beziehung über "/" zu
004    REM definieren, so dass der Unterordner zu finden ist.
005    DIM stTag AS STRING
006    stTag = oEvent.Source.Model.Tag 'Tag wird unter den Zusatzinformationen
          eingegeben
007    aForms() = Split(stTag, ",")   'Hier steht zuerst der Formularname für das neue
          Formular, dann der für das alte Formular
008    aForms1() = Split(aForms(0),"/")
009    aForms2() = Split(aForms(1),"/")
010    IF UBound(aForms1()) = 0 THEN
011       ThisDatabaseDocument.FormDocuments.getByName( Trim(aForms1(0)) ).open
012    ELSE
013       ThisDatabaseDocument.FormDocuments.getByName( Trim(aForms1(0)) ).getByName(
             Trim(aForms1(1)) ).open
014    END IF
015    IF UBound(aForms2()) = 0 THEN
016       ThisDatabaseDocument.FormDocuments.getByName( Trim(aForms2(0)) ).close
017    ELSE
018       ThisDatabaseDocument.FormDocuments.getByName( Trim(aForms2(0)) ).getByName(
             Trim(aForms2(1)) ).close
019    END IF
020 END SUB
```

Formulardokumente, die in einem Verzeichnis liegen, werden in den Zusatzinformationen als Verzeichnis/Formular angegeben. Dies muss umgewandelt werden zu **`...getByName("Verzeichnis").getByName("Formular")`**.

Datensatz im Formular direkt öffnen

Wird von einem Formular zum anderen gesprungen, so kann das Zielformular natürlich über eine Filtertabelle mit einem bestimmten Schlüssel versehen werden und direkt nur mit einem Datensatz geöffnet werden. Manchmal ist es aber erforderlich, direkt eine Übersicht über mehrere Datensätze zu haben und dennoch den korrekten Datensatz direkt in einem Tabellenkontrollfeld angezeigt zu bekommen. Das im folgenden vorgestellte Makro springt bei entsprechender Vorgabe direkt zu dem gewünschten Datensatz.

```
001  SUB DatumsAenderung(oEvent AS OBJECT)
002     DIM oDoc AS OBJECT
003     DIM oDrawpage AS OBJECT
004     DIM oForm AS OBJECT
005     DIM oDatField AS OBJECT
006     DIM oConnection AS OBJECT
007     DIM oSQL_Statement AS OBJECT
008     DIM oResult AS OBJECT
009     DIM iRow AS INTEGER
010     DIM stDatum AS STRING
011     DIM stSql AS STRING
012     oDatField = oEvent.Source.Model
013     stDatum = oDatField.CurrentValue.Year & "-" &
           Right("0" & oDatField.CurrentValue.Month , 2) & "-" &
           Right("0" & oDatField.CurrentValue.Day , 2)
014     oDoc = thisComponent
015     oDrawpage = oDoc.drawpage
016     oForm = oDrawpage.forms.getByName("Uebersicht")
017     oConnection = oForm.activeConnection()
018     oSQL_Statement = oConnection.createStatement()
019     stSql = oForm.SingleSelectQueryComposer.Query
020     oResult = oSQL_Statement.executeQuery(stSql)
021     Do
022        oResult.next
023        iRow = iRow +1
024        IF oResult.isLast THEN Exit Do
025     LOOP UNTIL stDatum = oResult.getString(2)
026     oForm.last
027     oForm.absolute(iRow)
028  END SUB
```

Bei diesem Beispiel wird zuerst aus einem Formularfeld für ein Datum ein bestimmter Wert ausgelesen. Dieser Datumswert wird in die für SQL übliche Schreibweise umgewandelt. Das Formular soll auf den ersten Datensatz eingestellt werden, der mit dem Datumswert übereinstimmt.

Das Formular wird angesteuert. Für die Ermittlung der Position des Datensatzes ist es nun wichtig, genau zu wissen, mit welchem SQL-Kommando denn der Inhalt des Formulars gefüllt wurde. Dieses SQL-Kommando befindet sich nicht eindeutig in **oForm.Command**. Das ist lediglich das Kommando, das bei der Erstellung des Formulars ausgesucht wird. Im Formular selbst kann aber noch sortiert und gefiltert werden. Entweder müssten also zusätzlich zu dem Command noch die Filterung und Sortierung berücksichtigt werden oder nach einer fertigen Zusammenstellung in den Formulareigenschaften gesucht werden. Die fertige Zusammenstellung unter Berücksichtigung von Filter und Sortierung befindet sich in **oForm.SingleSelectQueryComposer.Query**. Dieser Composer steht nur dann zur Verfügung, wenn die Abfrage nicht im direkten SQL-Modus ausgeführt wird.

In der dem Formular zugrundeliegenden Abfrage wird jetzt nach dem Datum gesucht. Dazu wird mit **DO ... LOOP UNTIL** eine Schleife durchlaufen, die dann endet, wenn der in der Abfrage an 2. Position stehende Datumswert genau der Vorgabe entspricht, oder wenn der letzte Datensatz erreicht ist. Für jede neu ausgelesene Zeile wird der Zähler **iRow** um '1' erhöht.

Zum Schluss wird das Formular auf den letzten Datensatz eingestellt und von dort aus dann zurück auf die ermittelte Datenzeile. Das hat den Vorteil, dass das Datum in diesem Fall im

Tabellenkontrollfeld auf jeden Fall oben steht und außerdem noch klar ist, wie viele Datensätze denn im Moment über das Formular verfügbar sind.

Tabellen, Abfragen, Formulare und Berichte öffnen

Ähnlich wie im vorhergehenden Kapitel lassen sich von einem Formular aus auch Berichte öffnen. Berichte sind wie Formulare in die Base-Datei eingebundene separate Dokumente. Statt **FormDocuments** ist hier lediglich **ReportDocuments** einzutragen. Außerdem ist noch darauf zu achten, dass sowohl Formulare als auch Berichte in Unterverzeichnissen liegen können. Schwieriger ist es hingegen, auch auf Tabellen, Abfragen und Ansichten zuzugreifen, da diese nicht als separate Dokumente vorliegen.

```
001 SUB Navigation(oEvent AS OBJECT)
002    DIM stTag AS STRING
003    DIM inType AS INTEGER
004    stTag = oEvent.Source.Model.Tag
005    aOpen() = Split(stTag, ",")
006    SELECT CASE Trim(aOpen(0))
007       CASE "form", "report"
008          REM Forms and Reports could be saved also in subfolders.
009          aForms1() = Split(Trim(aOpen(1)),"/")
010          IF Trim(aOpen(0)) = "form" THEN
011             oDoc = ThisDatabaseDocument.FormDocuments
012          ELSE
013             oDoc = ThisDatabaseDocument.ReportDocuments
014          END IF
015          IF Ubound(aForms1()) > 0 THEN
016             oDoc.getByName( Trim(aForms1(0)) ).getByName( Trim(aForms1(1)) ).open
017          ELSE
018             oDoc.getByName( Trim(aForms1(0)) ).open
019          END IF
020          IF Trim(aOpen(0)) = "form" AND Ubound(aOpen()) > 1 THEN
021             REM The Form, which starts the Macro, could also be closed ...
022             aForms2() = Split(Trim(aOpen(2)),"/")
023             IF Ubound(aForms2()) > 0 THEN
024                ThisDatabaseDocument.FormDocuments.
                      getByName( Trim(aForms2(0)) ).getByName( Trim(aForms2(1)) ).close
025             ELSE
026                ThisDatabaseDocument.FormDocuments.
                      getByName( Trim(aForms2(0)) ).close
027             END IF
028          END IF
029          EXIT SUB
030       CASE "query"
031          inType = 1
032          Open_Table_Query_View(Trim(aOpen(1)),inType)
033       CASE "table"
034          inType = 0
035          Open_Table_Query_View(Trim(aOpen(1)),inType)
036    END SELECT
037 END SUB
```

Über die Prozedur Navigation wird das Makro gestartet. Von den Buttons wird aus den Zusatzinformationen (**Tag**) die Information ausgelesen, ob ein Formular (**form**), ein Bericht (**report**) usw. aufgerufen werden soll. Der Name des Formulars, Berichtes usw. wird in den Zusatzinformationen durch ein Komma von dieser Information getrennt.

Enthält der erste Teil des daraus ermittelten Arrays die Bezeichnung **form**, so wird anschließend das Formular geöffnet. Entsprechendes gilt für die Bezeichnung **report**, die den **SELECT CASE** für den Bericht ergibt.

Für Abfragen und Tabellen muss ein anderer Weg beschritten werden. Hier wird sowohl der Name der Abfrage bzw. Tabelle als auch eine Integer-Zahl an die folgende Prozedur **Open_Table_Query_View** weitergegeben.

```
001 SUB Open_Table_Query_View(stName AS STRING, inType AS INTEGER)
002    DIM oController AS OBJECT
003    DIM oConnection AS OBJECT
004    oController = ThisDatabasedocument.CurrentController
005    IF NOT oController.isconnected THEN oController.connect
006    oConnection = oController.ActiveConnection
007    DIM URL AS NEW com.sun.star.util.URL
008    DIM Args(5) AS NEW com.sun.star.beans.PropertyValue
009    URL.Complete = ".component:DB/DataSourceBrowser"
010    Dispatch = StarDesktop.queryDispatch(URL,"_Blank",8)
011    Args(0).Name = "ActiveConnection"
012    Args(0).Value = oConnection
013    Args(1).Name = "CommandType"
014    Args(1).Value = inType   '0=Table 1=SQL_Query 2=Command
015    Args(2).Name = "Command"
016    Args(2).Value = stName
017    Args(3).Name = "ShowMenu"
018    Args(3).Value = True
019    Args(4).Name = "ShowTreeView"
020    Args(4).Value = False
021    Args(5).Name = "ShowTreeViewButton"
022    Args(5).Value = False
023    Dispatch.dispatch(URL, Args)
024 END SUB
```

Zuerst wird die Verbindung zur Datenbank hergestellt, sofern sie noch nicht existiert. Diese Verbindung muss mit einigen zusätzlichen Informationen, unter anderem der Art des zu öffnenden Elementes (Tabelle oder Abfrage) sowie dem Namen des Elementes, in einem Array weiter gegeben werden.

Die Tabelle bzw. Abfrage wird schließlich über den **queryDispatch** mit dem Kommando **dispatch** geöffnet.

Hierarchische Listenfelder

Einstellungen in einem Listenfeld sollen die Einstellungen in einem zweiten Listenfeld direkt beeinflussen. Auf einfachere Art und Weise wurde dies schon bei der Filterung von Datensätzen weiter oben beschrieben. Jetzt soll aber hinzu kommen, dass das erste Listenfeld den Inhalt des zweiten Listenfeldes beeinflusst, der wiederum den Inhalt des dritten Listenfeldes beeinflusst usw.[40]

40 Siehe hierzu die Beispieldatenbank «Beispiel_Suchen_und_Filter.odb»

Jahrgang
1
2
3
4
5
6
7
8
9
10
11
12
13

Klasse
a
b
c
d
e
f
g

Name
Karl Müller
Evelyn Maier
Maria Gott
Eduard Abgefahren
Kurt Drechsler
Kunigunde Schimmel

Abbildung 61: Beispielhafte Listenfelder für eine hierarchische Anordnung von Listenfeldern.

In diesem Beispiel enthält Listenfeld 1 alle Jahrgänge der Schule. Die Klassen der jeweiligen Jahrgänge sind durch Buchstaben kenntlich gemacht. Die Namen enthalten die Schülerinnen und Schüler der Klasse.

Unter normalen Umständen zeigt das Listenfeld für den Jahrgang alle 13 Jahrgänge an. Das Listenfeld für die Klasse alle Buchstaben und das Listenfeld für die Schüler und Schülerinnen alle Schüler und Schülerinnen der Schule.

Wird mit hierarchischen Listenfeldern gearbeitet, so wird nach Auswahl des Jahrgangs das Listenfeld für die Klasse eingegrenzt. Es werden nur noch die Klassenbezeichnungen angezeigt, die es in dem Jahrgang tatsächlich gibt. So könnte eben bei steigender Schüler- und Schülerinnenzahl die Anzahl der Klassen im Jahrgang ebenfalls steigen. Das letzte Listenfeld, die Namen, ist jetzt bereits stark eingegrenzt. Statt alle vermutlich deutlich über 1000 Schüler und Schülerinnen anzuzeigen, listet das letzte Feld nur noch die ca. 30 Schüler und Schülerinnen der einen letztlich ausgewählten Klasse auf.

Zum Beginn steht nur die Auswahl des Jahrgangs zur Verfügung. Ist ein Jahrgang ausgewählt, so steht die (bereits eingeschränkte) Auswahl der Klasse zur Verfügung. Erst zum Schluss wird schließlich das Listenfeld für die Namen freigegeben.

Wird das Listenfeld des Jahrgangs geändert, so muss der Durchlauf wieder wie vorher starten. Wird nur das Listenfeld der Klasse geändert, so muss der Wert des Jahrgangs für das letzte Listenfeld der Namen weiter gelten.

Filterung des Formulars mit hierarchischen Listenfeldern

Um solch eine Funktion bereitzustellen, muss innerhalb eines Formulars eine Variable zwischengespeichert werden. Dies erfolgt in einem versteckten Kontrollfeld.

Der Makrostart wird an die Veränderung des Inhaltes eines Listenfeldes gekoppelt: **Eigenschaft Listenfeld → Ereignisse → Modifiziert.** In den Zusatzinformationen des Listenfeldes werden die notwendige Variablen gespeichert.

Hier der beispielhafte Inhalt der Zusatzinformationen:
`Jahrgang,verstecktes Kontrollfeld,Listenfeld 2`

Das aktuelle Listenfeld ist als «Listenfeld 1» bezeichnet. Dieses Listenfeld stellt den Inhalt des Tabellenfeldes «Jahrgang» dar. Nach diesem Eintrag muss also das darauffolgende Listenfeld gefiltert werden. Das versteckte Kontrollfeld ist in diesem Fall auch gleich mit dem entsprechenden Namen gekennzeichnet. Und schließlich wird noch darauf hingewiesen, dass ein 2. Listenfeld, «Listenfeld 2», existiert, an das die Filterung weiter gegeben wird.

```
001 SUB Hierarchisches_Kontrollfeld(oEvent AS OBJECT)
002    DIM oDoc AS OBJECT
003    DIM oDrawpage AS OBJECT
004    DIM oForm AS OBJECT
005    DIM oFeldHidden AS OBJECT
006    DIM oFeld AS OBJECT
007    DIM oFeld1 AS OBJECT
008    DIM stSql AS STRING
009    DIM aInhalt()
010    DIM stTag AS STRING
011    oFeld = oEvent.Source.Model
012    stTag = oFeld.Tag
013    oForm = oFeld.Parent
014    REM Tag wird unter den Zusatzinformationen eingegeben
015    REM Hier steht:
016    REM 0. Feldname des zu filterndes Feldes in der Tabelle,
017    REM 1. Feldname des versteckten Konrollfeldes, das den Filterwert speichern
          soll,
018    REM 2. eventuell weiteres Listfeld
019    REM Der Tag wird von dem auslösenden Element ausgelesen. Die Variable wird an
          die Prozedur weitergegeben, die gegebenenfalls alle weiteren Listenfelder
          einstellt.
020    aFilter() = Split(stTag, ",")
021    stFilter = ""
```

Nachdem die Variablen deklariert wurden, wird der Inhalt des Tags in ein Array übertragen. So kann auf die einzelnen Elemente zugegriffen werden. Anschließend wird der Zugang zu den verschiedenen Feldern im Formular deklariert.

Das Listenfeld wird aus dem Aufruf heraus ermittelt. Aus dem Listenfeld wird der Wert ausgelesen. Nur wenn dieser Wert einen Inhalt hat, wird er mit dem Feldnamen des zu filternden Feldes, in unserem Beispiel «Jahrgang», zu einer SQL-Bedingung kombiniert. Ansonsten bleibt der Filter leer. Sind die Listenfelder zur Filterung eines Formulars gedacht, dann ist kein verstecktes Kontrollfeld vorhanden. Unter dieser Bedingung wird der Filterwert direkt im Formular gespeichert.

```
022    IF Trim(aFilter(1)) = "" THEN
023       IF oFeld.getCurrentValue <> "" THEN
024          stFilter = """"+Trim(aFilter(0))+"""='"+oFeld.getCurrentValue()+"'"
```

Existiert bereits vorher ein Filter (weil es sich z.B. um das Listenfeld 2 handelt, das jetzt betätigt wurde), so wird der neue Inhalt an den vorherigen angehängt, der in dem versteckten Feld zwischengespeichert wurde.

```
025          IF oForm.Filter <> ""
                AND InStr(oForm.Filter, """"+Trim(aFilter(0))+"""='") = 0 THEN
026             stFilter = oForm.Filter + " AND " + stFilter
```

Dies darf allerdings nur dann geschehen, wenn das gleiche Feld noch nicht gefiltert wurde. Schließlich ist z.B. bei einer Filterung nach dem «Jahrgang» kein Datensatz unter «Name» mehr zu erwarten, wenn zusätzlich eine weitere Filterung nach «Jahrgang» erfolgt. Eine Person kann immer nur in einem «Jahrgang» existieren. Es muss also ausgeschlossen werden, dass in der Filterung der Filtername bereits vorkommt.

Existiert bereits ein Filter und kommt das Feld, nach dem gefiltert werden soll, bereits im Filter vor, so muss die vorherige Filterung ab diesem Feldnamen gelöscht und die neue Filterung eingefügt werden.

```
027          ELSEIF oForm.Filter <> "" THEN
028             stFilter = Left(oForm.Filter,
029             InStr(oForm.Filter, """"+Trim(aFilter(0))+"""='")-1) + stFilter
```

```
030         END IF
031      END IF
```

Anschließend wird der Filter in das Formular eingetragen. Dieser Filter kann auch leer sein, wenn direkt das erste Listenfeld ohne Inhalt gewählt wurde.

```
032      oForm.Filter = stFilter
033      oForm.reload()
```

Die gleiche Prozedur wird durchlaufen, wenn nicht ein Formular direkt gefiltert werden soll. In dem Fall wird der Filterwert in einem versteckten Kontrollfeld zwischengespeichert.

```
034   ELSE
035      oFeldHidden = oForm.getByName(Trim(aFilter(1)))
036      IF oFeld.getCurrentValue <>"" THEN
037         stFilter = """"+Trim(aFilter(0))+"""='"+oFeld.getCurrentValue()+"'"
038         IF oFeldHidden.HiddenValue <> ""
               AND InStr(oFeldHidden.HiddenValue, """"+Trim(aFilter(0))+"""='") = 0
               THEN
039            stFilter = oFeldHidden.HiddenValue + " AND " + stFilter
040         ELSEIF oFeldHidden.HiddenValue <> "" THEN
041            stFilter = Left(oFeldHidden.HiddenValue,
                  InStr(oFeldHidden.HiddenValue, """"+Trim(aFilter(0))+"""='")-1) +
                  stFilter
042         END IF
043      END IF
044      oFeldHidden.HiddenValue = stFilter
045   END IF
```

Ist in den Zusatzinformationen ein 4. Eintrag (Arraynummerierung beginnt bei 0!) vorhanden, so muss das folgende Listenfeld jetzt auf den entsprechenden Eintrag des aufrufenden Listenfeldes eingestellt werden.

```
046   IF UBound(aFilter()) > 1 THEN
047      oFeld1 = oForm.getByName(Trim(aFilter(2)))
048      aFilter1() = Split(oFeld1.Tag,",")
```

Die notwendigen Daten für die Filterung werden aus den Zusatzinformationen («Tag») des entsprechenden Listenfeldes ausgelesen. Leider ist es nicht möglich, lediglich den SQL-Code in dem Listenfeld neu zu schreiben und anschließend das Listenfeld einzulesen. Vielmehr müssen die entsprechenden Werte direkt in das Listenfeld geschrieben werden.

Bei der Erstellung des Codes wird davon ausgegangen, dass die Tabelle, auf der das Formular beruht, die gleiche ist, auf der auch die Listenfelder beruhen. Für eine Weitergabe von Fremdschlüsseln an die Tabelle ist so ein Listenfeld also erst einmal nicht gedacht.

```
049      IF oFeld.getCurrentValue <> "" THEN
050         stSql = "SELECT DISTINCT """+Trim(aFilter1(0))+""" FROM """+oForm.Command+
               """ WHERE "+stFilter+" ORDER BY """+Trim(aFilter1(0))+""""
051         oDatenquelle = ThisComponent.Parent.CurrentController
052         If NOT (oDatenquelle.isConnected()) THEN
053            oDatenquelle.connect()
054         END IF
055         oVerbindung = oDatenquelle.ActiveConnection()
056         oSQL_Anweisung = oVerbindung.createStatement()
057         oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
```

Die Werte werden in ein Array eingelesen. Das Array wird anschließend direkt in das Listenfeld übertragen. Die entsprechenden Zähler für das Array werden durch die Schleife kontinuierlich erhöht.

```
058         inZaehler = 0
059         WHILE oAbfrageergebnis.next
060            ReDim Preserve aInhalt(inZaehler)
061            aInhalt(inZaehler) = oAbfrageergebnis.getString(1)
062            inZaehler = inZaehler+1
063         WEND
064      ELSE
065         aInhalt(0) = ""
```

```
066         END IF
067         oFeld1.StringItemList = aInhalt()
```

Der Inhalte des Listenfeldes wurde neu erstellt. Das Listenfeld muss neu eingelesen werden. Anschließend wird anhand der Zusatzinformationen des neu eingestellten Listenfeldes jedes eventuell weiter folgende Listenfeld entsprechend geleert, indem eine Schleife für alle folgenden Listenfelder gestartet wird, bis eben ein letztes Listenfeld keinen 4. Eintrag in den Zusatzinformationen enthält.

```
068         oFeld1.refresh()
069         WHILE UBound(aFilter1()) > 1
070            DIM aLeer()
071            oFeld2 = oForm.getByName(Trim(aFilter1(2)))
072            DIM aFilter1()
073            aFilter1() = Split(oFeld2.Tag,",")
074            oFeld2.StringItemList = aLeer()
075            oFeld2.refresh()
076         WEND
077      END IF
078   END SUB
```

Die sichtbaren Inhalte des Listenfeldes werden in `oFeld1.StringItemList` gespeichert. Soll zusätzlich auch ein Wert gespeichert werden, der als Fremdschlüssel an die darunterliegende Tabelle weitergegeben wird, wie bei Listenfeldern in Formularen üblich, so ist dieser Wert in der Abfrage zusätzlich zu ermitteln und anschließend mit `oFeld1.ValueItemList` abzuspeichern.

Für so eine Erweiterung sind allerdings zusätzliche Variablen notwendig wie z.B. neben der Tabelle, in der die Werte des Formulars gespeichert werden, noch die Tabelle, aus der die Listenfeldinhalte gelesen werden.

Besondere Aufmerksamkeit ist dabei der Formulierung des Filters zu widmen.

```
001      stFilter = """"+Trim(aFilter(1))+"""='"+oFeld.getCurrentValue()+"'"
```

funktioniert dann nur noch, wenn es sich bei der zugrundeliegenden LO-Version um eine Version ab LO 4.1 handelt, da hier als `CurrentValue()` der Wert wiedergegeben wird, der auch abgespeichert wird – nicht der Wert, der lediglich angezeigt wird. Damit das einwandfrei über verschiedene Versionen hinweg funktioniert, sollte unter **Eigenschaften: Listenfeld → Daten → Gebundenes Feld → '0'** angegeben sein.

Hierarchische Listenfelder in der Formulareingabe nutzen

Auch bei der Eingabe von Formularen können solche hierarchischen Listenfelder genutzt werden. Die hier aufgeführten Makros erledigen dabei nur die notwendigen Grundlagen[41]. Die Felder für das 3. Listenfeld werden z.B. nicht automatisch zurückgestellt, wenn aus dem ersten Listenfeld ein neuer Wert ausgesucht wird.

Das Makro ist an **Eigenschaften: Listenfeld → Ereignisse → Vor dem Aktualisieren** gebunden. Diese Eigenschaft steht bei Listenfeldern auch innerhalb von Tabellenkontrollfeldern zur Verfügung. So ist das Makro sowohl bei Tabellenkontrollfeldern als auch bei einfachen Formularfeldern universell nutzbar.

```
001   SUB Listenfeldfilter(oEvent AS OBJECT)
002      DIM stSql(0) AS STRING
003      DIM oForm AS OBJECT
004      DIM oFeld AS OBJECT
005      DIM oFeld2 AS OBJECT
006      DIM stFeld AS STRING
007      DIM stWert AS STRING
008      DIM stFeld2 AS STRING
009      DIM stSqlFeld2 AS STRING
010      oFeld = oEvent.Source
011      stFeld = oFeld.DataField
012      stWert = oFeld.CurrentValue
013      oForm = oFeld.Parent ' oForm ist bei Tabellenkontrollfelder das Tabellenobjekt
```

41 Siehe hierzu die Beispieldatenbank «Beispiel_hierarchische_Listenfelder.odb»

```
014     stFeld2 = oFeld.Tag
015     oFeld2 = oForm.getByname(stFeld2)
016     stSqlFeld2 = oFeld2.ListSource(0)
```

In dem auslösenden Feld steht lediglich in den Zusatzinformationen der Name des Formularfeldes, das neu eingestellt werden soll.

Aus dem auslösenden Feld wird das Datenfeld ausgelesen. Sollte hier in der Datenbank für das Feld eine andere Bezeichnung gewählt worden sein als für die Abfrage im folgenden Listenfeld notwendig, so muss dies ebenfalls in den Zusatzinformationen aufgeführt werden.

Aus dem auslösenden Feld wird auch der momentane Wert ausgelesen. Das ist seit der Version LO 4.1 der Wert, der tatsächlich in der Datenbank abgespeichert wird.

Das Zielfeld wird angesteuert und der Inhalt der dort enthaltenen Abfrage über **`ListSource(0)`** ausgelesen. Da es sein kann, dass durch eine vorherige Betätigung des auslösenden Feldes hier bereits eine **`WHERE`**-Bedingung steht, muss diese gegebenenfalls in der folgenden Schleife entfernt werden.

```
017     IF InStr(stSqlFeld2, "WHERE") THEN
018        ar = Split(stSqlFeld2, "WHERE")
019        stSqlFeld2 = ar(0)
020     END IF
```

Zum Schluss wird der Code als der erste Wert des Arrays stSql zusammengestellt. Er wird als **`ListSource`** an das Zielfeld übergeben. Das Feld wird mit einem **`refresh`** auf den neuen Inhalt eingestellt.

```
021     stSql(0) = stSqlFeld2 & " WHERE """+stFeld+""" = '"+stWert+"'"
022     oFeld2.ListSource = stSql
023     oFeld2.refresh
024 END SUB
```

Wir so ein Makro für Listenfelder in einem **Tabellenkontrollfeld** genutzt, so muss **Formulareigenschaften → Daten → Daten ändern → 'Nein'** ausgewählt sein. Sonst werden die Listenfelder in den vorhergehenden Formularfeldern geändert und können dort gegebenenfalls die alten Daten nicht mehr anzeigen.

Die folgenden beiden Makros nutzen eine Filtertabelle. Das hat den Vorteil, dass in den Listenfeldern beliebiger SQL-Code stehen kann. Die Felder müssen lediglich in dem SQL-Code einen Verweis auf den Tabellenwert stehen haben. Das obere Beispiel funktioniert so wie aufgeschrieben hingegen nur, wenn der SQL-Code direkt nach der Tabellenbenennung endet und keine **`WHERE`**-Bedingung und keine Sortierung enthält.

```
001 SUB Listfeldfilter_Tabelle(oEvent AS OBJECT)
002     DIM oDatasource AS OBJECT
003     DIM oConnection AS OBJECT
004     DIM oSQL_Statement AS OBJECT
005     DIM oForm AS OBJECT
006     DIM oFeld AS OBJECT
007     DIM oFeld2 AS OBJECT
008     DIM stFeld AS STRING
009     DIM stWert AS STRING
010     DIM stSql AS STRING
011     oFeld = oEvent.Source
012     stFeld = oFeld.DataField
013     stWert = oFeld.CurrentValue
014     oForm = oFeld.Parent ' oForm ist bei Tabellenkontrollfelder das Tabellenobjekt
015     stFeld2 = oFeld.Tag
016     oFeld2 = oForm.getByname(stFeld2)
017     oDatasource = thisDatabaseDocument.CurrentController
018     IF NOT (oDatasource.isConnected()) THEN oDatasource.connect()
019     oConnection = oDatasource.ActiveConnection()
020     oSQL_Statement = oConnection.createStatement()
021     stSql ="UPDATE ""Filter"" SET """+stFeld+""" = '"+stWert+"' WHERE ""ID"" = TRUE"
022     oSQL_Statement.executeUpdate(stSql)
023     oFeld2.refresh
024 END SUB
```

Die Prozedur «Listfeldfilter_Tabelle» startet wie die vorhergehende Prozedur. Der SQL-Code des Zielfeldes spielt hier gar keine Rolle. Er muss lediglich so gestaltet sein, dass er durch Werte in der Tabelle "Filter" beeinflusst wird. Hier der Beispielcode für das Listenfeld, das die Klasse ausgeben soll:

```
SELECT "Klasse", "ID" FROM "Klasse"
WHERE "J_ID" =
    COALESCE ( ( SELECT "J_ID" FROM "Filter" WHERE "ID" = TRUE ), "J_ID" )
ORDER BY "Klasse" ASC
```

In dem Makro wird jetzt schlicht z.B. der Inhalt des Feldes "J_ID" aus der Tabelle "Filter" neu beschrieben und das Ziellistenfeld neu eingelesen.

Der Reset des Filters ist notwendig, damit bei der nächsten Eingabe wieder alle Werte in den Listenfeldern vorhanden sind. Er wird deshalb an die **Formulareigenschaften → Ereignisse → Vor dem Datensatzwechsel** gebunden.

```
SUB Filter_Reset(oEvent AS OBJECT)
   DIM oConnection AS OBJECT
   DIM oSQL_Statement AS OBJECT
   DIM oForm AS OBJECT
   DIM stSql AS STRING
   oForm = oEvent.Source
   IF inStr(oForm.ImplementationName,"ODatabaseForm") THEN
      oConnection = oForm.activeConnection()
      oSQL_Statement = oConnection.createStatement()
      stSql = "UPDATE ""Filter"" SET ""J_ID"" = NULL, ""K_ID"" = NULL
         WHERE ""ID"" = TRUE"
      oSQL_Statement.executeUpdate(stSql)
   END IF
END SUB
```

Zeiteingaben mit Millisekunden

Um Zeiten im Millisekunden-Bereich zu speichern, ist in der Tabelle ein Timestamp-Feld erforderlich, das zudem per SQL separat darauf eingestellt wird (siehe *Zeitfelder in Tabellen*) (HSQLDB, FIREBIRD erlaubt auch Millisekunden für normale Zeiten). Ein solches Feld kann vom Formular aus mit einem formatierten Feld beschrieben werden, das auch das Format **MM:SS,00** anbietet. Allerdings scheitert der erste Schreibversuch daran, dass der Eingabe der entsprechende Datumszusatz fehlt. Dies kann mit dem folgenden Makro erreicht werden, das an die **Formulareigenschaften → Ereignisse → Vor der Datensatzaktion** gebunden wird:

```
SUB Timestamp
   DIM unoStmp AS NEW com.sun.star.util.DateTime
   DIM oDoc AS OBJECT
   DIM oDrawpage AS OBJECT
   DIM oForm AS OBJECT
   DIM oFeld AS OBJECT
   DIM stZeit AS STRING
   DIM ar()
   DIM arMandS()
   DIM loNano AS LONG
   DIM inSecond AS INTEGER
   DIM inMinute AS INTEGER
   oDoc = thisComponent
   oDrawpage = oDoc.Drawpage
   oForm = oDrawpage.Forms.getByName("MainForm")
   oFeld = oForm.getByName("Zeit")
   stZeit = oFeld.Text
```

Die Variablen werden vorher deklariert. Nur wenn das Feld «Zeit» einen Inhalt hat, wird der weitere Code ausgeführt. Sonst tritt der Mechanismus des Formulars in Kraft, der das Feld auf **NULL** setzt.

```
   IF stZeit <> "" THEN
      ar() = Split(stZeit,",")
```

```
020         loNano = CLng(ar(1)&"0000000")
021         arMandS() = Split(ar(0),":")
022         inSecond = Cint(arMandS(1))
023         inMinute = Cint(arMandS(0))
```

Die Einträge aus dem Feld «Zeit» werden in ihre Bestandteile zerlegt.

Zuerst werden die Hundertstelsekunden abgetrennt und mit so vielen Nullen rechts aufgefüllt, dass sich insgesamt eine neunstellige Zahl ergibt. Eine so hohe Zahl kann nur in einer Long-Variablen gespeichert werden.

Anschließend werden aus dem verbleibenden Rest durch eine Trennung am Trennzeichen «:» die Minuten von den Sekunden getrennt und in Integer-Zahlen umgewandelt.

```
024         WITH unoStmp
025            .NanoSeconds = loNano
026            .Seconds = inSecond
027            .Minutes = inMinute
028            .Hours = 0
029            .Day = 30
030            .Month = 12
031            .Year = 1899
032         END WITH
```

Dem Zeitstempel wird nun das Standarddatum 30.12.1899 zugewiesen, das dem Standard-Startdatum von LibreOffice entspricht. Hier kann natürlich auch das aktuelle Datum mitgespeichert werden.

Hinweis

Aktuelles Datum ermitteln und speichern:

```
001 DIM Jetzt AS DATE
002 Jetzt = Now()
003 WITH unoStmp
004    .NanoSeconds = loNano
005    .Seconds = inSecond
006    .Minutes = inMinute
007    .Hours = Hour(Jetzt)
008    .Day = Day(Jetzt)
009    .Month = Month(Jetzt)
010    .Year = Year(Jetzt)
011 END WITH
```

```
033         oFeld.BoundField.updateTimestamp(unoStmp)
034      END IF
035   END SUB
```

Anschließend wird der erzeugte Zeitstempel über **`updateTimestamp`** in das Feld übertragen und mit dem Formular abgespeichert.

In älteren Anleitungen wird hier statt **`NanoSeconds`** der Begriff **`HundrethSeconds`** verwendet. Dieser entspricht aber nicht der API von LibreOffice und erzeugt deshalb nur Fehlermeldungen.

Ein Ereignis – mehrere Implementationen

Bei Formularen kommt es vor, dass ein Makro, mit einem Ereignis verknüpft, gleich zweimal ausgeführt wird. Dies liegt daran, dass mehrere Prozesse gleichzeitig z.B. mit dem Abspeichern eines geänderten Datensatzes verbunden sind. Die unterschiedlichen Ursachen für so ein Ereignis lassen sich folgendermaßen ermitteln:

```
001 SUB Ereignisursache_ermitteln(oEvent AS OBJECT)
002    DIM oForm AS OBJECT
003    oForm = oEvent.Source
004    MsgBox oForm.ImplementationName
005 END SUB
```

Beim Abspeichern eines geänderten Datensatzes ergeben sich so zwei Implementationsnamen: **org.openoffice.comp.svx.FormController** und **com.sun.star.comp.forms.ODatabaseForm**. Über diese Namen kann jetzt gesteuert werden, dass ein Makro letztlich nur einmal den ganzen Code durchläuft. Die doppelte Durchführung ist oft nur eine (kleine) Bremse im Programmablauf. Sie kann aber auch dazu führen, dass sich z.B. ein Cursor nicht nur einen, sondern gleich zwei Datensätze zurück bewegt. Die Implementationen lassen auch nur bestimmte Befehle zu, so dass eine Kenntnis des Namens der Implementation von Bedeutung sein kann.

Sicherer soll hier die Abfrage sein, ob eine der Implementationen ein bestimmtes **UnoInterface** benutzt. Das ist fest in der API verankert und wird damit wohl nicht geändert:

```
001  IF hasUnoInterfaces(oForm, "com.sun.star.form.XForm" ) THEN
```

weist darauf hin, dass es sich um die Implementation **com.sun.star.comp.forms.ODatabaseForm** handelt. Auch die Ermittlung des **ServiceName** kann eine klare Abgrenzung bewirken. Die **SupportedServiceNames** sind in einem Array in **oForm** enthalten. Über

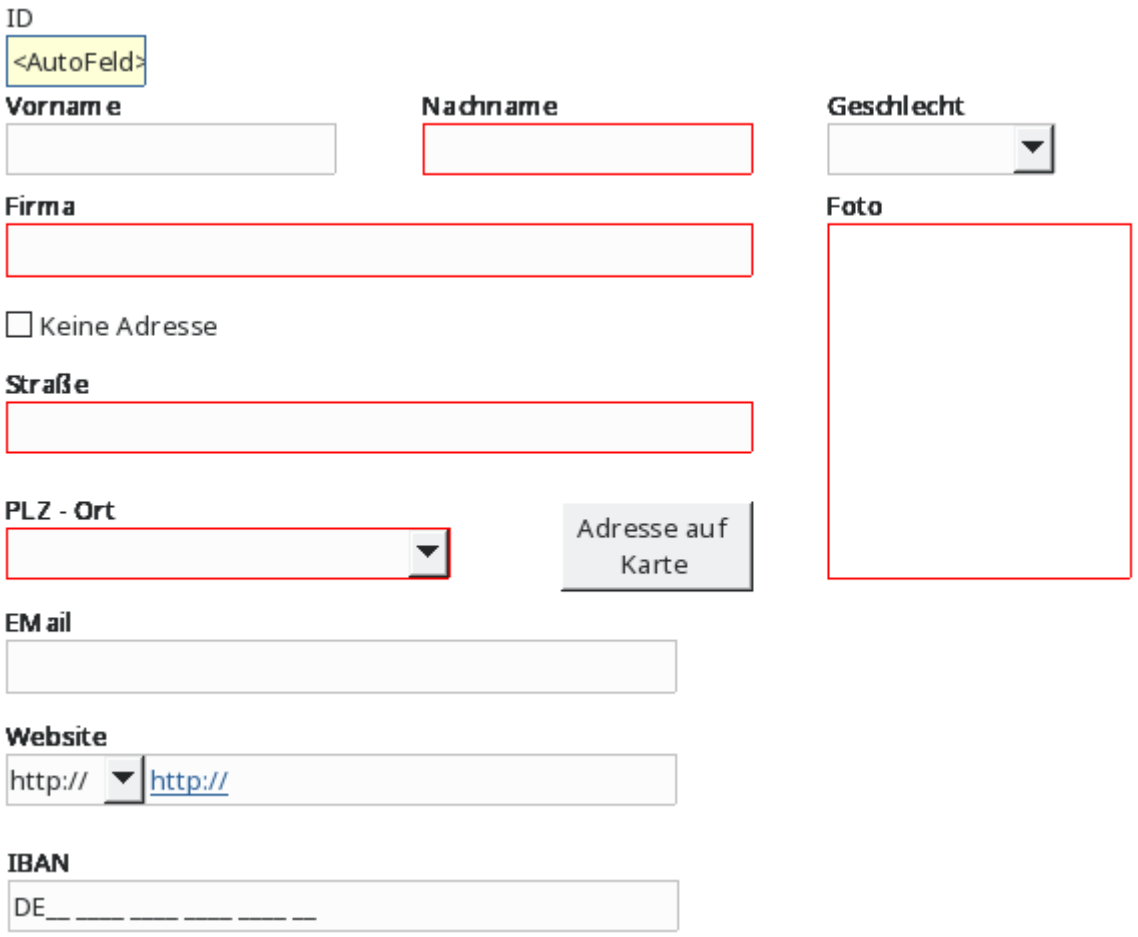
```
001  IF oForm.supportsService("com.sun.star.form.component.DataForm") THEN
```

kann hier **ODatabaseForm** ermittelt werden.

Eingabekontrolle bei Formularen

Ein Formular sollte für die Eingabe so weit wie möglich abgesichert sein, bevor die Daten in die Datenbank geschrieben werden. Dies erfolgt natürlich schon allein dadurch, dass Felder des Formulars passend zu den Inhalten aus der Datenbank gewählt werden. Auch lassen sich Felder so einstellen, dass sie eine zwingende Eingabe benötigen. Diese zwingende Eingabe muss zur Zeit allerdings auch in der Tabelle der Datenbank definiert sein. Die diesem Abschnitt zugrundeliegende Datenbank[42] zeigt fehlende Eingabe direkt an und vermeidet in einigen Feldern auch eventuell fehlerhafte Eingaben.

42 Siehe hierzu die Beispieldatenbank «Beispiel_Formular_Eingabekontrolle.odb»



Mehrere Elemente des Formulars fallen sofort auf:

- Das Feld «ID» ist mit einem separaten Hintergrund und einer separaten Umrandung versehen. Es ist über **Eigenschaften → Nur lesen** von der Eingabe ausgeschlossen. Außerdem ist **Tabstop → Nein** gesetzt. Hierzu ist kein Makroeinsatz erforderlich.
- Die Felder «Nachname», «Firma», «Straße», «PLZ – Ort» und «Foto» sind rot umrandet. Hier ist eine Eingabe erforderlich. Die Umrandung verschwindet sobald dort ein Text eingetragen wird.
- Das Feld «Keine Adresse» kann hier genutzt werden um die Felder «Straße» und «PLZ – Ort» zu deaktivieren. Die rote Umrandung verschwindet dann, die Hintergrundfarbe wird geändert und eine Eingabe ist nicht mehr möglich.
- Das Feld «Website» ist mit einem zusätzlichen Listenfeld versehen. Dies soll sicherstellen, dass die Eingabe mit 'http://' oder 'https://' beginnt. Die entsprechende Vorwahl steht bereits in dem Eingabefeld und wird durch die weitere Eingabe ergänzt.
- Bei dem Feld «Website» fällt bereits auf, dass in dem Feld der Text in blauer Farbe mit einfacher Unterstreichung abgebildet wird. Das Feld enthält einen Link, der bei gedrückter Strg-Taste mit der Maustaste angeklickt und geöffnet werden kann. Die entsprechende Funktion ist auch bei den Feldern «Email» und «Foto» (zur Großdarstellung des Fotos) hinterlegt.
- Das Feld «IBAN» ist ein Maskiertes Feld, das nur die Eingabe von Zahlen erlaubt. Hier erfolgt nach dem Verlassen des Feldes eine entsprechende Überprüfung auf korrekte Eingabe. Ähnliche Funktionen sind bei den Feldern «EMail» und «Website» hinterlegt.
- Der Button «Adresse auf Karte» schließlich öffnet eine Karte, auf der die eingegebene Adresse im Webbrowser angezeigt wird, sofern sie wirklich existiert und bei OpenStreetmap verzeichnet ist.

Erforderliche Eingaben absichern

Zu beginn werden einige globale Variablen festgelegt. Die Standardfarbe für den Rahmen und den Hintergrund eines Feldes muss verfügbar sein, ebenso die Farbe, in der der Rahmen erscheinen soll, wenn eine Eingabe notwendig ist. Alle Formularfelder, bei denen zum Start des Formulars **Daten → Eingabe erforderlich → Ja** eingestellt ist, werden in einem zentralen Array gespeichert. Ohne diese Speicherung wäre es nicht möglich, die erforderliche Eingabe z.B. für die Adresse ein- und wieder auszuschalten.

```
001 GLOBAL loBorderDefault AS LONG
002 GLOBAL loBorderInputRequired AS LONG
003 GLOBAL loColorStandard AS LONG
004 GLOBAL arFormInputRequired()
```

Die globalen Variablen werden beim Öffnen des Formulardokumentes mit Inhalt versehen. Dies regelt die Prozedur «FormVars».

Die Farbvariablen werden direkt festgelegt. Anschließend wird das gesamte Formular durchgegangen und alle Felder einzeln untersucht. Nur die Felder, die zu dem **DataAwareControlModel** gehören, können auch Daten aufnehmen. Andere Felder wie Beschriftungsfelder, Buttons oder versteckte Felder können nicht für eine Eingabe genutzt werden.

Jetzt kann noch vorkommen, dass bei einem Feld zwar die Eingabe notwendig ist, leider aber keine Umrandungsfarbe einstellbar ist. Deswegen werden schließlich in das Array für die als notwendig zu versehenden Eingaben nur die übernommen, die auch die Eigenschaft **BorderColor** unterstützen.

```
001 SUB FormVars(oEvent AS OBJECT)
002    DIM oForm AS OBJECT, oField AS OBJECT
003    DIM k AS INTEGER, i AS INTEGER
004    loBorderDefault = RGB(192,192,192) 'Grau
005    loBorderInputRequired = RGB(255,0,0) 'Rot
006    loColorStandard = RGB(250,250,250) 'sehr helles Grau
007    oForm = oEvent.Source
008    FOR i = 0 TO oForm.Count - 1
009       oField = oForm.getByIndex(i)
010       IF oField.supportsService("com.sun.star.form.DataAwareControlModel") THEN
011          IF oField.InputRequired THEN
012             IF oField.getPropertySetInfo.hasPropertyByName("BorderColor") THEN
013                REDIM PRESERVE arFormInputRequired(k)
014                arFormInputRequired(k) = oField.Name
015                k = k + 1
016             END IF
017          END IF
018       END IF
019    NEXT
020    FormChange(oEvent)
021 END SUB
```

In der vorhergehenden Prozedur wird bereits die Prozedur «FormChange» aufgerufen. Mit dieser Prozedur wird die Kennzeichnung der notwendigen Eingaben vorgenommen.

```
001 SUB FormChange(oEvent AS OBJECT)
002    DIM oForm AS OBJECT, oField AS OBJECT
003    DIM i AS INTEGER, n AS INTEGER, k AS INTEGER
004    DIM stTest AS STRING
005    DIM a(), aa(), ab()
006    oForm = oEvent.Source
```

In der ersten Schleife durch das Array der Formularfelder, bei denen eine Eingabe nötig ist, wird überprüft, ob das Feld einen Wert enthält. Hier muss zwischen Feldern unterschieden werden, die eine Verbindung zur Datenbank haben und solchen, die ohne Verbindung zur Datenbank existieren (Kombinationsfeld, für das der Fremdschlüssel über Makro ermittelt wird). Die einfache Abfrage nach **CurrentValue** führt bei Bildfeldern zu einem Fehler, weil dort diese Eigenschaft nicht existiert. Ist dies nicht der Fall, dann wird rot umrandet. Ist dies der Fall, dann wird die Standardumrandung gewählt.

```
007     FOR i = LBound(arFormInputRequired()) TO UBound(arFormInputRequired())
008        oField = oForm.getByName(arFormInputRequired(i))
009        oField.InputRequired = True
010        IF NOT IsNULL(oField.BoundField) THEN
011           IF oField.BoundField.String = "" THEN
012              oField.BorderColor = loBorderInputRequired
013           ELSE
014              oField.BorderColor = loBorderDefault
015           END IF
016        ELSEIF oField.CurrentValue = "" THEN
017           oField.BorderColor = loBorderInputRequired
018        ELSE
019           oField.BorderColor = loBorderDefault
020        END IF
021     NEXT
```

Die darauffolgende zweite Schleife ist nur deswegen notwendig, weil das Formular ein Feld enthält, das die Eingabe für die Adresse ausschließt. In Abhängigkeit von diesem Feld muss also noch einmal überprüft werden, welche Felder denn jetzt eine Eingabe erfordern und mit einer roten Umrandung gezeigt werden müssen.

```
022     FOR i = LBound(arFormInputRequired()) TO UBound(arFormInputRequired())
023        oField = oForm.getByName(arFormInputRequired(i))
024        IF NOT IsNULL(oField.BoundField) THEN
025           stTest = oField.BoundField.String
026        ELSE
027           stTest = oField.CurrentValue
028        END IF
029        IF stTest <> "" AND oField.Tag <> "" THEN
```

In den Feldern, für die eine Eingabe notwendig ist, wird vermerkt von welchem Feld diese Eingabe abhängt. In der Beispieldatenbank steht in den Zusatzinformationen von «Nachname» **notrequired[txtFirma]**. Das soll bedeuten: Ist ein Nachname eingetragen, so ist bei der Firma kein Eintrag mehr notwendig. Entsprechend steht in den Zusatzinformationen von «Firma» **notrequired[txtNachname]**. Auch für andere Bereiche wurde in der Beispieldatenbank nach einem Kennwort eine Liste der entsprechenden Felder in eckigen Klammern gewählt. In den Zusatzinformationen können so mehrere Kennworte mit entsprechenden Listen untergebracht werden. Die abschließende eckige Klammer ist der Trenner, nach dem jetzt zuerst einmal gesucht wird:

```
030           a = split(oField.Tag,"]")
031           FOR n = LBound(a()) TO Ubound(a())-1
```

Da die abschließende Klammer auch am Ende aller Eintragungen steht ist das letzte Arrayelement auf jeden Fall leer. Die Schleife muss also nur bis zum vorletzten Arrayelement laufen.

Enthält das Arrayelement den Begriff «notrequired», so wird hier jetzt weiter nach den enthaltenen Felder gesucht. Zuerst wird mit Hilfe der geöffneten eckigen Klammer das Kennwort von den Feldbezeichnungen getrennt, dann werden die Feldbezeichnungen getrennt, sofern überhaupt innerhalb der eckigen Klammern mehrerer Bezeichnungen, getrennt durch ein Komma, existieren.

Die Felder, bei denen jetzt kein Eintrag mehr notwendig sind, werden mit einem normalen Standardrahmen versehen. Die erforderliche Eingabe wird auf **False** gestellt.

```
032              IF InStr(a(n),"notrequired") THEN
033                 aa = split(a(n),"[")
034                 ab = split(aa(1),",")
035                 FOR k = LBound(ab()) TO UBound(ab())
036                    oField = oForm.getByName(ab(k))
037                    oField.BorderColor = loBorderDefault
038                    oField.InputRequired = False
039                 NEXT
040              END IF
041           NEXT
042        END IF
```

```
043     NEXT
044 END SUB
```

Die folgende Prozedur «NotRequired entspricht in Teilen der vorhergehenden Prozedur. Sie wird allerdings beim Verlassen eines Formularfeldes, nicht beim Wechsel eine Formulars aufgerufen. Hier wird nach dem Verlassen ein anderes Feld auf **Eingabe erforderlich → Nein** gesetzt, wenn das Ausgangsfeld einen Inhalt enthält. Enthält es keinen Inhalt, so wird bei **Eingabe erforderlich → Ja** gesetzt. Entsprechend werden auch die Rahmenfarben angepasst.

```
001 SUB NotRequired(oEvent AS OBJECT)
002     DIM oFieldStart AS OBJECT, oForm AS OBJECT
003     DIM n AS INTEGER, k AS INTEGER
004     DIM a(), aa(), ab()
005     oFieldStart = oEvent.Source.Model
006     oForm = oFieldStart.Parent
007     a = split(oFieldStart.Tag,"]")
008     FOR n = LBound(a()) TO UBound(a())-1
009        IF InStr(a(n),"notrequired") THEN
010           aa = split(a(n),"[")
011           ab = split(aa(1),",")
012           FOR k = LBound(ab()) TO UBound(ab())
013              oField = oForm.getByName(ab(k))
014              IF oFieldStart.CurrentValue <> "" THEN
015                 oField.BorderColor = loBorderDefault
016                 oField.InputRequired = False
017              ELSE
018                 oField.BorderColor = loBorderInputRequired
019                 oField.InputRequired = True
020              END IF
021           NEXT
022        END IF
023     NEXT
024 END SUB
```

Mit der Prozedur «EnableDisable» werden Felder abhängig von einem anderen Feld so eingeschaltet, dass gegebenenfalls keine Eingabe mehr notwendig ist. So steht in den Zusatzinformationen zu dem Markierfeld «Keine Adresse» **`inaktiv[txtStraße,comPLZOrt]`**. Es sollen also die Felder für die «Straße» und für «PLZ – Ort» inaktiv gesetzt werden, wenn das Markierfeld ausgewählt wurde (**`State = True`**)

Der Zugriff ist hier gleich dem der vorhergehenden Prozeduren. Wenn die Eingabe nicht mehr möglich sein soll, dann werden sowohl Rahmen als auch Hintergrundfarbe des Feldes auf die Standardrahmenfarbe eingestellt.

```
001 SUB EnableDisable(oEvent AS OBJECT)
002     DIM oForm AS OBJECT, oField AS OBJECT
003     DIM stTag AS STRING
004     DIM i AS INTEGER, k AS INTEGER
005     DIM a(), aa(), ab()
006     oForm = oEvent.Source.Model.Parent
007     stTag = oEvent.Source.Model.Tag
008     a = split(stTag,"]")
009     FOR i = LBound(a()) TO UBound(a())-1
010        IF InStr(a(i),"inaktiv") THEN
011           aa = split(a(i),"[")
012           ab = split(aa(1),",")
013           FOR k = LBound(ab()) TO UBound(ab())
014              oField = oForm.getByName(ab(k))
015              IF oEvent.Source.Model.State THEN
016                 oField.Enabled = False
017                 oField.BorderColor = loBorderDefault
018                 oField.BackgroundColor = loBorderDefault
019              ELSE
020                 oField.Enabled = True
021                 oField.BorderColor = loBorderInputRequired
022                 oField.BackgroundColor = loColorStandard
023              END IF
```

```
024             NEXT
025          END IF
026       NEXT
027    END SUB
```

Die Prozedur «FieldRequired» wird an die Felder gebunden, bei denen die Eingabe zu Beginn auf erforderlich gesetzt wurde. Enthält das Feld keinen Wert, so wird beim Fokusverlust der Rahmen rot dargestellt. Umgekehrt wird der Rahmen auf die Normalfarbe gesetzt, wenn das Feld Inhalt enthält. Ist außerdem in den Zusatzinformationen des Feldes noch das Stichwort 'notrequired' enthalten, dann wird die Prozedur «NotRequired» anschließend gestartet.

```
001  SUB FieldRequired(oEvent AS OBJECT)
002     DIM oField AS OBJECT
003     oField = oEvent.Source.Model
004     IF oField.CurrentValue <> "" THEN
005        oField.BorderColor = loBorderDefault
006     ELSE
007        oField.BorderColor = loBorderInputRequired
008     END IF
009     IF inStr(oField.Tag,"notrequired") THEN
010        NotRequired(oEvent)
011     END IF
012  END SUB
```

Fehlerhafte Eingaben vermeiden

In der Beispieldatenbank ist für mehrere Felder ein Prozedur eingebaut, die eine fehlerhafte Eingabe so weit wie möglich verhindern soll. Die folgende Prozedur erledigt dies für die Eingabe der IBAN. Sie ist an ein maskiertes Feld gebunden und wird beim Verlassen des Feldes aufgerufen.

```
001  SUB IBANValid(oEvent AS OBJECT)
002     DIM oField AS OBJECT, oForm AS OBJECT, oController AS OBJECT, oView AS OBJECT
003     DIM stMsg AS STRING, stText AS STRING, stLand AS STRING, stPruef AS STRING
004     DIM i AS INTEGER
005     DIM a()
006     oField = oEvent.Source.Model
007     stText = oField.Text
```

Nur wenn das Feld Text enthält soll die Prozedur auch ablaufen. Das bedeutet, wenn nur einmal der Cursor in dem maskierten Feld gelandet ist und keine Eingabe gemacht wurde ist auch nichts zu überprüfen. Bei der ersten Eingabe, die auch ruhig wieder gelöscht werden kann, würde allerdings die Eingabemaske als Text angesehen. Der Text würde, sofern ein Eintrag fehlt, jetzt mindestens einen Unterstrich '_' enthalten. Außerdem würde der Beginn des Textes 'DE' lauten.

```
008     IF stText <> "" THEN
009        IF inStr(stText,"_") THEN
010           IF Val(Mid(stText,3)) > 0 THEN
011              stMsg = "Die IBAN ist zu kurz."
```

Aus dem Text wird ab dem 3. Zeichen versucht, den Wert einer Dezimalzahl auszulesen. Schließlich wird die IBAN ab dem 3. Zeichen nur aus Zahlen zusammengesetzt. Leerzeichen ignoriert die Funktion **Val()**. Ist der Wert größer als 0 und enthält der Text gleichzeitig Unterstriche, so ist die IBAN-Angabe zu kurz. Ist der Wert 0, so soll keine Eingabe erfolgt sein. Das Feld wird mit dem Kommando **reset** zurückgesetzt und erscheint als leerer Text.

```
012           ELSE
013              oField.reset
014           END IF
```

Enthält das maskierte Feld an jeder Stelle Zeichen, so ist prinzipiell das Format korrekt. In Vierergruppen sind die Zahlen gebündelt eingegeben und vollständig. Die erste Vierergruppe enthält die Landesbezeichnung und die zweistellige Prüfziffer. Die Gruppen werden hier als ein Array aufgetrennt. Trenner ist standardmäßig das Leerzeichen.

```
015         ELSE
016            a = split(stText)
017            stLand = "1314" & Right(a(0),2)
```

Der Landescode wird in Zahlen umgesetzt. 'A': '10', 'B': '11', 'C': '12', 'D': '13', 'E': '14' usw., so dass aus 'DE' die Kombination '1314' wird. Dieser Kombination wird noch die Prüfziffer hinzugefügt. Die Berechnung der Korrektheit der Prüfziffer erfolgt nach dem Prinzip

1. alle Zahlen ab der 3. Zahl zusammen mit der Länderzahl und der Prüfziffer ergeben die Gesamtzahl
2. Die Gesamtzahl wird durch 97 geteilt
3. Aus der Ganzzahldivision muss ein Rest von 1 hervorgehen.

Leider ist dieses Verfahren nicht so einfach möglich, weil die Gesamtzahl zu groß ist. Der Variablentyp LONG kann maximal 2147483648 annehmen, aber nicht 24 Stellen. Das Rechenverfahren wird hier wie eine schriftliche Division in der Schule umgesetzt: Rest berechnen, weitere Werte hinzuholen, Rest berechnen usw. Nur bei der ersten Berechnung können hier 8 Zahlen aus dem Array übernommen werden. Ist dort der Rest über 21, so würde bereits die zweite Teilberechnung fehl schlagen. Deshalb wird bei den folgenden Teilberechnung jeweils nur mit einer Zugabe von einem Arrayelement, das eben 4 Zahlen enthält, weiter gerechnet.

```
018            i = cLng(a(1) & a(2)) Mod 97
019            i = cLng(i & a(3)) Mod 97
020            i = cLng(i & a(4)) Mod 97
021            i = cLng(i & a(5)) Mod 97
022            i = cLng(i & stLand) Mod 97
023            IF i <> 1 THEN
```

Ist die Prüfung fehl geschlagen, weil der Rest nicht gleich '1' ist, so wird hier als kleiner Zusatz noch die eventuell mögliche Prüfziffer berechnet. Dies ist bei einer angenommenen Prüfziffer von '00' der Rest zu '98'. Eine Gewähr für eine korrekte IBAN bietet dies aber nicht, da ja der Fehler auch an anderer Stelle innerhalb der IBAN liegen kann.

Ist die Prüfung fehl geschlagen, so muss eine Fehlermeldung auf dem Bildschirm präsentiert werden.

```
024               stPruef = "131400"
025               i = cLng(a(1) & a(2)) Mod 97
026               i = cLng(i & a(3)) Mod 97
027               i = cLng(i & a(4)) Mod 97
028               i = cLng(i & a(5)) Mod 97
029               i = cLng(i & stPruef) Mod 97
030               i = 98 - i
031               stMsg "Die IBAN ist fehlerhaft." & CHR(13) & "Die Prüfziffer müsste "
032                  & i & " sein."
033            END IF
034         END IF
```

Erfolgte eine Fehlermeldung, so darf die Prüfung hiermit aber nicht abgeschlossen sein. Der Cursor muss so lange ist das Eingabefeld zurückgesetzt werden bis die Prüfung das Ergebnis annimmt. Jede Prüfroutine sucht also bei einem Fehler anschließend den Controller des Dokumentes auf und setzt den Cursor in das Feld zurück.

```
035         IF stMsg <> "" THEN
036            msgbox (stMsg, 0, "Eingabe fehlerhaft")
037            oForm = oField.Parent
038            oController = thisComponent.getCurrentController()
039            oView = oController.getControl(oForm.getByname(oField.Name))
040            oView.setFocus
041         END IF
042      END IF
043   END SUB
```

Abspeichern nach erfolgter Kontrolle

Die Felder, bei denen eine Eingabe notwendig ist, verhindern nicht, dass eine Person dennoch eine Abspeicherung der Daten vornehmen will. Mit der folgenden Funktion «SaveRequired» wird jetzt noch einmal überprüft, ob in allen Feldern, für die eine Eingabe notwendig ist, auch eine Eingabe steht. Ansonsten wird die Speicherung unterbrochen und eine Fehlermeldung ausgegeben.

```
001 FUNCTION SaveRequired(oEvent AS OBJECT) AS BOOLEAN
002    DIM oForm AS OBJECT, oField AS OBJECT
003    DIM stLabel AS STRING
004    DIM k AS INTEGER, i AS INTEGER
005    SaveRequired = True
006    oForm = oEvent.Source
007    IF oForm.ImplementationName = "org.openoffice.comp.svx.FormController" THEN
```

Beim Abspeichern wird zuerst der «FormController» aktiviert. Anschließend auch noch die Implementation für «ODatabaseForm». Das Auslesen der Felder ist hier unterschiedlich, so dass direkt der «FormController» zur Auswertung genutzt wird. Für «oDatabaseForm» muss die Funktion grundsätzlich **True** wiedergeben. In dem «FormController» sind die einzelnen Felder nur über das Model erreichbar.

```
008       FOR i = 0 TO oForm.Model.Count - 1
009          oField = oForm.Model.getByIndex(i)
010          IF oField.supportsService("com.sun.star.form.DataAwareControlModel") THEN
011             IF oField.InputRequired AND NOT IsNULL(oField.BoundField) THEN
012                IF oField.BoundField.String = "" THEN
013                   stLabel = stLabel & ", " & oField.LabelControl.Label
014                   k = k + 1
015                END IF
016             ELSEIF oField.InputRequired THEN
017                IF oField.CurrentValue = "" THEN
018                   stLabel = stLabel & ", " & oField.LabelControl.Label
019                   k = k + 1
020                END IF
021             ELSE
022             END IF
023          END IF
024       NEXT
```

Die Schleife erfolgt hier in zwei Schritten. In dem Formular befindet sich ein Bildfeld, das die Eigenschaft **CurrentValue** nicht bedienen kann. Es befindet sich aber auch ein Kombinationsfeld darin, das gar nicht an die zugrundeliegende Tabelle gekoppelt ist und damit kein gebundenes Feld der Tabelle ansprechen kann.

Ist der Zähler größer als 1, so muss eine Fehlermeldung erfolgen. Hier wurde bereits über die den Feldern zugewiesenen Beschriftungsfelder (**Label**) entsprechend die lesbare Bezeichnung der leeren Felder herausgesucht. «stLabel» endet allerdings mit einem Komma, gefolgt von einer Leertaste. Dies ist für den Abschluss des Strings überflüssig und wird abgetrennt.

```
025       IF k > 0 THEN
026          stLabel = Right(stLabel, Len(stLabel)-2)
027          IF k = 1 Then
028             stText = "Für das Feld "
029          ELSE
030             stText = "Für die Felder "
031          END IF
032          MsgBox ("Alle rot umrandeten Felder müssen einen Inhalt aufweisen." &
                CHR(13) & stText & stLabel & " ist eine Eingabe erforderlich." ,
                0 ,"Speichern gestoppt")
033          SaveRequired = False
034       END IF
035    END IF
036 END FUNCTION
```

Bei einem Fehler gibt die Funktion **False** zurück. Die Abspeicherung wird unterbrochen und eine Suche nach den Fehlern kann beginnen.

Primärschlüssel aus Nummerierung und Jahreszahl

Bei der Erstellung von Rechnungen werden jährlich Bilanzen gezogen. Das führt manchmal zu dem Wunsch, die Rechnungstabellen einer Datenbank nach Jahren getrennt zu sichern und jedes Jahr mit einer neuen Tabelle zu beginnen.

Die folgende Makrolösung geht einen anderen Weg. Sie schreibt automatisch den Wert für das Feld «ID» in die Tabelle, berücksichtigt dabei aber das «Jahr», das in der Tabelle als zweiter Primärschlüssel existiert. So tauchen dann in der Tabelle als Primärschlüssel z.B. die folgenden Werte auf:[43]

Jahr	***ID***
2014	1
2014	2
2014	3
2015	1
2015	2

Damit lässt sich eine auf das Jahr bezogene Übersicht auch in den Dokumenten besser erzeugen.

```
001 SUB Datum_aktuell_ID_einfuegen
002    DIM oDatenquelle AS OBJECT
003    DIM oVerbindung AS OBJECT
004    DIM oSQL_Anweisung AS OBJECT
005    DIM stSql AS STRING
006    DIM oAbfrageergebnis AS OBJECT
007    DIM oDoc AS OBJECT
008    DIM oDrawpage AS OBJECT
009    DIM oForm AS OBJECT
010    DIM oFeld1 AS OBJECT
011    DIM oFeld2 AS OBJECT
012    DIM oFeld3 AS OBJECT
013    DIM inIDneu AS INTEGER
014    DIM inYear AS INTEGER
015    DIM unoDate
016    oDoc = thisComponent
017    oDrawpage = oDoc.drawpage
018    oForm = oDrawpage.forms.getByName("MainForm")
019    oFeld1 = oForm.getByName("fmtJahr")
020    oFeld2 = oForm.getByName("fmtID")
021    oFeld3 = oForm.getByName("datDatum")
022    IF IsEmpty(oFeld2.getCurrentValue()) THEN
023       IF IsEmpty(oFeld3.getCurrentValue()) THEN
024          unoDate = createUnoStruct("com.sun.star.util.Date")
025          unoDate.Year = Year(Date)
026          unoDate.Month = Month(Date)
027          unoDate.Day = Day(Date)
028          inYear = Year(Date)
029       ELSE
030          inYear = oFeld3.CurrentValue.Year
031       END IF
032       oDatenquelle = ThisComponent.Parent.CurrentController
033       If NOT (oDatenquelle.isConnected()) THEN
034          oDatenquelle.connect()
035       END IF
036       oVerbindung = oDatenquelle.ActiveConnection()
037       oSQL_Anweisung = oVerbindung.createStatement()
038       stSql = "SELECT MAX( ""ID"" )+1 FROM ""Auftraege"" WHERE ""Jahr"" = '"
             + inYear + "'"
```

43 Dem Handbuch liegt die Datenbank «Beispiel_Fortlaufende_Nummer_Jahr.odb» bei.

```
039         oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
040         WHILE oAbfrageergebnis.next
041            inIDneu = oAbfrageergebnis.getInt(1)
042         WEND
043         IF inIDneu = 0 THEN
044            inIDneu = 1
045         END IF
046         oFeld1.BoundField.updateInt(inYear)
047         oFeld2.BoundField.updateInt(inIDneu)
048         IF IsEmpty(oFeld3.getCurrentValue()) THEN
049            oFeld3.BoundField.updateDate(unoDate)
050         END IF
051      END IF
052   END SUB
```

Alle Variablen werden deklariert. Die Formularfelder in dem Hauptformular werden angesteuert. Der Rest des Codes läuft nur ab, wenn der Eintrag für das Formularfeld «fmtID» noch leer ist. Dann wird, wenn nicht schon ein Datum eingegeben wurde, zuerst ein Datumsstruct erstellt, um das aktuelle Datum und das aktuelle Jahr in die entsprechenden Felder übertragen zu können. Anschließend wird der Kontakt zu der Datenbank aufgebaut, sofern noch kein Kontakt existiert. Es wird zu dem höchsten Eintrag des Feldes "ID", bezogen auf das Jahr des Datumsfeldes, der Wert '1' addiert. Bleibt die Abfrage leer, so existiert noch kein Eintrag in dem Feld "ID". Jetzt könnte genauso gut '0' direkt in das Formularfeld «fmtID» eingetragen werden. Die Numerierung für die Aufträge sollte aber mit '1' beginnen, so dass der Variablen «inIDneu» eine '1' zugewiesen wird.

Die ermittelten Werte für das Jahr, die ID und, sofern nicht bereits ein Datum eingetragen wurde, das aktuelle Datum, werden schließlich in das Formular übertragen.

Im Formular sind die Felder für die Primärschlüssel "ID" und "Jahr" schreibgeschützt. Die Zuweisung kann so nur durch das Makro erfolgen.

Datenbankaufgaben mit Makros erweitert

Verbindung mit Datenbanken erzeugen

```
001 oDatasource = ThisComponent.Parent.DataSource
002 IF NOT oDatasource.IsPasswordRequired THEN
003    oConnection = oDatasource.GetConnection("","")
```

Hier wäre es möglich, fest einen Benutzernamen und ein Passwort einzugeben, wenn eine Passworteingabe erforderlich wäre. In den Klammer steht dann ("Benutzername","Passwort").

Statt einen Benutzernamen und ein Passwort in Reinschrift einzutragen, wird für diesen Fall der Dialog für den Passwortschutz aufgerufen:

```
004 ELSE
005    oAuthentication = createUnoService("com.sun.star.sdb.InteractionHandler")
006    oConnection = oDatasource.ConnectWithCompletion(oAuthentication)
007 END IF
```

Dies funktioniert aber nicht, wenn bei der Verbindung bereits eine Benutzername- und Passworteingabe in Base vorgegeben wurde. Hier muss mit

```
oDatasource.Password = "mein Passwort"
```

das Passwort der Verbindung mitgegeben werden. Dann erscheint der Dialog nicht mehr. Anschließend muss noch mit der Datenquelle verbunden werden, um direkt auf z.B. ein Formular zugreifen zu können:

```
008 ThisComponent.CurrentController.Connect()
```

Wird allerdings von einem Formular innerhalb der Base-Datei auf die Datenbank zugegriffen, so reicht bereits

```
001 oDatasource = ThisComponent.Parent.CurrentController
```

```
IF NOT (oDatasource.isConnected()) Then
   oDatasource.connect()
End IF
oConnection = oDatasource.ActiveConnection()
```

Die Datenbank ist hier bekannt, ein Nutzername und ein Passwort sind nicht erforderlich, da diese bereits in den Grundeinstellungen von Base für die internen Datenbankversionen ausgeschaltet sind.

Für Formulare außerhalb von Base wird die Verbindung über das erste Formular hergestellt:

```
oDatasource = ThisComponent.Drawpage.Forms(0)
oConnection = oDatasource.ActiveConnection
```

Hinweis

Es ist auch möglich, eine Datenbankverbindung ohne eine vorliegende Datenbankdatei zu erzeugen. Dies dürfte dann sinnvoll sein, wenn nur einzelne Informationen aus einer Datenquelle ausgelesen werden sollen und so etwas wie abgespeicherte Abfragen, Formulare und Berichte nicht benötigt werden.
Siehe hierzu die Ausführungen von Andrew Pitonyak in *https://www.pitonyak.org/database/AndrewBase.pdf* . Im Kapitel «Connections without a data source» ab S. 91 wird hier eine Verbindung über

```
oManager = CreateUnoService("com.sun.star.sdbc.DriverManager")
```

beschrieben. Diese Verbindung wird bei den Beispielen in diesem Handbuch bisher nicht weiter genutzt.

Daten von einer Datenbank in eine andere kopieren

Die interne Datenbank ist erst einmal eine Ein-Benutzer-Datenbank. Die Daten werden innerhalb der *.odb-Datei abgespeichert. Ein Austausch von Daten zwischen verschiedenen Datenbankdateien ist eigentlich nicht vorgesehen, über Export und Import allerdings möglich.

Manchmal werden aber auch *.odb-Dateien so eingesetzt, dass ein möglichst automatischer Datenaustausch von einer Datenbankdatei zu einer anderen erfolgen soll. Die folgende Prozedur kann da hilfreich sein.[44]

Nach der Deklaration der Variablen wird der Pfad der aktuellen Datenbankdatei von einem Button im Formular aus ausgelesen. Von dem Pfad wird der Dateiname abgetrennt. Die Zieldatei für die Daten befindet sich ebenfalls in dem Verzeichnis. Der Name dieser Datei wird jetzt an den Pfad angehängt, damit der Kontakt zur Zieldatenbankdatei erstellt werden kann.

Der Kontakt zur Ausgangsdatenbank wird im Verhältnis zum Formular ermittelt, in dem der Button liegt: **`ThisComponent.Parent.CurrentController`**. Der Kontakt zur externen Datenbank wird über den **`DatabaseContext`** und den Pfad zur Datenbank erstellt.

```
SUB Datenkopie
   DIM oDatabaseContext AS OBJECT
   DIM oDatenquelle AS OBJECT
   DIM oDatenquelleZiel AS OBJECT
   DIM oVerbindung AS OBJECT
   DIM oVerbindungZiel AS OBJECT
   DIM oDB AS OBJECT
   DIM oSQL_Anweisung AS OBJECT
   DIM oSQL_AnweisungZiel AS OBJECT
   DIM oAbfrageergebnis AS OBJECT
   DIM oAbfrageergebnisZiel AS OBJECT
   DIM stSql AS String
   DIM stSqlZiel AS String
   DIM inID AS INTEGER
   DIM inIDZiel AS INTEGER
   DIM stName AS STRING
```

44 Das Beispiel "Datenkopie_Quelle_Ziel" ist als gepacktes Verzeichnis diesem Handbuch beigefügt.

```
017     DIM stOrt AS STRING
018     oDB = ThisComponent.Parent
019     stDir = Left(oDB.Location,Len(oDB.Location)-Len(ConvertToURL(oDB.Title))+8)
020     stDir = ConvertToUrl(stDir & "ZielDB.odb")
021     oDatenquelle = ThisComponent.Parent.CurrentController
022     If NOT (oDatenquelle.isConnected()) THEN
023        oDatenquelle.connect()
024     END IF
025     oVerbindung = oDatenquelle.ActiveConnection()
026     oDatabaseContext = createUnoService("com.sun.star.sdb.DatabaseContext")
027     oDatenquelleZiel = oDatabaseContext.getByName(stDir)
028     oVerbindungZiel = oDatenquelleZiel.GetConnection("","")
029     oSQL_Anweisung = oVerbindung.createStatement()
030     stSql = "SELECT * FROM ""Tabelle"""
031     oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
032     WHILE oAbfrageergebnis.next
033        inID = oAbfrageergebnis.getInt(1)
034        stName = oAbfrageergebnis.getString(2)
035        stOrt = oAbfrageergebnis.getString(3)
036        oSQL_AnweisungZiel = oVerbindungZiel.createStatement()
037        stSqlZiel = "SELECT ""ID"" FROM ""Tabelle"" WHERE ""ID"" = '"+inID+"'"
038        oAbfrageergebnisZiel = oSQL_AnweisungZiel.executeQuery(stSqlZiel)
039        inIDZiel = - 1
040        WHILE oAbfrageergebnisZiel.next
041           inIDZiel = oAbfrageergebnisZiel.getInt(1)
042        WEND
043        IF inIDZiel = - 1 THEN
044           stSqlZiel = "INSERT INTO ""Tabelle"" (""ID"",""Name"",""Ort"") VALUES
              ('"+inID+"','"+stName+"','"+stOrt+"')"
045           oSQL_AnweisungZiel.executeUpdate(stSqlZiel)
046        END IF
047     WEND
048  END SUB
```

Die komplette Tabelle der Ausgangsdatenbank wird ausgelesen und Zeile für Zeile anschließend über den Kontakt zur Zieldatenbank in die Tabelle der Zieldatenbank eingefügt. Vor dem Einfügen wird allerdings getestet, ob der Wert für den Primärschlüssel bereits vorhanden ist. Ist der Schlüsselwert vorhanden, so wird der Datensatz nicht kopiert.

Hier könnte gegebenenfalls auch eingestellt werden, dass statt einer Kopie des Datensatzes ein Update des bereits existierenden Datensatzes erfolgen soll. Auf jeden Fall wird so sichergestellt, dass die Zieldatenbank die Datensätze mit den entsprechenden Primärschlüsseln der Quelldatenbank enthält.

Direkter Import von Daten aus Calc

Häufig passiert es, dass Calc statt einer Datenbank zur Eingabe von Daten in eine Tabelle genutzt wird. Solche Daten lassen sich dann über die Zwischenablage oder per Drag-and-Drop in eine Base-Tabelle einlesen. Auch der Export in eine bereits in Base eingebundene *.csv-Textdatei ist möglich.

Soll allerdings Calc auf Dauer zur Dateneingabe genutzt werden und die Daten regelmäßig aus Calc ausgelesen werden, so ist der Kopierschritt vielleicht zu umständlich. Hier setzt das folgende Makro an, das aus einem Formular heraus über einen Button gestartet wird.[45]

Das Makro geht von folgenden Voraussetzungen aus:

1. Die Daten liegen auf dem ersten Tabellenblatt des Calc-Dokuments
2. Auf diesem Tabellenblatt liegen nur die Daten in einer Spalte, nicht zusätzliche Einträge.
3. Die erste Datenzeile enthält die Feldbenennungen, die genau den Feldbezeichnungen in der Base-Tabelle entsprechen.

45 Die Beispieldatenbank «Beispiel_Daten_Import.odb» liegt diesem Handbuch bei.

Die Dateneingabe muss nicht links oben auf dem Tabellenblatt erfolgen. Auch müssen die Felder nicht die gleiche Reihenfolge wie in der Tabelle haben. Spalten, deren Spaltenüberschrift nicht Feldern der Tabelle entspricht, werden ignoriert.

```
001 Sub CalcDataImport(oEvent AS OBJECT, stBaseTab AS STRING, stCalcTab AS STRING)
002    DIM oDatasource AS OBJECT
003    DIM oConnection AS OBJECT
004    DIM oSQL_Command AS OBJECT
005    DIM oResult AS OBJECT
006    DIM oDB AS OBJECT
007    DIM oDoc AS OBJECT
008    DIM oDocView AS OBJECT
009    DIM oRange AS OBJECT
010    DIM Arg()
011    DIM aColumn()
012    DIM aColumns()
013    DIM aType()
014    DIM stSql AS STRING
015    DIM stRow AS STRING
016    DIM stDir AS STRING
017    DIM stColumns AS STRING
018    DIM stStartCol AS STRING
019    DIM stEndCol AS STRING
020    DIM stStartRow AS STRING
021    DIM stEndRow AS STRING
022    DIM stRange AS STRING
023    DIM inCounter AS INTEGER
024    DIM loColumns AS LONG
025    DIM loRows AS LONG
026    DIM loID AS LONG
027    oDatasource = ThisComponent.Parent.CurrentController
028    If NOT (oDatasource.isConnected()) THEN
029       oDatasource.connect()
030    END IF
031    oConnection = oDatasource.ActiveConnection()
032    oSQL_Command = oConnection.createStatement()
```

Nach der Deklaration der Variablen werden Spaltennamen und Spaltentypen aus der vorgegebenen Tabelle der Datenbank ausgelesen. Der Primärschlüssel der Tabelle mit der Bezeichnung "ID" wird als Integer-Feld unabhängig von der Calc-Tabelle erstellt.

```
033    stSql = "SELECT COLUMN_NAME, TYPE_NAME FROM INFORMATION_SCHEMA.SYSTEM_COLUMNS
          WHERE TABLE_NAME = '"+stBaseTab+"' AND NOT COLUMN_NAME = 'ID'"
034    oResult = oSQL_Command.executeQuery(stSql)
035    inCounter = 0
036    stColumns = ""
```

Die Spaltennamen und Spaltentypen werden ausgelesen und in getrennten Arrays gespeichert.

Hinweis

Für Firebird muss der SQL-Code angepasst werden. Vor allem die Zuordnung der Datentypen in SQL-Format ist umständlicher:

```
033  stSql = "SELECT TRIM(""a"".RDB$FIELD_NAME), TRIM(CASE
          ""b"".RDB$FIELD_TYPE||'|'||
          COALESCE(""b"".RDB$FIELD_SUB_TYPE,0) "+ _
          "WHEN '7|0' THEN 'SMALLINT' "+ _
          "WHEN '8|0' THEN 'INTEGER' "+ _
          "WHEN '8|1' THEN 'NUMERIC' "+ _
          "WHEN '8|2' THEN 'DECIMAL' "+ _
          "WHEN '10|0' THEN 'FLOAT' "+ _
          "WHEN '12|0' THEN 'DATE' "+ _
          "WHEN '13|0' THEN 'TIME' "+ _
          "WHEN '14|0' THEN 'CHAR' "+ _
          "WHEN '16|0' THEN 'BIGINT' "+ _
          "WHEN '35|0' THEN 'TIMESTAMP' "+ _
          "WHEN '37|0' THEN 'VARCHAR' "+ _
          "WHEN '261|0' THEN 'BLOB' "+ _
          "WHEN '261|1' THEN 'BLOB Text' "+ _
          "WHEN '261|2' THEN 'BLOB BLR' "+ _
          "WHEN '261|3' THEN 'BLOB ACL' "+ _
          "END) AS ""SQL_Datentyp"" "+ _
          "FROM RDB$RELATION_FIELDS AS ""a"", RDB$FIELDS AS ""b"" "+ _
          "WHERE ""a"".RDB$FIELD_SOURCE = ""b"".RDB$FIELD_NAME "+ _
          "AND ""a"".RDB$RELATION_NAME = '"+stBaseTab+"' "+ _
          "AND ""a"".RDB$FIELD_NAME <> 'ID'"
```

Die Felder müssen außerdem mit **TRIM** eingeschränkt werden, da Firebird die Ausgabe mit fester Zeichenlänge macht und einfach Leerzeichen hinten anfügt. Für die Auswertung im Makro und den Vergleich ist das unpraktikabel.

```
037     WHILE oResult.next
038        ReDim Preserve aColumn(inCounter)
039        ReDim Preserve aType(inCounter)
040        aColumn(inCounter) = oResult.getString(1)
041        aType(inCounter) = oResult.getString(2)
042        inCounter = inCounter+1
043     WEND
```

Der zur Zeit höchste Eintrag für den Primärschlüsselwert wird ermittelt und um 1 erhöht. In diesem Beispiel wird also der Schlüsselwert nicht automatisch von der HSQLDB hoch geschrieben, sondern über das Makro verwaltet. Dies geschieht hier zu Testzwecken, da die Tabelle laufend unter unterschiedlichen Kriterien testweise eingelesen wurde. Entsprechend könnte natürlich auch der Tabellenindex heruntergesetzt werden, wie dies in dem Makro «Tabellenindex_runter» passiert.

```
044     stSql = "SELECT MAX(""ID"") FROM """+stBaseTab+""""
045     oResult = oSQL_Command.executeQuery(stSql)
046     WHILE oResult.next
047        loID = oResult.getInt(1) + 1
048     WEND
```

Der Pfad zur Calc-Datei wird anhand der Lage der Base-Datei im Dateisystem ermittelt. Die Calc-Datei liegt hier im gleichen Verzeichnis wie die Base-Datei.

Anschließend wird die Calc-Datei geladen und gleich unsichtbar geschaltet, damit sie sich nicht in den Vordergrund schiebt.

```
049     oDB = ThisComponent.Parent
050     stDir = Left(oDB.Location,Len(oDB.Location)-Len(ConvertToURL(oDB.Title))+8)
051     stDir = ConvertToUrl(stDir & "Daten_Calc.ods")
052     oDoc = StarDesktop.loadComponentFromURL(stDir,"_blank", 0, Arg() )
053     oDocView = oDoc.CurrentController.Frame.ContainerWindow
054     oDocView.Visible = False
```

Die Position des beschrifteten Bereiches des Tabellenblattes wird ermittelt. Dies geschieht über die **ColumnDescriptions** und **RowDescriptions**. Sie geben genau die Anzahl der beschrifteten Spalten und Zeilen wieder. Außerdem kann darüber die Bezeichnung der Spalte und der Zeile wie z.B. «B2» ausgelesen werden.

```
055     loColumns = uBound(oDoc.Sheets.getByName(stCalcTab).ColumnDescriptions)
           ' Anzahl der Spalten
056     stStartCol = split(oDoc.Sheets.getByName(stCalcTab).ColumnDescriptions(0))(1)
057     stEndCol = split(oDoc.Sheets.getByName(stCalcTab).
           ColumnDescriptions(loColumns))(1)
058     loRows = uBound(oDoc.Sheets.getByName(stCalcTab).RowDescriptions)
           ' Anzahl der Zeilen
059     stStartRow = split(oDoc.Sheets.getByName(stCalcTab).RowDescriptions(0))(1)
060     stEndRow = split(oDoc.Sheets.getByName(stCalcTab).RowDescriptions(loRows))(1)
061     stRange = stStartCol & stStartRow & ":" & stEndCol & StEndRow
```

Der Bereich wird als String zusammengesetzt. Dies erfolgt in der gleichen Art und Weise wie bei der Benennung innerhalb von Calc, also z.B. «A1:C7». Die Daten aus so einem Bereich können mit der Funktion **GetDataArray()** ausgelesen werden.

```
062     oRange = oDoc.Sheets.getByName(stCalcTab).getCellRangeByName(stRange)
063     aDat = oRange.getDataArray()
```

Die Spaltennamen stehen in der ersten Zeile. Sie können eine andere Reihenfolge haben, als dies innerhalb der Tabelle von Base vorgesehen wird. Deshalb werden diese Bezeichnungen für einen späteren Vergleich in einem separaten Array gespeichert. Sie werden in der folgenden Schleife nicht als Werte zum Einlesen in die Tabelle abgefragt. Deshalb beginnt die Schleife mit **i=1**.

```
064     aColumns = aDat(0)
065     FOR i = 1 TO uBound(aDat)
066        aRow = aDat(i)
067        stColumns = """ID"""
068        stRow = loID
069        FOR k = 0 TO loColumns
070           FOR n = 0 TO uBound(aColumn)
```

Im folgenden werden die Spaltenbezeichnungen aus Calc mit denen der Base-Tabelle verglichen. Die Base-Tabelle enthält hier neben Textspalten auch eine Spalte für einen Währungsbetrag, die als **DECIMAL** definiert ist, sowie ein Datum.

Bei dem Währungsbetrag muss das Dezimalkomma gegebenenfalls zu einem Dezimalpunkt umgewandelt werden. Deshalb hier die Funktion **Cdbl**, gekoppelt mit der Funktion **Str**.

Bei dem Datumsfeld gibt Calc den Inhalt als ISO-Zahlencode aus. Dieser Code muss zuerst daraufhin überprüft werden, ob denn überhaupt ein Datum daraus gebildet werden kann. Ist dies möglich, so wird ein SQL-konformes Datum im Format YYYY-MM-DD zusammengestellt, das auch bei einstelligen Tageswerten und Monatswerten nicht versagt.

```
071              IF aColumns(k) = aColumn(n) THEN
072                 IF aType(n) = "DECIMAL" THEN
073                    IF aRow(k) <> "" THEN
074                       aRow(k) = Str(CDbl(aRow(k)))
075                    END IF
076                 END IF
077                 IF aType(n) = "DATE" THEN
078                    IF isDate(CDate(aRow(k))) AND aRow(k) <> "" THEN
079                       aRow(k) = Year(CDate(aRow(k))) & "-"
                             & Right("0" & Month(CDate(aRow(k))), 2) &"-"
                             & Right("0" & Day(CDate(aRow(k))), 2)
080                    ELSE
081                       aRow(k) = ""
082                    END IF
083                 END IF
084                 IF aType(n) = "TIME" THEN
085                    IF isDate(CDate(aRow(k))) AND aRow(k) <> "" THEN
```

```
086                     aRow(k) = Right("0" & Hour(CDate(aRow(k))), 2) & ":"
                           & Right("0" & Minute(CDate(aRow(k))), 2) & ":"
                           & Right("0" & Second(CDate(aRow(k))), 2)
087                  ELSE
088                     aRow(k) = ""
089                  END IF
090               END IF
091               IF aType(n) = "TIMESTAMP"
092                  IF isDate(CDate(aRow(k))) AND aRow(k) <> "" THEN
093                     aRow(k) = Year(CDate(aRow(k))) & "-"
                           & Right("0" & Month(CDate(aRow(k))), 2) & "-"
                           & Right("0" & Day(CDate(aRow(k))), 2) & " "
                           & Right("0" & Hour(CDate(aRow(k))), 2) & ":"
                           & Right("0" & Minute(CDate(aRow(k))), 2) & ":"
                           & Right("0" & Second(CDate(aRow(k))), 2)
094                  ELSE
095                     aRow(k) = ""
096                  END IF
097               END IF
```

Ist der Zellinhalt leer oder für ein Dezimalfeld, Datumsfeld, Zeitfeld oder Timestampfeld gegebenenfalls nicht gültig, so wird an die Base-Tabelle **NULL** weitergegeben. Andernfalls wird der Inhalt in Hochkommata gesetzt. Anschließend werden die Inhalte über Kommata miteinander verbunden. Auch die Bezeichnung der Spalten erfolgt in der Reihenfolge, in der sie aus der Calc-Tabelle ausgelesen wurden.

```
098               IF aRow(k) = "" THEN
099                  aRow(k) = "NULL"
100               ELSE
101                  aRow(k) = "'" & aRow(k) & "'"
102               END IF
103               stRow = stRow & "," & aRow(k)
104               stColumns = stColumns & ",""" & aColumns(k) & """"
105            END IF
106         NEXT
107      NEXT
108      stSql = "INSERT INTO """+stBaseTab+""" ("& stColumns &")
            VALUES ("& stRow &")"
109      oSQL_Command.executeUpdate(stSql)
```

Der Primärschlüsselwert wird hier innerhalb des Makros um 1 heraufgesetzt.

```
110      loID = loID + 1
111   NEXT
112   oDoc.close(True) ' Schließen des Calc-Documentes
113   oEvent.Source.Model.parent.reload()   ' Neuladen des Formulars
114 End Sub
```

Ist der gesamte Inhalt aus dem Calc-Tabellenblatt ausgelesen, so wird das unsichtbare Dokument geschlossen. Anschließend wird das Formular, in dem sich der auslösende Button befindet, neu eingelesen. Dabei wird das Formular über den Button mit **oEvent.Source.Model.parent** ermittelt.

Die Prozedur wird über eine andere Prozedur gestartet. Dadurch kann die Prozedur auch für mehrere Tabellen der Calc-Datei bzw. der Base-Datei genutzt werden:

```
001 SUB Import1_Start(oEvent AS OBJECT)
002    CalcDataImport(oEvent, "tbl_Dez_Dat_NULL", "Tab_Dez_Dat_NULL")
003 END SUB
```

Das auslösende Ereignis wird einfach weiter gereicht. Als zweiter Parameter wird der Tabellenname der Base-Tabelle angegeben. Als dritter Parameter erfolgt die Angabe des Tabellennamens aus der Calc-Datei.

Zugriff auf Abfragen

Abfragen lassen sich in der grafischen Benutzeroberfläche einfacher zusammenstellen als den gesamten Text in Makros zu übertragen, zumal dann auch noch innerhalb des Makros alle Felder- und Tabellenbezeichnungen in zweifach doppelte Anführungszeichen gesetzt werden müssen.

```
001 SUB Abfrageninhalt
002    DIM oDatenDatei AS OBJECT
003    DIM oAbfragen AS OBJECT
004    DIM stQuery AS STRING
005    oDatenDatei = ThisComponent.Parent.CurrentController.DataSource
006    oAbfragen = oDatenDatei.getQueryDefinitions()
007    stQuery = oAbfragen.getByName("Query").Command
008    msgbox stQuery
009 END SUB
```

Aus einem Formular heraus wird auf den Inhalt der *.odb-Datei zugegriffen. Die Abfragen werden über **getQueryDefinitions**() ermittelt. Die SQL-Formulierung der Abfrage "Query" wird über den Zusatz **Command** bereit gestellt. **Command** kann schließlich dazu genutzt werden, eine entsprechende Abfrage auch innerhalb eines Makros weiter zu nutzen.

Allerdings muss bei der Nutzung des SQL-Codes der Abfrage darauf geachtet werden, dass sich der Code nicht wiederum auf eine Abfrage bezieht. Das führt dann unweigerlich zu der Meldung, dass die (angebliche) Tabelle der Datenbank unbekannt ist. Einfacher ist es daher, aus Abfragen Ansichten zu erstellen und entsprechend auf die Ansichten in Makros zuzugreifen.

Soll eine Abfrage in einem Formular weiter genutzt werden, so geht dies über die **COMMAND**-Variable des Formulars:

```
008 oForm.Command = stQuery
009 oForm.CommandType = com.sun.star.sdb.CommandType.COMMAND
010 oForm.reload()
```

Auch die Änderung von Abfragen über ein Makro ist über die Abfragedefinition möglich:

```
007 oAbfrage = oAbfragen.getByName("Query")
008 oAbfrage.Command = "SELECT …"
```

Mit Hilfe solch eine Konstruktion kann z. B. anschließend ein Bericht gestartet werden, der sich auf die Abfrage bezieht. Dadurch kann eine entsprechende Filterung der Daten vorgenommen werden ohne dass eine separate Filtertabelle erstellt werden muss. Dies ist vor allem dann von Vorteil, wenn es sich bei der zugrundeliegenden Datenbank nicht um eine Datenbank handelt, die Abfragen über mehrere Tabellen zulässt. Dies ist beispielsweise bei dBase der Fall. dBase-Tabellen lassen sich nicht in einer Abfrage kombinieren, über Makros dann aber sehr wohl entsprechend filtern, indem der Wert einer Makroabfrage anschließend dem Code der bestehenden Abfrage hinzugefügt wird.

Datenbanksicherungen erstellen

Vor allem beim Erstellen von Datenbanken kann es hin und wieder vorkommen, dass die *.odb-Datei unvermittelt beendet wird. Vor allem beim Berichtsmodul ist ein häufigeres Abspeichern nach dem Editieren sinnvoll.

Ist die Datenbank erst einmal im Betrieb, so kann sie durch Betriebssystemabstürze beschädigt werden, wenn der Absturz gerade während des Schließens der Base-Datei erfolgt. Schließlich wird in diesem Moment der Inhalt der Datenbank in die Datei zurückgeschrieben.

Außerdem gibt es die üblichen Verdächtigen für plötzlich nicht mehr zu öffnende Dateien wie Festplattenfehler usw. Da kann es dann nicht schaden, eine Sicherheitskopie möglichst mit dem aktuellsten Datenstand parat zu haben. Der Datenbestand ändert sich allerdings nicht, während die *.odb-Datei geöffnet ist. Deshalb kann die Sicherung direkt mit dem Öffnen der Datei verbunden werden. Es werden einfach Kopien der Datei in das unter **Extras → Optionen → LibreOffice → Pfade** angegebene Backup-Verzeichnis erstellt. Existiert das Verzeichnis noch

nicht, so wird es erstellt. Das Makro beginnt nach einer voreingestellten Zahl an Kopien (**inMax**) damit, die jeweils älteste Variante zu überschreiben.

Die Prozedur «Datenbankbackup» wird unter **Extras → Anpassen → Ereignisse → Dokument öffnen** eingebunden.

```
001 SUB Datenbankbackup(inMax AS INTEGER)
002    DIM oPath AS OBJECT
003    DIM oDoc AS OBJECT
004    DIM sTitel AS STRING
005    DIM sUrl_Ziel AS STRING
006    DIM sUrl_Start AS STRING
007    DIM i AS INTEGER
008    DIM k AS INTEGER
009    oDoc = ThisComponent
010    sTitel = oDoc.Title
011    sUrl_Start = oDoc.URL
012    DO WHILE sUrl_Start = ""
013       oDoc = oDoc.Parent
014       sTitel = oDoc.Title
015       sUrl_Start = oDoc.URL
016    LOOP
```

Wird das Makro direkt beim Start der *.odb-Datei ausgeführt, so stimmen **sTitel** und **sUrl_Start**. Wird es hingegen von einem Formular aus ausgeführt, so muss erst einmal ermittelt werden, ob überhaupt eine URL verfügbar ist. Ist die URL leer, so wird eine Ebene höher (**oDoc.Parent**) nach einem Wert nachgesehen.

```
017    oPath = createUnoService("com.sun.star.util.PathSettings")
018    MkDir(oPath.Backup)
019    FOR i = 1 TO inMax + 1
020       IF NOT FileExists(oPath.Backup & "/" & i & "_" & sTitel) THEN
021          IF i > inMax THEN
022             FOR k = inMax - 1 TO 1 STEP -1
023                IF FileDateTime(oPath.Backup & "/" & k & "_" & sTitel) <=
024                   FileDateTime(oPath.Backup & "/" & k+1 & "_" & sTitel) THEN
025                   IF k = 1 THEN
026                      i = k
027                      EXIT FOR
028                   END IF
029                ELSE
030                   i = k + 1
031                   EXIT FOR
032                END IF
033             NEXT
034          END IF
035          EXIT FOR
036       END IF
037    NEXT
038    sUrl_Ziel = oPath.Backup & "/" & i & "_" & sTitel
039    FileCopy(sUrl_Start,sUrl_Ziel)
040 END SUB
```

Werden vor der Ausführung der Prozedur «Datenbankbackup» während der Nutzung von Base die Daten aus dem Cache in die Datei zurückgeschrieben, so kann ein entsprechendes Backup auch z.B. nach einer bestimmten Nutzerzeit oder durch Betätigung eines Buttons sinnvoll sein. Das Zurückschreiben regelt die folgende Prozedur:

```
001 SUB Daten_aus_Cache_schreiben
002    DIM oDaten AS OBJECT
003    DIM oDataSource AS OBJECT
004    oDaten = ThisDatabaseDocument.CurrentController
005    IF NOT ( oDaten.isConnected() ) THEN oDaten.connect()
006    oDataSource = oDaten.DataSource
007    oDataSource.flush
008 END SUB
```

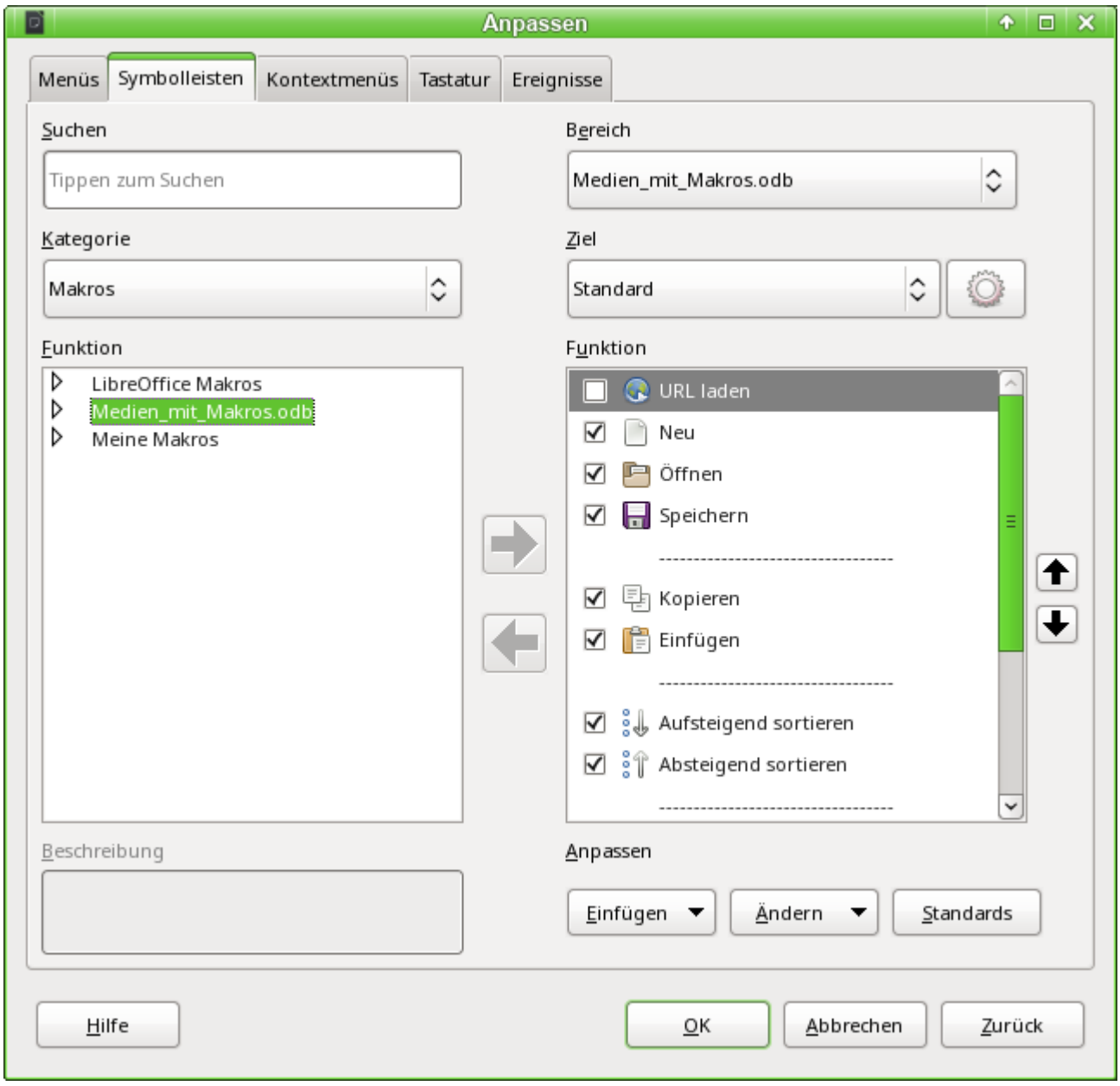
Hinweis

Für Firebird lässt sich dieser Code gut nutzen um beim Schließen der Datenbankdatei die Daten automatisch zu schreiben. Sonst erscheint nach Datenänderungen nämlich die Aufforderung zum Speichern, da das im Gegensatz zu internen Hsqldb nicht automatisch abläuft.
Die Prozedur «Daten_aus_Cache_schreiben» wird unter **Extras → Anpassen → Ereignisse → Ansicht wird geschlossen** eingebunden. Dann erfolgt die Datenspeicherung vor der Abfrage zur Speicherung der Datei.

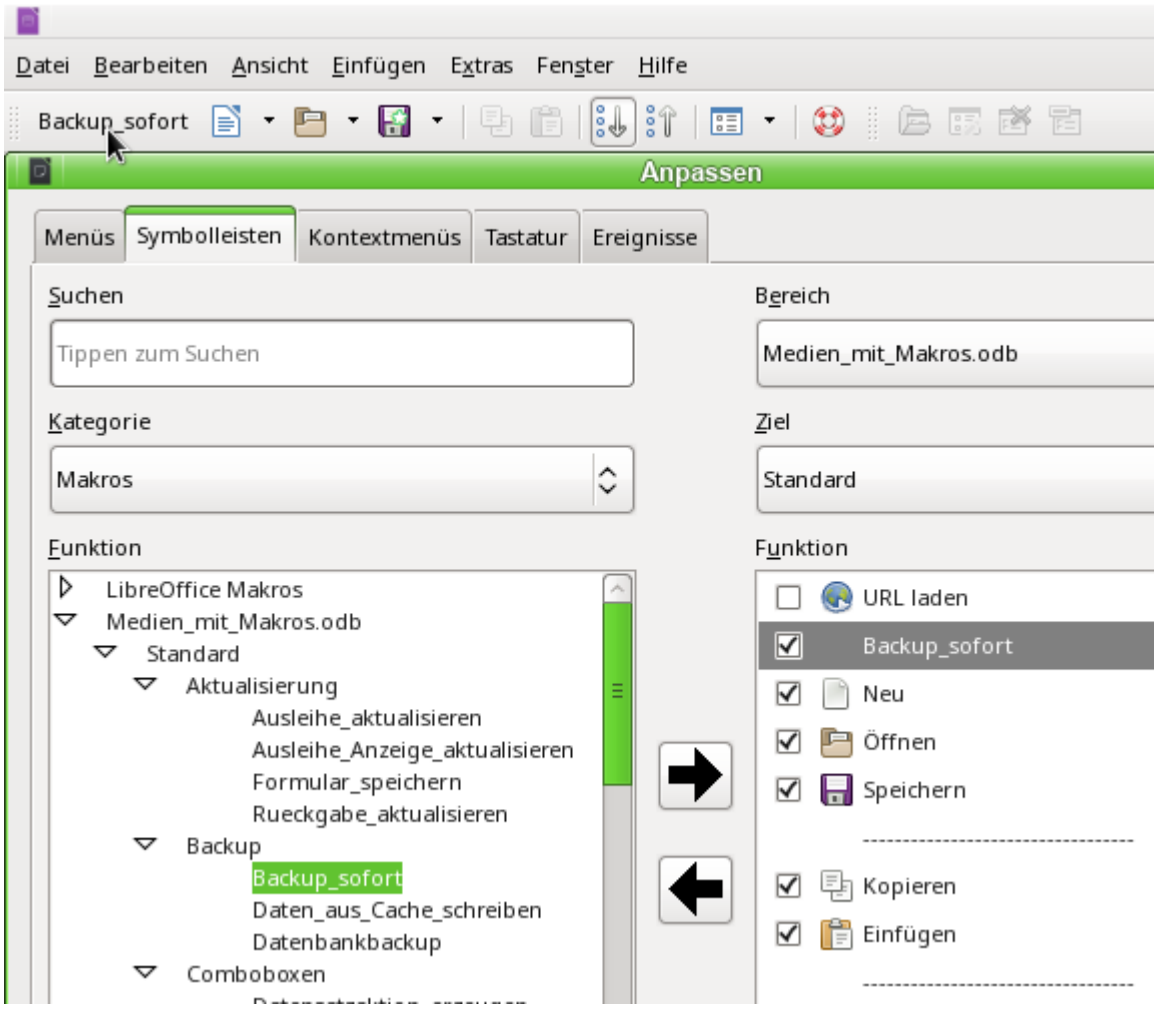
Soll alles zusammen aus einem Formular heraus über einen Button gestartet werden, so müssen beide Prozeduren über eine weitere Prozedur angesprochen werden:

```
001 SUB Backup_sofort
002    Daten_aus_Cache_schreiben
003    Datenbankbackup(10)
004 END SUB
```

Gerade bei einem Sicherungsmakro ist es vielleicht sinnvoll, das Makro über die Symbolleiste der Datenbank erreichbar zu machen. Dies geschieht im Hauptfenster der Base-Datei unter **Extras → Anpassen → Symbolleisten**. Dort wird als **Bereich** die aktuelle Datenbankdatei, als **Kategorie** «Makros» und als **Ziel** die für alle Bereiche zuständige Symbolleiste «Standard» ausgesucht.



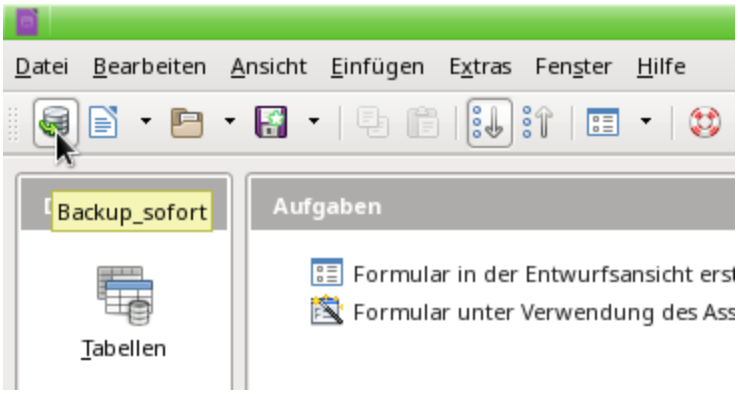
Bei den Makros sind die verfügbaren allgemeinen Makros sowie die Makros aus der Datenbankdatei auswählbar. Aus den Datenbankmakros wird die Prozedur «Backup_Sofort» ausgesucht.



Der Befehl ist jetzt in der Symbolleiste an erster Stelle verfügbar. Um jetzt die Prozedur auszuführen genügt ein Auslösen des Buttons in der Symbolleiste.

Jetzt bietet es sich noch an, dem Befehl ein Symbol zuzuweisen. Über **Ändern → Symbol austauschen** wird der folgende Dialog geöffnet.

Hier wird jetzt ein passendes Symbol gesucht. Es kann auch ein eigenes Symbol erstellt und eingebunden werden.



Das Symbol erscheint anschließend statt der Benennung der Prozedur. Die Benennung wird als Tooltip angezeigt.

Interne Datenbanken sicher schließen

Die internen Datenbanken müssen vor dem Schließen erst einmal den Inhalt aus dem Speicher zurück in den Datenbankcontainer, die *.odb-Datei, schreiben. Ein einfaches **oDoc.close(True)** führt hier bei dem Datenbankdokument dazu, dass es zwar geschlossen wird, die Änderungen seit dem letzten Speichern nicht mehr gesichert sind. Hierzu muss über **oDoc.Datasource.flush** der Speicherinhalt in die interne Datenbank geschrieben und dann die Speicherung vollzogen werden. Die obige Prozedur Daten_aus_Cache_schreiben muss also für das Schließen zum Schluss nach dem Schreiben aus dem Cache nur um den Eintrag **ThisDatabaseDocument.close(True)** erweitert werden.

Tabellenindex heruntersetzen bei Autowert-Feldern

Werden viele Daten aus Tabellen gelöscht, so stören sich Nutzer häufig daran, dass die automatisch erstellten Primärschlüssel einfach weiter hochgezählt werden, statt direkt an den bisher höchsten Schlüsselwert anzuschließen. Die folgende Prozedur liest für eine Tabelle den bisherigen Höchstwert des Feldes "ID" aus und stellt den nächsten Schlüsselstartwert um 1 höher als das Maximum ein.

Heißt das Primärschlüsselfeld nicht "ID", so müsste das Makro entsprechend angepasst werden.

```
001  SUB Tabellenindex_runter(stTabelle AS STRING)
002     REM Mit dieser Prozedur wird das automatisch hochgeschriebene
           Primärschlüsselfeld mit der vorgegebenen Bezeichnung "ID" auf den niedrigst
           möglichen Wert eingestellt.
003     DIM stAnzahl AS STRING
004     DIM inAnzahl AS INTEGER
005     DIM inSequence_Value AS INTEGER
006     oDatenquelle = ThisComponent.Parent.CurrentController
           ' Zugriffsmöglichkeit aus dem Formular heraus
007     IF NOT (oDatenquelle.isConnected()) THEN
008        oDatenquelle.connect()
009     END IF
010     oVerbindung = oDatenquelle.ActiveConnection()
011     oSQL_Anweisung = oVerbindung.createStatement()
012     stSql = "SELECT MAX(""ID"") FROM """+stTabelle+""""
           ' Der höchste in "ID" eingetragene Wert wird ermittelt
013     oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)    ' Abfrage starten und
           den Rückgabewert in einer Variablen oAbfrageergebnis speichern
014     WHILE oAbfrageergebnis.next
015        stAnzahl = oAbfrageergebnis.getString(1) ' Erstes Datenfeld wird ausgelesen
```

```
016     WEND   ' nächster Datensatz, in diesem Fall nicht mehr erforderlich, da nur ein
           Datensatz existiert
017     IF stAnzahl = "" THEN   ' Falls der höchste Wert gar kein Wert ist, also die
           Tabelle leer ist wird der höchste Wert als -1 angenommen
018        inAnzahl = -1
019     ELSE
020        inAnzahl = cInt(stAnzahl)
021     END IF
022     inSequence_Value = inAnzahl+1  ' Der höchste Wert wird um 1 erhöht
023     REM Ein neuer Befehl an die Datenbank wird vorbereitet. Die ID wird als neu
           startend ab inAnzahl+1 deklariert.
024     REM Diese Anweisung hat keinen Rückgabewert, da ja kein Datensatz ausgelesen
           werden muss
025     oSQL_Anweisung1 = oVerbindung.createStatement()
026     oSQL_Anweisung1.executeQuery("ALTER TABLE """ + stTabelle + """
027        ALTER COLUMN ""ID"" RESTART WITH " + inSequence_Value + "")
028  END SUB
```

Drucken aus Base heraus

Der Standard, um aus Base heraus ein druckbares Dokument zu erzeugen, ist die Nutzung eines Berichtes. Daneben gibt es noch Möglichkeiten, Tabellen und Abfragen einfach nach Calc zu kopieren und dort zum Druck aufzubereiten. Auch der direkte Druck eines Formularinhaltes vom Bildschirm ist natürlich möglich.

Druck von Berichten aus einem internen Formular heraus

Um einen Bericht zu starten, muss normalerweise die Benutzeroberfläche von Base aufgesucht werden. Ein Mausklick auf den Berichtsnamen startet dann die Ausführung des Berichtes. Einfacher geht es natürlich, den Bericht direkt aus dem Formular heraus zu starten:

```
001  SUB Berichtsstart
002     ThisDatabaseDocument.ReportDocuments.getByName("Bericht").open
003  END SUB
```

Sämtliche Berichte werden über **ReportDocuments** mit ihrem Namen angesprochen. Mit **open** werden sie geöffnet. Wird ein Bericht jetzt an eine Abfrage gebunden, die über das Formular gefiltert wird, so kann auf diese Art und Weise der zum aktuellen Datensatz anfallende Ausdruck erfolgen.

Start, Formatierung, direkter Druck und Schließen des Berichts

Noch schöner ist es, wenn der Bericht direkt an den Drucker geschickt wird. Die folgende Kombination an Prozeduren legt sogar noch ein paar kleine Features zu. Sie selektiert zuerst den aktiven Datensatz des Formulars, formatiert danach den Bericht um, indem die Felder für den Text auf automatische Höhe eingestellt werden, um anschließend den Bericht zu starten. Schließlich wird der Bericht auch noch gedruckt und ggf. noch als *.pdf-Dokument abgespeichert. Und all das passiert nahezu vollständig im Hintergrund, da der Bericht direkt nach dem Öffnen auf unsichtbar geschaltet und nach dem Ausdruck wieder geschlossen wird. Anregungen für die verschiedenen Prozeduren stammen hier von Andrew Piontak, Thomas Krumbein und Lionel Elie Mamane.

```
001  SUB BerichtStart(oEvent AS OBJECT)
002     DIM oForm AS OBJECT
003     DIM stSql AS STRING
004     DIM oDatenquelle AS OBJECT
005     DIM oVerbindung AS OBJECT
006     DIM oSQL_Anweisung AS OBJECT
007     DIM oReport AS OBJECT
008     DIM oReportView AS OBJECT
009     oForm = oEvent.Source.model.parent
010     stSql = "UPDATE ""Filter"" SET ""Integer"" = '" +
           oForm.getInt(oForm.findColumn("ID")) + "' WHERE ""ID"" = TRUE"
```

```
011     oDatenquelle = ThisComponent.Parent.CurrentController
012     If NOT (oDatenquelle.isConnected()) THEN
013        oDatenquelle.connect()
014     END IF
015     oVerbindung = oDatenquelle.ActiveConnection()
016     oSQL_Anweisung = oVerbindung.createStatement()
017     oSQL_Anweisung.executeUpdate(stSql)
018     oReport = ThisDatabaseDocument.ReportDocuments.getByName("Berichtsname").open
019     oReportView = oReport.CurrentController.Frame.ContainerWindow
020     oReportView.Visible = False
021     BerichtZeilenhoeheAuto(oReport)
022  END SUB
```

Die Prozedur «BerichtStart» wird mit einem Button innerhalb eines Formulars verknüpft. Es kann dabei über den Button der Primärschlüssel des aktuellen Datensatzes des Formulars ausgelesen werden. Dies geschieht hier über das auslösende Ereignis, von dem heraus auf das Formular **oForm** geschlossen wird. Der Name des Schlüsselfeldes ist hier mit **"ID"** angegeben. Über **oForm.getInt(oForm.findColumn("ID"))** wird aus dem Feld der Schlüsselwert als Integer-Wert gelesen. Dieser Wert wird anschließend in einer Filtertabelle abgespeichert. Diese Filtertabelle steuert über eine Abfrage, dass nur der aktuelle Datensatz des Formulars für den Bericht verwendet wird.

Ohne Bezug auf das Formular könnte auch nur der Bericht aufgerufen werden. Dabei ist der aufgerufene Bericht gleich als Objekt ansprechbar (**oReport**). Anschließend wird das Fenster auf unsichtbar eingestellt. Dies geht leider nicht direkt mit dem Aufruf, so dass ganz kurz das Fenster erscheint, dann aber ggf. in Ruhe im Hintergrund mit dem entsprechenden Inhalt gefüllt wird.

Anschließend wird die Prozedur «BerichtZeilenhoeheAuto» gestartet. Dieser Prozedur wird der Hinweis auf den geöffneten Bericht mitgegeben.

Hinweis

Diese Prozedur ist ab LO 6.4 nicht mehr notwendig. Seitdem ist im Report-Designer für aufgezogene Felder eine automatische Größeneinstellung möglich. Aber Achtung: Das funktioniert nur ab LO 6.4 und neuer. Die Höhe wird bei älteren Versionen nicht automatisch eingestellt.

Die Zeilenhöhe kann beim Berichtsentwurf bis LO 6.4 nicht automatisch angepasst werden. Ist zu viel Text für ein Feld vorgesehen, so wird der Text abgeschnitten und darauf mit Hilfe eines roten Dreiecks hingewiesen. Solange dies nicht funktioniert, stellt die folgende Prozedur sicher, dass z.B. in allen Tabellen mit der Bezeichnung «Detail» die automatische Höhe eingeschaltet wird.

```
001  SUB BerichtZeilenhoeheAuto(oReport AS OBJECT)
002     DIM oTables AS OBJECT
003     DIM oTable AS OBJECT
004     DIM inT AS INTEGER
005     DIM inI AS INTEGER
006     DIM oRows AS OBJECT
007     DIM oRow AS OBJECT
008     oTables = oReport.getTextTables()
009     FOR inT = 0 TO oTables.count() - 1
010        oTable = oTables.getByIndex(inT)
011        IF Left$(oTable.name, 6) = "Detail" THEN
012           oRows = oTable.Rows
013           FOR inI = 0 TO oRows.count - 1
014              oRow = oRows.getByIndex(inI)
015              oRow.IsAutoHeight = True
016           NEXT inI
017        ENDIF
018     NEXT inT
```

```
019     BerichtDruckenUndSchliessen(oReport)
020 END SUB
```

Bei dem Entwurf des Berichtes muss darauf geachtet werden, dass alle in einer Zeile des Bereiches «Detail» befindlichen Felder tatsächlich die gleiche Höhe haben. Sonst kann es zusammen mit der Automatik passieren, dass ein Feld plötzlich auf die doppelte Zeilenhöhe gesetzt wird.

Nachdem in allen Tabellen mit der Bezeichnung «Detail» die automatische Höhe eingestellt wurde, wird anschließend der Bericht über die Prozedur «BerichtDruckenUndSchliessen» weiter an den Drucker geschickt.

Das Array Props enthält die verschiedenen Werte, die mit dem Drucker bei einem Dokument verbunden sind. Für den Druckbefehl ist hier lediglich der Name des Standarddruckers wichtig. Das Berichtsdokument soll so lange geöffnet bleiben, bis der Druck tatsächlich abgeschlossen ist. Dies geschieht, indem dem Druckbefehl der Name und der Befehl «Warte, bis ich fertig bin» (**Wait**) mitgegeben wird.

```
001 Sub BerichtDruckenUndSchliessen(oReport AS OBJECT)
002     DIM Props
003     DIM stDrucker AS STRING
004     Props = oReport.getPrinter()
005     stDrucker = Props(0).value
006     DIM arg(1) AS NEW com.sun.star.beans.PropertyValue
007     arg(0).name = "Name"
008     arg(0).value = "<" & stDrucker & ">"
009     arg(1).name = "Wait"
010     arg(1).value = True
011     oReport.print(arg())
012     oReport.close(true)
013 End Sub
```

Erst wenn der Druck komplett an den Drucker abgeschickt wurde, wird das Dokument geschlossen.

Zu Einstellungen des Druckers siehe *Drucker und Druckeinstellungen* aus der LibreOffice API.

Soll statt des Drucks oder zusätzlich zu dem Druck auch eine *.pdf-Datei des Dokumentes als Sicherungskopie abgelegt werden, so wird darauf mit der Methode **storeToURL()** zugegriffen:

```
001 Sub BerichtAlsPDFspeichern(oReport AS OBJECT)
002     DIM stUrl AS STRING
003     DIM arg(0) AS NEW com.sun.star.beans.PropertyValue
004     arg(0).name = "FilterName"
005     arg(0).value = "writer_pdf_Export"
006     stUrl = "file:///...."
007     oReport.storeToURL(stUrl, arg())
008 End Sub
```

Bei der URL muss natürlich eine komplette URL-Adresse angegeben werden. Noch sinnvoller ist es, diese Adresse z.B. gekoppelt mit einem unverwechselbaren Merkmal des gedruckten Dokumentes zu versehen, wie z.B. der Rechnungsnummer. Sonst könnte es passieren, dass eine Sicherungsdatei beim nächsten Druck einfach überschrieben wird.

Druck von Berichten aus einem externen Formular heraus

Schwierig wird es, wenn mit externen Formularen gearbeitet wird. Die Berichte liegen dann in der *.odb-Datei und sind auch über den Datenquellenbrowser erst einmal nicht verfügbar.

```
001 SUB Berichtsstart(oEvent AS OBJECT)
002     DIM oFeld AS OBJECT
003     DIM oForm AS OBJECT
004     DIM oDocument AS OBJECT
005     DIM oDocView AS OBJECT
006     DIM Arg()
007     oFeld = oEvent.Source.Model
008     oForm = oFeld.Parent
009     sURL = oForm.DataSourceName
```

```
010     oDocument = StarDesktop.loadComponentFromURL(sURL, "_blank", 0, Arg() )
011     oDocView = oDocument.CurrentController.Frame.ContainerWindow
012     oDocView.Visible = False
013     oDocument.getCurrentController().connect
014     Wait(100)
015     oDocument.ReportDocuments.getByName("Bericht").open
016     oDocument.close(True)
017 END SUB
```

Der Bericht wird von einem Button des externen Formulars gestartet. Über den Button wird das Formular ermittelt, in dem der Pfad zur *.odb-Datei verzeichnet ist: **oForm.DataSourceName**. Anschließend wird mit **loadComponentFromUrl** die *.odb-Datei geöffnet. Die Datei soll nur im Hintergrund liegen. Deshalb wird gleich auf die Ansicht zugegriffen und die Oberfläche der *.odb-Datei auf **Visible = False** gestellt. Dies sollte auch direkt beim Aufruf über die Argumentenliste **Arg()** funktionieren, brachte aber bei Tests nicht den gewünschten Erfolg.

Wird jetzt direkt der Bericht des geöffneten Dokumentes aufgerufen, so ist die Datenbankverbindung noch nicht verfügbar. Der Bericht erscheint nur mit einem grauen Hintergrund und LibreOffice verzeichnet einen Absturz. Schon eine kleine Wartezeit von 100 Millisekunden (**Wait(100)**) löst dieses Problem. Hier müssen praktische Tests zeigen, wie kurz diese Zeit eingestellt werden kann. Anschließend wird der Bericht gestartet. Da es sich bei dem ausgeführten Bericht um eine separate Textdatei handelt, kann die geöffnete *.odb-Datei anschließend geschlossen werden. Mit **oDocument.close(True)** wird der Befehl an die *.odb-Datei weiter gegeben. Die Datei wird allerdings erst dann geschlossen, wenn sie nicht mehr aktiv z.B. Daten an die Berichtsdatei weiter geben muss.

Mit einem entsprechenden Zugriff können auch die Formulare innerhalb der *.odb-Datei gestartet werden. Hier sollte dann aber das Schließen des Dokumentes unterbleiben.

Deutlich schneller als mit dem Report-Designer und trotzdem gut gestaltet geht der Druck aber über Makros mit Hilfe von Serienbrieffunktionen oder Textfeldern.

Serienbriefdruck aus Base heraus

Manchmal reicht einfach ein Bericht nicht aus, um sauber Briefe an die Adressaten zu erstellen. Schon die Textfelder in einem Bericht sind hier in der Nutzung doch sehr eingeschränkt. Hierzu wird dann ein Serienbrief im Writer erstellt. Es muss aber nicht sein, dass erst der Writer geöffnet wird, dort dann über den Serienbriefdruck alle Auswahlmöglichkeiten und Eingaben gemacht werden und schließlich der Druck erfolgt. Dies alles geht auch per Makro direkt aus Base heraus.

```
001 SUB Serienbriefdruck
002     DIM oMailMerge AS OBJECT
003     DIM aProps()
004     oMailMerge = createunoservice("com.sun.star.text.MailMerge")
```

Als Name der Datenquelle wird der Name angegeben, unter dem die Datenbank in LO angemeldet ist. Dieser Name muss nicht identisch mit dem Dateinamen sein. Der Anmeldename in diesem Beispiel lautet "Adressen"

```
005     oMailMerge.DataSourceName = "Adressen"
```

Die Pfadbeschreibung mit der Serienbriefdatei erfolgt in der Art der jeweiligen Betriebssystemumgebung, hier ab dem Wurzelpfad eines Linux-Systems.

```
006     oMailMerge.DocumentURL = ConvertToUrl("home/user/Dokumente/Serienbrief.odt")
```

Der Typ des Kommandos wird festgelegt. '0' steht für eine Tabelle, '1' für eine Abfrage und '2' für ein direktes SQL-Kommando.

```
007     oMailMerge.CommandType = 1
```

Hier wurde eine Abfrage gewählt, die den Namen "Serienbriefabfrage" trägt.

```
008     oMailMerge.Command = "Serienbriefabfrage"
```

Über den Filter wird festgelegt, für welche Datensätze aus der Serienbriefabfrage ein Druck erfolgen soll. Dieser Filter könnte z.B. über ein Formularfeld aus Base heraus an das Makro wei-

tergegeben werden. Mit dem Primärschlüssel eines Datensatzes könnte so der Ausdruck eines einzelnen Dokumentes erfolgen.

In diesem Beispiel wird aus der "Serienbriefabfrage" das Feld "Geschlecht" aufgesucht und dort nach Datensätzen gesucht, die in diesem Feld mit einem 'm' versehen sind.

```
009     oMailMerge.Filter = """Geschlecht""='m'"
```

Es gibt die Ausgabetypen Drucker (1), Datei (2) und Mail (3). Hier wurde zu Testzwecken die Ausgabe in eine Datei gewählt. Diese Datei wird in dem angegebenen Pfad abgespeichert. Für jeden Serienbriefdatensatz wird ein Druck erzeugt. Damit dieser Druck unterscheidbar ist, wird das Feld "Nachname" in den Dateinamen aufgenommen.

```
010     oMailMerge.OutputType = 2
011     oMailMerge.OutputUrl = ConvertToUrl("home/user/Dokumente")
012     oMailMerge.FileNameFromColumn = True
013     oMailMerge.Filenameprefix = "Nachname"
014     oMailMerge.execute(aProps())
015 END SUB
```

Wird allein der Filter über ein Formular bestückt, so kann auf diese Art und Weise, also ohne die Öffnung des Writer-Dokuments, ein Serienbriefdruck erfolgen. Mit der entsprechenden Eingabe von anderen Parametern ist es auch möglich, direkt eine Mail mit persönlichem Anhang zu erstellen. Siehe zu den Parametern *https://api.libreoffice.org/docs/idl/ref/servicecom_1_1sun_1_1star_1_1text_1_1MailMerge.html*.

Drucken über Textfelder

Über **Einfügen → Feldbefehl → Funktionen → Platzhalter** wird im Writer eine Vorlage für das zukünftig zu druckende Dokument erstellt. Die Platzhalter sollten dabei sinnvollerweise mit dem Namen versehen werden, den die Felder auch in der Datenbank bzw. der Tabelle/Abfrage für das Formular haben, aus dem heraus das Makro aufgerufen wird.

Für einfache Zwecke wird «Text» als Typ für den Platzhalter gewählt.

In dem Makro wird der Pfad zur Vorlage hinterlegt. Es wird ein neues Dokument «Unbenannt1.odt» erstellt. Vom Makro werden die Platzhalter über die Abfrage des Inhaltes des aktuellen Datensatzes des Formulars befüllt. Das offene Dokument kann nun noch nach Belieben verändert werden.

In der Beispieldatenbank «Beispiel_Datenbank_Serienbrief_direkt.odb» wird gezeigt, wie mit Hilfe von Textfeldern und einem Zugriff auf eine in der Vorlage bereits vorgesehenen Tabelle eine komplette Rechnung erstellt werden kann. Im Gegensatz zum Report-Designer sind bei dieser Form der Rechnungserstellung die entsprechenden Felder für den Tabelleninhalt nicht in der Höhe begrenzt. Deshalb wird immer aller Text angezeigt.

Hier Teile des Codes, der im Wesentlichen diesem Beitrag von DPunch zu verdanken ist: *http://de.openoffice.info/viewtopic.php?f=8&t=45868#p194799*

```
001 SUB Textfelder_Fuellen
002    oForm = thisComponent.Drawpage.Forms.MainForm
003    IF oForm.RowCount = 0 THEN
004       msgbox "Kein Datensatz zum Drucken vorhanden"
005       EXIT SUB
006    END IF
```

Das Hauptformular wird angesteuert. Hier könnte auch die Lage des auslösenden Buttons das Formular selbst ermitteln. Anschließend wird geklärt, ob in dem Formular überhaupt Daten liegen, die einen Druck ermöglichen.

```
007    oColumns = oForm.Columns
008    oDB = ThisComponent.Parent
```

Der Zugriff auf die URL ist nicht vom Formular aus direkt möglich. Es muss auf den darüber liegenden Frame der Datenbank Bezug genommen werden.

```
009    stDir = Left(oDB.Location,Len(oDB.Location)-Len(ConvertToURL(oDB.Title))+8)
```

Der Titel der Datenbank wird von der URL abgetrennt.

```
010     stDir = stDir & "Beispiel_Textfelder.ott"
```

Die Vorlage wird aufgesucht und geöffnet

```
011     DIM args(0) AS NEW com.sun.star.beans.PropertyValue
012     args(0).Name = "AsTemplate"
013     args(0).Value = True
014     oNewDoc = StarDesktop.loadComponentFromURL(stDir,"_blank",0,args)
```

Die Textfelder werden eingelesen

```
015     oTextfields = oNewDoc.Textfields.createEnumeration
016     DO WHILE oTextfields.hasMoreElements
017        oTextfield = oTextfields.nextElement
018        IF oTextfield.supportsService("com.sun.star.text.TextField.JumpEdit") THEN
019           stColumnname = oTextfield.PlaceHolder
```

Placeholder ist die Benennung für das Textfeld

```
020           IF oColumns.hasByName(stColumnname) THEN
```

Wenn der Name des Textfeldes gleich dem Spaltennamen der Daten ist, die dem Formular zugrunde liegen, wird der Inhalt aus der Datenbank auf das Feld in dem Textdokument übertragen.

```
021              inIndex = oForm.findColumn(stColumnname)
022              oTextfield.Anchor.String = oForm.getString(inIndex)
023           END IF
024        END IF
025     LOOP
026  END SUB
```

Drucken über Tabellen in Writer

In Tabellenansichten zu drucken liegt bei den vielen Tabellen in einem Datenbankdokument ja nahe. Eine solche Konstruktion über den Report-Designer zu bewerkstelligen ist allerdings eine echte Geduldsprobe. Schon die Positionierung der Felder zusammen mit den Tabellenköpfen zu einem einheitlichen Bild ist wegen der internen Maße in Punkt statt in Zentimeter ein Problem. Eine Umrandungsfunktion für die Zellen gibt es nicht, eine senkrechte oder waagerechte Linie gibt es nur in der Farbe Schwarz als Haarlinie. Und ist so ein Dokument schließlich erstellt, dann dauert der Bericht deutlich länger als mit einer vorgefertigten Writer-Tabelle, weil ein Bericht aus vielen kleinen Tabellen zusammengesetzt wird. Der Druck über Tabellen im Writer[46] wird im Folgenden aufeinander aufbauend an Beispielen erklärt.

Einfacher Druck von Text in Tabellen

Adressen

ID	Vorname	Nachname	Straße	Nr	PLZ	Ort
1	Erika	Schlank	Winkelstr.	3b	34567	Obersumpfingen
2	Marleen	Water	Am Deich	17	28671	Allessiel
3	Jürgen	Gibtsnich	Zur Alm	7	87654	Waldübertal

Für den Druck wird eine Vorlage erstellt, die lediglich aus der Überschrift und einer leeren Tabelle besteht. Die Tabelle besteht in der Vorlage aus der Titelzeile und einer weiteren Zeile für den Inhalt. Die entsprechende Formatierung wird in der Vorlage vorgenommen. Der Tabelle

46 Die Beispieldatenbank «Beispiel_Druck_Writer_Tabellen.odb» liegt den Beispieldatenbanken bei. Jede Druckform ist dort in einem separaten Modul unter gebracht und kommentiert. Notwendige Vorlagen liegen ebenfalls bei. Selbst die Erstellung eines Drucks ohne Vorlage ist in der Beispieldatenbank enthalten, hier aber nicht weiter erklärt. Damit können dann nacheinander mehrere Tabellen erstellt werden. Eine Gruppierung wie im Report-Designer ist möglich.

wird dabei über **Tabelleneigenschaften → Tabelle → Eigenschaften → Name** ein fester Name zugewiesen. Über diesen festen Namen wird die Tabelle im Makro gefunden und mit Inhalt gefüllt.

```
001 SUB FillTable
002    DIM oDB AS OBJECT, oNewDoc AS OBJECT, oTables AS OBJECT, oTable AS OBJECT
003    DIM oRows AS OBJECT, oDatasource AS OBJECT, oConnection AS OBJECT
004    DIM oSQL_Statement AS OBJECT, oResult AS OBJECT
005    DIM stDir AS STRING, stSql AS STRING, stText AS STRING
006    DIM i AS INTEGER, k AS INTEGER, inCols AS INTEGER
```

Der Zugriff auf die Datenbankdatei erfolgt vom Formular aus. Die *.odb-Datei ist Parent des Formulardokumentes. Anschließend wird daraus der Pfad ermittelt, in dem die Vorlagendatei liegt. In diesem Beispiel liegt die Vorlage im gleichen Pfad wie Datenbankdatei.

```
007    oDB = ThisComponent.Parent
008    stDir = Left(oDB.Location,Len(oDB.Location)-Len(ConvertToURL(oDB.Title))+8)
```

Der Titel der Datenbank wird von der URL abgetrennt. Die Längenermittlung mit **ConvertToURL** ist notwendig, falls der Titel Leerzeichen enthält.

```
009    stDir = stDir & "PrintStart.ott"
```

Die Vorlage wird geöffnet und das Writer-Dokument damit erstellt.

```
010    DIM args(0) AS NEW com.sun.star.beans.PropertyValue
011    args(0).Name = "AsTemplate"
012    args(0).Value = True
013    oNewDoc = StarDesktop.loadComponentFromURL(stDir,"_blank",0,args)
```

Die Vorlage enthält eine Tabelle, die den Namen «Printout_Addresses» hat. Diese Tabelle besteht aus 2 Zeilen - der Titelzeile und der ersten Zeile für den Inhalt

```
014    oTables = oNewDoc.getTextTables
015    oTable = oTables.getByName("Printout_Addresses")
016    inCols = oTable.Columns.Count
```

Mit der Variablen **i** wird die Inhaltszeile angesteuert. Die Zeilenzählung bei Tabellen beginnt mit der Zeile 0.

```
017    i = 1
018    oDatasource = ThisComponent.Parent.CurrentController
019    If NOT (oDatasource.isConnected()) THEN
020       oDatasource.connect()
021    END IF
022    oConnection = oDatasource.ActiveConnection()
023    oSQL_Statement = oConnection.createStatement()
024    stSql = "SELECT * FROM ""tbl_Adressen"""
025    oResult = oSQL_Statement.executeQuery(stSql)
```

Die Daten aus "tbl_Adressen" werden ausgelesen. Anschließend werden die Inhalte direkt in die entsprechenden Zellen der Tabelle als Text eingefügt.

```
026    WHILE oResult.next
027       FOR k = 0 TO inCols-1
028          stText = oResult.getString(k+1)
029          oTable.getCellByPosition(k,i).setString(stText)
030       NEXT
```

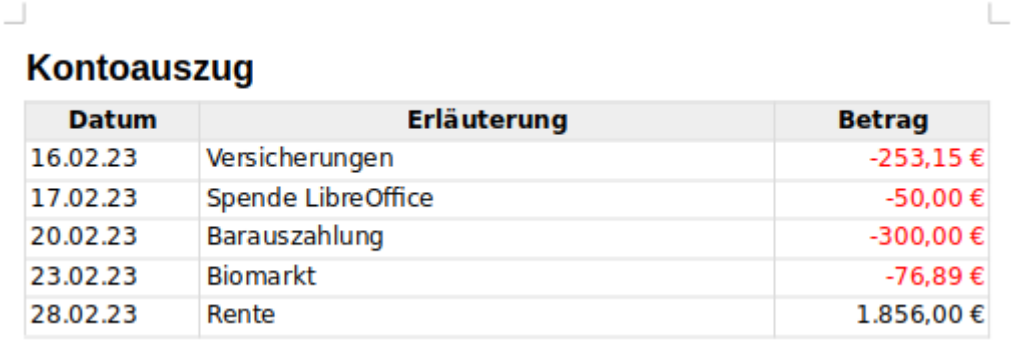
Für jeden neuen Datensatz muss eine zusätzliche Inhaltszeile erstellt werden.

```
031       oRows = oTable.getRows()
032       oRows.insertByIndex(oRows.getCount(),1)
033       i = i + 1
034    WEND
```

Durch die Schleife bleibt zum Schluss eine leere Inhaltszeile übrig. Diese Zeile könnte weiter genutzt werden (z.B. für eine Summierung bei Rechnungen). Hier ist die Zeile nicht nötig und wird deswegen entfernt.

```
035    oRows.removeByIndex(oRows.getCount()-1,1)
036 END SUB
```

Tabellendruck mit formatierten Zellen



Kontoauszug

Datum	Erläuterung	Betrag
16.02.23	Versicherungen	-253,15 €
17.02.23	Spende LibreOffice	-50,00 €
20.02.23	Barauszahlung	-300,00 €
23.02.23	Biomarkt	-76,89 €
28.02.23	Rente	1.856,00 €

Bereits mit einer kleinen Erweiterung lässt sich das entsprechende Format für die Spalten erstellen. Hierzu wird in der Vorlage die Spalte für das Datum (Zelle A2) als Datum formatiert, die Spalte für den Betrag (Zelle C2) als Währungsfeld formatiert. Außerdem wird noch die Spalte für den Betrag rechtsbündig gesetzt.

Alle weiteren durch das Makro hinzugefügten Zeilen übernehmen die Formatierung aus der vorhergehenden Zeile. Der Makro-Code wird um einige Zeilen ergänzt.

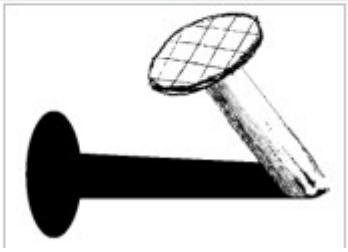
```
024     stSql = "SELECT ""Datum"", ""Erläuterung"", ""Betrag"" FROM ""tbl_Konto"""
025     arFormat = array("Date","String","Number")
026     oResult = oSQL_Statement.executeQuery(stSql)
```

Die Daten aus "tbl_Konto" werden als Text ausgelesen. Anschließend werden die Inhalte in Abhängigkeit vom gewünschten Format (siehe das Array in Zeile 25) als Wert oder Text eingefügt. Die Werte für das Datum und die Zahl werden über die Funktionen **CDate** und **Val** berechnet. Das Einfügen der Werte erfolgt im Gegensatz zum Text mit **setValue**.

```
027     WHILE oResult.next
028        FOR k = 0 TO inCols-1
029           stText = oResult.getString(k+1)
030           SELECT CASE arFormat(k)
031              CASE "Date"
032                 oTable.getCellByPosition(k,i).setValue(CDate(stText))
033              CASE "Number"
034                 oTable.getCellByPosition(k,i).setValue(Val(stText))
035              CASE "String"
036                 oTable.getCellByPosition(k,i).setString(stText)
037           END SELECT
038        NEXT
039        oRows = oTable.getRows()
040        oRows.insertByIndex(oRows.getCount(),1)
041        i = i + 1
042     WEND
```

Tabellendruck mit Bildern in der Tabelle

Tabelle mit Bildern

ID	Name	Inhalt	Bild
1	Stern	Gezeichnet in Draw, exportiert als *.png-Datei	
2	Nägel	Eine Datei im tif-Format	

Das Einfügen von Bildern aus der Datenbank in die Tabelle ist etwas aufwändiger. Sind die Bilder in die Tabelle der Datenbank eingelesen, so müssen sie erst einmal als externe Bilder ausgelesen werden. Danach erfolgt dann das Einfügen des Bildes in die Tabelle. Damit das Bild auch so wie im Screenshot positioniert wird muss zum Schluss noch die Verankerung von der Verankerung **Am Absatz** zur Verankerung **Als Zeichen** geändert werden.

```
028         FOR k = 0 TO inCols-1
029            SELECT CASE arFormat(k)
030               CASE "Date"
031                  stText = oResult.getString(k+1)
032                  oTable.getCellByPosition(k,i).setValue(CDate(stText))
033               CASE "Number"
034                  stText = oResult.getString(k+1)
035                  oTable.getCellByPosition(k,i).setValue(Val(stText))
036               CASE "String"
037                  stText = oResult.getString(k+1)
038                  oTable.getCellByPosition(k,i).setString(stText)
039               CASE "Image"
040                  stText = oResult.getString(k+1)
041                  ImportImage(stText,oNewDoc,oTable.getCellByPosition(k,i))
042               CASE "ImageIntern"
043                  oStream = oResult.getBinaryStream(k+1)
044                  ImportImageIntern(oStream,oNewDoc,oTable.getCellByPosition(k,i))
045            END SELECT
046         NEXT
```

in der Hauptprozedur werden lediglich zwei zusätzliche Formattypen eingefügt.

Ist in der Datenbank lediglich der Link zu dem Bild abgespeichert, dann wird aus der Abfrage ein Text ausgelesen und dieser Link zusammen mit dem Objekt des Writer-Dokumentes und der genauen Position an die Prozedur **ImportImage** weiter gereicht.

Handelt es sich dagegen um ein Bild, das in die Datenbank eingelesen wurde, so muss der Inhalt über **getBinaryStream** ausgelesen werden. Dieser Stream wird genutzt, um ihn als Bilddatei im temporären Pfad von LibreOffice ab zu speichern.

```
001 SUB ImportImageIntern(oStream,oNewDoc,oPos)
002    DIM oPath AS OBJECT, oSimpleFileAccess AS OBJECT
003    DIM stPath AS STRING
```

Das ausgelesene Bild wird im temporären Pfad unter dem Namen **DbFile** zwischengespeichert. Anschließend wird die Prozedur aufgerufen, die das Bild in das Writer-Dokument übernimmt.

```
004    oPath = createUnoService("com.sun.star.util.PathSettings")
005    stPath = oPath.Temp & "/DbFile"
006    oSimpleFileAccess = createUnoService("com.sun.star.ucb.SimpleFileAccess")
007    oSimpleFileAccess.writeFile(stPath, oStream)
008    ImportImage(stPath,oNewDoc,oPos)
009 END SUB
```

Die Prozedur zum Einfügen des Bildes in die Tabelle benötigt die Informationen über den Pfad zum Bild, das Writer-Dokument und die genaue Position innerhalb der Tabelle.

```
001 SUB ImportImage(stPath,oNewDoc,oPos)
002    DIM oCursor AS OBJECT, oTextRange AS OBJECT, oDocCtrl AS OBJECT
003    DIM oDispatcher AS OBJECT, oGraphic AS OBJECT
```

Der sichtbare Cursor wird in die Tabellenzelle gesetzt. Zuerst wird die Position am Ende des Absatzes gewählt und dann der Cursor positioniert.

```
004    oCursor = oNewDoc.CurrentController.ViewCursor
005    oTxtRange = oPos.getEnd
006    oCursor.goToRange(oTxtRange,False)
```

Um das Bild einzufügen muss auf den den Controller zugegriffen werden. Anschließend wird über den **Dispatcher** mit **uno:InsertGraphic** das Bild eingelesen. Die wichtigsten Eigenschaften sind hier benannt: Der Pfad zum Bild und dass das Bild nicht als Verknüpfung eingefügt wird. Prinzipiell ist auch das Einlesen von Bildern über die **Drawpage** der Seite möglich. Nur nimmt der **Dispatcher** hier sehr viel Arbeit ab: Das Bild wird korrekt positioniert, die Größe des Bildes wird maximal der Spaltenbreite angepasst und die Größenverhältnisse werden automatisch beibehalten.

```
007    oDocCtrl   = oNewDoc.CurrentController.Frame
008    oDispatcher = createUnoService("com.sun.star.frame.DispatchHelper")
009    DIM args(1) AS NEW com.sun.star.beans.PropertyValue
010    args(0).Name = "FileName"
011    args(0).Value = stPath
012    args(1).Name = "AsLink"
013    args(1).Value = false
014    oDispatcher.executeDispatch(oDocCtrl, ".uno:InsertGraphic", "", 0, args())
```

Auf die gerade eingefügte Grafik wird über den Index zugegriffen, damit der Anker gesetzt werden kann. Die Standardverankerung an den Absatz erzeugt sonst eine leere Zeile unterhalb des Bildes, so dass der Abstand zur Tabellenzeile nach unten zu groß wird.

```
015    oGraphic =
          oNewDoc.getGraphicObjects.getByIndex(oNewDoc.getGraphicObjects.Count - 1)
```

Mit **AnchorType = 1** wird die Verankerung als Zeichen gesetzt. Der Cursor wird direkt nach dem Bild positioniert und erzeugt keinen zusätzlichen Rand.

```
016    oGraphic.AnchorType = 1
017 END SUB
```

Hier die verschiedenen möglichen Ankertypen als Zusatzinformation:

- 0 → AT_PARAGRAPH
- 1 → AS_CHARACTER
- 2 → AT_PAGE
- 3 → AT_FRAME
- 4 → AT_CHARACTER

Siehe: *https://api.libreoffice.org/docs/idl/ref/namespacecom_1_1sun_1_1star_1_1text.html#a470b1caeda4ff15fee438c8ff9e3d834*

Rechnungen mit Übertrag im Tabellendruck

	LI-Ionen-Akkus und Akkuladegerät		
1	Spiegelreflex-Kamera mit Zoomoptik 25-70mm incl. Speicherkarte 32GB, LI-Ionen-Akkus und Akkuladegerät	834,56 €	834,56 €
		Übertrag:	*9.640,50 €*
		Übertrag:	*9.640,50 €*
1	Spiegelreflex-Kamera	834,56 €	834,56 €

Eine Rechnung mit Übertrag auszudrucken ist über den Report-Designer nur möglich, wenn die Zeilen nicht automatisch anwachsen. Dann kann genau vorausberechnet werden, an welcher Stelle der Seitenumbruch erfolgt und wann dann ein Übertrag eingeblendet werden muss. Sobald die Zeilen der Tabelle sich automatisch an den Textinhalt anpassen und damit größer werden können funktioniert ein Ausdruck wie im obigen Screenshot nicht mehr mit dem Report-Designer.

Die ursprüngliche Prozedur für das Einfügen von Inhalt in eine Tabelle wird erweitert. Zuerst wird das Linienformat für die Absetzung von Schlusssumme und eventuell Überträgen (**TopBorder**, **BottomBorder**) erstellt.

```
018     oBorder = CreateUnoStruct("com.sun.star.table.BorderLine2")
019     oBorder.Color = 0
020     oBorder.LineWidth = 1
021     WHILE oResult.next
022        FOR k = 0 TO inCols-1
023           stText = oResult.getString(k+1)
024           SELECT CASE arFormat(k)
025              CASE "Number"
026                 doValue = Val(stText)
027                 oTable.getCellByPosition(k,i).setValue(doValue)
```

In der 4. Spalte soll die Aufsummierung erfolgen:

```
028                 IF k = 3 THEN
```

Die Summe des vorhergehenden Durchgangs wird in **doCarryOver** gespeichert. Der gerade neu ausgelesene Wert wird zu der vorhergehenden Summe addiert und in **doVal** gespeichert.

```
029                    doCarryOver = doVal
030                    doVal = doVal + doValue
031                 END IF
032              CASE "String"
033                 oTable.getCellByPosition(k,i).setString(stText)
034           END SELECT
035        NEXT
036        oRows = oTable.getRows()
037        oRows.insertByIndex(oRows.getCount(),1)
038        i = i + 1
```

Nachdem eine neue Tabellenzeile eingefügt wurde muss der sichtbare Cursor in diese Zeile gesetzt werden. Für die neue Zeile wird die Seitenzahl bestimmt. Ist die Seitenzahl größer als die vorhergehende Seitenzahl, so muss ein Übertrag eingefügt werden. Der Übertrag muss vor der vorhergehenden Zeile erfolgen.

```
039         oPos = oTable.getCellByPosition(0,i)
040         oTxtRange = oPos.getEnd
041         oCursor.goToRange(oTxtRange,False)
042         IF oCursor.Page > inPageStart THEN
```

Es werden 2 leere Zeile oberhalb des letzten Eintrags eingefügt. Beide Zeilen erhalten den Eintrag "Übertrag:" in der 3. Spalte und den Wert für den Übertrag in der 4. Spalte. Für die entsprechenden Zellen wird das Absatzformat "Übertrag" gesetzt. Achtung: Die Schriftart für das Absatzformat "Übertrag" darf für die Positionierung nicht größer sein als die für den Tabelleninhalt.

```
043             FOR ink = 1 TO 2
044                oRows = oTable.getRows()
045                oRows.insertByIndex(oRows.getCount()-2,1)
046                oTable.getCellByPosition(2,i-1).setString("Übertrag:")
047                oTable.getCellByPosition(3,i-1).setValue(doCarryOver)
048                oTable.getCellByPosition(2,i-1).getEnd.ParaStyleName = "Übertrag"
049                oTable.getCellByPosition(3,i-1).getEnd.ParaStyleName = "Übertrag"
050                i = i + 1
051             NEXT
```

Der Cursor wird jetzt von der unteren leeren Tabellenzeile 2 Zeilen nach oben auf die 2. Zeile für den Übertrag gesetzt. Die 2. Zeile für den Übertrag darf nicht auf der gleichen Seite wie die 1. Zeile für den Übertrag stehen. Die 2. Zeile kann zufällig auf der gleichen Seite wie die erste Zeile stehen, wenn der letzte inhaltliche Eintrag einen Absatzumbruch enthält. In dem Fall nimmt die Tabellenzeile für den letzten Eintrag eine größere Höhe in Anspruch. Die 1. Zeile für den Übertrag steht auf **inPageStart**.

```
052             oCursor.goUp(2,False)
053             DO WHILE oCursor.Page = inPageStart
```

Es werden so lange zwischen der 1. Zeile für den Übertrag und der 2. Zeile für den Übertrag Tabellenzeilen eingefügt, bis die 2. Zeile auf der Folgezeile steht.

```
054                oRows = oTable.getRows()
055                oRows.insertByIndex(oRows.getCount()-3,1)
056                oPos = oTable.getCellByPosition(0,i-1)
057                oTxtRange = oPos.getEnd
058                oCursor.gotoRange(oTxtRange,false)
059                i = i + 1
060             LOOP
```

Die jetzt letzte Seite wird als **inStartPage** zur neuen Vergleichsseite für den nächsten Übertrag.

```
061             inPageStart = oCursor.Page
062          END IF
063       WEND
```

Durch die Schleife bleibt eine leere Inhaltszeile übrig. Diese Zeile wird für die Summierung genutzt. Wie beim Übertrag erfolgen hier die Einträge und die Formatierungen. Zusätzlich wird die oben stehende Umrandung als Abgrenzung für den Gesamtpreis in die Zellen eingefügt.

```
064       oTable.getCellByPosition(2,i).setString("Gesamt:")
065       oTable.getCellByPosition(3,i).setValue(doVal)
066       oTable.getCellByPosition(2,i).getEnd.ParaStyleName = "Übertrag"
067       oTable.getCellByPosition(3,i).getEnd.ParaStyleName = "Übertrag"
068       oTable.getCellByPosition(2,i).TopBorder = oBorder
069       oTable.getCellByPosition(3,i).TopBorder = oBorder
070    END SUB
```

Aufruf von Anwendungen zum Öffnen von Dateien

Mit dieser Prozedur kann durch einen Mausklick in ein Textfeld ein Programm aufgerufen werden, das im eigenen Betriebssystem mit der Dateinamensendung verbunden ist. Damit werden auch Links ins Internet oder sogar das Öffnen des Mailprogramms mit einer bestimmten Mailadresse möglich, die gerade in der Datenbank gespeichert wurde.

Siehe zu diesem Abschnitt auch die Beispieldatenbank «Beispiel_Mailstart_Dateiaufruf.odb»[47]

```
SUB Link_oeffnen
   DIM oDoc AS OBJECT
   DIM oDrawpage AS OBJECT
   DIM oForm AS OBJECT
   DIM oFeld AS OBJECT
   DIM oShell AS OBJECT
   DIM stFeld AS STRING
   oDoc = thisComponent
   oDrawpage = oDoc.Drawpage
   oForm = oDrawpage.Forms.getByName("Formular")
   oFeld = oForm.getByName("Adresse")
```

Aus dem benannten Feld wird der Inhalt ausgelesen. Dies kann eine Webadresse, beginnend mit **'http://'**, eine Mailadresse mit einem **'@'** oder ein einfaches Dokument sein, das durch eine entsprechende Pfadangabe aufgesucht werden soll (z.B. externe Bilder, *.pdf-Dateien, Tondokumente ...).

Hinweis

Der Verwendung von Sonderzeichen in den Pfaden und Dateinamen sollte hier möglichst vermieden werden. Die Links sind dann teilweise auf diese Art nicht mehr aufrufbar.

```
   stFeld = oFeld.Text
   IF stFeld = "" THEN
      EXIT SUB
   END IF
```

Ist das Feld leer, so soll das Makro sofort enden. Bei der Eingabe passiert es ja sehr oft, dass Felder mit der Maus aufgesucht werden. Ein Mausklick in das Feld, um dort zum ersten Mal schreiben zu können, soll aber noch nicht das Makro ausführen.

Jetzt wird gesucht, ob in dem Feld ein **'@'** enthalten ist. Dies deutet auf eine E-Mail-Adresse hin. Es soll also das Mailprogramm gestartet werden und eine Mail an diese Mailadresse gesandt werden.

```
   IF InStr(stFeld,"@") THEN
      stFeld = "mailto:"+stFeld
```

Ist kein **'@'** vorhanden, so wird der Begriff so konvertiert, dass die Datei im Dateisystem gefunden werden kann. Steht ein **'http://'** am Anfang, so wird bei dieser Funktion nicht im Dateisystem, sondern über den Webbrowser direkt im Internet nachgesehen. Ansonsten beginnt der erstellte Pfad anschließend mit dem Begriff **'file:///'**

```
   ELSE
      stFeld = convertToUrl(stFeld)
   END IF
```

Bei der Verwendung von Sonderzeichen in URLs kann es sinnvoll sein, die Konvertierung für den Pfad zu unterlassen. Der Shell-Befehl funktioniert auch mit der systeminternen Schreibweise. Hier müsste dann allerdings separat für die Endungen **'http://'** und **'https://'** eine Konvertierung vorgenommen werden. Jetzt wird das Programm aufgesucht, das in dem eigenen Betriebssystem mit der entsprechenden Dateiendung verbunden ist. Bei dem Stichwort **'mailto:'** ist dies das Mailprogramm, bei **'http://'** der Webbrowser und bei allen anderen ist die Entscheidung des Systems mit den Endungen der Datei verbunden.

```
   oShell = createUnoService("com.sun.star.system.SystemShellExecute")
   oShell.execute(stFeld,,0)
END SUB
```

47 In der Datenbank «Beispiel_Formular_Eingabekontrolle.odb» ist hierzu eine Prozedur enthalten, die alle Anwendungen öffnet, die irgendwie als Dateiendung mit dem System verbunden sind: Webseiten, Email-Programme, Bilddateien, Textdateien ...

Aufruf eines Mailprogramms mit Inhaltsvorgaben

Eine Erweiterung des vorhergehenden Beispiels zum Programmaufruf stellt dieser Aufruf eines Mailprogramms mit Vorgaben in der Betreffzeile und inhaltlichen Vorgaben dar.

Siehe auch zu diesem Abschnitt die Beispieldatenbank «Mailstart_Dateiaufruf.odb»

Der Mailaufruf erfolgt mit **'mailto:Empfänger?subject= &body= &cc= &bcc= '**. Die letzten beiden Eingaben sind im Formular allerdings nicht aufgeführt. Anhänge kommen in der Definition von **'mailto'** nicht vor. Manchmal funktioniert allerdings **'attachment='**.

```
001 SUB Mail_Aufruf
002    DIM oDoc AS OBJECT
003    DIM oDrawpage AS OBJECT
004    DIM oForm AS OBJECT
005    DIM oFeld1 AS OBJECT
006    DIM oFeld2 AS OBJECT
007    DIM oFeld3 AS OBJECT
008    DIM oFeld4 AS OBJECT
009    DIM oShell AS OBJECT
010    DIM stFeld1 AS STRING
011    DIM stFeld2 AS STRING
012    DIM stFeld3 AS STRING
013    DIM stFeld4 AS STRING
014    oDoc = thisComponent
015    oDrawpage = oDoc.Drawpage
016    oForm = oDrawpage.Forms.getByName("Formular")
017    oFeld1 = oForm.getByName("E_Mail_Adresse")
018    oFeld2 = oForm.getByName("E_Mail_Betreff")
019    oFeld3 = oForm.getByName("E_Mail_Inhalt")
020    stFeld1 = oFeld1.Text
021    IF stFeld1 = "" THEN
022       msgbox "Keine Mailadresse vorhanden." & CHR(13) &
          "Das Mailprogramm wird nicht aufgerufen" , 48, "Mail senden"
023       EXIT SUB
024    END IF
```

Die Konvertierung zu URL ist notwendig, damit Sonderzeichen und Zeilenumbrüche den Aufruf nicht stören. Dabei wird allerdings auch der Begriff **'file:///'** vorangestellt. Diese 8 Zeichen zu Beginn werden nicht mit übernommen.

```
025    stFeld2 = Mid(ConvertToUrl(oFeld2.Text),9)
026    stFeld3 = Mid(ConvertToUrl(oFeld3.Text),9)
```

Im Gegensatz zum einfachen Programmaufruf werden hier Details des Mailaufrufes über den Aufruf des Mailprogramms mitgegeben.

```
027    oShell = createUnoService("com.sun.star.system.SystemShellExecute")
028    oShell.execute("mailto:" + stFeld1 + "?subject=" + stFeld2 + "&body=" +
          stFeld3,,0)
029 END SUB
```

Hinweis

Das Versenden von Mails mit Hilfe des Mailprogramms kann auch mit folgendem Code erfolgen. Ab LO 4.2 kann hier auch über das Attribut Body der Inhalt der Mail mit eingefügt werden. Dieser Code ermöglicht außerdem das Anfügen von Anhängen. Für mehrere Anhänge muss einfach das Array erweitert werden. Auch Adressen im CC sowie im BCC werden in ein Array geschrieben.

```
027     DIM attachs(0)
028     IF GetGuiType() = 1 THEN
029        oMailer =
              createUnoService("com.sun.star.system.SimpleSystemMail")
              ' Sonst Linux/Mac
030        ELSE
031        oMailer =
              createUnoService("com.sun.star.system.SimpleCommandMail")
032     END IF
033     oMailProgramm = oMailer.querySimpleMailClient()
034     oNeueNachricht = oMailProgramm.createSimpleMailMessage()
035     oNeueNachricht.setRecipient(stFeld1)
036     oNeueNachricht.setSubject(stFeld2)
037     oNeueNachricht.Body(stFeld3)
038     attachs(0) = "file:///..."
039     oNeueNachricht.setAttachement(attachs())
040     oMailprogramm.sendSimpleMailMessage(oNeueNachricht, 0 )
041  END SUB
```

Zu den möglichen Parametern siehe: *http://api.libreoffice.org/docs/idl/ref/interfacecom_1_1sun_1_1star_1_1system_1_1XSimpleMailMessage.html*
Manchmal kommt es unter Linux zu Problemen mit Kommas in Subject und Body. Der Text wird dadurch einfach abgeschnitten, taucht teilweise auch als Empfänger auf. Auch Zeilenumbrüche können zu Problemen führen. Sobald allerdings unter **Extras → Optionen → Internet → E-Mail** das passende Mailprogramm angegeben wurde klappt die Übertragung einwandfrei.

Aufruf einer Kartenansicht zu einer Adresse

Eine Datenbank enthält lauter Adressen. Jetzt soll zu einer Adresse aufgezeigt werden, in welcher Umgebung denn das Haus liegt. Die folgende Prozedur «MapPosition» wird mit einem Button gestartet, der in dem gleichen Formular liegt, in dem die Angaben zur Adresse verzeichnet sind.[48]

```
001  SUB MapPosition(oEvent AS OBJECT)
002     DIM oForm AS OBJECT, oShell AS OBJECT
003     DIM i AS INTEGER
004     DIM stLink AS STRING, stTag AS STRING
005     DIM arFields()
006     stTag = oEvent.Source.Model.Tag
007     oForm = oEvent.Source.Model.Parent
008     arFields = Split(stTag,",")
```

In den Zusatzinformationen des Buttons sind, durch Kommas getrennt, die Namen der Felder aufgeführt, die zusammen die Adresse ergeben. Dies sind in der Beispieldatenbank **comPLZOrt,txtStraße**. Das erste Feld ist ein Kombinationsfeld, das die Postleitzahl und den Ort enthält, das zweite Feld enthält die Straße und die Hausnummer. Die beiden Feldbezeichnungen werden voneinander getrennt und in ein Array geschrieben.

```
009     FOR i = LBound(arFields) TO UBound(arFields)
010        IF stLink = "" THEN
011           stLink = oForm.getByName(arFields(i)).CurrentValue
012        ELSE
013           stLink = stLink & "+" & oForm.getByName(arFields(i)).CurrentValue
014        END IF
```

48 Siehe Beispiel_Formular_Eingabekontrolle.odb

```
015     NEXT i
```

Die Inhalte der beiden Felder werden ausgelesen und mit einem **+** verbunden in der Variablen **stLink** gespeichert. Dieser Suchstring wird jetzt in den Link für nominatim.openstreetmap.org eingefügt. Beim Einfügen wird darauf geachtet, dass auch die Leerzeilen in dem String mit **+** ausgefüllt werden. Dies geschieht, indem der String einfach einmal an den Leerzeichen durch **Split** aufgetrennt wird und dann wieder über **Join** mit einem **+** die Teile verbunden werden.

```
016     IF stLink <> "" THEN
017        stLink = "https://nominatim.openstreetmap.org/search.php?q=" &
              Join(Split(stLink),"+") & "&polygon_geojson=1&viewbox="
018        oShell = createUnoService("com.sun.star.system.SystemShellExecute")
019        oShell.execute(stLink,,0)
020     END IF
021  END SUB
```

Die weiteren Elemente des Links sind lediglich aus dem Link entstanden, den die Website bei direkter Nutzung der Suchfunktion angibt. Wie in den vorhergehenden Beispielen wird dieser Link über die **SystemShell** gestartet. Dort wird dann der Browser aufgerufen, der bei einer in der Karte verzeichneten Adresse die auch direkt findet.[49]

Mauszeiger ändern

Manchmal erscheint es sinnvoll, die Mauszeiger so anzupassen, dass sie Zusatzinformationen zur Verwendung des Inhaltes eines Feldes geben.

Änderung beim Überfahren eines Links

Im Internet üblich, bei Base nachgebaut: Der Mauszeiger fährt über ein Textfeld und verändert seine Form zu einer zeigenden Hand. Der enthaltene Text kann jetzt noch in den Eigenschaften des Feldes zu der Farbe Blau und unterstrichen geändert werden – schon ist der Eindruck eines Links aus dem Internet perfekt. Jeder Nutzer erwartet nach einem Klick, dass sich ein externes Programm öffnet.

Siehe auch zu diesem Abschnitt die Beispieldatenbank «Mailstart_Dateiaufruf.odb»[50].

Diese kurze Prozedur sollte mit dem Ereignis **'Maus innerhalb'** des Textfeldes verbunden werden.

```
001  SUB Mauszeiger(oEvent AS OBJECT)
002     REM Siehe auch Standardbibliotheken: Tools → ModuleControls → SwitchMousePointer
003     DIM oPointer AS OBJECT
004     oPointer = createUnoService("com.sun.star.awt.Pointer")
005     oPointer.setType(27) 'Typen in com.sun.star.awt.SystemPointer
006     oEvent.Source.Peer.SetPointer(oPointer)
007  END SUB
```

Änderung bei gedrückter Strg-Taste und Mausklick

```
001  SUB Mauszeiger(oEvent AS OBJECT)
002     DIM oPointer AS OBJECT
003     oPointer = createUnoService("com.sun.star.awt.Pointer")
004     IF oEvent.Modifiers = 2 THEN
005        'KeyModifier (ohne: 0 | Shift: 1 | Ctrl: 2 | Alt: 4 …),
              Typen in com.sun.star.awt.KeyModifier
006        oPointer.setType(0) 'Typen in com.sun.star.awt.SystemPointer
007     ELSE
008        oPointer.setType(3)
009     END IF
010     oEvent.Source.Peer.SetPointer(oPointer)
011  END SUB
```

49 Bei Nutzung dieser Möglichkeit der Kartendarstellung sollten die Bedingungen der Website beachtet werden: *https://operations.osmfoundation.org/policies/nominatim/* .

50 Die Datenbank «Beispiel_Mailstart_Dateiaufruf.odb» ist diesem Handbuch beigefügt.

Über den **KeyModifier** wird ermittelt, ob eine der entsprechenden Tasten zusätzlich zu dem Mausklick an der Auslösung des Makros beteiligt war. Hier wurde mit '2' als zusätzliche Taste *STRG* ausgewählt. Wird *STRG* nicht gedrückt, so wird auf den Textcursor geschaltet.

Formulare ohne Symbolleisten präsentieren

Neunutzer von Base sind häufig irritiert, dass z.B. eine Menüleiste existiert, diese aber im Formular so gar nicht verfügbar ist. Diese Menüleisten können auf verschiedene Arten ausgeblendet werden. Am erfolgreichsten unter allen LO-Versionen sind die beiden im Folgenden vorgestellten Vorgehensweisen.

Fenstergrößen und Symbolleisten werden in der Regel über ein Makro beeinflusst, das in einem Formulardokument unter **Extras → Anpassen → Ereignisse → Dokument öffnen** gestartet wird. Gemeint ist hier das Dokument, nicht ein einzelnes Haupt- oder Unterformular.

Formulare ohne Symbolleisten in einem Fenster

Ein Fenster lässt sich in der Größe variieren. Über den entsprechenden Button lässt es sich auch schließen. Diese Aufgaben übernimmt der Window-Manager des jeweiligen Betriebssystems. Lage und Größe des Fensters auf dem Bildschirm kann beim Start über ein Makro mitgegeben werden.

```
001 SUB Symbolleisten_Ausblenden
002     DIM oFrame AS OBJECT
003     DIM oWin AS OBJECT
004     DIM oLayoutMng AS OBJECT
005     DIM aElemente()
006     oFrame = StarDesktop.getCurrentFrame()
```

Diese Startvariante ist für **eigenständige Formulare** geeignet, nicht aber für Formulare in der Basedatei. In der Basedatei würde dort das Hauptfenster, nicht aber die dem **untergeordneten Formulare** ohne Symbolleiste versehen. Dort erfolgt der Start über

```
001 SUB Symbolleisten_Ausblenden(oEvent AS OBJECT)
002     DIM oFrame AS OBJECT
003     DIM oWin AS OBJECT
004     DIM oLayoutMng AS OBJECT
005     DIM aElemente()
006     oFrame = oEvent.Source.CurrentController.Frame
```

Der Titel für das Formular wird in der Titelleiste des Fensters angezeigt.

```
007     oFrame.setTitle "Mein Formular"
008     oWin = oFrame.getContainerWindow()
```

Das Fenster wird auf die maximale Größe eingestellt. Dies entspricht nicht dem Vollbildmodus, da z.B. eine Kontrollleiste noch sichtbar ist und das Fenster eine Titelleiste hat, über die die Größe des Fensters geändert und das Fenster geschlossen werden kann.

```
009     oWin.IsMaximized = true
```

Es besteht auch die Möglichkeit, das Fenster in einer ganz bestimmten Größe und mit einer festen Position darzustellen. Dies würde mit **'oWin.setPosSize(0,0,600,400,15)'** geschehen. Hier wird das Fenster an der linken oberen Ecke des Bildschirms mit einer Breite von 600 Punkten und einer Höhe von 400 Punkten dargestellt. Die letzte Ziffer weist darauf hin, dass alle Punkte angegeben wurden. Sie wird als **'Flag'** bezeichnet. Das **'Flag'** wird aus den folgenden Werten über eine Summierung berechnet: x=1, y=2, Breite=4, Höhe=8. Da x, y, Breite und Höhe angegeben sind, hat das **'Flag'** die Größe 1 + 2 + 4 + 8 = 15.

Da es inzwischen auch viele verschiedene Bildschirmauflösungen gibt und die Fenstergröße eventuell angepasst werden soll, hier die Ermittlung der Bildschirmauflösung in dpi: **'oWin.Info.PixelPerMeterX * 2.54/100'**. Die Auflösung wird genauso für die y-Achse angegeben ist aber vermutlich immer gleich.

```
010     oLayoutMng = oFrame.LayoutManager
011     aElemente = oLayoutMng.getElements()
```

```
012     FOR i = LBound(aElemente) TO UBound(aElemente)
013        IF aElemente(i).ResourceURL =
              "private:resource/toolbar/formsnavigationbar" THEN
014        ELSE
015           oLayoutMng.hideElement(aElemente(i).ResourceURL)
016        END IF
017     NEXT
018     ThisComponent.CurrentController.Sidebar.Visible = False
019     ThisComponent.CurrentController.ViewSettings.ZoomValue = 200
020     ThisComponent.CurrentController.ViewSettings.ShowRulers = False
021     ThisComponent.CurrentController.ViewSettings.ShowParaBreaks = False
022  END SUB
```

Wenn es sich um die Navigationsleiste handelt, soll nichts geschehen. Das Formular soll schließlich bedienbar bleiben, wenn nicht das Kontrollfeld für die Navigationsleiste eingebaut und die Navigationsleiste sowieso ausgeblendet wurde. Nur wenn es sich nicht um die **Navigationsleiste** handelt, soll die entsprechende Leiste verborgen werden. Deswegen erfolgt zu dieser Bedingung **keine Aktion**. Neben den Symbolleisten wird anschließend noch die Seitenleiste unsichtbar gemacht, in diesem Beispiel dann auch noch der **ZoomValue** eingestellt, die Lineale links und oben ausgeblendet und die Absatzmarke nicht mehr angezeigt, falls sonst im Writer eben Formatierungszeichen angezeigt werden.

Bei unterschiedlichen Bildschirmen kann es passieren, dass der voreingestellte **ZoomValue** nicht die gleiche Formularansicht wiedergibt. Hier kann das Auslesen von Bildschirmbreite und Bildschirmhöhe helfen:

```
018     ' Ausschnitt mit allen Elementen des Formulars: 1487*765
019     ' bei 96 dpi, 3779 PixelPerMeter – gezoomte Bildschirme haben mehr dpi
020     inDpiX = 1440 \ TwipsPerPixelX()
021     inDpiY = 1440 \ TwipsPerPixelY()
022     inx = Int(oWin.Info.Width * 100 * 96 / (1487 * inDpiX))
023     iny = Int(oWin.Info.Height * 100 * 96 / (765 * inDpiY))
024     IF inx < iny THEN
025        inZoom = inx
026     ELSE
027        inZoom = iny
028     END IF
```

Über einen Screenshot wurde die Größe des Formulars (links oben unterhalb der Symbolleisten beginnen bis rechts unten incl. des letzten Elementes; ggf. auch die Navigationsleiste mit einbeziehen) in Pixeln bei einem **ZoomValue** von 100 ermittelt. Das Verhältnis von tatsächlicher Bildschirmbreite zu Formularbreite bzw. Bildschirmhöhe zu Formularhöhe soll den neuen Prozentwert für den **ZoomValue** bestimmen. Damit auch das gesamte Formular auf den Bildschirm passt soll der kleinere Wert übernommen werden. Das Ganze wird in die vorhergehende Prozedur nach der Deklaration von **oWin** eingebaut. Statt der '200' für den **ZoomValue** steht dort dann eben **inZoom**. Alle hier auftauchenden Zahlenvariablen sind Ganzzahlen im Integer-Format.

Bei Addons im Bereich der Symbolleisten wird die Eigenschaft **ResourceURL** leider etwas hinter einer Integer-Variablen versteckt. Hier ist dann zur Bestimmung der URL der folgende Weg notwendig:

```
016     obj = aElemente(i)
017     invoc = CreateUnoService("com.sun.star.script.Invocation")
018     invocCurrObj = invoc.createInstanceWithArguments(Array(obj))
019     ResourceURL= invocCurrObj.getValue("ResourceURL")
020     oLayoutMng.hideElement(ResourceURL)
```

Dieser Code sollte gegebenenfalls in die **FOR**-Schleife vor **END IF** eingefügt werden.

Werden die Symbolleisten nicht wieder direkt beim Beenden des Formulars eingeblendet, so bleiben sie weiterhin verborgen. Sie können natürlich über **Ansicht → Symbolleisten** wieder aufgerufen werden. Etwas irritierend ist es jedoch, wenn gerade die Standardleiste (**Ansicht → Symbolleisten → Standardleiste**) oder die Statusleiste (**Ansicht → Statusleiste**) fehlt.

Mit dieser Prozedur werden die Symbolleisten aus dem Versteck (**'hideElement'**) wieder hervorgeholt (**'showElement'**). Der Kommentar enthält die Leisten, die oben als sonst fehlende Leisten am ehesten auffallen.

```
001 SUB Symbolleisten_Einblenden
002    DIM oFrame AS OBJECT
003    DIM oLayoutMng AS OBJECT
004    DIM aElemente()
005    oFrame = StarDesktop.getCurrentFrame()
006    oLayoutMng = oFrame.LayoutManager
007    aElemente = oLayoutMng.getElements()
008    FOR i = LBound(aElemente) TO UBound(aElemente)
009       oLayoutMng.showElement(aElemente(i).ResourceURL)
010    NEXT
011    ' eventuell fehlende wichtige Elemente:
012    ' "private:resource/toolbar/standardbar"
013    ' "private:resource/statusbar/statusbar"
014    ThisComponent.CurrentController.Sidebar.Visible = True
015    ThisComponent.CurrentController.ViewSettings.ZoomValue = 100
016    ThisComponent.CurrentController.ViewSettings.ShowRulers = True
017    ThisComponent.CurrentController.ViewSettings.ShowParaBreaks = True
018 END SUB
```

Die Makros werden an die Eigenschaften des Formularfensters gebunden: **Extras → Anpassen → Ereignisse → Dokument öffnen → Symbolleisten_Ausblenden** bzw. **… Dokument wird geschlossen → Symbolleisten_Einblenden**

Auch diese Prozedur sowie die folgende muss in internen Formularen von Base anders gestartet werden, da sonst das Hauptfenster eingestellt wird.

```
001 SUB Symbolleisten_Einblenden(oEvent AS OBJECT)
       ' …
002    oFrame = oEvent.Source.CurrentController.Frame
```

Leider tauchen häufig Symbolleisten trotzdem nicht wieder auf. In hartnäckigen Fällen kann es daher helfen, nicht die Elemente auszulesen, die der Layoutmanager bereits kennt, sondern definitiv bestimmte Symbolleisten erst zu erstellen und danach schließlich zu zeigen:

```
001 Sub Symbolleisten_Einblenden
002    DIM oFrame AS OBJECT
003    DIM oLayoutMng AS OBJECT
004    DIM i AS INTEGER
005    DIM aElemente(5) AS STRING
006    oFrame = StarDesktop.getCurrentFrame()
007    oLayoutMng = oFrame.LayoutManager
008    aElemente(0) = "private:resource/menubar/menubar"
009    aElemente(1) = "private:resource/statusbar/statusbar"
010    aElemente(2) = "private:resource/toolbar/formsnavigationbar"
011    aElemente(3) = "private:resource/toolbar/standardbar"
012    aElemente(4) = "private:resource/toolbar/formdesign"
013    aElemente(5) = "private:resource/toolbar/formcontrols"
014    FOR i = LBound(aElemente) TO UBound(aElemente)
015       IF NOT(oLayoutMng.requestElement(aElemente(i))) THEN
016          oLayoutMng.createElement(aElemente(i))
017       END IF
018    oLayoutMng.showElement(aElemente(i))
019    NEXT
020    ThisComponent.CurrentController.Sidebar.Visible = True
021    ThisComponent.store()
022 END SUB
```

Die darzustellenden Symbolleisten werden explizit benannt. Ist eine der entsprechenden Symbolleisten nicht für den Layoutmanager vorhanden, so wird sie zuerst über **createElement** erstellt und danach über **showElement** gezeigt. Deshalb muss das Dokument anschließend abgespeichert werden. Diese Prozedur muss über **Extras → Anpassen → Ereignisse → Dokument wird geschlossen → Symbolleisten_Einblenden** eingebunden werden.

Formulare im Vollbildmodus

Beim Vollbildmodus wird der gesamte Bildschirm vom Formular bedeckt. Hier steht keine Kontrollleiste o.ä. mehr zur Verfügung, die gegebenenfalls anzeigt, ob noch irgendwelche anderen Programme laufen.

```
001 FUNCTION Fullscreen(boSwitch AS BOOLEAN)
002    DIM oDispatcher AS OBJECT
003    DIM Props(0) AS NEW com.sun.star.beans.PropertyValue
004    oDispatcher = createUnoService("com.sun.star.frame.DispatchHelper")
005    Props(0).Name = "FullScreen"
006    Props(0).Value = boSwitch
007    oDispatcher.executeDispatch(ThisComponent.CurrentController.Frame,
008       ".uno:FullScreen", "", 0, Props())
009 END FUNCTION
```

Diese Funktion wird durch die folgenden Prozeduren eingeschaltet. In den Prozeduren läuft gleichzeitig die vorhergehende Prozedur zum Ausblenden der Symbolleisten ab – sonst erscheint die Symbolleiste, mit der der Vollbildmodus wieder ausgeschaltet werden kann. Auch dies ist eine Symbolleiste, wenn auch nur mit einem Symbol.

```
001 SUB Vollbild_ein
002    Fullscreen(true)
003    Symbolleisten_Ausblenden
004 END SUB
```

Aus dem Vollbild-Modus geht es wieder heraus über die **'ESC'**-Taste. Wenn stattdessen ein Button mit einem entsprechenden Befehl belegt werden soll, so reichen auch die folgenden Zeilen:

```
001 SUB Vollbild_aus
002    Fullscreen(false)
003    Symbolleisten_Einblenden
004 END SUB
```

Formular direkt beim Öffnen der Datenbankdatei starten

Wenn jetzt schon die Symbolleisten weg sind oder gar das Formular im Vollbildmodus erscheint, dann müsste nur noch die Datenbankdatei beim Öffnen direkt in dieses Formular hinein starten. Der einfache Befehl zum Öffnen von Formularen reicht dabei leider nicht aus, da die Datenbankverbindung beim Öffnen des Base-Dokumentes noch nicht besteht.

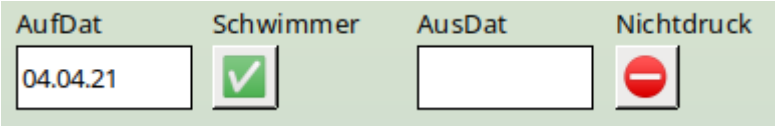
Das folgende Makro wird über **Extras → Anpassen → Ereignisse → Dokument öffnen** gestartet. Dabei ist **Speichern in → Datenbankdatei.odb** zu wählen.

```
001 SUB Formular_Direktstart
002    DIM oDatenquelle AS OBJECT
003    oDatenquelle = ThisDatabaseDocument.CurrentController
004    If NOT (oDatenquelle.isConnected()) THEN
005       oDatenquelle.connect()
006    END IF
007    ThisDatabaseDocument.FormDocuments.getByName("Formularname").open
008    REM alternativ geht auch:
          'oDatenquelle.loadComponent(com.sun.star.sdb.application.DatabaseObject.FORM,
          "Formularname",FALSE)
009 END SUB
```

Zuerst muss der Kontakt mit der Datenquelle hergestellt werden. Der Controller hängt ebenso mit **ThisDatabaseDocument** zusammen wie das Formular. Anschließend kann das Formular gestartet werden und liest auch die Datenbankinhalte aus.

Markierfelder durch Schaltflächen ersetzen

Markierfelder und auch Optionsfelder sind von der Größe und dem Erscheinungsbild her nicht bearbeitbar. Die folgende Lösung erstellt statt Markierfeldern Schaltflächen, die mit einem entsprechenden Symbol versehen sind und wie gewohnt als Markierfelder ansprechbar sind.



Buttons mit Symbolen aus Fonts statt Markierfelder: Vergrößerbar und optisch anpassbar.

Zuerst werden globale Variablen für die beiden Zeichen erstellt, die auf den Buttons abgebildet werden sollen. Die Variablen werden in der Prozedur «BoolStart» mit dem entsprechenden Inhalt versehen, der in dem Beispiel je einem UTF8-Zeichen entspricht.

```
001 GLOBAL stChecked AS STRING
002 GLOBAL stUnChecked AS STRING
```

Damit das Makro nicht speziell auf ein Formular angepasst ist wird der Name des nachgebauten Markierfeldes in einem versteckten Kontrollfeld «hidCheckbox» notiert; bei mehreren Feldern sind diese durch ein Semikolon getrennt. In den Zusatzinformationen jedes einzelnen nachgebauten Markierfeldes steht dann der Name des dazugehörigen Datenfeldes aus der Datenquelle.

Das Einstellen des Wertes des Boolean-Feldes erfolgt beim Wechsel des Datensatzes im Formular. Die Prozedur wird also über **Ereignisse → Nach dem Datensatzwechsel** ausgelöst.

```
001 SUB BoolStart(oEvent AS OBJECT)
002    DIM oForm AS OBJECT
003    DIM oField AS OBJECT
004    DIM stCheckbox AS STRING
005    DIM stCheck AS STRING
006    DIM stValue AS STRING
007    DIM arCheck()
008    oForm = oEvent.Source
```

Damit das Makro nur dann durchläuft, wenn wirklich die entsprechende Checkbox verfügbar ist, wird hier das entsprechende **UnoInterface** abgefragt. Anschließend wird die Variable für «True» und «False» mit dem entsprechenden Zeichen versehen. Schließlich wird aus dem versteckten Feld ausgelesen, welche anderen Felder jetzt Markierfelder darstellen sollen.

```
009    IF hasUnoInterfaces( oForm, "com.sun.star.form.XForm" ) THEN
010       stChecked = "  "
011       stUnChecked = "  "
012       stCheckbox = oForm.getByName("hidCheckbox").HiddenValue
013       arCheck = split(stCheckbox,";")
014       FOR n = LBound(arCheck()) TO UBound(arCheck())
015          oField = oForm.getByName(Trim(arCheck(n)))
016          stCheck = oField.Tag
```

Wenn es sich um einen leeren Datensatz handelt (letzter neuer Datensatz), dann soll der Wert für das Feld 'False' sein. Es wird also kein Markierfeld mit 3 verschiedenen Einstellmöglichkeiten dargestellt.

Ist der ausgelesene Wert aus der Tabelle 'True', dann wird das Markierfeld mit dem entsprechenen Symbol versehen. Für alle anderen Werte wird 'false' angenommen.

```
017          IF oForm.IsRowCountFinal AND oForm.RowCount = 0 THEN
018             stValue = "false"
019          ELSE
020             stValue = oForm.getString(oForm.findColumn(stCheck))
021          END IF
022          IF stValue = "true" THEN
023             oField.Label = stChecked
024          ELSE
025             oField.Label = stUnChecked
026          END IF
027       NEXT
028    END IF
029 END SUB
```

Der Button wird in der Regel durch die Maus ausgelöst. Damit aber nicht ein Ansteuern über den Tabulator, das auch das **Ereignis → Taste gedrückt** auslöst, das Feld umstellt, muss hier vorher der KeyCode für den Tablulator, 1282, siehe *http://api.libreoffice.org/docs/idl/ref/namespacecom_1_1sun_1_1star_1_1awt_1_1Key.html* herausgefiltert werden. Dies wurde hier in eine separate Prozedur ausgelagert, da die Schaltfläche auch über die Maus ausgelöst werden kann.

```
001 SUB BoolChangeKey(oEvent AS OBJECT)
002    IF oEvent.KeyCode <> 1282 THEN
003       BoolChange(oEvent)
004    END IF
005 END SUB
```

Wenn die Beschriftung des Feldes auf 'stChecked' steht, dann wird sie durch dieses Makro umgestellt auf 'stUnChecked' außerdem wird das Datenfeld auf 'false' eingestellt.

Das Makro wird von der Schaltfläche über **Ereignisse → Taste gedrückt** (Umweg über SUB BoolChangeKey) und über **Ereignisse → Maustaste gedrückt** ausgelöst.

```
001 SUB BoolChange(oEvent AS OBJECT)
002    DIM oField AS OBJECT
003    DIM oForm AS OBJECT
004    oField = oEvent.Source.Model
005    oForm = oField.Parent
006    IF oField.Label = stChecked THEN
007       oField.Label = stUnChecked
008       oForm.updateBoolean(oForm.findColumn(oField.Tag),false)
009    ELSE
010       oField.Label = stChecked
011       oForm.updateBoolean(oForm.findColumn(oField.Tag),true)
012    END IF
013 END SUB
```

MySQL-Datenbank mit Makros ansprechen

Sämtliche bisher vorgestellten Makros wurden mit der internen HSQLDB verbunden. Bei der Arbeit mit externen Datenbanken sind ein paar Änderungen und Erweiterungen notwendig.

MySQL-Code in Makros

Wird die interne Datenbank angesprochen, so werden die Tabellen und Felder mit doppelten Anführungszeichen gegenüber dem SQL-Code abgesetzt:

```
001 SELECT "Feld" FROM "Tabelle"
```

Da in Makros der SQL-Befehl Text darstellt, müssen die doppelten Anführungszeichen zusätzlich maskiert werden:

```
001 stSQL = "SELECT ""Feld"" FROM ""Tabelle"""
```

MySQL-Abfragen können hingegen anders maskiert werden:

```
001 SELECT `Feld` FROM `Datenbank`.`Tabelle`
```

Durch diese andere Form der Maskierung wird daraus im Makro-Code:

```
001 stSql = "SELECT `Feld` FROM `Datenbank`.`Tabelle`"
```

Temporäre Tabelle als individueller Zwischenspeicher

In den vorhergehenden Kapiteln wurde häufiger eine einzeilige Tabelle zum Suchen oder Filtern von Tabellen genutzt. In einem Mehrbenutzersystem kann darauf nicht zurückgegriffen werden, da sonst andere Nutzer von dem Filterwert eines anderen Nutzers abhängig würden. Temporäre Tabellen sind in MySQL nur für den Nutzer der gerade aktiven Verbindung zugänglich, so dass für die Such- und Filterfunktionen auf diese Tabellenform zugegriffen werden kann.

Diese Tabellen können natürlich nicht vorher erstellt worden sein. Sie müssen beim Öffnen der Base-Datei erstellt werden. Deshalb ist das folgende Makro mit dem Öffnen der *.odb-Datei zu verbinden:

```
001 SUB CreateTempTable
002    oDatenquelle = thisDatabaseDocument.CurrentController
003    IF NOT (oDatenquelle.isConnected()) THEN oDatenquelle.connect()
004    oVerbindung = oDatenquelle.ActiveConnection()
005    oSQL_Anweisung = oVerbindung.createStatement()
006    stSql = "CREATE TEMPORARY TABLE IF NOT EXISTS `Suchtmp` (`ID` INT PRIMARY KEY,
          `Name` VARCHAR(50))"
007    oSQL_Anweisung.executeUpdate(stSql)
008 END SUB
```

Zum Start der *.odb-Datei besteht noch keine Verbindung zur externen MySQL-Datenbank. Die Verbindung muss erst einmal hergestellt werden. Dann wird eine temporäre Tabelle mit entsprechend notwendigen Feldern erstellt.

Leider zeigt Base die temporären Tabellen nicht im Tabellencontainer an. Es kann über Abfragen auf diese Tabellen zugegriffen werden. Der Zugriff ist allerdings nur lesend möglich, so dass neue Inhalte für diese Tabellen nur über die direkte SQL-Eingabe oder über Makros erfolgen kann. Für einen einfachen Filterzugriff bietet sich deshalb an, statt einer temporären Tabelle eine feste Tabelle zu nutzen, in der die Filterinhalte zusammen mit der Verbindungsnummer (**CONNECTION_ID**) gespeichert werden.

Filterung über die Verbindungsnummer

Hier wird die Filtertabelle bereits vorher über die GUI erstellt. Die Tabelle wird beim Öffnen der Datenbankdatei allerdings direkt mit entsprechendem Inhalt versorgt:

```
001    stSql = "REPLACE INTO `Filter` (`Connection_ID`,`Name`)
          VALUES(CONNECTION_ID(),NULL)"
```

Die Tabelle ist jetzt auch in Formularen beschreibbar und kann entsprechend einfacher genutzt werden. Für andere Nutzer ist jetzt allerdings sichtbar, nach welchen Begriffen der einzelnen Nutzer gerade sucht. Prinzipiell lässt sich aber der entsprechende auf den einzelnen Nutzer festgelegte Datensatz immer über **CONNECTION_ID()** ermitteln.

Wird die Datenbankdatei wieder geschlossen, so kann auch die Filter-Tabelle entsprechend bereinigt werden:

```
001 SUB DeleteFilter
002    oDatasource = thisDatabaseDocument.CurrentController
003    IF NOT (oDatasource.isConnected()) THEN oDatasource.connect()
004    oConnection = oDatasource.ActiveConnection()
005    oSQL_Command = oConnection.createStatement()
006    stSql = "DELETE FROM `Filter` WHERE `Connection_ID` = CONNECTION_ID()"
007    oSQL_Command.executeUpdate(stSql)
008 END SUB
```

Gespeicherte Prozeduren

In MySQL/MariaDB können Prozeduren gespeichert werden. Sollen diese Prozeduren zu bestimmten Zeiten ablaufen, so können sie über **Extras → SQL** mit dem Befehl **CALL `Prozedurname`();** aufgerufen werden. Erstellen solche Prozeduren von sich aus eine Ergebnismenge in einer temporären Tabelle, so lässt sich diese temporäre Tabelle als nicht bearbeitbare Informationsquelle nutzen.

Automatischer Aufruf einer Prozedur

Die folgende Prozedur **AlleNamen()** könnte beim Laden eines Formulars ausgelöst werden. Sie muss ablaufen, bevor das Formular selbst Inhalt laden will. Kann das nicht erfolgen, so muss

zusätzlich auf das auslösende Formular über das Ereignis Bezug genommen werden und das Formular nach der Ausführung der Prozedur erneut geladen werden.

```
001 SUB ProcExecute
002    oDatasource = thisDatabaseDocument.CurrentController
003    IF NOT (oDatasource.isConnected()) THEN oDatasource.connect()
004    oConnection = oDatasource.ActiveConnection()
005    oSQL_Command = oConnection.createStatement()
006    oSql_Command.executeUpdate("CALL `AlleNamen`();")
007 END SUB
```

Die Prozedur ersetzt lediglich den Umweg, das Kommando **`CALL `AlleNamen`();`** über **Extras → SQL** eingeben zu müssen. Die Prozedur wird ohne Rückgabewert genutzt. Der Rückgabewert muss per SQL in der Prozedur selbst definiert sein.

Übertragung der Ausgabe einer Prozedur in eine temporäre Tabelle

Dieses Makro geht davon aus, dass die gespeicherte Prozedur von MySQL/MariaDB einen Rückgabewert hat, der aber leider nicht über eine Abfrage, sondern nur direkt über SQL auf der Konsole direkt ausgegeben wird.

```
001 SUB ProcContentShow
002    oDatasource = thisDatabaseDocument.CurrentController
003    IF NOT (oDatasource.isConnected()) THEN oDatasource.connect()
004    oConnection = oDatasource.ActiveConnection()
005    oSQL_Command = oConnection.createStatement()
006    oResult = oSql_Command.executeQuery("CALL `AlleNamen`();")
007    stFields = ""
008    FOR i = 1 TO oResult.Columns.Count
009       stFields = stFields + "`" + oResult.Columns.ElementNames(i-1) + "` TINYTEXT,"
010    NEXT
011    stFields = Left(stFields, Len(stFields)-1)
012    stProcedure = "("
013    WHILE oResult.next
014       FOR i = 1 TO oResult.Columns.Count
015          stProcedure = stProcedure + "'" + oResult.getString(i) +"',"
016       NEXT
017       stProcedure =  Left(stProcedure, Len(stProcedure)-1)
018       stProcedure = stProcedure + "),("
019    WEND
020    stProcedure = Left(stProcedure, Len(stProcedure)-2)
021    oSQL_Command.executeUpdate("DROP TEMPORARY TABLE IF EXISTS `TempNamen`")
022    oSQL_Command.executeUpdate("CREATE TEMPORARY TABLE `TempNamen` ("+stFields+")")
023    oSQL_Command.executeUpdate("INSERT INTO `TempNamen` VALUES "+stProcedure+";")
024 END SUB
```

Zuerst wird die Prozedur ausgeführt. Ein eventueller Rückgabewert wird in **`oResult`** gespeichert. Aus diesem Rückgabewert lassen sich die Spaltennamen (**`oResult.Columns.ElementNames()`**) und der Inhalt (**`oResult.getString()`**) auslesen. Die Feldtypen sind leider nicht zu ermitteln, so dass der Inhalt jeder Spalte einfach als Text interpretiert wird. Dieser Text wird als **`TINYTEXT`** mit einer Maximallänge von 255 Zeichen anschließend in einer temporären Tabelle gespeichert. Diese Tabelle kann dann zum Recherchieren genutzt werden.

PostgreSQL und Makros

Autowertrückgabe mit Returning

Existiert bei PostgreSQL ein AutoWert-Feld, so kann aus diesem Feld mit dem folgenden Befehl der gerade neu erstellte AutoWert ermittelt werden:

```
001 Sub Insert_Returning
002    DIM oDatasource AS OBJECT
```

```
003     DIM oConnection AS OBJECT
004     DIM stSql AS STRING
005     DIM oResult AS OBJECT
006     DIM loID AS LONG
007     oDatasource = thisDatabaseDocument.CurrentController
008     IF NOT (oDatasource.isConnected()) THEN oDatasource.connect()
009     oConnection = oDatasource.ActiveConnection()
010     oSQL_Statement = oConnection.createStatement()
011     stSql = "INSERT INTO ""Test1"" (""Neu"") Values('Ich') RETURNING ""ID"""
012     oResult = oSQL_Statement.executeQuery(stSql)
013     oResult.Next
014     loID = oResult.getLong(1)
015 End Sub
```

Der zurückgegebene Schlüsselwert kann nur ausgelesen werden, wenn der Datensatz über **executeQuery** eingefügt wird.

Liegt die Tabelle nicht im Schema «public», dann ist der Name des Schemas mit aufzunehmen:

```
011     stSql = "INSERT INTO ""loffice"".""Test1"" (""Neu"") Values('Ich')
           RETURNING ""ID"""
```

fügt einen Wert in eine Tabelle ein, die in dem selbst erstellten Schema «loffice» liegt.

Datentyp «Array»

Mit PostgreSQL kann einem Feld auch der Datentyp «Array» zugewiesen werden[51]. Dies geht allerdings nur über **Extras → SQL**:

```
001 CREATE TABLE "public"."tbl_Array" (
002     "ID" int4 NOT NULL,
003     "Nachname" varchar(100),
004     "Vornamen" varchar(200)[],
005     PRIMARY KEY ("ID"));
```

In das Feld "Vornamen" können jetzt über eine geschweifte Klammer, getrennt mit Kommas, Array eingegeben werden. Diese Werte können in Abfragen einzeln ausgelesen werden.

```
001 SELECT "Nachname", "Vornamen"[1] FROM "tbl_Array"
```

Dies gibt den ersten Vornamen in der Liste wieder.

Das Einfügen und auslesen von Werten bei diesen Feldern ist mit Makros etwas umständlich. Natürlich funktioniert die direkte Eingabe mit den geschweiften Klammern, aber bei **prepared Statements** hakt es:

```
001     DIM ar
002     stSql = "INSERT INTO ""public"".""tbl_Array"" (""ID"", ""Nachname"",
           ""Vornamen"") VALUES (?, ?, ?)"
003     oSQL_Statement = oConnection.prepareStatement(stSql)
004     oSQL_Statement.setLong(1, 2)
005     oSQL_Statement.setString(2, "Big")
006     ar = array("Will","John","Jack")
007     oSQL_Statement.setArray(3, ar)
```

Hier kommt es bei **setArray** direkt zum Crash: *Bug 154464*

Das Feld muss, wenn bereits ein Array vorgesehen ist, nicht über **setArray** mit Inhalt versehen werden, sondern über **setString**:

```
006     ar = array("Will","John","Jack")
007     stAr = "{"
008     FOR i = LBound(ar()) TO UBound(ar())
009        stAr = stAr & ar(i) & ","
010     NEXT
```

51 Die Arrayfunktion ist nur beim direkten Treiber so implementiert, dass die entsprechenden Werte auch in der Tabelle direkt eingebbar und sichtbar sind. Der JDBC-Treiber unterstützt Arrays nicht in gleichem Umfang.

```
011     stAr = Left(stAr,Len(stAr)-1) & "}"
012     oSQL_Statement.setString(3, stAr)
```

Beim Auslesen der Werte über eine Abfrage in Makros funktioniert die entsprechende Methode **getString** allerdings nicht. Dies würde nur für einen Wert (**"Vornamen"[1]**), nicht aber für das Array funktionieren. Stattdessen müssen die Werte über **getArray** ausgelesen werden:

```
001     DIM ar
002     DIM stAr
003     stSql = "SELECT ""ID"", ""Nachname"", ""Vornamen"" FROM
           ""public"".""tbl_Array"""
004     oResult = oSQL_Statement.executeQuery(stSql)
005     WHILE oResult.Next
006        loID = oResult.getLong(1)
007        stSurname = oResult.getString(2)
008        ar = oResult.getArray(3)
009        stAr = ar.getArray(NULL)
010        FOR i = LBound(stAr) TO UBound(stAr)
011           PRINT stAr(i)
012        NEXT
013     WEND
```

Die Werte aus dem Arrayfeld werden hier zu Demonstrationszwecken lediglich auf dem Bildschirm ausgegeben. Sie können entsprechend anderweitig umgeformt und ausgegeben werden.

Dialoge

Statt Formularen können für Base auch Dialoge zur Eingabe von Daten, zum Bearbeiten von Daten oder auch zur Wartung der Datenbank genutzt werden. Dialoge lassen sich auf das jeweilige Anwendungsgebiet direkt zuschneiden, sind aber natürlich nicht so komfortabel vordefiniert wie Formulare. Hier eine kurze Einführung, die mit einem recht komplexen Beispiel zur Datenbankwartung endet.

Dialoge starten und beenden

Zuerst muss für den Dialog[52] ein entsprechender Ordner erstellt werden. Dies geschieht über **Extras → Makros → Dialoge verwalten → Datenbankdatei → Standard → Neu**. Der Dialog erscheint mit einer grauen Fläche und einer Titelleiste sowie einem Schließkreuz. Bereits dieser leere Dialog könnte jetzt aufgerufen und über das Schließkreuz wieder geschlossen werden.

Wird der Dialog angeklickt, so gibt es bei den allgemeinen Eigenschaften die Möglichkeit, die Größe und Position einzustellen. Außerdem kann dort der Inhalt des Titels «Dialoge starten» eingegeben werden.

52 Die Beispieldatenbank «Beispiel_Dialoge.odb» zu den folgenden Kapiteln ist den Beispieldatenbanken für dieses Handbuch beigefügt.

In der am unteren Fensterrand befindlichen Symbolleiste befinden sich die verschiedensten Formular-Steuerelemente. Aus diesen Steuerelementen sind für den abgebildeten Dialog zwei Schaltflächen ausgesucht worden, von denen aus andere Dialoge gestartet werden sollen. Die Bearbeitung des Inhaltes und der Verknüpfung zu Makros ist gleich den Schaltflächen im Formular.

Die Lage der Deklaration der Variablen für den Dialog ist besonders zu beachten. Der Dialog wird als globale Variable gesetzt, damit auf ihn von unterschiedlichen Prozeduren aus zugegriffen werden kann. In diesem Falle ist der Dialog mit der Variablen oDialog0 versehen, weil es noch weitere Dialoge gibt, die einfach entsprechend durchnummeriert wurden.

```
001 DIM oDialog0 AS OBJECT
```

Zuerst wird die Bibliothek für den Dialog geladen. Sie liegt in dem Verzeichnis «Standard», sofern bei der Erstellung des Dialogs keine andere Bezeichnung gewählt wurde. Der Dialog selbst ist über den Reiter mit der Bezeichnung «Dialog0» in dieser Bibliothek erreichbar. Mit **Execute()** wird der Dialog aufgerufen.

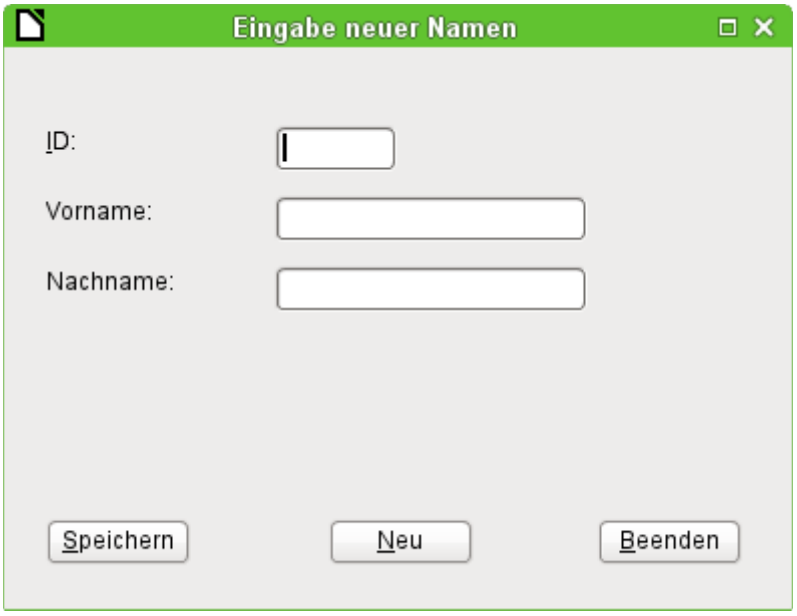
```
001 SUB Dialog0Start
002    DialogLibraries.LoadLibrary("Standard")
003    oDialog0 = createUnoDialog(DialogLibraries.Standard.Dialog0)
004    oDialog0.Execute()
005 END SUB
```

Prinzipiell kann ein Dialog durch Betätigung des Schließkreuzes geschlossen werden. Soll dafür aber ein entsprechender Button vorgesehen werden, so reicht hier einfach der Befehl **EndExecute()** innerhalb einer Prozedur.

```
001 SUB Dialog0Ende
002    oDialog0.EndExecute()
003 END SUB
```

Mit diesem Rahmen können beliebig gestaltete Dialoge gestartet und wieder geschlossen werden.

Einfacher Dialog zur Eingabe neuer Datensätze



Dieser Dialog stellt eine Vorstufe für den nächstfolgenden Dialog zur Bearbeitung von Datensätzen dar. Erst einmal sollen nur grundlegende Vorgehensweisen im Umgang mit der Tabelle einer Datenbank geklärt werden. Hier ist dies das Speichern von Datensätzen mit neuem Primärschlüssel bzw. das komplett neue Eingeben von Datensätzen. Wie weit so ein kleiner Dialog

ausreichend für eine bestimmte Datenbankaufgabe ist, hängt natürlich von den Bedürfnissen des Nutzer ab.

Mit

```
001 DIM oDialog1 AS OBJECT
```

wird wieder direkt auf der untersten Ebene des Moduls vor allen Prozeduren die globale Variable für den Dialog erstellt.

Der Dialog wird genauso gestartet und beendet wie der vorhergehende Dialog. Lediglich die Bezeichnung ändert sich von Dialog0 auf Dialog1. Die Prozedur zum Beenden des Dialogs ist mit dem Button Beenden verbunden.

Über den Button Neu werden alle Kontrollfelder des Dialogs durch die Prozedur «Datenfelder-Leeren» von vorhergehenden Eingaben befreit:

```
001 SUB DatenfelderLeeren
002    oDialog1.getControl("NumericField1").Text = ""
003    oDialog1.getControl("TextField1").Text = ""
004    oDialog1.getControl("TextField2").Text = ""
005 END SUB
```

Jedes Feld, das in einen Dialog eingefügt wird, ist über einen eigenen Namen ansprechbar. Im Gegensatz zu Feldern eines Formulars wird hier durch die Benutzeroberfläche darauf geachtet, dass keine Namen doppelt vergeben werden.

Über **getControl** wird zusammen mit dem Namen auf ein Kontrollfeld zugegriffen. Auch ein numerisches Feld hat hier die Eigenschaft **Text** zur Verfügung. Nur so lässt sich schließlich ein numerisches Feld leeren. Einen leeren Text gibt es, eine leere Nummer hingegen nicht – stattdessen müsste 0 in das Feld für den Primärschlüssel geschrieben werden.

Der Button Speichern löst schließlich die Prozedur «Daten1Speichern» aus:

```
001 SUB Daten1Speichern
002    DIM oDatenquelle AS OBJECT
003    DIM oVerbindung AS OBJECT
004    DIM oSQL_Anweisung AS OBJECT
005    DIM loID AS LONG
006    DIM stVorname AS STRING
007    DIM stNachname AS STRING
008    loID = oDialog1.getControl("NumericField1").Value
009    stVorname = oDialog1.getControl("TextField1").Text
010    stNachname = oDialog1.getControl("TextField2").Text
011    IF loID > 0 AND stNachname <> "" THEN
012       oDatenquelle = thisDatabaseDocument.CurrentController
013       If NOT (oDatenquelle.isConnected()) THEN
014          oDatenquelle.connect()
015       END IF
016       oVerbindung = oDatenquelle.ActiveConnection()
017       oSQL_Anweisung = oVerbindung.createStatement()
018       stSql = "SELECT ""ID"" FROM ""Name"" WHERE ""ID"" = '"+loID+"'"
019       oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
020       WHILE oAbfrageergebnis.next
021          MsgBox ("Der Wert für das Feld 'ID' existiert schon",16,
                "Doppelte Dateneingabe")
022          EXIT SUB
023       WEND
024       stSql = "INSERT INTO ""Name"" (""ID"", ""Vorname"", ""Nachname"")
             VALUES ('"+loID+"','"+stVorname+"','"+stNachname+"')"
025       oSQL_Anweisung.executeUpdate(stSql)
026       DatenfelderLeeren
027    END IF
028 END SUB
```

Wie in der Prozedur «DatenfelderLeeren» wird auf die Eingabefelder zugegriffen. Dieses Mal erfolgt der Zugriff allerdings lesend, nicht schreibend. Nur wenn das Feld «ID» eine Eingabe größer als 0 enthält und in dem Feld für den Nachnamen auch Text steht, soll der Datensatz

weitergegeben werden. Die Null muss alleine schon deshalb ausgeschlossen werden, weil eine Zahlenvariable für Zahlen ohne Nachkommastellen grundsätzlich mit dem Wert 0 initialisiert wird. Auch bei einem leeren Feld würde also schließlich 0 zur Speicherung weitergegeben.

Sind die beiden Felder entsprechend mit Inhalt versehen, so wird eine Verbindung zur Datenbank aufgenommen. Da sich die Kontrollfelder nicht in einem Formular befinden, muss die Datenbankverbindung über **`thisDatabaseDocument.CurrentController`** hergestellt werden.

Zuerst wird eine Abfrage an die Datenbank geschickt, ob vielleicht ein Datensatz mit dem vorgegebenen Primärschlüssel schon existiert. Hat die Abfrage Erfolg, so wird eine Meldung über eine Messagebox ausgegeben, die mit einem Stopp-Symbol versehen ist (Code: **16**) und die Überschrift «Doppelte Dateneingabe» trägt. Danach wird durch **`Exit SUB`** die Prozedur beendet.

Hat die Abfrage keinen Datensatz gefunden, der den gleichen Primärschlüssel hat, so wird der neue Datensatz über den Insert-Befehl in die Datenbank eingefügt. Anschließend wird über die Prozedur «DatenfelderLeeren» wieder ein leeres Formular präsentiert.

Dialog zum Bearbeiten von Daten in einer Tabelle

Dieser Dialog stellt schon deutlich mehr Möglichkeiten zur Verfügung als der vorhergehende Dialog. Hier lassen sich alle Datensätze anzeigen, durch Datensätze navigieren, Datensätze neu erstellen, ändern oder auch löschen. Natürlich ist der Code entsprechend umfangreicher.

Der Button Beenden ist mit der entsprechend auf den Dialog2 abgewandelten Prozedur des vorhergehenden Dialogs zur Eingabe neuer Datensätze verbunden. Hier werden nur die weiteren Buttons mit ihren entsprechenden Funktionen beschrieben.

Die Dateneingabe ist im Dialog so beschränkt, dass im Feld «ID» der Mindestwert auf '1' eingestellt wurde. Diese Einschränkung hat mit dem Umgang mit Variablen in Basic zu tun: Zahlenvariablen werden bei der Definition bereits mit '0' als Grundwert vorbelegt. Werden Zahlenwerte von leeren Feldern und Feldern mit '0' ausgelesen, so ist für Basic der anschließende Inhalt der Variablen gleich. Es müsste bei der Nutzung von '0' im Feld «ID» also zur Unterscheidung erst Text ausgelesen und vielleicht später in eine Zahl umgewandelt werden.

Der Dialog wird unter den gleichen Voraussetzungen geladen wie vorher auch. Hier wird allerdings die Ladeprozedur davon abhängig gemacht, ob die Variable, die der Prozedur «DatenLaden» mitgegeben wird, 0 ist.

```
001  SUB DatenLaden(loID AS LONG)
002     DIM oDatenquelle AS OBJECT
```

```
003     DIM oVerbindung AS OBJECT
004     DIM oSQL_Anweisung AS OBJECT
005     DIM stVorname AS STRING
006     DIM stNachname AS STRING
007     DIM loRow AS LONG
008     DIM loRowMax AS LONG
009     DIM inStart AS INTEGER
010     oDatenquelle = thisDatabaseDocument.CurrentController
011     If NOT (oDatenquelle.isConnected()) THEN
012        oDatenquelle.connect()
013     END IF
014     oVerbindung = oDatenquelle.ActiveConnection()
015     oSQL_Anweisung = oVerbindung.createStatement()
016     IF loID < 1 THEN
017        stSql = "SELECT MIN(""ID"") FROM ""Name"""
018        oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
019        WHILE oAbfrageergebnis.next
020           loID = oAbfrageergebnis.getInt(1)
021        WEND
022        inStart = 1
023     END IF
```

Die Variablen werden deklariert. Die Datenbankverbindung wird, wie weiter oben erklärt, für den Dialog hergestellt. Zum Start ist **loID 0**. Für diesen Fall wird per SQL der niedrigste Wert für den Primärschlüssel ermittelt. Der entsprechende Datensatz soll in dem Dialog später angezeigt werden. Gleichzeitig wird die Variable **inStart** auf 1 gestellt, damit der Dialog später gestartet wird. Enthält die Tabelle keine Daten, so bleibt **loID 0**. Entsprechend muss auch nicht nach dem Inhalt und der Anzahl irgendwelcher Datensätze im Folgenden gesucht werden.

Nur wenn **loID** größer als 0 ist, wird zuerst mit einer Abfrage überprüft, welche Daten in dem Datensatz enthalten sind. Anschließend werden in einer zweiten Abfrage alle Datensätze für die Datensatzanzeige gezählt. Mit der dritten Abfrage wird die Position des aktuellen Datensatzes ermittelt, indem alle Datensätze, die einen kleineren oder eben den aktuellen Primärschlüssel haben, zusammengezählt werden.

```
024     IF loID > 0 THEN
025        stSql = "SELECT * FROM ""Name"" WHERE ""ID"" = '"+loID+"'"
026        oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
027        WHILE oAbfrageergebnis.next
028           loID = oAbfrageergebnis.getInt(1)
029           stVorname = oAbfrageergebnis.getString(2)
030           stNachname = oAbfrageergebnis.getString(3)
031        WEND
032        stSql = "SELECT COUNT(""ID"") FROM ""Name"""
033        oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
034        WHILE oAbfrageergebnis.next
035           loRowMax = oAbfrageergebnis.getInt(1)
036        WEND
037        stSql = "SELECT COUNT(""ID"") FROM ""Name"" WHERE ""ID"" <= '"+loID+"'"
038        oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
039        WHILE oAbfrageergebnis.next
040           loRow = oAbfrageergebnis.getInt(1)
041        WEND
042        oDialog2.getControl("NumericField1").Value = loID
043        oDialog2.getControl("TextField1").Text = stVorname
044        oDialog2.getControl("TextField2").Text = stNachname
045     END IF
046     oDialog2.getControl("NumericField2").Value = loRow
047     oDialog2.getControl("NumericField3").Value = loRowMax
048     IF loRow = 1 THEN
049        ' Vorheriger Datensatz
050        oDialog2.getControl("CommandButton4").Model.enabled = False
051     ELSE
052        oDialog2.getControl("CommandButton4").Model.enabled = True
053     END IF
054     IF loRow <= loRowMax THEN
```

```
055         ' Nächster Datensatz | Neuer Datensatz | Löschen
056         oDialog2.getControl("CommandButton5").Model.enabled = True
057         oDialog2.getControl("CommandButton2").Model.enabled = True
058         oDialog2.getControl("CommandButton6").Model.enabled = True
059      ELSE
060         oDialog2.getControl("CommandButton5").Model.enabled = False
061         oDialog2.getControl("CommandButton2").Model.enabled = False
062         oDialog2.getControl("CommandButton6").Model.enabled = False
063      END IF
064      IF inStart = 1 THEN
065         oDialog2.Execute()
066      END IF
067   END SUB
```

Die ermittelten Werte für die Formularfelder werden übertragen. Die Einträge für die Nummer des aktuellen Datensatzes sowie die Anzahl aller Datensätze werden auf jeden Fall mit einer Zahl versorgt. Ist kein Datensatz vorhanden, so wird hier über den Default-Wert für eine numerische Variable 0 eingefügt.

Die Buttons zum Navigieren > («CommandButton5») und < («CommandButton4») sind nur verfügbar, wenn es möglich ist, einen entsprechenden Datensatz über die Navigation zu erreichen. Ansonsten werden sie vorübergehend mit **enabled = False** deaktiviert. Gleiches gilt für die Buttons Neu und Löschen. Sie sollen dann nicht verfügbar sein, wenn die Zahl der angezeigten Zeilen höher ist als die maximal ermittelte Zeilenzahl. Dies ist für die Eingabe neuer Datensätze die Standardeinstellung dieses Dialogs.

Der Dialog soll möglichst nur dann gestartet werden, wenn er direkt aus einer Startdatei über **DatenLaden(0)** erstellt werden soll. Deshalb wurde die gesonderte Variable **inStart** mit dem Wert 1 zu Beginn der Prozedur versehen..

Über den Button < soll zu dem vorhergehenden Datensatz navigiert werden können. Der Button ist nur dann aktiv, wenn nicht bereits der erste Datensatz angezeigt wird. Zum Navigieren wird von dem aktuellen Datensatz der Wert für den Primärschlüssel aus dem Feld «NumericField1» ausgelesen.

Hier gilt es zwei Fälle zu unterscheiden:

1. Es wurde vorher vorwärts zu einer Neueingabe navigiert, so dass das entsprechende Feld keinen Wert enthält. **loID** gibt dann den Standardwert wieder, der durch die Definition als Zahlenvariable vorgegeben ist: 0.
2. Ansonsten enthält loID einen Wert, der größer als 0 ist. Entsprechend kann über eine Abfrage die nächstkleinere «ID» ermittelt werden.

```
001   SUB vorherigerDatensatz
002      DIM loID AS LONG
003      DIM loIDneu AS LONG
004      loID = oDialog2.getControl("NumericField1").Value
005      oDatenquelle = thisDatabaseDocument.CurrentController
006      If NOT (oDatenquelle.isConnected()) THEN
007         oDatenquelle.connect()
008      END IF
009      oVerbindung = oDatenquelle.ActiveConnection()
010      oSQL_Anweisung = oVerbindung.createStatement()
011      IF loID < 1 THEN
012         stSql = "SELECT MAX(""ID"") FROM ""Name"""
013      ELSE
014         stSql = "SELECT MAX(""ID"") FROM ""Name"" WHERE ""ID"" < '"+loID+"'"
015      END IF
016      oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
017      WHILE oAbfrageergebnis.next
018         loIDneu = oAbfrageergebnis.getInt(1)
019      WEND
020      IF loIDneu > 0 THEN
021         DatenLaden(loIDneu)
022      END IF
023   END SUB
```

Bei einem leeren «ID»-Feld soll auf den Datensatz mit dem höchsten Wert in der Primärschlüsselnummer gewechselt werden. Können hingegen aus dem «ID»-Feld Daten entnommen werden, so wird der entsprechend nachrangige Wert für die "ID" ermittelt.

Das Ergebnis dieser Abfrage dient dazu, die Prozedur «DatenLaden» mit dem entsprechenden Schlüsselwert erneut durchlaufen zu lassen.

Über den Button > wird zum nächsten Datensatz navigiert. Diese Navigationsmöglichkeit steht nur zur Verfügung, wenn nicht bereits der Dialog für die Eingabe eines neuen Datensatzes geleert wurde. Dies ist natürlich auch beim Start und leerer Tabelle der Fall.

Zwangsläufig ist in dem Feld «NumericField1» ein Wert vorhanden. Von diesem Wert ausgehend kann also per SQL nachgesehen werden, welcher Primärschlüsselwert der nächsthöhere in der Tabelle ist. Bleibt die Abfrage leer, weil es keinen entsprechenden Datensatz gibt, so ist der Wert für **`loIDneu = 0`**. Ansonsten kann über die Prozedur «DatenLaden» der Inhalt des nächsten Datensatzes geladen werden.

```
001 SUB naechsterDatensatz
002    DIM loID AS LONG
003    DIM loIDneu AS LONG
004    loID = oDialog2.getControl("NumericField1").Value
005    oDatenquelle = thisDatabaseDocument.CurrentController
006    If NOT (oDatenquelle.isConnected()) THEN
007       oDatenquelle.connect()
008    END IF
009    oVerbindung = oDatenquelle.ActiveConnection()
010    oSQL_Anweisung = oVerbindung.createStatement()
011    stSql = "SELECT MIN(""ID"") FROM ""Name"" WHERE ""ID"" > '"+loID+"'"
012    oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
013    WHILE oAbfrageergebnis.next
014       loIDneu = oAbfrageergebnis.getInt(1)
015    WEND
016    IF loIDneu > 0 THEN
017       DatenLaden(loIDneu)
018    ELSE
019       Datenfelder2Leeren
020    END IF
021 END SUB
```

Existiert beim Navigieren zum nächsten Datensatz kein weiterer Datensatz, so löst die Navigation die folgende Prozedur «Datenfelder2Leeren» aus, die zur Eingabe neuer Daten dient.

Mit der Prozedur «Datenfelder2Leeren» werden nicht nur die Datenfelder selbst geleert. Die Position des aktuellen Datensatzes wird um einen Datensatz höher als die maximale Datensatzzahl eingestellt. Das soll verdeutlichen, dass der aktuell bearbeitete Datensatz noch nicht in der Datenbank enthalten ist.

Sobald «Datenfelder2Leeren» ausgelöst wird, wird außerdem die Möglichkeit des Sprungs zum vorhergehenden Datensatz aktiviert. Sprünge zu einem nachfolgenden Datensatz, das erneute Aufrufen der Prozedur über Neu oder das Löschen sind deaktiviert.

```
001 SUB Datenfelder2Leeren
002    loRowMax = oDialog2.getControl("NumericField3").Value
003    oDialog2.getControl("NumericField1").Text = ""
004    oDialog2.getControl("TextField1").Text = ""
005    oDialog2.getControl("TextField2").Text = ""
006    oDialog2.getControl("NumericField2").Value = loRowMax + 1
007    oDialog2.getControl("CommandButton4").Model.enabled = True   ' Vorh. Datensatz
008    oDialog2.getControl("CommandButton5").Model.enabled = False  ' Nächster Datens.
009    oDialog2.getControl("CommandButton2").Model.enabled = False  ' Neuer Datensatz
010    oDialog2.getControl("CommandButton6").Model.enabled = False  ' Löschen
011 END SUB
```

Das Speichern der Daten soll nur möglich sein, wenn in den Feldern für «ID» und «Nachname» ein Eintrag erfolgt ist. Ist diese Bedingung erfüllt, so wird überprüft, ob der Datensatz ein neuer Datensatz ist. Das funktioniert über den Datensatzanzeiger, der bei neuen Datensätzen so

eingestellt wurde, dass er für den aktuellen Datensatz einen um 1 höheren Wert als den maximalen Wert an Datensätzen ausgibt.

Im Falle eines neuen Datensatzes gibt es weiteren Überprüfungsbedarf, damit eine Speicherung einwandfrei funktionieren kann. Kommt die Ziffer für den Primärschlüssel bereits einmal vor, so erfolgt eine Warnung. Wird die entsprechende Frage mit Ja bestätigt, so wird der alte Datensatz mit der gleichen Schlüsselnummer überschrieben. Ansonsten erfolgt keine Speicherung. Solange noch gar kein Datensatz in der Datenbank enthalten ist (**loRowMax = 0**) braucht diese Überprüfung nicht zu erfolgen. In dem Falle kann der Datensatz direkt als neuer Datensatz abgespeichert werden. Bei einem neuen Datensatz wird schließlich noch die Zahl der Datensätze um 1 erhöht und die Eingabe für den nächsten Datensatz frei gemacht.

Bei bestehenden Datensätzen wird einfach der alte Datensatz durch ein Update mit dem neuen Datensatz überschrieben.

```
001 SUB Daten2Speichern(oEvent AS OBJECT)
002    DIM oDatenquelle AS OBJECT
003    DIM oVerbindung AS OBJECT
004    DIM oSQL_Anweisung AS OBJECT
005    DIM oDlg AS OBJECT
006    DIM loID AS LONG
007    DIM stVorname AS STRING
008    DIM stNachname AS STRING
009    DIM inMsg AS INTEGER
010    DIM loRow AS LONG
011    DIM loRowMax AS LONG
012    DIM stSql AS STRING
013    oDlg = oEvent.Source.getContext()
014    loID = oDlg.getControl("NumericField1").Value
015    stVorname = oDlg.getControl("TextField1").Text
016    stNachname = oDlg.getControl("TextField2").Text
017    IF loID > 0 AND stNachname <> "" THEN
018       oDatenquelle = thisDatabaseDocument.CurrentController
019       If NOT (oDatenquelle.isConnected()) THEN
020          oDatenquelle.connect()
021       END IF
022       oVerbindung = oDatenquelle.ActiveConnection()
023       oSQL_Anweisung = oVerbindung.createStatement()
024       loRow = oDlg.getControl("NumericField2").Value
025       loRowMax = oDlg.getControl("NumericField3").Value
026       IF loRowMax < loRow THEN
027          IF loRowMax > 0 THEN
028             stSql = "SELECT ""ID"" FROM ""Name"" WHERE ""ID"" = '"+loID+"'"
029             oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
030             WHILE oAbfrageergebnis.next
031                inMsg = MsgBox ("Der Wert für das Feld 'ID' existiert schon." &
                      CHR(13) & "Soll der Datensatz überschrieben werden?",20,
                      "Doppelte Dateneingabe")
032                IF inMsg = 6 THEN
033                   stSql = "UPDATE ""Name"" SET ""Vorname""='"+stVorname+"',
                         ""Nachname""='"+stNachname+"' WHERE ""ID"" = '"+loID+"'"
034                   oSQL_Anweisung.executeUpdate(stSql)
035                   DatenLaden(loID) ' Beim Update wurde ein bestehender Datensatz
                         überschrieben. Neueinlasen zur Korrektur der Datensatzzahlen
036                END IF
037                EXIT SUB
038             WEND
039          END IF
040          stSql = "INSERT INTO ""Name"" (""ID"", ""Vorname"", ""Nachname"") VALUES
                ('"+loID+"','"+stVorname+"','"+stNachname+"')"
041          oSQL_Anweisung.executeUpdate(stSql)
042          oDlg.getControl("NumericField3").Value = loRowMax + 1
                ' Nach dem Insert existiert ein Datensatz mehr
043          Datenfelder2Leeren
                ' Nach einem Insert wird grundsätzlich zum nächsten Insert geschaltet
```

```
044         ELSE
045            stSql = "UPDATE ""Name"" SET ""Vorname""='"+stVorname+"',
                  ""Nachname""='"+stNachname+"' WHERE ""ID"" = '"+loID+"'"
046            oSQL_Anweisung.executeUpdate(stSql)
047         END IF
048      END IF
049   END SUB
```

Die Löschprozedur ist mit einer Nachfrage versehen, die versehentliches Löschen verhindern soll. Dadurch, dass der Button deaktiviert wird, wenn die Eingabefelder leer sind, dürfte es nicht vorkommen, dass das Feld «NumericField1» leer ist. Deshalb könnte die Überprüfung der Bedingung **`IF loID > 0`** auch entfallen.

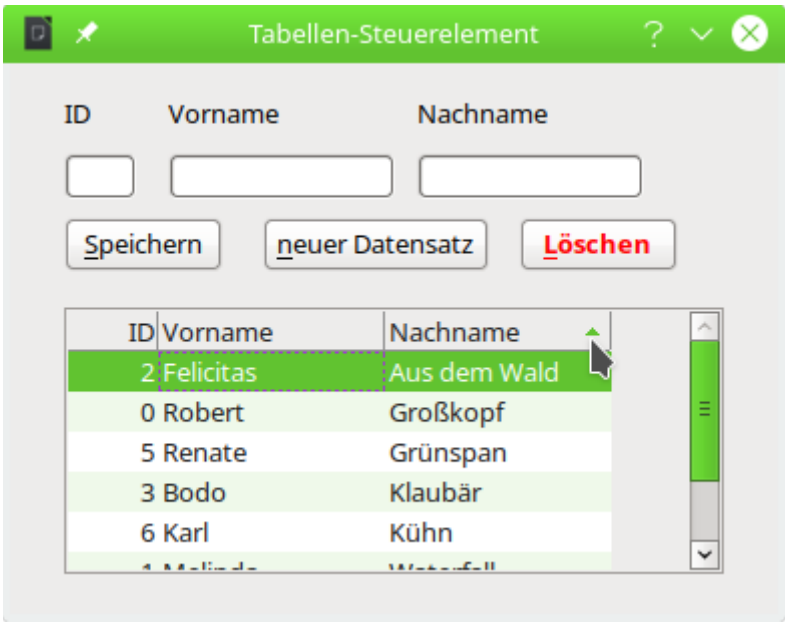
Beim Löschen wird die Zahl der Datensätze um einen Datensatz herabgesetzt. Dies muss entsprechend mit **`loRowMax – 1`** korrigiert werden. Anschließend wird der dem aktuellen Datensatz folgende Datensatz angezeigt.

```
001   SUB DatenLoeschen(oEvent AS OBJECT)
002      DIM oDatenquelle AS OBJECT
003      DIM oVerbindung AS OBJECT
004      DIM oSQL_Anweisung AS OBJECT
005      DIM oDlg AS OBJECT
006      DIM loID AS LONG
007      oDlg = oEvent.Source.getContext()
008      loID = oDlg.getControl("NumericField1").Value
009      IF loID > 0 THEN
010         inMsg = MsgBox ("Soll der Datensatz wirklich gelöscht werden?",20,
            "Löschen eines Datensatzes")
011         IF inMsg = 6 THEN
012            oDatenquelle = thisDatabaseDocument.CurrentController
013            If NOT (oDatenquelle.isConnected()) THEN
014               oDatenquelle.connect()
015            END IF
016            oVerbindung = oDatenquelle.ActiveConnection()
017            oSQL_Anweisung = oVerbindung.createStatement()
018            stSql = "DELETE FROM ""Name"" WHERE ""ID"" = '"+loID+"'"
019            oSQL_Anweisung.executeUpdate(stSql)
020            loRowMax = oDlg.getControl("NumericField3").Value
021            oDlg.getControl("NumericField3").Value = loRowMax - 1
022            naechsterDatensatz
023         END IF
024      ELSE
025         MsgBox ("Kein Datensatz gelöscht." & CHR(13) &
            "Es fehlt eine Datensatzauswahl.",64,"Löschung nicht möglich")
026      END IF
027   END SUB
```

Bereits dieser kleine Dialog zur Bearbeitung von Daten zeigt, dass der Aufwand im Makrocode schon erheblich ist, um die Grundlagen einer Datenbearbeitung zu gewährleisten. Der Zugriff über ein Formular ist hier erheblich einfacher. Der Dialog kann dagegen recht flexibel an die Bedürfnisse des Programms angepasst werden. Nur ist das eben nicht für die Erstellung einer Datenbankbedienung im Schnellverfahren gedacht.

Dialog zum Bearbeiten von Daten aus einer Tabellenübersicht

Zusammen mit dem Tabellen-Steuerelement der Dialoge ist es möglich, aus einer vorhandenen Datenmenge Datensätze auszuwählen und zu bearbeiten, neue Datensätze einzufügen oder auch vorhandene Daten zu löschen. Das Tabellen-Steuerelement dient dabei zur Auswahl der Datensätze. Die Bearbeitung erfolgt wie bei den vorhergehenden Dialogen über einfache Formularfelder. Um die Verwaltung der Daten einfacher zu machen ist bei der verwendeten Tabelle ein automatisch erstellter Primärschlüssel verwendet worden.



Das Tabellen-Steuerelement bietet neben den in wechselnden Farben erscheinenden Tabellenzeilen auch die Möglichkeit, die Daten nach den Tabellenköpfen zu sortieren. Im Screenshot ist die aktuelle Sortierung für das Feld «Nachname» durch ein kleines grünes Dreieck zu erkennen.

```
001 DIM oDialog3 AS OBJECT
```

```
001 SUB Dialog3Start
002    DialogLibraries.LoadLibrary("Standard")
003    oDialog3 = createUnoDialog(DialogLibraries.Standard.Dialog3)
004    GridDatenZeigen
```

```
005     oDialog3.Execute()
006 END SUB
```

```
001 SUB Dialog3Ende
002     oDialog3.EndExecute()
003 END SUB
```

Wie bei den anderen Dialogen muss die Variable für den Dialog außerhalb der Prozeduren notiert werden, damit sie in dem gesamten Modul verfügbar ist. Über «Dialog3Start» wird der Dialog gestartet. Wichtig ist für das Tabellen-Kontrollfeld, dass die Prozedur «GridDatenZeigen» vor der Ausführung des Dialogs abläuft. Sonst bleibt die Tabelle leer.

Die Prozedur zum Beenden des Dialogs ist nur notwendig, wenn ein Button zum Beenden mit eingebaut werden soll. Das Schließen des Dialogs über das X erfolgt unabhängig von der Prozedur.

```
001 SUB GridDatenZeigen
002     DIM oGrid AS OBJECT
003     DIM oColumnModel AS OBJECT
004     DIM oColumn1 AS OBJECT
005     DIM oColumn2 AS OBJECT
006     DIM oColumn3 AS OBJECT
007     DIM oDataModel AS OBJECT
008     DIM oDatenquelle AS OBJECT
009     DIM oVerbindung AS OBJECT
010     DIM oSQL_Anweisung AS OBJECT
011     DIM stSql AS STRING
012     DIM oAbfrageergebnis AS OBJECT
013     DIM l AS LONG
014     DIM stID AS STRING
015     DIM stVorname AS STRING
016     DIM stNachname AS STRING
017     oGrid = oDialog3.Model.getByName("GridControl1")
018     oColumnModel = oGrid.ColumnModel
```

Nach Deklaration der Variablen wird auf die Tabelle Zugegriffen. Zuerst werden die Spalten der Tabelle erstellt. Die folgenden Einstellungen sind für jede Spalte notwendig. Sie können natürlich auch platzsparender über Arrays erzeugt werden.

```
019     oColumn1 = createUnoService("com.sun.star.awt.grid.GridColumn")
020     oColumn1.Title = "ID"
021     oColumn1.ColumnWidth = 20
022     oColumn1.HorizontalAlign = 2
023     oColumn1.Flexibility = False
```

Die Ausrichtung in **HorizontalAlign** wird über die Zuweisung von Werten geregelt. '0' steht für linksbündig, '1' für zentriert und '2' für rechtsbündig. Wird die **Flexibility** nicht auf **False** gesetzt, dann wird die Spaltenbreite von **ColumnWidth** nicht richtig übertragen. Die erste Spalte wird durch die Automatik dann viel zu breit.

```
024     oColumn2 = createUnoService("com.sun.star.awt.grid.GridColumn")
025     oColumn2.Title = "Vorname"
026     oColumn2.ColumnWidth = 50
027     oColumn2.HorizontalAlign = 0
028     oColumn2.Flexibility = False
029     oColumn3 = createUnoService("com.sun.star.awt.grid.GridColumn")
030     oColumn3.Title = "Nachname"
031     oColumn3.ColumnWidth = 50
032     oColumn3.HorizontalAlign = 0
033     oColumn3.Flexibility = False
034     oColumnModel.AddColumn(oColumn1)
035     oColumnModel.AddColumn(oColumn2)
036     oColumnModel.AddColumn(oColumn3)
```

Nachdem die Spalten mit den Benennungen erstellt wurden, werden die Daten hinzugefügt. Dazu wird zuerst die Datenbankverbindung überprüft und gegebenenfalls erzeugt. Die gesamten Daten werden abgefragt und Datensatz für Datensatz über **addRow** hinzugefügt. In der

Klammer von **addRow** steht zuerst die Datensatznummer und danach ein Array mit den Inhalten des Datensatzes.

```
037     oDataModel = oGrid.GridDataModel
038     oDatenquelle = thisDatabaseDocument.CurrentController
039     If NOT (oDatenquelle.isConnected()) THEN
040        oDatenquelle.connect()
041     END IF
042     oVerbindung = oDatenquelle.ActiveConnection()
043     oSQL_Anweisung = oVerbindung.createStatement()
044     stSql = "SELECT * FROM ""Name_ID_Autowert"" "
045     oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
046     l = 1
047     WHILE oAbfrageergebnis.next
048        stID = oAbfrageergebnis.getString(1)
049        stVorname = oAbfrageergebnis.getString(2)
050        stNachname = oAbfrageergebnis.getString(3)
051        oDataModel.addRow (l, Array(stID, stVorname, stNachname))
052        l = l + 1
053     WEND
054 END SUB
```

Mit der Prozedur **GridRow** werden die Daten aus dem markierten Datensatz in die Formularfelder oberhalb der Tabelle übertragen. Dadurch können dann Daten geändert oder gelöscht werden.

```
001 SUB GridRow(oEvent AS OBJECT)
002     DIM oGrid AS OBJECT
003     DIM loRow AS LONG
004     DIM oDataModel AS OBJECT
005     DIM stData AS STRING
006     DIM oDatenquelle AS OBJECT
007     DIM oVerbindung AS OBJECT
008     DIM oSQL_Anweisung AS OBJECT
009     DIM stSql AS STRING
010     DIM oAbfrageergebnis AS OBJECT
011     DIM loID AS LONG
012     DIM stVorname AS STRING
013     DIM stNachname AS STRING
014     oGrid = oEvent.Source
015     IF oGrid.hasSelectedRows THEN
```

Die Klicks auf das Tabellen-Kontrollfeld lösen dieses Makro aus. Sie landen nicht unbedingt auf irgendeinem Datensatz sondern z.B. auch auf den Tabellenköpfen. Wenn der Hintergrund oder die Tabellenköpfe angeklickt wurden können keine Daten ausgelesen werden. Deswegen muss erst einmal klar sein, ob eine Zeile ausgewählt wurde.

Aus dem markierten Datensatz wird das erste Feld '0' über **getCellData** ausgewählt. In ihm ist in diesem Falle der Primärschlüssel gespeichert. Dadurch kann der Datensatz eindeutig erkannt werden. Neben den Inhalten aus der Datenbank wird auch die Zeilennummer aus dem Tabellen-Kontrollfeld ausgelesen und in einem versteckten Formularfeld zwischengespeichert.

```
016        loRow = oGrid.CurrentRow()
017        oDataModel = oGrid.Model.GridDataModel
018        stData = oDataModel.getCellData(0, loRow)
019        oDatenquelle = thisDatabaseDocument.CurrentController
020        If NOT (oDatenquelle.isConnected()) THEN
021           oDatenquelle.connect()
022        END IF
023        oVerbindung = oDatenquelle.ActiveConnection()
024        oSQL_Anweisung = oVerbindung.createStatement()
025        stSql = "SELECT * FROM ""Name_ID_Autowert"" WHERE ""ID"" = '"+stData+"'"
026        oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
027        WHILE oAbfrageergebnis.next
028           loID = oAbfrageergebnis.getLong(1)
029           stVorname = oAbfrageergebnis.getString(2)
030           stNachname = oAbfrageergebnis.getString(3)
```

```
031         WEND
032         oDialog3.getControl("NumericField1").Value = loID
033         oDialog3.getControl("TextField1").Text = stVorname
034         oDialog3.getControl("TextField2").Text = stNachname
035         oDialog3.getControl("Zeilennummer").Value = loRow
036      END IF
037   END SUB
```

Wird der Speicher-Button betätigt, so läuft die folgende Prozedur ab.

```
001   SUB GridDatenSpeichern
002      DIM stID AS STRING
003      DIM stVorname AS STRING
004      DIM stNachname AS STRING
005      DIM loRow AS LONG
006      DIM oDatenquelle AS OBJECT
007      DIM oVerbindung AS OBJECT
008      DIM oSQL_Anweisung AS OBJECT
009      DIM stSql AS STRING
010      DIM oAbfrageergebnis AS OBJECT
011      DIM oGridData AS OBJECT
012      DIM l AS LONG
013      stID = oDialog3.getControl("NumericField1").Text
014      stVorname = oDialog3.getControl("TextField1").Text
015      stNachname = oDialog3.getControl("TextField2").Text
016      loRow = oDialog3.getControl("Zeilennummer").Value
```

Die Einträge aus den Formularfeldern werden ausgelesen. Dabei erfolgt das Auslesen auch des numerischen Feldes für die 'ID' als Text. So kann auf ein leeres Feld anschließend besser überprüft werden. Bei einem leeren Feld für die 'ID' handelt es sich um einen neuen Datensatz. Bei einem belegten Feld um eine Änderung des Datensatzes.

```
017      oDatenquelle = thisDatabaseDocument.CurrentController
018      If NOT (oDatenquelle.isConnected()) THEN
019         oDatenquelle.connect()
020      END IF
021      oVerbindung = oDatenquelle.ActiveConnection()
022      oSQL_Anweisung = oVerbindung.createStatement()
023      IF stID <> "" THEN
024         stSql = "UPDATE ""Name_ID_Autowert"" SET ""Vorname"" = '"+stVorname+"',
            ""Nachname"" = '"+stNachname+"' WHERE ID = '"+stID+"'"
025      ELSE
026         stSql = "INSERT INTO ""Name_ID_Autowert"" (""Vorname"", ""Nachname"") VALUES
            ('"+stVorname+"', '"+stNachname+"')"
027      END IF
028      oSQL_Anweisung.executeUpdate(stSql)
```

Bei Firebird muss der Code für den INSERT angepasst werden. Statt der einfachen INSERT-Anweisung ist der folgende Weg nötig:

```
026         stSql = "SELECT NEXT VALUE FOR RDB$1 FROM RDB$DATABASE"
            oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
            oAbfrageergebnis.next
            loID = oAbfrageergebnis.getLong(1)
            stSql = "INSERT INTO ""Name_ID_Autowert"" (""ID"", ""Vorname"", ""Nachname"")
               VALUES ('"+stID+"','"+stVorname+"', '"+stNachname+"')"
```

Der Name des Generator für die ID wird in einer Abfrage mit direktem SQL über

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

ermittelt. Über den Namen des Generators für die ID wird dann der nächste freie Wert des Generators abgefragt und in der Variablen **loID** zwischengespeichert. Dieser Wert ist damit vergeben und kann in dem folgenden **INSERT** für die "ID" genutzt werden. Dies ist zur Zeit die sicherste Variante, da **RETURNING** nicht funktioniert und die anschließende Abfrage des höchsten Wertes für "ID" nur dann sicher ist, wenn der Generator nicht zurückgesetzt wurde und das

System kein Mehrbenutzersystem ist. In Mehrbenutzersystemen würde gegebenenfalls sonst die "ID" eines anderen eingefügten Datensatzes abgefragt.

Nach der Datensatzänderung oder dem Einfügen eines neuen Datensatzes muss auch das Tabellen-Kontrollfeld mit den neuen Daten versorgt werden. Bei der Änderung müssen die Felder über **updateCellData** mit den neuen Daten versorgt werden. Die erste Zahl in der Klammer steht hier für die Spalte, die zweite Zahl in der Klammer für den Datensatz.

```
029     oGridData = oDialog3.Model.getByName("GridControl1").GridDataModel
030     IF stID <> "" THEN
031        oGridData.updateCellData(1,loRow,stVorname)
032        oGridData.updateCellData(2,loRow,stNachname)
033     ELSE
```

Beim Einfügen eines neuen Datensatzes muss zuerst ermittelt werden, wie der automatisch erstellte Primärschlüsselwert lautet. In diesem Fall wird gleich der gesamte Datensatz noch einmal eingelesen, so dass in dem Tabellen-Kontrollfeld auf jeden Fall das steht, was auch in der Datenbank erscheint.

Über **addRow** wird der Datensatz dem Tabellen-Kontrollfeld hinzugefügt. Anschließend werden noch die gerade ermittelten Werte für die 'ID' und auch die Zeilennummer in die Formularfelder eingefügt. Die Zeilennummer entspricht dabei der Gesamtzahl der Datensätze, da der neue Datensatz als letzte Zeile in das Tabellen-Kontrollfeld aufgenommen wird.

```
034        stSql = "SELECT ""Name_ID_Autowert"".*, (SELECT COUNT(""ID"") FROM
              ""Name_ID_Autowert"") FROM ""Name_ID_Autowert"" WHERE ""ID"" = IDENTITY()"
```

In Firebird wird hier **IDENTITY()** durch **'"+loID+"'** ersetzt. Die Variable wurde ja bereits ermittelt.

```
035        oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
036        WHILE oAbfrageergebnis.next
037           stID = oAbfrageergebnis.getString(1)
038           stVorname = oAbfrageergebnis.getString(2)
039           stNachname = oAbfrageergebnis.getString(3)
040           l = oAbfrageergebnis.getLong(4)
041        WEND
042        oGridData.addRow (l, Array(stID, stVorname, stNachname))
043        oDialog3.getControl("NumericField1").Value = stID
044        oDialog3.getControl("Zeilennummer").Value = l
045     END IF
046  END SUB
```

Um neue Daten einzufügen müssen die Daten aus den Eingabefeldern entleert werden. Dies ist besonders wichtig bei dem Feld für den Primärschlüssel (das nicht beschreibbar ist) und dem versteckten Feld für die Zeilennummer.

```
001  SUB GridDatenNeu
002     oDialog3.getControl("NumericField1").Text = ""
003     oDialog3.getControl("TextField1").Text = ""
004     oDialog3.getControl("TextField2").Text = ""
005     oDialog3.getControl("Zeilennummer").Text = ""
006     oDialog3.getControl("GridControl1").deselectAllRows
007  END SUB
```

Sämtliche Eingabefelder werden geleert. Damit nicht beim nächsten Klick, z. B. zur Sortierung der markierte Datensatz in der Eingabezeile auftaucht, wird für alle Zeilen in dem Tabellen-Steuerelement die Selektion aufgehoben.

Die folgende Prozedur zum Löschen ist bereits im 2. Dialog ähnlich enthalten. Es können nur die Daten gelöscht werden, die in den Formularfeldern angezeigt werden. Dafür wird der versteckte Wert des Feldes für die Zeilennummer ausgelesen. Nach einer Kontrollabfrage wird dann der entsprechende SQL-Befehl ausgeführt und die Zeile über **removeRow** aus dem Tabellen-Kontrollfeld entfernt.

```
001  SUB GridDatenLoeschen
002     DIM loID AS LONG
003     DIM loRow AS LONG
```

```
004     DIM oGridData AS OBJECT
005     DIM inMsg AS INTEGER
006     DIM oDatenquelle AS OBJECT
007     DIM oVerbindung AS OBJECT
008     DIM oSQL_Anweisung AS OBJECT
009     DIM stSql AS STRING
010     loID = oDialog3.getControl("NumericField1").Value
011     loRow = oDialog3.getControl("Zeilennummer").Value
012     oGridData = oDialog3.Model.getByName("GridControl1").GridDataModel
013     IF loRow > 0 THEN
014        inMsg = MsgBox ("Soll der Datensatz wirklich gelöscht werden?",20,
              "Löschen eines Datensatzes")
015        IF inMsg = 6 THEN
016           oDatenquelle = thisDatabaseDocument.CurrentController
017           If NOT (oDatenquelle.isConnected()) THEN
018              oDatenquelle.connect()
019           END IF
020           oVerbindung = oDatenquelle.ActiveConnection()
021           oSQL_Anweisung = oVerbindung.createStatement()
022           stSql = "DELETE FROM ""Name_ID_Autowert"" WHERE ""ID"" = '"+loID+"'"
023           oSQL_Anweisung.executeUpdate(stSql)
024           oGridData.removeRow(loRow)
025        END IF
026     ELSE
027        MsgBox ("Kein Datensatz gelöscht." & CHR(13) &
              "Es fehlt eine Datensatzauswahl.",64,"Löschung nicht möglich")
028     END IF
029     GridDatenNeu
030  END SUB
```

Fortschrittsbalken für den Ablauf mehrerer Prozeduren

Wird für den Ablauf mehrerer Prozeduren hintereinander eine längere Zeit benötigt, so neigt der Nutzer / die Nutzerin schnell dazu, von einem «Hängen» des Systems auszugehen. Da bietet es sich an klar zu zeigen, dass der Rechner noch beschäftigt ist. Der folgende Dialog wurde für das Einlesen von Daten in eine im Netz befindliche PostgreSQL-Datenbank genutzt.

Zwei Beschriftungsfelder und ein Fortschrittsbalken ergeben den Aufbau des gesamten Dialogs. Der Dialog zeigt über die Vorgangsnummer und den Fortschrittsbalken an, wie weit der Import bisher fortgeschritten ist.

```
001  GLOBAL oDialog1 AS OBJECT
```

Zuerst wird eine allgemeingültige Variable für den Dialog erstellt. Dadurch kann von der Startprozedur aus der Code für die Änderung der Anzeige ausgelagert werden.

```
001  SUB Dialog1Start
002     DIM inWidth AS INTEGER
003     DIM inHeight AS INTEGER
004     DIM inx AS INTEGER
005     DIM iny AS INTEGER
006     DialogLibraries.LoadLibrary("Standard")
007     oDialog1 = createUnoDialog(DialogLibraries.Standard.D_Import)
008     oFrame = ThisComponent.CurrentController.Frame
009     oWin = oFrame.getContainerWindow()
```

```
010     inWidth = 315
011     inHeight = 100
012     inx = Int((oWin.Size.Width - inWidth) / 2)
013     iny = Int((oWin.Size.Height - inHeight) / 2)
014     oDialog1.setPosSize(inx, iny, inWidth, inHeight, 15)
015     oDialog1.setVisible(true)
016     FOR i = 1 TO 13
017        stLabel = "Vorgang " & i & " von 13"
018        Progress(i*100/13, stLabel)
019        SELECT CASE i
020           CASE 1
021              Import_Gemeinde
022           CASE 2
023              Import_Ort
024           CASE 3
                 …
025           CASE 13
026              Import_Eigentuemer
027        END SELECT
028     NEXT
029     oDialog1.dispose()
030  END SUB
```

Zuerst werden in der Prozedur **Dialog1Start** die Variablen deklariert. Hier sind nicht alle Variablen aufgeführt, die im Weiteren Verwendung finden.

Die Zeilen 8 bis 14 dienen dazu, den Dialog auf dem Bildschirm zu zentrieren. Dazu wird die Größenvorgabe des Dialogs mit **inWidth** und **inHeight** sowie die Größenvorgabe des verfügbaren Platzes für das Fenster (**oWin.Size.Width** und **oWin.Size.Height**) benötigt. Der Parameter '15' bei **oDialog1.setPosSize** besagt, dass sowohl die Position als auch die Größe des Dialogs geändert werden.

In Zeile 15 wird der Dialog mit **oDialog1.setVisible(true)** sichtbar geschaltet. Würde hier mit **Execute** der Dialog gestartet, so könnten keine weiteren Prozeduren ablaufen.

Zeile 17 und 18 in der Schleife, die ab Zeile 16 beginnt beeinflussen das Erscheinungsbild des Beschriftungsfeldes direkt über dem Fortschrittsbalken sowie die Länge des Fortschrittsbalkens selbst. Die Änderung dieses Erscheinungsbildes ist in die Prozedur Progress ausgelagert:

```
001  SUB Progress(doVal AS DOUBLE, stLabel AS STRING)
002     oDialog1.getControl("ProgressBar").setValue(doVal)
003     oDialog1.getControl("lblProgressBar").Text = stLabel
004  END SUB
```

Der Fortschrittsbalken hat über das Dialogdesign die Bezeichnung "ProgressBar" erhalten. Der Balken wurde im Design für 100 Einheiten ausgelegt. Da insgesamt 13 Prozeduren nacheinander aufgerufen werden wurde entsprechend 100/13 als die Länge für den Ablauf der ersten Prozedur übernommen.

Das Beschriftungsfeld ist über das Dialogdesign mit "lblProgressBar" ansprechbar. Hier wird nur jeweils die Anzeige der aktuellen Vorgangsnummer geändert.

Mit jedem Schleifendurchgang ändert sich die Variable **i**. Entsprechend tritt ein anderer **SELECT CASE** ein. Die Prozeduren, die ab Zeile 20 aufgerufen werden, sind also schlicht durchnummeriert über **CASE 1**, **CASE 2** usw. Die Schleife springt zum nächsten Wert, wenn die Prozedur abgelaufen ist.

Mit **oDialog1.dispose()** in Zeile 48 wird schließlich der Dialog wieder abgeschaltet. Jetzt könnte noch eine **Messagebox** den erfolgreichen Ablauf aller Prozeduren anzeigen.

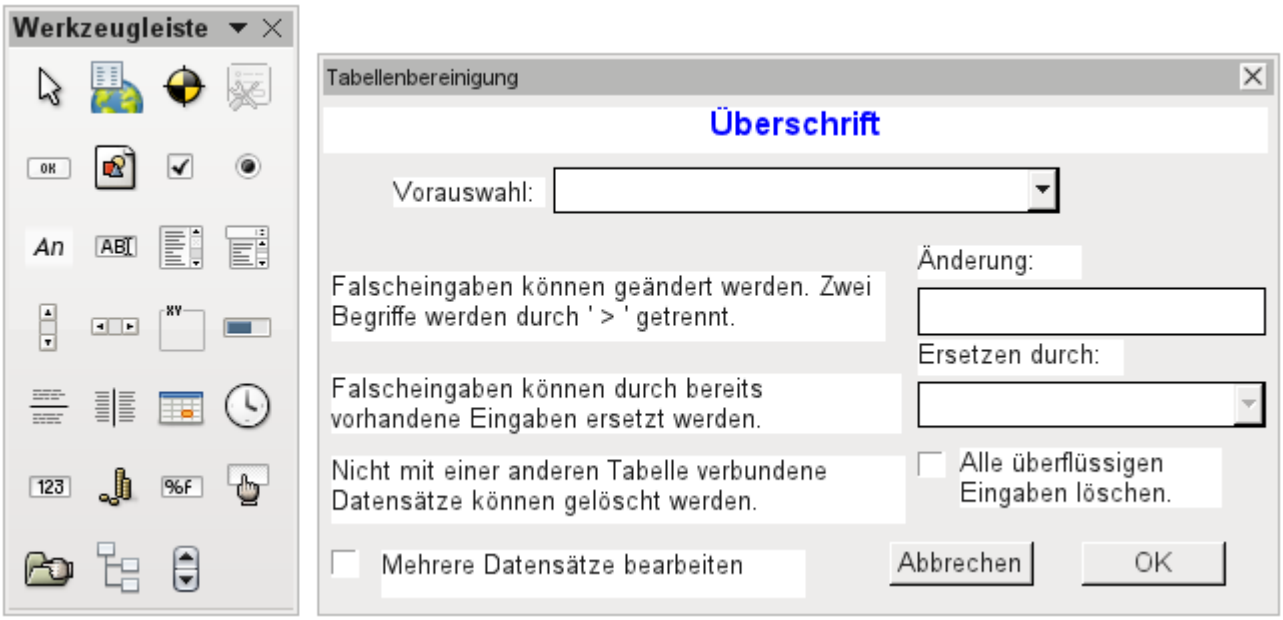
Fehleinträge von Tabellen mit Hilfe eines Dialogs bereinigen

Fehleingaben in Feldern fallen häufig erst später auf. Manchmal müssen auch gleich mehrere Datensätze mit der gleichen Eingabe auf einmal geändert werden. Dies ist in der normalen

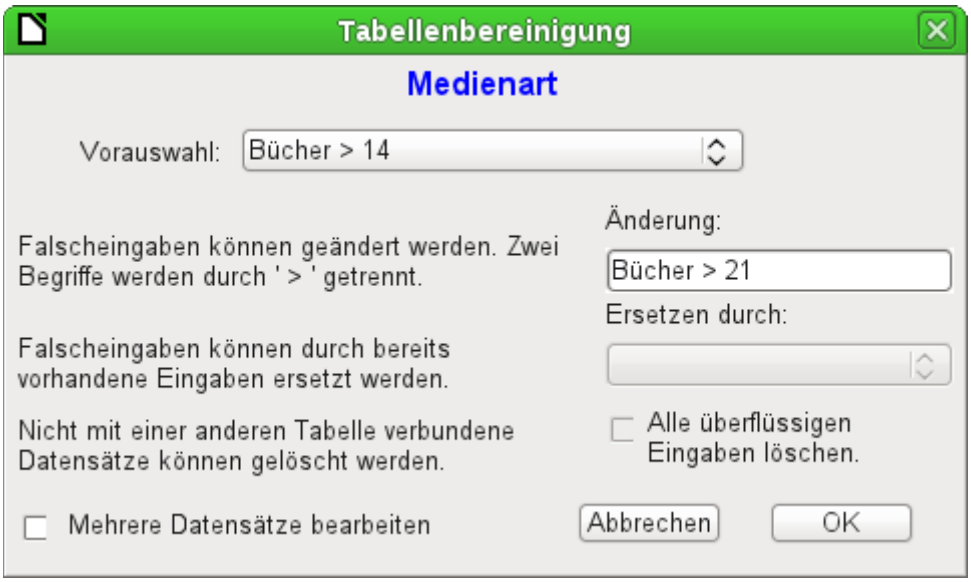
Tabellenansicht umständlich, je mehr Änderungen vorgenommen werden müssen, da für jeden Datensatz einzeln eine Eingabe erforderlich ist.

Formulare könnten hier mit Makros greifen. Wird aber für viele Tabellen ein identisch aufgebautes Formular benötigt, so bietet sich an, dies mit Dialogen zu erledigen. Ein Dialog wird zu Beginn mit den notwendigen Daten zu der jeweiligen Tabelle versehen und kann so statt mehrerer Formulare genutzt werden.

Diese Dialoge müssen für Firebird wegen der unterschiedlichen Systemtabellen entsprechend angepasst werden. Zu den Systemtabellen siehe den Anhang dieses Handbuches.



Dialoge werden neben den Modulen für Makros abgespeichert. Ihre Erstellung erfolgt ähnlich der eines Formulars. Hier stehen auch weitgehend ähnliche Kontrollfelder zur Verfügung. Lediglich das Tabellenkontrollfeld aus dem Formular fehlt als besondere Eingabemöglichkeit.



Wird ein Dialog ausgeführt, so erscheinen die Kontrollfelder entsprechend der Einstellung der grafischen Benutzeroberfläche.

Der oben abgebildete Dialog der Beispieldatenbank soll dazu dienen, die Tabellen zu bearbeiten, die nicht direkt in einem der Formulare als Grundlage vorhanden sind. So ist z.B. die Medienart über ein Listenfeld zugänglich, in der Makro-Version bereits durch ein Kombinationsfeld.

In der Makro-Version können die Inhalte der Felder zwar durch neue Inhalte ergänzt werden, eine Änderung alter Inhalte ist aber nicht möglich. In der Version ohne Makros erfolgt die Änderung über ein separates Tabellenkontrollfeld.

Während die Änderung noch ohne Makros recht einfach in den Griff zu bekommen ist, so ist es doch recht umständlich, die Medienart vieler Medien auf eine andere Medienart zu ändern. Angenommen, es gäbe die Medienarten 'Buch, gebunden', 'Buch, kartoniert', 'Taschenbuch' und 'Ringbuch'. Jetzt stellt sich nach längerem Betrieb der Datenbank heraus, dass muntere Zeitgenossen noch weitere ähnliche Medienarten für gedruckte Werke vorgesehen haben. Nur ist uns die Differenzierung viel zu weitgehend. Es soll also reduziert werden, am liebsten auf nur einen Begriff. Ohne Makro müssten jetzt die Datensätze in der Tabelle Medien (mit Hilfe von Filtern) aufgesucht werden und einzeln geändert werden. Mit Kenntnis von SQL geht dies über die SQL-Eingabe schon wesentlich besser. Mit einer Eingabe werden alle Datensätze der Tabelle Medien geändert. Mit einer zweiten SQL-Anweisung wird dann die jetzt überflüssige Medienart gelöscht, die keine Verbindung mehr zur Tabelle "Medien" hat. Genau dieses Verfahren wird mit diesem Dialog über **Ersetzen durch:** angewandt – nur dass eben die SQL-Anweisung erst über das Makro an die Tabelle "Medienart" angepasst wird, da das Makro auch andere Tabellen bearbeiten können soll.

Manchmal schleichen sich auch Eingaben in eine Tabelle ein, die im Nachhinein in den Formularen geändert wurden, also eigentlich gar nicht mehr benötigt werden. Da kann es nicht schaden, solche verwaisten Datensätze einfach zu löschen. Nur sind die über die grafische Oberfläche recht schwer ausfindig zu machen. Hier hilft wieder eine entsprechende SQL-Abfrage, die mit einer Löschanweisung gekoppelt ist. Diese Anweisung ist im Dialog je nach betroffener Tabelle unter **Alle überflüssigen Eingaben löschen** hinterlegt.

Sollen mit dem Dialog mehrere Änderungen durchgeführt werden, so ist dies über das Markierfeld **Mehrere Datensätze bearbeiten** anzugeben. Dann endet der Dialog nicht mit der Betätigung des Buttons OK.

Der Makrocode für diesen Dialog ist aus der Beispieldatenbank ersichtlich. Im Folgenden werden nur Ausschnitte daraus erläutert.

```
001  SUB Tabellenbereinigung(oEvent AS OBJECT)
```

Das Makro soll über Einträge im Bereich **Zusatzinformationen** des jeweiligen Buttons gestartet werden.

```
002  0: Formular, 1: Unterformular, 2: UnterUnterformular, 3: Kombinationsfeld oder
     Tabellenkontrollfeld, 4: Fremdschlüsselfeld im Formular, bei Tabellenkontrollfeld
     leer, 5: Tabellenname Nebentabelle, 6: Tabellenfeld1 Nebentabelle, 7: Tabellenfeld2
     Nebentabelle, ggf. 8: Tabellenname Nebentabelle für Tabellenfeld 2
```

Die Einträge in diesem Bereich werden zu Beginn des Makros als Kommentar aufgelistet. Die damit verbundenen Ziffern geben die Ziffern wieder, unter denen der jeweilige Eintrag aus dem Array ausgelesen wird. Das Makro kann Listenfelder verarbeiten, die zwei Einträge, getrennt durch «**>**», enthalten. Diese beiden Einträge können auch aus unterschiedlichen Tabellen stammen und über eine Abfrage zusammengeführt sein, wie z.B. bei der Tabelle "Postleitzahl", die für die Orte lediglich das Fremdschlüsselfeld "Ort_ID" enthält, zur Darstellung des Ortes also die Tabelle "Ort" benötigt.

```
003     DIM aFremdTabellen(0, 0 to 1)
004     DIM aFremdTabellen2(0, 0 to 1)
```

Unter den zu Beginn definierten Variablen fallen zwei Arrays auf. Während normale Arrays auch durch den Befehl **`Split()`** während der Laufzeit der Prozedur erstellt werden können, müssen zweidimensionale Arrays vorher definiert werden. Zweidimensionale Arrays werden z.B. benötigt, um aus einer Abfrage mehrere Datensätze zu speichern, bei denen die Abfrage selbst über mehr als ein Feld geht. Die beiden obigen Arrays müssen Abfragen auswerten, die sich jeweils auf zwei Tabellenfelder beziehen. Deshalb werden sie in der zweiten Dimension mit **`0 to 1`** auf zwei unterschiedliche Inhalte festgelegt.

```
005     stTag = oEvent.Source.Model.Tag
006     aTabelle() = Split(stTag, ", ")
007     FOR i = LBound(aTabelle()) TO UBound(aTabelle())
```

```
008         aTabelle(i) = trim(aTabelle(i))
009     NEXT
```

Die mitgegebenen Variablen werden ausgelesen. Die Reihenfolge steht im obigen Kommentar. Es gibt maximal 9 Einträge, wobei geklärt werden muss, ob ein 8. Eintrag für das Tabellenfeld2 und ein 9. Eintrag für eine zweite Tabelle existieren.

Wenn Werte aus einer Tabelle entfernt werden, so muss zuerst einmal berücksichtigt werden, ob sie nicht noch als Fremdschlüssel in anderen Tabellen existieren. In einfachen Tabellenkonstruktionen gibt es von einer Tabelle aus lediglich eine Fremdschlüsselverbindung zu einer anderen Tabelle. In der vorliegenden Beispieldatenbank aber wird z.B. die Tabelle "Ort" genutzt, um die Erscheinungsorte der Medien und die Orte für die Adressen zu speichern. Es wird also zweimal der Primärschlüssel der Tabelle "Ort" in unterschiedlichen Tabellen eingetragen. Diese Tabellen und Fremdschlüsselbezeichnungen könnten natürlich auch über die «Zusatzinformationen» eingegeben werden. Schöner wäre es aber, wenn sie universell für alle Fälle ermittelt werden. Dies geschieht durch die folgende Abfrage.

```
010     stSql = "SELECT ""FKTABLE_NAME"", ""FKCOLUMN_NAME"" FROM
           ""INFORMATION_SCHEMA"".""SYSTEM_CROSSREFERENCE"" WHERE ""PKTABLE_NAME"" = '"
           + aTabelle(5) + "'"
```

In der Datenbank sind im Bereich "INFORMATION_SCHEMA" alle Informationen zu den Tabellen der Datenbank abgespeichert, so auch die Informationen zu den Fremdschlüsseln. Die entsprechende Tabelle, die diese Informationen enthält, ist über "INFORMATION_SCHEMA"."SYSTEM_CROSSREFERENCE" erreichbar. Mit "PKTABLE_NAME" wird die Tabelle erreicht, die ihren Primärschlüssel ("Primary Key") in die Beziehung mit einbringt. Mit "FKTABLE_NAME" wird die Tabelle erreicht, die diesen Primärschlüssel als Fremdschlüssel ("Foreign Key") nutzt. Über "FKCOLUMN_NAME" wird schließlich die Bezeichnung des Fremdschlüsselfeldes ermittelt.

Die Tabelle, die einen Primärschlüssel als Fremdschlüssel zur Verfügung stellt, befindet sich in dem vorher erstellten Array an der 6. Position. Da die Zählung mit 0 beginnt, wird der Wert aus dem Array mit **aTabelle(5)** ermittelt.

```
011     inZaehler = 0
012     stFremdIDTab1Tab2 = "ID"
013     stFremdIDTab2Tab1 = "ID"
014     stNebentabelle = aTabelle(5)
```

Bevor die Auslesung des Arrays gestartet wird, müssen einige Standardwerte gesetzt werden. Dies sind der Zähler für das Array, in das die Werte der Nebentabelle geschrieben werden, der Standardprimärschlüssel, wenn nicht der Fremdschlüssel für eine zweite Tabelle benötigt wird und die Standardnebentabelle, die sich auf die Haupttabelle bezieht, bei Postleitzahl und Ort z.B. die Tabelle für die Postleitzahl.

Bei der Verknüpfung von zwei Feldern zur Anzeige in den Listenfeldern kann es ja, wie oben erwähnt, zu einer Verknüpfung über zwei Tabellen kommen. Für die Darstellung von Postleitzahl und Ort lautet hier die Abfrage

```
SELECT "Postleitzahl"."Postleitzahl" || ' > ' || "Ort"."Ort"
FROM "Postleitzahl", "Ort"
WHERE "Postleitzahl"."Ort_ID" = "Ort"."ID"
```

Die Tabelle, die sich auf das erste Feld bezieht (Postleitzahl), ist mit der zweiten Tabelle über einen Fremdschlüssel verbunden. Lediglich die Information der beiden Tabellen und der Felder "Postleitzahl" und "Ort" wurde dem Makro mitgegeben. Die Primärschlüssel sind standardmäßig in dem Beispiel mit der Bezeichnung "ID" versehen. Der Fremdschlüssel von "Ort" in "Postleitzahl" muss also über das Makro ermittelt werden.

Genauso muss über das Makro jede andere Tabelle ermittelt werden, mit der die Inhalte des Listenfeldes über Fremdschlüssel in Verbindung stehen.

```
015     oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
016     WHILE oAbfrageergebnis.next
017        ReDim Preserve aFremdTabellen(inZaehler,0 to 1)
```

Das Array muss jedes Mal neu dimensioniert werden. Damit die alten Inhalte erhalten bleiben, erfolgt über (Preserve) eine Sicherung des vorherigen Inhaltes.

```
018         aFremdTabellen(inZaehler,0) = oAbfrageergebnis.getString(1)
```

Auslesen des ersten Feldes mit dem Namen der Tabelle, die den Fremdschlüssel enthält. Ergebnis für die Tabelle "Postleitzahl" ist hier die Tabelle "Adresse".

```
019         aFremdTabellen(inZaehler,1) = oAbfrageergebnis.getString(2)
```

Auslesen des zweiten Feldes mit der Bezeichnung des Fremdschlüsselfeldes. Ergebnis für die Tabelle "Postleitzahl" ist hier das Feld "Postleitzahl_ID" in der Tabelle "Adresse".

Für den Fall, dass dem Aufruf der Prozedur auch der Name einer zweiten Tabelle mitgegeben wurde, erfolgt die folgende Schleife. Nur wenn der Name der zweiten Tabelle als Fremdschlüsseltabelle für die erste Tabelle auftaucht, erfolgt hier eine Änderung der Standardeinträge. In unserem Fall kommt dies nicht vor, da die Tabelle "Ort" keinen Fremdschlüssel der Tabelle "Postleitzahl" enthält. Der Standardeintrag für die Nebentabelle bleibt also bei "Postleitzahl"; schließlich ist die Kombination von Postleitzahl und Ort eine Grundlage für die Adressentabelle, die einen Fremdschlüssel zu der Tabelle "Postleitzahl" enthält.

```
020         IF UBound(aTabelle()) = 8 THEN
021            IF aTabelle(8) = aFremdTabellen(inZaehler,0) THEN
022               stFremdIDTab2Tab1 = aFremdTabellen(inZaehler,1)
023               stNebentabelle = aTabelle(8)
024            END IF
025         END IF
026         inZaehler = inZaehler + 1
```

Da eventuell noch weitere Werte auszulesen sind, erfolgt eine Erweiterung des Zählers zur Neudimensionierung des Arrays. Anschließend wird die Schleife beendet.

```
027      WEND
```

Existiert im Aufruf der Prozedur ein zweiter Tabellenname, so wird die gleiche Abfrage jetzt mit dem zweiten Tabellennamen gestartet:

```
028      IF UBound(aTabelle()) = 8 THEN
```

Der Ablauf ist identisch. Nur wird in der Schleife jetzt gesucht, ob vielleicht der erste Tabellenname als Fremdschlüssel-Tabellenname auftaucht. Das ist hier der Fall: Die Tabelle "Postleitzahl" enthält den Fremdschlüssel "Ort_ID" aus der Tabelle "Ort". Dieser Fremdschlüssel wird also jetzt der Variablen **stFremdIDTab1Tab2** zugewiesen, so dass die Beziehung der Tabellen untereinander definiert werden kann.

```
029         IF aTabelle(5) = aFremdTabellen2(inZaehler,0) THEN
030            stFremdIDTab1Tab2 = aFremdTabellen2(inZaehler,1)
031         END IF
```

Nach einigen weiteren Einstellungen zur korrekten Rückkehr nach Aufruf des Dialogs in die entsprechenden Formulare (Ermittlung der Zeilennummer des Formulars, damit nach einem Neueinlesen auf die Zeilennummer wieder gesprungen werden kann) startet die Schleife, die den Dialog gegebenenfalls wieder neu erstellt, wenn die erste Aktion erfolgt ist, der Dialog aber für weitere Aktionen offen gehalten werden soll. Die Einstellung zur Wiederholung erfolgt über das entsprechende Markierfeld.

```
032      DO
```

Bevor der Dialog gestartet wird, wird erst einmal der Inhalt der Listenfelder ermittelt. Dabei muss berücksichtigt werden, ob die Listenfelder zwei Tabellenfelder darstellen und eventuell sogar einen Bezug zu zwei Tabellen haben.

```
033         IF UBound(aTabelle()) = 6 THEN
```

Das Listenfeld bezieht sich nur auf eine Tabelle und ein Feld, da das Array bei dem Tabellenfeld1 der Nebentabelle endet.

```
034            stSql = "SELECT """ + aTabelle(6) + """ FROM """ + aTabelle(5)
035               + """ ORDER BY """ + aTabelle(6) + """"
036         ELSEIF UBound(aTabelle()) = 7 THEN
```

Das Listenfeld bezieht sich auf zwei Tabellenfelder, aber nur auf eine Tabelle, da das Array bei dem Tabellenfeld2 der Nebentabelle endet.

```
037             stSql = "SELECT """ + aTabelle(6) + """||' > '||""" + aTabelle(7)
038                + """ FROM """ + aTabelle(5) + """ ORDER BY """ + aTabelle(6) + """"
039          ELSE
```

Das Listenfeld hat zwei Tabellenfelder und zwei Tabellen als Grundlage. Diese Abfrage trifft also auf das Beispiel mit der Postleitzahl und den Ort zu.

```
040             stSql ="SELECT """ + aTabelle(5) + """."""" + aTabelle(6) + """||' > '||"""
                   + aTabelle(8) + """."""" + aTabelle(7) + """ FROM """ + aTabelle(5)
                   + """, """ + aTabelle(8) + """ WHERE """ + aTabelle(8) + """."""
                   + stFremdIDTab2Tab1 + """ = """ + aTabelle(5) + """."""
                   + stFremdIDTab1Tab2 + """ ORDER BY """ + aTabelle(6) + """"
041          END IF
```

Hier erfolgt die erste Auswertung zur Ermittlung von Fremdschlüsseln. Die Variablen **stFremdIDTab2Tab1** und **stFremdIDTab1Tab2** starten mit dem Wert "ID". Für **stFremdIDTab1Tab2** wurde in der Auswertung der vorhergehenden Abfrage ein anderer Wert ermittelt, nämlich der Wert "Ort_ID". Damit ergibt die vorherige Abfragekonstruktion genau den Inhalt, der weiter oben bereits für Postleitzahl und Ort formuliert wurde – lediglich erweitert um die Sortierung.

Jetzt muss der Kontakt zu den Listenfeldern erstellt werden, damit diese mit dem Inhalt der Abfragen bestückt werden. Diese Listenfelder existieren noch nicht, da noch gar kein Dialog existiert. Dieser Dialog wird mit den folgenden Zeilen erst einmal im Speicher erstellt, bevor er tatsächlich auf dem Bildschirm ausgeführt wird.

```
042          DialogLibraries.LoadLibrary("Standard")
043          oDlg = CreateUnoDialog(DialogLibraries.Standard.Dialog_Tabellenbereinigung)
```

Anschließend werden Einstellungen für die Felder, die der Dialog enthält, ausgeführt. Hier als Beispiel das Auswahllistenfeld, das mit dem Ergebnis der obigen Abfrage bestückt wird:

```
044          oCtlList1 = oDlg.GetControl("ListBox1")
045          oCtlList1.addItems(aInhalt(),0)
```

Der Zugriff auf die Felder des Dialogs erfolgt über **GetControl** sowie die entsprechende Bezeichnung. Bei Dialogen ist es nicht möglich, für zwei Felder die gleichen Bezeichnungen zu verwenden, da sonst eine Auswertung des Dialoges problematisch wäre.

Das Listenfeld wird mit den Inhalten aus der Abfrage, die in dem Array **aInhalt()** gespeichert wurden, ausgestattet. Das Listenfeld enthält nur die darzustellenden Inhalte als ein Feld, wird also nur in der Position '0' bestückt.

Nachdem alle Felder mit den gewünschten Inhalten versorgt wurden, wird der Dialog gestartet.

```
046          Select Case oDlg.Execute()
047          Case 1  'Case 1 bedeutet die Betätigung des Buttons "OK"
048          Case 0  'Wenn Button "Abbrechen"
049             inWiederholung = 0
050          End Select
051       LOOP WHILE inWiederholung = 1
```

Der Dialog wird so lange durchgeführt, wie der Wert für **inWiederholung** auf '1' steht. Diese Setzung erfolgt mit dem entsprechenden Markierfeld.

Hier der Inhalt nach Betätigung des Buttons OK im Kurzüberblick:

```
052       Case 1
053          stInhalt1 = oCtlList1.getSelectedItem() 'Wert aus Listbox1 auslesen ...
054          REM ... und den dazugehoerigen ID-Wert bestimmen.
```

Der ID-Wert des ersten Listenfeldes wird in der Variablen **inLB1** gespeichert.

```
055          stText = oCtlText.Text   ' Den Wert des Feldes auslesen.
```

Ist das Textfeld nicht leer, so wird nur der Eintrag im Textfeld erledigt. Weder das Listenfeld für eine andere Zuweisung noch das Markierfeld für eine Löschung aller Daten ohne Bezug werden

berücksichtigt. Dies wird auch dadurch verdeutlicht, dass bei Texteingabe die anderen Felder inaktiv geschaltet werden.

```
056         IF stText <> "" THEN
```

Ist das Textfeld nicht leer, dann wird der neue Wert anstelle des alten Wertes mit Hilfe des vorher ausgelesenen ID-Feldes in die Tabelle geschrieben. Dabei werden wieder zwei Einträge ermöglicht, wie dies auch in dem Listenfeld geschieht. Das Trennzeichen ist «**>**». Bei Zwei Einträgen in verschiedenen Tabellen müssen auch entsprechend zwei UPDATE-Kommandos gestartet werden, die hier gleichzeitig erstellt und, durch ein Semikolon getrennt, weitergeleitet werden.

```
057         ELSEIF oCtlList2.getSelectedItem() <> "" THEN
```

Wenn das Textfeld leer ist und das Listenfeld 2 einen Wert aufweist, muss der Wert des Listenfeldes 1 durch den Wert des Listenfeldes 2 ersetzt werden. Das bedeutet, dass alle Datensätze der Tabellen, in denen die Datensätze der Listenfelder Fremdschlüssel sind, überprüft und gegebenenfalls mit einem geänderten Fremdschlüssel beschrieben werden müssen.

```
058         stInhalt2 = oCtlList2.getSelectedItem()
            REM Den Wert der Listbox auslesen.
            REM ID für den Wert das Listenfeld ermitteln.
```

Der ID-Wert des zweiten Listenfeldes wird in der Variablen **inLB2** gespeichert. Auch dieses erfolgt wieder unterschiedlich, je nachdem, ob ein oder zwei Felder in dem Listenfeld enthalten sind sowie eine oder zwei Tabellen Ursprungstabellen des Listenfeldinhaltes sind.

Der Ersetzungsprozess erfolgt danach, welche Tabelle als die Tabelle definiert wurde, die für die Haupttabelle den Fremdschlüssel darstellt. Für das oben erwähnte Beispiel ist dies die Tabelle "Postleitzahl", da die "Postleitzahl_ID" der Fremdschlüssel ist, der durch Listenfeld 1 und Listenfeld 2 wiedergegeben wird.

```
059     IF stNebentabelle = aTabelle(5) THEN
060         FOR i = LBound(aFremdTabellen()) TO UBound(aFremdTabellen())
```

Ersetzen des alten ID-Wertes durch den neuen ID-Wert. Problematisch ist dies bei n:m-Beziehungen, da dann der gleiche Wert doppelt zugeordnet werden kann. Dies kann erwünscht sein, muss aber vermieden werden, wenn der Fremdschlüssel hier Teil des Primärschlüssels ist. So darf in der Tabelle "rel_Medien_Verfasser" ein Medium nicht zweimal den gleichen Verfasser haben, da der Primärschlüssel aus der "Medien_ID" und der "Verfasser_ID" gebildet wird. In der Abfrage werden alle Schlüsselfelder untersucht, die zusammen die Eigenschaft **UNIQUE** haben oder als Fremdschlüssel mit der Eigenschaft **UNIQUE** über einen Index definiert wurden.

Sollte also der Fremdschlüssel die Eigenschaft **UNIQUE** haben und bereits mit der gewünschten zukünftigen **inLB2** dort vertreten sein, so kann der Schlüssel nicht ersetzt werden.

```
061 stSql = "SELECT ""COLUMN_NAME"" FROM ""INFORMATION_SCHEMA"".""SYSTEM_INDEXINFO""
        WHERE ""TABLE_NAME"" = '" + aFremdTabellen(i,0) + "' AND ""NON_UNIQUE"" = False
        AND ""INDEX_NAME"" = (SELECT ""INDEX_NAME"" FROM
        ""INFORMATION_SCHEMA"".""SYSTEM_INDEXINFO"" WHERE ""TABLE_NAME"" = '"
        + aFremdTabellen(i,0) + "' AND ""COLUMN_NAME"" = '" + aFremdTabellen(i,1) + "')"
```

Mit **"NON_UNIQUE" = False** werden die Spaltennamen angegeben, die **UNIQUE** sind. Allerdings werden nicht alle Spaltennamen benötigt, sondern nur die, die gemeinsam mit dem Fremdschlüsselfeld einen Index bilden. Dies ermittelt der «Subselect» mit dem gleichen Tabellennamen (der den Fremdschlüssel enthält) und dem Namen des Fremdschlüsselfeldes.

Wenn jetzt der Fremdschlüssel in der Ergebnismenge vorhanden ist, dann darf der Schlüsselwert nur dann ersetzt werden, wenn gleichzeitig andere Felder dazu benutzt werden, den entsprechenden Index als **UNIQUE** zu definieren. Hierzu muss beim Ersetzen darauf geachtet werden, dass die Einzigartigkeit der Indexkombination nicht verletzt wird.

```
062     IF aFremdTabellen(i,1) = stFeldbezeichnung THEN
063         inUnique = 1
064     ELSE
065         ReDim Preserve aSpalten(inZaehler)
066         aSpalten(inZaehler) = oAbfrageergebnis.getString(1)
```

```
067         inZaehler = inZaehler + 1
068      END IF
```

Alle Spaltennamen, die neben dem bereits bekannten Spaltennamen des Fremdschlüsselfeldes als Index mit der Eigenschaft **UNIQUE** auftauchen, werden in einem Array abgespeichert. Da der Spaltenname des Fremdschlüsselfeldes auch zu der Gruppe gehört, wird durch ihn gekennzeichnet, dass die Einzigartigkeit bei der Datenänderung zu berücksichtigen ist.

```
069  IF inUnique = 1 THEN
070     stSql = "UPDATE """ + aFremdTabellen(i,0) + """ AS ""a"" SET """
           + aFremdTabellen(i,1) + """='" + inLB2 + "' WHERE """ + aFremdTabellen(i,1)
           + """='" + inLB1 + "' AND ( SELECT COUNT(*) FROM """ + aFremdTabellen(i,0)
           + """ WHERE """ + aFremdTabellen(i,1) + """='" + inLB2 + "' )"
071     IF inZaehler > 0 THEN
072        stFeldgruppe = Join(aSpalten(), """||  ||""")
```

Gibt es mehrere Felder, die neben dem Fremdschlüsselfeld gemeinsam einen **UNIQUE**-Index bilden, so werden die hier für eine SQL-Gruppierung zusammengeführt. Ansonsten erscheint als **stFeldgruppe** nur **aSpalten(0)**.

```
073        stFeldbezeichnung = ""
074        FOR ink = LBound(aSpalten()) TO UBound(aSpalten())
075           stFeldbezeichnung = stFeldbezeichnung + " AND """ + aSpalten(ink)
              + """ = ""a"".""" + aSpalten(ink) + """ "
```

Die SQL-Teilstücke werden für eine korrelierte Unterabfrage zusammengefügt.

```
076        NEXT
077        stSql = Left(stSql, Len(stSql) - 1)
```

Die vorher erstellte Abfrage endet mit einer Klammer. Jetzt sollen noch Inhalte zu der Unterabfrage hinzugefügt werden. Also muss die Schließung wieder aufgehoben werden. Anschließend wird die Abfrage durch die zusätzlich ermittelten Bedingungen ergänzt.

```
078        stSql=stSql + stFeldbezeichnung + "GROUP BY (""" + stFeldgruppe + """) ) < 1"
079     END IF
```

Wenn die Feldbezeichnung des Fremdschlüssels nichts mit dem Primärschlüssel oder einem **UNIQUE**-Index zu tun hat, dann kann ohne weiteres auch ein Inhalt doppelt erscheinen

```
080     ELSE
081        stSql = "UPDATE """ + aFremdTabellen(i,0) + """ SET """ + aFremdTabellen(i,1)
           + """='" + inLB2 + "' WHERE """ + aFremdTabellen(i,1) + """='" + inLB1 + "'"
082  END IF
083  oSQL_Anweisung.executeQuery(stSql)
084  NEXT
```

Das Update wird so lange durchgeführt, wie unterschiedliche Verbindungen zu anderen Tabellen vorkommen, d. h. die aktuelle Tabelle einen Fremdschlüssel in anderen Tabellen liegen hat. Dies ist z. B. bei der Tabelle "Ort" zweimal der Fall: in der Tabelle "Medien" und in der Tabelle "Postleitzahl".

Anschließend kann der alte Wert aus dem Listenfeld 1 gelöscht werden, weil er keine Verbindung mehr zu anderen Tabellen hat.

```
085  stSql = "DELETE FROM """ + aTabelle(5) + """ WHERE ""ID""='" + inLB1 + "'"
086  oSQL_Anweisung.executeQuery(stSql)
```

Das gleiche Verfahren muss jetzt auch für eine eventuelle zweite Tabelle durchgeführt werden, aus der die Listenfelder gespeist werden. In unserem Beispiel ist die erste Tabelle die Tabelle "Postleitzahl", die zweite Tabelle die Tabelle "Ort".

Wenn das Textfeld leer ist und das Listenfeld 2 ebenfalls nichts enthält, wird nachgesehen, ob eventuell das Markierfeld darauf hindeutet, dass alle überflüssigen Einträge zu löschen sind. Dies ist für die Einträge der Fall, die nicht mit anderen Tabellen über einen Fremdschlüssel verbunden sind.

```
087  ELSEIF oCtlCheck1.State = 1 THEN
088     stBedingung = ""
089     IF stNebentabelle = aTabelle(5) THEN
090        FOR i = LBound(aFremdTabellen()) TO UBound(aFremdTabellen())
```

```
091             stBedingung = stBedingung + """ID"" NOT IN (SELECT """ +
                   aFremdTabellen(i,1) + """ FROM """ + aFremdTabellen(i,0) + """) AND "
092          NEXT
093       ELSE
094          FOR i = LBound(aFremdTabellen2()) TO UBound(aFremdTabellen2())
095             stBedingung = stBedingung + """ID"" NOT IN (SELECT """ +
                   aFremdTabellen2(i,1) + """ FROM """ + aFremdTabellen2(i,0) + """) AND "
096          NEXT
097       END IF
```

Das letzte **AND** muss abgeschnitten werden, da sonst die Löschanweisung mit einem **AND** enden würde:

```
098       stBedingung = Left(stBedingung, Len(stBedingung) - 4)
099       stSql = "DELETE FROM """ + stNebentabelle + """ WHERE " + stBedingung + ""
100       oSQL_Anweisung.executeQuery(stSql)
```

Da nun schon einmal die Tabelle bereinigt wurde, kann auch gleich der Tabellenindex überprüft und gegebenenfalls nach unten korrigiert werden. Siehe hierzu die in dem vorhergehenden Kapitel *Interne Datenbanken sicher schließen* erwähnte Prozedur.

```
101       Tabellenindex_runter(stNebentabelle)
```

Anschließend wird noch gegebenenfalls das Listenfeld des Formulars, aus dem der Tabellenbereinigungsdialog aufgerufen wurde, auf den neuesten Stand gebracht. Unter Umständen ist das gesamte Formular neu einzulesen. Hierzu wurde zu Beginn der Prozedur der aktuelle Datensatz ermittelt, so dass nach einem Auffrischen des Formulars der aktuelle Datensatz auch wieder eingestellt werden kann.

```
102       oDlg.endExecute()  'Dialog beenden ...
103       oDlg.Dispose()     '... und aus dem Speicher entfernen
104    END SUB
```

Dialoge werden mit **endExecute()** beendet und mit **Dispose()** komplett aus dem Speicher entfernt.

Makrozugriff mit Access2Base

In LibreOffice ist seit der Version 4.2 die Erweiterung Access2Base integriert. Der Zugriff auf diese Bibliothek erfolgt über

```
001 Sub DBOpen(Optional oEvent As Object)
002    If GlobalScope.BasicLibraries.hasByName("Access2Base") then
003       GlobalScope.BasicLibraries.loadLibrary("Access2Base")
004    End If
005    Call Application.OpenConnection(ThisDatabaseDocument)
006 End Sub
```

Eine englischsprachige Beschreibung mit Beispielen ist auf der Seite *http://www.access2base.com/access2base.html* zu finden.

Die Bibliothek stellt nicht zusätzliche Funktionen zur Verfügung, sondern versucht, dem Anwender den Zugriff auf die Möglichkeiten der LibreOffice-API zu vereinfachen. Eine kurze Beschreibung ist auch in der Hilfe zu LO zu finden.

Wartung

Allgemeines zur Wartung von Datenbanken

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

Datenbank komprimieren

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

Seit der Version LO 3.6 wird die Komprimierung für die HSQLDB mit **SHUTDOWN COMPACT** standardmäßig spätestens beim Schließen der Datenbank durchgeführt.

Die interne Firebird Datenbank lässt sich nicht einfach komprimieren. Hier müsste ein Sicherung und Wiederherstellung der Daten durchgeführt werden. Einen einfachen Befehl wie für die HSQLDB gibt es nicht. Deswegen werden die Datenbankdateien mit der internen Firebird Datenbank beständig größer.

Autowerte neu einstellen

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

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

```
001 ALTER TABLE "Tabellenname" ALTER COLUMN "ID" RESTART WITH <NeuerWert>
```

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

Bei Firebird muss der Zugriff anders lauten:

```
001 ALTER TABLE "Tabellenname"
002 ALTER "ID" 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.

Datenbankeigenschaften abfragen

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

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

```
001 SELECT "COLUMN_NAME",
002    "TYPE_NAME",
003    "COLUMN_SIZE",
004    "COLUMN_DEF" AS "Default Value"
005 FROM "INFORMATION_SCHEMA"."SYSTEM_COLUMNS"
006 WHERE "TABLE_NAME" = 'Suchtabelle'
007 ORDER BY "ORDINAL_POSITION"
```

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

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

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

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

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

```
001 SELECT "a".RDB$RELATION_NAME AS "Tables",
002    "a".RDB$FIELD_NAME AS "Fields",
003    "c".RDB$TYPE_NAME AS "Types",
004    "a".RDB$Field_POSITION AS "Fieldposition",
005    "a".RDB$NULL_FLAG AS "Nullflag",
006    "b".RDB$FIELD_LENGTH AS "Fieldlength"
007 FROM RDB$RELATION_FIELDS AS "a", RDB$FIELDS AS "b", RDB$TYPES AS "c"
008 WHERE "a".RDB$FIELD_SOURCE = "b".RDB$FIELD_NAME
009    AND "b".RDB$FIELD_TYPE = "c".RDB$TYPE
010    AND "c".RDB$FIELD_NAME = 'RDB$FIELD_TYPE'
011    AND "a".RDB$RELATION_NAME = 'Suchtabelle'
012 ORDER BY "Tables", "Fieldposition"
```

Weitere Informationen hierzu auch im Anhang.

Daten exportieren

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

```
001 SCRIPT 'Datenbankname'
```

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

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

Das Format der exportierten Datei lässt sich über

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

andern.

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

```
001 CREATE SCHEMA PUBLIC AUTHORIZATION DBA
```

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

```
002 CREATE USER SA PASSWORD ""
003 GRANT DBA TO SA
004 SET WRITE_DELAY 60
005 SET SCHEMA PUBLIC
```

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

Für Firebird besteht dieser direkte Export nicht. Es muss mit entsprechenden Zusatzprogrammen gearbeitet werden. Siehe dazu *https://www.firebirdfaq.org/faq86/*.

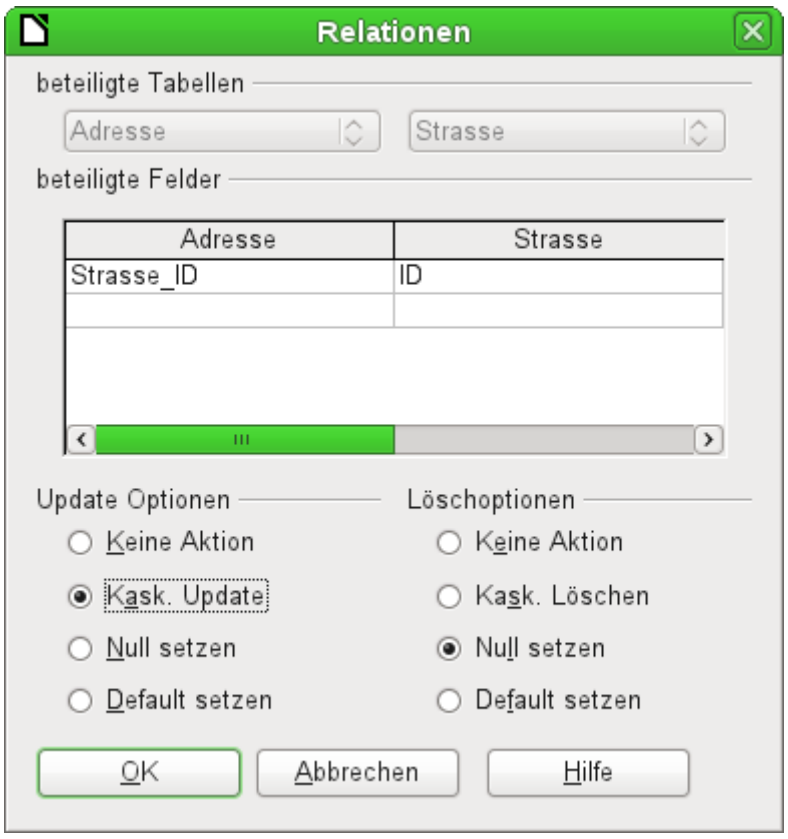
Tabellen auf unnötige Einträge überprüfen

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

Einträge durch Beziehungsdefinition kontrollieren

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

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



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

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

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

Entsprechend werden die Löschoptionen gehandhabt.

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

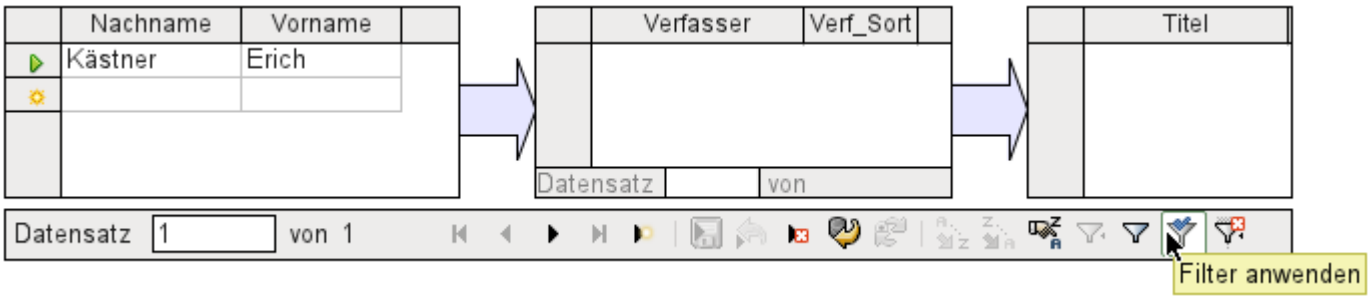
Die Beziehungsdefinition hilft also nur, die Beziehung selbst sauber zu halten, nicht aber unnötige Datensätze in der Tabelle zu entfernen, die ihren Primärschlüssel als Fremdschlüssel in der Beziehung zur Verfügung stellt. Es können also beliebig viele Straßen ohne Adresse existieren.

Einträge durch Formular und Unterformular bearbeiten

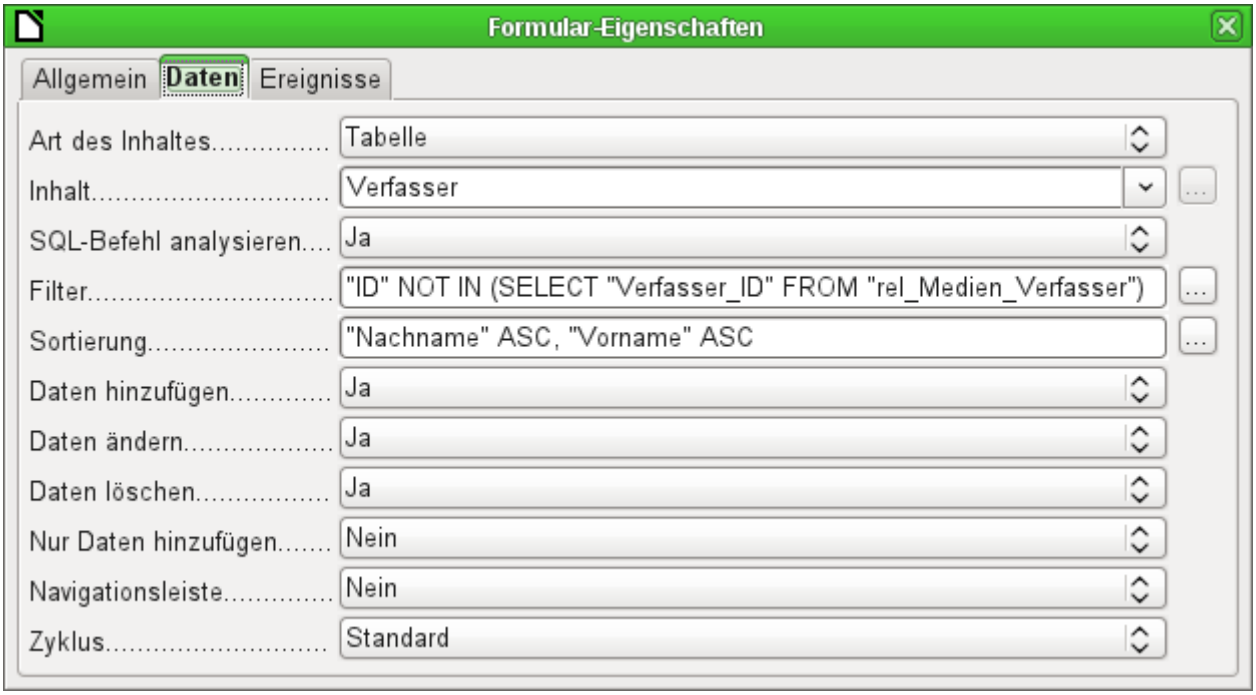
Vom Prinzip her kann der gesamte Zusammenhang zwischen Tabellen innerhalb von Formularen dargestellt werden. Am einfachsten ist dies natürlich, wenn eine Tabelle nur zu einer anderen Tabelle in Beziehung steht. So gibt z.B. in dem folgenden Beispiel der Verfasser seinen Primärschlüssel als Fremdschlüssel an die Tabelle "rel_Medien_Verfasser" weiter; "rel_Medien_Verfasser" enthält außerdem einen Fremdschlüssel von "Medien", so dass die folgende Anordnung eine n:m-Beziehung mit drei Formularen aufzeigt. Jede wird durch eine Tabelle präsentiert.

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

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



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



Verwaiste Einträge durch Abfrage ermitteln

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

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

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

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

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

Datenbankgeschwindigkeit

Einfluss von Abfragen

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

```
003 … WHERE "ID" NOT IN (SELECT "Verfasser_ID“ FROM "rel_Medien_Verfasser")
```

eine große Menge an vorhandenen Fremdschlüsseln der Tabelle "rel_Medien_Verfasser" ergeben, die erst mit dem Primärschlüssel der Tabelle "Verfasser" für jeden Datensatz der Tabelle "Verfasser" verglichen werden muss. So eine Abfrage sollte also nicht für den täglichen Gebrauch gedacht sein, sondern nur vorübergehend wie hier zum Beispiel für die Wartung von Tabellen. Suchfunktionen sind anders aufzubauen, damit die Suche von Daten nicht endlos dauert und die Arbeit mit der Datenbank im Alltagsbetrieb verleidet.

Einfluss von Listenfeldern und Kombinationsfeldern

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

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

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

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

Server neu gestellt wird. Ansichten (*Views*) hält der Server als fertige Abfragen schließlich vorrätig.

Einfluss des verwendeten Datenbanksystems

Die interne HSQLDB ist auf eine gut funktionierende Zusammenarbeit von Base mit Java ausgelegt. Bei Tests mit mehreren tausend Datensätzen ist leider das Scrollverhalten vom ersten zum letzten Datensatz in Base gegenüber der internen Firebird Datenbank deutlich langsamer.

Externe Datenbanken laufen hier deutlich schneller. Von der Geschwindigkeit sind die direkten Verbindungen zu MySQL oder PostgreSQL sowie die Verbindungen über ODBC nahezu gleichwertig. JDBC ist ebenfalls auf das Zusammenspiel mit Java angewiesen, funktioniert daher nicht schneller als eine interne Verbindung zur HSQLDB. Siehe dazu auch im Kapitel «Datenbank erstellen» die Anmerkungen zur *Geschwindigkeit*.

Anhang

Barcode

Um die Barcode-Druckfunktion nutzen zu können, muss der Font «ean13.ttf» installiert sein. Dieser Font ist frei verfügbar.

EAN13-Barcodes können mittels «ean13.ttf» folgendermaßen erstellt werden:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Zahl	Großbuchstaben, A=0 B=1 usw.						*	Kleinbuchstaben, a=0 b=1 usw.						+

Siehe hierzu die Abfrage "Barcode_EAN13_ttf_Bericht" der Beispieldatenbank «Medien_ohne_Makros»

Datentypen des Tabelleneditors

Ganzzahlen

Typ	***Zusatz***	***HSQLDB***	***Umfang***	***Speicherbedarf***
Tiny Integer	TINYINT	TINYINT ~~Firebird~~	2^8 = 256 | - 128 bis + 127	1 Byte
Small Integer	SMALLINT	SMALLINT	2^{16} = 65536 | - 32768 bis + 32767	2 Byte
Integer	INTEGER	INTEGER | INT	2^{32} = 4294967296 | - 2147483648 bis + 2147483647	4 Byte
BigInt	BIGINT	BIGINT	2^{64} = 18446744073709551615 | - 9223372036854775808 bis + 9223372036854775807	8 Byte

Fließkommazahlen

Typ	***Zusatz***	***HSQLDB***	***Umfang***	***Speicherbedarf***
Dezimal	DECIMAL	DECIMAL	Unbegrenzt, durch GUI auf 50 Stellen, einstellbar, feste Nachkommastellen, exakte Genauigkeit	variabel
Zahl	NUMERIC	NUMERIC	Unbegrenzt, durch GUI auf 50 Stellen, einstellbar, feste Nachkommastellen, exakte Genauigkeit	variabel
Float	FLOAT	(Double wird stattdessen genutzt)	Einstellbar, nicht exakt, 7 Dezimalstellen maximal	4 Byte
Real	REAL	REAL Firebird: BLOB		
Double	DOUBLE	DOUBLE [PRECISION] | FLOAT	Einstellbar, nicht exakt, 15 Dezimalstellen maximal	8 Byte

Text

Typ	Zusatz	HSQLDB	Umfang	Speicherbedarf
Text	VARCHAR	VARCHAR	Einstellbar FIREBIRD: 32000, in der internen Version 8000 Zeichen	variabel
Text	VARCHAR_IGNORECASE	VARCHAR_IGNORECASE ~~FIREBIRD~~	Einstellbar, Auswirkung auf Sortierung, ignoriert Unterschiede zwischen Groß- und Kleinschreibung	variabel
Text (fix)	CHAR	CHAR | CHARACTER	Einstellbar, Rest zum tatsächlichen Text wird mit Leerzeichen aufgefüllt	fest
Memo	LONGVARCHAR	LONGVARCHAR FIREBIRD: CLOB	unbegrenzt	variabel

Zeit

Typ	Zusatz	HSQLDB	Umfang	Speicherbedarf
Datum	DATE	DATE		4 Byte
Zeit	TIME	TIME	FIREBIRD: 1/10000 Sekunden maximal: 23:59:59	4 Byte
Datum/Zeit	TIMESTAMP	TIMESTAMP | DATETIME	HSQLDB: Einstellbar (0, 6 – 6 bedeutet 1/1000Sekunden)	8 Byte

Sonstige

Typ	Zusatz	HSQLDB	Umfang	Speicherbedarf
Ja/Nein	BOOLEAN	BOOLEAN | BIT		
Binärfeld (fix)	BINARY	BINARY	Wie Integer	fest
Binärfeld	VARBINARY	VARBINARY	Wie Integer	variabel
Bild FIREBIRD: BLOB[53]	LONGVARBINARY FIREBIRD: BLOB	LONGVARBINARY FIREBIRD: BLOB	Wie Integer	variabel, für größere Bilder gedacht
OTHER	OTHER	OTHER | OBJECT		

In den Tabellendefinitionen und bei der Änderung von Datentypen in Abfragen mit den Funktionen *CONVERT* oder *CAST* werden bei einigen Datentypen Informationen zur Anzahl an Zeichen (a), zur Genauigkeit (g, entspricht der Gesamtzahl an Ziffern) und zu den Dezimalstellen (d) erwartet: CHAR(a), VARCHAR(a), DOUBLE(g), NUMERIC(g,d), DECIMAL(g,d) und TIMESTAMP(g).

53 In Firebird funktioniert nur der Feldtyp BLOB [BLOB] korrekt mit dem grafischen Kontrollfeld von Formularen.

HSQLDB: TIMESTAMP(g) kann nur die Werte '0' oder '6' annehmen. Die Genauigkeit des Timestamps kann *nur direkt über den SQL-Befehl* eingegeben werden. Sollen also Zeitangaben im Sportbereich eingetragen werden, so ist TIMESTAMP(6) über **Extras → SQL** voreinzustellen.

Bei der Migration von HSQLDB nach FIREBIRD wird aus dem Feld für ein Bild ein Datentyp **`Bild [BLOB]`** erstellt. Dieser Datentyp arbeitet leider nicht korrekt mit dem Grafischen Kontrollfeld zusammen, so dass die Bilder nicht mehr angezeigt und auch nicht vergrößert dargestellt werden können. Abhilfe: Der Name der importierten Tabelle wird geändert und die gesamte Tabelle wird kopiert. Beim Einfügen wird die Tabelle mit altem Namen neu erstellt. Jetzt wird beim Datentyp für das Bild der Datentyp **`Blob [BLOB]`** gewählt. Dieser Datentyp zeigt Bilder (und ggf. auch andere Dokumente) wieder korrekt an.

Datentypen in StarBasic

Zahlen

Typ	***Entspricht in HSQLDB***	***Startwert***	***Anmerkung***	***Speicherbedarf***
Integer	SMALLINT	0	2^{16} = - 32768 bis + 32767	2 Byte
Long	INTEGER	0	2^{32} = - 2147483648 bis + 2147483647	4 Byte
Single		0.0	Dezimaltrenner: «.»	4 Byte
Double	DOUBLE	0.0	Dezimatrenner: «.»	8 Byte
Currency	Ähnlich DECIMAL, NUMERIC	0.0000	Währung, 4 Dezimalstellen fest	8 Byte

Sonstige

Typ	***Entspricht in HSQLDB***	***Startwert***	***Anmerkung***	***Speicherbedarf***
Boolean	BOOLEAN	False	1 = „ja“, alles andere: „nein“	1 Byte
Date	TIMESTAMP	00:00:00	Datum und Zeit	8 Byte
String	VARCHAR	Leerer String	bis 65536 Zeichen	variabel
Object	OTHER	Null		variabel
Variant		Leer	Kann jeden (anderen) Datentyp annehmen	variabel

Vor allem bei Zahlenwerten besteht große Verwechselungsgefahr. In der Datenbank steht z.B. häufig im Primärschlüssel der Datentyp «INTEGER». Wird jetzt per Makro ausgelesen, so muss dort die aufnehmende Variable den Typ «Long» haben, da diese vom Umfang her mit «INTEGER» aus der Datenbank übereinstimmt. Der entsprechende Auslesebefehl heißt dann auch «getLong».

Eingebaute Funktionen und abgespeicherte Prozeduren

In der eingebauten HSQLDB bzw. in Firebird sind die folgenden Funktionen verfügbar. Ein paar Funktionen können leider nur dann genutzt werden, wenn in der Abfrage «SQL-Kommando direkt ausführen» gewählt wurde. Dies verhindert dann gleichzeitig, dass die Abfragen editierbar bleiben.

Funktionen, die mit der grafischen Benutzeroberfläche zusammenarbeiten, sind gekennzeichnet mit GUI. Funktionen, die nur über «SQL-Kommando direkt ausführen» ansprechbar sind, sind gekennzeichnet mit ~~GUI~~.

Bei den Funktionen gibt es solche, die mit der HSQLDB laufen, nicht aber mit der eingebauten Version von Firebird - und umgekehrt. Ist der Datenbankname grün dargestellt, so ist die Funktion verfügbar und gibt ordnungsgemäße Werte aus. Ist der Datenbankname rot und durchgestrichen dargestellt, so ist die Funktion nicht verfügbar oder aber, was deutlich bedenklicher ist, verfügbar, nur eben mit falschen Ergebnissen. Das ist dann mit dem Zusatz ~~(Bug)~~ gekennzeichnet.

Numerisch

Da hier mit Fließkommazahlen gerechnet wird, empfiehlt es sich, gegebenenfalls auf die Einstellung der Felder bei einer Abfrage zu achten. Meist ist hier die Anzeige der Nachkommazahlen begrenzt, so dass eventuell überraschende Ergebnisse zustande kommen. Dann wird z.B. in Spalte1 0,00, in Spalte2 1000 angezeigt. In Spalte3 soll dann Spalte1 * Spalte2 stehen – dort steht plötzlich 1.

ABS(d) HSQLDB FIREBIRD	Gibt des absoluten Wert einer Zahl wieder, entfernt also ggf. das Minus-Vorzeichen. GUI
ACOS(d) HSQLDB FIREBIRD	Gibt den Arcuscosinus wieder. Bereich: [-1, 1]. GUI
ASIN(d) HSQLDB FIREBIRD	Gibt den Arcussinus wieder. Bereich: [-1, 1]. GUI
ATAN(d) HSQLDB FIREBIRD	Gibt den Arcustangens wieder. GUI
ATAN2(a,b) HSQLDB FIREBIRD	Gibt den Arcustangens über Koordinaten wieder. 'a' ist der Wert der x-Achse, 'b' der Wert der y-Achse GUI
BITAND(a,b) HSQLDB BIN_AND(a,b) FIREBIRD	Sowohl die binäre Schreibweise von 'a' als auch die binäre Schreibweise von 'b' müssen an der gleichen Stelle eine '1' stehen haben, damit die '1' in das Ergebnis übernommen wird. **BITAND(3,5)** ergibt 1; 0011 AND 0101 = 0001 GUI
BITOR(a,b) HSQLDB BIN_OR(a,b) FIREBIRD	Entweder die binäre Schreibweise von 'a' oder die binäre Schreibweise von 'b' müssen an der gleichen Stelle eine '1' stehen haben, damit die '1' in das Ergebnis übernommen wird. **BITOR(3,5)** ergibt 7; 0011 OR 0101 = 0111 GUI
BIN_SHL(a,b) ~~HSQLDB~~ FIREBIRD	a*2b
BIN_SHR(a,b) ~~HSQLDB~~ FIREBIRD	a/2b

BITXOR(a,b) HSQLDB BIN_XOR(a,b) FIREBIRD	Entweder die binäre Schreibweise von 'a' oder die binäre Schreibweise von 'b', nicht aber beide, müssen an der gleichen Stelle eine '1' stehen haben, damit die '1' in das Ergebnis übernommen wird. **BITXOR(3,5)** ergibt 6; 0011 XOR 0101 = 0110 GUI
CEIL(d) ~~HSQLDB~~ FIREBIRD CEILING(d) HSQLDB FIREBIRD	Gibt die kleinste Ganzzahl an, die nicht kleiner als d ist. GUI
COS(d) HSQLDB FIREBIRD	Gibt den Cosinus wieder. GUI
COSH(d) ~~HSQLDB~~ FIREBIRD	Gibt den Cosinus-Hyperbolicus wieder. GUI
COT(d) HSQLDB FIREBIRD	Gibt den Cotangens wieder. GUI
DEGREES(d) HSQLDB ~~FIREBIRD~~	Gibt zu Bogenmaßen die Gradzahl wieder. GUI
EXP(d) HSQLDB FIREBIRD	Gibt ed (e: (2.718...)) wieder. GUI
FLOOR(d) HSQLDB FIREBIRD	Gibt die größte Ganzzahl an, die nicht größer als d ist. GUI
LOG(d) HSQLDB ~~FIREBIRD~~ LN(d) ~~HSQLDB~~ FIREBIRD	Gibt den natürlichen Logarithmus zur Basis 'e' wieder. GUI
LOG(x,y) ~~HSQLDB~~ FIREBIRD	Gibt den Logarithmus zur Basis x wieder. **LOG(2,8)** ergibt 3, weil $2^3 = 8$ GUI
LOG10(d) HSQLDB FIREBIRD	Gibt den Logarithmus zur Basis 10 wieder. GUI
MOD(a,b) HSQLDB FIREBIRD	Gibt den Rest als Ganzzahl wieder, der bei der Division von 2 Ganzzahlen entsteht. **MOD(11,3)** ergibt 2, weil 3*3+2=11 GUI
PI() HSQLDB FIREBIRD	Gibt π (3.1415...) wieder. GUI
POWER(a,b) HSQLDB FIREBIRD	ab , **POWER(2,3)** = 8, weil $2^3 = 8$ GUI
RADIANS(d) HSQLDB ~~FIREBIRD~~	Gibt zu den Gradzahlen das Bogenmaß wieder. GUI
RAND() HSQLDB FIREBIRD	Gibt eine Zufallszahl x größer oder gleich 0.0 und kleiner als 1.0 wieder. GUI
ROUND(a,b) HSQLDB FIREBIRD	Rundet a auf b Stellen nach dem Dezimalzeichen. GUI
ROUNDMAGIC(d) HSQLDB ~~FIREBIRD~~	Löst Rundungsprobleme, die durch Fließkommazahlen entstehen. 3.11-3.1-0.01 ist eventuell nicht genau 0, wird aber als 0 in der GUI angezeigt. ROUNDMAGIC macht daraus einen tatsächlichen 0-Wert. GUI

SIGN(d) HSQLDB FIREBIRD	Gibt -1 wieder, wenn 'd' kleiner als 0 ist, 0 wenn 'd' = 0 und 1 wenn 'd' größer als 0 ist. GUI
SIN(A) HSQLDB FIREBIRD	Gibt den Sinus eines Bogenmaßes wieder. GUI
SINH(d) ~~HSQLDB~~ FIREBIRD	Gibt den Sinus-Hyperbolicus wieder. GUI
SQRT(d) HSQLDB FIREBIRD	Gibt die Quadratwurzel wieder. GUI
TAN(A) HSQLDB FIREBIRD	Gibt den Tangens eines Bogenmaßes wieder. GUI
TANH(d) ~~HSQLDB~~ FIREBIRD	Gibt den Tangens-Hyperbolicus wieder. GUI
TRUNCATE(a,b) HSQLDB ~~FIREBIRD~~ TRUNC(a,b) ~~HSQLDB~~ FIREBIRD	Schneidet 'a' auf 'b' Zeichen nach dem Dezimalpunkt ab. TRUNCATE(2.37456,2) = 2.37 GUI
Text	
ASCII(s) HSQLDB ASCII_VAL(s) FIREBIRD	Gibt den ASCII-Code des ersten Buchstaben des Strings wieder. GUI
BIT_LENGTH(s) HSQLDB FIREBIRD	Gibt die Länge des Textes s in Bits wieder. HSQLDB: Jedes Zeichen wird als 16bit ausgegeben. Firebird: Normale Zeichen 8bit, Sonderzeichen 16bit. GUI
CHAR(c) HSQLDB ASCII_CHAR(c) FIREBIRD	Gibt den Buchstaben wieder, der zu dem ASCII-Code c gehört. Dabei geht es nicht nur um Buchstaben, sondern auch um Steuerzeichen. **`ASCII_CHAR()`** ist auf die Zahlen 0 – 255 begrenzt. **`CHAR(13)`** erzeugt in einer Abfrage einen Zeilenumbruch, der in mehrzeiligen Feldern eines Formulars oder in Berichten sichtbar wird. GUI
CHAR_LENGTH(s) CHARACTER_LENGTH() HSQLDB FIREBIRD	Gibt die Länge des Textes in Buchstaben wieder. GUI
CHAR_TO_UUID (ascii_uuid) ~~HSQLDB FIREBIRD (BUG)~~	Ein String mit 36 Zeichen und einem '-' an Position 9, 14, 19 und 24 und ansonsten gültigen Hexadezimalzeichen wird in einen String ohne '-' und in Großschreibung geändert. CHAR_TO_UUID('A0bF4E45-3029-2a44-D493-4998c9b439A3') soll ergeben: A0BF4E4530292A44D4934998C9B439A3 GUI
CONCAT(str1,str2) HSQLDB ~~FIREBIRD~~	Verbindet str1 + str2 GUI
'str1'||'str2'||'str3' HSQLDB FIREBIRD	Verbindet str1 + str2 + str3, einfachere Alternative zu CONCAT GUI
'str1'+'str2'+'str3' HSQLDB ~~FIREBIRD~~	Verbindet str1 + str2 + str3, einfachere Alternative zu CONCAT GUI

DIFFERENCE(s1,s2) HSQLDB ~~FIREBIRD~~	Gibt den «Klangunterschied» zwischen s1 und s2 wieder. Hier wird lediglich eine Ganzzahl ausgegeben. 0 bedeutet dabei gleichen Klang. So erscheint 'for' und 'four' mit 0 gleich, 'Kürzen' und 'Würzen' wird auf 1 gesetzt, 'Mund' und 'Mond' wieder auf 0 GUI

GEN_UUID
~~HSQLDB~~ FIREBIRD
	Ergibt einen einzigartigen String mit 16 Byte Länge erstellen.
	`004 SELECT UUID_TO_CHAR(GEN_UUID()) AS "UUID"`
	`005 FROM "RDB$DATABASE"`
	GUI

HASH(s) ~~HSQLDB~~ FIREBIRD	Erstellt einen Hash (Streuwert) für einen eingegebenen Text. Das Ergebnis ist eine BIGINT-Zahl. GUI
HEXTORAW(s) HSQLDB ~~FIREBIRD~~	Übersetzt Hexadezimalcode in andere Zeichen. Der Hexadezimalcode muss dabei aus einer durch 4 teilbaren Anzahl an Zeichen bestehen. **`HEXTORAW('0041')`** = 'A' (siehe auch den Hexadezimalcode über **Einfügen → Sonderzeichen** im Writer) GUI
INSERT(s1,start,len,s2) HSQLDB ~~FIREBIRD~~ OVERLAY(s1 PLACING s2 FROM start [FOR len]) ~~HSQLDB~~ FIREBIRD	Gibt einen Text wieder, bei dem Teile ersetzt werden. Beginnend mit «start» wird über eine Länge «len» aus dem Text s1 Text ausgeschnitten und durch den Text s2 ersetzt. **`INSERT( 'Bundesbahn', 3, 4, 'mmel')`** macht aus 'Bundesbahn' 'Bummelbahn', wobei die Länge des eingefügten Textes auch ohne weiteres größer als die des ausgeschnittenen Textes sein darf. So ergibt **`INSERT( 'Bundesbahn', 3, 5, 's und B')`** oder **`OVERLAY( 'Bundesbahn' PLACING 's und B' FROM 3 FOR 5)`** 'Bus und Bahn'. GUI (HSQLDB) ~~GUI (FIREBIRD)~~
LCASE(s) HSQLDB ~~FIREBIRD~~	Wandelt den String in Kleinbuchstaben um. GUI
LEFT(s,count) HSQLDB FIREBIRD	Gibt die mit count angegebene Zeichenanzahl vom Beginn des Textes s wieder. GUI
LENGTH(s) HSQLDB ~~FIREBIRD~~	Gibt die Länge eines Textes in Anzahl der Buchstaben wieder. GUI
LIST ([ALL | DISTINCT] expression [, separator]) ~~HSQLDB~~ FIREBIRD	Fasst alle Einträge zu einem Feldinhalt zusammen, die in einem Feld (in Abhängigkeit von der Gruppierung anderer Felder) stehen. **`SELECT "Nachname", CAST( LIST( "Vorname", ', ' ) AS VARCHAR ( 8000 ) ) AS "Vornamen" FROM "Tabelle" GROUP BY "Nachname"`** Die Umformung in den VARCHAR-Dateityp war bei Einführung von LO 5.3 noch notwendig. Mit LO 5.3.1 war dieser Bug behoben. GUI
LOCATE(search,s[,start]) HSQLDB ~~FIREBIRD~~	Gibt den ersten Treffer für den Begriff aus search in dem Text s wieder. Der Treffer wird numerisch angegeben: (1=left, 0=not found) Die Angabe eines Startes innerhalb des Textes ist optional. GUI
LOWER(s) HSQLDB FIREBIRD	Wie LCASE(s) GUI

LPAD(s,len[,pad]) ~~HSQLDB~~ FIREBIRD	Der String s wird mit den in pad angegebenen Zeichen bis zur Länge len von links an aufgefüllt. Ist pad nicht angegeben, so wird ein Leerzeichen genutzt. Ist s länger als len, so wird s auf len von rechts aus gekürzt. GUI
LTRIM(s) HSQLDB ~~FIREBIRD~~	Entfernt führende Leerzeichen und nicht druckbare Zeichen von einem Text. GUI
OCTET_LENGTH(s)	Gibt die Länge eines Textes in Bytes an. Dies entspricht dem doppelten Wert der Zeichenanzahl. HSQLDB Dies entspricht dem UTF8-Wert der Zeichenanzahl. Sonderzeichen haben eine Länge von 2. FIREBIRD GUI
POSITION(s1 IN s2) HSQLDB FIREBIRD POSITION(s1, s2[, start]) ~~HSQLDB~~ FIREBIRD	Wenn der erste Text in dem zweiten enthalten ist wird die Position des ersten Textes wiedergeben, ansonsten 0. Bei der 2. Version kann auch der Startpunkt der Suche festgelegt werden. Dies könnte statt einer Suchmöglichkeit mit «LIKE» besonders bei umfangreichen Texten genutzt werden. GUI
RAWTOHEX(s) HSQLDB ~~FIREBIRD~~	Verwandelt in die Hexadezimalschreibweise, Umkehr von HEXTO-RAW() GUI
REPEAT(s,count) HSQLDB ~~FIREBIRD~~	Wiederholt den Text s count Mal GUI
REPLACE(s1,replace,s2) HSQLDB FIREBIRD	Ersetzt alle vorkommenden Textstücke mit dem Inhalt replace im Text s1 durch den Text s2 GUI
REVERSE(s) ~~HSQLDB~~ FIREBIRD	Schreibt den String verkehrt herum. GUI
RIGHT(s,count) HSQLDB FIREBIRD	Umgekehrt zu LEFT; gibt die mit count angegebene Zeichenzahl vom Textende aus wieder. GUI
RPAD(s,len[,pad]) ~~HSQLDB~~ FIREBIRD	Der String s wird mit den in pad angegebenen Zeichen bis zur Länge len von rechts an aufgefüllt. Ist pad nicht angegeben, so wird ein Leerzeichen genutzt. Ist s länger als len, so wird s auf len von rechts aus gekürzt. GUI
RTRIM(s) HSQLDB ~~FIREBIRD~~	Entfernt alle Leerzeichen und nicht druckbaren Zeichen am Tex-tende. GUI
SOUNDEX(s) HSQLDB ~~FIREBIRD~~	Gibt einen Code von 4 Zeichen wieder, die dem Klang von s ent-sprechen sollen – passt zu der Funktion DIFFERENCE() GUI
SPACE(count) HSQLDB ~~FIREBIRD~~	Gibt die in count angegebene Zahl an Leertasten wieder. GUI
SUBSTR(s,start[,len]) HSQLDB ~~FIREBIRD~~	Kürzel für SUBSTRING GUI

SUBSTRING(s,start[,len]) HSQLDB ~~FIREBIRD~~ SUBSTRING(s FROM start [FOR len]) HSQLDB FIREBIRD	Gibt den Text s ab der Startposition wieder. (1=links) . Wird die Länge ausgelassen, so wird der gesamte Text wiedergegeben. Auch bei einer größeren Länge als der Textlänge wird der restliche Text wiedergegeben. HSQLDB: Startposition < 0 beginnt von rechts rückwärts. GUI Liefert den Teil eines Textes ab der in FROM angegebenen Start-position, optional in der in FOR angegebenen Länge. Steht im Feld "Name" z.B. 'Roberta', so ergibt **SUBSTRING("Name" FROM 3 FOR 3)** den Teilstring 'bert'. FIREBIRD: Startposition muss > 0 sein. GUI
TRIM([{LEADING | TRAI-LING | BOTH}] FROM s) HSQLDB FIREBIRD	Nicht druckbare Sonderzeichen und Leerzeichen werden entfernt. GUI
UCASE(s) HSQLDB ~~FIREBIRD~~	Wandelt den String in Großbuchstaben um. GUI
UPPER(s) HSQLDB FIREBIRD	Wie UCASE(s) GUI
UUID_TO_CHAR ~~HSQLDB~~ FIREBIRD	Macht aus einer 16-Byte UUID eine 36 Zeichen lange UUID-Kette mit Bindestrichen. Sonderzeichen haben eine Länge von 2 Byte, so dass dafür die Zeichenzahl gekürzt werden müsste. **UUID_TO_CHAR('LibreOffice Base')** ergibt 4C696272-654F-6666-6963-652042617365 GUI
Datum/Zeit	
CURDATE() HSQLDB ~~FIREBIRD~~	Gibt das aktuelle Datum wieder. GUI
CURRENT_DATE HSQLDB FIREBIRD	Synonym für CURDATE(), SQL-Standard GUI
CURTIME() HSQLDB ~~FIREBIRD~~	Gibt die aktuelle Zeit wieder. GUI
CURRENT_TIME HSQLDB FIREBIRD	Synonym für CURTIME(), SQL-Standard GUI
CURRENT_TIMESTAMP HSQLDB FIREBIRD	Synonym für NOW(), SQL-Standard, FIREBIRD liefert eine Genauigkeit von 1/1000 Sekunden GUI

DATEADD(string, n, datetime) DATEADD(n string TO datetime) ~~HSQLDB~~ FIREBIRD	Addiert zu einer Datums bzw. Datumszeitangabe eine entsprechende Zeit, die über die Ganzzahl n und den string festgelegt wird. Der Eintrag in string entscheidet darüber, in welcher Einheit der Unterschied wiedergegeben wird: millisecond, second, minute, hour, day, week, month, year. Die string-Eingaben dürfen nicht mit einfachen Anführungszeichen maskiert sein. ~~GUI~~ Alternativ ist für Tagesangaben in FIREBIRD möglich: "Datum" + "Zeit" Timestamp GUI "Datum" + 1 Datum, Integerzahl wird als Tag gewertet GUI "Datum" – 1 Datum, Integerzahl wird als Tag gewertet GUI "Zeit" + 1 Zeit, Integerzahl wird als Sekunde gewertet GUI "Zeit" – 1 Zeit, Integerzahl wird als Sekunde gewertet GUI "Zeitstempel" + 2.75 Zeitstempel, Ganzzahl wird als Datum, Nachkommastellen als Bruchteil des Tages gewertet, hier also 2 Tage und 18 Stunden GUI "Zeitstempel" – 2.75 Zeitstempel GUI Nur als Differenz, nicht als Summe funktionieren: "Datum1" – "Datum2" Differenz in Tagen GUI "Zeit1" – "Zeit2" Differenz in Sekunden GUI "Zeitstempel1" – "Zeitstempel2" Differenz in Tagen und Bruchteilen von Tagen GUI Besonderheit: CURRENT_DATE – "Datum"~~GUI~~ CAST(CURRENT_DATE AS DATE) – "Datum" GUI
DATEDIFF(string, datetime1, datetime2) HSQLDB FIREBIRD DATEDIFF(string FROM datetime1 TO datetime2) ~~HSQLDB~~ FIREBIRD	Datumsunterschied zwischen zwei Datums- bzw. Datumszeitangaben. HSQLDB: Der Eintrag in string entscheidet darüber, in welcher Einheit der Unterschied wiedergegeben wird: 'ms'='millisecond', 'ss'='second', 'mi'='minute', 'hh'='hour', 'dd'='day', 'mm'='month', 'yy' = 'year'. Sowohl die Langfassung als auch die Kurzfassung ist für den einzusetzenden string möglich. GUI Firebird: Der Eintrag in string entscheidet darüber, in welcher Einheit der Unterschied wiedergegeben wird: millisecond, second, minute, hour, day, week, month, year. Die string-Eingaben dürfen nicht mit einfachen Anführungszeichen maskiert sein. ~~GUI~~
DAY(date) HSQLDB ~~FIREBIRD~~	Gibt den Tag im Monat wieder. (1-31) GUI
DAYNAME(date) HSQLDB ~~FIREBIRD~~	Gibt den englischen Namen des Tages wieder. GUI
DAYOFMONTH(date) HSQLDB ~~FIREBIRD~~	Gibt den Tag im Monat wieder. (1-31), Synonym für DAY() GUI
DAYOFWEEK(date) HSQLDB ~~FIREBIRD~~	Gibt den Wochentag als Zahl wieder. (1 bedeutet Sonntag) GUI
DAYOFYEAR(date) HSQLDB ~~FIREBIRD~~	Gibt den Tag im Jahr wieder. (1-366) GUI
HOUR(time) HSQLDB ~~FIREBIRD~~	Gibt die Stunde wieder. (0-23) GUI

MINUTE(time) HSQLDB ~~FIREBIRD~~	Gibt die Minute wieder. (0-59) GUI
MONTH(date) HSQLDB ~~FIREBIRD~~	Gibt den Monat wieder. (1-12) GUI
MONTHNAME(date) HSQLDB ~~FIREBIRD~~	Gibt den englischen Namen des Monats wieder. GUI
NOW() HSQLDB ~~FIREBIRD~~ DATE 'NOW', TIME 'NOW', TIMESTAMP 'NOW' ~~HSQLDB~~ FIREBIRD	Gibt das aktuelle Datum und die aktuelle Zeit zusammen als Zeitstempel wieder. Stattdessen kann auch CURRENT_TIMESTAMP genutzt werden. GUI Kurzform **`TIMESTAMP 'NOW'`** usw.: ~~GUI~~ **`Time 'Now'`** liefert in FIREBIRD eine Genauigkeit von 1/1000 Sekunden Langform **`CAST('NOW' AS TIMESTAMP)`** usw.: GUI
QUARTER(date) HSQLDB ~~FIREBIRD~~	Gibt das Quartal im Jahr wieder. (1-4) GUI
SECOND(time) HSQLDB ~~FIREBIRD~~	Gibt die Sekunden einer Zeitangabe wieder. (0-59) GUI
WEEK(date) HSQLDB ~~FIREBIRD~~	Gibt die Woche des Jahres wieder. (1-53) GUI
YEAR(date) HSQLDB ~~FIREBIRD~~	Gibt das Jahr aus einer Datumseingabe wieder. GUI
TODAY HSQLDB ~~FIREBIRD~~	Synonym für CURDATE() GUI
DATE 'TODAY' ~~HSQLDB~~ FIREBIRD	Gibt das aktuelle Datum an, Langform: CAST('TODAY' AS DATE) GUI (Langform) ~~GUI~~ (Kurzform)
DATE 'YESTERDAY' ~~HSQLDB~~ FIREBIRD	Gibt das Datum von Gestern an, Langform: CAST('YESTERDAY' AS DATE) GUI (Langform) ~~GUI~~ (Kurzform)
DATE 'TOMORROW' ~~HSQLDB~~ FIREBIRD	Gibt das Datum von Morgen an, Langform: CAST('TOMORROW' AS DATE) GUI (Langform) ~~GUI~~ (Kurzform)

TO_CHAR(datetime, string) HSQLDB ~~FIREBIRD~~	Setzt eine Datums- bzw. Timestampeingabe in einen entsprechenden String um. Folgende Zeichen sind definiert: 'Y' bis 'YYYY' (Jahr, 1 bis 4 Stellen) 'IY' bis 'IYYY' (Jahr nach ISO, 2 bis 4 Stellen) 'MM' (Monat, zweistellig) 'MON' (Monatsname, Kurzfassung, berücksichtigt GUI-Sprache) 'MONTH' (Monatsname, Langfassung) 'w' (Woche des Jahres) 'W' (Woche des Monats) 'IW' (Woche des Jahres nach ISO-Standard) 'd' (Tag der Woche) 'D' (Tagesname, Kurzfassung, berücksichtigt GUI-Sprache) 'DD' oder 'dd' (Tag des Monats, zweistellig) 'H' (Stunde von 0-23) 'HH' (Stunde von 0-11) 'm' (Minute) 's' (Sekunde) 'a' (AM oder PM für Stunde von 0-11) viele andere Zeichen werden direkt übernommen: **`TO_CHAR(CURRENT_DATE,'D, DD.MON.YYYY')`** = Sa, 05.Nov.2016 Die Funktion läuft nur mit bereits eingegebenen Datumswerten, die aus der Datenbank gelesen werden. GUI
EXTRACT(string FROM datetime) HSQLDB FIREBIRD	Kann viele der Datums- und Zeitfunktionen ersetzen. Gibt das Jahr, den Monat, den Tag usw. von einem Datums- bzw. Datumszeitwert wieder. **`EXTRACT(DAY FROM "Datum")`** gibt den Tag im Monat wieder. HSQLDB: YEAR, MONTH, DAY, HOUR, MINUTE, SECOND Firebird zusätzlich: WEEK, WEEKDAY, YEARDAY, MILLISECOND (~~GUI~~). GUI **Hinweis:** Die EXTRACT-Funktion scheint für die GUI bessere Ergebnisse zu liefern als die Kurzfunktionen YEAR(). Bei einer Formularkonstruktion, in der Formular und Unterformular über Felder verbunden wurden, die beide über YEAR("Datum") erstellt wurden, erschien im Unterformular kein Wert. Erst mit EXTRACT(YEAR FROM "Datum") arbeitete das Unterformular wie gewünscht.

Datenbankverbindung

DATABASE() HSQLDB ~~FIREBIRD~~	Gibt den kompletten Pfad und Namen der Datenbank, die zu dieser Verbindung gehört, wieder. GUI
USER() HSQLDB ~~FIREBIRD~~ USER ~~HSQLDB~~ FIREBIRD	Gibt den Benutzernamen dieser Verbindung wieder. Der Nutzername ist dann von Bedeutung, wenn die Datenbank in eine externe Datenbank umgewandelt werden soll. Standard in HSQLDB: SA Standard in Firebird: SYSDBA ~~GUI (HSQLDB)~~ GUI (FIREBIRD)
CURRENT_USER HSQLDB FIREBIRD	SQL Standardfunktion, Synonym für USER(). Zu beachten ist, dass hier keine Klammern zu setzen sind. GUI

CURRENT_CONNECTION ~~HSQLDB~~ FIREBIRD	Gibt einen INTEGER-Wert für die aktuelle Datenbankverbindung wieder. ~~GUI~~
CURRENT_ROLE ~~HSQLDB~~ FIREBIRD	Gibt in der Regel NONE für die aktuelle Datenbankverbindung wieder. Funktioniert nicht bei der Abfrage von Tabellen, sondern nur bei direkter Abfrage an die Datenbank selbst. ~~GUI~~
CURRENT_TRANSACTION ~~HSQLDB~~ FIREBIRD	Gibt die Transaktionsnummer für die aktuelle Datenbankverbindung wieder. Funktioniert nicht bei der Abfrage von Tabellen, sondern nur bei direkter Abfrage an die Datenbank selbst. ~~GUI~~
IDENTITY() HSQLDB ~~FIREBIRD~~	Gibt den letzten Wert für ein Autowertfeld wieder, das in der aktuellen Verbindung erzeugt wurde. Dies wird bei der Makroprogrammierung genutzt, um aus einem erstellten Primärschlüssel für eine Tabelle einen Fremdschlüssel für eine andere Tabelle zu erstellen. GUI
RDB$GET_CONTEXT('name', 'varname') ~~HSQLDB~~ FIREBIRD	Als 'name' können angegeben werden: SYSTEM, USER_SESSION, USER_TRANSACTION. USER_SESSION und USER_TRANSACTION sind zum Start leer und können mit beliebigen Variablen belegt werden. SYSTEM hat folgende vorbelegten 'varname': DB_NAME Pfad zu der ausgepackten Firebird-Datenbank NETWORK_PROTOCOL 'TCPv4', 'WNET', 'XNET' oder NULL CLIENT_ADDRESS abhängig von PROTOCOL, intern NULL CURRENT_USER wie CURRENT_USER direkt CURRENT_ROLE wie CURRENT_ROLE direkt SESSION_ID wie CURRENT_CONNECTION direkt TRANSACTION_ID wie CURRENT_TRANSACTION direkt ISOLATION_LEVEL 'READ COMMITTED', 'SNAPSHOT' oder 'CONSISTENCY'. ENGINE_VERSION Firebird-Server, LO 5.3: '3.0.0' ~~GUI~~
RDB$SET_CONTEXT ('name', 'varname', wert | NULL) ~~HSQLDB~~ FIREBIRD	Ist das Gegenstück zu RDB$GET_CONTEXT(). Als 'name' können aber nur USER_SESSION und USER_TRANSACTION angegeben werden. SYSTEM kann nicht beschrieben werden. **RDB$SET_CONTEXT('USER_SESSION', 'Autor', 'Robert')** schreibt die Variable über eine Abfrage nach USER_SESSION. In der aktuellen Zeile der Abfrage ist der Wert der Variablen noch NULL oder eben der vorhergehende Wert. mit **RDB$GET_CONTEXT('USER_SESSION', 'Autor')** ist in den folgenden Zeilen der Abfrage der Wert 'Robert' verfügbar. ~~GUI~~

System	
IFNULL(exp,value) HSQLDB ~~FIREBIRD~~	Wenn exp NULL ist, wird value zurückgegeben, sonst exp. Stattdessen kann als Erweiterung auch COALESCE() genutzt werden. Exp und value müssen den gleichen Datentyp haben. IFNULL ist eine wichtige Funktion, wenn Felder durch Rechnung oder CONCAT miteinander verbunden werden. Der Inhalt des Ergebnisses wäre NULL, wenn auch nur ein Wert NULL ist. **`"Nachname"||', '||"Vorname"`** würde für Personen, bei denen z.B. der Eintrag für "Vorname" fehlt, ein leeres Feld, also NULL ergeben. **`"Nachname"||IFNULL(', '||"Vorname",'')`** würde stattdessen auch nur "Nachname" ausgeben. GUI
CASEWHEN(exp,v1,v2) HSQLDB ~~FIREBIRD~~ IIF(exp,v1,v2) ~~HSQLDB~~ FIREBIRD	Wenn exp wahr ist wird v1 zurückgegeben, sonst v2. Stattdessen kann auch CASE WHEN genutzt werden. **`CASEWHEN("a">10,'Ziel erreicht','noch üben')`** gibt 'Ziel erreicht' aus, wenn der Inhalt des Feldes "a" größer als 10 ist. GUI
CONVERT(term,type) HSQLDB ~~FIREBIRD~~ CAST(term AS type) HSQLDB FIREBIRD	Wandelt term in einen anderen Datentyp um. **`CONVERT("a",DECIMAL(5,2))`** macht aus dem Feld "a" ein Feld mit 5 Ziffern, davon 2 Nachkommastellen. Ist die Zahl zu groß, so wird ein Fehler ausgegeben. GUI
COALESCE(expr1, expr2, expr3,...) HSQLDB FIREBIRD	Wenn expr1 nicht NULL ist, wird expr1 wiedergegeben, ansonsten wird expr2 überprüft, danach dann expr3 usw. Sämtliche Ausdrücke müssen zumindest einen ähnlichen Datentyp haben. So geht die alternative Darstellung von Ganzzahlen und Fließkommazahlen, aber nicht auch noch des eines Datums- oder Zeitwertes. **`COALESCE("Spitzname", "Vorname", 'Herrn/Frau')`** wählt den Spitznamen, wenn dieser vorhanden ist, ansonsten den Vornamen und wenn dieser auch nicht bekannt ist, dann allgemein 'Herrn/Frau'. GUI
NULLIF(v1,v2) HSQLDB FIREBIRD	Wenn v1 gleich v2 ist, wird NULL wiedergegeben, ansonsten v1. Die Daten müssen vom Typ her vergleichbar sein. GUI
CASE v1 WHEN v2 THEN v3 [ELSE v4] END HSQLDB FIREBIRD DECODE(v1, v2, v3, v4) ~~HSQLDB~~ FIREBIRD	Wenn v1 gleich v2 ist, wird v3 wiedergegeben. Sonst wird v4 wiedergegeben oder NULL, wenn kein ELSE formuliert ist. ~~GUI (HSQLDB)~~ GUI (FIREBIRD)
CASE WHEN expr1 THEN v1[WHEN expr2 THEN v2] [ELSE v4] END HSQLDB FIREBIRD	Wenn expr1 wahr ist wird v1 zurückgegeben. [Optional können weitere Fälle angegeben werden] Sonst wird v4 wiedergegeben oder NULL, wenn kein ELSE formuliert ist. **`CASE WHEN DAYOFWEEK("Datum")=1 THEN 'Sonntag' WHEN DAYOFWEEK("Datum")=2 THEN 'Montag' … END`** könnte per SQL den Tagesnamen ausgeben, der sonst in der Funktion nur in Englisch verfügbar ist. GUI
MAXVALUE(expr [, expr ...]) ~~HSQLDB~~ FIREBIRD	Sucht den höchsten Wert im Vergleich zu verschiedenen Ausdrücken heraus. Ergibt NULL, wenn ein Wert leer ist. Im Gegensatz zur Aggregatfunktion MAX() können hier die Werte verschiedener Felder miteinander verglichen werden. GUI

MINVALUE(expr [, expr ...]) ~~HSQLDB~~ FIREBIRD	Sucht den niedrigsten Wert im Vergleich zu verschiedenen Ausdrücken heraus. Ergibt NULL, wenn ein Wert leer ist. Im Gegensatz zur Aggregatfunktion MIN() können hier die Werte verschiedener Felder miteinander verglichen werden. GUI

Window-Funktionen bei Firebird

Window-Funktionen[54] filtern nicht die Ergebnismenge einer Abfrage. Die Ergebnisse der Funktionen werden zusätzlich in entsprechenden Spalten angegeben.

Window-Funktionen können in der **SELECT**-Liste oder in der **ORDER BY**-Definition vorkommen. In den Window-Funktionen ist eine **OVER**-Klausel zwingend enthalten. In dieser Klausel können Partitionierungen und Sortierungen vorgenommen werden.

<Funktionsname>() OVER (PARTITION BY … | ORDER BY …)

Ranking-Funktionen

RANK() OVER (ORDER BY ...)

ID	Name	Laufzeit	RANK
3	Markus	01:37:32	1
2	Linda	01:39:12	2
5	Klaus	01:39:12	2
1	Erich	01:42:54	4
4	Nina	01:43:17	5

Datensatz 1 von 5

```
SELECT "ID", "Name", "Laufzeit",
RANK() OVER (ORDER BY "Laufzeit") FROM "tbl_Starter"
```

Kann zur Bestimmung der Reihenfolge in einer Datenmenge eingesetzt werden. Gleiche Werte werden dabei mit der gleichen Position versehen. Die Position wird wie bei Sportwettkämpfen erstellt. Richtet sich nach der Sortierung, die angegeben wird.
GUI

RANK() OVER (PARTITION BY ... ORDER BY ...)

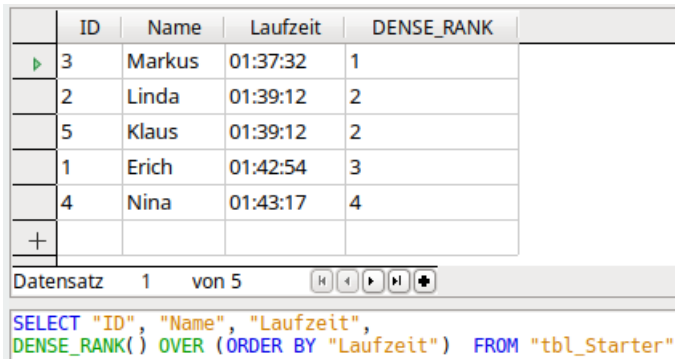
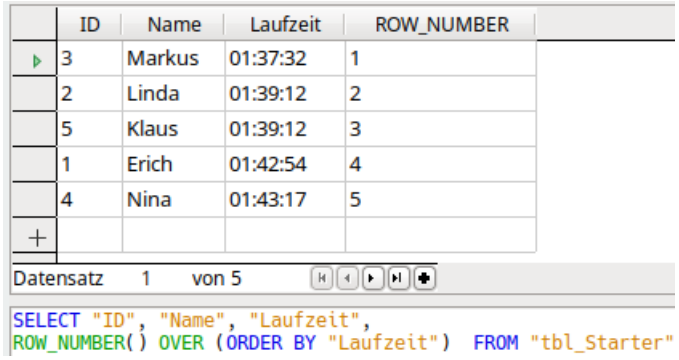
ID	Name	Laufzeit	Geschlecht	RANK
3	Markus	01:37:32	m	1
5	Klaus	01:39:12	m	2
1	Erich	01:42:54	m	3
2	Linda	01:39:12	w	1
4	Nina	01:43:17	w	2

Datensatz 1 von 5

```
SELECT "ID", "Name", "Laufzeit", "Geschlecht",
RANK() OVER (PARTITION BY "Geschlecht" ORDER BY "Laufzeit")
FROM "tbl_Starter"
```

Gleiche Funktion wie oben, nur mit der Ergänzung der Partitionierung. Hier ist die Reihenfolge abhängig vom Feld "Geschlecht".

54 Siehe die Sprachreferenz für Firebird 3.0: *https://www.firebirdsql.org/en/reference-manuals/*

DENSE_RANK() OVER (ORDER BY ...)	 Kann zur Bestimmung der Reihenfolge in einer Datenmenge eingesetzt werden. Gleiche Werte werden dabei mit der gleichen Position versehen. Dabei rücken tiefer liegende Werte auf. Richtet sich nach der Sortierung, die angegeben wird. GUI
ROW_NUMBER() OVER (ORDER BY ...)	 Kann zur Bestimmung der laufenden Nummer in einer Datenmenge eingesetzt werden. Gleiche Werte werden dabei nach dem Primärschlüssel zusätzlich sortiert. Richtet sich nach der Sortierung, die angegeben wird. GUI
Navigationsfunktionen	

FIRST_VALUE(...) OVER (ORDER BY ...)	`001 SELECT "ID", "Name", "Laufzeit",`
	`002 FIRST_VALUE("Laufzeit") OVER (ORDER BY "Laufzeit")`
	`003 FROM "Starter"`
	Hier wird der Wert für die niedrigste Laufzeit neben angezeigten aktuellen Laufzeit angegeben. Es könnte also direkt die Differenz zum ersten Platz ausgerechnet werden.
	GUI

LAG(...,[offset,[default]]) OVER (ORDER BY ...)	`001 SELECT "ID", "Name", "Laufzeit",`
	`002 LAG("Laufzeit") OVER (ORDER BY "Laufzeit")`
	`003 FROM "Starter"`
	Hier wird der Wert für die Laufzeit angezeigt, die die Person gelaufen hat, die direkt vorher ins Ziel gekommen ist. Es könnte also direkt die Differenz zur vorher durchs Zeile gekommenen Person ausgerechnet werden.
	Wird ein **`offset`** angegeben, so kann auch der Abstand zu weiter vorher durchs Ziel gekommenen Personen ermittelt werden. Als **`default`** wird ohne Angabe **`NULL`** gesetzt, ansonsten der korrekt formatierte Wert für das entsprechende Feld, hier z. B. '00:00:00'.
	GUI

LAST_VALUE(...) OVER (ORDER BY ...)	Hier wird der zuletzt ausgelesene Wert wieder gegeben. Dies ist also nicht der Wert für die zuletzt durchs Ziel gekommene Person, sondern der aktuelle Wert, der auch so in der Laufzeit steht. GUI
LEAD(...,[offset,[default]]) OVER (ORDER BY ...)	Dies ist die Umkehrung der Funktion von LAG. Es wird die Zeit angezeigt, die die Person gelaufen ist, die direkt nach der aktuellen Position ins Ziel gekommen ist. Wird ein **offset** angegeben, so kann auch der Abstand zu weiter hinten durchs Ziel gekommenen Personen ermittelt werden. Als **default** wird ohne Angabe **NULL** gesetzt, ansonsten der korrekt formatierte Wert für das entsprechende Feld, hier z. B. '00:00:00'. GUI

NTH_VALUE(...,offset) [FROM {FIRST | LAST}] OVER (ORDER BY ...)	`001 SELECT "ID", "Name", "Laufzeit",`
	`002 NTH_VALUE("Laufzeit",1) OVER (ORDER BY "Laufzeit")`
	`003 FROM "Starter"`
	Ein **offset** muss angegeben werden. Die obige Abfrage gibt mit dem **offset** '1' genau das gleiche Ergebnis wie **FIRST_VALUE**, kann durch das **offset** aber auf die verschiedenen anderen Werte eingestellt werden. Standardmäßig wird vom ersten Wert (**FROM FIRST**) aus ermittelt. Ansonsten vom zuletzt ermittelten Wert.
	GUI

Aggregatfunktionen als Windowfunktionen

Alle Aggregatfunktionen können auch als Windowfunktionen erstellt werden. Damit entfällt die Gruppierungsanweisung (die darüber hinaus die einzelnen Datensätze nicht mehr komplett anzeigt) oder die Notwendigkeit, innerhalb einer Abfrage eine separate Unterabfrage zu erstellen. Alle Aggregatfunktionen können genutzt werden. Hier nur ein Beispiel für die Nutzung von **SUM**.

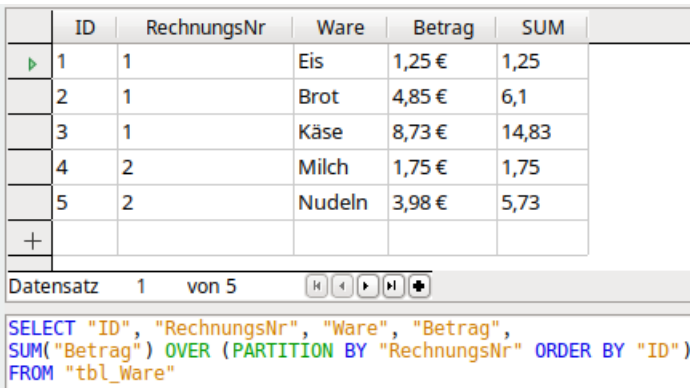
SUM(...) OVER (PARTITION BY ...)

	ID	RechnungsNr	Ware	Betrag	SUM
▶	1	1	Eis	1,25 €	14,83
	2	1	Brot	4,85 €	14,83
	3	1	Käse	8,73 €	14,83
	4	2	Milch	1,75 €	5,73
	5	2	Nudeln	3,98 €	5,73
+					

Datensatz 1 von 5

```
SELECT "ID", "RechnungsNr", "Ware", "Betrag",
SUM("Betrag") OVER (PARTITION BY "RechnungsNr")
FROM "tbl_Ware"
```

Die Summierung erfolgt nach der Ausgabe des gesamten Inhaltes der Tabelle. Keine Feld muss gruppiert werden. Die Summe wird für das Feld "Betrag" abhängig von der "RechnungsNr" berechnet.
GUI

SUM(…) OVER (PARTITION BY … ORDER BY …)	 Wird zusätzlich eine Sortierung vorgenommen, so wird die laufende Summe ermittelt. In der 2. Zeile also die Summe von Zeile 1 und 2, in der 3. Zeile dann die Summe von Zeile 1 und 2 und 3. Solch ein Abfrageergebnis ist sonst nur durch korrelierende Unterabfragen möglich. GUI

Migration HSQLDB → Firebird

Die folgenden Probleme können bei der Migration von Datenbanken der internen HSQLDB 1.8.0.10 zu der internen Firebird 3.0 auftauchen.

- Tabellennamen, Spaltennamen usw. sind in der HSQLDB von der Länge nicht begrenzt. Bei Firebird hingegen gilt eine Grenze von maximal 31 Zeichen.
- Texttabellen können in der HSQLDB zum Lesen, Schreiben und Ändern von Daten genutzt werden. Zur Zeit funktionieren Texttabelle in Firebird hingegen gar nicht, weil die Standardeinstellungen dafür bei der internen Datenbank ausgeschaltet sind. Selbst bei funktionierenden Texttabellen unterscheidet sich das Format der Tabellen, so dass z.B. keine *.csv-Dateien benutzt werden können. Außerdem ist das Ändern und Löschen von Daten in Firebird-Texttabellen nicht erlaubt.
- Unterformulare funktionieren nach der Nutzung des Migrationsassistenten nicht. Hier muss die Datenbankdatei mit einem Packprogramm geöffnet werden. Die content.xml muss zum Bearbeiten geöffnet werden. Der Eintrag **<db:driver-settings** db:system-driver-settings="" db:base-dn="" db:parameter-name-substitution="false"**/>** benötigt statt des **false** ein **true**. Entweder ist hier also **true** einzutragen oder der gesamte Inhalt db:parameter-name-substitution="false" zu löschen.
  Alternativ kann auch das folgende Makro von der migrierten Base-Datei aus gestartet werden:
  ```
  SUB FB_Parameter
     DIM oSettings AS OBJECT
     oSettings = ThisComponent.DataSource.Settings
     oSettings.ParameterNameSubstitution = True
  END SUB
  ```
  Nach einmaligem Start des Makros ist der Eintrag in der Base-Datei komplett verschwunden, wenn die Base-Datei anschließend abgespeichert wird.
- Sollen Bilder in der Datenbank gespeichert werden, so ist der Datentyp **BLOB [BLOB]** statt **Bild [BLOB]** zu währen. Nur Inhalte aus **BLOB [BLOB]** werden im grafischen Kontrollfeld angezeigt.

Hinweis

Eine Umstellung auf Firebird mit dem Migrationsassistenten kann auch wieder rückgängig gemacht werden. Die alte HSQLDB ist weiter in der Datenbankdatei enthalten. Aus
<db:connection-resource xlink:href="sdbc:embedded:firebird"
xlink:type="simple"**/>**
in der content.xml muss wieder
<db:connection-resource xlink:href="sdbc:embedded:hsqldb"
xlink:type="simple"**/>**
erstellt werden.
Nach dieser Änderung kann die Datei «firebird.fbk» in dem Verzeichnis «database» wieder entfernt werden.
Das Ganze geht natürlich auch über ein Makro[55]:

```
001  SUB FirebirdMigrationRemove
002     DIM oDoc AS OBJECT
003     DIM oDataSource AS OBJECT
004     DIM oSettings AS OBJECT
005     DIM oDocumentSubStorage AS OBJECT
006     DIM sURL AS STRING
007     oDoc = ThisComponent
008     sURL = "sdbc:embedded:hsqldb"
009     oDataSource = ThisComponent.DataSource
010     oDataSource.URL = sURL
011     oSettings = oDataSource.Settings
012     oSettings.JavaDriverClassPath = sURL
013     REM 4 = write mode
014     oDocumentSubStorage = oDoc.getDocumentSubStorage("database", 4)
015     oDocumentSubStorage.removeElement("firebird.fbk")
016     oDocumentSubStorage.commit()
017  END SUB
```

Aber Vorsicht: Die interne «firebird.fbk» ist jetzt auf jeden Fall gelöscht.

Im Folgenden sind Funktionen aufgelistet, die in Firebird nicht mit dem gleichen Namen existieren. Die erste Spalte enthält die Funktionsbezeichnung bei der Hsqldb, die zweite Spalte Funktionsbezeichnungen, die alternativ in der Hsqldb und in Firebird verwendet werden können. Existiert so eine Alternative nicht, so muss die dritte Spalte greifen, die Funktionen beschreibt, die in Firebird die gleiche Bedeutung haben wie die Funktionen der Hsqldb in der ersten Spalte.

Numerisch		
HSQLDB	***HSQLDB und Firebird***	***Firebird***
BITAND(a,b)		BIN_AND(a,b)
BITOR(a,b)		BIN_OR(a,b)
BITXOR(a,b)		BIN_XOR(a,b)
DEGREES(d)	<value for radians>*360/(2*PI())	
LOG(d)		LN(d)
RADIANS(d)	<value for degrees>*2*PI()/360	
ROUNDMAGIC(d)		~~Keine Funktion vorhanden~~
TRUNCATE(a,b)		TRUNC(a,b)

TEXT

55 Dank an Ratslinger, *https://ask.libreoffice.org/u/ratslinger*

HSQLDB	HSQLDB und Firebird	Firebird
ASCII(s)		ASCII_VAL(s)
BIT_LENGTH(s)	Gibt die Texlänge wieder. HSQLDB: 16bit pro Zeichen Firebird: 8bit, Sonderzeichen 16bit. Gleiche Funktion, aber unterschiedliche Zählweise	BIT_LENGTH(s)
CHAR(c)		ASCII_CHAR© (nur 0 - 255)
CONCAT(str1,str2) 'str1'+'str2'+'str3'	'str1'||'str2'||'str3'	
DIFFERENCE(s1,s2)		~~Keine Funktion vorhanden~~
HEXTORAW(s)		~~Keine Funktion vorhanden~~
INSERT(s1,start,len,s2)		OVERLAY(s1 PLACING s2 FROM start [FOR len])
LCASE(s)	LOWER(s)	
LENGTH(s)	CHAR_LENGTH(s) CHARACTER_LENGTH()	
LOCATE(search,s[,start])	POSITION(search IN s)	POSITION(search,s[,start])
LTRIM(s)	TRIM(LEADING FROM s)	
OCTET_LENGTH(s)	Gleiche Funktion, aber unterschiedliche Zählweise	OCTET_LENGTH(s)
RAWTOHEX(s)		~~Keine Funktion vorhanden~~
REPEAT(s,count)		~~Keine Funktion vorhanden~~
RTRIM(s)	TRIM(TRAILING FROM s)	
SOUNDEX(s)		~~Keine Funktion vorhanden~~
SPACE(count)		~~Keine Funktion vorhanden~~
SUBSTR(s,start[,len]) SUBSTRING(s,start[,len])	SUBSTRING(s FROM start [FOR len])	
UCASE(s)	UPPER(s)	

Datum/Zeit

HSQLDB	HSQLDB und Firebird	Firebird
CURDATE(), TODAY	CURRENT_DATE	
CURTIME()	CURRENT_TIME	
DATEDIFF(string, datetime1, datetime2) 'ms'='millisecond', 'ss'='second', 'mi'='minute', 'hh'='hour', 'dd'='day', 'mm'='month', 'yy' = 'year'		DATEDIFF(string, datetime1, datetime2) millisecond, second, minute, hour, day, week, month, year Nur die Langversion ohne «''»
DAY(date) DAYOFMONTH(date)	EXTRACT(DAY FROM date-time)	

DAYNAME(date)		~~Keine Funktion vorhanden~~
DAYOFWEEK(date)		EXTRACT(WEEKDAY FROM datetime)
DAYOFYEAR(date)		EXTRACT(YEARDAY FROM datetime)
HOUR(time)	EXTRACT(HOUR FROM datetime)	
MINUTE(time)	EXTRACT(MINUTE FROM datetime)	
MONTH(date)	EXTRACT(MONTH FROM datetime)	
MONTHNAME(date)		~~Keine Funktion vorhanden~~
NOW()	CURRENT_TIMESTAMP	
QUARTER(date)	CEILING(EXTRACT(MONTH FROM date)/3.00)	
SECOND(time)	EXTRACT(SECOND FROM datetime)	
WEEK(date)		EXTRACT(WEEK FROM datetime)
YEAR(date)	EXTRACT(YEAR FROM datetime)	
TO_CHAR(datetime, string)		~~Keine Funktion vorhanden~~
Datenbankverbindung		
HSQLDB	***HSQLDB und Firebird***	***Firebird***
DATABASE()		RDB$GET_CONTEXT('SYSTEM', 'DB_NAME')
USER()	CURRENT_USER	
IDENTITY()		~~Keine Funktion vorhanden~~
System		
HSQLDB	***HSQLDB und Firebird***	***Firebird***
IFNULL(exp,value)	COALESCE(expr1,expr2,expr3,...)	
CASEWHEN(exp,v1,v2)	CASE WHEN expr1 THEN v1[WHEN expr2 THEN v2] [ELSE v4] END	IIF(exp,v1,v2) (nur wenn CASE WHEN zu kompliziert sein sollte)
CONVERT(term,type)	CAST(term AS type)	

Steuerzeichen zur Nutzung in Abfragen

In Abfragen lassen sich Felder miteinander verknüpfen. Aus zwei Feldern in

```
001 SELECT "Vorname", "Nachname" FROM "Tabelle"
```

wird durch

```
001 SELECT "Vorname"||' '||"Nachname" FROM "Tabelle"
```

ein Feld. Hier wird noch ein Leerzeichen mit eingefügt. Natürlich lassen sich hier alle möglichen beliebigen Zeichen einfügen. Solange diese in `''` stehen werden sie als Text interpretiert. Manchmal ist es aber auch sinnvoll, Zeilenumbrüche z.B. für einen Bericht einzufügen. Deshalb hier eine kleine Liste von Steuerzeichen, die entsprechend durch einen Blick auf *http://de.wikipedia.org/wiki/Steuerzeichen* erweitert werden kann.

CHAR(9)	Horizontaler Tabulator	
CHAR(10)	Zeilenvorschub	Erzeugt in Serienbriefen und im Report-Builder in einem Feld einen Zeilenumbruch (Linux, Unix, Mac) Um den Zeilenumbruch auch in Abfragen anzuzeigen, muss das Feld in ein LONGVARCHAR-Feld umgewandelt werden.
CHAR(13)	Wagenrücklauf	Zeilenumbruch zusammen mit dem Zeilenvorschub in Windows. **`CHAR(13)||CHAR(10)`** ist auch für Linux, Mac usw. möglich, daher die universellere Variante.

Bei FIREBIRD ist statt **`CHAR( ) ASCII_CHAR( )`** erforderlich.

Einige uno-Befehle zur Nutzung mit einer Schaltfläche

Einer Schaltfläche können verschiedene uno-Befehle direkt zugeordnet werden. Dafür muss unter **Eigenschaften: Schaltfläche → Aktion → Dokument/Webseite öffnen** gewählt werden sowie z. B. als **URL → .uno:RecSearch** zum Öffnen der Suchfunktion eingetragen werden. Manchmal muss zusätzlich beachtet werden, dass **Fokussieren bei Klick → Nein** angewählt wird, wenn bestimmte Aktionen direkt auf ein Formularfeld zugreifen sollen, das dafür den Focus braucht, wie z. B. **.uno:Paste**, das den Inhalt aus der Zwischenablage einfügen kann.

Die folgende Liste gibt nur wenige Befehle wieder. Sämtliche Befehle aus der Navigationsleiste sind ja bereits in der Schaltfläche so verfügbar, könnten aber auch über die uno-Befehle erstellt werden. Viele Befehle lassen sich auch über den Makrorekorder ermitteln, der häufig diese uno-Befehle über einen Dispatcher aufruft. UNO-Befehle können auch z. B. über **Extras → Anpassen → Symbolleisten → Beschreibung** in jedem LO-Dokument ermittelt werden.

Uno-Befehl	***Anwendung für ...***
.uno:RecSearch	Öffnen der Suchfunktion im Formular
.uno:Paste	Zwischenablage einfügen, nur mit **Fokussieren bei Klick → Nein**
.uno:Copy	Kopiert den markierten Inhalt in die Zwischenablage, nur mit **Fokussieren bei Klick → Nein**
.uno:Print	Druckdialog für das Formular öffnen
.uno:PrintDefault	Drucken mit dem Standarddrucker ohne Dialog
.uno:NextRecord	Nächster Datensatz
.uno:PrevRecord	Vorhergehender Datensatz
.uno:FirstRecord	Erster Datensatz
.uno:LastRecord	Letzter Datensatz
.uno:NewRecord	Neuer Datensatz
.uno:RecSave	Datensatz speichern
.uno:DeleteRecord	Lösche den Datensatz
.uno:RecUndo	Nehme die letzten Eingaben zurück.

.uno:OpenURL	Öffne die angezeigte URL
.uno:Refresh	Angezeigte Daten aktualisieren
.uno:ViewFormAsGrid	Formular: Datenquelle als Tabelle aufrufen

Informationstabellen der HSQLDB

Innerhalb von HSQLDB-Datenbanken wird in dem Bereich "INFORMATION_SCHEMA" die Information über alle Tabelleneigenschaften sowie ihre Verbindung untereinander abgelegt. Diese Informationen ermöglichen in Base bei der Erstellung von Makros, Prozeduren mit weniger Parametern zu starten. Eine Anwendung findet sich in der Beispieldatenbank unter anderem im Modul «Wartung» in der Prozedur «Tabellenbereinigung» für die Ansteuerung des Dialoges.

In einer Abfrage können die einzelnen Informationen sowie sämtliche dazugehörigen Felder auf die folgende Art ermittelt werden.

```
001 SELECT * FROM "INFORMATION_SCHEMA"."SYSTEM_ALIASES"
```

Im Gegensatz zu einer normalen Tabelle ist es hier notwendig, dem jeweiligen folgenden Begriff "INFORMATION_SCHEMA" voranzustellen.

```
SYSTEM_ALIASES
SYSTEM_ALLTYPEINFO
SYSTEM_BESTROWIDENTIFIER
SYSTEM_CACHEINFO
SYSTEM_CATALOGS
SYSTEM_CHECK_COLUMN_USAGE
SYSTEM_CHECK_CONSTRAINTS
SYSTEM_CHECK_ROUTINE_USAGE
SYSTEM_CHECK_TABLE_USAGE
SYSTEM_CLASSPRIVILEGES
SYSTEM_COLUMNPRIVILEGES
SYSTEM_COLUMNS
SYSTEM_CROSSREFERENCE
SYSTEM_INDEXINFO
SYSTEM_PRIMARYKEYS
SYSTEM_PROCEDURECOLUMNS
SYSTEM_PROCEDURES
SYSTEM_PROPERTIES
SYSTEM_SCHEMAS
SYSTEM_SEQUENCES
SYSTEM_SESSIONINFO
SYSTEM_SESSIONS
SYSTEM_SUPERTABLES
SYSTEM_SUPERTYPES
SYSTEM_TABLEPRIVILEGES
SYSTEM_TABLES
SYSTEM_TABLETYPES
SYSTEM_TABLE_CONSTRAINTS
SYSTEM_TEXTTABLES
SYSTEM_TRIGGERCOLUMNS
SYSTEM_TRIGGERS
SYSTEM_TYPEINFO
SYSTEM_UDTATTRIBUTES
SYSTEM_UDTS
SYSTEM_USAGE_PRIVILEGES
SYSTEM_USERS
SYSTEM_VERSIONCOLUMNS
SYSTEM_VIEWS
SYSTEM_VIEW_COLUMN_USAGE
```

```
SYSTEM_VIEW_ROUTINE_USAGE
SYSTEM_VIEW_TABLE_USAGE
```

Die folgende Abfrage gibt z.B. eine komplette Übersicht über alle in der Datenbank genutzten Tabellen mit Feldtypen, Primärschlüsseln und Fremdschlüsseln:

```
001 SELECT
002    "A"."TABLE_NAME",
003    "A"."COLUMN_NAME",
004    "A"."TYPE_NAME",
005    "A"."NULLABLE",
006    "B"."KEY_SEQ" AS "PRIMARYKEY",
007    "C"."PKTABLE_NAME" || '.' || "C"."PKCOLUMN_NAME" AS "FOREIGNKEY FOR"
008 FROM "INFORMATION_SCHEMA"."SYSTEM_COLUMNS" AS "A"
009    LEFT JOIN "INFORMATION_SCHEMA"."SYSTEM_PRIMARYKEYS" AS "B"
010    ON ( "B"."TABLE_NAME" = "A"."TABLE_NAME" AND "B"."COLUMN_NAME" =
          "A"."COLUMN_NAME" )
011    LEFT JOIN "INFORMATION_SCHEMA"."SYSTEM_CROSSREFERENCE" AS "C"
012    ON ( "C"."FKTABLE_NAME" = "A"."TABLE_NAME" AND "C"."FKCOLUMN_NAME" =
          "A"."COLUMN_NAME" )
013 WHERE "A"."TABLE_SCHEM" = 'PUBLIC'
```

Informationstabellen der Firebird-Datenbank

Die Firebird-Informationstabellen sind deutlich mehr gesplittet als die der HSQLDB. So stehen z.B. Tabellennamen, Feldnamen und Feldtypen nicht zusammen in einer Tabelle, sondern müssen über mehrere Tabellen hinweg zusammen gebracht werden:

```
001 SELECT
002    "a".RDB$RELATION_NAME AS "Tables",
003    "a".RDB$FIELD_NAME AS "Fields",
004    "c".RDB$TYPE_NAME AS "Types",
005    "a".RDB$Field_POSITION AS "Fieldposition",
006    "a".RDB$NULL_FLAG AS "Nullflag"
007 FROM RDB$RELATION_FIELDS AS "a", RDB$FIELDS AS "b", RDB$TYPES AS "c"
008 WHERE "a".RDB$FIELD_SOURCE = "b".RDB$FIELD_NAME
009    AND "b".RDB$FIELD_TYPE = "c".RDB$TYPE
010    AND "c".RDB$FIELD_NAME = 'RDB$FIELD_TYPE'
011    AND "a".RDB$SYSTEM_FLAG = 0
012 ORDER BY "Tables", "Fieldposition"
```

Hiermit wird eine Übersicht über alle selbst erstellten Tabellen mit den Feldnamen, Feldtypen, Feldpositionen innerhalb der Tabelle und der Information, ob sie leer sein dürfen, ermöglicht. Um den Feldtypen zuordnen zu können muss also über die Tabelle "RDB$FIELD_TYPE" zur Tabelle "RDB$TYPE" verbunden werden.

Leider stimmen die dort aufgeführten Typen nur mit denen überein, die intern von Firebird genutzt werden. Sie lauten anders als die, die in dem Tabelleneditor vorkommen. Hier ein Code, um die entsprechende Übersicht zu bekommen, die nur über Nummern aus den Informationstabellen auslesbar ist:

```
001 SELECT TRIM("a".RDB$RELATION_NAME) AS "Tabelle",
002    "a".RDB$FIELD_NAME AS "Feld",
003    TRIM(CASE "b".RDB$FIELD_TYPE||'|'|| COALESCE("b".RDB$FIELD_SUB_TYPE,0)
004       WHEN '7|0' THEN 'SMALLINT'
005       WHEN '8|0' THEN 'INTEGER'
006       WHEN '8|1' THEN 'NUMERIC'
007       WHEN '8|2' THEN 'DECIMAL'
008       WHEN '10|0' THEN 'FLOAT'
009       WHEN '12|0' THEN 'DATE'
010       WHEN '13|0' THEN 'TIME'
```

```
011       WHEN '14|0' THEN 'CHAR'
012       WHEN '16|0' THEN 'BIGINT'
013       WHEN '27|0' THEN 'DOUBLE PRECISION'
014       WHEN '35|0' THEN 'TIMESTAMP'
015       WHEN '37|0' THEN 'VARCHAR'
016       WHEN '261|0' THEN 'BLOB'
017       WHEN '261|1' THEN 'BLOB Text'
018       WHEN '261|2' THEN 'BLOB BLR'
019       WHEN '261|3' THEN 'BLOB ACL'
020       END) AS "SQL_Datentyp",
021    COALESCE(
022       "b".RDB$CHARACTER_LENGTH,
023       "b".RDB$FIELD_PRECISION||','||
024       ("b".RDB$FIELD_SCALE*-1)) AS "Laenge,Nachkommastellen"
025 FROM RDB$RELATION_FIELDS AS "a", RDB$FIELDS AS "b"
026 WHERE "a".RDB$FIELD_SOURCE = "b".RDB$FIELD_NAME
027    AND "a".RDB$SYSTEM_FLAG = 0
```

Diese Abfrage gibt alle genutzten Tabellen mit den Feldnamen, den Feldtypen und der Feldlänge bzw. Genauigkeit bei Dezimalzahlen an.

Die Ausgabe in dieser Abfrage erfolgt in festen Spaltenbreiten. Von der HSQLDB ist es eher so, dass nur die Anzahl der lesbaren Zeichen wiedergegeben werden. Falls also eine Ausgabe nach den Bezeichnern ausgewertet werden soll, so sind die Leerzeichen durch **TRIM** zu entfernen.

Automatisch generierte Werte wie in der HSQLDB werden bei Firebird über Generatoren erstellt. Um den Startwert eines solchen Generators neu einstellen zu können ist der Name des Generators erforderlich. Die folgende Abfrage liefert den Generatornamen für alle Tabellen, die von dem Nutzer erstellt wurden:

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

Ansichten werden auch von Firebird unterstützt. Die folgende Abfrage liefert den Namen und die SQL-Formulierung für jede Ansicht:

```
001 SELECT RDB$RELATION_NAME, RDB$VIEW_SOURCE
002 FROM RDB$RELATIONS
003 WHERE RDB$VIEW_SOURCE IS NOT NULL
```

Die Firebird-Systemtabellen starten alle mit den Anfangsbuchstaben RDB$:

```
RDB$BACKUP_HISTORY
RDB$CHARACTER_SETS
RDB$CHECK_CONSTRAINTS
RDB$COLLATIONS
RDB$DATABASE
RDB$DEPENDENCIES
RDB$EXCEPTIONS
RDB$FIELDS
RDB$FIELD_DIMENSIONS
RDB$FILES
RDB$FILTERS
RDB$FORMATS
RDB$FUNCTIONS
RDB$FUNCTION_ARGUMENTS
RDB$GENERATORS
RDB$INDICES
RDB$INDEX_SEGMENTS
RDB$LOG_FILES
RDB$PAGES
RDB$PROCEDURES
RDB$PROCEDURE_PARAMETERS
```

```
RDB$REF_CONSTRAINTS
RDB$RELATIONS
RDB$RELATION_CONSTRAINTS
RDB$RELATION_FIELDS
RDB$ROLES
RDB$SECURITY_CLASSES
RDB$TRANSACTIONS
RDB$TRIGGERS
RDB$TRIGGER_MESSAGES
RDB$TYPES
RDB$USER_PRIVILEGES
RDB$VIEW_RELATIONS
```

Datenbankreparatur für *.odb-Dateien

Regelmäßige Datensicherung sollte eigentlich Grundlage für den Umgang mit dem PC sein. Sicherheitskopien sind so der einfachste Weg, auf einen halbwegs aktuellen Datenstand zurückgreifen zu können. Doch in der Praxis mangelt es eben häufig an dieser Stelle.

Formulare, Abfragen und Berichte können, sofern eine Vorversion der Datenbank gesichert wurde, über die Zwischenablage in eine neue Datenbank kopiert werden. Lässt sich allerdings, aus welchen Gründen auch immer, eine aktuelle Datenbankdatei nicht mehr öffnen, so ist das Hauptproblem: Wie komme ich (hoffentlich) an die Daten.

Hinweis

Bevor es an die Wiederherstellung einer Datenbank-Archivdatei geht sollte erst einmal geklärt werden, ob nicht irgendwo im System doch eine Sicherungskopie existiert, die nicht bewusst angelegt wurde.
Eine Möglichkeit ist der Backup-Pfad von LibreOffice: **Extras → Optionen → LibreOffice → Pfade → Sicherungskopien** gibt darüber Aufschluss. Falls dieser Pfad noch nicht existiert wurde er noch nicht genutzt.
Eine weitere Möglichkeit ist das temporäre Verzeichnis des Betriebssystems: **Extras → Optionen → LibreOffice → Pfade → Temporäre Dateien** dürfte hier den richtigen Pfad angeben. Unter Linux ist dies **/tmp**. In diesem Pfad befinden sich Verzeichnisse, die mit lu….tmp gekennzeichnet sind, also von der Namensgebung so aussehen, als wären es Dateinamen mit der Endung «tmp». In diesen Pfaden befinden sich Dateien von LibreOffice, sofern die Dateien gerade geöffnet sind und bearbeitet werden. Wird LibreOffice normal beendet, so werden diese Pfade wieder gelöscht. Alte Pfade weisen also darauf hin, dass das Programm zu dem Zeitpunkt nicht korrekt beendet wurde. Leider haben die Dateien in diesen Pfaden nicht die ursprünglichen Namen. Unter Linux lassen sich aber Dateitypen auch mit der Endung *.tmp problemlos den Programmtypen zuordnen.

Bei plötzlichen Abstürzen des PC kann es passieren, dass geöffnete Datenbanken von LO (interne Datenbank HSQLDB) nicht mehr zu öffnen sind. Stattdessen wird beim Versuch, die Datenbank zu öffnen, nach einem entsprechenden Filter für das Format gefragt.

Das Ganze liegt daran, dass Teile der Daten der geöffneten Datenbank im Arbeitsspeicher liegen und lediglich temporär zwischengespeichert werden. Oft wird erst beim Schließen der Datei die gesamte Datenbank in die Datei zurückgeschrieben und gepackt. Lediglich Eingaben, die direkt in die Tabellen erfolgen, werden auch direkt in die Datenbankdatei gesichert. Seit LO 5.1 kann außerdem die Datenbank noch über den Button Speichern gesichert werden. Dies war vorher nicht der Fall, da der Button nicht den Zugang zu einem Untermenü anbot und deshalb inaktiv war.

Wiederherstellung der Datenbank-Archivdatei

Um eventuell doch noch an die Daten zu kommen, kann das folgende Verfahren hilfreich sein:

1. Fertigen sie eine Kopie ihrer Datenbank für die weiteren Schritte an.
2. Versuchen Sie die Kopie mit einem Packprogramm zu öffnen. Es handelt sich bei der *.odb-Datei um ein gepacktes Format, ein Zip-Archiv. Lässt sich die Datei so nicht direkt öffnen, so funktioniert das Ganze vielleicht auch über die Umbenennung der Endung von *.odb zu *.zip.
   Funktioniert das Öffnen nicht, so ist vermutlich von der Datenbank nichts mehr zu retten.
3. Folgende Verzeichnisse sehen Sie nach dem Öffnen einer Datenbankdatei im Packprogramm auf jeden Fall:

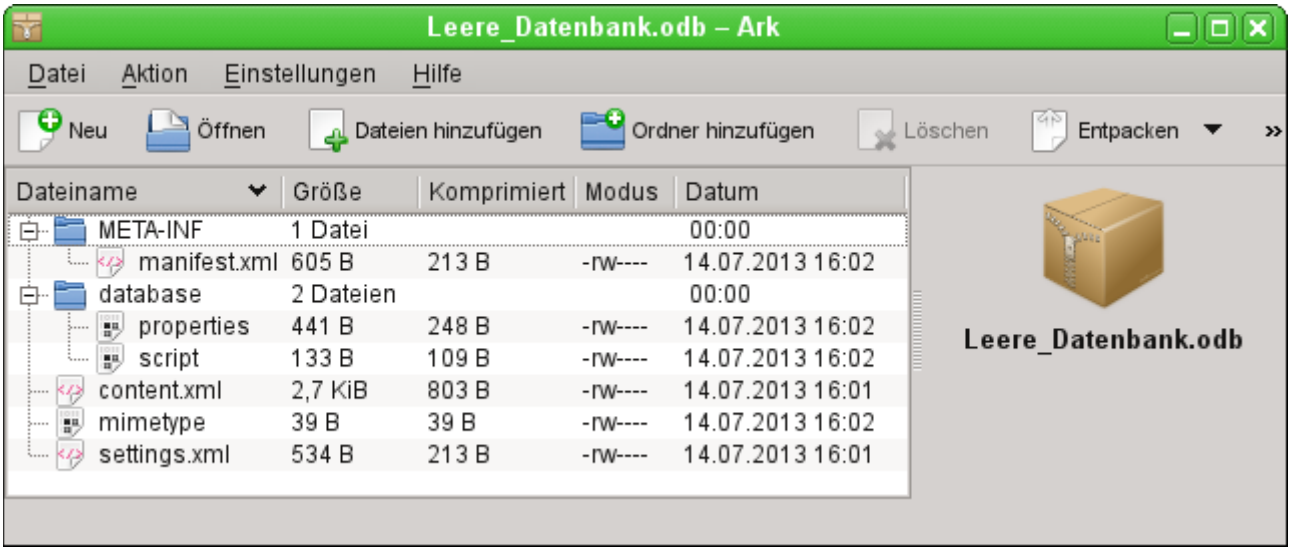


Abbildung 1: Aufbau einer Datenbankdatei ohne Tabellen, Abfragen, Formulare und Berichte

4. Die Datenbankdatei muss ausgepackt werden. Die entscheidenden Informationen für die Daten liegen im Unterverzeichnis «database» in den Dateien «data» und «script».
5. Gegebenenfalls empfiehlt es sich, die Datei «script» einmal anzuschauen und auf Ungereimtheiten zu überprüfen. Dieser Schritt kann aber auch erst einmal zum Testen übersprungen werden. Die «script»-Datei enthält vor allem die Beschreibung der Tabellenstruktur.
6. Gründen sie eine neue, leere Datenbankdatei und öffnen diese Datenbankdatei mit dem Packprogramm.
7. Ersetzen sie die Dateien «data» und «script» aus der neuen Datenbankdatei durch die unter «4.» entpackten Dateien.
8. Das Packprogramm muss nun geschlossen werden. War es, je nach Betriebssystem, notwendig, die Dateien vor dem Öffnen durch das Packprogramm nach *.zip umzubenennen, so ist das jetzt wieder nach *.odb zu wandeln.
9. Öffnen sie die Datenbankdatei jetzt mit LO. Sie können hoffentlich wieder auf ihre Tabellen zugreifen.
10. Wie weit sich jetzt auch Abfragen, Formulare und Berichte auf ähnliche Weise wiederherstellen lassen, bleibt dem weiteren Testen überlassen.

Siehe hierzu auch: *http://user.services.LO oder Ooo.org/en/forum/viewtopic.php?f=83&t=17125*

Weitere Informationen zur Datenbank-Archivdatei

Eine Datenbank-Archivdatei enthält im tatsächlichen Gebrauch neben dem grundlegenden Verzeichnis für die Datenbank und dem für das OpenDocument-Format vorgeschriebenen Verzeichnis «META-INF» noch weitere Verzeichnisse, um Formulare und Berichte abzuspeichern. Eine Beschreibung zum grundsätzliche Aufbau des OpenDocument-Formates ist u.a. unter *http://de.wikipedia.org/wiki/OpenDocument* zu finden.

Die folgende Übersicht zeigt eine Datenbank, die Tabellen ein Formular und einen Bericht enthält. Nicht offen sichtbar ist hier, dass auch eine Abfrage zur Datenbank gehört. Solche Abfragen werden nicht in separaten Verzeichnissen gespeichert, sondern sind in der Datei «content.xml» enthalten. Die dafür notwendigen Informationen beschränken sich schließlich auf eine einfache SQL-Formulierung.

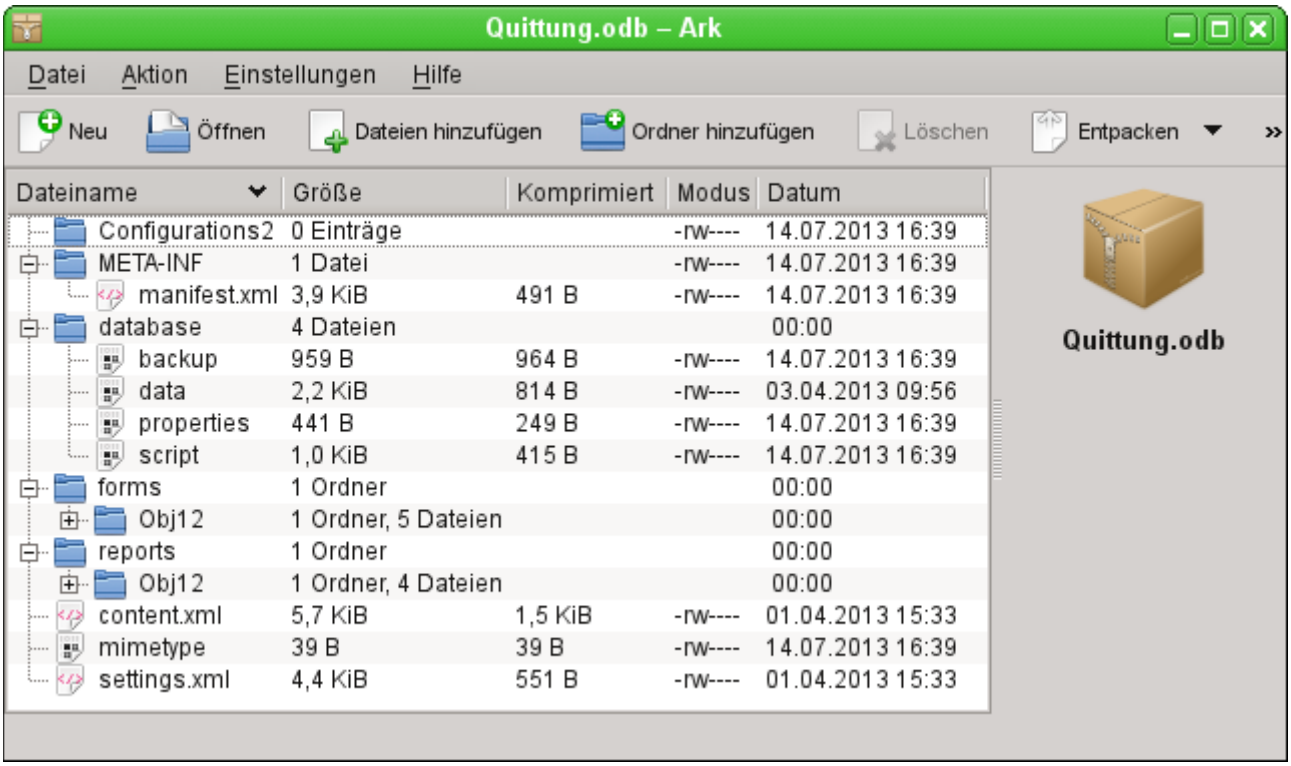


Abbildung 2: Datenbankdatei, die neben der Datenbank auch Informationen zu einem Formular und einem Bericht abgespeichert hat.

Hier einige der Dateien aus der Datenbank-Archivdatei im Überblick:

mimetype

```
004 application/vnd.oasis.opendocument.base
```

Diese kleine Textdatei enthält lediglich den Hinweis, dass es sich bei der Archivdatei um eine Datenbankdatei im OpenDocument-Format handelt.

content.xml einer Datenbank ohne Inhalt

```
005 <?xml version="1.0" encoding="UTF-8"?>
    <office:document-content
    xmlns:office="urn:oasis:names:tc:opendocument:xmlns:office:1.0"
    xmlns:style="urn:oasis:names:tc:opendocument:xmlns:style:1.0"
    xmlns:text="urn:oasis:names:tc:opendocument:xmlns:text:1.0"
    xmlns:table="urn:oasis:names:tc:opendocument:xmlns:table:1.0"
    xmlns:draw="urn:oasis:names:tc:opendocument:xmlns:drawing:1.0"
    xmlns:fo="urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns:meta="urn:oasis:names:tc:opendocument:xmlns:meta:1.0"
```

```
xmlns:number="urn:oasis:names:tc:opendocument:xmlns:datastyle:1.0"
xmlns:svg="urn:oasis:names:tc:opendocument:xmlns:svg-compatible:1.0"
xmlns:chart="urn:oasis:names:tc:opendocument:xmlns:chart:1.0"
xmlns:dr3d="urn:oasis:names:tc:opendocument:xmlns:dr3d:1.0"
xmlns:math="http://www.w3.org/1998/Math/MathML"
xmlns:form="urn:oasis:names:tc:opendocument:xmlns:form:1.0"
xmlns:script="urn:oasis:names:tc:opendocument:xmlns:script:1.0"
xmlns:ooo="http://openoffice.org/2004/office"
xmlns:ooow="http://openoffice.org/2004/writer"
xmlns:oooc="http://openoffice.org/2004/calc"
xmlns:dom="http://www.w3.org/2001/xml-events"
xmlns:db="urn:oasis:names:tc:opendocument:xmlns:database:1.0"
xmlns:xforms="http://www.w3.org/2002/xforms"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:rpt="http://openoffice.org/2005/report"
xmlns:of="urn:oasis:names:tc:opendocument:xmlns:of:1.2"
xmlns:xhtml="http://www.w3.org/1999/xhtml"
xmlns:grddl="http://www.w3.org/2003/g/data-view#"
xmlns:tableooo="http://openoffice.org/2009/table"
xmlns:drawooo="http://openoffice.org/2010/draw"
xmlns:calcext="urn:org:documentfoundation:names:experimental:calc:xmlns:calcext:1.0
"
 xmlns:field="urn:openoffice:names:experimental:ooo-ms-interop:xmlns:field:1.0"
xmlns:formx="urn:openoffice:names:experimental:ooxml-odf-interop:xmlns:form:1.0"
xmlns:css3t="http://www.w3.org/TR/css3-text/"
office:version="1.2">
   <office:scripts/>
   <office:font-face-decls/>
   <office:automatic-styles/>
   <office:body>
      <office:database>
         <db:data-source>
            <db:connection-data>
               <db:connection-resource xlink:href="sdbc:embedded:hsqldb"/>
               <db:login db:is-password-required="false"/>
            </db:connection-data>
            <db:driver-settings
               db:system-driver-settings=""
               db:base-dn=""
               db:parameter-name-substitution="false"/>
            <db:application-connection-settings
               db:is-table-name-length-limited="false"
               db:append-table-alias-name="false"
               db:max-row-count="100">
               <db:table-filter>
                  <db:table-include-filter>
                     <db:table-filter-pattern>%</db:table-filter-pattern>
                  </db:table-include-filter>
               </db:table-filter>
            </db:application-connection-settings>
         </db:data-source>
      </office:database>
   </office:body>
</office:document-content>
```

Zu Beginn wird die xml-Version und der verwendete Zeichensatz geklärt. Der gesamte nachfolgende Inhalt wird ohne Absatz direkt in einer Zeile ausgegeben. In der oben erstellten Übersicht wird hoffentlich der Inhalt etwas klarer. Zusammengehörige Elemente werden in «*Tags*» gefasst.

Mit den Anfangsdefinitionen werden durch «xmlns» (XML-Namespace) die Namensräume umschrieben, auf die innerhalb dieser Datei zugegriffen werden kann. Anschließend werden etwas konkretere Angaben zum Inhalt gemacht. Hier ist dann z.B. ersichtlich, dass es sich um eine interne HSQLDB-Datenbank handelt und die Angabe eines Passwortes nicht erforderlich ist.

content.xml einer Datenbank mit Inhalt
Der folgende Inhalt ist nur ein Auszug der content.xml-Datei und soll nur die Struktur klären.

```
006 <office:scripts/>
    <office:font-face-decls>
       <style:font-face style:name="F" svg:font-family=""/>
    </office:font-face-decls>
    <office:automatic-styles>
       <style:style
          style:name="co1"
          style:family="table-column"
          style:data-style-name="N0"/>
       <style:style
          style:name="co2"
          style:family="table-column"
          style:data-style-name="N107"/>
       <style:style style:name="ce1" style:family="table-cell">
          <style:paragraph-properties fo:text-align="start"/>
       </style:style>
       <number:number-style style:name="N0" number:language="de" number:country="DE">
          <number:number number:min-integer-digits="1"/>
       </number:number-style>
       <number:currency-style
          style:name="N107P0"
          style:volatile="true"
          number:language="de"
          number:country="DE">
          <number:number
             number:decimal-places="2"
             number:min-integer-digits="1"
             number:grouping="true"/>
          <number:text> </number:text>
          <number:currency-symbol
             number:language="de"
             number:country="DE">€
          </number:currency-symbol>
       </number:currency-style>
```

Hier wird ein Feld als Währungsfeld festgelegt. Die Anzahl der Dezimalstellen werden genannt, der Abstand zwischen Zahlen und Währungssymbol sowie das Währungssymbol selbst.

```
007    <number:currency-style
          style:name="N107"
          number:language="de"
          number:country="DE">
          <style:text-properties fo:color="#ff0000"/>
          <number:text>-</number:text>
          <number:number
             number:decimal-places="2"
             number:min-integer-digits="1"
             number:grouping="true"/>
          <number:text> </number:text>
          <number:currency-symbol
             number:language="de"
             number:country="DE">€
          </number:currency-symbol>
          <style:map style:condition="value()&gt;=0" style:apply-style-name="N107P0"/>
       </number:currency-style>
```

Im zweiten Abschnitt erfolgt die Festlegung, dass bis zu einem bestimmten Wert die Währung in der Farbe Rot («#ff0000») erscheinen soll.

```
008 </office:automatic-styles>
    <office:body>
       <office:database>
          <db:data-source>
```

Dieser Eintrag entspricht mit allen Unterpunkten dem aus der oben beschriebenen content.xml einer Datenbank-Archivdatei ohne Inhalt.

```
009         </db:data-source>
            <db:forms>
               <db:component
                  db:name="Quittung"
                  xlink:href="forms/Obj12"
                  db:as-template="false"/>
            </db:forms>
```

Die Datenbank-Archivdatei enthält einen Unterordner, in dem die Details zu einem Formular abgespeichert sind. Das Formular ist auf der Benutzeroberfläche mit dem Namen «Quittung» verzeichnet.

```
010         <db:reports>
               <db:component
                  db: name="Quittung"
                  xlink:href="reports/Obj12"
                  db:as-template="false"/>
            </db:reports>
```

Die Datenbank-Archivdatei enthält einen Unterordner, in dem die Details zu einem Bericht abgespeichert sind. Der Bericht ist auf der Benutzeroberfläche ebenfalls mit dem Namen «Quittung» verzeichnet.

```
011         <db:queries>
               <db:query
                  db:name="Verkauf_berechnet"
                  db:command="SELECT &quot;a&quot;.*, ( SELECT &quot;Preis&quot; *
                     &quot;a&quot;.&quot;Anzahl&quot; FROM &quot;Ware&quot; WHERE
                     &quot;ID&quot; = &quot;a&quot;.&quot;Ware_ID&quot; ) AS
                     &quot;Anzahl*Preis&quot; FROM &quot;Verkauf&quot; AS
                     &quot;a&quot;"/>
            </db:queries>
```

Sämtliche Abfragen werden direkt in der content.xml gespeichert. «"» steht dabei für ein doppeltes Anführungszeichen oben «"». Die oben stehende Abfrage ist in diesem Beispiel eigentlich recht umfangreich und besteht aus vielen korrelierenden Unterabfragen. Sie ist hier nur verkürzt wiedergegeben.

```
012         <db:table-representations>
               <db:table-representation db:name="Quittung"/>
               <db:table-representation db:name="Verkauf"/>
               <db:table-representation db:name="Ware">
                  <db:columns>
                     <db:column
                        db:name="ID"
                        db:style-name="co1"
                        db:default-cell-style-name="ce1"/>
                     <db:column
                        db:name="MWSt"
                        db:style-name="co1"
                        db:default-cell-style-name="ce1"/>
                     <db:column
                        db:name="Preis"
                        db:help-message="Angabe als Nettopreis"
                        db:style-name="co2"
                        db:default-cell-style-name="ce1"/>
                     <db:column
                        db:name="Ware"
                        db:style-name="co1"
                        db:default-cell-style-name="ce1"/>
                  </db:columns>
               </db:table-representation>
            </db:table-representations>
```

Wie sollen verschiedene Tabellen von der Ansicht her erscheinen? An dieser Stelle wird das Erscheinungsbild bestimmter Spalten gespeichert; in diesem Beispiel wurden Einstellungen der Tabelle "Ware" mit ihren einzelnen Feldern "ID", "MWSt" usw. abgespeichert. Hier wurde zum einen beim Preis eine Zusatzinformation angegeben. Zum anderen ist eine Formatierung der Angabe vorgenommen worden. Der **style-name** 'co2' entspricht beim Preis dem **style-name**, der zu Beginn der content.xml definiert wurde und mit dem **data-style-name** 'N107' verbunden ist. Der **data-style-name** 'N107' wiederum verweist auf die Formatierung des Feldes als Währungsfeld.

```
013     </office:database>
    </office:body>
```

Grundsätzlich ist in der content.xml der Inhalt der Abfragen und Informationen zum Erscheinungsbild der Tabellen direkt gespeichert. Außerdem ist eine Definition der Verbindung zur Datenbank enthalten. Schließlich kommen noch Verweise auf Formulare und Berichte hinzu.

settings.xml

```
014 <?xml version="1.0" encoding="UTF-8"?>
    <office:document-settings
    xmlns:office="urn:oasis:names:tc:opendocument:xmlns:office:1.0"
    xmlns:table="urn:oasis:names:tc:opendocument:xmlns:table:1.0"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns:number="urn:oasis:names:tc:opendocument:xmlns:datastyle:1.0"
    xmlns:svg="http://www.w3.org/2000/svg"
    xmlns:config="urn:oasis:names:tc:opendocument:xmlns:config:1.0"
    xmlns:ooo="http://openoffice.org/2004/office"
    xmlns:db="urn:oasis:names:tc:opendocument:xmlns:database:1.0"
    office:version="1.2"/>
```

Bei einer Datenbank ohne weiteren Inhalt stehen hier nur die Grunddefinitionen. Mit Inhalt werden hier aber die unterschiedlichsten Einstellungen abgespeichert. Nach dem Start mit der obigen Definition sind folgende Einstellungen in der Beispieldatei abgespeichert:

```
015 <office:settings>
      <config:config-item-set config:name="ooo:view-settings">
        <config:config-item-set config:name="Queries">
          <config:config-item-set config:name="Verkauf_berechnet">
            <config:config-item-set config:name="Tables">
              <config:config-item-set config:name="Table1">
                <config:config-item config:name="WindowName"
                  config:type="string">Verkauf</config:config-item>
                <config:config-item config:name="WindowLeft"
                  config:type="int">153</config:config-item>
                <config:config-item config:name="ShowAll"
                  config:type="boolean">true</config:config-item>
                <config:config-item config:name="WindowTop"
                  config:type="int">17</config:config-item>
                <config:config-item config:name="WindowWidth"
                  config:type="int">120</config:config-item>
                <config:config-item config:name="WindowHeight"
                  config:type="int">120</config:config-item>
                <config:config-item config:name="ComposedName"
                  config:type="string">Verkauf</config:config-item>
                <config:config-item config:name="TableName"
                  config:type="string">Verkauf</config:config-item>
              </config:config-item-set>
            </config:config-item-set>
            <config:config-item config:name="SplitterPosition"
              config:type="int">105</config:config-item>
            <config:config-item config:name="VisibleRows"
              config:type="int">1024</config:config-item>
          </config:config-item-set>
        </config:config-item-set>
      </config:config-item-set>
      <config:config-item-set config:name="ooo:configuration-settings">
        <config:config-item-set config:name="layout-settings">
```

```
            <config:config-item-set config:name="Tables">
               <config:config-item-set config:name="Table1">
                  <config:config-item config:name="WindowName"
                     config:type="string">Verkauf</config:config-item>
                  <config:config-item config:name="WindowLeft"
                     config:type="int">186</config:config-item>
                  <config:config-item config:name="ShowAll"
                     config:type="boolean">false</config:config-item>
                  <config:config-item config:name="WindowTop"
                     config:type="int">17</config:config-item>
                  <config:config-item config:name="WindowWidth"
                     config:type="int">120</config:config-item>
                  <config:config-item config:name="WindowHeight"
                     config:type="int">120</config:config-item>
                  <config:config-item config:name="ComposedName"
                     config:type="string">Verkauf</config:config-item>
                  <config:config-item config:name="TableName"
                     config:type="string">Verkauf</config:config-item>
               </config:config-item-set>
               <config:config-item-set config:name="Table2">
                  ... (identische config:type-Punkte wie "Table1"
                  <config:config-item config:name="TableName"
                     config:type="string">Ware</config:config-item>
               </config:config-item-set>
               <config:config-item-set config:name="Table3">
                  ... (identische config:type-Punkte wie "Table1"
                  <config:config-item config:name="TableName"
                     config:type="string">Quittung</config:config-item>
               </config:config-item-set>
            </config:config-item-set>
         </config:config-item-set>
      </config:config-item-set>
   </office:settings>
```

Die gesamte Übersicht bezieht sich auf verschiedene Ansichten der Fenster für die (eine) Abfrage "Verkauf berechnet" und für die Tabellen "Verkauf", "Ware" und "Quittung". Die letzten beiden wurden hier nur verkürzt wiedergegeben. Würden diese Einstellungen bei einer defekten *.odb-Datei fehlen, so wäre das also nicht weiter von Bedeutung. Sie würden wieder erstellt, wenn die entsprechenden Fenster das nächste Mal geöffnet werden.

META-INF/manifest.xml

```
016 <?xml version="1.0" encoding="UTF-8"?>
   <manifest:manifest
      xmlns:manifest="urn:oasis:names:tc:opendocument:xmlns:manifest:1.0">
      <manifest:file-entry
         manifest:full-path="/"
         manifest:media-type="application/vnd.oasis.opendocument.base"/>
      <manifest:file-entry
         manifest:full-path="database/script"
         manifest:media-type=""/>
      <manifest:file-entry
         manifest:full-path="database/properties"
         manifest:media-type=""/>
      <manifest:file-entry
         manifest:full-path="settings.xml"
         manifest:media-type="text/xml"/>
      <manifest:file-entry
         manifest:full-path="content.xml"
         manifest:media-type="text/xml"/>
   </manifest:manifest>
```

Bei dieser Datei im Unterverzeichnis META-INF handelt es sich um ein Inhaltsverzeichnis der gesamten Datenbank-Archivdatei. Da es sich bei der oben gezeigten Datei um die Datei handelt, die in der leeren Datenbank (1) enthalten ist, gibt es hier nur 5 Datei-Einträge («file-entry»). Bei der mit Formular und Bericht versehenen Datenbank-Archivdatei sind die Einträge in der META-INF entsprechend umfangreicher.

database/properties

```
017 #HSQL Database Engine 1.8.0.10
018 #Sun Jul 14 18:02:08 CEST 2013
019 hsqldb.script_format=0
020 runtime.gc_interval=0
021 sql.enforce_strict_size=true
022 hsqldb.cache_size_scale=8
023 readonly=false
024 hsqldb.nio_data_file=false
025 hsqldb.cache_scale=13
026 version=1.8.0
027 hsqldb.default_table_type=cached
028 hsqldb.cache_file_scale=1
029 hsqldb.lock_file=true
030 hsqldb.log_size=10
031 modified=no
032 hsqldb.cache_version=1.7.0
033 hsqldb.original_version=1.8.0
034 hsqldb.compatible_version=1.8.0
```

Die properties-Datei enthält die Grundeinstellungen für die interne HSQL Datenbank. Siehe dazu auch das folgende Kapitel.

database/script

```
035 SET DATABASE COLLATION "German"
036 CREATE SCHEMA PUBLIC AUTHORIZATION DBA
037 CREATE USER SA PASSWORD ""
038 GRANT DBA TO SA
039 SET WRITE_DELAY 60
```

In der script-Datei finden sich die Standardeinstellungen für die Verbindung zur Datenbank, zur benutzten Sprache usw. Hier erscheint auch der später erwähnte Benutzer «SA».

In einer mit Inhalt gefüllten Datenbank werden in dieser Datei die Grundlagen für die Tabellendefinitionen gespeichert:

```
040 SET DATABASE COLLATION "German"
041 CREATE SCHEMA PUBLIC AUTHORIZATION DBA
```

Die Tabellen werden definiert, bevor der Datenbanknutzer definiert wird. Zuerst werden die Tabellen mit ihren Feldern im Cache erstellt.

```
042 CREATE CACHED TABLE "Ware"
043    ("ID" INTEGER GENERATED BY DEFAULT AS IDENTITY(START WITH 0) NOT NULL
044    PRIMARY KEY,"Ware" VARCHAR(50),"Preis" DECIMAL(8,2),"MWSt" TINYINT)
045 CREATE CACHED TABLE "Verkauf"
046    ("ID" INTEGER GENERATED BY DEFAULT AS IDENTITY(START WITH 0) NOT NULL
047    PRIMARY KEY,"Anzahl" TINYINT,"Ware_ID" INTEGER,"Quittung_ID" INTEGER,
048    CONSTRAINT SYS_FK_59 FOREIGN KEY("Ware_ID") REFERENCES "Ware"("ID"))
049 CREATE CACHED TABLE "Quittung"
050    ("ID" INTEGER GENERATED BY DEFAULT AS IDENTITY(START WITH 0) NOT NULL
051    PRIMARY KEY,"Datum" DATE)
```

Anschließend werden noch Änderungen an den Tabellen vorgenommen, damit die Beziehungen («REFERENCES») stimmig sind

```
052 ALTER TABLE "Verkauf" ADD CONSTRAINT SYS_FK_76 FOREIGN KEY("Quittung_ID")
053    REFERENCES "Quittung"("ID")
054 SET TABLE "Ware" INDEX'608 20'
055 SET TABLE "Verkauf" INDEX'1872 1656 1872 12'
056 SET TABLE "Quittung" INDEX'2232 1'
```

Nach der Einstellung der Position des Indexes in der data-Datei (erscheint nur hier in der script-Datei, wird nie direkt in SQL eingegeben!) werden die automatisch hoch schreibenden Felder der Tabellen («AutoWert») so eingestellt, dass sie die nächsten Werte bei Neueingaben erstellen. So ist z.B. der letzte eingetragene Wert im Feld "ID" der Tabelle "Ware" die Nummer 19. Das automatische Hochschreiben beginnt also mit der Nummer 20.

```
057 ALTER TABLE "Ware" ALTER COLUMN "ID" RESTART WITH 20
```

```
058  ALTER TABLE "Verkauf" ALTER COLUMN "ID" RESTART WITH 12
059  ALTER TABLE "Quittung" ALTER COLUMN "ID" RESTART WITH 1
060  CREATE USER SA PASSWORD ""
061  GRANT DBA TO SA
062  SET WRITE_DELAY 60
```

Behebung von Versionsproblemen

Wenn, wie auf den folgenden Seiten beschrieben, die externe HSQLDB verwendet wird, kann eventuell ein weiteres Problem mit den *.odb-Dateien in Verbindung mit manchen LO-Versionen auftauchen. Wird eine externe HSQLDB genutzt, so ist der sicherste Weg der über das hsqldb.jar-Archiv, das mit LO mitgeliefert wird. Wird ein anderes Archiv verwendet, so kann das dazu führen, dass die internen Datenbanken plötzlich nicht mehr zugänglich sind. Dies liegt daran, dass LO Schwierigkeiten hat, zwischen interner und externer HSQLDB zu unterscheiden und Meldungen von einem Versionskonflikt produziert.

Es muss als externe Datenbank die mitgelieferte hsqldb.jar-Datei genutzt werden. Außerdem muss aus der *.odb-Datei das database-Verzeichnis extrahiert werden. Die Datei properties hat hier einen Eintrag, der in LO 3.3 zu dieser Fehlermeldung führt:

```
010  version=1.8.1
```

steht in Zeile 11.

Diese Zeile ist zu ändern auf

```
010  version=1.8.0
```

Danach ist das database-Verzeichnis wieder in das *.odb-Päckchen einzulesen und die interne Datenbank lässt sich auch wieder unter LO öffnen.

Weitere Tipps

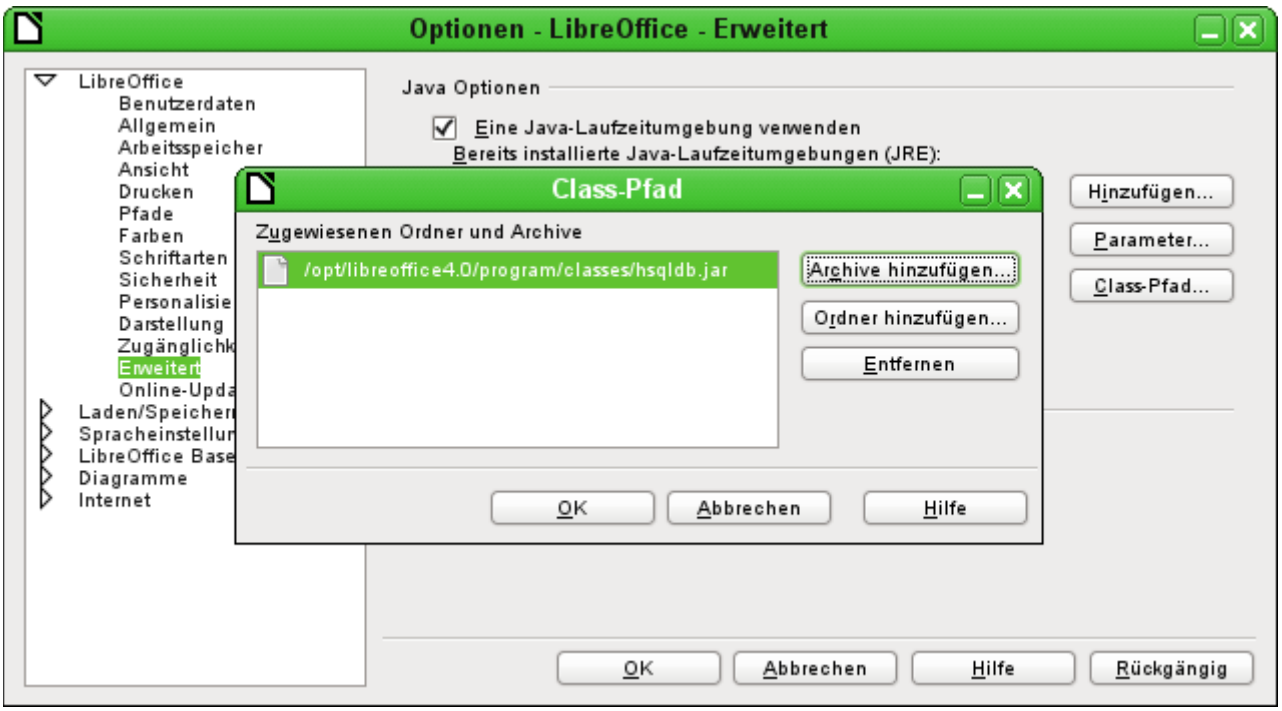
Wenn aus irgendwelchen Gründen wohl die Datenbankdatei geöffnet wird, aber kein Zugang mehr zu den Tabellen existiert, kann direkt über **Extras → SQL** der Befehl **`SHUTDOWN SCRIPT`** eingegeben werden. Anschließend wird die Datenbank geschlossen und neu gestartet. Das Ganze funktioniert aber nicht, wenn bereits ein «Error im Script-file» gemeldet wird.

Die Daten der Datenbank liegen in der *.odb-Datei im Unterverzeichnis «database». Hier gibt es eine Datei «data» und eine Datei «backup». Ist die Datei «data» defekt, so kann sie über die Datei «backup» wiederhergestellt werden. Hierzu muss die im Verzeichnis «database» liegende Datei «properties» bearbeitet werden. Hier gibt es eine Zeile «modified=no». Diese muss umgeschrieben werden zu «modified=yes». Das zeigt dem System an, dass die Datenbank nicht korrekt beendet wurde. Jetzt wird aus der komprimierten «backup»-Datei beim Neustart eine neue «data»-Datei erstellt.

Datenbankverbindung zu einer externen HSQLDB

Die interne HSQLDB unterscheidet sich erst einmal nicht von der externen Variante. Wenn, wie im Folgenden beschrieben, erst einmal nur der Zugriff auf die Datenbank nach außerhalb gelegt werden soll, dann ist keine Serverfunktion erforderlich. Hier reicht schlicht das Archiv, was in LO mitgeliefert wurde. Es liegt im LO-Pfad unter /program/classes/hsqldb.jar. Die Verwendung dieses Archivs ist die sicherste Variante, da dann keine Versionsprobleme auftauchen.

Die externe HSQLDB steht unter *http://hsqldb.org/* zum Download frei zur Verfügung. Diese anderen Versionen sollten allerdings nur genutzt werden, wenn Klarheit darüber besteht, wie Versionsprobleme zwischen interner und externer Variante behoben werden können.



Der Datenbanktreiber muss, sofern er nicht in dem Pfad der Java-Runtime liegt, als ClassPath unter Extras – Optionen – Java hinzugefügt werden.

Die Verbindung zu der externen Datenbank erfolgt über JDBC. Die Datenbankdateien sollen in einem bestimmten Verzeichnis abgelegt werden. Dieses Verzeichnis kann beliebig gewählt werden. Es liegt in dem folgenden Beispiel im home-Ordner. Nicht angegeben ist hier der weitere Verzeichnisverlauf sowie der Name der Datenbank.

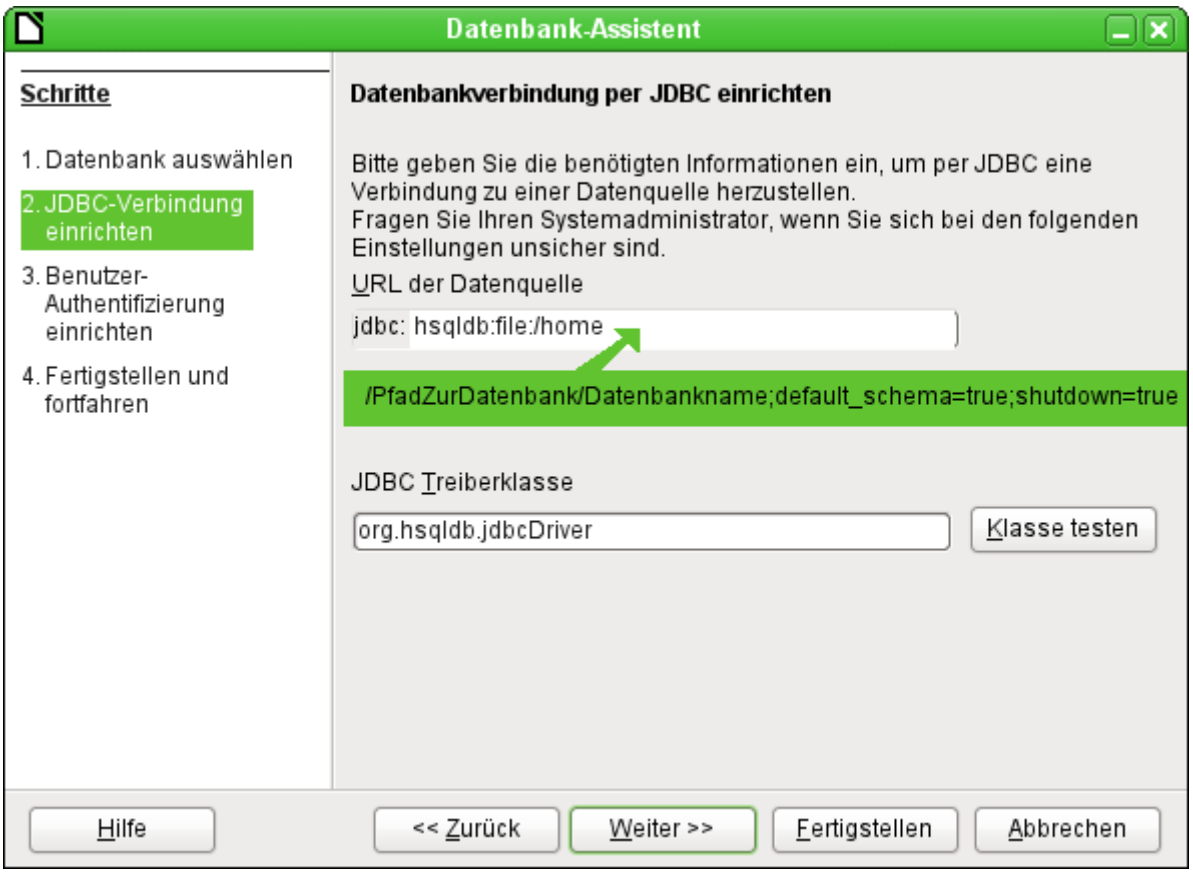
Wichtig, damit auch Daten in die Datenbank über die GUI geschrieben werden können, muss: ergänzend neben dem Datenbanknamen «**;default_schema=true**» stehen. Dies kann noch durch ein «**;shutdown=true**» ergänzt werden, damit die DB nach dem Schließen von LO heruntergefahren wird.

Also:

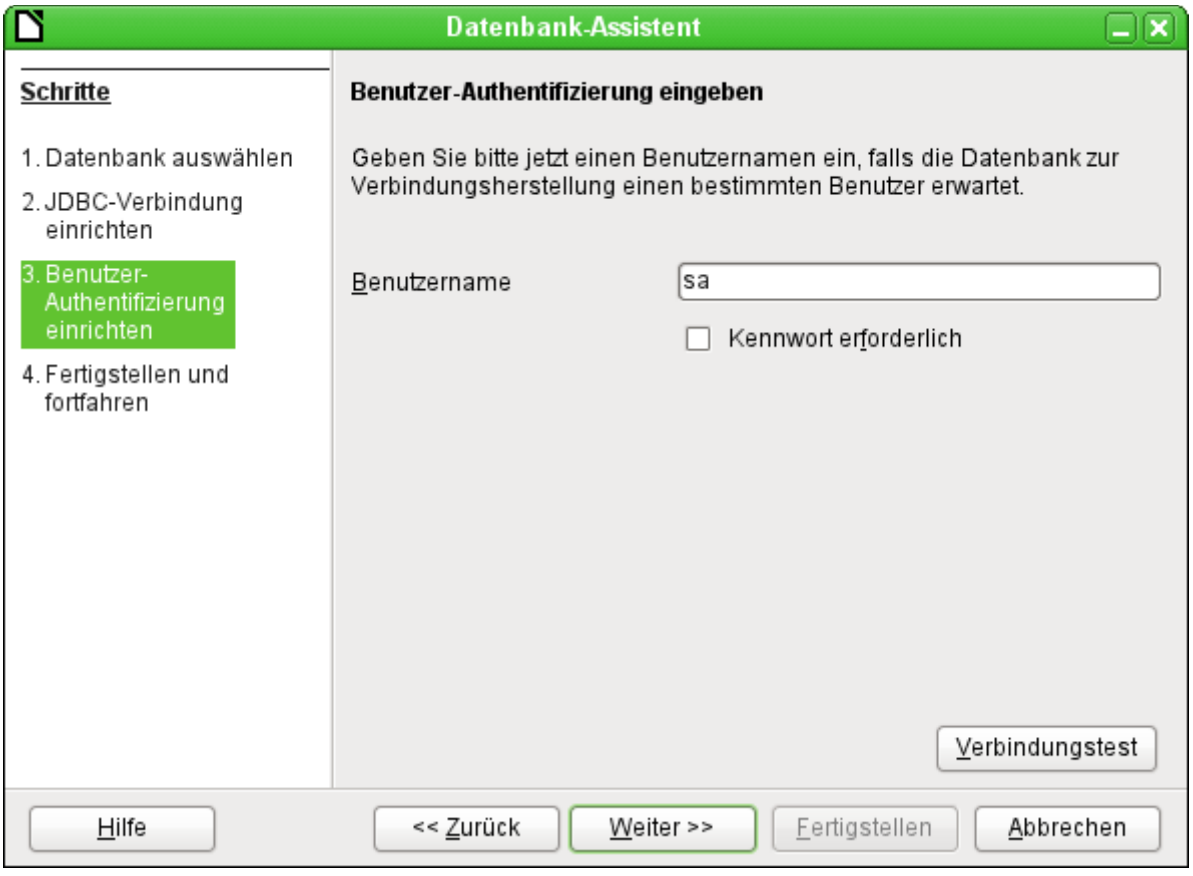
```
001 jdbc:hsqldb:file:/home/PfadZurDatenbank/Datenbankname;default_schema=true;
    shutdown=true
```

In dem Ordner befinden sich die Dateien

```
Datenbankname.backup
Datenbankname.data
Datenbankname.properties
Datenbankname.script
Datenbankname.log
```



Weiter geht es mit der Angabe des Standardnutzers, sofern nichts an der HSQLDB-Konfiguration geändert wurde:



Damit ist die Verbindung erstellt und es kann auf die Datenbank zugegriffen werden.

Vorsicht

Wird eine externe Datenbank mit einer Version HSQLDB 2.* bearbeitet, so kann sie anschließend nicht mehr in eine interne Datenbank unter LibreOffice umgewandelt werden. Dies liegt an den zusätzlichen Funktionen, die in der Version 1.8.* noch nicht vorhanden sind. Dadurch endet der Aufruf mit der Version 1.8.* bereits beim Einlesen der Script-Datei der Datenbank.
Ebenso wenig kann eine externe Datenbank, die einmal mit einer Version der 2er-Reihe bearbeitet wurde, anschließend wieder mit der externen Version 1.8.* bearbeitet werden, die kompatibel zu LibreOffice ist.
Auch kann bei der Nutzung einer anderen Version in der externen Datenbank die Version der internen Datenbank überschrieben werden. Der Aufruf als interne Datenbank ist dann unmöglich.

Parallelinstallation von interner und externer HSQLDB

Die Einbindung der externen Datei hsqldb.jar in den Class-Pfad kann bei unterschiedlichen Versionen dazu führen, dass interne Datenbanken anschließend nicht mehr zu öffnen sind. Base kommt mit den gleichlautenden Treibern nicht zurecht und will die externe Variante auch für die internen Datenbanken nutzen. Das geht beim ersten Öffnen noch gut. Beim zweiten Öffnen kommt dann aber die Meldung, dass die Datenbank nicht mehr zu öffnen ist, da sie mit einer neueren Version der HSQLDB geschrieben wurde.

Dem kann abgeholfen werden, indem für die externen Datenbanken die Datei hsqldb.jar nicht über den Class-Pfad in LO fest eingegeben wird, sondern stattdessen der Class-Pfad für die jeweilige Datenbankdatei über ein Makro geschrieben wird, wie auch *hier* (http://forum.openoffice.org/en/forum/viewtopic.php?f=40&t=61155) zu lesen ist.

```
001 SUB Start
002    Const cPath = "/home/robby/public_html/hsqldb_test/hsqldb.jar"
003    DIM oDataSource AS OBJECT
004    DIM oSettings AS OBJECT
005    DIM sURL AS STRING
006    sURL = ConvertToURL(cPath)
007    oDataSource = ThisComponent.DataSource
008    oSettings = oDataSource.Settings
009    oSettings.JavaDriverClassPath = sURL
010 END SUB
```

Hier wird die Datei «hsqldb.jar» unter Linux in dem o.g. Pfad abgelegt. Anschließend wird dieser Pfad der momentan geöffneten Datenbank als Treiberdatei zugewiesen.

Dieses Makro wird nach dem Öffnen der *.odb-Datei einmal aufgerufen. Es schreibt dann in die in der *.odb-Datei befindlichen Datei content.xml die entsprechende Verbindung zur Java-Klasse:

```
001 <db:data-source-settings>
002    <db:data-source-setting
003       db:data-source-setting-is-list="false"
004       db:data-source-setting-name="JavaDriverClass"
005       db:data-source-setting-type="string">
006    <db:data-source-setting-value>
007       org.hsqldb.jdbcDriver
008    </db:data-source-setting-value>
009    </db:data-source-setting>
010    <db:data-source-setting
011       db:data-source-setting-is-list="false"
012       db:data-source-setting-name="JavaDriverClassPath"
013       db:data-source-setting-type="string">
```

```
014         <db:data-source-setting-value>
015            file:///home/robby/public_html/hsqldb_test/hsqldb.jar
016         </db:data-source-setting-value>
017      </db:data-source-setting>
018   </db:data-source-settings>
```

Letztlich könnte also auch ohne das Makro ein entsprechender Pfad direkt in die content.xml der *.odb-Datei eingetragen werden. Nur ist dieser Weg für den Normalnutzer sicher nicht so komfortabel zu handhaben.

Änderung der Datenbankverbindung zur externen HSQLDB

Die interne HSQL-Datenbank hat den Nachteil, dass die Abspeicherung der Daten innerhalb eines gepackten Archivs erfolgt. Erst mit dem Packen werden alle Daten festgeschrieben. Dies kann leichter zu Datenverlust führen als es bei der Arbeit mit einer externen Datenbank der Fall ist. Im folgenden werden die Schritte gezeigt, die notwendig sind, um den Umstieg einer bestehenden Datenbank vom *.odb-Päckchen zur externen Version in HSQL zu erreichen.

Aus einer Kopie der bestehenden Datenbank wird das Verzeichnis «database» extrahiert. Der Inhalt wird in das oben beschriebene frei wählbare Verzeichnis kopiert. Dabei sind die enthaltenen Dateien um den Datenbanknamen zu ergänzen:

```
Datenbankname.backup
Datenbankname.data
Datenbankname.properties
Datenbankname.script
Datenbankname.log
```

Jetzt muss noch die «content.xml» aus dem *.odb-Päckchen extrahiert werden. Hier sind mit einem einfachen Texteditor die folgenden Inhalte zu suchen, die leider in einer durchlaufenden Zeile stehen, es sei denn, man verwendet für diese Arbeit einen speziellen sog. XML-Editor:

```
001 <db:connection-data>
002    <db:connection-resource xlink:href="sdbc:embedded:hsqldb"/>
003    <db:login db:is-password-required="false"/>
004 </db:connection-data>
005 <db:driver-settings
006    db:system-driver-settings=""
007    db:base-dn=""
008    db:parameter-name-substitution="false"/>
```

Diese Zeilen sind mit der Verbindung zur externen Datenbank zu ersetzen, hier der Verbindung zu einer Datenbank mit dem Namen "medien", die jetzt im Verzeichnis «hsqldb_data» liegt.

```
001 <db:connection-data>
002    <db:connection-resource
003       xlink:href="jdbc:hsqldb:file:/home/robby/Dokumente/Datenbanken/hsqldb_data/
004       medien;default_schema=true;shutdown=true"/>
005    <db:login db:user-name="sa" db:is-password-required="false"/>
006 </db:connection-data>
007 <db:driver-settings
008    db:java-driver-class="org.hsqldb.jdbcDriver"/>
```

Falls, wie oben geschrieben, die Grundkonfiguration der HSQLDB nicht angetastet wurde, stimmt auch der Nutzername und die nicht erforderliche Passworteinstellung.

Nach Änderung des Codes muss die content.xml wieder in das *.odb-Päckchen eingepackt werden. Das Verzeichnis «database» ist in dem Päckchen jetzt überflüssig. Die Daten werden in Zukunft durch die externe Datenbank zur Verfügung gestellt.

Änderung der Datenbankverbindung für einen Mehrbenutzerbetrieb

Für die Mehrbenutzerfunktion muss die HSQLDB über einen Server zur Verfügung gestellt werden. Wie die Installation des Servers erfolgt, ist je nach Betriebssystem unterschiedlich. Für OpenSuSE war nur ein entsprechendes Paket herunter zu laden und der Server zentral über YAST zu starten (Runlevel-Einstellungen). Nutzer anderer Betriebssysteme und Linux-Varianten finden sicher geeignete Hinweise im Netz.

Im Heimatverzeichnis des Servers, unter SuSE /var/lib/hsqldb, befinden sich unter anderem ein Verzeichnis «data», in dem die Datenbank abzulegen ist, und eine Datei «server.properties», die den Zugang zu den (eventuell also auch mehreren) Datenbanken in diesem Verzeichnis regelt.

Die folgenden Zeilen geben den kompletten Inhalt dieser Datei auf dem Rechner wieder.
Es wird darin der Zugang zu 2 Datenbanken geregelt, nämlich der ursprünglichen Standard-Datenbank (die als neue Datenbank genutzt werden kann) als auch der Datenbank, die aus der *.odb-Datei extrahiert wurde.

```
001 # Hsqldb Server cfg file.
002 # See the Advanced Topics chapter of the Hsqldb User Guide.
003
004 server.database.0   file:data/db0
005 server.dbname.0     firstdb
006 server.urlid.0      db0-url
007
008 server.database.1   file:data/medien
009 server.dbname.1     medien
010 server.urlid.1      medien-url
011
012 server.silent       true
013 server.trace        false
014
015 server.port         9001
016 server.no_system_exit         true
```

Die Datenbank 0 wird mit dem Namen "firstdb" angesprochen, obwohl die einzelnen Dateien in dem Verzeichnis data mit "db0" beginnen. Die neue Datenbank wird als "Datenbank 1" hinzugefügt. Hier sind Datenbankname und Dateibeginn identisch.

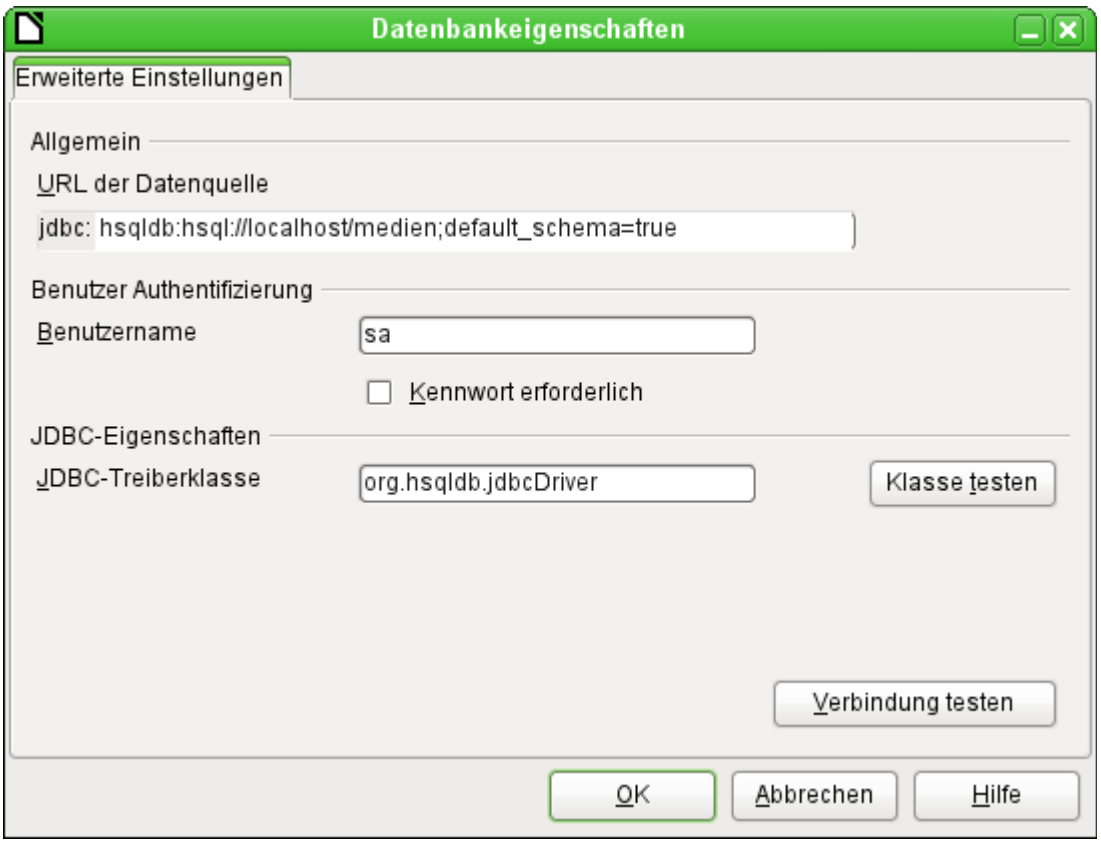
Die beiden Datenbanken werden mit folgenden Zugängen angesprochen:

```
001 jdbc:hsqldb:hsql://localhost/firstdb;default_schema=true
    username sa
    password
001 jdbc:hsqldb:hsql://localhost/medien;default_schema=true
    username sa
    password
```

Die URL wurde hier bereits jeweils um den für den Schreibzugang über die grafische Benutzeroberfläche von LO erforderlichen Zusatz «**;default_schema=true**» ergänzt.

Wenn tatsächlich im Serverbetrieb gearbeitet werden soll, ist natürlich aus Sicherheitsgründen zu überlegen, ob die Datenbank nicht mit einem Passwort geschützt werden soll.

Nun erfolgt die Serververbindung über LO. Im Hauptfenster von Base wird **Bearbeiten → Datenbank → Eigenschaften** aufgesucht.



Mit diesen Zugangsdaten wird auf den Server des eigenen Rechners zugegriffen. Im Netzwerk mit anderen Rechnern müsste dann entweder über Rechnernamen oder die IP-Adresse auf den Server, der ja auf dem aktuellen Rechner läuft, zugegriffen werden.

Beispiel: Der Rechner hat die IP 192.168.0.20 und ist im Netz bekannt mit dem Namen lin_serv. Jetzt ist an anderen Rechnern für die Verbindung zur Datenbank einzugeben:

```
001 jdbc:hsqldb:hsql://192.168.0.20/medien;default_schema=true
```

bzw.:

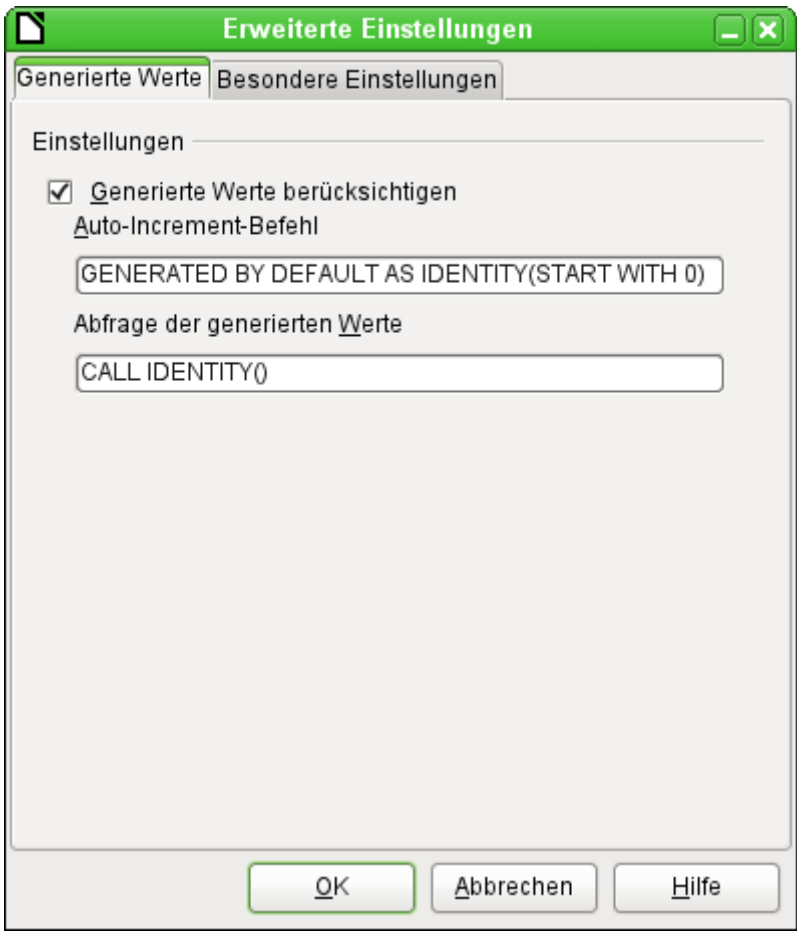
```
001 jdbc:hsqldb:hsql://lin_serv/medien;default_schema=true
```

Die Datenbank ist nun angebunden und kann beschrieben werden. Hier kann allerdings ein zusätzliches Problem auftauchen. Die vorher automatisch generierten Werte werden plötzlich nicht mehr hoch geschrieben. Hier fehlt es noch an einer zusätzlichen Einstellung.

Für die Extraktion und Erzeugung einer Verbindung zur externen neuesten Version der HSQLDB steht auch eine Extension zur Verfügung, die diese Aufgabe erledigen soll: *http://hsqldb.org/web/openoffice.html*

Autoinkrementwerte mit der externen HSQLDB

Für die Nutzung der Auto-Werte müssen je nach Version von Base bei der Tabellenerstellung verschiedene Wege beschritten werden. Allen gleich ist erst einmal der folgende Eintrag unter **Bearbeiten → Datenbank → Erweiterte Einstellungen** erforderlich:



Mit dem Zusatz **`GENERATED BY DEFAULT AS IDENTITY(START WITH 0)`** soll die Funktion des automatisch hoch zählenden Wertes für den Primärschlüssel erstellt werden. Die GUI von LO übernimmt zwar diesen Befehl (auch in der aktuellen LO-Version), schreibt davor aber leider die Anweisung **`NOT NULL`**, so dass die Reihenfolge der Befehlsfolge für die HSQLDB nicht lesbar ist. Hierbei ist zu berücksichtigen, dass die HSQLDB mit dem obigen Befehl ja bereits mitgeteilt bekommt, dass das entsprechende Feld den Primärschlüssel enthält.

Hinweis

In LO ist deshalb die Eingabe des Autowertes in der GUI nicht möglich. Nutzer dieser Versionen erstellen zuerst eine Tabelle mit einem Primärschlüsselfeld ohne Autowert und geben dann direkt über **Extras → SQL** ein:

```
001 ALTER TABLE "Tabellenname"
002 ALTER COLUMN "ID" INT GENERATED BY DEFAULT AS
    IDENTITY(START WITH 0)
```

... wobei davon ausgegangen wird, dass das Primärschlüsselfeld den Namen "ID" hat.

Mit dem Auslesen des letzten Wertes und dem Hochlesen zum nächsten Wert hingegen klappt es in allen Versionen von LO über den Befehl **`CALL IDENTITY()`**. Dies trifft dann z.B. auf die Lösung zu, die Datenbank zuerst einmal als «*.odb-Päckchen» zu erstellen, gründlich zu testen und danach dann die Datenbanktabellen einfach auszulagern.

Sämtliche Abfragen, Formulare und Berichte lassen sich so weiter nutzen, da die Datenbank für die «*.odb-Datei» weiter auf die gleiche Weise angesprochen wird und eventuell spezifische SQL-Befehle mit der externen HSQLDB weiter gelesen werden können.

Umgang mit der internen Firebird-Datenbank

Die interne Firebird-Datenbank ist bisher teilweise nur als experimentelle Funktion verfügbar. Um solch eine Datenbank zu erstellen, muss **Extras → Optionen → LibreOffice → Erweitert → Optionale (instabile) Einstellungen → Experimentelle Funktionen aktivieren** eingeschaltet sein. Schon dieser Weg zeigt auf, dass solch eine Datenbank noch nicht für den täglichen Gebrauch geeignet ist.

Der experimentelle Modus ist für den Betrieb bereits bestehender Firebird-Datenbanken ab LO 6.1 nicht erforderlich. Mit der Version LO 6.4.3 ist Firebird allerdings erst einmal wieder in den experimentellen Status zurück versetzt worden.

Über den folgenden Link können die wesentlichsten Bugs der internen Firebird-Datenbank zusammen mit LibreOffice eingesehen werden: *Gemeldete Bugs für Firebird in Zusammenhang mit Base* (https://bugs.documentfoundation.org/buglist.cgi?bug_status=UNCONFIRMED&bug_status=NEW&bug_status=ASSIGNED&bug_status=REOPENED&bug_status=NEEDINFO&component=Base&known_name=Firebird_open&list_id=522642&product=LibreOffice&query_based_on=Firebird_open&query_format=advanced&short_desc=Firebird&short_desc_type=allwordssubstr) .

Folgende Besonderheit fällt dem Nutzer gegenüber der HSQLDB direkt auf:

Werden neue Daten eingegeben, so werden diese bis zu LibreOffice-Version 7.6 nicht automatisch in der Datenbankdatei abgespeichert. Der Speicherbutton wird bei jeder Dateneingabe erneut aktiviert. Bei der eingebauten HSQLDB ist die Abspeicherung nach Dateneingaben nicht notwendig.

Funktionserweiterungen und -änderungen in Base im Laufe der LO-Versionen

LO 3.6

Der Befehl **`CHECKPOINT DEFRAG`** wird automatisch beim Schließen der Datenbankdatei ausgeführt. Damit wird die Datenbank auf minimale Größe zusammengeschrumpft. Vorher nahmen bereits gelöschte Datenzeilen weiterhin Platz in der Datenbank ein.

LO 4.1

Im **Abfrageeditor** wurde die Einstellung des **Limits** für die auszugebenden Daten möglich. Diese Einstellung arbeitet so auch mit der GUI und berührt nicht die Editierbarkeit von Abfragen.

Bei **Listenfeldern** ist die Angabe des **gebundenen Feldes** auch als 0 (gleiches Feld wie angezeigt) und als -1 (Position des ausgewählten Feldes) möglich.

Wird mit Makros auf ein **Listenfeld** zugegriffen, so ergibt seit dieser Version der **currentValue** den Wert, der an die Datenbank weitergegeben wird, und nicht unbedingt den Wert, den das Feld in dem Formular anzeigt.

Der **Report-Designer** ist in LO als Erweiterung integriert, so dass er nicht mehr als Erweiterung in den Erweiterungsdialogen erscheint.

LO 4.1.2

Der Zugriff auf **Datumswerte** mit Makros in Formularen wurde geändert. Der Datumswert wird jetzt im Datumsfeld als eine Kombination von Tag, Monat und Jahr wiedergegeben und nicht als ISO-Zahlenwert: oFeld.CurrentValue.Year ist so z.B. die Jahresangabe.

LO 4.2

Access2Base ist standardmäßig als Erweiterung in LO integriert.

Firebird ist als experimentelle Datenbank in LO eingebaut. Die HSQLDB bleibt aber weiterhin die Standarddatenbank. Zu einer externen Firebird-Datei kann eine Verbindung auch direkt erstellt werden.

LO 4.3.1

Aliaszuweisungen in der grafischen Benutzeroberfläche von LO erfolgen ohne den Zusatz **AS**. Die Funktionsweise der internen HSQLDB ist davon nicht berührt. Oracle konnte mit dieser Form der Aliaszuweisung nicht umgehen.

LO 4.4

Parameter in Abfragen: Ein leerer Parameter wurde bisher als leerer Text in Abfragen weiter gegeben, innerhalb von Formularen aber als **NULL** angesehen. Dies wurde geändert. Ein leerer Parameter ist jetzt auch in Abfragen **NULL** und kann also z.B. mit **IFNULL** abgefragt werden.

LO 5.0

Grafisches Kontrollfeld: Das grafische Kontrollfeld konnte bis zu diesem Zeitpunkt nur Bildformate einlesen und darstellen. Mit der Version 5.0 ist das Einlesen bzw. Verknüpfen sämtlicher Dateien möglich. Bei *.pdf-Dateien wird in dem Kontrollfeld die 1. Seite dargestellt. Bei unbekannten Formaten bleibt die Anzeige im Kontrollfeld einfach leer.

LO 5.3

Interne Firebird Datenbank: Die interne Datenbank wechselt von der Version 2.5 auf die Version 3.0.0. Die Version 3.0.0 hat keine Abwärtskompatibilität, so dass vorher erstellte interne Firebird-Datenbanken mit der Version LO 5.3 nicht geöffnet werden können.
Die neue Firebird-Version hat ein besonderes Archiv-Format. Ältere Firebird-Datenbanken können nur über das Entpacken der *.odb-Datei und ein Umwandeln mit einer externen Firebird-Datenbank in das Archivformat überführt werden. Ansonsten bietet es sich an, Tabellen älterer Datenbanken ggf. in eine Calc-Datei zu überführen und von dort in die neue Version zu importieren.
Die Verbindung zu einer externen Firebird-Datei funktioniert ab dieser Version nur noch mit Firebird-3-Datenbankdateien.

Hinweis

Die Umwandlung von der alten Firebird Datenbank zur Version 3.0 kann lt. Firebird-Homepage mit Hilfe einer Firebird 2.5 und einer Firebird 3.0 - Serverinstallation erfolgen (*http://www.firebirdsql.org/file/documentation/release_notes/html/en/3_0/rnfb30-compat-upgrade-secdb.html*)

LO 6.0

Symbolleiste «Formular»: Die Symbolleiste wurde mit Elementen aus dem Bereich «Weitere Symbole» erweitert. Bei den weiteren Symbolen fehlt das Tabellen-Kontrollelement. Dieses ist jetzt über den neuen Menüpunkt **Formular → Tabellen-Steuerelement** abrufbar.

LO 6.1

Interne Firebird Datenbank: Für bestehende interne Firebird-Datenbanken ist der experimentelle Status beendet. Neue interne Firebird Datenbanken können weiterhin nur über den experimentellen Modus erstellt werden. Dies bedeutet auch, dass nach weiteren intensiven Tests in einer der kommenden Versionen von LO die Unterstützung für die interne HSQLDB entfallen wird.
Der Migrationsassistent für die Migration der Daten von interner HSQLDB zu interner Firebird-Datenbank ist in dieser Version nur experimentell nutzbar. Vor der Anwendung der Migration sollte unbedingt eine Datensicherung erfolgen.

LO 6.2

MySQL-Verbindung: Die direkte Verbindung zu MySQL- bzw. MariaDB-Datenbanken, die vorher nur als separate Extension verfügbar war, wird jetzt direkt in LO mit eingebaut. Grundlage der Verbindung ist allerdings der MariaDB-C-Connector, der von der LGPL-Lizenz her zu LibreOffice passt.

LO 6.4

Report Builder: Zellen können jetzt mit einer automatischen Zeilenhöhe versehen werden, so dass nicht mehr ein rotes Dreieck am rechten Rand auf fehlenden Inhalt hinweisen muss.

LO 6.4.3

Firebird: Eingebettete Firebird Datenbanken sind nur noch erstellbar, wenn unter **Extras → Optionen → LibreOffice → Erweitert → Optionale Funktionen → Experimentelle Funktionen aktivieren** eingeschaltet ist. Dies liegt daran, dass zu viele kleine Bugs den Betrieb beeinträchtigen und die interne HSQLDB weiterhin besser mit der GUI zusammen arbeitet.

LO 7.0

Formularbearbeitung: Formulare werden seit diesem Update in der ODF-Version 1.3 erweitert gespeichert. Ein mit LO 7.0 abgespeichertes Formular sollte nicht mit einer älteren Version von LO bearbeitet werden. Das Formular ist nach dem Abspeichern sonst nicht mehr aufrufbar. Dieser Bug wurde in den nachfolgenden Versionen behoben.

LO 7.6

Firebird: Eingaben neuer Daten werden jetzt wie bei der internen HSQLDB direkt abgespeichert. Eine zusätzliche Betätigung des Speicherbuttons ist nicht mehr notwendig.

Glossar

Abgesicherter Modus
Der abgesicherte Modus ist ein Modus, in dem LibreOffice temporär mit einem frischen Benutzerprofil startet und die Hardwarebeschleunigung deaktiviert wird. Dies hilft, eine nicht mehr funktionierende LibreOffice-Instanz wiederherzustellen.

https://help.libreoffice.org/latest/de/text/shared/01/profile_safe_mode.html

Absolut speichern
In einigen Dialogen (z. B. **Bearbeiten → AutoText**) können Sie wählen, ob eine Datei relativ oder absolut gespeichert werden soll.

Wenn Sie absolut speichern wählen, werden alle Referenzen auf andere Dateien ebenfalls als absolut definiert, wobei sich diese Definition am jeweiligen Laufwerk, Volumen oder Quellverzeichnis orientiert. Der Vorteil dieser Methode besteht darin, dass das Dokument mit den Referenzen in ein anderes Verzeichnis oder einen anderen Ordner verschoben werden kann und die Referenzen weiterhin gültig bleiben.

Vergleichen Sie auch mit *Relativ speichern*.

Andocken
Einige Fenster in LibreOffice, zum Beispiel das Fenster *Formatvorlagen* und der *Navigator*, sind andockbare Fenster. Sie können diese Fenster verschieben, vergrößern oder sie an einer Kante des Arbeitsbereichs andocken. An jeder Kante können Sie auch mehrere Fenster über oder nebeneinander andocken. Sie können dann durch das Verschieben der Umrandungslinien die relativen Proportionen der Fenster verändern.

Lesen Sie den Abschnitt „Symbolleisten andocken und frei schweben lassen“ im Kapitel 01 „LibreOffice Einführung“ des Handbuchs *Erste Schritte* für eine Beschreibung des Vorgehens.

Andocken (AutoHide)
An jedem Fensterrand, an dem ein anderes Fenster angedockt ist, befindet sich eine Schaltfläche, die zum Einblenden oder Ausblenden des Fensters dient.

Wenn Sie zum Anzeigen des Fensters auf die Schaltfläche am Fensterrand klicken, bleibt das Fenster so lange eingeblendet, bis Sie es mit derselben Schaltfläche wieder ausblenden.

Wenn Sie das Fenster durch Klicken auf den Fensterrand einblenden, aktivieren Sie die Funktion *AutoHide*. *AutoHide* ermöglicht es, ein eigentlich ausgeblendetes Fenster durch Klicken auf dessen Rand kurzzeitig einzublenden. Sobald Sie mit der Maus in den Arbeitsbereich oder ein anderes Fenster klicken, wird das Fenster wieder verborgen.

ANSI
Das "American National Standards Institute" ist ein US-Normungsgremium, vergleichbar mit dem deutschen *DIN*.

ANSI-Zeichensatz
Der ANSI-Zeichensatz stimmt zwar weitgehend, aber nicht vollständig mit dem in den westeuropäischen Windows-Betriebssystemen von Microsoft häufig verwendeten *Windows-Zeichensatz* 1252 überein. Obwohl die gebräuchliche Bezeichnung etwas anderes suggeriert, ist der ANSI-Zeichensatz keine Norm der *ANSI*, sondern wurde durch Microsoft geprägt.

API
Eine API (Application Programming Interface) ist eine Schnittstelle, die es anderen Programmen ermöglicht, sich in das Programm bzw. die Bibliothek zu integrieren. In LibreOffice wird diese Schnittstelle als *Extension* (Erweiterung) bezeichnet.

ASCII

Abkürzung für „American Standard Code for Information Interchange“ (zu deutsch: amerikanischer Standardcode zum Informationsaustausch). ASCII ist ein Zeichensatz für die Zeichendarstellung bei Personal Computern (Zeichenkodierung) und entspricht der US-Variante von *ISO* 646. Er besteht aus 128 Zeichen mit Buchstaben, Ziffern, Satzzeichen sowie Sonderzeichen und dient als Grundlage für spätere auf mehr Bits basierende Kodierungen für Zeichensätze. Dieser Zeichensatz wurde von *MS-DOS* benutzt und ist nicht kompatibel zum *ANSI-Zeichensatz*, *Unicode-Zeichensatz* und *Windows-Zeichensatz*.

Der erweiterte ASCII-Zeichensatz (8 bit) enthält 256 Zeichen. Jedem Zeichen ist eine eindeutige Nummer zugewiesen, die man auch als ASCII-Code bezeichnet.

In *HTML*-Seiten sollten nur die Zeichen des 7-Bit-ASCII-Zeichensatzes vorkommen. Andere Zeichen, wie etwa die deutschen Umlaute, werden durch Umschreibungen gekennzeichnet. So wird das kleine „ü“ etwa zu „ü“. Sie können in LibreOffice Zeichen im erweiterten ASCII-Code eingeben; der Exportfilter sorgt für die erforderliche Umwandlung.

AskLibreOffice

AskLibreOffice ist eine webbasierte Kommunikations-Plattform für den gemeinsamen Austausch von Fragen und Antworten. Auf ihr können Mitglieder der Online-Community von LibreOffice zusammenkommen, um Fragen zu LibreOffice der *TDF* und verwandten Themen zu stellen und zu beantworten.

Hier können Sie sich über die Benutzung von AskLibreOffice Informieren:
Dies ist die Anleitung - Wie benutze ich die Ask Seite?

Austauschformat

Ein Dateiformat, welches dem Austausch von Daten dient.

Das Austauschformat können Grafiken oder allgemeine Daten sein, z. B. von einer Tabellenkalkulation. Austauschformate dienen auch zum Import oder Export von einer Anwendung in eine Andere.

Barcode

Als Barcode (oder Strichcode bzw. Streifencode) wird eine lesbare Schrift, die aus verschieden breiten, parallelen Linien besteht, bezeichnet. Diese Linien werden von einem Scanner gelesen und interpretiert, so dass diese elektronisch weiter verarbeitet werden können. Gängige Vertreter sind hier Etiketten (EAN-Code) oder ISBN-Nummern bei Büchern. Einen „Nachfolger“ stellt der QR-Code dar.

Benutzerprofil

Das *Benutzerprofil* ist der Ordner, in dem alle benutzerbezogenen Daten wie Erweiterungen, benutzerdefinierte Wörterbücher, Dokumentvorlagen usw. gespeichert sind.

Dieser Ordner wird bei der Deinstallation oder Aktualisierung von LibreOffice nicht entfernt, so dass Ihre bisher gemachten Anpassungen beibehalten werden.

Bézierobjekt

Bézierkurven sind nach einem vom französischen Mathematiker Pierre Bézier entwickelten Verfahren mathematisch dargestellte Kurven, wie sie in zweidimensionalen Grafikanwendungen zum Einsatz kommen. Eine solche Kurve definiert sich durch vier Punkte: den Anfangspunkt, den Endpunkt und zwei separate Zwischenpunkte. Ein Bézier-Objekt lässt sich durch Verschieben dieser Punkte mit der Maus verformen.

Bildkompression

Bildkompression beruht wie jede Art der Datenkompression darauf, den ursprünglichen Datensatz entweder über eine *Verlustfreie Kompression* in einer vollständig rekonstruierbaren Form zu speichern oder Daten zu entfernen, deren Verlust kaum wahrnehmbar ist (*Verlustbehaftete Kompression*). Es gibt sehr viele Grafikformate, von denen aber viele veraltet

sind und viele keine Kompression unterstützen, da sie *Austauschformate* für Grafikprogramme sind.

Blog TDF
Blog der The Document Foundation

Complex Text Layout (CTL)
Sprachen mit komplexen Skripten können unter anderem folgende Eigenschaften aufweisen:

- Es werden Zeichen verwendet, die aus mehreren Teilen zusammengesetzt sind.
- Die Schreibrichtung des Texts läuft von rechts nach links.

LibreOffice unterstützt derzeit Hindi, Thai, Hebräisch und Arabisch als CTL-Sprachen. Aktivieren Sie die CTL-Unterstützung unter **Extras → Optionen... → Spracheinstellungen → Sprachen**, um diese zu verwenden.

Dateiendung
Die Dateiendung gibt normalerweise das Format der Datei an und stellt die Buchstabenkombination hinter dem letzten Punkt des Dateinamens dar.

Datenbank
In einer Datenbank, meist Datenbanksystem (DBS) genannt, werden Daten gespeichert, verwaltet und miteinander in Beziehung gebracht. Zusätzlich können Informationen gesucht sowie gefiltert dargestellt werden. Stellt das System weitere Möglichkeiten, wie beispielsweise die Verwaltung von Anwendern, Transaktionskontrolle usw. zur Verfügung, spricht man von einem Datenbankmanagementsystem (DBMS).

dBASE
dBASE war eines der ersten dateibasierten Datenbankverwaltungssysteme für PCs. Die strukturierten Daten wurden in „DataBaseFiles“ (daher die Dateiendung DBF) gespeichert. dBASE-Dateien und -Programme sind heute technisch veraltet, wobei das Dateiformat hin und wieder als Austauschformat verwendet wird.

DDE
DDE steht für "Dynamic Data Exchange", also dem dynamischen Datenaustausch. Dies ist ein Vorgänger von *OLE*, dem "Object Linking and Embedding". Bei DDE wird ein *Objekt* in Form einer *Verknüpfung* zur Datei eingebunden, aber im Gegensatz zu OLE nicht selbst eingebettet.

DDE-Verknüpfungen lassen sich wie folgt erzeugen: Wählen Sie in einem LibreOffice-Calc Tabellendokument Zellen aus, kopieren Sie diese in die *Zwischenablage*, wechseln Sie in ein anderes Tabellendokument, und wählen Sie **Bearbeiten → Inhalte einfügen**. Wählen Sie die Option "Verknüpfen" aus, um den Inhalt als DDE-Verknüpfung einzufügen. Bei Aktivierung der Verknüpfung wird der eingefügte Zellbereich aus der Originaldatei eingelesen.

Debian
Debian ist eine Linux-Distribution, die es seit 1996 gibt und den Linux-Kernel sowie alle für den Betrieb des Systems erforderliche Programme enthält. Seit der Version 6.0 enthält Debian nur noch freie Software.

Diagramme
LibreOffice- Diagramme bietet die Möglichkeit, Diagramme zu erstellen und in Ihre Dokumente einzubetten. Diagramme können nicht nur einfach aus Daten erstellt, sondern auch ihr Typ, ihre Farbe und ihre Größe auf vielfältige Weise angepasst werden. Eine Vielzahl von Diagrammtypen steht zur Auswahl: Kreisdiagramme, Säulen- oder Balkendiagramme, Kursdiagramme, Streudiagramme, 2D und 3D-Optik ...

Diagramme können in mehrere andere Dokumente eingefügt werden: Tabellen, Präsentationen, Zeichnungen oder Textdokumente. Deshalb kann das Diagrammmodul auch aus diesen LibreOffice-Anwendungen heraus aufgerufen werden und die Daten entweder mit in Ihrem Dokument vorhandenen verknüpft oder unabhängig davon festgelegt werden.

DIN

Das "Deutsche Institut für Normung e. V." ist die bedeutendste nationale Normungsorganisation in der Bundesrepublik Deutschland.

Der heutige Name wurde 1975 im Zusammenhang mit dem zwischen der Organisation und der Bundesrepublik Deutschland abgeschlossenen Normenvertrag gewählt. Die unter der Leitung von Arbeitsausschüssen dieser Normungsorganisation erarbeiteten Standards werden als "DIN-Normen" bezeichnet. Das DIN arbeitet in den internationalen und europäischen Normengremien mit, um die deutschen Interessen zu vertreten und den internationalen freien Warenverkehr zu fördern. Es organisiert die Eingliederung internationaler Normen in das deutsche Normenwerk.

Direkte Formatierung

Wenn Sie Ihre Dokumente ohne Hilfe von Formatvorlagen formatieren, spricht man von direkter oder "harter" *Formatierung*. Darunter versteht man die Veränderung von Text oder von einem anderen *Objekt*, wie ein Rahmen oder eine Tabelle, über die Zuweisung verschiedener Attribute. Die Formatierung gilt nur für den ausgewählten Bereich und alle Änderungen müssen einzeln bearbeitet werden.

Direkte Formatierungen können Sie aus Ihrem Dokument entfernen, indem Sie mit den Text markieren und im Menü **Format → Direkte Formatierung löschen** wählen.

Dem gegenüber steht die zu bevorzugende *Indirekte Formatierung*.

Download LibreOffice

Download ist das Herunterladen von Inhalten aus dem Web.
Das LibreOffice-Programm finden Sie zum Download auf der *Webseite von LibreOffice*: *LibreOffice Download*

Drag&Drop

Sehen Sie unter *Ziehen&Ablegen* nach.

Drehfeld

Ein Drehfeld ist eine Eigenschaft eines Zahlen-, Währungs-, Datums- oder Zeitfelds bei Formular-Steuerelementen. Wenn die Eigenschaft *Drehfeld* aktiviert ist, zeigt das Feld zwei Symbole mit Pfeilen an. Diese zeigen entweder senkrecht oder waagerecht in entgegengesetzte Richtungen.

In der Basic IDE wird ein numerisches Feld mit zwei Pfeilsymbolen als Drehfeld bezeichnet. Sie können entweder einen numerischen Wert direkt in das Drehfeld eingeben oder mit dem Auf- und Abwärtspfeil auswählen. Mit den Tasten *Nach oben* und *Nach unten* Ihrer Tastatur lässt sich der Wert im Drehfeld ebenfalls vergrößern oder verkleinern. Mit den Tasten *Bild nach oben* und *Bild nach unten* können Sie den Höchst- und den Mindestwert für das Drehfeld erreichen.

Handelt es sich um ein Drehfeld für numerische Werte, dann können Sie auch eine Maßeinheit angeben, also z. B. „1 cm“, „5 mm“, „12 pt“ oder „2“.

DTP-Programm

Mittels eines DTP-Programms (Desktop-Publishing-Programm) werden hochwertige Dokumente, die aus Texten und Bildern bestehen, erstellt. Diese Dokumente können zum Beispiel für den Druck von Broschüren, Magazinen, Büchern oder Katalogen verwendet werden.

Extension
Eine Extension (deutsch: Erweiterung) ist ein Programm, welches sich in LibreOffice integriert, um dessen Funktionen zu verbessern oder zu ersetzen. Die Extensions steuern LibreOffice über eine *API* an.

Firebird
Firebird ist ein freies Datenbankverwaltungsprogramm, welches auf Interbase basiert. Seit der LibreOffice-Version 4.2 ist Firebird 2.5 in LibreOffice-Base integriert, vorerst aber nur als experimentelle Funktion neben der HSQLDB zugänglich. Ab LO 5.3 wird Firebird 3.0 verwendet. Zukünftig soll Firebird die HSQLDB ablösen.

Formatierung
Unter Formatieren versteht man in diesem Zusammenhang das optische Gestalten von Texten mit einem Textverarbeitungs- oder *DTP-Programm*. Dazu gehören das Festlegen des Papierformats, der Seitenränder, der Schriftarten, der Schrifteffekte sowie der Einzüge und Abstände.

Sie können Text durch *Direkte Formatierung* bei der Eingabe oder mithilfe von Formatvorlagen in LibreOffice formatieren (*Indirekte Formatierung*).

Gallery
Die LibreOffice Gallery ist ein Fenster, in dem Sie Bilder und Klänge in thematischen Ordnern verwalten und zur Nutzung in LibreOffice bereit stellen können.

GIF
Das "Graphics Interchange Format" (deutsch: Grafik-*Austauschformat*) ist ein Grafikformat, das eine gute *Verlustfreie Kompression* für Bilder mit geringer Farbtiefe besitzt (bis zu 256 verschiedene Farben pro Einzelbild).

Darüber hinaus können mehrere Einzelbilder in einer Datei abgespeichert werden, die von geeigneten Betrachtungsprogrammen wie Webbrowsern als Animationen interpretiert werden.

GUI
GUI (engl. **G**raphical **U**ser **I**nterface) (Kurzform **UI**) ist das, was Sie auf dem Bildschirm sehen - also die grafische Benutzeroberfläche und umfasst alle Schaltflächen, die bei der Anwendung von Programmen wie z. B. LibreOffice sichtbar sind. Wenn Sie das Verhalten eines Programmes nur durch Texteingaben in eine Kommandozeile steuern, besitzt dieses Programm keine GUI.

GPL
Die GNU General Public License (GPL) ist eine sehr verbreitete Software-Lizenz, welche den Anwendern die Freiheiten garantiert, die Software zu nutzen, studieren, verbreiten (kopieren) und ändern zu dürfen. Software unter GPL wird als „Freie Software“ bezeichnet. Die aktuelle Version hat die Version 3.0 und ist unter *http://www.gnu.org/licenses/gpl-3.0.html* zu finden.

HSQLDB
HSQLDB (Hyper Structured Query Language Database) ist ein freies, vollständig in Java programmiertes relationales Datenbankverwaltungssystem (RDBMS). Schon zu Zeiten von OpenOffice ist HSQLDB in Base integriert. Der Vorteil von HSQLDB ist, dass es sehr leicht in andere Produkte integriert werden kann (die entsprechende Bibliothek hat nur etwa eine Größe von einem Megabyte).

HTML
Die "Hypertext Markup Language" ist eine textbasierte Auszeichnungssprache zur Strukturierung von Inhalten wie Texten, Bildern und Hyperlinks in Dokumenten.

HTML-Dokumente sind die Grundlage des World Wide Web und werden von einem Webbrowser dargestellt.

Hurenkinderregelung
Schusterjungen und Hurenkinder sind historische Begriffe aus der Typographie, die seit langem benutzt werden.

Ein Hurenkind ist die letzte Zeile eines Absatzes, die alleine am oberen Rand der nächsten Seite steht. Mit einem Textdokument von LibreOffice können Sie diese unschöne Erscheinung automatisch für die gewünschte Absatzvorlage vermeiden. Dabei können Sie sogar wählen, wie viele Zeilen mindestens immer zusammen auf einer Seite gehalten werden sollen.

Dem gegenüber steht die *Schusterjungenregelung*.

IDE
Eine integrierte Entwicklungsumgebung (Integrated development environment, kurz IDE) ist eine Zusammenstellung von Programmierwerkzeugen, um die Erstellung von Software zu vereinfachen. LibreOffice enthält sehr leistungsfähige Werkzeuge, mit denen Sie Ihre Makros ausführen, bearbeiten und Fehler darin finden können.

IEC
Die Internationale Elektrotechnische Kommission (International Electrotechnical Commission) ist eine internationale Normungsorganisation mit Sitz in Genf für Normen im Bereich der Elektrotechnik und Elektronik. Einige Normen werden gemeinsam mit der *ISO* entwickelt.

IME
IME ist eine Abkürzung für "Input Method Editor", also ein Eingabemethoden-Editor. Dies ist ein Programm, das die Eingabe von komplexen Zeichen aus nichtwestlichen Zeichensätzen über eine Standardtastatur ermöglicht.

Indirekte Formatierung
Bei der indirekten oder "weichen" *Formatierung* nehmen Sie Formatierungen nicht direkt am Text vor, sondern durch Zuweisen von Formatvorlagen. Der große Vorteil besteht darin, dass Sie mit der Änderung einer Formatvorlage jedes *Objekt* (Absätze, Rahmen, usw.) ändern, dem Sie diese Formatvorlage zugewiesen haben.

Dem gegenüber steht die *Direkte Formatierung*.

ISO
Die Internationale Organisation für Normung (International Organization for Standardization) ist die internationale Vereinigung von Normungsorganisationen und erarbeitet internationale Normen in allen Bereichen mit Ausnahme der Elektrik, der Elektronik und der Telekommunikation, für welche die *IEC* zuständig ist.

Java
Java ist eine Programmiersprache, die von der Firma Oracle weiterentwickelt (ursprünglich von der Firma Sun Microsystems entwickelt) und bereitgestellt wird. Für die Ausführung von Java-Programmen in einer Anwendung ist die *JRE* (Java Runtime Environment) notwendig.

JDBC
Sie können die *API* der Java DataBase Connectivity (JDBC) zum Herstellen einer Verbindung mit einer *Datenbank* von LibreOffice verwenden. JDBC-Treiber sind in der Programmiersprache *Java* geschrieben. Dabei unterscheidet man vier Typen:

1. JDBC-ODBC bridge plus ODBC driver
   Die Verbindung zur Datenbank wird über ODBC hergestellt. Dabei stellt der JDBC-

Treiber nur eine Kapselung der Aufrufe dar. Zusätzlich muss der ODBC-Treiber installiert werden.

2. Native-API partly-Java driver
   Die Verbindung wird über einen Datenbanktreiber hergestellt. Der JDBC-Treiber ruft die entsprechenden Datenbank-Client-Funktionen auf. Der DBMS-Client muss dabei installiert sein.
3. JDBC-Net pure Java driver
   Hier werden durch den JDBC-Treiber die Aufrufe in das entsprechende Protokoll übersetzt und dann von einem entsprechenden Server (middleware) in das DBMS-Protokoll übersetzt und an das DBMS geschickt.
4. Native-protocol pure Java driver
   Hier werden durch den JDBC-Treiber die Aufrufe direkt in das Netzwerkprotokoll des DBMS übersetzt. Hierzu muss aber das entsprechende proprietäre Protokoll des DBMS unterstützt werden. Dieser Typ ist dann auch komplett plattformunabhängig.

JPEG
JPEG wurde von der Joint Photopraphic Experts Group im Jahr 1992 in der *ISO* Norm 10918-1 vorgestellt und ist heute eines der am meisten genutzten Bildformate.

JPG
Sehen Sie unter *JPEG* nach.

JRE
JRE (kurz für Java Runtime Environment) ist die Laufzeit- bzw. Softwareumgebung, mit der Java-Anwendungen ausgeführt werden können. *Java* und JRE werden von der Firma Oracle entwickelt und bereit gestellt.

LibreOffice ist ohne JRE lauffähig, JRE ist jedoch für die LibreOffice-Komponente Base wichtig.

Kerning
Kerning ist die englische Bezeichnung für eine Unterschneidung oder Spationierung. Darunter versteht man das Verringern oder Vergrößern des Abstandes zwischen Buchstabenpaaren zum optischen Ausgleich des Schriftbildes, z. B. bei „W“ und „a“.

In Kerning-Tabellen ist vermerkt, welche Buchstabenpaare mehr oder weniger Abstand benötigen. Diese Tabellen sind in der Regel Bestandteil der jeweiligen Schrift.

Kontextmenü
Kontextmenüs werden durch Rechtsklick auf ein Objekt oder in eine Markierung aufgerufen. Sie zeigen kontextsensitive Menüeinträge zu dem jeweiligen Objekt bzw. der Markierung in einem Aufklappmenü an. Nahezu überall in LibreOffice sind Kontextmenüs vorhanden.

LGPL
Die GNU Lesser General Public License (LGPL) ist von der GPL abgeleitet, erlaubt aber im Gegensatz zur GPL, Bibliotheken etc. in eigene (proprietäre) Software einzubinden.

Link
Ein Link (oder Hyperlink) verknüpft den Inhalt mit einem anderen Dokument oder einem Abschnitt im gleichen Dokument. Links können sich hinter Texten, Symbolen oder Bildern verbergen.

Makro
Ein Makro ist eine Abfolge von Befehlsschritten, die in einer Programmiersprache wie LibreOffice Basic, JavaScript, BeanShell oder Python definiert werden. Ein Makro kann über den Makrorekorder oder über eigene Textzeilenprogrammierung erstellt und über die Menü- oder *Symbolleiste* abgerufen werden.

MariaDB
MariaDB ist ein relationales Datenbankverwaltungssystem, welches als Fork (Abspaltung) von MySQL entstanden ist. MariaDB steht unter GPL Lizenz, die Client-Bibliotheken unter LGPL.

Markieren
Durch Markieren legen Sie fest, welche Teile eines Textes, einer Tabelle oder eines Bildes Sie bearbeiten wollen. Hierzu führen Sie mit gedrückter Maustaste den Mauszeiger über den gewünschten Bereich.

Migration zu LibreOffice
Wenn Sie von einem anderen Office-Programm zu LibreOffice migrieren, sollten Sie sich hier informieren:
Mein Übergang von MS-Office zu LibreOffice

MS-DOS
MS-DOS, kurz für Microsoft Disk Operating System, war Microsofts erstes Betriebssystem für PCs.

Es wurde ursprünglich für den Intel-Prozessor 8086/8088 entwickelt und war in den späten 1980er und frühen 1990er Jahren das dominierende Betriebssystem für Einzelplatzrechner. Die früheren Windows-Versionen 1.0 bis 3.11, 95 (4.0), 98 (4.1) und ME (4.9) waren von DOS abhängig. Windows NT und die darauf basierenden Microsoft-Betriebssysteme (Windows 2000, XP, Vista und 7) bauen nicht mehr auf MS-DOS auf und können DOS-Software nicht mehr oder nur noch eingeschränkt ausführen.

MySQL
MySQL ist ein sehr verbreitetes relationales Datenbankverwaltungssystem und steht für viele Betriebssysteme zur Verfügung (Unix, Linux, Windows, Mac OS, ...). Es steht unter einem dualen Lizenzsystem, einmal propriertär und einmal unter der GPL. Ursprünglich wurde MySQL von der schwedischen Firma MySQL AB entwickelt und dann von Sun Microsystems aufgekauft, das seinerseits von Oracle gekauft worden ist.

OASIS
Die "Organization for the Advancement of Structured Information Standards" ist eine internationale, nicht-gewinnorientierte Organisation, die sich mit der Weiterentwicklung von E-Business- und Webservice-Standards beschäftigt. Bekannte Standards der OASIS sind u. a. OpenDocument (*ODF*) und DocBook.

OASIS Open Document Format for Office Applications
Sehen Sie unter *ODF* nach.

Objekt
Ein Objekt ist ein Element auf dem Bildschirm, das Informationen enthält. Dabei kann es sich z. B. um Anwendungsdaten handeln, wie Texte oder Grafiken.

Objekte sind selbstständig und beeinflussen sich nicht gegenseitig. Jedem Objekt, das Daten enthält, werden bestimmte Befehle zugeordnet. Ein Grafikobjekt wird so mit Befehlen zur Bildbearbeitung versehen, eine Tabelle mit Befehlen zur Berechnung usw.

ODBC
Open DataBase Connectivity (ODBC) ist ein Protokoll für den Zugriff auf *Datenbank*-Systeme durch Anwendungen. Dabei kommt die Abfragesprache *SQL* zum Einsatz. In LibreOffice können Sie von Fall zu Fall entscheiden, ob die Abfragen durch Eingabe von SQL-Befehlen oder anhand der interaktiven Hilfe erfolgen sollen. Bei letzterer Methode definieren Sie die Abfrage per Mausklick und LibreOffice übersetzt sie dann automatisch in SQL.

Die nötigen 32-Bit ODBC Funktionen installieren Sie mit Hilfe eines Setup-Programms des Datenbankherstellers in Ihr Betriebssystem. Die Eigenschaften bearbeiten Sie dann in der Systemsteuerung.

ODF

Das von OASIS entwickelte ODF ("Open Document Format for Office Applications" kurz: OpenDocument / ODF; deutsch: Offenes DokumentFormat für Büroanwendungen) ist ein international genormter quelloffener Standard für Dateiformate von Bürodokumenten wie Texte, Tabellendokumente, Präsentationen, Zeichnungen, Bilder und Diagramme. Es wurde ursprünglich von Sun entwickelt, durch die Organisation *OASIS* als Standard spezifiziert und 2006 als internationale Norm *ISO/IEC* 26300 veröffentlicht.

OLE

OLE steht für "Object Linking and Embedding", also etwa Verknüpfung und Einbettung von Objekten. Ein OLE-*Objekt* kann wahlweise als *Verknüpfung* in ein Dokument eingefügt oder selbst darin eingebettet werden. Beim Einbetten wird eine Kopie des Objekts zusammen mit Angaben zum Quellprogramm in das Zieldokument eingefügt. Wenn Sie das Objekt bearbeiten möchten, müssen Sie nur darauf doppelklicken und aktivieren so das Quellprogramm.

OpenDocument

Sehen Sie unter *ODF* nach.

OpenCL

OpenCL ist ein Formelinterpreter, der massive parallele Berechnungen von Formelzellen mit GPU (Graphic Prozessor Unit) über OpenCL ermöglicht.

OpenGL

OpenGL ist eine ursprünglich von SGI (Silicon Graphics Inc) entwickelte 3D-Grafiksprache. Zwei Varianten dieser Sprache sind weit verbreitet: das auf die Verwendung unter Windows NT ausgerichtete Microsoft OpenGL und Cosmo OpenGL von SGI. Cosmo OpenGL ist eine für alle Plattformen und Computertypen geeignete, unabhängige Grafiksprache, die sogar auf Systemen ohne spezielle 3D-Grafikhardware eingesetzt werden kann.

PDF

Das Portable Document Format (PDF; deutsch: (trans)portables Dokumentenformat) ist ein plattformunabhängiges Dateiformat für Dokumente, das vom Unternehmen Adobe Systems entwickelt und 1993 veröffentlicht wurde.

Ziel war es, ein Dateiformat für elektronische Dokumente zu schaffen, das diese unabhängig vom ursprünglichen Anwendungsprogramm, vom Betriebssystem oder von der Hardwareplattform originalgetreu weitergeben kann. Ein Leser einer PDF-Datei soll das Dokument immer in der Form betrachten und ausdrucken können, die der Autor festgelegt hat. Die typischen Konvertierungsprobleme (wie zum Beispiel veränderter Seitenumbruch oder falsche Schriftarten) beim Austausch eines Dokuments zwischen verschiedenen Anwendungsprogrammen entfallen.

Neben Text, Bildern und Grafik kann eine PDF-Datei auch Hilfen enthalten, die die Navigation innerhalb des Dokumentes erleichtern. Dazu gehören zum Beispiel anklickbare Inhaltsverzeichnisse und miniaturisierte Seitenvorschauen.

PDF ist mittlerweile weit verbreitet und wird z. B. von vielen elektronischen Zeitschriften (E-Journals) genutzt. Mittlerweile gibt es auf dem Markt zahlreiche Softwareprodukte, die Dateien als PDF erzeugen können, wenn sie auch nicht immer den vollen Funktionsumfang von Adobe Acrobat bieten.

PDF-Dateien sind nicht auf PCs beschränkt, sondern kommen auch z. B. in Druckmaschinen zum Einsatz. Des Weiteren gibt es spezielle Erweiterungen zur Langzeitarchivierung.

Pixel
Das Bild eines digitalen Fotos und eines Monitores setzt sich aus Punkten (Pixeln) zusammen. Die Anzahl der Bildpunkte wird in einem Zahlenpaar angegeben.

Pixelgrafik
Eine Pixelgrafik, auch Rastergrafik genannt, ist eine Form der Beschreibung eines Bildes in Form von computerlesbaren Daten. Rastergrafiken bestehen aus einer rasterförmigen Anordnung von Bildpunkten (*Pixel* genannt), denen jeweils eine Farbe zugeordnet ist. Die Hauptmerkmale einer Rastergrafik sind daher die Bildgröße (Breite und Höhe gemessen in Pixeln) sowie die Farbtiefe.

Die Erzeugung und Bearbeitung von Rastergrafiken fällt in den Bereich der Computergrafik und Bildbearbeitung. Eine andere Art der Beschreibung von Bildern ist die *Vektorgrafik*.

PostgreSQL
PostgreSQL ist ein freies, objektrelationales Datenbankverwaltungssystem (ORDBMS) und ist weitestgehend mit dem SQL-Standard SQL:2008 konform. Zu den relationalen Funktionen kommt beispielsweise die Möglichkeit hinzu, selbstdefinierte Datentypen und Operationen zu verwenden.

Primärschlüssel
Ein Primärschlüssel (PK – primary key) dient zur eindeutigen Kennzeichnung eines Datenbankfeldes. Meist wird für den Primärschlüssel automatisch auch ein Index erstellt. Diese eindeutige Identifikation von Datenbankfeldern wird bei einer relationalen *Datenbank* verwendet, bei denen von einer Tabelle auf die Daten einer anderen Tabelle zugegriffen werden kann. Wird von einer anderen Tabelle auf einen Primärschlüssel verwiesen, so bezeichnet man ihn als Fremdschlüssel (FK – foreign key). Durch die beiden Schlüssel wird die Referentielle Integrität sichergestellt, was vereinfacht bedeutet, dass der Datensatz des Fremdschlüssels nur auf existierende Datensätze des Primärschlüssels verweisen darf. Dies wird vom DBMS-System sichergestellt.

In LibreOffice definieren Sie Primärschlüssel in der Entwurfsansicht einer Tabelle, indem Sie im *Kontextmenü* eines Zeilenkopfes für das ausgewählte Feld den entsprechenden Befehl wählen.

PNG
"Portable Network Graphics" (deutsch: portable Netzwerkgrafik) ist ein *Rastergrafik*-Format, das eine *Verlustfreie Kompression* benutzt. Es wurde als freier Ersatz für das ältere, bis zum Jahr 2004 mit Patentforderungen belastete Format *GIF* entworfen und ist weniger komplex als *TIFF*. Die Dateien werden mit einem wählbaren Faktor und, im Gegensatz zu *JPG*, stets verlustfrei komprimiert. PNG unterstützt neben unterschiedlichen Farbtiefen auch Transparenz per Alphakanal.

QR-Code
Der QR-Code (Quick Response Code) ist ein zweidimensionaler Code, der ursprünglich für die Markierung von Baugruppen und Komponenten entwickelt worden ist. Der Code enthält redundate Daten, so dass in einem gewissen Umfang eine Fehlerkorrektur möglich ist, wenn beispielsweise durch Verschmutzung nicht alles erkennbar ist (Level H erlaubt, dass 30% wiederhergestellt werden können). Heute wird der QR-Code nicht nur als Produktetikette verwendet, sondern beispielsweise für URLs (Verweise auf Webseiten), als Visitenkarte (VCARD) usw.

Rastergrafik
Sehen Sie unter *Pixelgrafik* nach.

Registerhaltigkeit
Registerhaltigkeit ist ein Begriff aus der Typographie. Darunter versteht man den deckungsgleichen Abdruck der Zeilen eines Satzspiegels auf der Vorder- und Rückseite von Büchern,

Zeitschriften und Zeitungen. Mit der Funktion Registerhaltigkeit können diese Seiten einfacher gelesen werden, indem verhindert wird, dass graue Schatten durch die Textzeilen hindurch scheinen. Von Zeilenregisterhaltigkeit spricht man auch dann, wenn sich bei Textspalten alle nebeneinander liegenden Zeilen auf gleicher Höhe befinden.

Wenn Sie einen Absatz, eine Absatzvorlage oder eine Seitenvorlage als registerhaltig definieren, werden die Grundlinien der betroffenen Zeichen an einem vertikalen Seitenraster ausgerichtet. Dabei spielt die Schriftgröße oder das Vorhandensein einer Grafik keine Rolle. Sie können die Einstellung für dieses Raster auch als eine Eigenschaft der Seitenvorlage festlegen.

Relationale Datenbank

Eine relationale *Datenbank* ist ein Datenbanksystem, in dem Daten in Form miteinander verbundener Tabellen verwaltet werden. Die Daten können in verschiedener Weise abgefragt werden, ohne dass die zugrunde liegenden Tabellen reorganisiert werden müssen.

Ein relationales Datenbankverwaltungssystem (RDBMS) ist ein Programm, mit dem Sie relationale Datenbanken erstellen, aktualisieren und verwalten können. Ein RDBMS akzeptiert *SQL*-Anweisungen, die entweder vom Benutzer eingegeben werden oder in einer Anwendung enthalten sind, und erzeugt, aktualisiert oder ermöglicht den Zugriff auf Datenbanken.

Als typisches Beispiel lässt sich eine Datenbank mit Kunden-, Verkaufs- und Rechnungstabellen heranziehen. In der Rechnungstabelle sind nicht die eigentlichen Kunden- oder Verkaufsdaten, sondern Referenzen, in Form von relationalen Verknüpfungen, oder Relationen auf die Tabellenfelder mit den entsprechenden Kunden- und Verkaufsdaten (z. B. das Kundennummernfeld aus der Kundentabelle) enthalten.

Relativ speichern

In einigen Dialogen (z. B. **Bearbeiten → AutoText**) können Sie wählen, ob eine Datei relativ oder absolut gespeichert werden soll.

Wenn Sie sich für das relative Speichern entscheiden, werden Referenzen auf eine eingebettete Grafik oder ein anderes *Objekt* im Dokument relativ zur Position im Dateisystem gespeichert. In diesem Fall spielt es keine Rolle, wo die referenzierte Verzeichnisstruktur eingetragen ist. Solange die Referenz auf derselben Festplatte bzw. demselben Volume bleibt, werden die Dateien unabhängig vom Speicherort immer aufgefunden. Dies ist für solche Dokumente von besonderer Bedeutung, die auch auf Computern mit einer möglicherweise ganz anderen Verzeichnisstruktur oder anderen Laufwerks- oder Volume-Namen verwendet werden sollen. Auch für das Anlegen von Verzeichnisstrukturen auf einem Internetserver empfiehlt es sich, Dokumente relativ zu speichern.

Vergleichen Sie auch mit *Absolut speichern*.

Release Notes

Beschreiben im Wiki der *TDF* die einzelnen LibreOffice-Versionen und deren Änderungen. Beispiel Release Version 7.5 in Englisch: *LibreOffice 7.5 Community: Release Notes*

RTF

RTF (Rich Text Format) ist ein für den Austausch von Textdateien entwickeltes Dateiformat. Es zeichnet sich dadurch aus, dass Formatierungsinformationen in direkt lesbare Textdaten konvertiert werden. Leider entstehen dabei im Vergleich zu anderen Formaten recht große Dateien.

Schusterjungenregelung

Schusterjungen und Hurenkinder sind historische Begriffe aus der Typographie, die seit langem benutzt werden.

Ein Schusterjunge ist die erste Zeile eines Absatzes, die alleine am unteren Rand der Vorseite steht. Mit einem Textdokument von LibreOffice können Sie diese unschöne Erscheinung automatisch für die gewünschte Absatzvorlage vermeiden. Dabei können Sie sogar

wählen, wie viele Zeilen mindestens immer zusammen auf einer Seite gehalten werden sollen.

Dem gegenüber steht die *Hurenkinderregelung*.

Sparkline
Miniaturdiagramm, dass in LibreOffice-Calc in einer Zelle dargestellt wird. Es muss zuvor ein Zellbereich mit Daten ausgewählt werden.

Spracheinstellungen
Spracheinstellungen können Sie in LibreOffice per Menü **Extras → Optionen → Spracheinstellungen → Sprache** vornehmen.

SQL
SQL (Structured Query Language) ist eine Sprache zur Definition von Datenstrukturen und zur Spezifikation von Abfragen einer relationalen *Datenbank*. In LibreOffice haben Sie die Möglichkeit, Abfragen entweder in SQL oder mithilfe der Maus zu definieren.

SQL (damals SEQUEL) wurde im Jahre 1968 im Rahmen einer studentischen Arbeit in Berkley entwickelt und 1986 als ANSI-Standard (SQL1) verabschiedet. Die letzte Änderung war 2011, daher wird diese Version als SQL:2011 bezeichnet.
SQL teilt sich in folgende Unterbereiche auf:

- DDL: Data Defintion Language
  z. B. CREATE TABLE
- DCL: Data Control Language
  GRANT, REVOKE
- DML: Data Manipulation Language
  z. B. INSERT, UPDATE, DELETE
- DQL: Data Query Language
  SELECT
- TCL: Transaction Control Language
  COMMIT, ROLLBACK

SQL-Datenbank
Eine *SQL*-Datenbank ist ein *Datenbank*-System, das eine SQL-Schnittstelle bietet. SQL-Datenbanken werden oft in Client/Server-Netzwerken eingesetzt, in denen verschiedene Clients auf einen zentralen Server (z. B. einen SQL-Server) zugreifen, daher bezeichnet man sie auch als SQL-Server-Datenbanken oder kurz SQL-Server.

In LibreOffice können Sie externe SQL-Datenbanken einbinden. Diese können sich sowohl auf einer Festplatte des Rechners, auf dem LibreOffice läuft, als auch im Netzwerk befinden. Der Zugriff erfolgt entweder über *ODBC*, *JDBC* oder über einen in LibreOffice integrierten systemeigenen Treiber.

SQL-Server
Sehen Sie unter *SQL-Datenbank* nach.

SWF
Das Kürzel „SWF“ steht für **S**hock**w**ave **F**lash. Unter dem Namen Shockwave vermarktete der damalige Hersteller Macromedia nicht nur Flash, sondern auch eine um 3D-Funktionen, eine objektorientierte Sprache und andere Features erweiterte Variante, die mit Adobe Director produziert werden kann. Während das Shockwave-Format von Anfang an für eine rechenintensive Nutzung konzipiert war, sollte mit dem Webbrowserplugin Flash ein Präsentationsformat geschaffen werden, welches der Universalität des Internets in Bezug auf Hardwareausstattung und Bandbreite entspricht.

Symbolleiste
Ein kleines Fenster mit mehreren Symbolen (Icons), mit denen gängige Aufgaben erledigt werden. Die Symbolleisten können in LibreOffice an beliebigen Stellen positioniert werden.

TIF
Sehen Sie unter *TIFF* nach.

TIFF
Das "Tagged Image File Format" (deutsch: Markiertes Grafikdatei Format) ist ein Dateiformat zur Speicherung von Bilddaten. Das TIFF-Format wurde ursprünglich von Aldus (1994 von Adobe übernommen) und Microsoft für die Farbseparation bei einer gescannten *Pixelgrafik* entwickelt.

Größter Nachteil von TIFF ist seine Komplexität. Die Vielfalt möglicher gültiger TIFF-Dateien kann nur schwer von einzelnen Programmen unterstützt werden. In der Spezifikation des Dateiformates ist deswegen eine Untermenge gültiger TIFF-Dateien definiert, die jedes TIFF-fähige Programm verarbeiten können sollte, genannt Baseline TIFF.

Unicode-Zeichensatz
Unicode ist ein internationaler Standard, in dem langfristig für jedes sinntragende Schriftzeichen oder Textelement aller bekannten Schriftkulturen und Zeichensysteme ein digitaler Code festgelegt wird. Ziel ist es, die Verwendung unterschiedlicher und inkompatibler Kodierungen in verschiedenen Ländern oder Kulturkreisen zu beseitigen. Unicode wird ständig um Zeichen weiterer Schriftsysteme ergänzt. ISO 10646 ist die von *ISO* verwendete, praktisch bedeutungsgleiche Bezeichnung des Unicode-Zeichensatzes; er wird dort als "Universal Character Set" (UCS) bezeichnet.

Es gibt drei verschiedene Hauptausführungen der Unicode-Zeichensätze, *UTF-8*, *UTF-16* und *UTF-32 und folgende Anpassungen bzgl. der Byte-Reihenfolge Big-Endian*(BG) und Little-Endian (LE): UTF-16BE, UTF-16LE, UTF-32BE, UTF-32LE. Bei Big-Endian wird das höchstwertige Byte zuerst gespeichert, bei Little-Endia zuletzt. Die Byte-Reihenfolge wird meist über die Byte Order Mark (BOM) umgesetzt. Dabei wird am Anfang eine bestimmte Bytefolge hinzugefügt, die die Bytereihenfolge festlegt.

- UTF-8: 00 00 FE FF
- UTF-16LE: FF FE
- UTF-16BE: FE FF
- UTF-32LE: FF FE 00 00
- ITF-32BE: 00 00 FE FF

UTF-8
UTF-8 (Abk. für 8-bit UCS Transformation Format) ist die am weitesten verbreitete Kodierung vom *Unicode-Zeichensatz*.

Bei der UTF-8 Kodierung wird jedem Unicode-Zeichen eine speziell kodierte Bytekette variabler Länge zugeordnet. UTF-8 unterstützt bis zu vier Byte, mit denen sich zur Zeit ca. 113.000 Zeichen abbilden lassen. Dabei sind die ersten 128-Zeichen wie ASCII-7bit kodiert. UTF-8 hat eine zentrale Bedeutung als globale Zeichenkodierung im Internet. Die „Internet Engineering Task Force“ verlangt von allen neuen Internetkommunikationsprotokollen, dass die Zeichenkodierung deklariert wird und dass UTF-8 eine der unterstützten Kodierungen ist. Das Internet Mail Consortium (IMC) empfiehlt, dass alle E-Mail-Programme UTF-8 darstellen und senden können. Auch bei dem in Webbrowsern verwendeten *HTML* setzt sich UTF-8 zur Darstellung sprachspezifischer Zeichen zunehmend durch und ersetzt die vorher benutzten HTML-Entities (Entity ist der Name für ein bestimmtes Sonderzeichen. Entities werden durch ein „&“-Zeichen am Anfang und ein Semikolon am Ende gekennzeichnet).

UTF-8 ist die Standard-Zeichenkodierung von MacOS X und einigen Linux-Distributionen. Windows kann zwar mit UTF-8 umgehen, benutzt aber intern einen anderen Zeichensatz (*Windows-Zeichensatz*). Speichern Sie ein Dokument in einem *ODF*-Format, ist es immer UTF-8 kodiert.

UTF-16
UTF-16 ist eine Kodierung vom *Unicode-Zeichensatz*, die für die häufig gebrauchten Zeichen optimiert ist. Es ist das älteste der Unicode-Kodierungsformate und wird standardmäßig in Java intern verwendet, daher hat ein char in Java 16 Bit.

UTF-32
UTF-32 ist eine Kodierung vom *Unicode-Zeichensatz*, bei der jedes Zeichen mit vier Byte (32 Bit) kodiert wird. Sie kann deshalb als die einfachste Kodierung bezeichnet werden, da alle anderen UTF-Kodierungen variable Bytelängen benutzen. Der entscheidende Nachteil von UTF-32 ist der hohe Speicherbedarf. Bei Texten, die überwiegend aus lateinischen Buchstaben bestehen, wird verglichen mit dem verbreiteten UTF-8-Zeichensatz etwa der vierfache Speicherplatz belegt, bei asiatischen Zeichen beispielsweise bietet UTF-32 einen Vorteil.

Vektorgrafik
Eine Vektorgrafik ist eine Computergrafik, die aus grafischen Primitiven wie Linien, Kreisen, Polygonen oder allgemeinen Kurven (Splines) zusammengesetzt ist. Meist sind mit Vektorgrafiken Darstellungen gemeint, deren Primitiven sich zweidimensional in der Ebene beschreiben lassen. Eine Bildbeschreibung, die sich auf dreidimensionale Primitiven stützt, wird eher 3D-Modell oder Szene genannt.

Um beispielsweise das Bild eines Kreises zu speichern, benötigt eine Vektorgrafik mindestens zwei Werte: die Lage des Kreismittelpunkts und den Kreisdurchmesser. Neben der Form und Position der Primitiven werden eventuell auch die Farbe, Strichstärke, diverse Füllmuster und weitere, das Aussehen bestimmende Daten, angegeben. Damit kann sie, im Gegensatz zu einer *Pixelgrafik* verlustfrei vergrößert oder verkleinert werden.

Verknüpfung
Verknüpfungen dienen zum schnellen Aufrufen von Dateien auf der Arbeitsoberfläche. Mit einem einfachen oder einem Doppelklick (je nach Einstellung des Betriebssystems) öffnet sich das entsprechende Programm.

Verlustbehaftete Kompression
Bei der verlustbehafteten Kompression wird versucht, den Informationsverlust unmerklich oder wenigstens ästhetisch erträglich zu halten. Diese Methoden nutzen aus, dass kleine Farbänderungen für das Auge nicht sichtbar sind. Ähnlich wie bei der verlustbehafteten Audiokomprimierung basiert die Bildkomprimierung auf einem Modell der menschlichen Wahrnehmung. Der Komprimierungsalgorithmus soll bevorzugt die Bildinformationen entfernen, die über die Aufnahmefähigkeit der menschlichen Bildwahrnehmung hinausgehen. Das Wahrnehmungsmodell ist jedoch, im Gegensatz zur Audiokompression, nicht explizit formuliert und in die Algorithmen eingearbeitet, sondern mehr intuitiv.

Technisch bedingt ist eine Wiederherstellung des Originalzustandes nicht mehr möglich.

Im Gegensatz dazu steht die *Verlustfreie Kompression*.

Verlustfreie Kompression
Bei der verlustfreien Kompression geht keine Information verloren. Die Daten werden nur anders als vorher organisiert, indem bestimmte Redundanzen erkannt und zusammengefasst werden. Zum Beispiel können sich wiederholende Bitfolgen einmal in einem Wörterbuch abgelegt und dann nur noch durch ihre Nummer repräsentiert werden. Es können beliebige allgemeine Komprimierungsverfahren verwendet werden, die sich auch auf andere Arten von Daten wie Text anwenden lassen.

Dadurch, das keine Information verloren geht, ist die Wiederherstellung des Originalzustandes jederzeit wieder möglich.

Im Gegensatz dazu steht die *Verlustbehaftete Kompression*.

Windows-Zeichensatz
„Windows-1252 Westeuropäisch (Western European)“ ist eine 8-Bit-Zeichenkodierung des Microsoft-Betriebssystems Windows, die die meisten westeuropäischen Sprachen unterstützt. Sie baut auf *ISO*-8859-1 und ISO-8859-15 auf.

Manche Applikationen vermischen die Definition von ISO-8859-1 und Windows-1252. Diese Codierungen unterscheiden sich jedoch nur in den nichtdruckbaren Steuerzeichen. Da diese beispielsweise in *HTML* keine Bedeutung haben, werden oft die druckbaren Zeichen aus Windows-1252 verwendet. Aus diesem Grund schreibt der neue HTML5-Standard vor, dass als ISO-8859-1 markierte Texte als Windows-1252 zu interpretieren sind.

XHTML
XHTML (Extensible Hypertext Markup Language) ist eine erweiterte Version des *HTML*.

XML
Die Extensible Markup Language (engl. für „erweiterbare Auszeichnungssprache“), abgekürzt XML, ist eine Auszeichnungssprache zur Darstellung hierarchisch strukturierter Daten in Form von Textdaten. XML wird u. a. für den plattform- und implementationsunabhängigen Austausch von Daten zwischen Computersystemen eingesetzt, insbesondere über das Internet. Ein XML-Dokument besteht aus Textzeichen (meist *UTF-8* Kodierung, aber jede andere Kodierung ist auch möglich), und ist damit menschenlesbar – Binärdaten enthält es per Definition nicht. Alle wesentlichen Dokumentteile eines *ODF*-Dokuments sind intern in XML verfasst.

Zahlensystem
Ein Zahlensystem ist durch die Anzahl der zur Darstellung von Zahlen verfügbaren Zeichen gekennzeichnet. So basiert beispielsweise das Dezimalsystem auf zehn Ziffern (0 bis 9 – Basis 10), das Binär- oder Dualzahlensystem auf zwei Ziffern (0 und 1 – Basis 2) und das Hexadezimalsystem auf 16 Zeichen (0 bis 9 und A bis F – Basis 16).

Ziehen&Ablegen
Ziehen&Ablegen (auch unter Drag&Drop bekannt) bezeichnet eine bestimmte Mausbewegung, um ein *Objekt* zu kopieren oder zu verschieben. Dabei wird ein Objekt markiert, bei festgehaltener linker Maustaste von einem Platz zu einem anderen gezogen und dort mit Loslassen der Maustaste abgelegt. Dies kann innerhalb eines Dokumentes aber auch von einem Dokument zu einem anderen erfolgen.

Zwischenablage
Die Zwischenablage ist ein virtueller Speicher zum Austausch von Daten zwischen mehreren Programmen. Mit Herunterfahren des PCs werden die Daten der Zwischenablage gelöscht.

Weitere Informationen

Dokumentation / Handbücher

Hier finden Sie die **Dokumentation / Handbücher**:
https://de.libreoffice.org/get-help/documentation/

Noch Fragen?

Für Fragen zum Thema gehen Sie zu **Ask.LibreOffice**:
https://ask.libreoffice.org

Mitarbeit bei LibreOffice

Besuchen Sie uns auch auf der Seite **Dabei sein!**:
https://de.libreoffice.org/community/get-involved/

Spenden

LibreOffice ist freie Software und wird kostenfrei zur Verfügung gestellt. Ihre Spende, die rein freiwillig ist, unterstützt unsere weltweite Gemeinschaft. Wenn Ihnen die Software gefällt, freuen wir uns über eine **Spende** auf *https://de.libreoffice.org/donate/*.

Stichwortverzeichnis

Abfragen....25, 277
  Beziehungsdefinition....289
  Bezugstabellen....328
  Eingabe....277
  Eingabemöglichkeiten....320
  Ergebnis....281
  Erweiterungen mit SQL....298
  Funktionen....285
  in Formularen verwenden....318
  Korrelierte Unterabfrage....328
  Parameter....325
  per GUI....277
  SQL....282
  Tabellenansichten....333
  Unterabfragen....327
  Zeilenumbruch....469
  Zusammenfassung....332
Abgesicherter Modus....680
Addon....369
Alter ermitteln....460
Anmeldung der Datenbank....392
AskLibreOffice....681
Barcode....635, 681
Benutzeroberfläche....132
Benutzerprofil....681
Berichte....25, *338*
Beziehungen....23, 125
  Eins-zu-Eins (1:1)....126
  Eins-zu-Viele (1:n)....125
  Viele-zu-Viele (n:m)....126
Blog TDF....682
Codeschnipsel....459
Comboboxen....534
Container....20
Daten....
  aus Calc exportieren....419
  Einfügen als Felder....396, 398
  Felder....398
  Gruppieren....470
  in Calc einfügen....414
  Text....395
  über Zwischenablage einfügen....421
  Zusammenfassen....470
Datenbank....18, 64, 682
  Anbindung....392
  anlegen....64
  anmelden....76
  Assistent....71
Aufgaben....431
aus Calc importieren....419
Autowerte neu einstellen....627
Benutzername....74
Calc....414
Database management system....13
Datenbankmanagementsystem....682
Datenbanksystem....682
dBase....88
dBASE....682
DBMS....13, 18, 682
DBS....682
Eigenschaften....627
Einträge bearbeiten....630
extern....67
externe HSQLDB einbinden....669
Firebird....684
Geschwindigkeit....64, 632
HSQLDB....68, 684
Informationstabellen....657
intern....64
JDBC....68, 78
Kennwort....74
Konvertieren....421
Managementsystem....18
MySQL....65, 687
Name....73
neu erstellen....72
odb....66
ODBC....68, 77
ORDBMS....689
PostgreSQL....689
RDBMS....684, 690
relational....690
Reparatur....660
übers Netz ansprechen....73
unnötige Einträge....629
Verbindung herstellen....67
Versionsprobleme....669
Verwaiste Einträge ermitteln....632
Wartung....627
Wiederherstellung....661
Datenfilterung....431
  Abfrage....431
  Suche....433
  Suchen und Filtern....264
Datenquelle....392
  Browser....393
  Calc....392
  Writer....392
Defaultwert....131
Diagramme....682
  in Berichten....352

in Formularen....448
Download LibreOffice....683
Drehfeld....683
Drucken....
aus externem Formular....579
aus Formularen direkt....274
Berichtsstart aus Formular....577
mit Formatierung im Hintergrund....577
Serienbriefdruck in Base starten....580
über Berichte....338
über Textfelder....581
Erweiterung....684
Etikettendruck....392
Extension....684
FAQ....14, 16
Feld....24
Firebird....*18,* 652, 677
Formular....22
Ansicht....256
Daten....187
Eigenschaften....187
Entwurf....184
Ereignisse....189
erstellen....181
Fehlermeldungen....263
Feld....185
Hauptformular....242
Kontrollfelder....190
Name....187
Navigator....184, 259
Unterformular....242
Formularfelder....185
Beschriftungsfeld....218
Datumsfeld....200
Drehfeld und Bildlaufleiste....226
Formatiertes Feld....202
Grafische Schaltfläche....224
Grafisches Steuerelement....214
Gruppierungsrahmen....219
Kombinationsfeld....209
Listenfeld....24, *204*
Markierfeld....212
Maskiertes Feld....215
Navigationsleiste....224
Numerisches Feld....199
Optionsfeld....213
Schaltfläche....222
Tabellen-Kontrollfeld....217
Textfeld....198
Verstecktes Steuerelement....227
Währungsfeld....201
Zeitfeld....201
Forum....14
Fremdschlüssel....24, 128
Funktionen....638
Datenbankverbindung....646
Datum/Zeit....643
Numerisch....638
System....648
Text....640
GUI....684
Handbuch....13
Hilfesystem....13
HSQLDB....*18,* 652
Eingaben....146
externe Verbindung....669
Mehrbenutzerbetrieb....674
Version aktuell....147
Version in Base....147
Hyperlink....686
IDE....685
Index....139
Java....685
Archiv einbinden....69
Class-Path....70
Java Runtime Environment....686
JRE....686
Verzeichnispfad....69
JDBC....69, 79, 685
Kombinationsfelder....533
Kontextmenü....686
Kontostand nach Kategorie ermitteln....465
Link....686
localhost....73
Mailinglisten....14
Makros....686
allgemein....474
Basic....474
benutzen....478
Bestandteile....482
Datenbankaufgaben erweitern....564
Datenbanksicherungen....571
Dialoge....602
DIM....482
Editor....474
Filtern von Datensätzen....507
Fremdschlüsssel übertragen....536
Kontrollfunktionen....543
Mailprogramm....590
Makrosicherheit....475
Navigation Formular....543
Prozedur....474
Rahmen....482
Suchen von Datensätzen....525

Variablen definieren....482
verwalten....474
Zugriff auf Elemente....485
Zugriff auf Formular....484
zuweisen....479
MariaDB....687
Markieren....687
Mehrfachselektion....228
MySQL....
JDBC....69
ODBC....70
ODBC-Verbindung....70
Navigationsleiste....224
NULL....141, 157
Objekt....687
ODBC....70, 687
OpenCL....688
PDF....688
Primärschlüssel....23, 125, 128, 689
foreign key....689
Fremdschlüssel....689
PK....689
primary key....689
Programmiersprachen....
Basic....686
BeanShell....686
JavaScript....686
Python....686
QR-Code....689
Referentielle Integrität....24
Relation....23, 290
Release Notes....690
Report-Designer....339
aufrufen....339
Bedingte Anzeige....367
Bedingte Formatierung....368
Benutzerdefinierte Funktionen....365
Dateneigenschaften von Feldern....357
Diagramme einbinden....352
Eigenschaften von Feldern....347
Formeleingabe....367
Formeleingaben....358
Funktionen....358, 360
RESTRICT....154
Schaltfläche....222
Schlüsselfeld....23
Serienbrief....27, 65
Seriendruck....398
Assistent....398
Etiketten....405
Externe Formulare....411
Feldbefehl....409
Felder....409
Serienbrief....399
Serienbriefassistent....399
Sparkline....691
Speicherfresser....124
speichern....
absolut....680
Spracheinstellungen....691
SQL....130, 691
Alias....314
ALTER COLUMN....627
ALTER TABLE....151, 627
Bedingungen....166
CASCADE....154
CASEWHEN....460
Char....469
COUNT....466
CREATE TABLE....147
CURDATE....460
DATEDIFF....460
Datenbank....691
DAYOFYEAR....460
DCL....691
DDL....691
DELETE....169
direkte Eingabe....145
DISTINCT....299, 432
DML....691
DQL....691
DROP TABLE....154
Fehlercode....287
GENERATED BY DEFAULT....676
GROUP BY....286, 299
Group_Concat....470
HAVING....299
IF EXISTS....154
INSERT INTO....167
LIKE....434
LIMIT....299
Listenfelder....315
LOWER....434
ORDER BY....299, 432
RESTART WITH....627
RESTRICT....154
SELECT....282, 299, 432
SEQUEL....691
Server....691
SQL:2011....691
Sub-Select....432
SUM....466
Tabelle erstellen....147
Tabelle löschen....154

Tabellenänderung....151
TCL....691
UNION....299
UPDATE....168
WHERE....299
Streifencode....681
Strichcode....681
Support....14
Symbolleiste....692
Tabellen....23, 124
allgemein....124
AutoWert....136, 160
Bezeichnung....130
Beziehung....124
Datentypen....124
Defaultwert....131
Dokument....90
Editor....
Datentypen....635
Eingabe von Daten....160
Eltern-Tabelle....156
filtern....165
Index....139
NULL....141
Quellen....90
sortieren....163
Sortierung....139
Spalten ausblenden....160
Suchfunktion....163
Timestamp....137
Unterschied zu Tabellenkalkulation....124
verknüpfen....155
Wiederholungen....124
Zeitfelder....137
Tabellenkalkulationen....18
Tabellenkontrollfeld....237
Thunderbird Adressbuch....95
Treiberverbindungen aufbewahren....115
Unicode....692
uno-Befehle....656
URL....689
UTF....
BOM....692
UTF-16....693
UTF-16BE....692
UTF-16LE....692
UTF-32....693
UTF-32BE....692
UTF-32LE....692
UTF-8....692
View....*333,* 393
Zahlensystem....694
Zeichensatz....
ASCII....681
ISO-8859-1....694
ISO-8859-15....694
UCS....692
UTF....692
Windows-1252....694
Zeilennummerierung....466

LibreOffice
The Document Foundation

Base-Handbuch

Verwalten Sie ihre Daten

Zu diesem Buch:
Dieses Buch bietet einen Einblick in das Datenbankmodul Base der Office Suite LibreOffice. Egal, ob Sie noch nie mit Datenbanken gearbeitet haben oder ob Sie ein anderes Datenbankprogramm aus einer Office-Suite gewohnt sind.
Dieses Buch führt Sie in die gebräuchlichsten Funktionen von LibreOffice-Base ein:
- Datenbank erstellen
- Ein- und Ausgabe der Datenbank: Tabellen, Formulare, Abfragen, Berichte
- Aufgaben einer Datenbank
- Makros
- Pflege und Wartung von Datenbanken und einiges mehr

Es zeigt Wege zur Migration von der bisherigen internen HSQLDB zur Firebird Datenbank.

Zu den Autoren:
Dieses Buch entstand durch Freiwillige der LibreOffice-Gemeinschaft.

Eine PDF-Version dieses Buches und aller Updates zu diesem Buch kann frei unter http://de.libreoffice.org/get-help/documentation heruntergeladen werden.

Über LibreOffice:
LibreOffice ist eine leistungsfähige Office-Suite, für verbreitete Betriebssysteme wie Windows, GNU/Linux 32-/64-Bit und Apple Mac OS X geeignet. Es bietet sechs Anwendungen für die Erstellung von Dokumenten und zur Datenverarbeitung:
Writer, Calc, Impress, Draw, Base und Math.

LibreOffice entsteht aus der kreativen Zusammenarbeit von Entwicklern und der Gemeinschaft der Stiftung "The Document Foundation". Die Stiftung hat ihren Sitz in Deutschland.

LibreOffice kann unter der Adresse http://de.libreoffice.org/download kostenlos heruntergeladen werden.

7.6

LibreOffice ist ein eingetragenes Markenzeichen der The Document Foundation.
Weitere Informationen finden Sie unter https://de.libreoffice.org