LibreOffice
The Document Foundation

Base-Beispiele

Datenbanklösungen für einzelne Problemstellungen

7.6

Copyright

Mitwirkende/Autoren

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

Rückmeldung (Feedback)

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

Vorsicht

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

Datum der Veröffentlichung und Softwareversion

Veröffentlicht am 01.08.2023. Basierend auf der Version LibreOffice 7.6.

Inhaltsverzeichnis

Vorwort 13

Zeitmessung 15

Einführung 16

Tabellen 16

Start 16

Zeitspeicher 17

Abfragen 18

Abfrage_Start 18

Abfrage_Unterwegs 19

Abfrage_Ziel 19

Formulare 20

Einzelmessung_System 20

Einzelmessung_Zeit 21

Meldung_Liste_Teilnehmer 22

Meldung_Zeit_Teilnehmer 23

Nur_Start 24

Nur_Ziel 25

Start_Ziel_Platz 27

Start_Ziel_Gruppe 28

Berichte 29

Ergebnisliste 29

Ergebnisliste_Zeit 30

Urkunden 31

Makros 32

Starten und Stoppen der Zeit 33

Zeitmessung_Start_System 33

Zeitmessung_Ende_System 33

Zeitmessung_Start 34

Zeitmessung_Ende 35

Sekunden_2_SQLTime 36

Formulare mit Zeitmessung verknüpfen 37

Zeitmessung_Start_Einzel 37

Zeitmessung_Ende_Einzel 38

Zeitmessung_Start_Gruppe 38

Zeitmessung_Ende_Gruppe 38

Zeitmessung_Start_Ziel 39

Zeitmessung_Ende_Listenfeld 39

Aktuelle Standardwerte für Datum und Zeit 40

Einführung 41

Tabellen 41

Datum_Aenderdatum 41

Zeitstempel_Aenderstempel 42

Ansichten 43

Abfragen 43

DatumZeitvorgabe 43

Zeitdifferenzen 44

Zeitdifferenzen_Formularvorlage 45

Formulare 46

Defaultdatum_Subform_neue_Datensaetze 46

Defaultdatum_Zeit_Subform_neue_Datensaetze 48

Defaultdatum_Aenderung 48

Defaultdatum_Makro_SQL 49

Defaultdatum_Zeit_Makro_SQL 49

Defaultdatum_Makro....50
Defaultdatum_Zeit_Makro....51
Defaultdatum_Makro_Standarddatum....51
Defaultdatum_Makro_Standarddatum_verlegt....52
Defaultdatum_Makro_Standardzeitstempel....52
Defaultdatum_Zeitdifferenz_Makro....53
Berichte....53
Bericht_Zeitdifferenzen....54
Bericht_Zeitdifferenzen_Formularvorlage....56
Makros....56
Update....56
Datum_aktuell....57
Datum_Zeit_aktuell....58
Standarddatum....59
Standarddatum_verlegt....59
StandardZeitstempel....60
UpdateTimestamp....60
Dateien einbinden....62
Einführung....63
Tabellen....64
Dateien....64
Dateien_intern....64
Ansichten....65
Abfragen....65
Formulare....66
Pfadeingabe....66
Pfadeingabe_Tabellenkontrollfeld....67
Dateiaufnahme....68
Dateiaufnahme_Name....68
Berichte....69
Makros....71
Betrachten....71
DateiAuslesen....71
DateiAuslesen_mitName....72
DateiName....73
Serienbrief direkt....74
Einführung....75
Tabellen....75
Anschrift....75
Rechnung....76
Rechnungsinhalt....76
Verkauf....77
Waren....77
Abfragen....78
Adresseingabe....78
Etiketten....79
Rechnung_einzeilig....79
Rechnung_nicht_gedruckt....80
Rechnung_Textfelder....80
Formulare....81
Anschrift....82
Anschrift_Textfelder....84
Etiketten_Serienbrief_im_Formular....85
Rechnung....89
Rechnung_Textfelder....91

Rechnung_Textfelder_Uebertrag....94
Rechnung_Textfelder_Uebertrag_Seriendruck....96
Waren....97
Berichte....97
Makros....97
Rechnung....97
Rechnungsinhalt_zusammenstellen....98
Serienbrief....99
Output_MailMerge....99
Textfelder_Fuellen....100
Textfelder_Fuellen_ID....101
Tabellen_fuellen....102
Rechnungsinhalt_zusammenstellen_Tabelle....103
Tabellen_fuellen_Uebertrag....104
Rechnungsinhalt_zusammenstellen_Tabelle_Uebertrag....106
Rechnungsinhalt_zusammenstellen_Tabelle_Uebertrag_Direktdruck....106
PDF_Export....107
Direktdruck_Start....108
Terminübersicht Ferienhäuser....110
Einführung....111
Tabellen....111
Mieter....111
Haus....112
Reservierung....112
Filter....113
NrBis31....113
Ansichten....114
Ansicht_Datum....114
Ansicht_Datum_lfdNr....115
Abfragen....115
Belegung....115
Bericht....117
Formulare....118
Berichte....121
Makros....121
Startdatum....121
Enddatum....122
BuchungsAenderung....124
NeueBuchung....124
DatumsAenderung....125
Mailaufruf....127
Einführung....128
Tabellen....128
Kontakte....128
Mails....129
Mail_CC....129
Anhaenge....130
Ansichten....131
Abfragen....131
Formulare....131
Kontakte_Mail_Web....131
Mail_mit_Anhang....133
Berichte....134
Makros....134
Mauszeiger....134

FormSave....134
Website_Aufruf....135
Mail_Aufruf....136
Attachment....138
Mail_Attachment....139
Suchen und Filtern....142
Einführung....143
Tabellen....143
Filter....144
Kategorien....145
Medien....145
Medienart....146
Ort....146
rel_Medien_Verfasser....146
Verfasser....147
Ansichten....147
Ansicht_Suche....147
Ansicht_Filter_Start....148
Ansicht_Standardfilter....148
Standardfilter_Firebird....149
Abfragen....149
Filter....149
Filter_Form....149
Filter_Form_Subform....150
Filter_Form_Subform_3Filter....151
Filter_Ansicht....151
Filter_Form_Nullwerte....152
Filter_Leerfeld....152
Suche....152
Suche_Form....152
Suche_Form_Subform....153
Parametersuche_Medien....154
Filter und Suche kombiniert....155
Suche_Filter_Form_Subform_3Filter....155
Filter_Such_Ansicht....156
Listenfeldabfragen....157
Listenfeld_Kategorie_ohne_Eintrag....157
Listenfeld_Kategorie_bedingt....158
Listenfeld_Kategorie_bedingt_Datensaetze....159
Listenfeld_Kategorie_bedingt_Suche_Filter....159
Listenfeld_Kategorie_bedingt_Suche_Filter_Datensaetze....160
Listenfeld_Medienart_bedingt....160
Listenfeld_Medienart_bedingt_Datensaetze....160
Listenfeld_Medienart_bedingt_Suche_Filter....161
Listenfeld_Medienart_bedingt_Suche_Filter_Datensaetze....161
Listenfeld_Verfasser_bedingt....161
Listenfeld_Verfasser_bedingt_Datensaetze_langsam....162
Listenfeld_Verfasser_bedingt_Datensaetze_optimiert....162
Listenfeld_Verfasser_bedingt_Suche_Filter....163
Listenfeld_Verfasser_bedingt_Suche_Filter_Datensaetze....163
Formulare....164
Filter_ohne_Makros....164
Filter_Formular....164
Filter_Formular_Nullwerte....166
Filter_Formular_Subformular....166
Filter_Formular_Subformular_3Filter....167
Standardfilter_Formular_Subformular....168
Standardfilter_Formular_Subformular_Datensatzkorrektur....169

Filter_mit_Makros....169
Filter_Formular....170
Filter_Formular_direkt_Nullwerte....171
Filter_Formular_Listfeldbegrenzung....172
Filter_Formular_Subformular....173
Filter_Formular_Subformular_bedingende_Filter....174
Filter_Formular_Subformular_bedingende_Filter_allround....175
Filter_Formular_Subformular_bedingende_Filter_Datensaetze....175
Suche_ohne_Makros....176
Suche_Formular....176
Suche_Formular_Subformular....177
Parameter_Suche_Formular_Subformular....178
Suche_mit_Makros....178
Suche_Formular....179
Suche_Formular_Subformular....180
Suche_Filter_Formular_Subformular_ohne_Makros....181
Suche_Filter_hierarchisch_mit_Makros....182
Suche_Filter_hierarchisch_allround_Datensaetze_mit_Makros....183
Berichte....184
Makros....184
Filter....184
Filter_Formular_direkt....185
Filter_bedingt....186
Filter_bedingt_allround....186
Filter_Suche....187
Filter_Suche_hierarchisch....188
Filter_Suche_hierarchisch_Start....189
Filter_Suche_hierarchisch_allround....190
ListFilter....191
Haushaltsbuch....193
Einführung....194
Tabellen....194
Kasse....194
Adressat....195
Konto....196
Kategorie....196
rel_Kasse_Kategorie....197
Filter....198
Ansichten....198
Ansicht_Kasse_Kategorie....198
Ansicht_Kasse_mit_Umbuchungen....200
Ansicht_Filter....201
Ansicht_Kategorie_Diagramm....202
Ansicht_Konto_Diagramm....202
Ansicht_Bericht_Kategorie....203
Abfragen....204
Kategorieverlauf....204
Kasse_Saldo_Konto_Kategorie....204
Kasse_Saldo_Konto_Kategorie_Umbuch....205
Kontoverlauf....206
Saldo_Kategorie....207
Saldo_Kategorie_ungesplittet....208
Saldo_Konto....209
Formulare....210
Formulare ohne Makros....210
Kasse_oM....210
Konto_oM....210

Konto_Salden_oM....211
Konto_Salden_separat_oM....212
Formulare mit einfachem Aktualisierungsmakro....213
Konto_Salden_komplett_eM....213
Konto_Salden_komplett_Umbuchung_eM....214
Formulare mit mehrfachem Makroeinsatz....215
Kasse....215
Kasse_Umbuchung....216
Konto....217
Konto_Salden....218
Konto_Salden_komplett....219
Konto_Salden_komplett_Umbuchung....220
Buchung_Umbuchung_Salden....221
Hsqldb:....223
Firebird:....223
Kategorie_Konto_Auswertung....224
Berichte....225
Makros....228
Aktualisieren_einfach....228
Aktualisieren....228
ZeileZwischenspeichern....229
SplitRest....230
periodische_Buchung....232
Adressat_neu....234
Monat_aktuell....236
Formular_starten....236
Aktuelles_Buchungsdatum....236
Kontofilter_Formstart....237
Kontofilter_Feldstart....238
Kontoliste_filtern....238
ChangeData....239
Termine....242
Einführung....243
Tabellen....244
Tabellen zur Dateneingabe über Formulare oder direkt....244
Termine....244
Terminart....245
Zielgruppe....246
Ferien....246
Filtertabelle....247
Durch Makros gefüllt Tabellen....248
tmp_Halbjahreskalender....248
tmp_Gruppe_multi....249
tmp_Kalender....249
tmp_Monat....249
tmp_Terminuebersicht....250
tmp_Wochenkalender....250
tmp_Wochenkalender_halb....251
Ansichten....251
Ansicht_Gruppe_multi....251
Ansicht_Kalender_Export....253
Ansicht_Kalender_ohne_Ferien_Export....255
Ansicht_Termine_Monat_Jahre....256
Ansicht_Wochen_einzeln....256
Ansicht_Wochentage....257
Abfragen....258
Liste_Gruppe_Multi....258

Termine_Gruppe_multi....258
Terminuebersicht....259
Wochenkalender_Mo_Fr....259
Formulare....260
Terminart_Termin....260
Termine_Kalenderübersicht....261
Termine_Monat_Terminart....263
Termine_Tabelleneingabe....264
Berichte....265
Halbjahreskalender_A4_quer....265
Liste_Gruppe_multi....266
Liste_Gruppe_multi_1HJ....266
Liste_Gruppe_multi_2HJ....267
Monatskalender_Mo_Fr....267
Monatskalender_Mo_So....268
Wochenkalender_halb....268
Wochentage_Mensaabo....270
Zeitplan....270
Makros....270
AA_Variablen....270
Datenbankstart....271
OfficeVersion....271
Datumswert....272
Listenfeld_Text....272
ID_Ermittlung....273
Abfragen....273
Kalender....273
Monatskalender....276
Wochenkalender_halb....279
Halbjahreskalender....281
Zeitplan....283
GruppeMultiUebersicht....286
Monat....289
Monatsansicht....290
Date_2_SQLDate....291
Date_2_Datum....291
SQLDate_2_Datum....292
Date_2_Monat....292
Datum_Ostersonntag....293
FeiertageArray....294
BerichtZeilenhoeheAuto....294
Bericht_Start....295
Backup....298
Datenbankbackup....298
Backup_sofort....299
Eingabe....299
DatenSpeichern....300
DatenAktualisieren_Termine....300
DatenAktualisieren_Terminort_Termine....300
LetzterDatensatz....301
LetzterDatensatz_Termine_Kalender....301
Anfangswert_Datum....302
Monatsansicht_aktualisieren....302
Monat_aktualisieren....304
Termindaten_uebertragen....304
GruppeMultiSpeichern....306
DialogStart....306
DialogEnde....307
Export....307

Ical_Export....307
Export_Start....309
Import....310
Ical_Import....310
Ical_Timestamp....313
Import_Start....313
LetzterSonntag....314
Filter....315
Wartung....315
Tabellenindex_runter....315
Tabellenstart....316
Dialog Datumseingabe....316
Vereinsverwaltung....318
Einführung....319
Tabellen....319
Zentrale Mitgliedstabelle....320
Adresse....323
tbl_Adresse....323
tbl_Ort....324
Konto....326
tbl_Konto....326
tbl_Bank....327
Vorstand....328
tbl_Mitglied_VorstandPosten....328
tbl_VorstandPosten....329
Schlüssel....330
tbl_MitgliedSchluessel....331
tbl_Schluessel....332
tbl_SchluesselAnzahl....332
Abteilung....333
tbl_Mitglied_Abteilung....334
tbl_Abteilung....334
Beitrag....335
tbl_Gruppe_Beitrag....336
tbl_Beitrag....336
tbl_Beitrag_Staffel....337
Boot....338
tbl_Boot....339
tbl_BootArt....341
tbl_BootStaender....341
tbl_BootHalle....342
Tabellen ohne Relationen....343
tbl_Altergruppen....343
tbl_Einstellungen....343
tbl_Filter....344
Ansichten....346
viw_Abteilungen_Mitglied....346
viw_Adresse_Num_Namen....347
viw_Anschrift....348
viw_Beitrag....350
viw_Beitrag_Gruppe_kalkuliert....355
viw_Boote_Mitglied....357
viw_Filter_Mitglieder_komplett_AusEin....358
viw_Gruppe....360
viw_Mitglieder_komplett....362
viw_Reports....364
viw_Schluessel_Mitglied....366
viw_ServVar....367

viw_Statistik_Alter_Detail ... 368
Abfragen ... 369
qry_Adresse_Gruppe ... 370
qry_Anschriften_2spaltig ... 370
qry_Anschriften_3spaltig ... 371
qry_Beitraege ... 373
qry_Bootsstaender ... 374
qry_Filter ... 375
qry_Filter_Austritt ... 376
qry_Filter_Gruppe ... 378
qry_Filter_Mitglied ... 378
qry_Filter_Vorstand ... 379
qry_Forms ... 380
qry_Jubilaeum ... 380
qry_Mitglied_Konto ... 382
qry_Mitgliedsdauer ... 383
qry_Schluessel_Uebersicht ... 384
qry_Vorstand_aktuell ... 385
qry_Vorstand_aktuell_lt_Satzung ... 386
Formulare ... 387
frm_Beitrag ... 387
frm_Berichte ... 388
frm_Boot ... 390
frm_Einstellungen ... 392
frm_Mitglied ... 394
frm_Schluessel ... 399
frm_Vorstand ... 400
Berichte ... 401
rpt_Anschriften_Tabelle ... 401
rpt_Austritt ... 402
rpt_Beitraege ... 404
rpt_Beitraege_Konto ... 406
rpt_Bootstaender_Belegung ... 407
rpt_Etiketten_2_Spalten ... 408
rpt_Etiketten_3_Spalten ... 408
rpt_Jubilaeum_Ehrenurkunde ... 409
rpt_Mitglieder_AusEin ... 411
rpt_Mitglieder_AusEin_Konto ... 413
rpt_Mitglieder_komplett ... 415
rpt_Mitglieder_komplett_Konto ... 417
rpt_Mitgliedsdauer ... 418
rpt_Schluesseluebersicht ... 419
rpt_Statistik ... 420
rpt_Vorstand_aktuell ... 421
rpt_Vorstand_aktuell_lt_Satzung ... 422
Makros ... 423
Backup ... 423
DatabaseBackup ... 423
DatabaseStart ... 424
FilterIDSet ... 424
Preferences ... 425
Design ... 426
PreferencesHide ... 426
BoolChange ... 427
BoolChangeKey ... 427
BoolStart ... 428
AfterRecordChange ... 428
Group ... 429

Druck....431
ReportStart....431
Eingabekontrolle....432
NewBoatAllowed....432
SelectBoat....433
SelectBoatRack....434
SelectSQL....434
SelectGroup....435
BoardFilterSelect....436
SelectBoard....437
SelectKey....438
IBAN_Test....438
IBAN_Length_Change....439
Filter....440
Filtern....440
Navigation....441
FormStart....441
Navigation....441
Speichern....443
ListboxContentChange....443
NewGroup....443
ShowTableControl....444
SubscriptionChanged....444
SubscriptionSave....444
Wartung....445
Tabellenindex_runter....445
Tabellenstart....446
Tabellenbereinigung....446
Tabellenbereinigung_Ort....447
Tabellenbereinigung_Bank....447
ViewsErstellen....447

Vorwort

Diese Sammlung verschiedener Datenbanken wird laufend erweitert. Sie ist aufgrund von Nachfragen in Foren und Mailinglisten entstanden.

Nahezu alle Datenbanken existieren sowohl in einer Fassung für die interne HSQLDB als auch für die interne FIREBIRD Datenbank. Die internen HSQLDB Datenbanken wurden vom Code her so weit angepasst, dass bei einer Migration nach FIREBIRD möglichst wenig Probleme auftreten. So taucht z.B. die Funktion **IFNULL** nicht mehr auf, weil sowohl in der HSQLDB als auch in FIREBIRD die Funktion **COALESCE** genau zum gleichen Ergebnis führt, **IFNULL** aber für FIREBIRD unbekannt ist. Eine Liste der Funktionen, die bei der Migration von der einen zu der anderen Datenbank berücksichtigt werden müssen, ist im Handbuch aufgeführt.

Gerade im Umgang mit Datums- und Zeitwerten gäbe es in FIREBIRD im Verhältnis zur alten HSQLDB, die in Base integriert ist, große Vorteile. Gegebenenfalls sind dann Anmerkungen zu alternativen Lösungen in der Beschreibung vorhanden.

Zur besseren Übersicht ist in dieser Beschreibung die Datenbankbezeichnung immer grün gekennzeichnet.

Folgende Kennzeichnungen sind in diesem Text zu finden:

HSQLDB oder FIREBIRD	Datenbanknamen werden mit Kapitälchen und in grüner Farbe geschrieben
~~HSQLDB~~ oder ~~FIREBIRD~~	Die Funktion ist für die entsprechende Datenbank nicht verfügbar. Eine einfache Codeänderung reicht nicht aus, um die Funktion zu ermöglichen.
"Tabellenname" und "Feldname"	Wie beim SQL-Code werden Tabellennamen und Feldnamen mit doppelten Anführungszeichen maskiert
'Robert' oder '1'	Werte aus Abfragen werden mit einfache Anführungszeichen maskiert.
«Zeitmessung»	Andere Anführungszeichen dienen der allgemeinen Hervorhebung

Zeitmessung

Einführung

Zeitmessungen könnten z. B. für die Durchführung von Sportveranstaltungen genutzt werden. Gedacht ist hier nicht an große Veranstaltungen, wo so etwas sowieso elektronisch erfolgt. Vielmehr könnte es um Veranstaltungen im Schulbereich gehen.

Der PC ersetzt hier die Stoppuhr und das Niederschreiben der Ergebnisse. Er kann außerdem auch die Auswertung erstellen sowie Urkunden drucken. Im Vordergrund soll hier die Funktion der Stoppuhr stehen.

Um mit der Datenbank «Zeitmessung»[1] auch Zeitmessungen durchführen zu können werden in geringem Umfang Makrokenntnisse gebraucht.

Tabellen

Die Datenbank besteht im sparsamsten Aufbau aus zwei Tabellen.

Start

Datenziel	
Formular:	*Einzelmessung_System, Einzelmessung_Zeit, Meldung_Liste_Teilnehmer, Meldung_Zeit_Teilnehmer, Nur_Start, Nur_Ziel, Start_Ziel_Platz*
Makro:	*Zeitmessung_Ende_System, Zeitmessung_Ende*
Abfrage:	*Abfrage_Unterwegs, Abfrage_Ziel*
Ansicht, Bericht:	keine direkt

Feldname	***Feldtyp***	***Beschreibung***
ID	Integer	Primärschlüssel der Tabelle. Der Primärschlüssel muss eindeutig sein. Das Feld ist als Auto-Wert-Feld gesetzt.
Startzeit	Datum/Zeit	Fremdschlüssel für "Zeitspeicher"."Startzeit". Hier wird Tag und Uhrzeit des vorgesehenen Startes festgehalten. Personen, die zusammen starten, haben die gleiche Startzeit. Eingabe erforderlich: Nein
Name	Text	Der Name der Person, die startet. Bei Schulwettkämpfen wären hier Zusätze wie Klasse, Schule oder gegebenenfalls auch Geburtsdaten notwendig. Eingabe erforderlich: Ja
Dauer HSQLDB	Integer	Die Dauer wird in Millisekunden gemessen. Es handelt sich dabei um einen Zahlenwert ohne Nachkommastellen, keinen Wert wie bei einer Uhrzeit. Eingabe erforderlich: Nein
Zeit HSQLDB	Datum/Zeit	Die Zeitdauer wird hier als Zeit im Format Stunden:Minuten:Sekunden,Hundertstelsekunden aus dem Formular «Einzelmessung_Zeitfeld» übernommen. Der Datumsteil wird durch ein Makro hinzugefügt. Eingabe erforderlich: Nein
Zeit FIREBIRD	Zeit	Die Zeitdauer wird hier mit Hundertstelsekunden abgespeichert. Dies ist in der ~~HSQLDB~~ nicht möglich. Eingabe erforderlich: Nein

1 Beispieldatenbank Beispiel_Zeitmessung.odb

Zeitspeicher

Datenziel	
Formular:	*Einzelmessung_System, Einzelmessung_Zeit, Meldung_Liste_Teilnehmer, Meldung_Zeit_Teilnehmer, Nur_Ziel, Start_Ziel_Platz*
Makro:	*Zeitmessung_Start_System, Zeitmessung_Ende_System, Zeitmessung_Start, Zeitmessung_Ende*
Abfrage:	*Abfrage_Start, Abfrage_Unterwegs, Abfrage_Ziel*
Ansicht, Bericht:	keine direkt

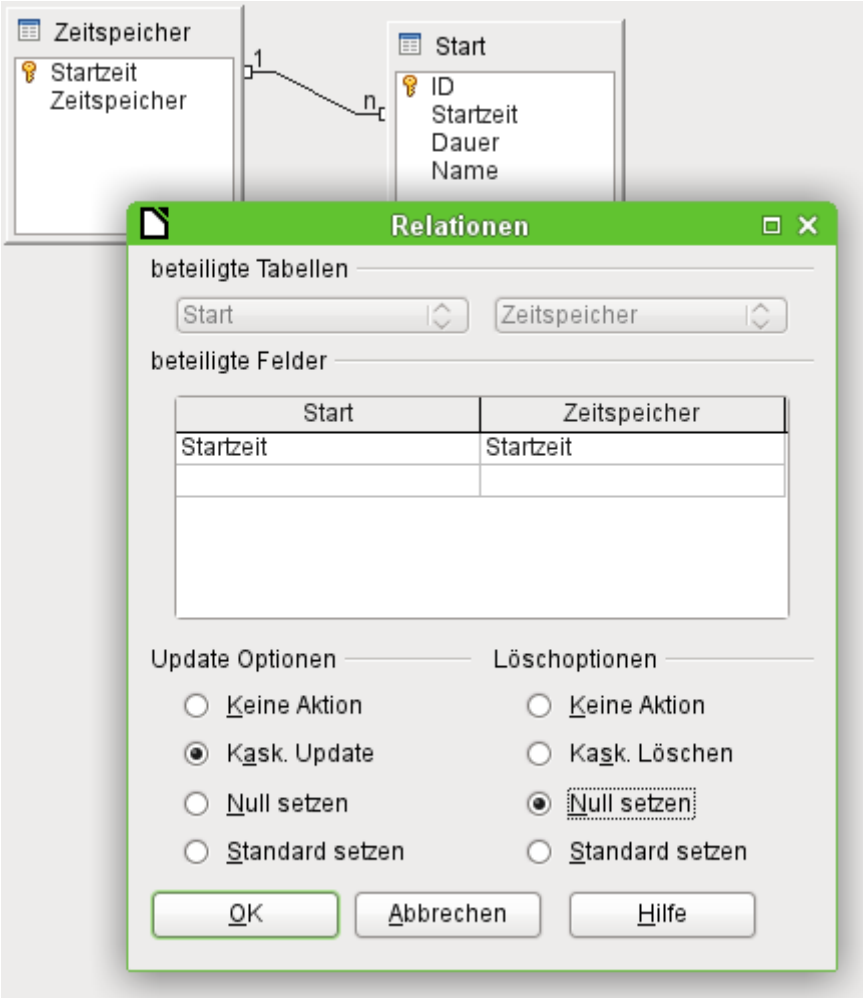
Feldname	***Feldtyp***	***Beschreibung***
Startzeit	Datum/Zeit	Die Startzeit ist Primärschlüssel dieser Tabelle. Sie ist gleichzeitig Fremdschlüssel der Tabelle "Start". Damit wird für alle Personen, die die gleiche Startzeit haben, auch bei der Durchführung der einheitliche Start festgelegt.
Zeitspeicher HSQLDB	Integer	Hier wird ein interner Basic-Wert für die tatsächliche Startzeit abgespeichert. Der Wert wird erst dann eingefügt, wenn der Button Start betätigt wurde. Eingabe erforderlich: Nein
Timestamp HSQLDB	Datum/Zeit	Hier wird die tatsächliche Startzeit zusammen mit Datum abgespeichert. Wird statt "Zeitspeicher" genutzt, wenn direkt eine Zeit gemessen wird. Eingabe erforderlich: Nein
Zeitspeicher FIREBIRD	Zeit	Direkte Eingabe der Startzeit. In der ~~HSQLDB~~ nicht möglich. Eingabe erforderlich: Nein

Die Datum/Zeit-Felder in der HSQLDB können, sofern sie durch die GUI erstellt wurden, keine Bruchteile einer Sekunde darstellen. Sie müssen erst mit dem Befehl

```
001 ALTER TABLE "Start" ALTER COLUMN "Zeit" TIMESTAMP(6)
001 ALTER TABLE "Zeitspeicher" ALTER COLUMN "Timestamp" TIMESTAMP(6)
```

für die Aufnahme von 1/1000 Sekunden vorbereitet werden.

Beide Tabellen werden miteinander in den Beziehungen verbunden. Es muss also eine Startzeit festgelegt werden, bevor eine Person in die Startliste übertragen werden kann.



Die Beziehung zwischen den Tabellen wird mit einem rechten Mausklick auf die Verbindungslinie genauer definiert. Wird im "Zeitspeicher" eine "Startzeit" geändert, so werden alle "Startzeit"-Einträge in der Tabelle "Start" entsprechend angepasst. Dies ist über die **Update Optionen → Kask. Update** gewährleistet. Wird eine "Startzeit" aus der Tabelle "Zeitspeicher" gelöscht, so sollen Einträge, die sich aus der Tabelle "Start" genau auf die gelöschte "Startzeit" beziehen, geleert werden. Dies wird über die **Löschoptionen → Null setzen** erreicht.

Die tatsächlichen Startzeiten müssen nicht mit der eingetragenen Startzeit übereinstimmen. Auch wenn eine eingetragene Startzeit z. B. 10:00 Uhr ist, kann tatsächlich um 15:13 Uhr gestartet und trotzdem ein genaues Messergebnis erwartet werden.

Abfragen

Abfrage_Start

Datenquelle
Tabelle: *Zeitspeicher*

Datenziel
Formular: *Nur_Start, Start_Ziel_Platz, Start_Ziel_Gruppe*

HSQLDB

```
001 SELECT * FROM "Zeitspeicher" WHERE "Timestamp" IS NULL
```

FIREBIRD

```
001 SELECT * FROM "Zeitspeicher" WHERE "Zeitspeicher" IS NULL
```

Starten darf natürlich nur die Gruppe, die nicht bereits unterwegs ist.

Mit dem Start wird ein Wert in das Feld "Timestamp" (HSQLDB) bzw. "Zeitspeicher" (FIREBIRD) geschrieben. Nur die Startzeit kann noch genutzt werden, bei der dieses Feld leer ist.

Die HSQLDB hat daneben noch das Feld "Zeitspeicher", das aber nur einen Integer-Wert für die Zeitmessung über Makro mit Systemticks speichert.

Abfrage_Unterwegs

Datenquelle
Tabelle: *Zeitspeicher, Start*

Datenziel
Formular: *Start_Ziel_Platz, Start_Ziel_Gruppe*

```
001 SELECT "Start".*
002 FROM "Start", "Zeitspeicher"
003 WHERE "Start"."Startzeit" = "Zeitspeicher"."Startzeit"
004 AND "Start"."Zeit" IS NULL
005 AND "Zeitspeicher"."Timestamp" IS NOT NULL
```

Diese Abfrage unterscheidet sich bei FIREBIRD nur dadurch, dass statt des "Timestamp"-Feldes in Zeile 5 das "Zeitspeicher"-Feld angesprochen wird.

Diese Abfrage listet alle die Personen auf, die bereits gestartet (Zeile 5), aber noch unterwegs sind (Zeile 4). Ihre Zielzeit wurde noch nicht gemessen.

Abfrage_Ziel

Datenquelle
Tabelle: *Zeitspeicher, Start*

Datenziel
Formular: *Nur_Ziel, Start_Ziel_Platz, Start_Ziel_Gruppe*
Bericht: *Ergebnisliste, Ergebnisliste_Zeit, Urkunden*

Variante für HSQLDB mit dem Feld "Dauer" und der Zusammenstellung eines Zeitformates:

```
001 SELECT "a".*,
002 ( SELECT "Timestamp" FROM "Zeitspeicher" WHERE "Startzeit" =
    "a"."Startzeit" ) AS "Zeitspeicher",
003 ( SELECT COUNT( "ID" ) FROM "Start" WHERE "Zeit" <= "a"."Zeit" AND
    "Startzeit" = "a"."Startzeit" ) AS "Platz"
004 FROM "Start" AS "a"
005 WHERE ( SELECT "Timestamp" FROM "Zeitspeicher" WHERE "Startzeit" =
    "a"."Startzeit" ) IS NOT NULL
```

Änderung für FIREBIRD:

```
001 SELECT "a".*,
002 ( SELECT "Zeitspeicher" FROM "Zeitspeicher" WHERE "Startzeit" =
    "a"."Startzeit" ) AS "Zeitspeicher",
```

```
003 ( SELECT COUNT( "ID" ) FROM "Start" WHERE "Zeit" <= "a"."Zeit" AND
    "Startzeit" = "a"."Startzeit" ) AS "Platz",
004 CAST( "a"."Zeit" AS TIMESTAMP ) AS "RBZeit
005 FROM "Start" AS "a"
006 WHERE ( SELECT "Zeitspeicher" FROM "Zeitspeicher" WHERE "Startzeit" =
    "a"."Startzeit" ) IS NOT NULL
```

Hier werden alle Felder der Tabelle "Start" angezeigt (Zeile 1). Dabei wird der Tabelle "Start" ein Alias "a" zugewiesen (Zeile 4 bzw. 5), damit von den Unterabfragen auf die äußere Abfrage zugegriffen werden kann. Angezeigt werden sollen nur die Datensätze, die einen Eintrag in der Tabelle "Zeitspeicher" im Feld "Timestamp" (HSQLDB) bzw. "Zeitspeicher" (FIREBIRD) haben (Zeile 2 und 5 bzw. 6). Dieses Feld ist über eine korrelierende Unterabfrage eingebunden, weil für die Ermittlung des Platzes sowieso eine korrelierende Unterabfrage erstellt werden musste und die Abfrage sonst auch nicht mehr über ein Formular beschrieben werden könnte.

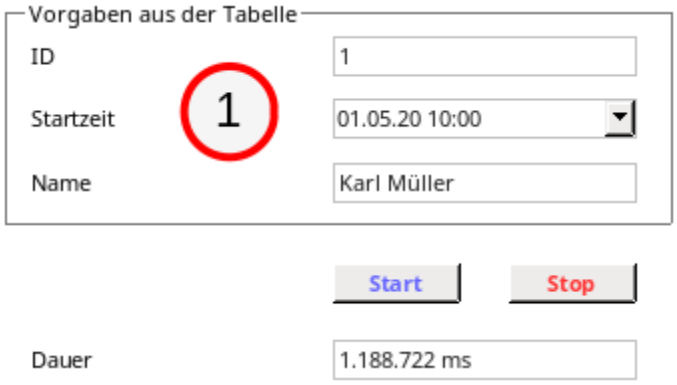
Der Platz wird so ermittelt, dass alle die Datensätze gezählt werden, die einen kleineren Eintrag bei dem Feld "Zeit" haben und gleichzeitig die gleiche "Startzeit" aufweisen können wie der aktuelle Datensatz (Zeile 3). Die Beziehung zum aktuellen Datensatz stellt hier der Alias "a" her.

Für die Darstellung von Zeiten mit Nachkommastellen im Report-Builder ist bei FIREBIRD die Zeile 4 notwendig. Hier wird nur für diesen Zweck der Datentyp von TIME zu TIMESTAMP konvertiert. Der Report-Builder zeigt bis einschließlich Version LO 6.4 leider nur Felder des Typs TIMESTAMP mit Zeiten im Bereich von unter einer Sekunde an. Der Bug ist bisher vermutlich nicht aufgefallen, weil die interne HSQLDB eben Zeitfelder nicht mit kleineren Einheiten als einer Sekunde abspeichert.

Formulare

Die Messung erfolgt innerhalb von Formularen. Ein Formular zeigt nur die Messung eines Wertes. In den folgenden Formularen werden dann auch Mitbewerber angezeigt. Schließlich wird eine komplette Zeitmessung für ein Rennen ausgestellt, bei dem gleichzeitig gestartete Teilnehmer in eine Rangliste eingeordnet werden. Hier zuerst einmal die einfachste Variante eines Formulars mit einfachen Feldern.

Einzelmessung_System



1	Formular (Tabelle: *Start*)
1	Listenfeld Startzeit (Tabelle: *Zeitspeicher*)

Makros in Feldeigenschaften		
1	Schaltfläche Start (Aktion ausführen)	Module1.*Zeitmessung_Start_System*
1	Schaltfläche Stop (Aktion ausführen)	Module1.*Zeitmessung_Ende_System*

Das Formular sieht erst einmal recht karg aus. Die ursprünglich geplante Startzeit ist zusammen mit dem Datum als Zeitstempel abgespeichert. Da für die Datenbank über die oben genannten 2 Tabellen auch andere Funktionen gewährleistet werden sollen, kann hier nur dann eine Startzeit/Startdatum angegeben werden, wenn die entsprechende Zeit bereits in der anderen Tabelle vorhanden ist. Deshalb ist für die Startzeit ein Listenfeld vorgesehen. Das Listenfeld liest die "Startzeit" aus der Tabelle "Zeitspeicher" und trägt genau diesen Wert in die Tabelle "Start" als "Startzeit" ein. Für die Messung ist diese Zeit wegen der Abspeicherung der Startzeit in der zweiten Tabelle zwingend notwendig.

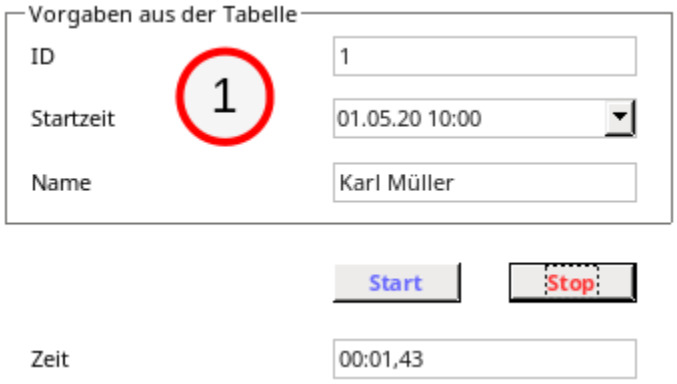
Start startet die Zeitmessung. Mit Stop wird die Zeitmessung beendet und die gemessene Zeit in dem Feld «Dauer» abgespeichert.

Die Zeitmessung erfolgt hier rein über ein Makro, das als Messhilfe die Systemticks des Rechners nutzt. Dies erlaubt eine Angabe im Millisekundenbereich. Der abgespeicherte Wert ist schließlich eine Integer-Ganzzahl, also kein Zeitwert im Sinne der Datenbank.

Diese Formular ist so nur in der HSQLDB realisiert, da in FIREBIRD der Feldtyp TIME bereits die Abspeicherung von 1/1000 Sekunden erlaubt.

Einzelmessung_Zeit

Dieses Formular unterscheidet sich von dem Formular «Einzelmessung_System» äußerlich nur gering. Statt der Angabe einer Zeit in Millisekunden erfolgt hier die Zeitangabe in der Schreibweise **`Minuten:Sekunden,Hundertstelsekunden`**.



1	Formular (Tabelle: *Start*)
1	Listenfeld Startzeit (Tabelle: *Zeitspeicher*)

Makros in Feldeigenschaften		
1	Schaltfläche Start (Aktion ausführen)	Module1.*Zeitmessung_Start_Einzel*
1.1	Schaltfläche Stop (Aktion ausführen)	Module1.*Zeitmessung_Ende_Einzel*

Das Feld "Zeit" in der Tabelle "Start" ist in der HSQLDB als «Datum/Zeit [TIMESTAMP]» - Feld definiert. Die reinen Zeitfelder sind in der HSQLDB nicht in der Lage, Zeiten im Bereich unterhalb einer Sekunde zu speichern. Leider sind aber auch die TIMESTAMP-Felder von der Standardeinstellung in Base nicht dafür vorgesehen. Hier muss entsprechend über SQL nachgeholfen werden. In **Extras → SQL** ist einzugeben:

```
001 ALTER TABLE "Start" ALTER COLUMN "Zeit" TIMESTAMP(6)
```

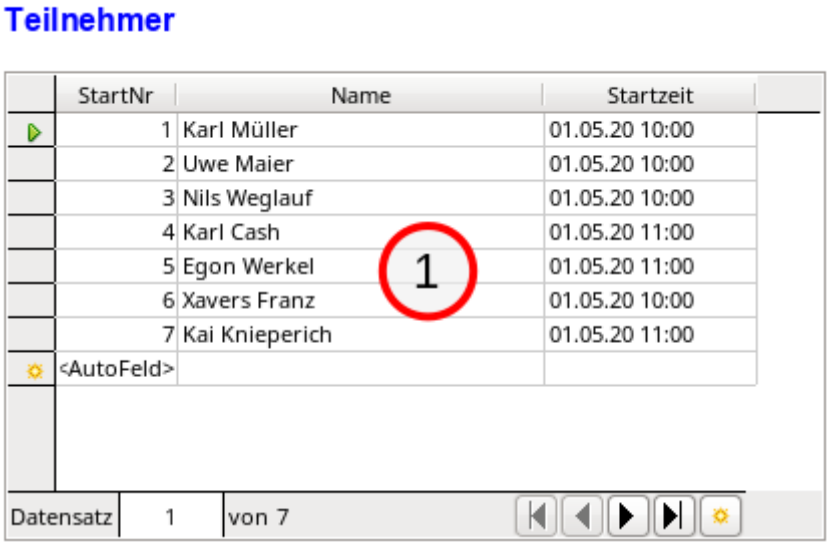
Jetzt ist das Feld in der Lage, auch im Millisekundenbereich Zeiten aufzunehmen.

Die FIREBIRD-Version dieser Datenbank greift dagegen direkt auf ein Feld des Typs TIME zu, das von vornherein Zeiten im 1/1000-Sekunden-Bereich speichern kann.

Das nächste Problem für ein Formular besteht darin, dass so ein Feld sowohl ein Datum als auch eine Zeit abspeichert. Deshalb erstellt Base auch im Formularassistenten oder bei dem Weg über das direkte Hinzufügen von Feldern für ein Datum/Zeit – Feld zwei Eingabemasken. Eine dient zur Eingabe des Datums, eine andere zur Eingabe der Zeit. Diese ist für eine einfache Zeitangabe mit Nachkommastellen nicht brauchbar. Stattdessen muss ein Feld her, dass beides darstellen könnte und außerdem in der Lage ist, auch noch die Hundertstel Sekunden mit aufzunehmen. Deshalb wird das Feld als «Formatiertes Feld» erstellt. Dieses Feld wird hier nur zur Anzeige benutzt und deswegen in **Eigenschaften → Allgemein → Nur Lesen** auf «Ja» gestellt. Zur Eingabe von Werten ist es so auf dem direkten Wege nicht geeignet, da neben der Zeiteingabe eine Datumseingabe erwartet wird, das Feld aber bei der vorliegenden Formatierung diese Datumseingabe nicht mitliefert.

Die Ermittlung der korrekten Zeit muss wieder über ein Makro geleistet werden.

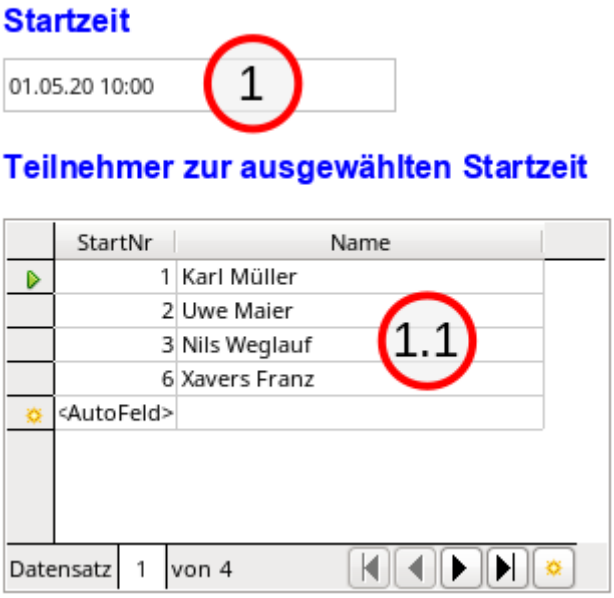
Meldung_Liste_Teilnehmer



1	Teilnehmer (Tabelle: *Start*)
1	Listenfeld Startzeit (Tabelle: *Zeitspeicher*)

In diesem Formular werden die Teilnehmer eingegeben. Die Startnummer ist gleich dem Primärschlüsselfeld. Der Name wird einfach komplett eingetragen. Für Sportevents noch nicht brauchbar – aber das ist schnell durch zusätzlich Felder in der Tabelle zu beheben. Wichtig ist lediglich, dass den startenden Personen eine Startzeit zugewiesen wird. Hier ist das Feld, auf dem Screenshot so nicht erkennbar, als Listenfeld ausgelegt. Es liest direkt die Schlüsselwerte aus der Tabelle "Zeitspeicher". So können in diesem Formular fehlerfrei und schnell die Startzeiten den Personen zugeordnet werden.

Meldung_Zeit_Teilnehmer



1	Formular (Tabelle: *Zeitspeicher*)	**1.1**	Teilnehmer (Tabelle: *Start*)

Fehlt eine Startzeit, so ist das vorhergehenden Formular im Nachteil. Die Startzeit muss vorgegeben sein, damit ein Teilnehmer auch starten kann. Sie kann aber nicht zugewiesen werden, wenn sie im Listenfeld nicht auftaucht.

Hier hilft die einfache Formular – Unterformular – Konstruktion. Im Hauptformular wird die Startzeit eingegeben, im Unterformular dazu tauchen alle die Personen auf, die zu dieser Zeit starten wollen.

Der Formularsprung von Hauptformular zu Unterformular muss hier mit einem kleinen Trick bewerkstelligt werden, da das Hauptformular nur ein Feld hat. Bewegt sich der Nutzer mit dem Tabulator normal durch ein Formular, so geht der Cursor nach dem letzten Eingabefeld zum ersten Eingabefeld des nächsten Datensatzes im Formular, nicht aber zum Unterformular. Um zum Unterformular zu kommen müssen 3 Bedingungen erfüllt sein:

1. Das Hauptformular muss mit allen Feldern im Formularnavigator vor dem Unterformular erscheinen.
2. Das Hauptformular braucht mindestens 2 Felder, die für den Tabulatorsprung sortiert werden können.
3. Die Sortierung der Felder erfolgt über **Aktivierungsreihenfolge → Automatische Sortierung**.

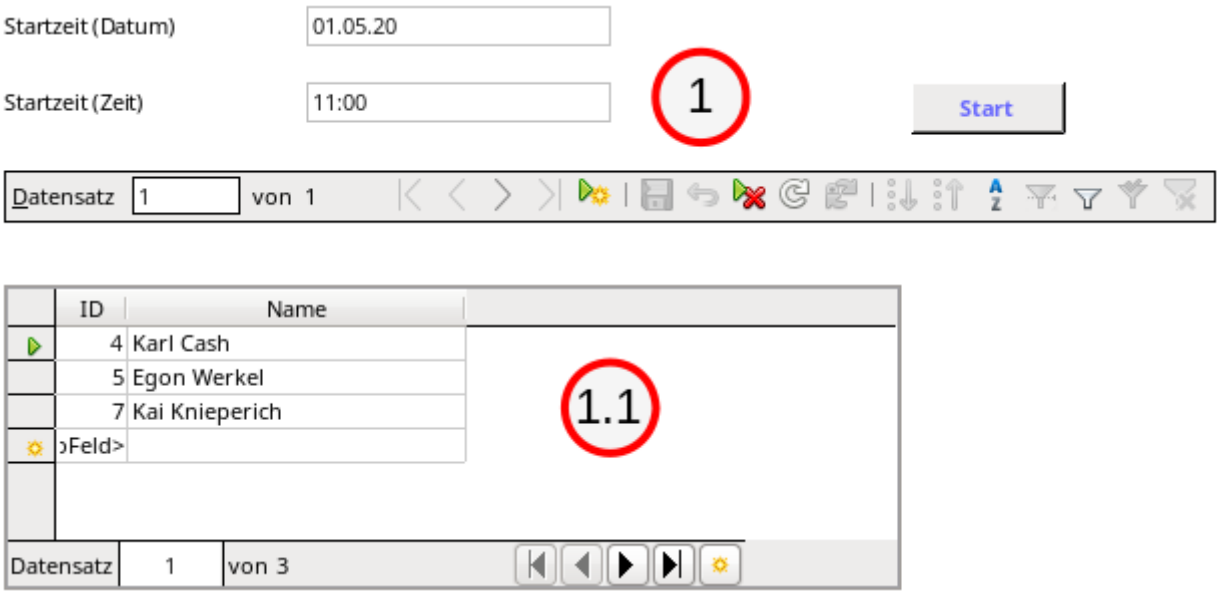
In der Tabelle "Zeitspeicher" befinden sich die Felder "Startzeit" und "Zeitspeicher". Nur das Feld "Startzeit" ist oben im Formular zu sehen. Zum Erstellen des Formulars wurde aber auch das Feld "Zeitspeicher" vorübergehend in das Formular mit aufgenommen. Danach wurde das Unterformular erstellt und die **Automatische Sortierung** aufgerufen. Jetzt ist das Unterformular so eingebunden, dass der Tabulatorsprung vom Hauptformular zum Unterformular geht. Das Feld "Zeitspeicher" kann entfernt werden und der Sprung zum Unterformular funktioniert auch weiter.

Aus dem Unterformular heraus geht es übrigens wieder mit *Strg* + *Tab*. Nur muss dann doch anschließend beim Wechsel der Startzeit einmal die Navigationsleiste aufgesucht werden, damit ein nächster Datensatz im Hauptformular eingegeben werden kann.

Nur_Start

Bei den bisherigen Einzelmessungen kann es ohne weiteres passieren, dass ein Start mehrmals hintereinander eingegeben, also die Zeit nicht korrekt gemessen wird. Dies lässt sich vermeiden, wenn nach dem erfolgten Start ein erneutes Beschreiben des Start-Wertes unmöglich gemacht wird.

Aus dem folgenden Formular verschwinden nach dem Start die Datensätze der Personen, die zur gleichen Startzeit unterwegs sind.



1	Zeitspeicher (Abfrage: *Abfrage_Start*)	**1.1**	Formular (Tabelle: *Start*)

Makros in Feldeigenschaften		
1	Schaltfläche Start (Aktion ausführen)	Module1.*Zeitmessung_Start*

In dem Unterformular werden die Personen angezeigt, die zu der obigen Startzeit starten wollen. Es wäre auch möglich, hier zu der Gruppe noch schnell ein paar Personen neu hinzuzufügen, die sich nicht rechtzeitig gemeldet haben, jetzt aber doch noch an den Start gehen wollen.

In dem Hauptformular wird der Start für alle untenstehenden Personen durch Start ausgelöst. Das Hauptformular beruht auf einer Abfrage, die dafür sorgt, dass hier nur die Gruppen zur Verfügung stehen, die noch keinen Ersteintrag bei "Zeitspeicher"."Timestamp" (HSQLDB) oder "Zeitspeicher"."Zeitspeicher" (FIREBIRD) haben, also noch nicht gestartet sind.

Der Startwert wird dann in die Tabelle "Zeitspeicher" als Datum/Zeit (HSQLDB) oder als Zeit (FIREBIRD) per Makro übertragen.

Diese recht einfache Abfrage hätte auch direkt im Formular «Zeitspeicher» als Filterwert eingegeben werden können:

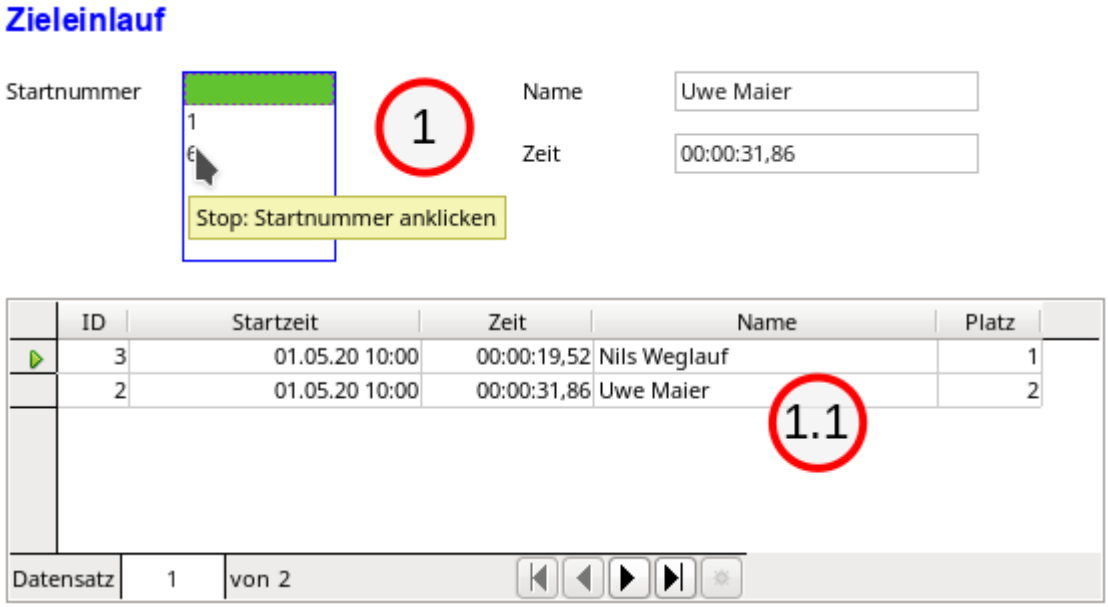
rechte Maustaste über dem Formularnamen "Zeitspeicher" → Eigenschaften → Daten → Filter

In das freie Feld kann eingetragen werden: **`("Zeitspeicher" IS NULL)`**

...nur hat das natürlich den Nachteil, dass dieser Filter auch ausgeschaltet werden kann und prompt Rennen wieder verfügbar wären, die eigentlich schon gestartet wurde.

Nur_Ziel

Dieses Formular muss wesentlich mehr Funktionen erfüllen als die vorhergehenden Formulare. Es soll dazu dienen, nur nach Auswahl der Startnummer den Zieldurchgang zu regeln. Dabei soll es anzeigen, welche Zeit die gerade das Ziel passierte Person benötigt hat. Außerdem soll noch die Einordnung in Form einer Rangliste erfolgen.



1	Ziel (Abfrage: *Abfrage_Ziel*)	**1.1**	Platz (Abfrage: *Abfrage_Ziel*)
1	Listenfeld Startzeit (Tabelle: *Zeitspeicher, Start*)		

Makros in Feldeigenschaften		
1.1	Listenfeld «Startnummern» (Modifiziert)	Module1.*Zeitmessung_Ende_Listenfeld*

Hier ist gerade Uwe Maier als 2. Person ins Ziel gekommen. Die Startnummer ("ID") von Uwe Maier wurde im Listenfeld links oben beim Zieldurchlauf angeklickt. Dadurch wurde ein Makro ausgelöst, der Name zu der Startnummer und die Zeit wurde angezeigt, gleichzeitig die Positionierung in dem Feld der Personen, die zur gleichen Zeit gestartet waren. Das Listenfeld zeigt jetzt nur noch die Startnummer der Personen, die noch unterwegs sind, an.

Das Listenfeld «Startnummer» zeigt den gleichen Wert an, den es auch an die darunterliegende Tabelle weitergibt. Die Datenquelle für das Listenfeld ist auf SQL eingestellt.

```
001 SELECT "Start"."ID"
002    FROM "Start", "Zeitspeicher"
003    WHERE "Start"."Startzeit" = "Zeitspeicher"."Startzeit"
004       AND "Zeitspeicher"."Zeitspeicher" IS NOT NULL
005       AND "Start"."Zeit" IS NULL
```

Diese Abfrage stellt einmal das Feld "Start"."ID" dar. Die Einstellung **Eigenschaften des Listenfeldes → Daten → Gebundenes Feld = 0** bewirkt, dass genau dieses Feld Daten an die darunterliegenden Tabelle überträgt. Bei älteren Versionen von LO muss mit der Einstellung **Gebundenes Feld = 1** und einer doppelten Abfrage des Feldes "Start"."ID" für die Weitergabe des Wertes gesorgt werden.

Die Abfrage stellt zuerst einmal die Beziehung zwischen den Tabellen "Start" und "Zeitspeicher" her (Zeile 3). Dann muss die Bedingung erfüllt sein, dass "Zeitspeicher"."Zeitspeicher" nicht leer ist (Zeile 4). Das bedeutet, dass eben ein Start bereits erfolgt ist. Und schließlich sollen nur die Datensätze erscheinen, bei denen in "Start"."Zeit" bisher kein Eintrag existiert (Zeile 5). Datensätze von Personen, deren Zeit schon gemessen wurde, fallen also auch heraus.

Das Formular "Ziel" hat als Datengrundlage eine Abfrage, und zwar "Abfrage_Ziel". Diese Abfrage wird zusätzlich über **Formular-Eigenschaften → Daten → Filter** eingeschränkt, so dass nur der aktuelle Datensatz angezeigt wird, der dem Eintrag in der Tabelle "Filter" entspricht:

```
001 ("a"."ID" = (SELECT "Start_ID" FROM "Filter" WHERE "ID" = TRUE))
```

Das Unterformular «Platz» greift auf die gleiche Abfrage zu. Hier ist die Filterung dann allerdings anders angelegt. Unter **Formular-Eigenschaften → Daten → Filter** steht:

```
001 ( "a"."Dauer" IS NOT NULL )
```

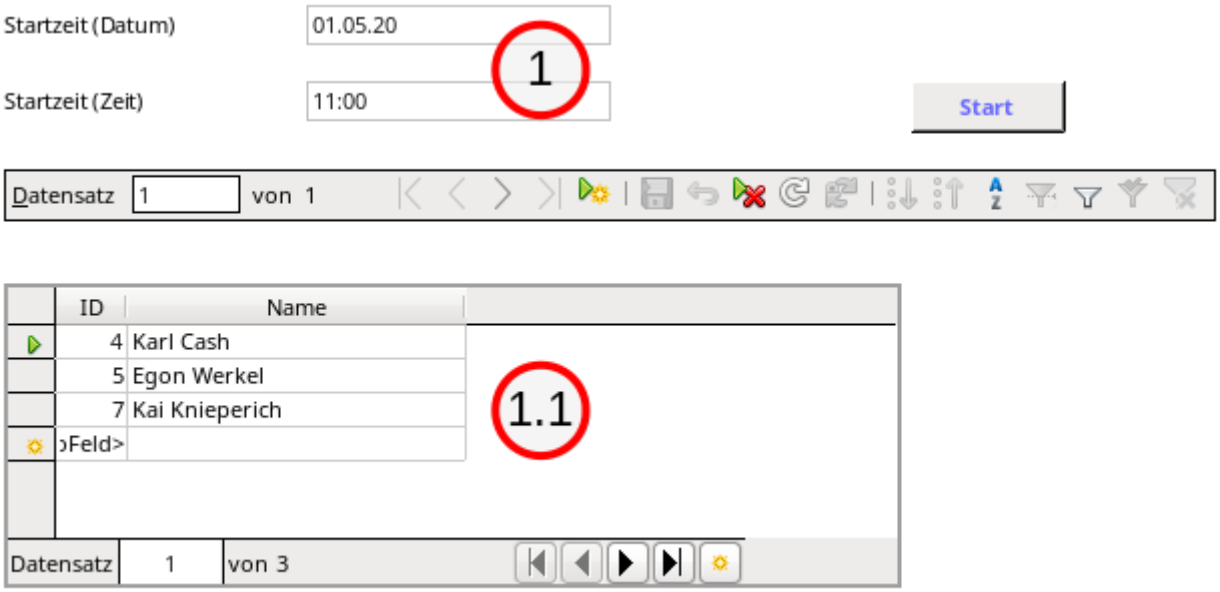
Es werden nur die Werte aus der Abfrage angezeigt, die einen Eintrag im Feld "a"."Dauer" haben. Es werden also nur die Personen angezeigt, die auch tatsächlich mit einer entsprechend vermerkten Zeit das Ziel erreicht haben.

Unter **Formular-Eigenschaften → Daten → Sortierung** steht:

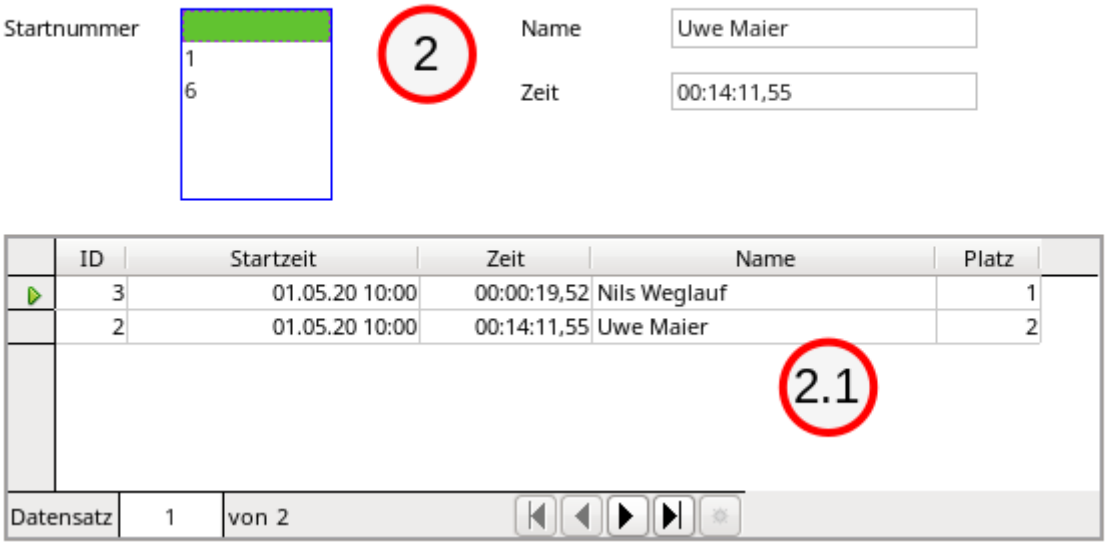
```
001 "Platz" ASC
```

Dies bedeutet, dass Datensätze nach der erreichten Platzierung, vermerkt im Feld "Platz", angezeigt werden sollen.

Start_Ziel_Platz



Zieleinlauf

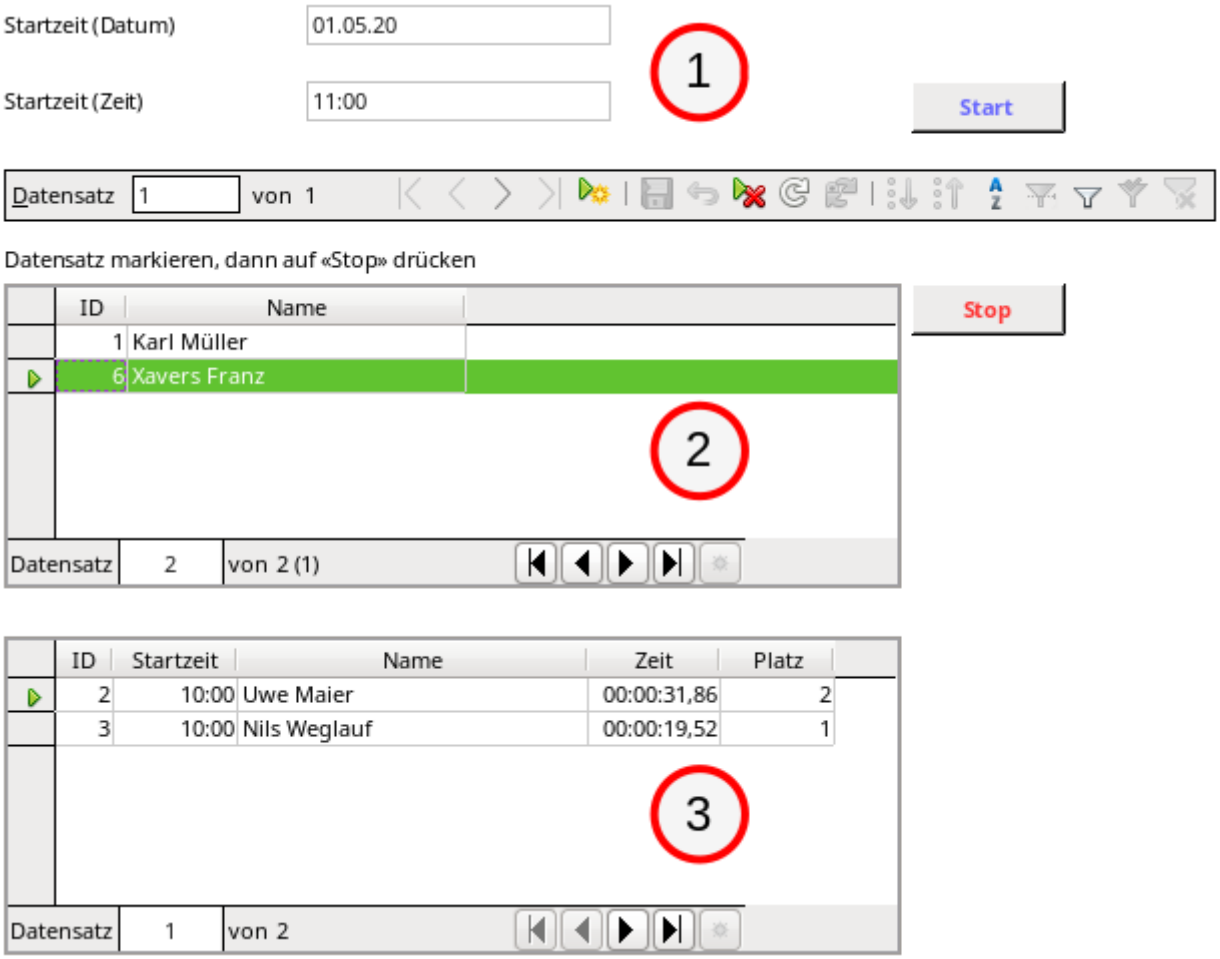


1	Zeitspeicher (Abfrage: *Abfrage_Start)*	**1.1**	Formular (Tabelle: *Start*)
2	Ziel (Abfrage: *Abfrage_Ziel*)	**2.1**	Platz (Abfrage: *Abfrage_Ziel*)
2	Listenfeld Startzeit (Tabelle: *Zeitspeicher, Start*)		

Makros in Feldeigenschaften		
1	Schaltfläche Start (Aktion ausführen)	Module1.*Zeitmessung_Start_Ziel*
2	Listenfeld «Startnummern» (Modifiziert)	Module1.*Zeitmessung_Ende_Listenfeld*

Dieses Formular ist lediglich eine Kombination der beiden Formulare "Nur_Start" und "Nur_Ziel". Die Übersicht zeigt schon, dass diese beiden Formulare nebeneinander auf der Formularfläche liegen. Es kann jetzt also im gleichen Formular gestartet und gestoppt werden.

Start_Ziel_Gruppe



1	Zeitspeicher (Abfrage *Abfrage_Start*)
2	Ziel (Abfrage *Abfrage_Unterwegs*)
3	Platz (Abfrage *Abfrage_Ziel*)

Makros in Feldeigenschaften		
1	Schaltfläche Start (Aktion ausführen)	Module1.*Zeitmessung_Start_Gruppe*
2	Schaltfläche Stop (Aktion ausführen)	Module1.*Zeitmessung_Ende_Gruppe*

Im Formular «Zeitspeicher» kann hier gegebenenfalls noch die Startzeit angepasst werden. Ansonsten ist das komplette Formular aber vor weiteren Eingaben geschützt, wie ein Blick auf die Tabellenkontrollfelder ohne zusätzliche Eingabezeile anzeigen.

Über Start wird an die Tabelle "Zeitspeicher" die Startzeit übergeben. Im Formular "Ziel" werden dann direkt alle gestarteten Personen mit den Startnummern "ID" angezeigt.

Kommt eine Person auf das Ziel zu, so wird der Startsatz markiert. Beim Zieldurchlauf wird Stop gedrückt. Dadurch wird ein Makro ausgelöst, das die Startzeit von der Zielzeit abzieht und als "Zeit" in die Tabelle "Start" überträgt. Anschließend wird Das Formular "Ziel" und das Formular "Platz" neu eingelesen. Die Person, die gerade das Ziel durchlaufen hat, verschwindet aus dem oberen Tabellenkontrollfeld und taucht mit Zeitangabe und Platzangabe direkt als erster Datensatz im unteren Tabellenkontrollfeld auf.

Das untere Formular «Platz» benötigt die folgenden zusätzlichen Einstellungen:

Formulareigenschaften → Daten → Filter

```
001 ( ( SELECT COUNT( "ID" ) FROM "Start" WHERE "Zeit" <= "a"."Zeit" AND
    "Startzeit" = "a"."Startzeit" ) > 0 )
```

Formulareigenschaften → Daten → Sortierung

```
001 "Startzeit" DESC, "Zeit" DESC
```

Der Filter sorgt dafür, dass nur die Datensätze angezeigt werden, bei denen der Wert für "Platz" größer als '0' ist. Hier schafft es das Filtermodul nicht, mit dem entsprechenden Alias zu arbeiten. Es benötigt stattdessen den Originalcode aus der Abfrage, der den Platz berechnet. Mit diesem Code hat jede Person zum Start den Platz '0'.

Die Sortierung zweigt die Einläufe nach der Startzeit (höchster Wert zuerst) und nach der Zeit (höchster Wert zuerst) sortiert an. Die Sortierung auch nach der Startzeit ist notwendig, weil in dem Tabellenkontrollfeld für das Formular «Platz» die zu unterschiedlichen Zeiten startenden Personen gemischt aufgelistet werden.

Berichte

Ergebnisliste

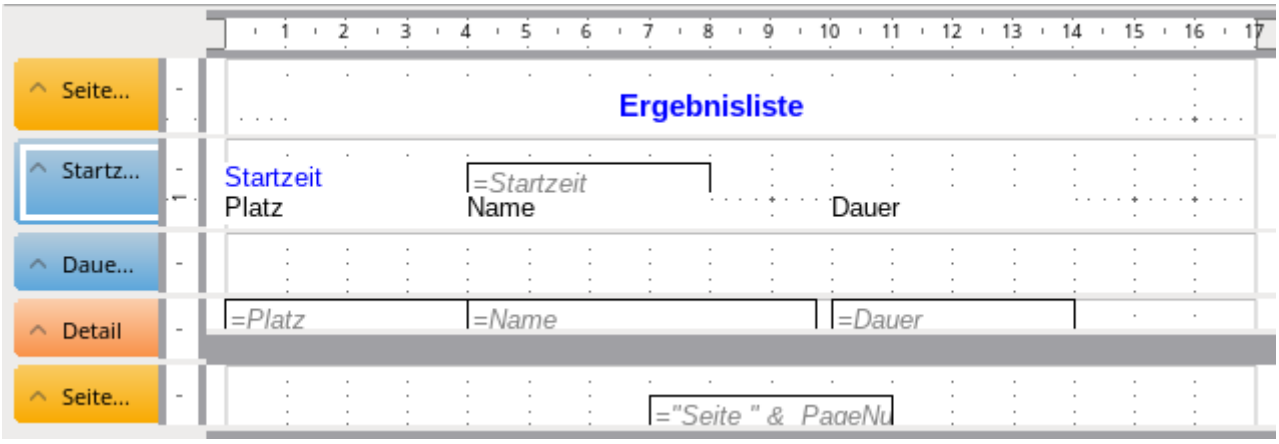
Ergebnisliste

Startzeit	01.05.20 10:00	
Platz	Name	Dauer
1	Karl Müller	212857 ms
2	Xavers Franz	216749 ms
3	Nils Weglauf	224895 ms
4	Uwe Maier	248272 ms
Startzeit	01.05.20 11:00	
Platz	Name	Dauer
1	Karl Cash	6224 ms
2	Kai Knieperich	9698 ms
3	Egon Werkel	13068 ms

Datenquelle
Abfrage: *Abfrage_Ziel*

Eine direkte Auswertung der Abfrage "Abfrage_Ziel" stellt die Ergebnisliste dar. Die Felder sind bereits in der Abfrage errechnet, so dass nur noch eine Gruppierung nach der Startzeit und nach der Platzierung notwendig ist. Dieser Bericht wurde nur für das Formular «Einzelmessung_System» erstellt, das eine Abspeicherung der Zeiten als Integer-Zahlen beinhaltet. Deswegen ist der Bericht auch nur in der HSQLBD-Beispieldatenbank vertreten.

Im Entwurf sieht dieser Bericht so aus:



Damit die Bezeichnungen «Platz», «Name» und «Dauer» nur pro Rennen einmal erscheinen sind sie in dem Gruppenkopf für die Startzeit verankert worden.

Der Gruppenkopf für das Feld «Dauer» dient nur der Sortierung nach der Zeit. Hier könnte also genauso gut das Feld «Platz» zur Sortierung genutzt werden. Dieser Gruppenkopf wird in dem Bericht nicht angezeigt (**Eigenschaften Gruppenkopf → Allgemein → Sichtbar: Nein**)

Auf jeder Seite erscheint eine Seitennummerierung in der Form Seite ... von ... .

Ergebnisliste_Zeit

Ergebnisliste

Startzeit	01.05.20 10:00	
Platz	Name	Dauer
1	Karl Müller	00:03:32,86
2	Xavers Franz	00:03:36,75
3	Nils Weglauf	00:03:44,90
4	Uwe Maier	00:04:08,27
Startzeit	01.05.20 11:00	
Platz	Name	Dauer
1	Karl Cash	00:00:06,22
2	Kai Knieperich	00:00:09,70
3	Egon Werkel	00:00:13,07

Datenquelle
Abfrage: *Abfrage_Ziel*

Dieser Bericht unterscheidet sich von dem vorangehenden Bericht nur dadurch, dass jetzt aus der Datenquelle das Feld "Zeit" statt des Feldes "Dauer" eingebunden wurde. Dadurch kann die Zeit dann ansprechender so dargestellt werden, dass die Darstellung einem üblichen Format entspricht.

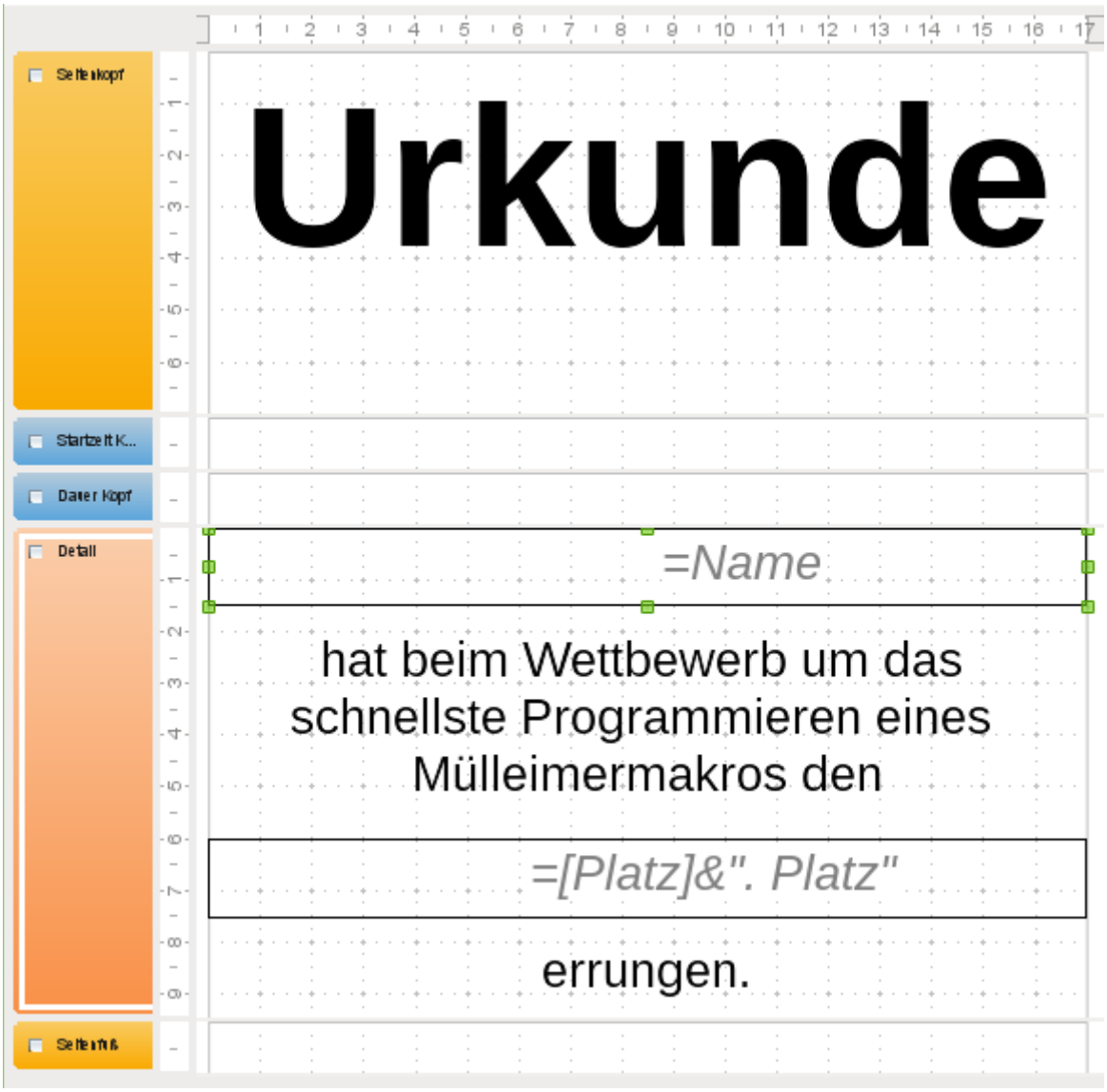
Da der Report-Builder reine Felder des Typs TIME nicht korrekt mit Nachkommastellen im Zeitbereich versieht, musste hier bei dem TIME-Feld von FIREBIRD eine Umwandlung zum TIME-STAMP-Feld in der Abfrage erfolgen. Das Feld für FIREBIRD heißt deswegen gesondert "RBZeit".

Urkunden

Datenquelle
Abfrage: *Abfrage_Ziel*

Auch ein Urkundendruck ist problemlos möglich. Sinnvoll ist hier allerdings, den Druckjob möglichst zu vereinfachen. Die einfachste Lösung wäre, nur die veränderlichen Felder im Bericht zu positionieren und auf einen schön gestalteten Urkundenvordruck zu drucken. Je einfacher der Bericht gehalten wird, desto schneller baut sich schließlich auch das Fenster für den Druck auf. Und je weniger grafische Elemente eingefügt sind, desto kürzer ist auch die Druckzeit, die während irgendeiner Veranstaltung sowieso recht knapp ist. Der Aufruf des Urkundendrucks für alle TeilnehmerInnen ist hier kein Problem. Da geht es eher um die Schnelligkeit, mit der der Report Builder die Seiten erstellt und der Drucker die Seiten verarbeitet.

Die Blattübersicht ist hier stark verkleinert. Schließlich ist der Druck «Urkunde» im Original auf eine Schriftart von 110 Punkten gesetzt worden.



In der verkleinerten Ansicht des Editors erscheint das Feld «Name» nicht richtig zentriert. Der Ausdruck zeigt aber eine entsprechenden saubere Zentrierung.

Für die Ausgabe der Platzierung ist eine kleine Formel erforderlich, da sonst die Zentrierung Probleme bereitet. Mit [Platz] wird das entsprechende Feld aus der Datenquelle angesprochen, mit &". Platz" wird an die Platzierung aus dem Datenfeld ein Punkt, ein Leerzeichen und der Begriff «Platz» angefügt.

Damit nur eine Urkunde pro Seite gedruckt wird ist bei **Eigenschaften Detail → Allgemein → Seitenumbruch erzwingen: Nach Bereich** ausgewählt.

Makros

Zum Starten und Stoppen der Zeiten ist jeweils eine Prozedur notwendig. In dieser Datenbank sind nur deshalb mehrere Prozeduren vorhanden, weil zum einen unterschiedliche Formen der Zeitmessung genutzt werden, zum anderen aber auch unterschiedliche Formulare vertreten sind, die jeweils auf ihre besondere Art und Weise den Zusatzkommandos zum Start oder zum Abspeichern des Zieldurchlaufs benötigen.

Starten und Stoppen der Zeit

Zeitmessung_Start_System

Aufruf aus
Formular: *Einzelmessung_System*

Benötigt	
Tabelle: *Zeitspeicher*.Zeitspeicher	nur HSQLDB

```
001  SUB Zeitmessung_Start_System(oEvent AS OBJECT)
002     DIM oForm AS OBJECT
003     DIM oFeld AS OBJECT
004     DIM oConnection AS OBJECT
005     DIM oSQL_Statement AS OBJECT
006     DIM stStartzeit AS STRING
007     DIM stSql AS STRING
008     DIM loZeit AS LONG
009     oForm = oEvent.Source.Model.Parent
010     stStartzeit = Left(oForm.getString(oForm.findColumn("Startzeit")),19)
011     loZeit = getSystemTicks() ' Gibt die Systemzeit in Millisekunden an
012     oConnection = oForm.activeConnection()
013     oSQL_Statement = oConnection.createStatement()
014     stSql = "UPDATE ""Zeitspeicher"" SET ""Zeitspeicher"" = '"+loZeit+"' WHERE
           ""Startzeit"" = '"+stStartzeit+"'"
015     oSQL_Statement.executeUpdate(stSql)
016  END SUB
```

Über das auslösende Ereignis, die Betätigung von Start, wird das Formular ermittelt (Zeile 9). Aus dem Feld "Startzeit" soll der Zeitstempel für die Startzeit ausgelesen werden (Zeile 10). Dieser kann auch Nachkommastellen enthalten. Deshalb wird der Wert als String ausgelesen und berücksichtigt nur die ersten 19 Zeichen (10 Zeichen für das Datum, ein Leerzeichen, 8 Zeichen für die Zeit).

Die momentane Zeit wird über die SystemTicks ermittelt (Zeile 11). Diese geben einen Ganzzahlwert wieder, der beständig durchläuft und jede Millisekunde um einen Wert erhöht wird.

Die Verbindung zur Datenbank wird aus dem Formular ausgelesen und das SQL-Statement vorbereitet. Die Formulierung für das Statement wird in der Variablen stSql erfasst (Zeile 14). Hier wird in der Tabelle "Zeitspeicher" das Feld "Zeitspeicher" mit dem Wert der SystemTicks beschrieben. Dabei wird aber nur das Feld beschrieben, das zu der Startzeit passt. In Zeile 15 wird diese SQL-Formulierung abgeschickt.

Zeitmessung_Ende_System

Aufruf aus
Formular: *Einzelmessung_System*

Benötigt	
Tabelle: *Zeitspeicher*.Zeitspeicher, *Start*.Dauer	nur HSQLDB

```
001  SUB Zeitmessung_Ende_System(oEvent AS OBJECT)
002     DIM oForm AS OBJECT
003     DIM oConnection AS OBJECT
004     DIM oSQL_Statement AS OBJECT
005     DIM inRow AS INTEGER
006     DIM stID AS STRING
007     DIM stSql AS STRING
008     DIM loZeit AS LONG
```

```
009     oForm = oEvent.Source.Model.Parent
010     stStartzeit = Left(oForm.getString(oForm.findColumn("Startzeit")),19)
011     stID = oForm.getString(oForm.findColumn("ID"))
012     loZeit = getSystemTicks() ' Gibt die Systemzeit in Millisekunden an
013     oConnection = oForm.activeConnection()
014     oSQL_Statement = oConnection.createStatement()
015     stSql = "SELECT "+loZeit+" - ""Zeitspeicher""FROM ""Zeitspeicher"" WHERE
           ""Startzeit"" = '"+stStartzeit+"'"
016     oResult = oSQL_Statement.executeQuery(stSql)
017     WHILE oResult.next
018        loZeit = oResult.getLong(1)
019     WEND
020     stSql = "UPDATE ""Start"" SET ""Dauer"" = '"+loZeit+"' WHERE
           ""ID"" = '"+stID+"'"
021     oSQL_Statement.executeUpdate(stSql)
022     inRow = oForm.getRow()
023     oForm.reload()
024     oForm.absolute(inRow)
025  END SUB
```

Wie in der vorhergehenden Prozedur werden Formular und Startzeit ausgelesen. Hinzugekommen ist das Auslesen des Feldes "ID" aus der Tabelle "Start" (Zeile 11). Dies ist ja der Primärschlüssel (und gleichzeitig die Startnummer), hinter der die Person steht, für die die Zielzeit eingetragen werden soll.

Die aktuellen SystemTicks werden in Zeile 12 ausgelesen. Über eine Abfrage wird von diesem Wert der vorher eingetragene Wert in der Tabelle "Zeitspeicher" abgezogen (Zeile 15). Der Wert wird in die Variable loZeit in Zeile 18 eingelesen.

Jetzt muss diese Zeit als Zeit für die entsprechende Person noch in die Tabelle "Start" übertragen werden. Dies erfolgt wieder über einen Update-Befehl (Zeile 20).

Zum Schluss wird die Datensatznummer des Formulars bestimmt, das Formular zur Übernahme der Daten neu geladen und auf die vorher ausgelesene Datensatznummer wieder eingestellt. Ohne das Auslesen der Datensatznummer würde stattdessen immer wieder nur der erste Datensatz erscheinen und nach jedem Zieldurchlauf müsste dann wieder gescrollt werden.

Zeitmessung_Start

Aufruf aus	
Formular:	*Nur_Start*
Makro:	*Zeitmessung_Start_Einzel, Zeitmessung_Start_Gruppe, Fehler: Verweis nicht gefunden*

Benötigt	
Tabelle:	*Zeitspeicher*.Timestamp

```
001  SUB Zeitmessung_Start(oEvent AS OBJECT)
002     DIM oForm AS OBJECT
003     DIM oConnection AS OBJECT
004     DIM oSQL_Statement AS OBJECT
005     DIM stStartzeit AS STRING
006     DIM stSql AS STRING
007     oForm = oEvent.Source.Model.Parent
008     IF oForm.RowCount > 0 THEN
009        stStartzeit = Left(oForm.getString(oForm.findColumn("Startzeit")),19)
010        oConnection = oForm.activeConnection()
011        oSQL_Statement = oConnection.createStatement()
012        stSql = "UPDATE ""Zeitspeicher"" SET ""Timestamp"" = NOW() WHERE
              ""Startzeit"" = '"+stStartzeit+"'"
013        oSQL_Statement.executeUpdate(stSql)
014        oForm.reload()
015     END IF
```

```
016 END SUB
```

Nur in den einfachen Formularen ist es möglich, einzelne Personen doppelt starten zu lassen. Einige Formular bieten hingegen irgendwann niemanden mehr an, der noch an den Start gehen kann. Hier kann es also passieren, dass bei Drücken von Start, gar kein Datensatz mehr vorhanden ist. Dies würde bei der obigen Prozedur dann zu einer Fehlermeldung führen. Deswegen ist hier die Nachfrage nach der angezeigten Datensatzanzahl wichtig. Nur wenn sie über 0 ist (Zeile 8) läuft der Rest der Prozedur ab.

Diesmal werden keine SystemTicks eingelesen sondern nur das Feld "Timestamp" mit dem aktuellen Wert beschrieben (Zeile 12). Es ist also nachher auch in der Tabelle "Zeitspeicher" möglich, die ganz genaue Startzeit zu erkennen.

In Zeile 12 unterscheidet sich der Code bei FIREBIRD nur geringfügig von dem obigen Code: **""Zeitspeicher"" = TIME 'NOW'** ersetzt hier **""Timestamp"" = NOW()**. Im Feld "Zeitspeicher" wird in FIREBIRD direkt die Zeit als Datentyp TIME gespeichert.

Zum Schluss wird das Formular wieder neu geladen, da die einmal gestarteten Personen nicht mehr zur Auswahl stehen sollen.

Zeitmessung_Ende

Aufruf aus	
Makro:	*Zeitmessung_Ende_Einzel, Zeitmessung_Ende_Gruppe, Zeitmessung_Ende_Listenfeld*

Benötigt	
Tabelle:	*Zeitspeicher*.Timestamp, *Start*.Zeit
Makro:	*Sekunden_2_SQLTime*

```
001 SUB Zeitmessung_Ende(oEvent AS OBJECT)
002    DIM oForm AS OBJECT
003    DIM oConnection AS OBJECT
004    DIM oSQL_Statement AS OBJECT
005    DIM oResult AS OBJECT
006    DIM stID AS STRING
007    DIM stStartzeit AS STRING
008    DIM stSql AS STRING
009    DIM stZeit AS STRING
010    DIM loZeit AS LONG
011    oForm = oEvent.Source.Model.Parent
012    IF oForm.RowCount > 0 THEN
013       stID = oForm.getString(oForm.findColumn("ID"))
014       stStartzeit = Left(oForm.getString(oForm.findColumn("Startzeit")),19)
015       oConnection = oForm.activeConnection()
016       oSQL_Statement = oConnection.createStatement()
017       stSql = "SELECT DATEDIFF('ms', ""Timestamp"", NOW()) FROM ""Zeitspeicher""
             WHERE ""Startzeit"" = '"+stStartzeit+"'"
018       oResult = oSQL_Statement.executeQuery(stSql)
019       WHILE oResult.next
020          loZeit = oResult.getLong(1) ' Zeitangabe in Millisekunden
021       WEND
022       stZeit = Sekunden_2_SQLTime(loZeit)
023       stSql = "UPDATE ""Start"" SET ""Zeit"" = '"+stZeit+"' WHERE ""ID"" =
             '"+stID+"'"
024       oSQL_Statement.executeUpdate(stSql)
025       oForm.reload()
026    END IF
027 END SUB
```

Wer einmal durch den Zieleinlauf gegangen ist soll hier nicht noch einmal registriert werden. Da die Formulare so angelegt sind, dass eben die Anzahl der Datensätze von den Personen, die unterwegs sind, beständig abnimmt, kann es passieren, dass gar kein Datensatz mehr da ist

und trotzdem Stop gesetzt würde. Wie beim Start wird deshalb auch hier überprüft, ob das Formular überhaupt noch Daten enthält (Zeile 12).

Mit der Abfrage in Zeile 17 wird der Unterschied zwischen dem Feld "Timestamp" und der aktuellen Zeit NOW() in der Maßeinheit Millisekunden ermittelt. Dieser Wert wird in Zeile 20 ausgelesen und in Zeile 22 an die Funktion «Sekunden_2_SQLTime» weiter gegeben. Diese Funktion macht daraus einen Timestamp, der anschließend in Zeile 23 in den SQL-Code eingefügt und als Zeit für die Person abgespeichert werden kann.

Zum Schluss wird das Formular wieder neu geladen, damit die Anzeige aktualisiert wird.

Der Code dieser Prozedur unterscheidet sich bei FIREBIRD ab Zeile 17:

```
017       stSql = "SELECT TIME 'NOW' - ""Zeitspeicher"" FROM ""Zeitspeicher"" WHERE
             ""Startzeit"" = '"+stStartzeit+"'"
018       oResult = oSQL_Statement.executeQuery(stSql)
019       WHILE oResult.next
020          doZeit = oResult.getDouble(1)
021       WEND
022       stZeit = Sekunden_2_SQLTime(doZeit)
023       stSql = "UPDATE ""Start"" SET ""Zeit"" = '"+stZeit+"' WHERE ""ID"" =
             '"+stID+"'"
```

Die Zeit aus dem Zeitspeicher wird direkt von der momentanen Zeit subtrahiert. Das Ergebnis ist eine Zahl mit Nachkommastellen, die sich nach der Maßeinheit Sekunde richtet. Deswegen wird hier die Variable als Double ausgelesen. Die Funktion, mit der die Zeit daraus für SQL-Zwecke ermittelt werden soll, hat zwar in FIREBIRD den gleichen Namen wie in HSQLDB. Der Inhalt ist aber nicht komplett identisch, da die Funktion hier ja einen anderen Wert geliefert bekommt und auch einen anderen Datentyp für die Weitergabe in die Tabelle "Start" liefern muss.

Der Zeitwert wird schließlich an die Tabelle Start für den entsprechenden Teilnehmer weiter gegeben.

Sekunden_2_SQLTime

Aufruf aus
Makro: *Zeitmessung_Ende*

```
001  FUNCTION Sekunden_2_SQLTime(loHSek AS LONG) AS STRING
002     DIM stSek AS STRING
003     DIM doHSek AS DOUBLE
004     DIM inSek AS INTEGER
005     DIM inMin AS INTEGER
006     DIM inStd AS INTEGER
007     DIM stHSek AS STRING
008     doSek = loHSek/1000
009     stSek = CStr(doSek)
010     stHSek = Mid(stSek, InStr(stSek,",")+1)
011     doHSek = doSek - Fix(doSek)
012     inSek = (doSek - doHSek) Mod 60
013     inMin = ( (doSek - inSek - doHSek) / 60 ) Mod 60
014     inStd = ( doSek - inSek - doHSek - 60 * inMin) / 3600
015     IF inStd < 0 THEN
016        inStd = inStd + 24
017     END IF
018     Sekunden_2_SQLTime = Year(Now) & "-" & Right( "0" & Month(Now) , 2) & "-" &
           Right( "0" & Day(Now) , 2) & " " & Right( "0" & inStd , 2) & ":" &
           Right( "0" & inMin , 2) & ":" & Right( "0" & inSek , 2) & "." & stHSek
019  END FUNCTION
```

Diese Funktion ist ausgelagert aus der Prozedur «Zeitmessung_Ende», die die erreichte Zeit in der Datenbank speichern soll. Sie ist zuerst für Firebird erstellt und hier nur für HSQLDB etwas abgewandelt worden.

Die Zeitangabe erfolgt in Millisekunden. Für die Weitergabe als Timestamp wird aber die Zeit in Sekunden, Minuten und Stunden benötigt. Zuerst wird deshalb die eingehende Angabe durch 1000 dividiert (Zeile 8). Die Nachkommastellen dieser Variablen stellen jetzt die Millisekunden dar. Sie werden in Zeile 10 als Text über den Dezimaltrenner ausgelesen. Basic richtet sich hier nach dem landesüblichen Trenner, hier also das Komma.

In den weiteren Schritten wird jeweils durch Restberechnung und Subtraktion der bereits herausgenommenen Reste zuerst die Anzahl der Sekunden, dann die der Minuten und schließlich die der Sekunden ermittelt.

In FIREBIRD kann es passieren, dass die Stundenanzahl schließlich negativ ist, wenn der Start am Vortag gewesen ist und der Zieldurchlauf am folgenden Tag. Schließlich wird ja nur die Zeit gespeichert, und die geht bis maximal 24 Stunden. Deswegen wird in Zeile 16 in diesem Fall einfach 24 Stunden zu der Zeitangabe hinzugefügt.

Zum Schluss wird schließlich bei der HSQLDB der Timestamp mit dem aktuellen Datum und der berechneten Zeit erstellt.

In FIREBIRD ist die Variable, die an die Funktion weitergegeben wird, bereits eine Variable mit Nachkommastellen, die dann die Tausendstelsekunden darstellt. In sofern entfällt am Anfang die Umwandlung.

Zum Schluss unterscheidet sich der Code auch noch, weil hier nicht ein Timestamp, sondern lediglich eine Zeit weitergegeben wird:

```
020     Sekunden_2_SQLTime = inStd & ":" & Right( "0" & inMin , 2) & ":" &
           Right( "0" & inSek , 2) & "." & stHSek
```

Die Umwandlung ist hier deutlich kürzer, weil eben das gesamte Datum fehlt. Wie bei der Umwandlung zum Timestamp wird auch hier dabei darauf geachtet, dass einstellige Zahlen für die Minuten und Sekunden durch eine zusätzliche 'o' zu zweistelligen Einträgen werden. Mit **`Right("0" & inMin,2)`** werden die 2 Zeichen von rechts ausgelesen. Bei zweistelligen Einträgen wird also die '0' nicht übernommen.

Formulare mit Zeitmessung verknüpfen

Jede der folgenden Prozeduren stellt nur die Verbindung der Formulare mit den Prozeduren zum Starten und zum Beenden her. In unterschiedlichen Formularen werden unterschiedliche zusätzliche Funktionen benötigt, die vor oder nach dem Abspeichern ablaufen müssen. Diese Prozeduren sind in HSQLDB und FIREBIRD gleich.

Zeitmessung_Start_Einzel

Aufruf aus
Formular: *Einzelmessung_Zeit*

Benötigt
Makro: *Zeitmessung_Start*

```
001 SUB Zeitmessung_Start_Einzel(oEvent AS OBJECT)
002    DIM oForm AS OBJECT
003    DIM inRow AS INTEGER
004    oForm = oEvent.Source.Model.Parent
005    inRow = oForm.getRow()
006    Zeitmessung_Start(oEvent)
007    oForm.absolute(inRow)
008 END SUB
```

Bevor der Start erfolgt wird hier die Zeile des aktuellen Datensatzes des Formulars ausgelesen. Nach Ablauf der Start-Prozedur (Zeile 6) wird das Formular wieder auf diese Zeile eingestellt.

Zeitmessung_Ende_Einzel

Aufruf aus
Formular: *Einzelmessung_Zeit*

Benötigt
Makro: *Zeitmessung_Ende*

```
001  SUB Zeitmessung_Ende_Einzel(oEvent AS OBJECT)
002     DIM oForm AS OBJECT
003     DIM inRow AS INTEGER
004     oForm = oEvent.Source.Model.Parent
005     inRow = oForm.getRow()
006     Zeitmessung_Ende(oEvent)
007     oForm.absolute(inRow)
008  END SUB
```

Wie «Zeitmessung_Start_Einzel», dieses Mal nur für das Ende der Zeitmessung (Zeile 6).

Zeitmessung_Start_Gruppe

Aufruf aus
Formular: *Start_Ziel_Gruppe*

Benötigt
Makro: *Zeitmessung_Start*

```
001  SUB Zeitmessung_Start_Gruppe(oEvent AS OBJECT)
002     DIM oForm AS OBJECT
003     Zeitmessung_Start(oEvent)
004     oForm = oEvent.Source.Model.Parent
005     oFormZiel = oForm.Parent.getByName("Ziel")
006     oFormZiel.reload()
007  END SUB
```

Die Prozedur «Zeitmessung_Start» läuft in einem anderen Formular ab. Das Formular «Ziel» wird danach aufgesucht und neu geladen, damit dort der Start registriert wird.

Zeitmessung_Ende_Gruppe

Aufruf aus
Formular: *Start_Ziel_Platz*, *Start_Ziel_Gruppe*

Benötigt
Makro: *Zeitmessung_Ende*

```
001  SUB Zeitmessung_Ende_Gruppe(oEvent AS OBJECT)
002     DIM oForm AS OBJECT
003     Zeitmessung_Ende(oEvent)
004     oForm = oEvent.Source.Model.Parent
005     oFormPlatz = oForm.Parent.getByName("Platz")
006     oFormPlatz.reload()
007  END SUB
```

Die Prozedur «Zeitmessung_Ende» läuft in einem anderen Formular ab. Das Formular «Platz» wird danach aufgesucht und neu geladen, damit dort der Start registriert wird.

Zeitmessung_Start_Ziel

```
001  SUB Zeitmessung_Start_Ziel(oEvent AS OBJECT)
002     DIM oForm AS OBJECT
003     DIM oFeld AS OBJECT
004     Zeitmessung_Start(oEvent)
005     oForm = oEvent.Source.Model.Parent
006     oFormZiel = oForm.Parent.getByName("Ziel")
007     oFeld = oFormZiel.getByName("Startnummern")
008     oFeld.refresh()
009  END SUB
```

Die Prozedur «Zeitmessung_Start» läuft in einem anderen Formular ab. Das Formular «Ziel» wird danach aufgesucht. Das Listenfeld «Startnummern» wird anschließend neu eingelesen (Zeile 8).

Zeitmessung_Ende_Listenfeld

Aufruf aus
Formular: *Nur_Ziel, Start_Ziel_Platz*

Benötigt
Makro: *Zeitmessung_Ende*

```
001  SUB Zeitmessung_Ende_Listenfeld(oEvent AS OBJECT)
002     DIM oForm AS OBJECT
003     DIM oFeld AS OBJECT
004     DIM stID AS STRING
005     oFeld = oEvent.Source.Model
006     stID = oFeld.CurrentValue
007     IF stID <> "" THEN
008        oForm = oFeld.Parent
009        oForm.Filter = """ID"" = '"+stID+"'"
010        oForm.ApplyFilter = TRUE
011        oForm.reload()
012        Zeitmessung_Ende(oEvent)
013        oFeld.refresh()
014     END IF
015  END SUB
```

Diese Prozedur wird von dem Listenfeld ausgelöst. In Zeile 6 wird der aktuelle Wert des Listenfeldes ausgelesen. Ist die Wert nicht leer (Zeile 7), so wird zuerst das Formular mit einem Filter versehen und neu geladen. In dem Formular ist jetzt der Datensatz zu sehen, der abgespeichert werden soll. Danach läuft die Prozedur «Zeitmessung_Ende» ab. Anschließend wird das Listenfeld neu eingelesen, da jetzt ja eine "ID" weniger zur Verfügung steht. Die Person ist durchs Ziel gekommen.

Aktuelle Standard-werte für Datum und Zeit

Einführung

In Formularen gibt es für Formularfelder die Möglichkeit, Standardwerte zu setzen. Damit werden über die grafische Benutzeroberfläche Wertvorgaben für bestimmte Felder gemacht, die abgespeichert werden, wenn sie nicht überschrieben werden.

Auch bei der Erstellung von Tabellen ist in den Feldeigenschaften ein Feld für den «Defaultwert» vorgesehen. Ein Eintrag an dieser Stelle erzeugt ebenfalls einen festen Vorgabewert für die Eingabe in ein Feld – jetzt aber bei Tabellen. Er hat im übrigen nichts mit dem Default-Wert der eigentlichen Datenbank zu tun. Der Default-Wert, definiert in einer Datenbank, wird nur dann geschrieben, wenn ein Feld als leeres Feld abgespeichert werden soll.

Alle Standardwerte der grafischen Benutzeroberfläche haben den Nachteil, dass sie nicht flexibel sind. Es kann nicht das beim Öffnen des Formulars aktuelle Datum oder die aktuelle Zeit eingegeben werden.

Die folgende Datenbank[2] zeigt an Beispielen, wie es möglich ist, ein aktuelles Datum bzw. eine aktuelle Datum-Zeit-Kombination beim Erstellen von Datensätzen zu erzeugen. Es zeigt außerdem, welche Möglichkeiten es gibt, ein Datum hinterher mit möglichst geringem Aufwand ändern zu lassen. Typische Beispiel wären hier eine Zeitmessung in Form von Arbeitsbeginn und Arbeitsende oder ein Vermerk, wann ein Datensatz erstellt und wann er zuletzt geändert worden ist.

Tabellen

Die Tabellen stellen nur einfache Beispieltabellen dar. Sie sollen nur zeigen, wie aktuelle Standardwerte für ein Datum bzw. ein Datum mit Zeitangabe gesetzt werden können.

Datum_Aenderdatum

Datenziel
Formular: *Defaultdatum_Aenderung, Defaultdatum_Makro, Defaultdatum_Makro_Standarddatum, Defaultdatum_Makro_Standarddatum_verlegt, Defaultdatum_Subform_neue_Datensaetze*
Abfrage: *DatumZeitvorgabe*
Bericht, Makro: keine direkt

Feldname	***Feldtyp***	***Beschreibung***
ID	Integer	Primärschlüssel der Tabelle. Der Primärschlüssel muss eindeutig sein. Das Feld kann auch als Auto-Wert-Feld gesetzt werden.
Name	Text	Der Name einer Person oder irgendeine Bezeichnung. Steht hier nur als Beispieleintrag ohne weitere Bedeutung. Eingabe erforderlich: Nein
Datum	Datum	Hier soll ein Startdatum gespeichert werden, das nachher nicht mehr verändert wird. Eingabe erforderlich: Nein
AenderDatum	Datum	Hier soll beim Erstellen des Datensatzes das gleiche Datum wie im Feld "Datum" abgespeichert werden. Dieses Feld wird beim erneuten Aufrufen des Datensatzes aber gegebenenfalls überschrieben. Eingabe erforderlich: Nein

2 Beispieldatenbank Beispiel_Default_Datum_Zeit.odb

Die Tabelle «Datum_SQLDefault_Aenderdatum» ist von den Feldern her völlig gleich aufgebaut.

Datenziel
Formular: *Defaultdatum_Makro_SQL*
Abfrage, Bericht, Makro: keine direkt

Hier ist im Unterschied zur Tabelle «Datum_Aenderdatum» allerdings über **Extras → SQL** hinterher ein SQL-Defaultwert eingestellt worden:

```
001 ALTER TABLE "Datum_SQLDefault_Aenderdatum"
002 ALTER COLUMN "Datum"
003 SET DEFAULT CURRENT_DATE;
001 ALTER TABLE "Datum_SQLDefault_Aenderdatum"
002 ALTER COLUMN "AenderDatum"
003 SET DEFAULT CURRENT_DATE;
```

Damit werden in dieser Tabelle beim Abspeichern eines neuen Datensatzes die Felder "Datum" und "AenderDatum" automatisch mit dem aktuellen Datum versehen, sofern nicht ein anderes Datum eingetragen wurde.

```
001 ALTER TABLE "Datum_SQLDefault_Aenderdatum"
002 ADD CHECK (COALESCE("AenderDatum","Datum")>="Datum");
```

Außerdem wird durch die Check-Bedingung sicher gestellt, dass der Eintrag in "AenderDatum" immer größer oder gleich dem Eintrag in "Datum" ist.

Zeitstempel_Aenderstempel

Datenziel
Formular: *Defaultdatum_Makro_Standardzeitstempel, Defaultdatum_Zeit_Makro, Defaultdatum_Zeit_Subform_neue_Datensaetze*
Abfrage: *Zeitdifferenzen, Zeitdifferenzen_Formularvorlage*
Bericht, Makro: keine direkt

Feldname	***Feldtyp***	***Beschreibung***
ID	Integer	Primärschlüssel der Tabelle. Der Primärschlüssel muss eindeutig sein. Das Feld kann auch als Auto-Wert-Feld gesetzt werden.
Name	Text	Der Name einer Person oder irgendeine Bezeichnung. Steht hier nur als Beispieleintrag, z.B. zur Ermittlung von Arbeitszeit. Eingabe erforderlich: Nein
Zeitstempel	Datum/Zeit	Hier soll eine Startzeit zusammen mit Datum gespeichert werden (**TIMESTAMP**), die nachher nicht mehr verändert wird. Eingabe erforderlich: Nein
Zeitstempel_Aenderung	Datum/Zeit	Hier soll beim Erstellen des Datensatzes die gleiche Zeit mit Datum wie im Feld "Zeitstempel" abgespeichert werden. Dieses Feld wird beim erneuten Aufrufen des Datensatzes aber gegebenenfalls überschrieben. Eingabe erforderlich: Nein

Die Tabelle «Zeitstempel_SQLDefault_Aenderstempel» ist von den Feldern her völlig gleich aufgebaut. Hier ist allerdings über **Extras → SQL** hinterher ein SQL-Defaultwert eingestellt worden.

Der Tabellenname musste für FIREBIRD gekürzt werden, da FIREBIRD für *Tabellennamen und Feldnamen nur maximal 30 Zeichen* zulässt. Dabei sind Standardzeichen gemeint. Sonderzeichen

wie Umlaute nehmen in der Regel gleich den Platz für 2 Zeichen ein. In FIREBIRD heißt die Tabelle also «Zeitstempel_SQLD_Aenderstempel».

Datenziel
Formular: *Defaultdatum_Zeit_Makro_SQL*
Abfrage, Bericht, Makro: keine direkt

```
001 ALTER TABLE "Zeitstempel_SQLDefault_Aenderstempel"
002 ALTER COLUMN "Zeitstempel"
003 SET DEFAULT CURRENT_TIMESTAMP;
001 ALTER TABLE "Zeitstempel_SQLDefault_Aenderstempel"
002 ALTER COLUMN "Zeitstempel_Aenderung"
003 SET DEFAULT CURRENT_TIMESTAMP;
```

Damit werden in dieser Tabelle beim Abspeichern eines neuen Datensatzes die Felder "Zeitstempel" und "Zeitstempel_Aenderung" automatisch mit der aktuellen Zeit und dem aktuellen Datum versehen, sofern nicht eine andere Zeit und ein anderes Datum eingetragen wurde.

```
001 ALTER TABLE "Zeitstempel_SQLDefault_Aenderstempel"
002 ADD CHECK
    (COALESCE("Zeitstempel_Aenderung","Zeitstempel")>="Zeitstempel");
```

Außerdem wird durch die Check-Bedingung sicher gestellt, dass der Eintrag in "Zeitstempel_Aenderung" immer größer oder gleich dem Eintrag in "Zeitstempel" ist.

Ansichten

Diese Datenbank enthält für die HSQLDB keine Ansichten.

In FIREBIRD sind statt der Abfragen *Zeitdifferenzen* und *Zeitdifferenzen_Formularvorlage* Ansichten notwendig. Die Abfragen sind dort wegen veränderten Codes nur im direkten SQL-Modus ausführbar und damit für die Ausgabe in Berichten nicht sortierbar und auch sonst nur begrenzt brauchbar.

Die Ansichten in FIREBIRD haben die Bezeichnung «Ansicht_Zeitdifferenzen» und «Ansicht_Zeitdifferenzen_Form». Die zweite Benennung musste wegen der Beschränkung auf 30 Zeichen für Tabellennamen gekürzt werden.

Abfragen

Die Datenbank enthält drei Abfragen. Für die Eingabe des Standarddatums ist lediglich die Abfrage «DatumZeitvorgabe» von Bedeutung. Die weiteren Abfragen dienen als Berichtsgrundlage («Zeitdifferenzen») oder als Vorlage für ein Formular, in dem gleichzeitig berechnete Differenzen betrachtet werden sollen («Zeitdifferenzen_Formularvorlage»).

DatumZeitvorgabe

Datenquelle
Tabelle: *Datum_Aenderdatum*

Datenziel
Formular: *Defaultdatum_Subform_neue_Datensaetze,* *Defaultdatum_Zeit_Subform_neue_Datensaetze*

Die Abfrage «DatumZeitvorgabe» dient lediglich dazu, das aktuelle Datum und den aktuellen Zeitstempel von der Datenbank zu erfragen. Damit kann ohne weitere Zuhilfenahme von direk-

ter SQL-Eingabe oder Makros das aktuelle Datum und (etwas begrenzt) auch der Zeitstempel in ein Formular übertragen werden.

```
001 SELECT DISTINCT CURRENT_TIMESTAMP AS "Jetzt",
002 CURRENT_DATE AS "Heute"
003 FROM "Datum_Aenderdatum"
```

Für eine Abfrage muss immer eine Tabelle als Basis stehen. Dies ist hier die Tabelle "Datum_Aenderdatum" (Zeile 3). In diesem Fall spielt es keine Rolle, welche Tabelle das ist, da nur Standardwerte aus der Datenbank ermittelt werden sollen: das aktuelle Datum **CURRENT_DATE** und die aktuelle Zeit kombiniert mit dem Datum **CURRENT_TIMESTAMP**. Durch den Zusatz **DISTINCT** (Zeile 1) werden die Werte nur einmal dargestellt. Ansonsten würde die Datenbank so viele Datensätze wiedergeben, wie die Tabelle bereits an Datensätzen hat – mit lauter gleichen Datums- und Zeitstempelangaben.

Zeitdifferenzen

Datenquelle
Tabelle: *Zeitstempel_Aenderstempel*

Datenziel
Bericht: *Bericht_Zeitdifferenzen*

Zur Auswertung von Zeiteingaben in einem Bericht wurde die Abfrage «Zeitdifferenzen» erstellt. Da in FIREBIRD diese Abfrage nur in etwas abgewandelter Form und in direktem SQL-Modus ausführbar ist (siehe unten) existiert dort stattdessen keine Abfrage sondern eine Ansicht mit der Bezeichnung «Ansicht_Zeitdifferenzen».

```
001 SELECT "ID",
002    "Name",
003    "Zeitstempel",
004    "Zeitstempel_Aenderung",
005    CAST ( EXTRACT ( YEAR FROM "Zeitstempel" ) || '-' ||
          RIGHT( '0' || EXTRACT ( MONTH FROM "Zeitstempel" ), 2 ) || '-' ||
          RIGHT( '0' || EXTRACT ( DAY FROM "Zeitstempel" ), 2 ) AS DATE )
          AS "StartDatum",
006    DATEDIFF( 'mi', "Zeitstempel", "Zeitstempel_Aenderung" )
          AS "Differenz_Minuten"
007 FROM "Zeitstempel_Aenderstempel"
```

Zuerst wird die komplette Tabelle "Zeitstempel_Aenderstempel" ausgewählt. Das Feld "ID" ist dabei nur notwendig, wenn so eine Abfrage für die Eingabe von Daten benutzt werden soll. Um eine Abfrage editierbar zu halten muss der Primärschlüssel der in der Abfrage enthaltenen Tabellen ebenfalls enthalten sein.

Aus dem Feld "Zeitstempel" ließe sich mit Hilfe der Formatierung auch lediglich das Datum darstellen. In der Abfrage wird zu Demonstrationszwecken gezeigt, wie die Datumsdarstellung auch erreicht werden kann:

Tag, Monat und Jahr können nur über separate Abfragen des Feldes "Zeitstempel" ermittelt werden. Um auch bei einstelligen Tageszahlen und Monatszahlen die vorangestellte '0' zu erhalten werden eine '0' und der ermittelte Tag zusammengefasst (**'0' || DAY("Zeitstempel")**) und dann aus dem entstehenden Text von rechts aus die 2 übrigbleibenden Zeichen ausgelesen. Das Datum wird dann in einem für SQL lesbares Datumsformat geschrieben: vierstellige Jahreszahl, zweistellige Monatszahl und zweistellige Tageszahl, verbunden mit einem Bindestrich (**YYYY-MM-DD**). Damit ist die Abfrage auch für andere Datumsschreibweisen nutzbar, denn dieser Text wird jetzt in das Datumsformat der Datenbank über **CAST (Datumstext AS DATE)** umgewandelt. Dadurch kann die Ausgabe z.B. im Bericht wieder entsprechend nach lokalen Einstellungen formatiert werden.

Mit **DATEDIFF('mi', "Zeitstempel", "Zeitstempel_Aenderung")** wird die Zeitdifferenz zwischen der Startzeit und der Endzeit in Minuten ermittelt. Diese Zeitdifferenz kann später im Bericht addiert werden und gegebenenfalls auch durch Umrechnung als eine Angabe von Minuten und Stunden erfolgen.

FIREBIRD benötigt hier eine andere Bezeichnung: **DATEDIFF(minute, "Zeitstempel", "Zeitstempel_Aenderung")**. Diese Schreibweise ohne die Maskierung mit einfachen Anführungszeichen versteht die GUI nicht, so dass dann die Abfrage nur noch im direkten SQL-Modus funktioniert.

Zeitdifferenzen_Formularvorlage

Datenquelle
Tabelle: *Zeitstempel_Aenderstempel*

Datenziel
Formular: *Defaultdatum_Zeitdifferenz_Makro*
Bericht: *Bericht_Zeitdifferenzen_Formularvorlage*

Als Formularvorlage ist die wesentlich umfangreichere Abfrage «Zeitdifferenzen_Formularvorlage» gedacht. Da in FIREBIRD diese Abfrage nur in etwas abgewandelter Form und in direktem SQL-Modus ausführbar ist (siehe unten) existiert dort stattdessen keine Abfrage sonder eine Ansicht mit der Bezeichnung «Ansicht_Zeitdifferenzen_Form».

```
001 SELECT "ID",
002    "Name",
003    "Zeitstempel",
004    "Zeitstempel_Aenderung",
005    CONVERT ( EXTRACT ( YEAR FROM "Zeitstempel" ) || '-' ||
          RIGHT( '0' || EXTRACT ( MONTH FROM "Zeitstempel" ), 2 ) || '-' ||
          RIGHT( '0' || EXTRACT ( DAY FROM "Zeitstempel" ), 2 ) , DATE )
          AS "StartDatum",
006    ( DATEDIFF( 'mi', "Zeitstempel", "Zeitstempel_Aenderung" ) -
          MOD( DATEDIFF( 'mi', "Zeitstempel", "Zeitstempel_Aenderung" ),
          60 ) ) / 60 || ':' || RIGHT( '0' || MOD( DATEDIFF( 'mi',
          "Zeitstempel", "Zeitstempel_Aenderung" ), 60 ), 2 )
          AS "Differenz_Stunden",
007    ( ( SELECT SUM( DATEDIFF( 'mi', "Zeitstempel",
          "Zeitstempel_Aenderung" ) ) FROM "Zeitstempel_Aenderstempel"
          WHERE "Name" = "a"."Name" ) - MOD( ( SELECT SUM( DATEDIFF( 'mi',
          "Zeitstempel", "Zeitstempel_Aenderung" ) ) FROM
          "Zeitstempel_Aenderstempel" WHERE "Name" = "a"."Name" ),
          60 ) ) / 60 || ':' || RIGHT( '0' || MOD( ( SELECT SUM( DATEDIFF(
          'mi', "Zeitstempel", "Zeitstempel_Aenderung" ) ) FROM
          "Zeitstempel_Aenderstempel" WHERE "Name" = "a"."Name" ), 60 ), 2
          ) AS "Gesamt_Stunden"
008 FROM "Zeitstempel_Aenderstempel" AS "a"
```

Bis zur Abfrage des Startdatums ist diese Abfrage gleich der Abfrage «Zeitdifferenzen»: Danach wird beständig die Zeitdifferenz in Minuten genutzt: **DATEDIFF('mi', "Zeitstempel", "Zeitstempel_Aenderung")** (Zeile 6). Die Zahlenangabe kann nicht in ein Zeitformat der Datenbank umgewandelt werden, da in der Datenbank nur Zeiten bis maximal 24 Stunden verwaltet werden. Es wird hier also lediglich in einer Textform die Zeitdarstellung nachgebildet.

Zuerst werden die Minuten durch 60 dividiert, um die Anzahl in Stunden zu ermitteln. Eigentlich müsste es hier genügen, beim Ergebnis die Nachkommastellen abzuschneiden oder auch abzurunden. Dies gelingt leider nicht, da bei den Befehlen TRUNCATE() und CEILING() das Ergebnis immer als Dezimalzahl mit einer Nachkommastelle ausgegeben wird – auch wenn die Nach-

kommastelle 0 ist. Um ein Ergebnis ohne Nachkommastellen zu erhalten muss von vornherein der verbleibende Rest der Division **`MOD ( Minuten, 60)`** subtrahiert werden. Dann wird bei den anschließenden Division die korrekte Ganzzahl ausgegeben.

Anschließend wird ein Doppelpunkt über **`||':'`** angehängt und an diesen wiederum der Rest aus der Division der Minuten durch 60. Vor die Division wird eine führende '0' gesetzt, damit auf jeden Fall eine zweistellige Zahl für die Minutendarstellung vorhanden ist. Wie beim Datum werden mit **`RIGHT (Text,2)`** einfach die zwei Zeichen des Textes übernommen, die am weitesten rechts stehen.

Mit dem entsprechenden Verfahren wird auch die Gesamtzahl an Stunden ermittelt. Da diese pro Person ermittelt werden soll sind hier für die Ermittlung der Stunden und Minuten korrelierende Unterabfragen notwendig. Dazu ist zuerst einmal der Tabelle der Hauptabfrage "Zeitstempel_Aenderung" der **`Alias "a"`** zugeordnet. Anschließend wird in der Unterabfrage die Summe **`SUM`** für die Zeitdifferenz in Minuten **`( DATEDIFF( 'mi', "Zeitstempel", "Zeitstempel_Aenderung" ) )`** gebildet – aber nur bezogen auf den jeweiligen Namen, der im aktuellen Datensatz der Hauptabfrage steht: **`WHERE "Name" = "a"."Name"`**. Die Berechnungen und Umwandlungen der Ergebnisse aus den korrelierenden Unterabfragen sind gleich den Berechnungen und Umwandlungen aus der direkten Ermittlung der Zeitdifferenz in Minuten für den jeweiligen Datensatz. Das Ergebnis wird als "Gesamt_Stunden" ausgegeben.

In FIREBIRD sind hier die Zeilen 6 und 7 etwas anders zu formulieren:

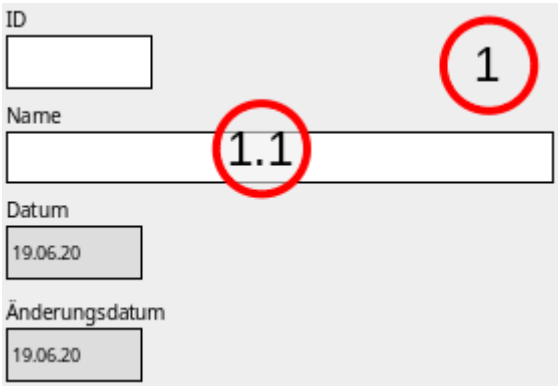
```
 006 ( ( DATEDIFF(minute, "Zeitstempel", "Zeitstempel_Aenderung" ) -
     MOD( DATEDIFF(minute, "Zeitstempel", "Zeitstempel_Aenderung" ), 60 ) ) /
     60 ) || ':' || RIGHT( '0' || MOD( DATEDIFF( minute, "Zeitstempel",
     "Zeitstempel_Aenderung" ), 60 ), 2 ) AS "Differenz_Stunden",
 007 ( ( ( SELECT SUM( DATEDIFF( minute, "Zeitstempel", "Zeitstempel_Aenderung"
     ) ) FROM "Zeitstempel_Aenderstempel" WHERE "Name" = "a"."Name" ) -
     MOD( ( SELECT SUM( DATEDIFF( minute, "Zeitstempel",
     "Zeitstempel_Aenderung" ) ) FROM "Zeitstempel_Aenderstempel" WHERE "Name"
     = "a"."Name" ), 60 ) ) / 60 ) || ':' || RIGHT( '0' || MOD( ( SELECT
     SUM( DATEDIFF(minute, "Zeitstempel", "Zeitstempel_Aenderung" ) ) FROM
     "Zeitstempel_Aenderstempel" WHERE "Name" = "a"."Name" ), 60 ), 2 ) AS
     "Gesamt_Stunden"
```

Zum einen ist wieder der Befehl **`DATEDIFF`** durch die Angabe von **`minute`** ohne die Maskierung mit einfachen Anführungszeichen erforderlich. Dadurch muss die Abfrage als direkte SQL-Abfrage ausgeführt werden.

Zum anderen hat FIREBIRD Probleme, Ohne eine entsprechende Klammerung Rechungen von Strings zu unterscheiden. Deshalb ist in Zeile 6 und Zeile 7 eine schließende Klammer hinter **`/ 60 )`** erforderlich. Natürlich muss entsprechend auch eine öffnende Klammer weiter vorne mit eingesetzt werden. Nur so funktioniert die Rechnung, die innerhalb einer Verbindung mit **`||`** erfolgt.

Formulare

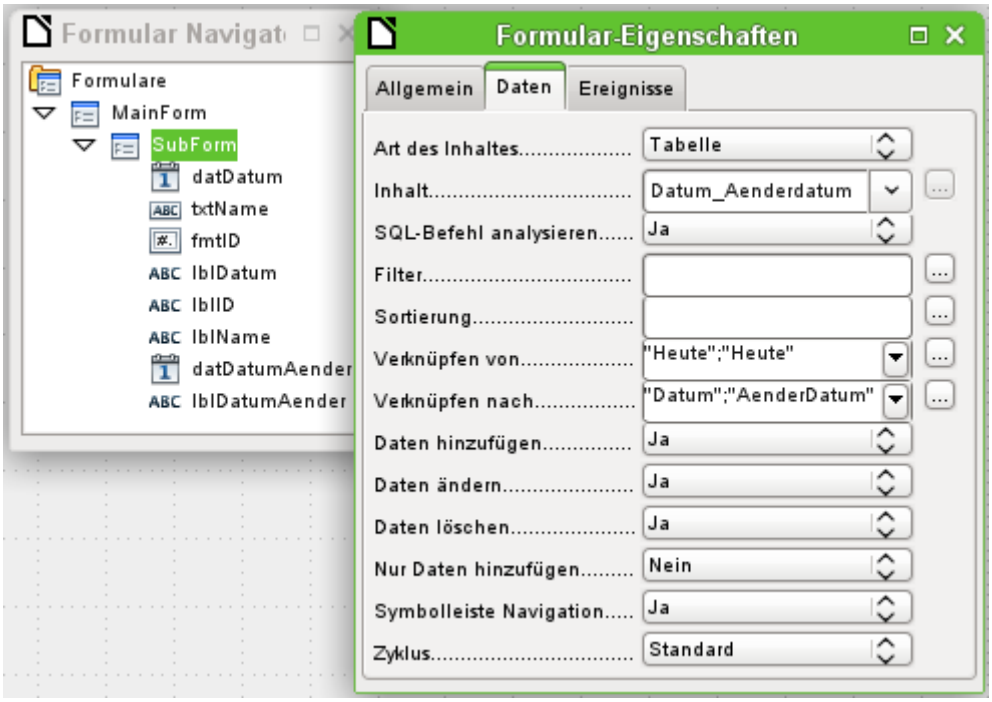
Defaultdatum_Subform_neue_Datensaetze



1	MainForm (Abfrage: *DatumZeitvorgabe*)	**1.1**	SubForm (Tabelle: *Datum_Aenderdatum*)

Die Formulare «Defaultdatum_Subform_neue_Datensaetze» und «Defaultdatum_Zeit_Subform_neue_Datensaetze» arbeiten nach dem gleichen Prinzip. Wird das Formular geöffnet, so ist bereits das Datum bzw. der Zeitstempel vorgegeben. Im Datumsformular erscheinen die Datensätze des aktuellen Tages und können auch nur Datensätze für den aktuellen Tag eingegeben werden.

Es ist lediglich das Datenfeld für den Primärschlüssel "ID" sowie für das Textfeld "Name" ausfüllbar. Die in Grau gehaltenen Formularfelder sind schreibgeschützt.

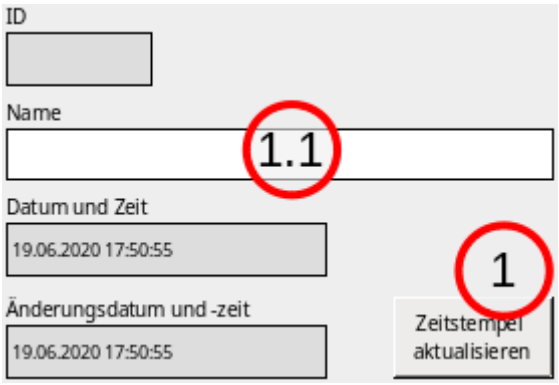


Wird das Formular zum Bearbeiten geöffnet und der Formularnavigator aufgerufen, so wird die Verbindung von Hauptformular **`MainForm`** und Unterformular **`SubForm`** sichtbar. Das Hauptformular hat als Inhalt die Abfrage «DatumZeitvorgabe». Von dem Hauptformular wird das ermittelte aktuelle Datum an das Unterformular übergeben. Das Feld "Heute" ist mit den Feldern "Datum" und "AenderDatum" des Unterformulars verknüpft.

Im Hauptformular des Datumformulars existieren keine Formularfelder.

Mit Hilfe der Hauptformular-Unterformular-Konstruktion kann auf einfache Weise das aktuelle Datum übertragen und automatisch mit Datensätzen abgespeichert werden. Änderungen von bereits eingegebenen Datumswerten sind nur über die direkte Eingabe per Hand möglich.

Defaultdatum_Zeit_Subform_neue_Datensaetze



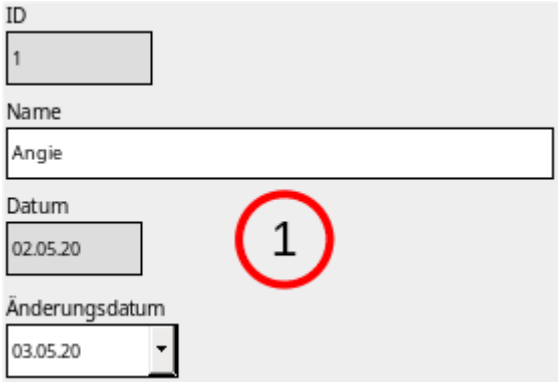
1	MainForm (Abfrage: *DatumZeitvorgabe*)	**1.1**	SubForm (Tabelle: *Zeitstempel_Aenderstempel*)

Im Zeitformular erscheinen auf keinen Fall die vorhergehenden Datensätze, da die Zeit zum Öffnen des Formulars maßgebend ist.

Im Hauptformular des Zeitformulars ist ein Button enthalten. Dieser Button ist in den **Eigenschaften: Schaltfläche → Allgemein → Aktion** auf **Formular aktualisieren** eingestellt. Dadurch werden die Zeiten gegebenenfalls neu eingelesen und übertragen. Würde dies nicht möglich sein, so hätten alle nach dem Öffnen des Formulars erstellten Datensätze den gleichen Zeitstempel.

Bei Zeitangaben muss schon eine laufende Aktualisierung durch einen Button erfolgen. Sie sind in dieser Konstruktion also nur bedingt brauchbar. Änderungen von bereits eingegebenen Zeitstempelwerten sind nur über die direkte Eingabe per Hand möglich.

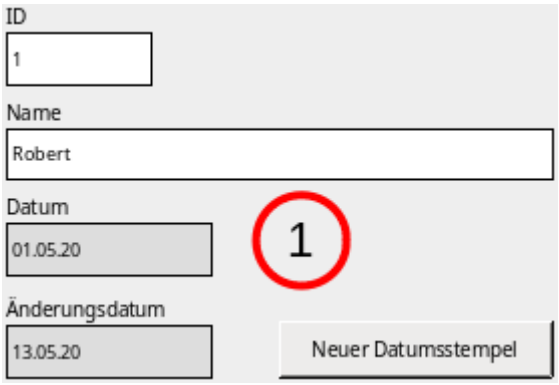
Defaultdatum_Aenderung



1	MainForm (Tabelle: *Datum_Aenderdatum*)

Um ein Datum bzw. einen Zeitstempel zu ändern ist ein separates Formular erforderlich, hier das Formular «Defaultdatum_Aenderung». Die Änderung kann nicht in dem vorhergehenden Formular erfolgen, da das Änderungsfeld dort mit dem Hauptformular verbunden ist und bereits eine Datums- bzw. Zeitvorgabe hat.

Defaultdatum_Makro_SQL



1	MainForm (Tabelle: *Datum_SQLDefault_Aenderdatum*)

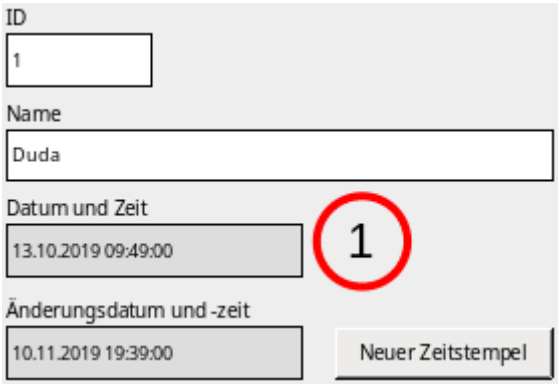
Makros in Feldeigenschaften		
1	Button Neuer Datumsstempel (Aktion ausführen)	Module1.*Update*

Die Formulare «Defaultdatum_Makro_SQL» und «Defaultdatum_Zeit_makro_SQL» sind wiederum nach dem gleichen Prinzip aufgebaut.

Bei diesen Formularen erscheint beim Erstellen eines neuen Datensatzes kein Eintrag in den Datums- bzw. Zeitstempel-Feldern. Die Tabelle, die mit dem Formular bearbeitet wird, hat einen Defaultwert über SQL für die Datums- bzw. Zeitstempel-Felder erhalten. Dieser Wert wird beim Abspeichern eines neuen Datensatzes in die Tabelle geschrieben. Der Nutzer des Formulars sieht den Wert also erst, wenn ein Datensatz abgespeichert wurde.

Wenn ein Datensatz geändert werden soll, so kann ein neues Datum bzw. ein neuer Zeitstempel über den Button erstellt werden. Der Button ist hierfür über das **Ereignis → Aktion ausführen** einem Makro verbunden.

Defaultdatum_Zeit_Makro_SQL



1	MainForm (Tabelle: *Zeitstempel_SQLDefault_Aenderstempel*)

Makros in Feldeigenschaften		
1	Button Neuer Zeitstempel (Aktion ausführen)	Module1.*Update*

In den Zusatzinformationen das Buttons ist der SQL-Code angegeben, der bei einer Änderung ausgeführt werden soll. Außerdem ist hinter einem Semikolon vermerkt, in welchen der Formularfelder der Wert für den Primärschlüssel steht. Der Primärschlüssel wird für die Änderung nur

des einen Datensatzes benötigt. Ohne den Schlüssel würden Änderungen für alle Datensätze gelten.

Eintrag für den Zeitstempel-Button:

```
001 UPDATE "Zeitstempel_SQLDefault_Aenderstempel" SET "Zeitstempel_Aenderung"
    = CURRENT_TIMESTAMP;fmtID
```

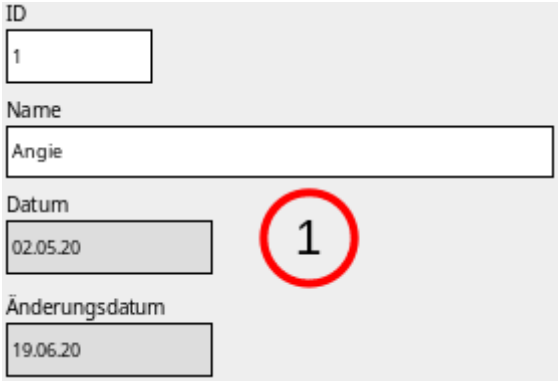
Eintrag für den Datumsstempel-Button:

```
001 UPDATE "Datum_SQLDefault_Aenderdatum" SET "AenderDatum" =
    CURRENT_DATE;fmtID
```

Der Button kann also so zum einen abspeichern, zum anderen auch direkt den Wert aus der Datenbank wieder auslesen, so dass die Eingabe auf jeden Fall für die Person, die das Formular bedient, erscheint.

Bei FIREBIRD muss im Befehl für den Zeitstempel der Tabellenname noch angepasst werden, da die Bezeichnung über 30 Zeichen lang ist.

Defaultdatum_Makro

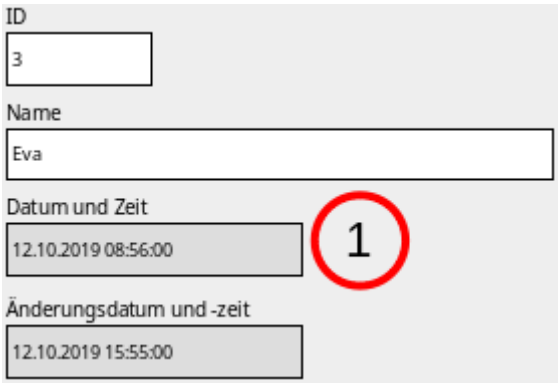


1	MainForm (Tabelle: *Datum_Aenderdatum*)

Makros in Formulareigenschaften		
1	Vor der Datensatzaktion	Module1.*Datum_aktuell*

In dem Formular befinden sich nur zwei beschreibbare Eingabefelder: Das Primärschlüsselfeld und das Namensfeld. Die Werte in den Feldern für das Speicherdatum und das Änderungsdatum bzw. den Speicher-Zeitstempel und den Änderungs-Zeitstempel werden ohne Eingriff des Nutzers automatisch geschrieben. So wird immer die letzte Änderung in dem Änderungsstempel festgehalten.

Defaultdatum_Zeit_Makro

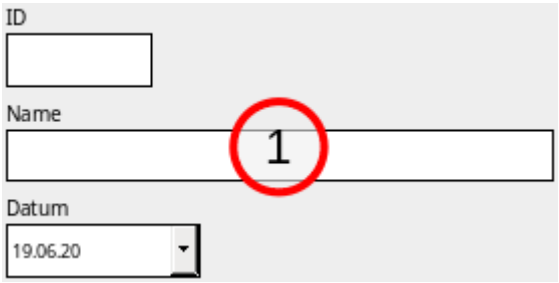


1	MainForm (Tabelle: *Zeitstempel_Aenderstempel*)

Makros in Formulareigenschaften		
1	Vor der Datensatzaktion	Module1.*Datum_Zeit_aktuell*

Dieses Formular setzt immer automatisch über das entsprechende Makro den Zeitstempel ein. Dierekt beim ersten Abspeichern sind so die beiden Felder mit gleichen Zeitstempeln versehen. Beim erneuten Abspeichern, z.B. nach einer Datensatzänderung, wird dann nur das Feld für «Ändetrungsdatum und -zeit» mit einem neuen Wert versehen.

Defaultdatum_Makro_Standarddatum



1	MainForm (Tabelle: *Datum_Aenderdatum*)

Makros in Formulareigenschaften		
1	Nach dem Datensatzwechsel	Module1.*Standarddatum*

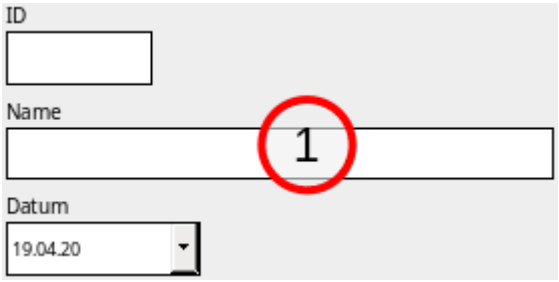
Über die Eigenschaften eines Kontrollfeldes lassen sich Standardwerte festlegen. Beim Datumsfeld ist dies mit **Eigenschaften: Datumsfeld → Allgemein → Standarddatum** zu erreichen. Diese Einstellungen sind dann dauerhaft in dem Formular gespeichert. Ein aktuelles Datum oder ein aktueller Zeitstempel können in der grafischen Benutzeroberfläche nicht vorgegeben werden.

Die Standardwerte erscheinen, wie in dem Formular zu sehen, bereits beim Erstellen eines neuen Datensatzes. Sie werden für das Formular allerdings nicht als Datensatzänderung verzeichnet. Erst wenn ein anderes Feld oder eben das Feld mit dem Standardwert geändert wurde nimmt die grafische Benutzeroberfläche das als Änderung wahr und speichert den Datensatz auf die übliche Art und Weise ab (Button »Speichern», Navigation zum nächsten Datensatz, Navigation zum Unterformular u.ä.).

Bei diesem Formular ist zu Bedenken, dass der Wert beim Erreichen eines neuen Datensatzes bereits im Formular erscheint. Bei Datumswerten ist das allerdings weniger von Bedeutung, es

sei denn, die Bearbeitung eines Datensatzes beginnt z.B. um 23:59 Uhr und endet um 00:01 Uhr des Folgetages.

Defaultdatum_Makro_Standarddatum_verlegt

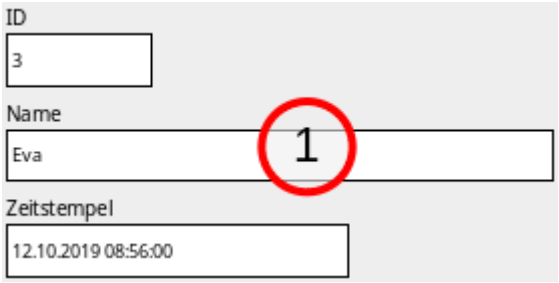


1	MainForm (Tabelle: *Datum_Aenderdatum*)

Makros in Formulareigenschaften		
1	Nach dem Datensatzwechsel	Module1.*Standarddatum_verlegt*

Beim Datumsfeld legt der Standardwert gleichzeitig fest, mit welcher Monatsansicht die Aufklappfunktion des Datumsfeldes startet. Werden z.B. häufig Datumswerte eingegeben, die im Vormonat liegen, so kann über ein Standarddatum aus dem Vormonat die Auswahl entsprechend eingestellt werden.

Defaultdatum_Makro_Standardzeitstempel



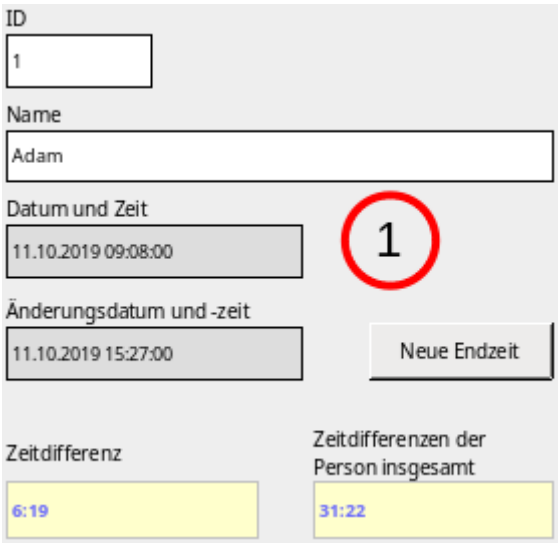
1	MainForm (Tabelle: *Zeitstempel_Aenderstempel*)

Makros in Formulareigenschaften		
1	Nach dem Datensatzwechsel	Module1.*StandardZeitstempel*

Über die Eigenschaften eines Kontrollfeldes lassen sich Standardwerte festlegen. Hier beim formatierten Feld für den Zeitstempel über **Eigenschaften: Formatiertes Feld → Allgemein → Standardwert**.

Bei diesem Formular ist zu Bedenken, dass der Wert beim Erstellen eines neuen Datensatzes bereits im Formular erscheint. Es handelt sich also bei der Zeit nicht um den Zeitpunkt, zu dem die Abspeicherung des Datensatzes erfolgte.

Defaultdatum_Zeitdifferenz_Makro



1	HSQDB: MainForm (Abfrage: *Zeitdifferenzen_Formularvorlage*)		
1	FIREBIRD: MainForm (Tabelle: *Zeitstempel_Aenderstempel*)	**1.1**	FIREBIRD: SubForm (Ansicht: *Ansicht_Zeitdifferenzen_Form*)

Makros in Formulareigenschaften		
1	Vor der Datensatzaktion	Module1.*Datum_Zeit_aktuell*
Makros in Feldeigenschaften		
1	Button Neue Endzeit (Aktion ausführen)	Module1.*UpdateTimestamp*

Dieses Formular zeigt neben dem Zeitstempel für die erste Abspeicherung und dem Zeitstempel für die Änderung des Datensatzes die Differenz zwischen den Zeiten sowie die Differenz zwischen allen Zeiten, die mit der Person verbunden sind, an.

Wird auf den Button «Neue Endzeit» gedrückt, so wird das Änderungsdatum und die Änderungszeit neu gesetzt und die Zeitdifferenzen werden neu eingelesen.

Berichte

Für diese Beispieldatenbank wurden zwei Berichte erstellt. Beide greifen auf die gleiche Tabelle «Zeitstempel_Aenderstempel» zu – allerdings über verschiedene Abfragen. Bei dem ersten Bericht ist vor allem mit Formeln innerhalb des Berichtes gearbeitet worden. Beim zweiten Bericht ist das Ergebnis nahezu identisch – nur sind die Inhalte komplett aus der Abfrage übernommen worden, so dass innerhalb des Berichtes keine Rechnung mehr ausgeführt wird.

Bericht_Zeitdifferenzen

Name: *Adam*

StartDatum	Zeitstempel	Zeitstempel Änderung	Differenz in Minuten
11.10.19	11.10.19 09:08	11.10.19 15:27	379
12.10.19	12.10.19 00:03	13.10.19 01:02	1499
18.10.19	18.10.19 21:15	18.10.19 21:19	4

Zeitsumme in Minuten: **1882**
Zeitsumme in Stunden: **31:22**

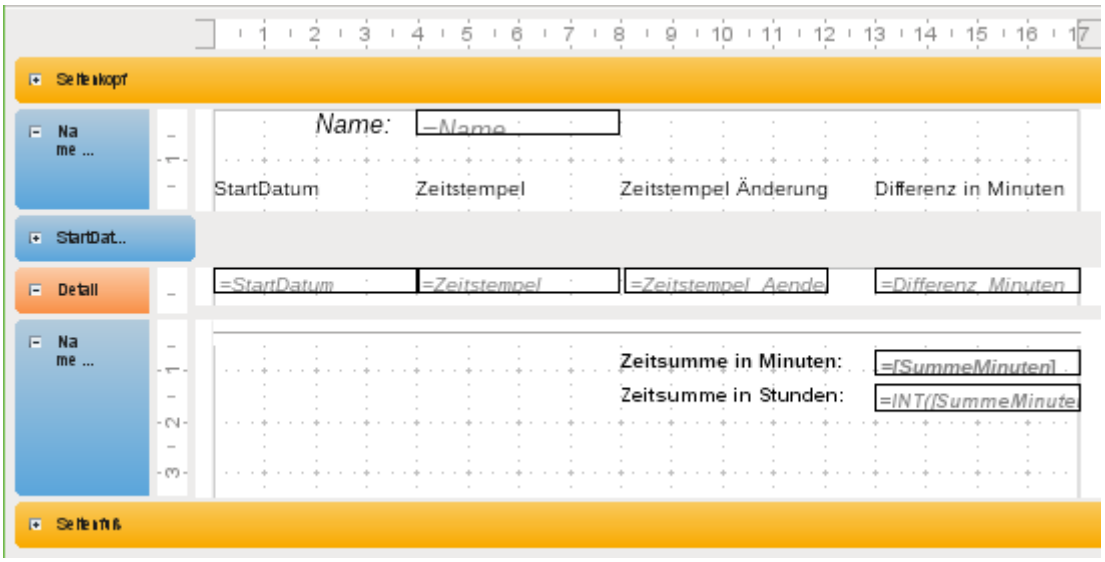
Name: *Eva*

StartDatum	Zeitstempel	Zeitstempel Änderung	Differenz in Minuten
11.10.19	11.10.19 09:12	11.10.19 15:07	355
12.10.19	12.10.19 08:56	12.10.19 15:55	419
13.10.19	13.10.19 08:31	14.10.19 09:30	1499
18.10.19	18.10.19 21:21	18.10.19 21:23	2
18.10.19	18.10.19 20:56	18.10.19 21:19	23
18.10.19	18.10.19 20:42	18.10.19 20:57	15
19.10.19	19.10.19 09:26	19.10.19 09:50	24

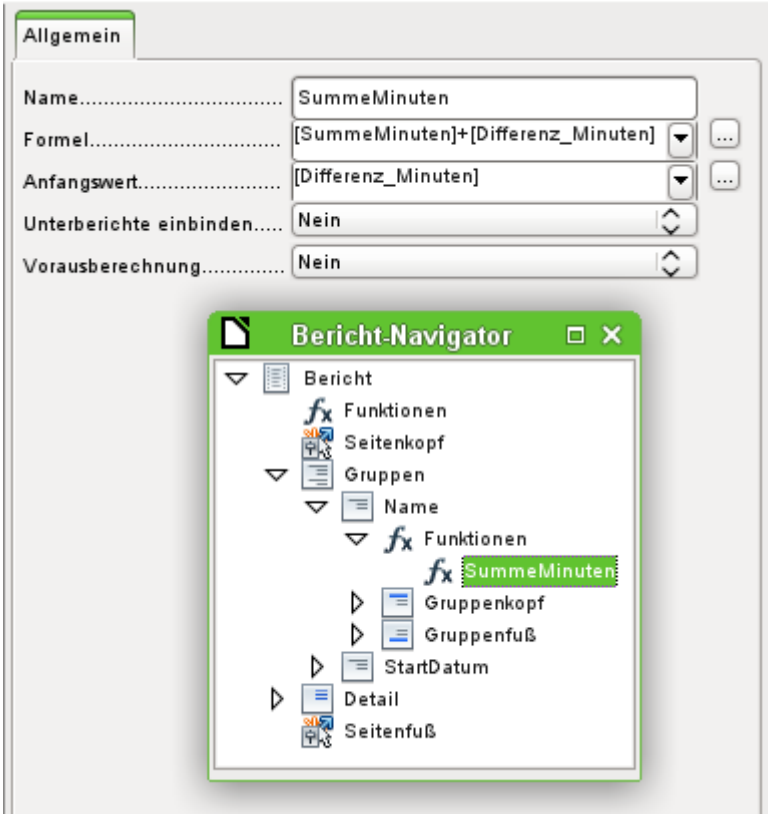
Zeitsumme in Minuten: **2337**
Zeitsumme in Stunden: **38:57**

Datenquelle			
Abfrage:	*Zeitdifferenzen*	FIREBIRD:	*Ansicht_Zeitdifferenzen*

Der Bericht zeigt, aufgelistet nach Personen, die Zeitdifferenzen für jedes einzelne Startdatum in der Maßeinheit Minuten. Außerdem wird die Summe der Zeitdifferenzen in Minuten und schließlich auch in Stunden angegeben. So könnte z.B. eine Zusammenstellung über Arbeitszeiten funktionieren.

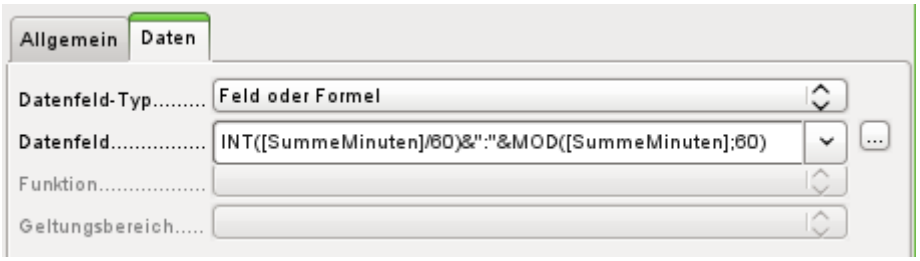


Gruppierungen und Sortierungen greifen beim Report-Builder ineinander. Der Bericht ist zuerst gruppiert nach dem Namen, zu dem es auch einen Gruppenfuß gibt. Dann erfolgt eine weitere Gruppierung nach dem Startdatum. Diese Gruppierung steht zwar in der Übersicht, wird aber im Bericht nicht gezeigt. Dadurch wird lediglich nach dem Startdatum sortiert. Seitenkopf und Seitenfuß sind hier ausgeblendet und stehen in den Eigenschaften auf **Sichtbar → Nein**.



Im Gruppenfuß befindet sich als erstes das Feld für die Summierung der Summe. Die Funktion wurde händisch erstellt, da die Automatik von LO leider versagte. Über den Berichtsnavigator ist so eine Funktion anschließend erreichbar und kann dann auch noch bearbeitet werden.

[Differenz_Minuten] ist der Feldwert aus der Abfrage «Zeitdifferenzen». Der erste Wert wird gelesen, als Wert der Formel gespeichert und anschließend wird der nächste Wert der Abfrage einfach hinzu addiert. Auf die Formel kann anschließend wie auf das Feld der Abfrage mit eckigen Klammern zugegriffen werden: **[SummeMinuten]**.



Für die Zeitsumme in Stunden wird bei den Daten des Feldes eine Formel eingetragen. Sie liest zuerst den Wert der Funktion **[SummeMinuten]** aus und teilt diesen durch 60. Mit der Funktion **INT** werden daraus nur die Ganzzahlbestandteile übernommen. Anschließend wird mit **&":"&** die Anzahl der verbleibenden Minuten angehängt. **MOD([SummeMinuten];60)** stellt den «Rest» der Division der Minutenanzahl durch 60 dar.

Bericht_Zeitdifferenzen_Formularvorlage

Name: *Adam*

StartDatum	Zeitstempel	Zeitstempel Änderung	Differenz in Stunden
11.10.19	11.10.19 09:08	11.10.19 15:27	6:19
12.10.19	12.10.19 00:03	13.10.19 01:02	24:59
18.10.19	18.10.19 21:15	18.10.19 21:19	0:04
		Zeitsumme in Stunden:	**31:22**

Name: *Eva*

StartDatum	Zeitstempel	Zeitstempel Änderung	Differenz in Stunden
11.10.19	11.10.19 09:12	11.10.19 15:07	5:55
12.10.19	12.10.19 08:56	12.10.19 15:55	6:59
13.10.19	13.10.19 08:31	14.10.19 09:30	24:59
18.10.19	18.10.19 21:21	18.10.19 21:23	0:02
18.10.19	18.10.19 20:56	18.10.19 21:19	0:23
18.10.19	18.10.19 20:42	18.10.19 20:57	0:15
19.10.19	19.10.19 09:26	19.10.19 09:50	0:24
		Zeitsumme in Stunden:	**38:57**

Datenquelle
Abfrage: *Zeitdifferenzen_Formularvorlage* FIREBIRD: *Ansicht_Zeitdifferenzen_Form*

Dieser Bericht ist äußerlich gleich dem vorhergehenden aufgebaut. Nur ist hier die Differenz zwischen den Zeiten direkt in Stunden ausgedrückt.

Dem Bericht liegt statt der Abfrage «Zeitdifferenzen» die Abfrage «Zeitdifferenzen_Formular-vorlage» zu Grunde. In dieser Abfrage werden bereits die Zeiten berechnet, auch die Gesamt-zeiten gruppiert nach dem Namen. Dadurch ist für den Bericht keine Funktion und keine Formel notwendig.

Makros

Update

Aufruf aus
Formular: *Defaultdatum_Makro_SQL, Defaultdatum_Zeit_Makro_SQL*

```
001  SUB Update(oEvent AS OBJECT)
002     DIM oForm AS OBJECT
003     DIM oFeld AS OBJECT
004     DIM oFeldID AS OBJECT
005     DIM stTag AS STRING
006     DIM arList()
007     DIM loRow AS LONG
008     oFeld = oEvent.Source.Model
009     oForm = oFeld.Parent
010     loRow = oForm.getRow
```

```
011     stTag = oFeld.Tag
012     arList = Split(stTag, ";")
013     oFeldID = oForm.getByName(Trim(arList(1)))
014     IF NOT IsEmpty(oFeldID.currentValue) THEN
015        oDatenquelle = ThisComponent.Parent.CurrentController
016        If NOT (oDatenquelle.isConnected()) THEN
017           oDatenquelle.connect()
018        END IF
019        oVerbindung = oDatenquelle.ActiveConnection()
020        oSQL_Anweisung = oVerbindung.createStatement()
021        stSql = arList(0) + " WHERE ""ID"" = " + oFeldID.CurrentValue
022        oSQL_Anweisung.executeUpdate(stSql)
023        oForm.reload
024        oForm.absolute(loRow)
025     END IF
026  END SUB
```

Über den Ursprung des Auslösers wird der Button ermittelt: **`oEvent.Source.Model`**. Das Formular ist **`Parent`** zu dem Button (Zeile 9). In Zeile 10 wird die aktuelle Datensatznummer ausgelesen. Die Zusatzinformationen werden aus **`oFeld.Tag`** ausgelesen und in einem Array in die beiden Teile aufgeteilt (Zeile 11 und 12). Im zweiten Teil steht der Name des Feldes, in dem der Primärschlüssel steht. Mit **`Trim(arList(1))`** werden eventuelle Leerzeichen vor und hinter dem Eintrag entfernt (Zeile 13).

Ist das Primärschlüsselfeld leer, so wurde der Datensatz noch nicht abgespeichert. Es gibt also auch keine Änderung über den Button zu speichern. Nur wenn in dem Feld ein Wert steht, dann wird ein Kontakt zur Datenbank aufgebaut. Da aber das Formular bereits geöffnet ist müsste dieser Kontakt sowieso bestehen. Die entsprechende Nachfrage ist also nur eine Absicherung (Zeile 15 bis 18).

Schließlich wird der SQL-Code aus den Informationen in den Zusatzinformationen und den zusätzlichen Informationen zu dem Wert in dem Primärschlüsselfeld zusammengestellt (Zeile 21). Der SQL-Code wird an die Datenbank weitergegeben. Anschließend wird das Formular neu geladen und das Ergebnis über **`oForm.absolute(loRow)`** direkt sichtbar im Formular dargestellt.

Datum_aktuell

Aufruf aus
Formular: *Defaultdatum_Makro*

```
001  SUB Datum_aktuell
002     DIM oDoc AS OBJECT
003     DIM oDrawpage AS OBJECT
004     DIM oForm AS OBJECT
005     DIM oFeld AS OBJECT
006     DIM oFeldAender AS OBJECT
007     DIM unoDate
008     oDoc = thisComponent
009     oDrawpage = oDoc.drawpage
010     oForm = oDrawpage.forms.getByName("MainForm")
011     oFeld = oForm.getByName("datDatum")
012     oFeldAender = oForm.getByName("datDatumAender")
013     unoDate = createUnoStruct("com.sun.star.util.Date")
014     unoDate.Year = Year(Date)
015     unoDate.Month = Month(Date)
016     unoDate.Day = Day(Date)
017     IF isEmpty(oFeld.Date) THEN
018        oFeld.BoundField.updateDate(unoDate)
019     END IF
020     oFeldAender.BoundField.updateDate(unoDate)
021  END SUB
```

Der Zugriff auf die Felder im Formular wird über das Formulardokument erstellt (Zeile 8 bis 12). Anschließend wird das Datum als **Struct** zusammengebaut. In dieser Prozedur wird eine Methode gezeigt, wie die einzelnen Werte den verschiedenen Teilen des **Structs** für ein Datum zugeordnet werden. Ist das Feld mit dem Namen «datDatum» leer (Zeile 17), dann wird das aktuelle Datum über **oFeld.BoundField.updateDate()** in das Feld geschrieben. Ansonsten wird das aktuelle Datum nur in das Feld von «datDatumAender» geschrieben. Die endgültige Abspeicherung erfolgt über das Formular. Schließlich wurde das Makro nur direkt vor der Ausführung einer Datensatzaktion eingesetzt, und diese Aktion bedeutet in diesem Falle ein Schreiben oder Ändern, gegebenenfalls auch löschen eines Datensatzes.

Datum_Zeit_aktuell

Aufruf aus
Formular: *Defaultdatum_Zeit_Makro, Defaultdatum_Zeitdifferenz_Makro*
Makro: *UpdateTimestamp*

```
001 SUB Datum_Zeit_aktuell
002    DIM oDoc AS OBJECT
003    DIM oDrawpage AS OBJECT
004    DIM oForm AS OBJECT
005    DIM oFeld AS OBJECT
006    DIM oFeldAender AS OBJECT
007    DIM unoStmp
008    oDoc = thisComponent
009    oDrawpage = oDoc.drawpage
010    oForm = oDrawpage.forms.getByName("MainForm")
011    oFeld = oForm.getByName("fmtDatumZeit")
012    oFeldAender = oForm.getByName("fmtDatumZeitAender")
013    unoStmp = createUnoStruct("com.sun.star.util.DateTime")
014    WITH unoStmp
015       .Year = Year(Date)
016       .Month = Month(Date)
017       .Day = Day(Date)
018       .Hours = Hour(Time)
019       .Minutes = Minute(Time)
020       .Seconds = Second(Time)
021       .NanoSeconds = 000000
022    END WITH
023    IF isEmpty(oFeld.CurrentValue) THEN
024       oFeld.BoundField.updateTimestamp(unoStmp)
025    END IF
026    oFeldAender.BoundField.updateTimestamp(unoStmp)
027 END SUB
```

Das Makro für den Zeitstempel läuft vom Prinzip her gleich ab wie das Makro für das aktuelle Datum «Datum_aktuell». Hier wird lediglich ein anderes **Struct**, nämlich das **DateTime-Struct**, befüllt und der entsprechende Wert abgespeichert. Mit **WITH unoStmp** beginnt die Zusammensetzung des Structes (Zeile 14). **WITH** bedeutet hier nur, dass im Prinzip vor jedes der folgenden Elemente unoStmp gesetzt wird, bis eben **END WITH** auftaucht. Also **unoStmp.Year**, **unoStmp.Month** usw.

In dem Struct ist es auch möglich, Nanosekunden weiter zu geben. Die spielen hier aber keine Rolle. Das Struct wird also an dieser Position lediglich mit der entsprechenden Anzahl Nullen aufgefüllt.

Schließlich wird der erstelle Timestamp nur dann in das Feld «fmtDatumZeit» geschrieben, wenn das Feld leer ist. Ansonsten überschreibt der Timestamp nur den alten Timestamp im Feld «fmtDatumZeitAender».

Standarddatum

Aufruf aus
Formular: *Defaultdatum_Makro_Standarddatum*

```
001 SUB Standarddatum
002    DIM oDoc AS OBJECT
003    DIM oDrawpage AS OBJECT
004    DIM oForm AS OBJECT
005    DIM oFeld AS OBJECT
006    DIM unoDate
007    oDoc = thisComponent
008    oDrawpage = oDoc.drawpage
009    oForm = oDrawpage.forms.getByName("MainForm")
010    oFeld = oForm.getByName("datDatum")
011    unoDate = createUnoStruct("com.sun.star.util.Date")
012    unoDate.Year = Year(Date)
013    unoDate.Month = Month(Date)
014    unoDate.Day = Day(Date)
015    oFeld.setPropertyValue("DefaultDate", unoDate)
016 END SUB
```

Das Makro läuft ähnlich ab wie das Makro «Datum_aktuell». Nachdem allerdings das Struct mit dem aktuellen Datum befüllt ist wird jetzt das Standarddatum das Datumsfeldes damit gesetzt. Dies geschieht über den allgemeinen Zugriff auf bestimmte Eigenschaften, nämlich über **`setPropertyValue`**. Die Eigenschaft heißt beim Datumsfeld «DefaultDate». Der Wert, der hier gespeichert wird, muss vom Typ dem Struct entsprechen (Zeile `15`).

Standarddatum_verlegt

Aufruf aus
Formular: *Defaultdatum_Makro_Standarddatum_verlegt*

```
001 SUB Standarddatum_verlegt
002    DIM oDoc AS OBJECT
003    DIM oDrawpage AS OBJECT
004    DIM oForm AS OBJECT
005    DIM oFeld AS OBJECT
006    DIM unoDate
007    DIM daNew AS DATE
008    DIM i AS INTEGER
009    oDoc = thisComponent
010    oDrawpage = oDoc.drawpage
011    oForm = oDrawpage.forms.getByName("MainForm")
012    oFeld = oForm.getByName("datDatum")
013    unoDate = createUnoStruct("com.sun.star.util.Date")
014    i = -2
015    daNew = DateAdd("m", i, Date)
016    unoDate.Year = Year(daNew)
017    unoDate.Month = Month(daNew)
018    unoDate.Day = Day(daNew)
019    oFeld.setPropertyValue("DefaultDate", unoDate)
020 END SUB
```

Die Verlegung des Datums kam als Anfrage in einem Forum. Dort wurden häufig Daten bearbeitet, die die Eingabe eines Datums aus dem Vormonat erforderten. Dann musste in dem aufklappbaren Feld immer erst der Monat zurückgesetzt werden um dann den entsprechenden Datumswert zu setzen.

Wird das Defaultdatum stattdessen direkt auf den Vormonat festgelegt, so kann das Zurückbewegen zum Vormonat entfallen.

Das Makro hat erst einmal den gleichen Aufbau wie «Standarddatum». Für die Umrechnung des Datums wird die Funktion **DateAdd** genutzt. In diesem Fall der in Zeile 14 hinterlegte Wert mittels dieser Funktions zum Monatswert des aktuellen Datums addiert (Zeile 15). Die Funktion macht dann daraus einen gültigen Datumswert in der Zukunft (bei positiven Werte für **i**) oder in der Verangenheit (bei negativen Werten für **i**).

Schließlich wird der berechnet Wert wie beim Makro «Standarddatum» als vorgegebener Standardwert festgelegt.

StandardZeitstempel

Aufruf aus
Formular: *Defaultdatum_Makro_Standardzeitstempel*

```
001  SUB StandardZeitstempel
002     DIM oDoc AS OBJECT
003     DIM oDrawpage AS OBJECT
004     DIM oForm AS OBJECT
005     DIM oFeld AS OBJECT
006     DIM stStmp AS STRING
007     oDoc = thisComponent
008     oDrawpage = oDoc.drawpage
009     oForm = oDrawpage.forms.getByName("MainForm")
010     oFeld = oForm.getByName("fmtZeitstempel")
011     stStmp = Year(Date) & "-" & Month(Date) & "-" & Day(Date) & " " & Hour(Time) &
           ":" & Minute(Time) & ":" & Second(Time)
012     oFeld.setPropertyValue("EffectiveDefault", stStmp)
013  END SUB
```

Beim Zeitstempel muss der Wert anders festgelegt werden. Die Formularassistenten bilden häufig den Zeitstempel in zwei getrennten Feldern ab: zum einen ein Datumsfeld, zum anderen ein Zeitfeld. Hier wurde stattdessen ein formatiertes Feld gewählt. Dort ist der differenzierte Aufbau von Datum und Zeit zusammen nicht als Zeitstempelfeld in den Standardwert übertragbar. Stattdessen wird für den Zeitstempel ein Text mit dem entsprechenden Format erstellt (Zeile 11). Der Aufbau von stStmp ist hier mit der für Datenbanken üblichen Datumsschreibweise versehen: Zuerst die vierstellige Jahreszahl, gefolgt von '-', dann die zweistellige Monatszahl und wieder '-' und schließlich die zweistellige Tageszahl. Zeit und Datum werden durch eine Leertaste getrennt. Als Verbindungselement der Variablen muss hier das «&» gewählt werden. Wird stattdessen, wie sonst bei Texten gewohnt, ein «+» gewählt, dann werden stattdessen die verschiedenen Werte einfach addiert.

Der Standardwert wird hier über **EffectiveDefault** eingefügt (Zeile 12). Zu den entsprechenden Standardwerten siehe unter anderen das Base-Handbuch.

UpdateTimestamp

Aufruf aus
Formular: *Defaultdatum_Zeitdifferenz_Makro*

Benötigt
Makro: *Datum_Zeit_aktuell*

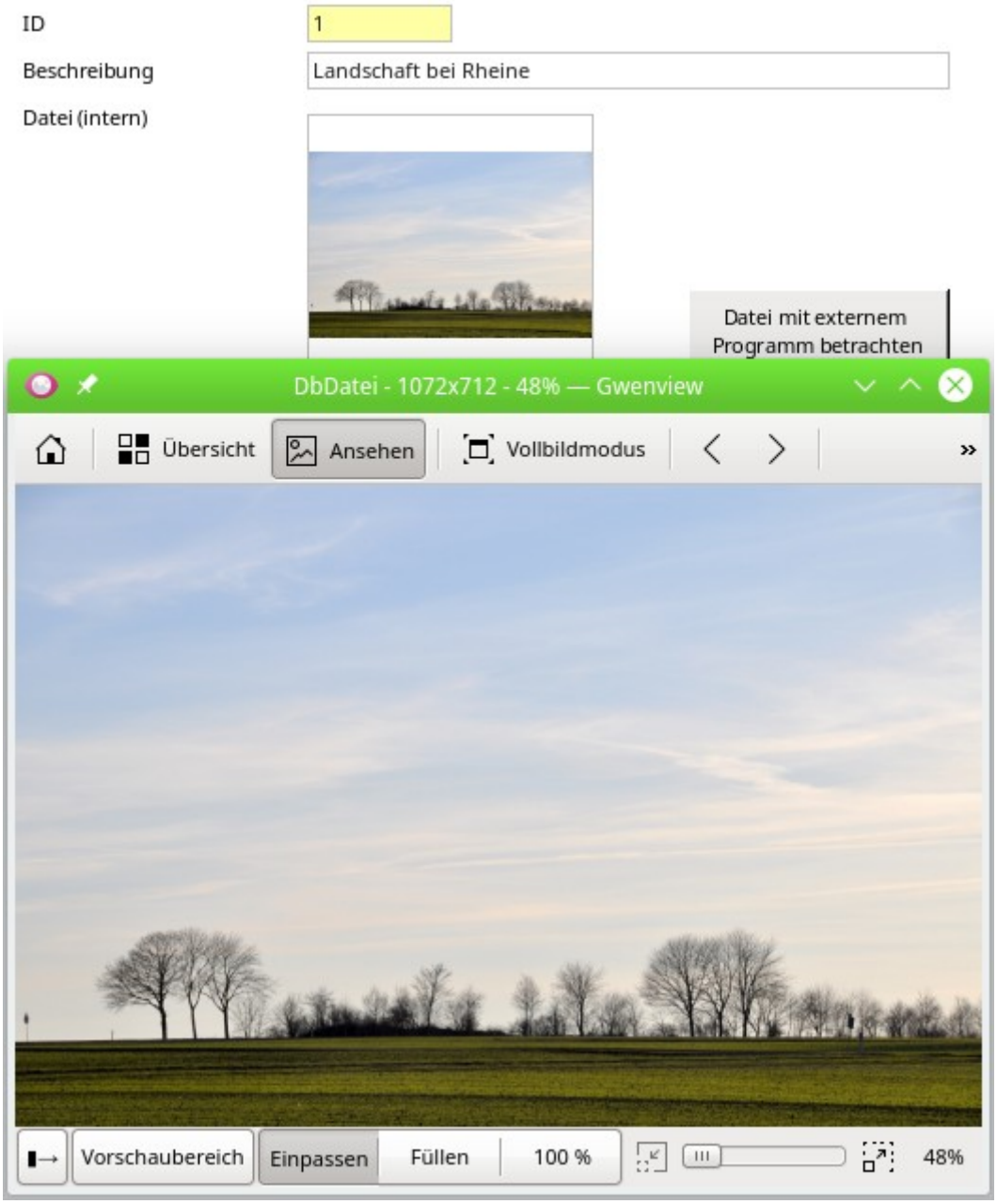
```
001  SUB UpdateTimestamp(oEvent AS OBJECT)
002     DIM oForm AS OBJECT
003     DIM oFeld AS OBJECT
004     DIM loRow AS LONG
005     Datum_Zeit_aktuell
006     oFeld = oEvent.Source.Model
007     oForm = oFeld.Parent
```

```
008     loRow = oForm.getRow
009     IF oForm.IsNew THEN
010        oForm.insertRow
011     ELSE
012        oForm.updateRow
013     END IF
014     oForm.reload
015     oForm.absolute(loRow)
016  END SUB
```

Das Makro «UpdateTimestamp» greift direkt auf das Objekt zu, durch das es ausgelöst wurde: **`oEvent AS OBJECT`**. Dadurch wird im Folgenden das Formular ermittelt, das mit einer (neuen) Endzeit versehen wird (Zeile 6 und 7). Existiert noch keine Zeiteingabe, so wird die Zeit neu geschrieben. Das Makro «Datum_Zeit_aktuell» wird dafür aus diesem Makro heraus zuerst einmal gestartet (Zeile 5).

Mit **`oForm.IsNew`** wird abgesichert, ob ein neuer Datensatz eingefügt (**`insertRow`**) oder ein bestehender Datensatz geändert (**`updateRow`**) werden muss. Anschließend wird der aktuell sichtbare Datensatz neu in das Formular eingelesen. Dazu wurde in Zeile 8 die aktuelle Datensatznummer ausgelesen. In Zeile 15 wird dann nach einem Neuladen des Formulars diese Datensatznummer Damit wird entsprechend auch die Berechnung sichtbar durchgeführt.

Dateien einbinden



Dass Bilder direkt in Base in eine Tabelle eingelesen werden können ist meist noch bekannt. Allerdings funktioniert die Verwaltung von Bildern in separaten Verzeichnissen genau so gut. Zusätzlich zur Bildaufnahme soll hier auch gezeigt werden, wie die Vorschau auf die Bilder an den Viewer des Betriebssystem übergeben werden kann, so dass nicht nur kleine Bildchen zu sehen sind und wie letztlich auch Bilder aus der Datenbank wieder nach außerhalb der Datenbank transportiert werden können.[3]

Inzwischen hat sich in Base an dieser Stelle seit der Version LO 5.0 etwas getan, so dass nicht nur Bilder mit dem Grafischen Steuerelement aufgenommen werden können. Der Dialog wurde auf alle Dateien erweitert. Von PDF-Dateien liefert das Grafische Steuerelement dann eine Vorschau auf die erste Seite des Dokumentes.

3 Beispieldatenbank Beispiel_Dateien_einbinden.odb

Tabellen

Die Tabellen haben in dieser Datenbank die Funktion, entweder Dateien in den Tabellen zu speichern oder nur den Pfad zu Dateien aufzunehmen.

Dateien

Datenziel
Formular: *Pfadeingabe, Pfadeingabe_Tabellenkontrollfeld*
Ansicht: *Ansicht_Dateiendung*
Abfrage, Bericht, Makro: keine direkt

Feldname	***Feldtyp***	***Beschreibung***
ID	Integer	Die ID ist Primärschlüssel dieser Tabelle. Das Feld ist als Auto-Wert-Feld gesetzt.
Beschreibung	Text	Soll einen Text aufnehmen, der als Dateibeschreibung dienen kann Eingabe erforderlich: Ja
Pfad	Text	Nimmt den Pfad zu der Datei auf. Eingabe erforderlich: Ja

In dieser Tabelle werden nur die Pfade zu den Dateien gespeichert. Soll die Datenbankdatei auf einem anderen Rechner funktionieren, so müssen die Dateien in der gleichen Pfadstruktur vorhanden sein. Die Dateien werden daher am besten in separaten Pfaden unterhalb der Datenbankdatei gelagert.

Dateien_intern

Datenziel
Formular: *Dateiaufnahme, Dateiaufnahme_Name*
Ansicht, Abfrage, Bericht, Makro: keine direkt

Feldname	***Feldtyp***	***Beschreibung***
ID	Integer	Die ID ist Primärschlüssel dieser Tabelle. Das Feld ist als Auto-Wert-Feld gesetzt.
Beschreibung	Text	Soll einen Text aufnehmen, der als Dateibeschreibung dienen kann. Eingabe erforderlich: Nein
Datei	Bild	Nimmt die Datei als Binärdatenstrom auf. Eingabe erforderlich: Ja
DateiName	Text	Sollte gegebenenfalls den Namen der Datei mit Dateiendung speichern. Eingabe erforderlich: Nein

Die Tabelle "Dateien_intern" steht in keiner Beziehung zur Tabelle "Dateien". Sie soll die interne Speichermöglichkeit von Dateien und den Umgang mit diesen Dateien aufzeigen.

Auch wenn im Tabelleneditor für das Feld des Typs **`LONGVARBINARY`** in der Übersetzung als Feldtyp **`Bild`** angegeben wird, so ist dieser Feldtyp sehr wohl in der Lage, alle möglichen Binärdaten aufzunehmen. Dies trifft auch z.B. auf Writer-Dateien zu.

Bei der Migration von HSQLDB nach FIREBIRD wird aus dem Feldtyp für die "Datei" ein Feldtyp **Bild [BLOB]** erstellt. Dieser Feldtyp 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 Tabelle wird geändert und die gesamte Tabelle wird kopiert. Beim Einfügen wird die Tabelle «Dateien_intern» neu erstellt. Jetzt wird beim Feldtyp für "Datei" der Datentyp **Blob [BLOB]** gewählt. Dieser Feldtyp zeigt Bilder wieder korrekt an.

Ansichten

Diese Datenbank hat nur eine Ansicht, die Ansicht_Dateiendung

Datenquelle
Tabelle: *Dateien*

Datenziel
Abfrage: *Bilder*

```
001 SELECT  "Dateien"."ID","Dateien"."Beschreibung","Dateien"."Pfad" ,
002 LOWER ( SUBSTRING ( RIGHT( "Pfad", 5 )
        FROM POSITION( '.' IN RIGHT( "Pfad", 5 ) ) + 1 ) ) AS "D_Endung"
003 FROM "Dateien"
```

Die Ansicht dient lediglich dazu, aus den Pfadbezeichnungen für die Dateien die Dateiendungen zu separieren. Damit der Code einfacher zu handhaben ist, ist dies nicht direkt in die Abfrage mit eingebaut worden.

In Zeile 2 wird zunächst nach den letzten 5 Zeichen der Pfadbezeichnung gesucht. In diesen 5 Zeichen muss die Dateiendung für Bilder stecken. Die Dateiendung ist durch einen **'.'** abgetrennt. Die Position dieses Punktes dient dazu, die Dateiendung schließlich ohne den Pfad zu lesen. Die Endung wird hier durch **LOWER** immer klein geschrieben wiedergegeben, so dass in der darauf zugreifenden Abfrage die gesuchten Dateiendungen nur klein geschrieben werden müssen.

Ein entsprechender Code ist auch für die Aufnahme der Dateien in die Datenbank selbst möglich, wenn dort der Dateiname mit abgespeichert wird.

Abfragen

Diese Datenbank hat nur eine Abfrage, die Abfrage Bilder

Datenquelle
Ansicht: *Ansicht_Dateiendung*

Datenziel
Bericht: *Bericht_Bilder*

```
001 SELECT * FROM "Ansicht_Dateiendung"
002 WHERE "D_Endung" IN ( 'jpg', 'jpeg', 'png', 'pdf' )
```

Die Abfrage filtert aus den Dateien diejenigen heraus, deren Endung den in Zeile 2 aufgeführten Endungen entspricht. Dies ist für den Bericht notwendig, der sonst bei anderen Dateien als Bilddateien und PDF-Dateien nur eine Fehlermeldung anzeigt. Dies Liste in Zeile 2 kann natürlich wesentlich alle Endungen enthalten, die auch in dem Dialog zur Aufnahme von Bildern enthalten sind.

Formulare

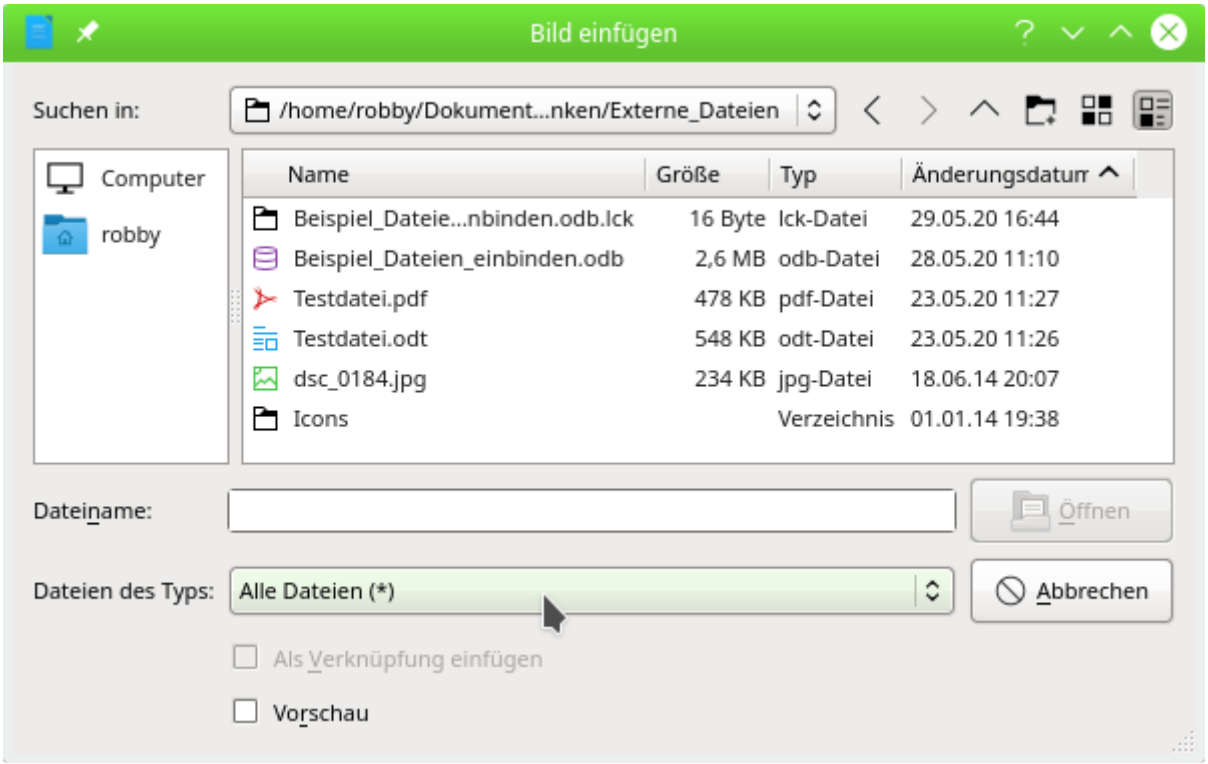
Die Formulare sind notwendiger Bestandteil zur Aufnahme von Bildern in die Datenbank. Die Bildaufnahme geht nicht über die Tabellensicht oder Sicht einer Abfrage. Es wäre höchstens möglich, eine Aufnahme per Makro an den Formularen vorbei zu realisieren.

Auch zur Betrachtung der Bilder sind wieder Formulare erforderlich. Lediglich ein Bericht kann neben den Formularen noch Bilder in der Base-Datei darstellen.

Pfadeingabe

1	Formular (Tabelle: *Dateien*)

Makros in Feldeigenschaften		
1	Schaltfläche Datei mit externem Programm betrachten (Aktion ausführen)	Module1.*Betrachten*

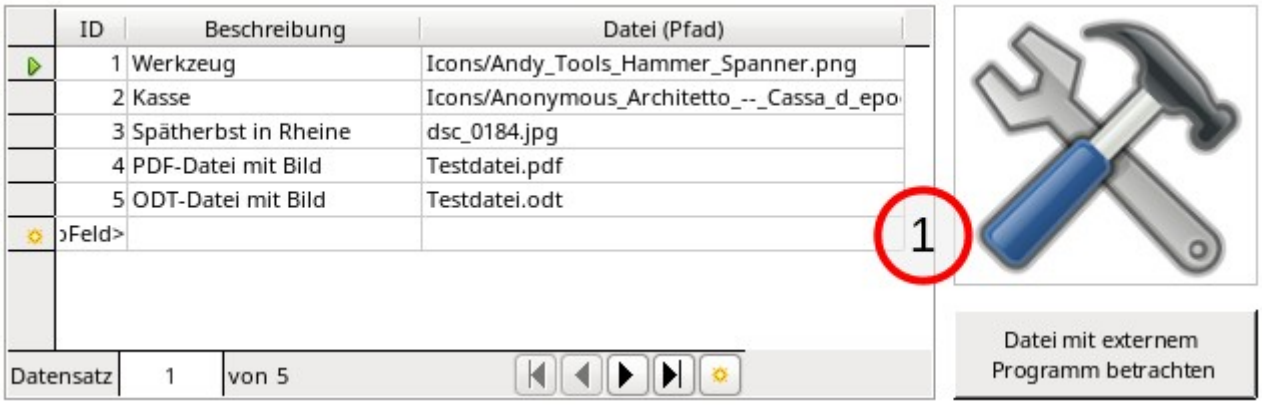


Das Formular enthält neben den Eingabefeldern für «ID» und «Beschreibung» ein grafisches Steuerelement. Das grafische Steuerelement ist mit dem Feld "Pfad" aus der Tabelle verbunden. Wird durch Doppelklick auf das Feld eine Datei aus dem Dateisystem ausgesucht, so wird nur der Pfad (relativ zur Datenbankdatei) gespeichert. Die Datei wird trotzdem in dem grafischen Steuerelement angezeigt, wenn sie anzeigbar ist. Dies gilt neben normalen Bilddateien auch für die erste Seite von PDF-Dateien.

Das Feld für den Primärschlüssel «ID» ist schreibgeschützt, da es über **Auto-Wert** gefüllt wird. Deshalb auch der gelbe Hintergrund.

Das grafische Steuerelement bietet nicht die Möglichkeit, Details des Bildes zu betrachten. Deshalb wird hier über Datei mit externem Programm betrachten der im System angegebenen Dateibetrachter für die entsprechende Datei gestartet. Jetzt können Details betrachtet und alle weiteren Funktionen des Viewers bzw. verknüpften Programms genutzt werden. Bei einer Writer-Datei wird z.B. direkt der Writer mit der datei gestartet und die Datei könnte auch bearbeitet werden.

Pfadeingabe_Tabellenkontrollfeld



1	Formular (Tabelle: *Dateien*)

Makros in Feldeigenschaften		
1	Schaltfläche Datei mit externem Programm betrachten (Aktion ausführen)	Module1.*Betrachten*

Wird statt der einzelnen Kontrollfelder ein Tabellenkontrollfeld genutzt, so kann das Bild nicht innerhalb des Tabellenkontrollfeldes dargestellt werden. Stattdessen wird hier der Pfad (relativ zur Datenbankdatei) direkt angezeigt. Das grafische Steuerelement ist hier direkt neben dem Tabellenkontrollfeld im selben Formular angeordnet. Es zeigt das Bild zu dem aktuellen markierten Datensatz an.

Die Felder «ID» und «Datei (Pfad)» sind schreibgeschützt. Leider erlaubt ein Tabellenkontrollfeld nicht, solche Felder mit einer separaten Formatierung (gelber Hintergrund wie im vorhergehenden Formular) zu kennzeichnen.

Dateiaufnahme

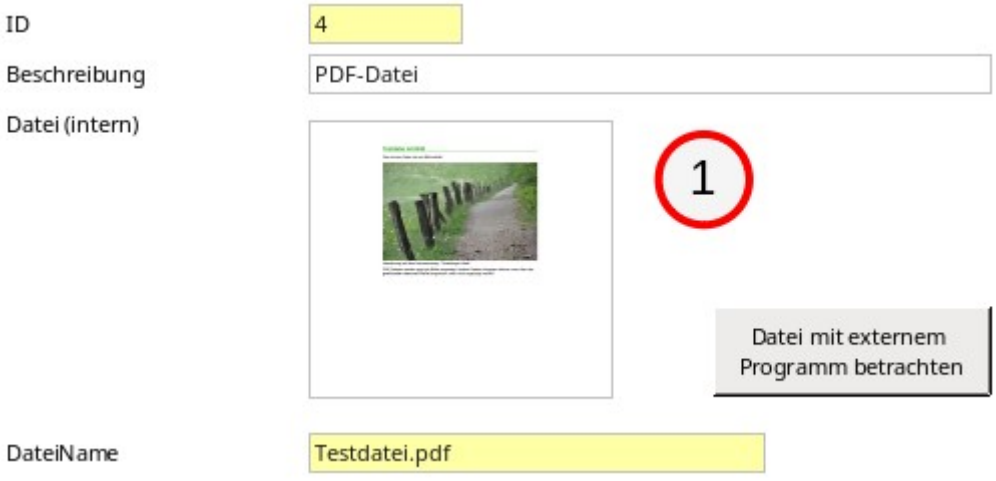
1	Formular (Tabelle: *Dateien_intern*)

Makros in Feldeigenschaften		
1	Schaltfläche Datei mit externem Programm betrachten (Aktion ausführen)	Module1.*DateiAuslesen*

Äußerlich unterscheidet sich dieses Formular nicht vom Formular «Pfadeingabe». Allerdings wird hier die Datei in die Tabelle selbst eingelesen. Eine Verknüpfung noch außen wird nicht gespeichert.

An dieser Stelle sollte aber darauf hingewiesen werden, dass das Abspeichern von Bilddaten sehr schnell eine Datenbank von der Größe her deutlich aufbläht. Deshalb sollten intern möglichst nur kleinere Bilder, z.B. Icons, aber nicht komplette Fotos aktueller Digitalkameras abgespeichert werde. Das Abspeichern von normalen Textdateien dagegen ist kein Problem.

Dateiaufnahme_Name



1	Formular (Tabelle: *Dateien_intern*)

Makros in Formulareigenschaften		
1	Vor der Datensatzaktion	Module1.*DateiName*

Makros in Feldeigenschaften		
1	Schaltfläche Datei mit externem Programm betrachten (Aktion ausführen)	Module1.*DateiAuslesen_mitName*
1	Graphisches Feld «Datei (intern)» (Bei Fokusverlust)	Module1.*DateiName*

Das Formular «Dateiaufnahme_Name» hat lediglich ein zusätzliches Feld, in dem der Name der Datei abgespeichert werden kann, also z.B. «Testdatei.pdf». Dies könnte für Betriebssysteme, die nach der Dateiendung entsprechende Programme für das Öffnen von Dateien bereitstellen, wichtig sein, da der eigentliche Bildname nach dem Speichern der Bilddaten in der Datenbank nicht mehr nachvollzogen werden kann.

Der Name wird hier automatisch über ein Makro beim Verlassen des Grafischen Kontrollfeldes oder beim Speichern des Datensatzes ausgelesen. Das Feld ist, wie an der gelben Farbe vom Design her zu sehen, ebenso wie das Feld «ID» schreibgeschützt.

Berichte

Diese Datenbank hat nur einen Bericht, den Bericht Bericht_Bilder

ID 1
Beschreibung Werkzeug
Pfad Icons/Andy_Tools_Hammer_Spanner.png

ID 2
Beschreibung Kasse
Pfad Icons/Anonymous_Architetto_--_Cassa_d_epoca.png

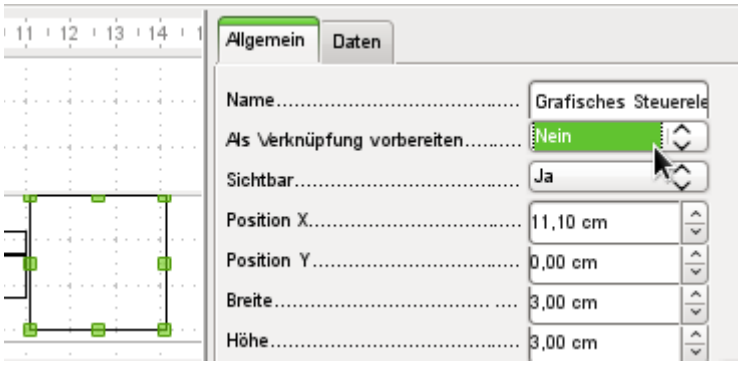
ID 3
Beschreibung Spätherbst in Rheine
Pfad dsc_0184.jpg

ID 4
Beschreibung PDF-Datei mit Bild
Pfad Testdatei.pdf

Datenquelle
Abfrage: *Bilder*

Die Bilder werden, wie im Formular, in einem grafischen Kontrollfeld dargestellt. Das Kontrollfeld ist so voreingestellt, dass das Seitenverhältnis beibehalten wird. Sonst könnte es dazu kommen, dass Bilder nicht mehr zu erkennen sind, weil nur ein Ausschnitt von ihnen gezeigt oder das Bild verzerrt in den vorgesehenen Rahmen eingepasst wird.

Mit den Standardeinstellungen kann es dazu kommen, dass eine Datei, die nicht im grafischen Kontrollfeld angezeigt werden kann, dafür sorgt, dass kein einziges Bild erscheint. Hier kann auf zweierlei Weise gegengesteuert werden:

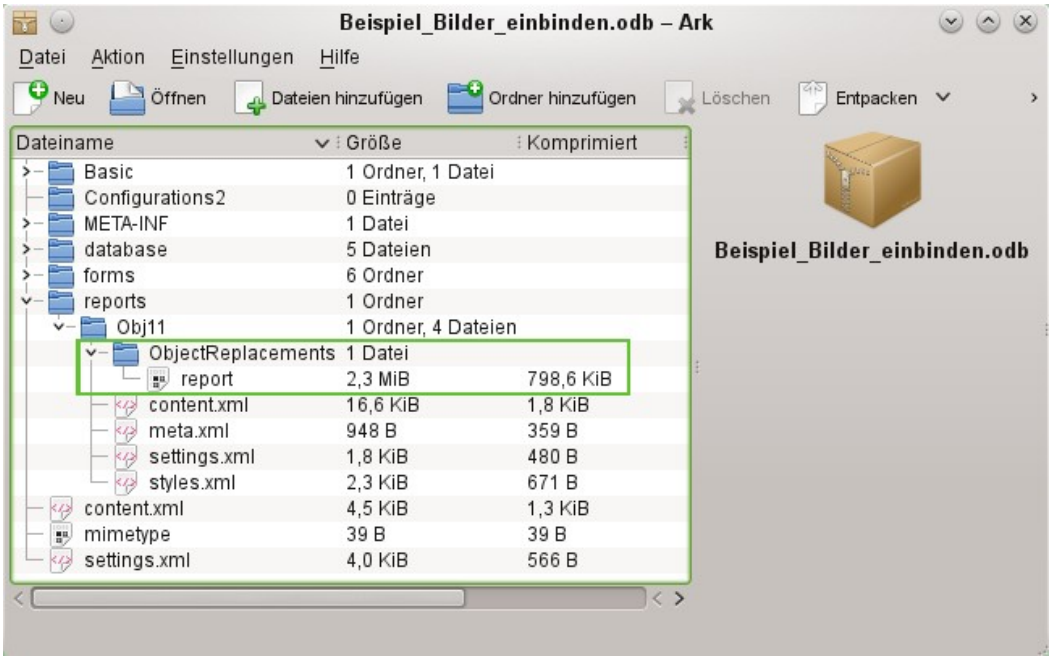


Der Bericht wird zum Editieren geöffnet. Das grafische Steuerelement wird markiert. In **Eigenschaften → Allgemein → Als Verknüpfung vorbereiten** wird «Nein» ausgewählt. Alle Bilder erscheinen dann problemlos. Bei anderen Dateien wie z.B. einer Writer-Datei erscheint dann ein kleines Rechteck mit einer Meldung, dass diese Datei nicht geladen werden konnte.

Bei dem obigen Bericht ist allerdings schon vorher darauf geachtet worden, dass Datensätze mit Dateien, die nicht dargestellt werden können, gar nicht erst übernommen werden. Dies wurde über die Abfrage «Bilder» geregelt.

Hinweis

Nach dem erneuten Editieren von Berichten mit Bildern fällt auf, dass die Datenbankdatei deutlich größer wird. Bei den Tests zu dieser Beschreibung mit großen Bilddateien entstand plötzlich eine Datei der Größe 16 MB. 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.
Wenn Berichte nicht wiederholt editiert werden, reicht ein einmaliges Löschen des Verzeichnisses aus.
Der Bug ist hier gemeldet: *https://bugs.freedesktop.org/show_bug.cgi?id=80320*

Makros

Betrachten

Aufruf aus
Formular: *Pfadeingabe, Pfadeingabe_Tabellenkontrollfeld*

```
001 SUB Betrachten
002    DIM oDoc AS OBJECT
003    DIM oDrawpage AS OBJECT
004    DIM oForm AS OBJECT
005    DIM oFeld AS OBJECT
006    DIM oShell AS OBJECT
007    DIM stUrl AS STRING
008    DIM stFeld AS STRING
009    DIM arUrl_Start()
010    oDoc = thisComponent
011    oDrawpage = oDoc.Drawpage
012    oForm = oDrawpage.Forms.getByName("Formular")
013    oFeld = oForm.getByName("GraphischesFeld")
014    stUrl = oFeld.BoundField.getString
015    arUrl_Start = split(oDoc.Parent.Url,oDoc.Parent.Title)
016    oShell = createUnoService("com.sun.star.system.SystemShellExecute")
017    stFeld = convertToUrl(arUrl_Start(0) + stUrl)
018    oShell.execute(stFeld,,0)
019 END SUB
```

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 benutzt wird. Die Auftrennung gibt so nur als erstes und einziges Element des Arrays den Pfad zur *.odb-Datei wieder (Zeile 17).

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 (Zeile 20). 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.

DateiAuslesen

Aufruf aus
Formular: *Dateiaufnahme*

```
001 SUB DateiAuslesen
002    DIM oDoc AS OBJECT
003    DIM oDrawpage AS OBJECT
```

```
004     DIM oForm AS OBJECT
005     DIM oFeld AS OBJECT
006     DIM oStream AS OBJECT
007     DIM oShell AS OBJECT
008     DIM oPath AS OBJECT
009     DIM oSimpleFileAccess AS OBJECT
010     DIM st AS STRING
011     DIM stPfad AS STRING
012     DIM stFeld AS STRING
013     oDoc = thisComponent
014     oDrawpage = oDoc.Drawpage
015     oForm = oDrawpage.Forms.getByName("Formular")
016     oFeld = oForm.getByName("GraphischesFeld")
017     oStream = oFeld.BoundField.getBinaryStream
018     oPath = createUnoService("com.sun.star.util.PathSettings")
019     st = ""
020     stPfad = oPath.Temp & "/DbDatei" & st
021     oSimpleFileAccess = createUnoService("com.sun.star.ucb.SimpleFileAccess")
022     oSimpleFileAccess.writeFile(stPfad, oStream)
023     oShell = createUnoService("com.sun.star.system.SystemShellExecute")
024     stFeld = convertToUrl(stPfad)
025     oShell.execute(stFeld,,0)
026 END SUB
```

Der wichtigste Unterschied zu der externen Bildverwaltung ist gleichzeitig auch die entscheidende Frage für den Datenbanknutzer: Wie komme ich eigentlich an die Bilddaten wieder heran, die jetzt in der Tabelle der Datenbank gespeichert liegen. Mit einem externen Viewer kann ich schließlich nicht interne Bilder ansehen. Die internen Bilder müssen dafür zumindest vorübergehend ausgelesen werden um damit über ein Betrachtungsprogramm drauf zugreifen zu können.

Mit **getBinaryStream** wird aus dem mit dem graphischen Kontrollfeld verbundenen Feld der Datenbanktabelle der Binärcode ausgelesen, der zu dem Bild gehört (Zeile 17).

Es kann, je nach Betriebssystem, notwendig sein, eine Dateiendung dem jeweils gewählten Namen beizufügen. In der Variablen «st» kann so eine Endung aufgeführt werden (z. B. ".png" oder ".jpg"). Aus den Pfadangaben in **Extras → Optionen → LibreOffice → Pfade** wird der Pfad für temporäre Dateien ausgelesen. Dorthin soll das Bild gespeichert werden (Zeile 18 bis 20).

Mit dem Struct **com.sun.star.ucb.SimpleFileAccess** wird die ausgelesene Datei in das temporäre Verzeichnis geschrieben. Der weitere Verlauf des Makros ist wieder identisch mit der Prozedur «Betrachten» für die externen Dateien.

DateiAuslesen_mitName

Aufruf aus
Formular: *Dateiaufnahme_Name*

In diesem Makro wird lediglich statt eines fest vorgegebenen Dateinamens die Möglichkeit geboten, einen Dateinamen mit Endung aus der Datenbank auszulesen. Hier nur die Unterschiede zur Prozedur «DateiAuslesen»

```
001 SUB DateiAuslesen_mitName
002     ...
003     oFeld2 = oForm.getByName("DateiName")
004     stName = oFeld2.Text
005     IF stName = "" THEN
006        stName = "DbDatei"
007     END IF
008     ...
009     stPfad = oPath.Temp & "/" & stName
010     ...
011 END SUB
```

Steht in dem Formularfeld "DateiName" eine Bezeichnung für die zu speichernde Datei, so wird das Bild unter dem entsprechenden Namen abgespeichert (Zeile 4). Damit besteht die Möglichkeit, der erzeugten Datei eine Endung mitzugeben und außerdem auch noch direkt nacheinander mehrere Bilder abzuspeichern, ohne die vorhergehenden zu überschreiben. Die gespeicherten Bilder können anschließend aus dem temporären Verzeichnis ist das entsprechende Wunschverzeichnis übertragen werden.

DateiName

Aufruf aus
Formular: *Dateiaufnahme_Name*

```
001 SUB DateiName
002    DIM oDoc AS OBJECT
003    DIM oDrawpage AS OBJECT
004    DIM oForm AS OBJECT
005    DIM oField AS OBJECT
006    DIM stImageUrl AS STRING
007    DIM stFile AS STRING
008    GlobalScope.BasicLibraries.LoadLibrary("Tools")
009    oDoc = thisComponent
010    oDrawpage = oDoc.Drawpage
011    oForm = oDrawpage.Forms.getByName("Formular")
012    oField = oForm.getByName("GraphischesFeld")
013    stImageUrl = oField.ImageUrl
014    IF stImageUrl <> "" THEN
015       stFile = FileNameoutofPath(stImageUrl)
016       oForm.UpdateString(oForm.FindColumn("DateiName"), stFile)
017    END IF
018 END SUB
```

Bei der Aufnahme der Datei über das Grafische Kontrollfeld wird die URL benötigt. Diese URL kann nur zu diesem Zeitpunkt ausgelesen werden. Dies erfolgt in Zeile 13. Ist diese URL nicht leer, d.h. wenn das Makro ausgelöst wurde, obwohl gerade keine neue Datei eingelesen wurde, so kann auch kein neuer Name ausgelesen werden. Ansonsten wird über die Funktion **FileNameoutofPath** (Zeile 15) der Dateiname bestimmt. Diese Funktion steht in den Standardbibliotheken zur Verfügung. Diese Bibliotheken müssen zuerst eingebunden werden (Zeile 8) bevor die Funktion nutzbar ist.

Die Verzweigung ist nur deshalb notwendig, weil das grafische Kontrollfeld kein Ereignis «Modifiziert» enthält. Deswegen wird das Makro bei jeder Datensatzänderung des Formulars aufgerufen.

Serienbrief direkt

Einführung

Für Serienbriefe wird in der Regel zuerst einmal eine Writer-Datei gestartet und dann von dort aus der Seriendruck in die Wege geleitet. Die folgende Variante soll aufzeigen, wie so etwas auf verschiedene Art auch direkt aus einem Base-Formular heraus erfolgen kann.

Die Dokumente zum Serienbrief sind in der vorliegenden Fassung nur zu gebrauchen, wenn

- die Datenbank in LO angemeldet ist unter dem Namen "Beispiel_Datenbank_Serienbrief_direkt"[4]
- alle Dateien in einem gemeinsamen Verzeichnis liegen
- die Makrofunktionalität ermöglicht wird.

Es wird über ein Makro die Serienbrieffunktionalität angesprochen. In der Datenbank befinden sich zwei Makros, die entsprechend an die Serienbriefe angepasst wurden. So etwas lässt sich mit Variablen auch in einem Makro zusammen fassen, sollte hier aber nicht zusätzlich verwirren.

Über die Makros wird ein Brief erzeugt und direkt in dem Verzeichnis abgespeichert, in dem sich auch der Serienbrief und die Datenbank befindet. Mehrmaliges Erstellen des Briefes führt zu mehreren durchnummerierten Briefen mit dem Feld "Nachname" als Dateibeginn.

Die Rechnungserstellung hat so noch einen Schönheitsfehler:

Der Serienbrief arbeitet mit Tabulatoren. Wird die Beschreibung des Gegenstandes zu groß, so verschiebt sich entsprechend der folgende Inhalt. Hier muss gegebenenfalls nachgearbeitet werden.

Bei den Textfeldern ist deshalb ein anderes Verfahren angewandt worden. Die Textfelder werden gefüllt. Gleichzeitig befindet sich in der Vorlage eine einzeilige Tabelle. Dieser Tabelle werden jeweils Datenzeilen hinzugefügt und anschließend wieder eine neue Tabellenzeile eingefügt.

Tabellen

Anschrift

Datenziel	
Formular:	*Anschrift, Rechnung, Rechnung_Textfelder, Rechnung_Textfelder_Uebertrag, Rechnung_Textfelder_Uebertrag_Seriendruck*
Abfrage:	*Adresseingabe, Rechnung_einzeilig, Fehler: Verweis nicht gefunden, Rechnung_Textfelder*

Feldname	***Feldtyp***	***Beschreibung***
ID	Integer	Die ID ist Primärschlüssel dieser Tabelle. Sie ist als Autowert festgelegt.
Vorname	Text	Eingabe erforderlich: Nein
Nachname	Text	Eingabe erforderlich: Ja
Geschlecht	Text	Länge des Textes ist 1 Zeichen. Hier wird nur 'm' oder 'w' eingetragen, damit später daraus eine Anrede ermittelt werden kann. Eingabe erforderlich: Nein
Straße_Nr	Text	Eingabe erforderlich: Nein

4 Beispieldatenbank Beispiel_Datenbank_Serienbrief_direkt.odb

Postleitzahl	Text	Länge des Textes ist 5 Zeichen. Postleitzahlen sollten nicht als Zahlen abgespeichert werden. Führende Nullen sind sonst nicht möglich. Eingabe erforderlich: Nein
Ort	Text	Eingabe erforderlich: Nein

Die Tabelle "Anschrift" zeigt das typische Anwendungsbeispiel eines Serienbriefes. Hier werden beispielhaft lediglich die Adressdaten in ein Druckdokument übertragen, das natürlich mit entsprechenden Feldern beliebig erweitert werden könnte.

Die weiteren Tabellen dienen dazu, auch den Druck einer Rechnung direkt aus der Datenbank heraus aufzuzeigen. Eine Rechnung unterscheidet sich in sofern von einem einfachen Serienbrief, da hier zu einem Datensatz (z.B. der Anschrift) viele Rechnungstitel gehören. Im Report-Builder wird so etwas durch Gruppierungen gelöst. Hier ist der eingeschlagene Weg etwas anders angefasst worden.

Rechnung

Datenziel	
Formular:	*Rechnung, Rechnung_Textfelder, Rechnung_Textfelder_Uebertrag*
Makro:	*PDF_Export*
Abfrage:	*Rechnung_einzeilig, Rechnung_nicht_gedruckt, Rechnung_Textfelder*

Feldname	***Feldtyp***	***Beschreibung***
ID	Integer	Die ID ist Primärschlüssel dieser Tabelle. Sie ist als Autowert festgelegt. Der Primärschlüssel ist gleichzeitig die spätere Rechnungsnummer.
Datum	Datum	Hier wird das Rechnungsdatum festgelegt. Eingabe erforderlich: Ja
Anschrift_ID	Integer	Hier wird die "ID" der Tabelle "Anschrift" als Fremdschlüssel gespeichert. Grundsätzlich kann so eine Person mehrere Rechnungen erhalten ohne jedes Mal die gesamte Anschrift erneuern zu müssen. Eingabe erforderlich: Nein
Druckdatum	Datum	Beim Seriendruck von Rechnungen wird hier das Datum der Übertragung des Rechnungsinhaltes in ein *.pdf-Dokument gespeichert. Eingabe erforderlich: Nein

Die Tabelle Rechnungsinhalt wird nur für die Durchführung des Rechnungsdruckes benötigt. Sie wird durch ein Makro gefüllt und nicht irgendwie durch Formulare bearbeitet.

Rechnungsinhalt

Datenziel	
Abfrage:	*Rechnung_einzeilig*
Makro:	*Textfelder_Fuellen, Rechnungsinhalt_zusammenstellen, Tabellen_fuellen_Uebertrag*
Formular:	keine direkt

Feldname	Feldtyp	Beschreibung
ID	Integer	Die ID ist Primärschlüssel dieser Tabelle. Die ID ist identisch mit der der Tabelle "Rechnung". Eine Beziehung wird hier aber nicht festgelegt.
Rechnungsinhalt	Text	Länge des Textes sollte groß gewählt werden, da hier für eine der Serienbrieffunktionen alle Rechnungsinhalte in einem Datensatz gespeichert werden. Im Beispiel ist der Länge auf 8000 Zeichen festgelegt (momentanes Maximum FIREBIRD). Eingabe erforderlich: Ja

Verkauf

Datenziel	
Formular:	*Rechnung, Rechnung_Textfelder, Rechnung_Textfelder_Uebertrag, Rechnung_Textfelder_Uebertrag_Seriendruck*
Abfrage:	*Rechnung_Textfelder*
Makro:	*Rechnungsinhalt_zusammenstellen, Rechnungsinhalt_zusammenstellen_Tabelle, Rechnungsinhalt_zusammenstellen_Tabelle_Uebertrag, Rechnungsinhalt_zusammenstellen_Tabelle_Uebertrag_Direktdruck*

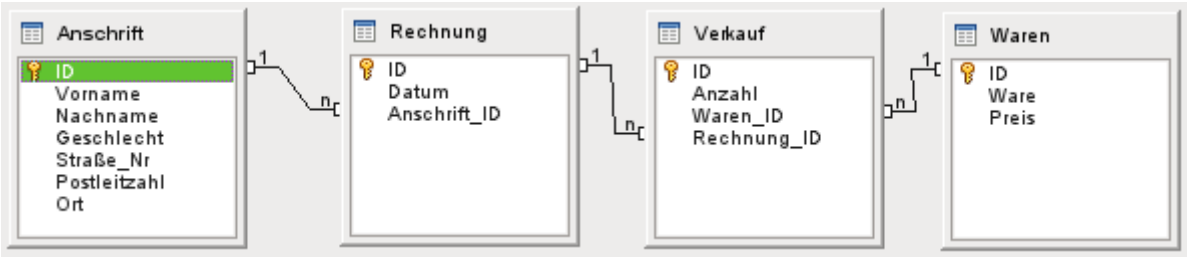
Feldname	Feldtyp	Beschreibung
ID	Integer	Die ID ist Primärschlüssel dieser Tabelle. Sie ist als Autowert festgelegt.
Anzahl	Tiny Integer	Je größer die Anzahl der Waren sein sollte, desto größer ist natürlich der Umfang dieses Feldes festzulegen. Mit Tiny Integer geht die Anzahl von -128 bis +127 – für ein einfaches Beispiel völlig ausreichend. Eingabe erforderlich: Ja
Waren_ID	Integer	Hier wird die "ID" der Tabelle "Waren" als Fremdschlüssel gespeichert. Eingabe erforderlich: Ja
Rechnung_ID	Integer	Hier wird die "ID" der Tabelle "Rechnung" als Fremdschlüssel gespeichert. Eingabe erforderlich: Ja

Waren

Datenziel	
Formular:	*Rechnung, Rechnung_Textfelder, Rechnung_Textfelder_Uebertrag, Rechnung_Textfelder_Uebertrag_Seriendruck, Waren*
Abfrage:	*Etiketten, Rechnung_einzeilig, Rechnung_Textfelder*
Makro:	*Rechnungsinhalt_zusammenstellen, Rechnungsinhalt_zusammenstellen_Tabelle, Rechnungsinhalt_zusammenstellen_Tabelle_Uebertrag, Rechnungsinhalt_zusammenstellen_Tabelle_Uebertrag_Direktdruck*

Feldname	Feldtyp	Beschreibung
ID	Integer	Die ID ist Primärschlüssel dieser Tabelle. Sie ist als Autowert festgelegt.
Ware	Text	Eingabe erforderlich: Ja
Preis	Dezimal	Bei der Preisangabe ist darauf zu achten, dass 2 Nachkommastellen vorgesehen sind. Sonst sind nur ganzzahlige Angaben möglich. Das Feld kann auch entsprechend schon mit der Währungsangabe vorformatiert werden. Die Formatierung wird in Tabellen, Abfragen und beim Druck über einen Serienbrief berücksichtigt. Eingabe erforderlich: Nein

Die Tabellen sind für die Rechnungserstellung folgendermaßen verbunden:



Die Tabelle Anschrift wird separat bestückt und dient als erstes Beispiel für den Seriendruck. Die Kombination mit der Rechnungserstellung zeigt dann ein erweitertes Anwendungsbeispiel.

Abfragen

Adresseingabe

Datenquelle
Tabelle: *Anschrift*

Datenziel
Formular: *Anschrift_Textfelder*

```
001 SELECT "ID", "Vorname", "Nachname",
002 "Geschlecht" AS "Geschl_kurz",
003 "Straße_Nr", "Postleitzahl", "Ort",
004 CASE WHEN "Geschlecht" = 'm' THEN 'Herrn' ELSE 'Frau' END AS "Geschlecht"
005 FROM "Anschrift"
```

Hier wird lediglich das ursprüngliche Feld "Geschlecht" wird mit einem Alias versehen, da die selbe Bezeichnung jetzt für die Anrede genutzt werden soll. Mit der Funktion **`CASE WHEN`** wird ausgelesen, ob in dem Feld "Geschlecht" der Wert 'm' steht. Steht dort dieser Wert, so wird 'Herrn' ausgegeben. Ansonsten erscheint 'Frau'.

Für FIREBIRD muss diese Abfrage direkt als SQL in das Formular «Anschrift_Textfelder» eingegeben werden. Dort wird zur Zeit (LO 6.4.4.2) ein Alias nicht einwandfrei ausgelesen, wenn er, wie hier, einer vorherigen Benennung eines anderen Feldes entspricht. Aus der Bezeichnung «Geschlecht» in Zeile `4` wird bei Nutzung der Abfrage schlicht «CASE».

Etiketten

Datenquelle
Tabelle: *Waren*

Datenziel
Formular: *Etiketten_Serienbrief_im_Formular*

```
001 SELECT RIGHT( '0000000' || "ID", 8 ) "WarenNr",
002 CASE WHEN LOCATE( CHAR( 10 ), "Ware" ) > 0 THEN SUBSTRING ( "Ware" FROM 1
    FOR (LOCATE( CHAR( 10 ), "Ware", LOCATE( CHAR( 10 ), "Ware" ) + 1 )-1 ))
    ELSE "Ware" END AS "Warenbezeichnung",
003 REPLACE( "Preis" || ' €', '.', ',' ) "Preis"
004 FROM "Waren"
```

Die "WarenNr" wird hier aus dem Feld "ID" erstellt. Es werden einfach sieben Nullen vor die Nummer gesetzt und anschließend der so entstehende Text mit den acht Zeichen von rechts aus wieder gegeben. So werden alle Warennummern 8-stellig dargestellt.

Bei der "Ware" gibt es Einträge, die mit einem Zeilenumbruch mehrere Zeilen untereinander darstellen können. Hier wird mit **LOCATE** gesucht, ob "Ware" so einen Eintrag enthält. Ist der Eintrag gegeben, so wird der Text mit der Funktion **SUBSTRING** direkt vor einem eventuell weiteren Zeilenumbruch abgeschnitten. "Warenbezeichnung" enthält so höchsten zwei Zeilen.

Für FIREBIRD ist hier ein anderer Code zu wählen, da dort weder **CHAR** noch **LOCATE** vorhanden sind:

```
002 CASE WHEN POSITION( ASCII_CHAR( 10 ) IN "Ware" ) > 0 THEN SUBSTRING
    ( "Ware" FROM 1 FOR (POSITION( ASCII_CHAR( 10 ), "Ware", POSITION
    ( ASCII_CHAR( 10 ) IN "Ware" ) + 1 )-1 ))  ELSE "Ware" END AS
    "Warenbezeichnung",
```

Auch für den Preis, der auf der Etikette dargestellt werden soll, ist ein Eingriff notwendig. Sonst wird der Preis nicht korrekt mit dem Eurozeichen dargestellt. An "Preis" wird eine Leerstelle und das Eurozeichen angefügt. Mit **REPLACE** muss jetzt noch die Vorgabe der Datenbank für den Dezimaltrenner, der Punkt, durch ein Komma ersetzt werden.

Rechnung_einzeilig

Datenquelle
Tabelle: *Waren, Anschrift, Rechnung, Rechnungsinhalt*

Datenziel
Makro: *Rechnung*

```
001 SELECT "Anschrift"."Vorname", "Anschrift"."Nachname",
    "Anschrift"."Straße_Nr", "Anschrift"."Postleitzahl", "Anschrift"."Ort",
002 "a"."Datum" AS "Rechnungsdatum", "a"."ID",
003 "Rechnungsinhalt"."Rechnungsinhalt",
004 REPLACE( ( SELECT SUM( "Verkauf"."Anzahl" * "Waren"."Preis" ) FROM
    "Verkauf", "Waren" WHERE "Verkauf"."Waren_ID" = "Waren"."ID" AND
    "Verkauf"."Rechnung_ID" = "a"."ID" ) || ' €', '.', ',' ) AS "Summe"
005 FROM "Rechnung" AS "a", "Anschrift", "Rechnungsinhalt"
006 WHERE "a"."Anschrift_ID" = "Anschrift"."ID" AND "a"."ID" =
    "Rechnungsinhalt"."ID"
```

Der Tabelle "Anschrift" werden alle notwendigen Daten für die Anschrift in dem Rechnungsformular entnommen. Die Tabelle "Rechnung" wird über einen Alias als "a" angesprochen. Dies ist notwendig, damit in der korrelierenden Unterabfrage nur die Beträge addiert werden, die zu der

entsprechenden Rechnungsnummer (**"a"."ID"**) passen. Bei der Summenbildung **SUM("Verkauf"."Anzahl" * "Waren"."Preis")** wird auf die Rechnungsnummer des Datensatzes außerhalb der Unterabfrage so Bezug genommen. Auch in der Abfrage ist schließlich wieder das Ersetzen des Dezimalpunktes durch das Dezimalkomma anzutreffen.

Für FIREBIRD muss zur Zeit (LO 6.4.4.2) diese Abfrage in eine **Ansicht** umgewandelt werden. FIREBIRD liefert sonst zusammen mit dem Makro eine Fehlermeldung über ein «Ambigous field name», weil in den benutzten Tabelle überall der Primärschlüssel "ID" lautet. Die Abfrage selbst funktioniert aber auch in Firebird einwandfrei.

Rechnung_nicht_gedruckt

Datenquelle
Tabelle: *Rechnung*

Datenziel
Formular: *Rechnung_Textfelder_Uebertrag_Seriendruck*

```
001 SELECT * FROM "Rechnung" WHERE "Druckdatum" IS NULL
```

Hier wird lediglich der Inhalt der Tabelle "Rechnung" auf die Datensätze beschränkt, die noch keinen Eintrag im Feld "Druckdatum" haben. Dieser Eintrag wird durch das Makro «PDF_Export» der Tabelle "Rechnung" hinzugefügt.

Rechnung_Textfelder

Datenquelle
Tabelle: *Anschrift, Rechnung, Verkauf, Waren*

Datenziel
Formular: *Rechnung_Textfelder, Rechnung_Textfelder_Uebertrag, Rechnung_Textfelder_Uebertrag_Seriendruck*
Makro: *Rechnung_Textfelder*

```
001 SELECT "Anschrift"."Vorname", "Anschrift"."Nachname",
    "Anschrift"."Straße_Nr", "Anschrift"."Postleitzahl", "Anschrift"."Ort",
002 RIGHT( '0' || EXTRACT ( DAY FROM "a"."Datum" ), 2 ) || '.' || RIGHT( '0'
    || EXTRACT ( MONTH FROM "a"."Datum" ), 2 ) || '.' || EXTRACT ( YEAR FROM
    "a"."Datum" ) "Rechnungsdatum",
003 'RE_' || RIGHT( '0000000' || "a"."ID", 8 ) AS "Rechnungsnummer,
004 "a"."ID",
005 REPLACE( ( SELECT ROUND( SUM( "Verkauf"."Anzahl" *
    "Waren"."Preis" )*1.00/1.19 , 2) FROM "Verkauf", "Waren" WHERE
    "Verkauf"."Waren_ID" = "Waren"."ID" AND "Verkauf"."Rechnung_ID" = "a"."ID"
    ) || ' €', '.', ',' ) AS "Netto",
006 REPLACE( ( SELECT ROUND( SUM( "Verkauf"."Anzahl" *
    "Waren"."Preis" )*0.19/1.19 , 2) FROM "Verkauf", "Waren" WHERE
    "Verkauf"."Waren_ID" = "Waren"."ID" AND "Verkauf"."Rechnung_ID" = "a"."ID"
    ) || ' €', '.', ',' ) AS "Umsatzsteuer",
007 REPLACE( ( SELECT SUM( "Verkauf"."Anzahl" * "Waren"."Preis" ) FROM
    "Verkauf", "Waren" WHERE "Verkauf"."Waren_ID" = "Waren"."ID" AND
    "Verkauf"."Rechnung_ID" = "a"."ID" ) || ' €', '.', ',' ) "Summe"
008 FROM "Rechnung" AS "a", "Anschrift"
009 WHERE "a"."Anschrift_ID" = "Anschrift"."ID"
```

Zuerst werden die Daten der Anschrift erfasst (Zeile `1`).

Beim Rechnungsdatum ist darauf zu achten, dass es, sofern in ein Textfeld eingelesen wird, in der korrekten Formatierung übergeben wird (Zeile 2). Wird es nicht entsprechend vorformatiert, so erscheint es in der Schreibweise, die bei Datenbanken Standard ist: 2014-05-07. Mit **`EXTRACT ( DAY FROM "a"."Datum" )`** wird aus der Tabelle "Rechnung" (hier angesprochen über den Alias "a") der Tag des Datums ausgelesen – also z.B. «7». Der Tag soll allerdings zweistellig dargestellt werden, gegebenenfalls also mit einer führenden «0». Dies wird über **`RIGHT( '0' || EXTRACT ( DAY FROM "a"."Datum" ), 2 )`** schließlich erreicht. Es wird eine «0» vor die «7» gesetzt und anschließend von rechts aus die letzten zwei Zeichen gelesen. Damit werden gleichzeitig zweistellige Tage, vor die eine «0» gesetzt wurde, wieder korrekt dargestellt. Entsprechend dem Tag wird auch mit dem Monat verfahren. Die Punkte als Trenner zwischen Tag und Monat sowie Monat und Jahr werden ebenfalls in die Verbindung über **`||`** mit einbezogen.

In Zeile 3 wird die Rechnungsnummer aus dem Primärschlüssel der Rechnung erstellt. Hier werden einfach 8 Nullen vor den Primärschlüsselwert geschrieben und die hinteren 8 Ziffern übernommen, so dass letztlich immer 8 Ziffern die Rechnungsnummer ergeben. Das Ganze wird dann noch mit dem führenden Text 'RE_' versehen.

Der Primärschlüssel der Rechnung muss noch separat mit aufgenommen werden (Zeile 4), da ein Auslesen aus der Nummer mittels Makro zu umständlich wäre, der Schlüsselwert aber in den Makros benötigt wird.

In den Zeilen 5 bis 7 werden die Berechnungen der Summe, bezogen auf die jeweilige Rechnung, vorgenommen. Die Zeilen 5 und 6 berechnen hier auf Basis der gleichen Rechnung wie in Zeile 7 die Summe, ermitteln daraus aber durch Multiplikation und Division den Nettowert und den Umsatzsteuerwert (bezogen auf 19%). Die Summenbildung sowie die Versorgung mit Dezimalkomma und €-Zeichen worden bereits in den Abfrage «Etiketten» und «Rechnung_einzeilig» erklärt. Hier deshalb nur noch die Umrechnungen für «Netto» und «Umsatzsteuer».

In Zeile 5 wird zuerst die Summe mit dem Wert 1,00 multipliziert. Dies ist deswegen notwendig, weil die Summe selbst nur 2 Nachkommastellen hat, für eine bessere Rundung aber mehr Nachkommastellen benötigt werden. Durch die Multiplikation mit 1,00 werden der bisherigen Summe lediglich 2 weitere Nachkommastellen hinzugefügt. Die Bruttosumme entspricht 119%, also wird die Nettosumme durch eine Division durch 1,19 ermittelt. Hier wird also fest von einem Steuersatz von 19% ausgegangen. Dies könnte bei einer komplexer aufgebauten Datenbank natürlich abhängig von dem Steuersatz, der bei der Ware gespeichert würde, berechnet werden.

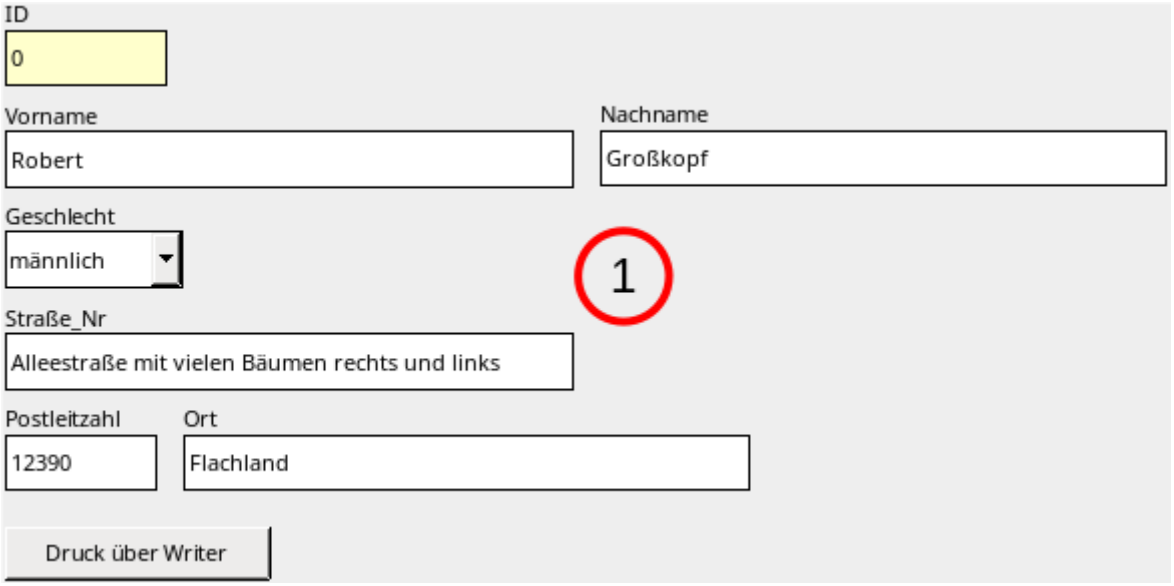
Die so ermittelte Zahl hat 4 Nachkommastellen. Um daraus wieder einen sinnvollen Währungsbetrag zu machen wird auf die 2. Nachkommastelle gerundet.

In Zeile 6 wird dann statt einer Multiplikation mit 1,00 (100%) eine Multiplikation mit 0,19 vorgenommen. Alle anderen Vorgehen stimmen mit Zeile 5 überein.

Formulare

Die Formulare greifen mit unterschiedlichem Schwierigkeitsgrad auf die Serienbrieffunktion bzw. auf Textfelder eine Vorlage zu. Der einfachste Zugriff erfolgt für den Inhalt, der in der Tabelle "Anschrift" gespeichert wird.

Anschrift



1	MainForm (Tabelle: *Anschrift*)

Makros in Feldeigenschaften		
1	Schaltfläche Druck über Writer (Aktion ausführen)	Module1.*Serienbrief*

Benötigt
Anmeldung der Datenbank über **Extras → Optionen → LibreOffice Base → Datenbanken → Neu** als «Beispiel_Datenbank_Serienbrief_direkt»

Das erste Formular stellt ein einfaches Eingabeformular für Adressen dar.

Das Formular speichert die Daten in der Tabelle "Anschrift" ab. Das Primärschlüsselfeld ist farblich abweichend gekennzeichnet, da der Wert automatisch erstellt wird.

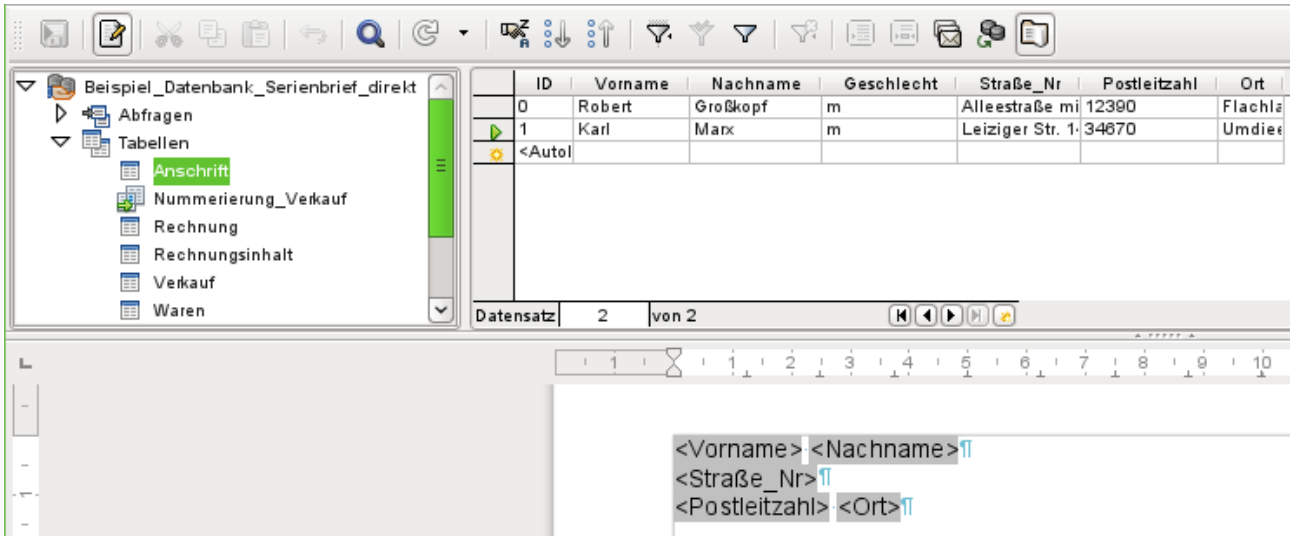
Das Listenfeld wird nicht über eine zusätzliche Tabelle per SQL geladen, sondern ist hier direkt über eine Werteliste definiert. Unter **Eigenschaften → Daten → Listeninhalt** sind die Werte «m» und «w». Um die Werte in dieser Form einzugeben muss lediglich das vorgesehene Feld mit dem Cursor betreten werden. Das Feld klappt auf und über die Eingabe von *Strg*+*Enter* kann nach der Eingabe des ersten Wertes in die zweite Zeile für einen zweiten Wert gewechselt werden.

Der Listeninhalt wird in die dem Formular zugrundeliegenden Tabelle "Anschrift" übertragen. Dargestellt werden allerdings die unter **Eigenschaften → Daten → Listen-Einträge** gemachten

Einträge. Hier ist auf die selbe Reihenfolge zu achten, damit nicht bei der Anzeige «weiblich» in der Tabelle «m» abgespeichert wird.

Wird in einem Datensatz der Button Druck über Writer betätigt, so wird eine neue Textdatei mit dem im Serienbrief abgelegten Feldinhalt erstellt. Der Dateiname wird aus dem Nachnamen und einer folgenden 0 sowie der Endung «odt» zusammengefügt. Die Datei hat den selben Inhalt, wie ihn auch ein Druck, ausgelöst von der ursprünglichen Serienbriefdatei, hätte.

Der Serienbrief selbst wurde hier einfach über den Datenbankbrowser erstellt:



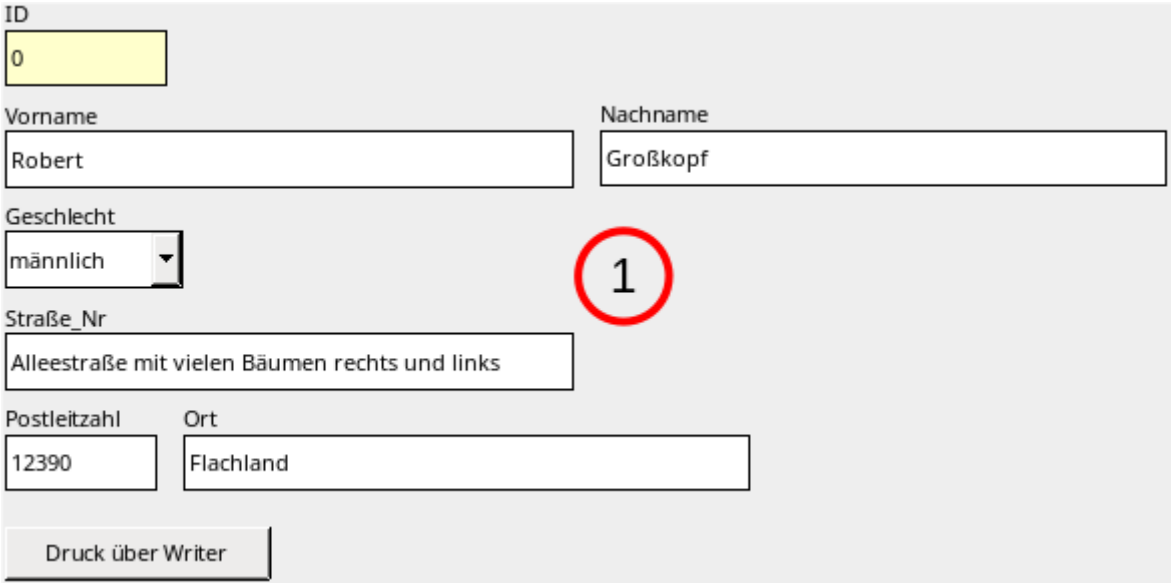
Mit F4 oder über **Ansicht → Datenquelle** oder auch über den entsprechenden Button in der Symbolleiste werden die Datenquellen angezeigt. Aus der Datenquelle wird die entsprechende Tabelle (hier «Anschrift») ausgesucht. Die Tabelle wird rechts davon angezeigt. Jetzt können die Serienbrieffelder durch einen Druck mit der linken Maustaste auf den jeweiligen Tabellenkopf in das darunterliegende Writer-Dokument gezogen werden.

Die Feldauswahl der Serienbrieffelder kann alternativ auch über **Einfügen → Feldbefehl → Andere → Datenbank → Seriendruckfeld** erfolgen.

Statt aber diesen Vorlagenbrief zuerst zu starten, dann den Datensatz auszusuchen, den Druck zu starten und die Bedingungen dafür festzulegen, erfolgt die Erstellung des Zieldokumentes bei Betätigung von Druck über Writer direkt, ohne das Ausgangsdokument auf dem Bildschirm zu öffnen.

Dieser Zugriff auf das Dokument wird über das Makro «Serienbrief» erreicht.

Anschrift_Textfelder

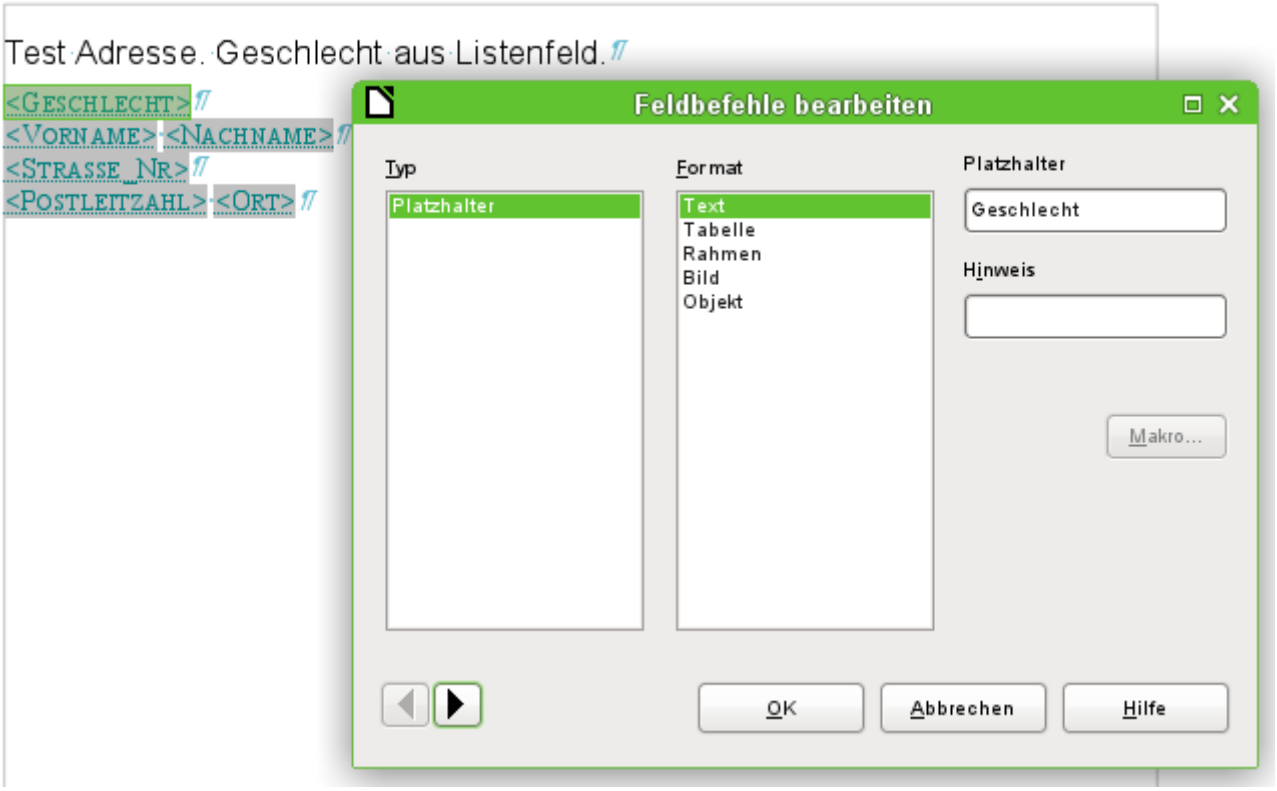


1	MainForm (Abfrage: *Adresseingabe*)

Makros in Feldeigenschaften		
1	Schaltfläche Druck über Writer (Aktion ausführen)	Module1.*Textfelder_Fuellen*

Das Formular «Anschrift_Textfelder» arbeitet nicht mit der Serienbrieffunktion von LibreOffice zusammen. Stattdessen wird auf Platzhalter zugegriffen.

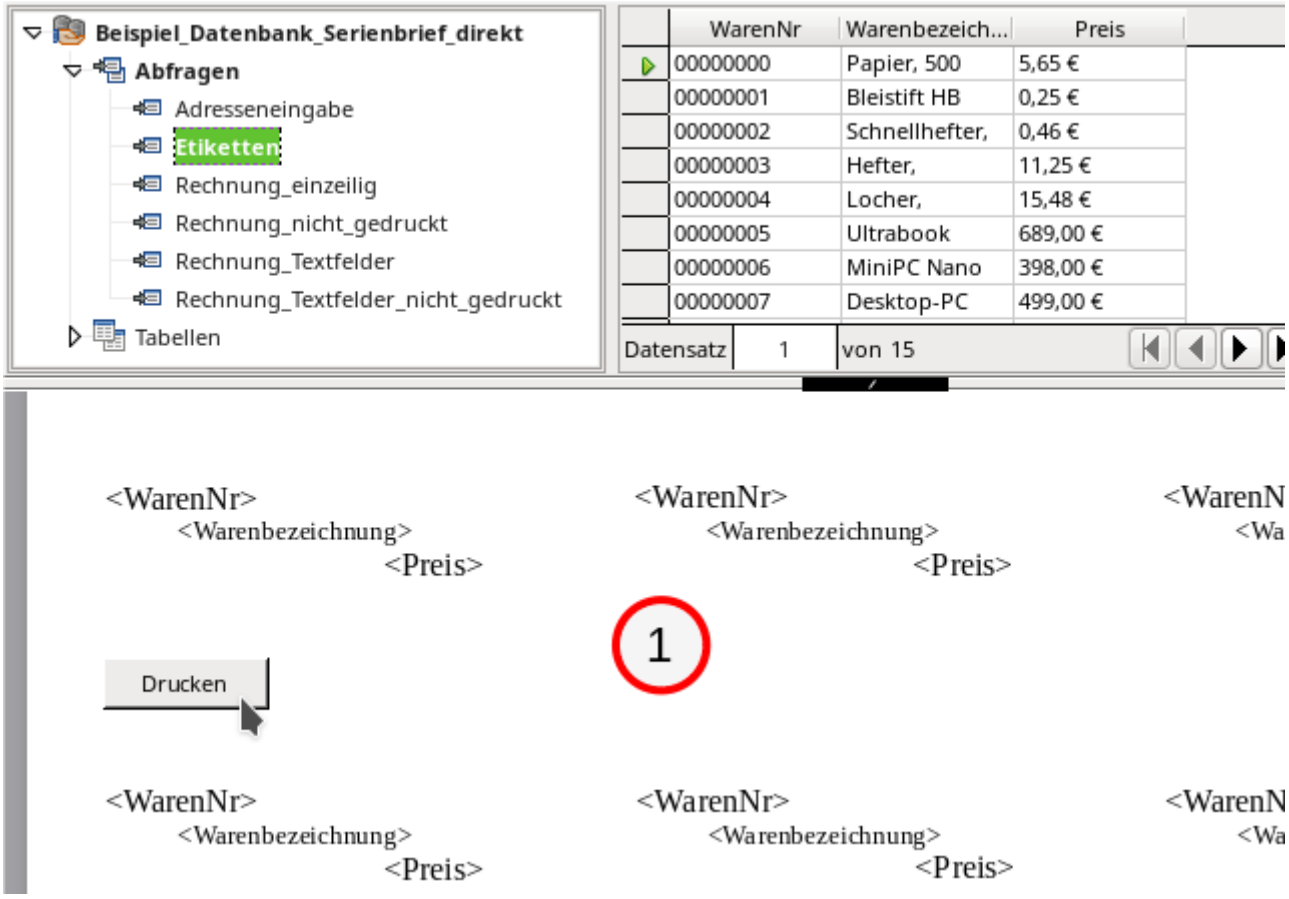
Zuerst muss im Writer eine Vorlage mit Platzhaltern erstellt werden. Dies geschieht über **Einfügen → Feldbefehl → Funktionen → Platzhalter**. Die Platzhalter werden der Einfachheit halber so benannt, wie das entsprechende Feld in der Tabelle bzw. in der Abfrage heißt, deren Inhalt der Platzhalter annehmen soll.



Für einfache Zwecke reicht hier der Typ «Text», mit den anderen Varianten, z.B. «Grafik», können dann weitere Inhalte umgesetzt werden.

In dem Makro wird der Pfad zur Vorlage hinterlegt. Von dem Makro werden die Platzhalter befüllt. Es wird ein neues Dokument erstellt, das direkt mit den Inhalten gefüllt wird. Das damit geöffnete Dokument muss nur noch abgespeichert oder direkt ausgedruckt werden.

Etiketten_Serienbrief_im_Formular



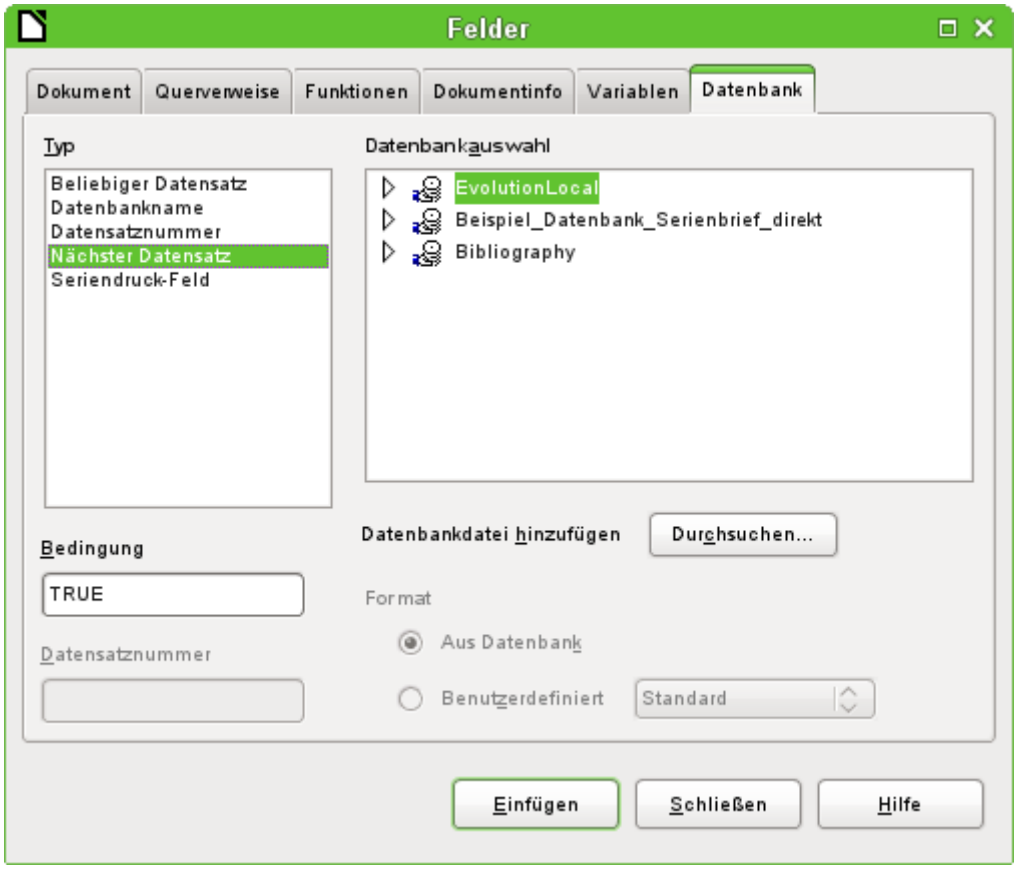
1	Formular (keine Datenquelle angegeben, benötigt wird aber Abfrage *Etiketten*)

Benötigt
Anmeldung der Datenbank über **Extras → Optionen → LibreOffice Base → Datenbanken → Neu** als «Beispiel_Datenbank_Serienbrief_direkt»

Die bisherigen Formulare zeigen, wie ein Seriendruck mit einer Vorlage außerhalb der Datenbankdatei bewerkstelligt wird. In diesem Formular wird ausgenutzt, dass die enthaltenen Formulare grundsätzlich über das Writer-Modul von LibreOffice erstellt werden. Es ist also auch möglich, einen Serienbrief als Formulardokument zu erzeugen. Wird dann das Formular aufgerufen und die Druckfunktion gewählt, so erscheint automatisch die Nachfrage, ob ein Seriendruck erfolgen soll. Die Datenbank muss für diese Funktion allerdings in LibreOffice angemeldet sein.

Um so ein Formular zu erstellen sollte bei der Erstellung darauf geachtet werden dass **Ansicht → Drucklayout** im Formular gewählt wird. Auch wenn das Formular später im Weblayout geöffnet wird richtet sich der Ausdruck nach den Einstellungen für die Druckseite.

In diesem Formular sollen Etiketten erstellt werden. Dazu wird zunächst der Rand der Seite recht klein gestellt. Hier gibt das vorliegende Etikettenmaterial die Größen vor. Für ein Etikett wird ein Rahmen ohne Rand und ohne Umlauf aufgezogen. Anschließend wird der Datenbankbrowser gestartet und die gewünschten Felder werden hinzugefügt.

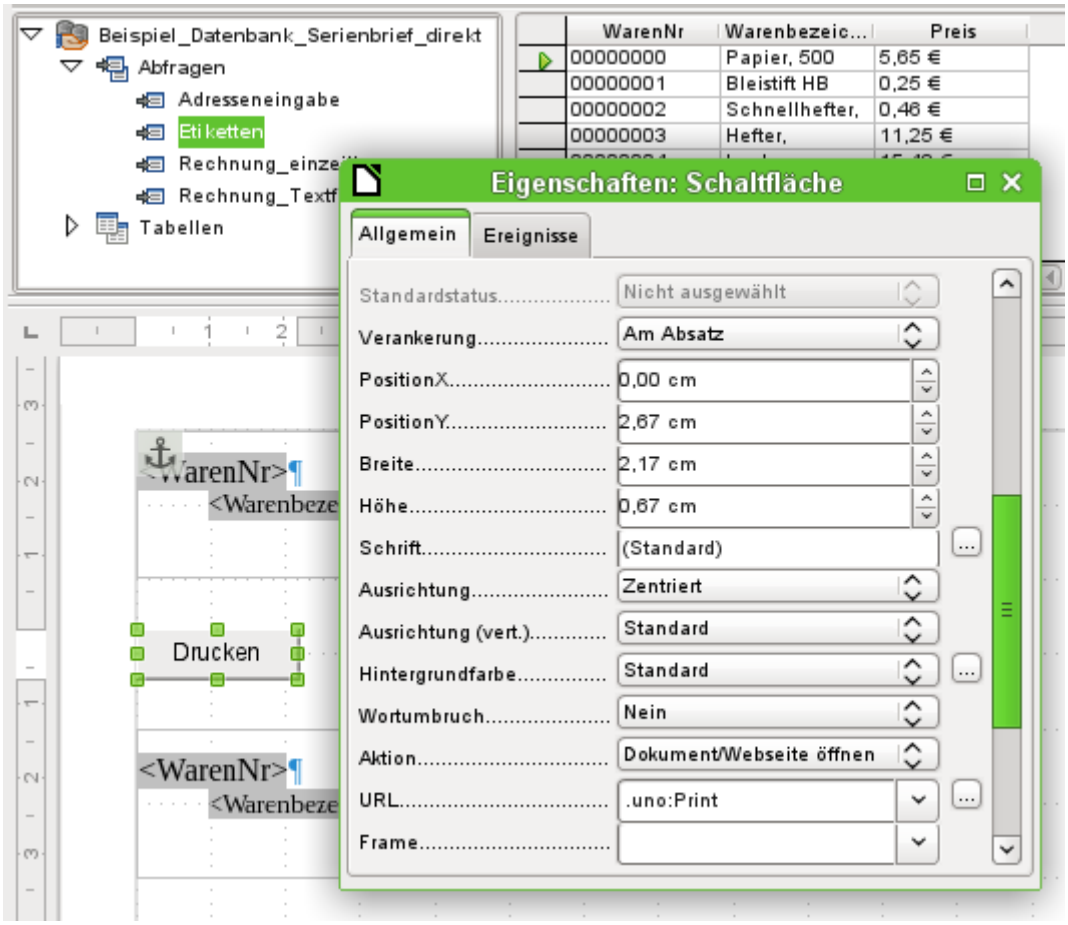


Über **Einfügen → Feldbefehl → Andere → Datenbank** wird als Abschlussfeld eines jeden Etiketts Nächster **Datensatz → TRUE** gewählt. Lediglich das letzte Etikett der Seite ist hiervon ausgenommen, da mit einem Seitenumbruch automatisch zum nächsten Datensatz gewechselt wird.

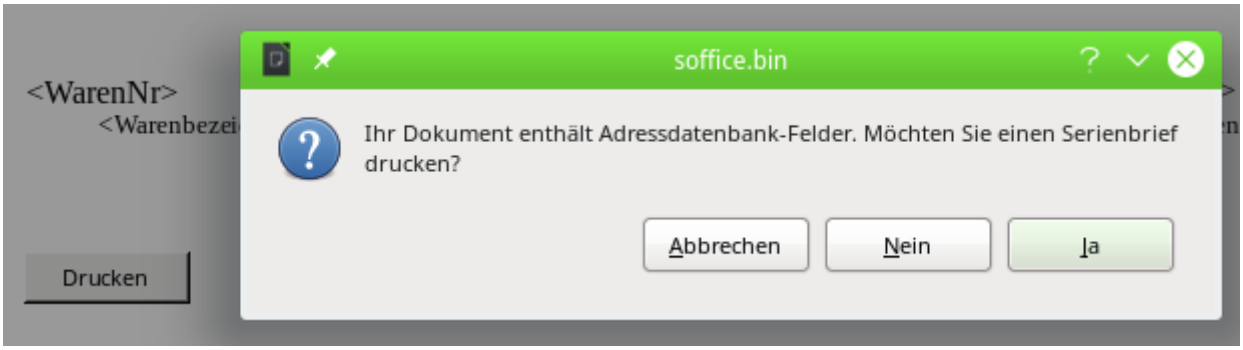
Die Rahmen werden in der gewünschten Anzahl kopiert, neu eingefügt und an entsprechender Stelle auf dem Blatt positioniert. In der Beispieldatei wurde dabei jeder Rahmen an die Seite gebunden – dies kann natürlich nach Belieben anders gehandhabt werden.

Die Informationen, die auf den Etiketten erscheinen sollen, sind in der Abfrage «Etiketten» gesammelt.

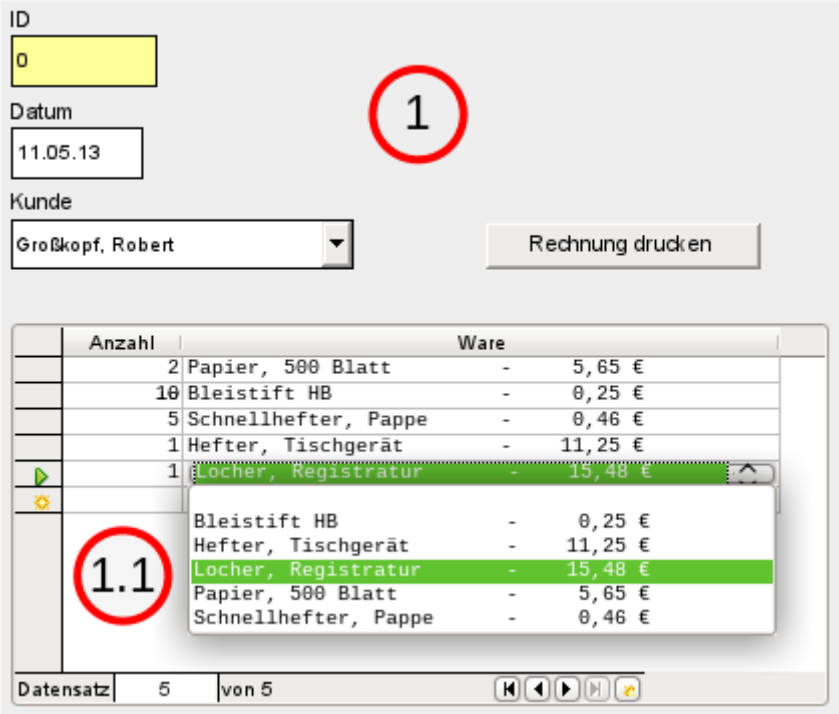
Als kleiner Zusatz wurde noch ein Button eingefügt, der direkt den Druckbefehl aufruft.



Der Druckbefehl wird gestartet, indem in **Eigenschaften: Schaltfläche → Allgemein** die **Aktion → Dokument/Webseite öffnen** und als **URL → .uno:Print** eingefügt wird.



Rechnung



1	MainForm (Tabelle: *Rechnung*)
1	Listenfeld Kunde (Tabelle: *Anschrift*)
1.1	SubForm (Tabelle: *Verkauf*)
1.1	Listenfeld Ware (Tabelle: *Waren*)

Makros in Feldeigenschaften		
1	Schaltfläche Rechnung drucken (Aktion ausführen)	Module1.*Rechnung*

Benötigt
Anmeldung der Datenbank über **Extras → Optionen → LibreOffice Base → Datenbanken → Neu** als «Beispiel_Datenbank_Serienbrief_direkt»

Das Listenfeld «Kunde» wird wie folgt mit Daten versorgt: **Kontextmenü → Steuerelement-Eigenschaften → Daten → Art des Listeninhalts** «SQL». Dort wird die folgende Abfrage erstellt:

```
001 SELECT "Nachname"||', '||"Vorname", "ID"
002 FROM "Anschrift"
003 ORDER BY "Nachname" ASC, "Vorname" ASC"
```

Als erstes Feld wird der Nachname und der Vorname, getrennt durch ein Komma, zusammengefasst. Das zweite Feld ist dann der Primärschlüssel zu dieser Person (Zeile `1`). In Zeile `3` wird dann die Abfrage sinnvollerweise zuerst nach dem Nachnamen und dann nach dem Vornamen sortiert.

Beim Formular «Rechnung» werden einige zusätzliche Problemstellungen angegangen. Die erste Problemstellung zeigt bereits der Screenshot während einer Eingabe: Im Listenfeld sind neben den Namen die Preis angezeigt – allerdings gleich so, dass das ganze wie eine Tabelle aussieht.

Um ein **Listenfeld** so wie oben erscheinen zu lassen muss zuerst einmal die Schriftart von einer proportionalen Schriftart zu einer Schriftart mit fester Zeichenbreite umgewandelt wer-

den. Das Tabellenkontrollfeld wird markiert. Über das **Kontextmenü → Kontrollfeld → Eigenschaften: Tabellen-Steuerelement → Allgemein → Schrift** wird die Schriftart «Liberation Mono, Standard» ausgesucht.

Anschließend wird der Tabellenkopf «Ware» angeklickt. Über das **Kontextmenü → Spalte → Eigenschaften: Listenfeld → Daten → Art des Listeninhalts** wird «SQL» gewählt. Danach wird der **Listeninhalt** festgelegt:

```
001 SELECT
002    LEFT("Ware"|| SPACE(25), 25) || ' - ' ||
          REPLACE( RIGHT( SPACE(8) ||"Preis", 8),'.',',') || ' €',
003    "ID"
004 FROM "Waren"
005 ORDER BY ("Ware" || ' - ' || "Preis" || ' €') ASC
```

Es werden Daten aus der Tabelle "Waren" ausgelesen. Mit **||** werden Elemente verschiedener Felder bzw. auch Text verbunden. An den Inhalt des Feldes werden 25 Leerzeichen (**SPACE**) angehängt. Anschließend wird der so entstandene Text von links aus (**LEFT**) auf 25 Zeichen zurechtgestutzt. Dadurch ist die Zeichenlänge für die Darstellung der Ware festgelegt. Der "Preis" wird ebenfalls mit einer entsprechenden Anzahl an Leerzeichen gekoppelt. Da der Preis aber rechtsbündig dargestellt wird, werden jetzt 8 Zeichen von rechts aus (**RIGHT**) gelesen. Höhere Preise sind in der Datenbank sowieso nicht vorgesehen. Damit der Preis außerdem mit einem Dezimalkomma und nicht mit einem Dezimalpunkt dargestellt wird, wird über **REPLACE** der Punkt durch ein Komma ersetzt. Schließlich wird noch das €-Zeichen an den Gesamtstring angehängt.

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.

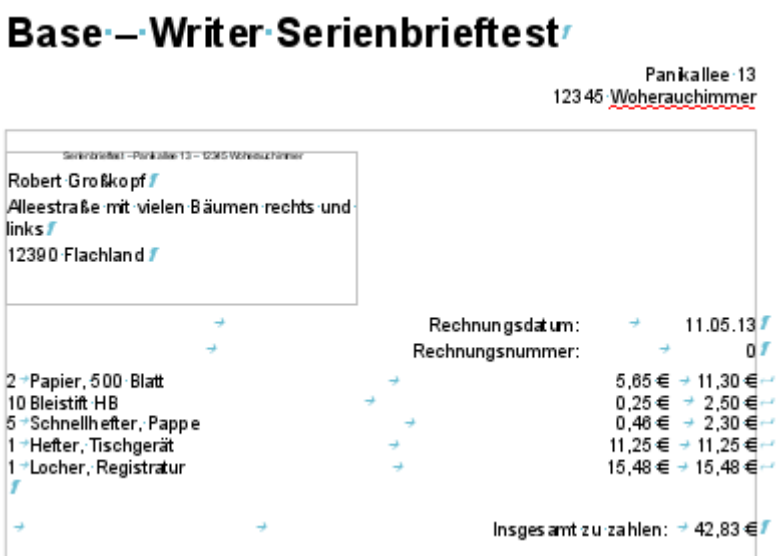
Für FIREBIRD muss dieser Code etwas angepasst werden

```
002    RPAD(REPLACE("Ware",ASCII_CHAR(10),' '),25) || ' - ' ||
          REPLACE(LPAD("Preis",8),'.',',') || ' €',
```

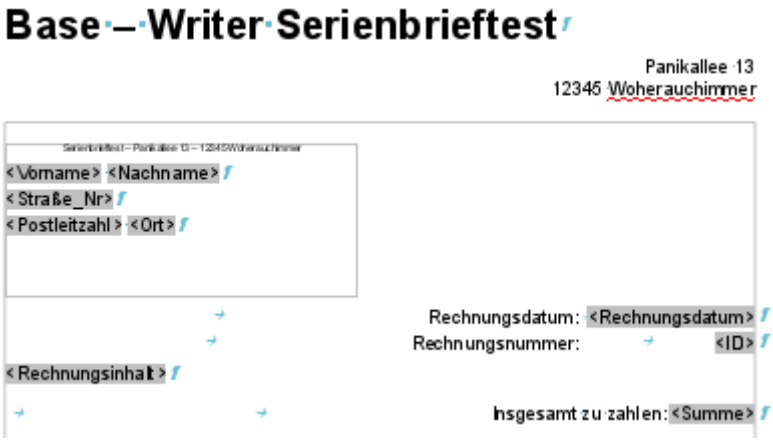
Zum einen ist wieder **CHAR** durch **ASCII_CHAR** zu ersetzen. Zum anderen ist die Funktion **SPACE** unbekannt, so dass hier mit der Funktion **RPAD** mit Leerzeichen von rechts aus und mit der Funktion **LPAD** mit Leerzeichen von links aus aufgefüllt wird. Diese beiden Funktionen sind wiederum in der HSQLDB unbekannt.

Wie die Anzahl der erforderlichen Leerzeichen auch automatisch ermittelt werden kann ist im Base-Handbuch beschrieben.

Wird über Rechnung drucken der Seriendruck gestartet, so läuft der Prozess genau wie beim Seriendruck aus dem Formular «Anschrift» im Hintergrund ab. Es wird eine Datei erstellt, die wie im folgenden Bild zu sehen gefüllt ist:

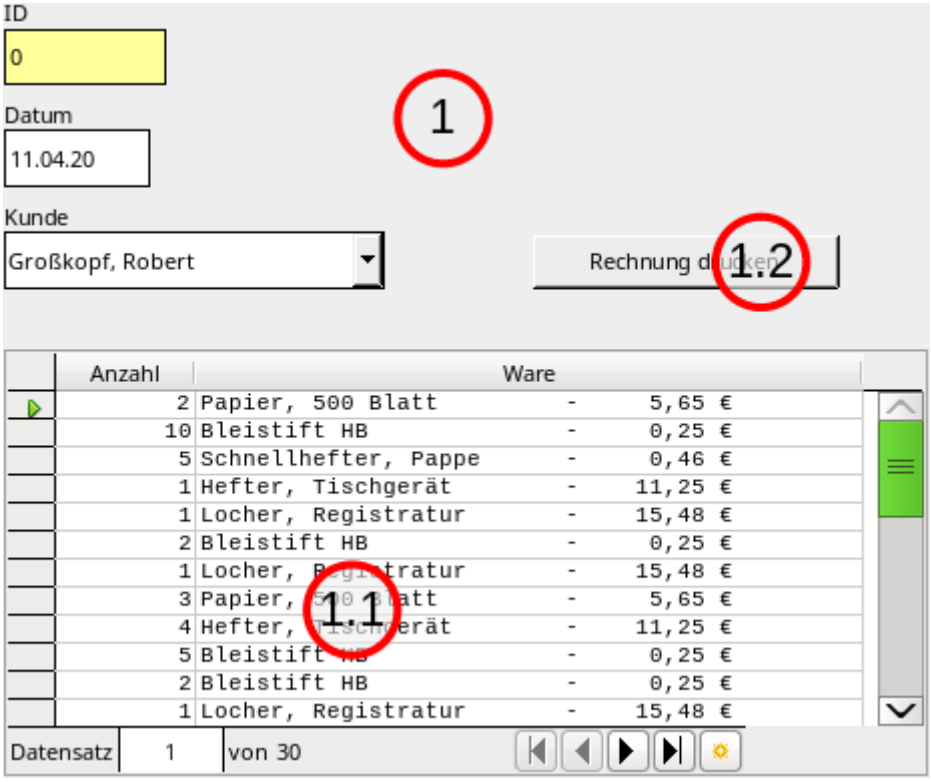


Durch die Anzeige der Steuerzeichen wird im Ausdruck deutlich, wie eine Formatierung in Tabellenform erreicht wurde. Der Rechnungsinhalt wurde durch Tabulatoren und weiche Umbrüche in das Dokument eingefügt. Die Lage der Tabulatoren im Absatz ist über das Absatzformat der Vorlage definiert. Sie kann natürlich noch angepasst werden, falls die Preise oder auch die Anzahl mehr Platz benötigen. Die Länge des Gesamtinhaltes ist allerdings durch die Zeile begrenzt.



In der Vorlage erscheint für den Rechnungsinhalt nur das Serienbrieffeld **< Rechnungsinhalt >**. Es wird, wie alle anderen Felder, aus einem einzigen Datensatz der Datenquelle ausgelesen. Der wesentliche Unterschied zur Erstellung eines Seriendrucks für die Anschrift liegt in der Erstellung des Inhaltes für das Feld "Rechnungsinhalt".

Rechnung_Textfelder



1	MainForm (Tabelle: *Rechnung*)
1	Listenfeld Kunde (Tabelle: *Anschrift*)
1.1	SubForm (Tabelle: *Verkauf*)
1.1	Listenfeld Ware (Tabelle: *Waren*)
1.2	Druck (Abfrage: *Rechnung_Textfelder*)

Makros in Feldeigenschaften		
1.2	Schaltfläche Rechnung drucken (Aktion ausführen)	Module1.*Rechnungsinhalt_zusammenstellen_Tabelle*

Der Ausdruck aus dem Formular «Rechnung_Textfelder» ähnelt erst einmal sehr dem Ausdruck über den Serienbrief aus dem Formular «Rechnung».

Rechnungstest¶

Panikallee·13¶
12345·Woherauchimmer¶

Rechnungstest – Panikallee·13 – 12345·Woherauchimmer¶

Karl·Marx¶
Leiziger·Str.·145¶
34670·Umdieecke¶

		Rechnungsdatum:	17.04.2020¶
		Rechnungsnummer:	RE_00000002¶
2¶	Schnellhefter,·Pappe¶	0,46·€¶	0,92·€¶
5¶	Papier,·500·Blatt¶	5,65·€¶	28,25·€¶
10¶	Bleistift·HB¶	0,25·€¶	2,50·€¶
1¶	Hefter,·Tischgerät¶	11,25·€¶	11,25·€¶
1¶	Locher,·Registratur¶	15,48·€¶	15,48·€¶
		Nettobetrag:	*49,08·€*¶
		Umsatzsteuer·19%:	*9,32·€*¶
		Insgesamt·zu·zahlen:	**58,40·€**¶

Bei genauer Beobachtung vor allem der Steuerzeichen wird allerdings klar, dass hier für den Rechnungsinhalt eine andere Darstellung gewählt wurde. Hinter der Anzahl und den Warenbezeichnungen sind Absatzendmarken zu sehen. Die Zeilen werden also nicht mittels eines Tabulators in tabellarischer Form gehalten.

Rechnungstest¶

Panikallee·13¶
12345·Woherauchimmer¶

Rechnungstest – Panikallee·13 – 12345·Woherauchimmer¶

<Vorname>·<Nachname>¶
<Strasse_Nr>¶
<Postleitzahl>·<Ort>¶

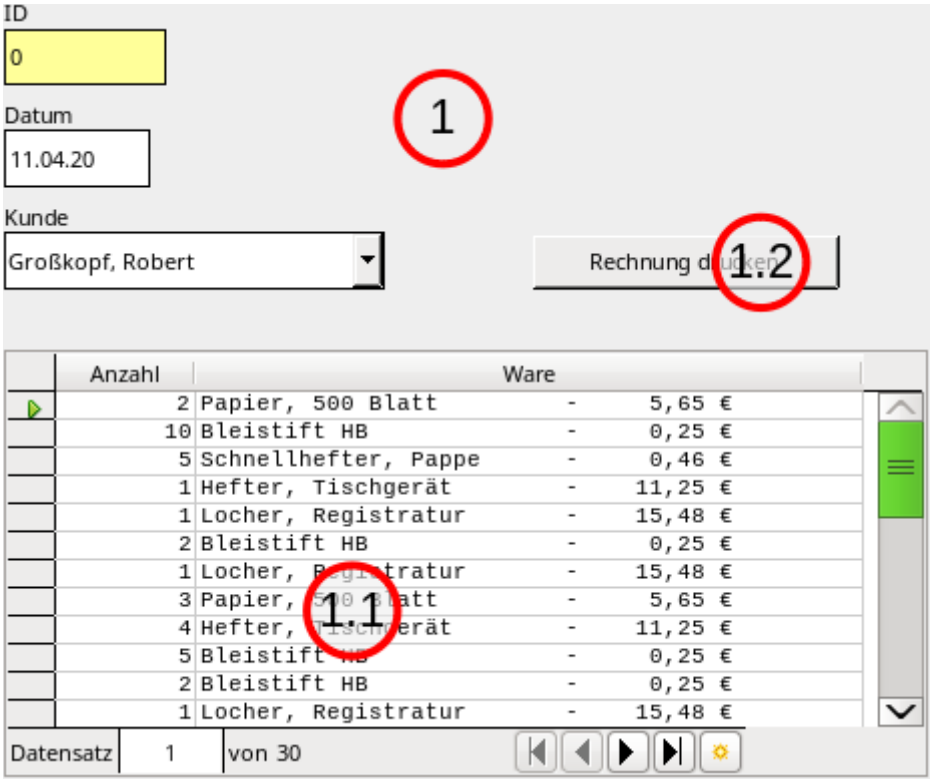
		Rechnungsdatum:	<Rechnungsdatum>¶
		Rechnungsnummer:	<Rechnungsnummer>¶
¶	¶		
		Nettobetrag:	*<Netto>*¶
		Umsatzsteuer·19%:	*<Umsatzsteuer>*¶
		Insgesamt·zu·zahlen:	**<Summe>**¶

Die Vorlage gibt näheren Aufschluss darüber, wie diese Absatzendmarken zustande kommen. Sie gehören zu den ersten zwei Spalten einer Tabelle. Die anderen Spalten sind rechtsbündig ausgerichtet und haben daher eine nur wenig sichtbare sichtbare Absatzendmarke. Die Tabelle

ist hier nur zum besseren Verständnis über **Ansicht → Tabellenbegrenzungen** eingefärbt worden. Vorteil dieser Vorlage gegenüber der des Formulars «Rechnung» ist, dass der Inhalt in einer Tabellenzelle auch ruhig etwas größer sein kann. Das Layout wird nicht durcheinander kommen, wenn die Warenbeschreibung über mehrere Zeilen geht. Leider erfordert dieser Vorteil auch eine etwas erweiterte Anwendung der Makroprogrammierung.

Lediglich die in der Abfrage erstellten Felder werden auch als Felder in dem Ausdruck benötigt. Der restliche Inhalt wird über ein Makro direkt in die betreffenden Zellen geschrieben.

Rechnung_Textfelder_Uebertrag

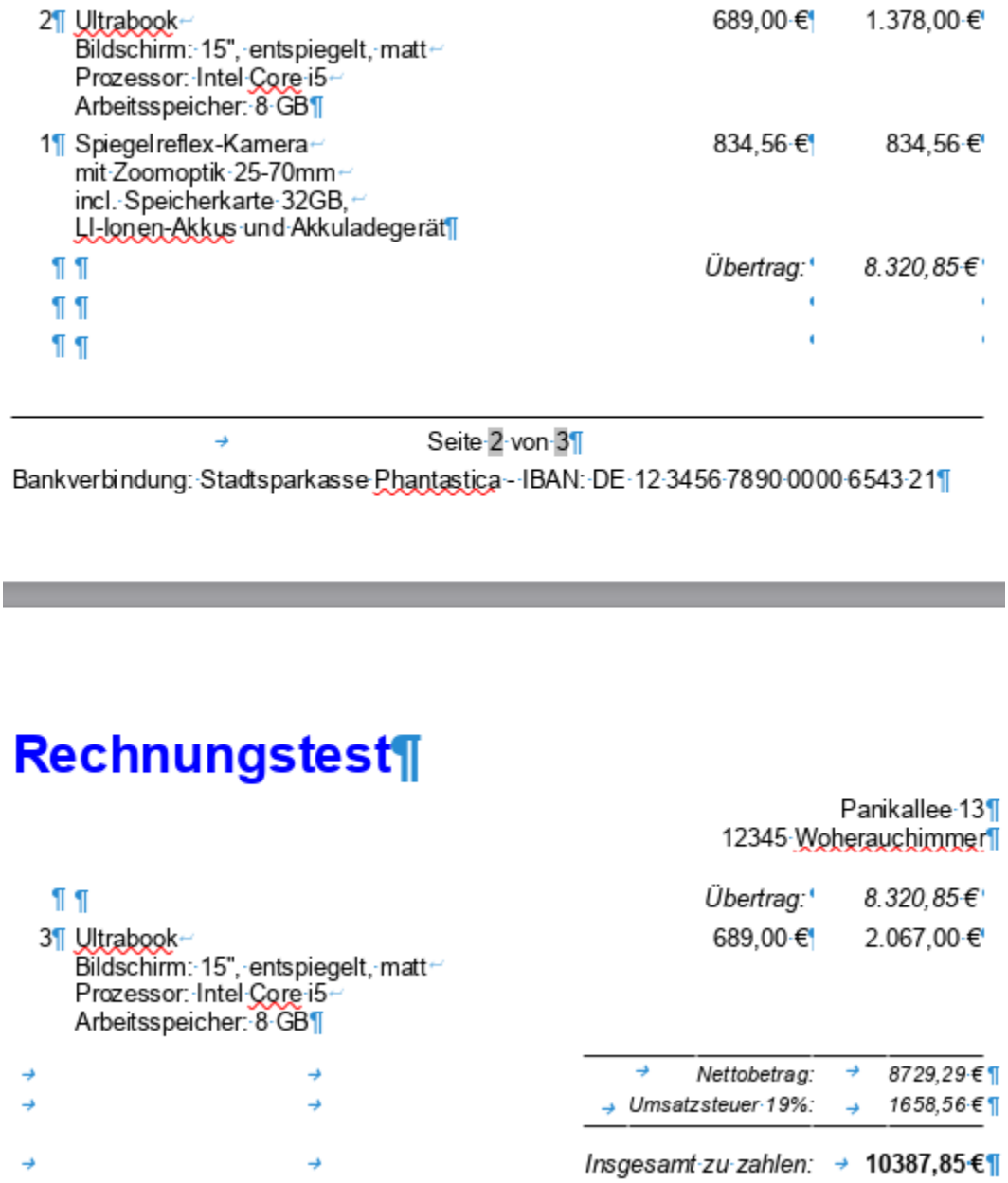


1	MainForm (Tabelle: *Rechnung*)
1	Listenfeld Kunde (Tabelle: *Anschrift*)
1.1	SubForm (Tabelle: *Verkauf*)
1.1	Listenfeld Ware (Tabelle: *Waren*)
1.2	Druck (Abfrage: *Rechnung_Textfelder*)

Makros in Feldeigenschaften		
1.2	Schaltfläche Rechnung drucken (Aktion ausführen)	Module1.*Rechnungsinhalt_zusammenstellen_Tabelle_Uebertrag*

Das Formular «Rechnung_Textfelder_Uebertrag» unterscheidet sich von dem Formular «Rechnung_Textfelder» äußerlich überhaupt nicht. Hinter dem Button Rechnung drucken wird lediglich ein etwas erweitertes Makro angesteuert.

Der Druck ergänzt die ursprüngliche Druckfunktion so, dass beim Übergang von einer zur nächsten Seite bei entsprechend vielen Rechnungstiteln ein Übertrag am Seitenschluss der vorhergehenden und am Seitenanfang der folgenden Seite erscheint.



2 Ultrabook — 689,00 € — 1.378,00 €
Bildschirm: 15", entspiegelt, matt
Prozessor: Intel Core i5
Arbeitsspeicher: 8 GB

1 Spiegelreflex-Kamera — 834,56 € — 834,56 €
mit Zoomoptik 25-70mm
incl. Speicherkarte 32GB,
LI-Ionen-Akkus und Akkuladegerät

Übertrag: *8.320,85 €*

Seite 2 von 3

Bankverbindung: Stadtsparkasse Phantastica - IBAN: DE 12 3456 7890 0000 6543 21

Rechnungstest

Panikallee 13
12345 Woherauchimmer

Übertrag: *8.320,85 €*

3 Ultrabook — 689,00 € — 2.067,00 €
Bildschirm: 15", entspiegelt, matt
Prozessor: Intel Core i5
Arbeitsspeicher: 8 GB

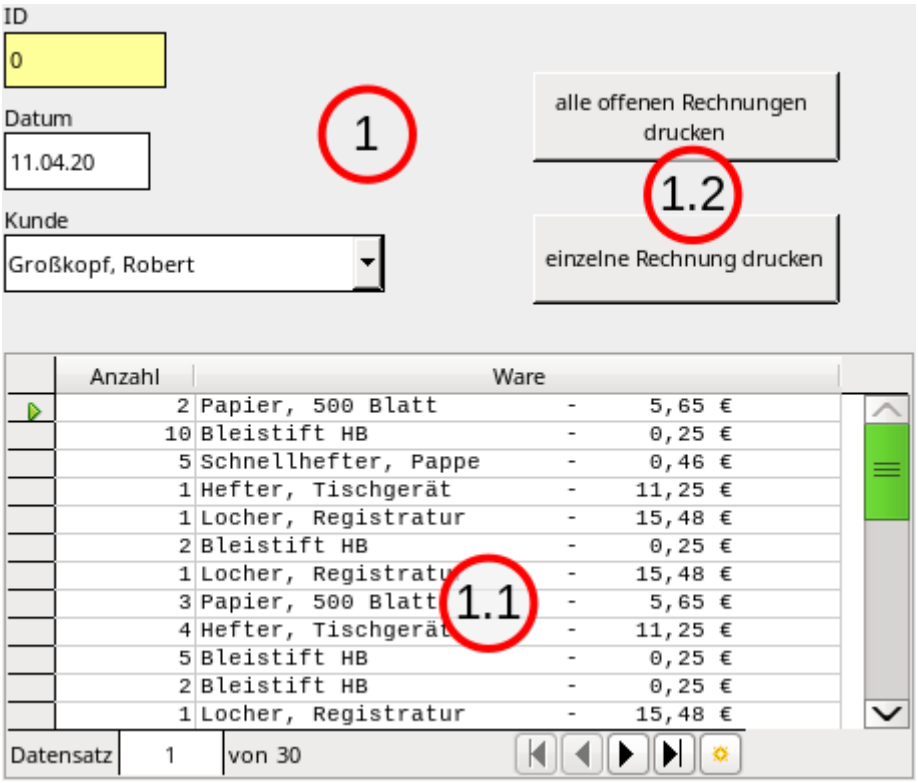
Nettobetrag:	*8729,29 €*
Umsatzsteuer 19%:	*1658,56 €*
Insgesamt zu zahlen:	**10387,85 €**

Das Bild zeigt solch einen Übertrag beim Seitenwechsel. Der Übertrag berücksichtigt dabei auch, dass vielleicht mehrzeilige Eingaben bei den Waren erscheinen. Die in dem Screenshot enthaltenen Steuerzeichen zeigen, dass hier weiterhin die Tabelle genutzt wird und dem Übertrag am Seitenende wegen des zusätzlichen Platzes noch zwei leere Tabellenzeilen folgen. Die verbuchte Ware benötigt hier schließlich vier Zeilen, die aber in der Tabelle auf der Vorseite keinen Platz mehr gehabt hätten. Außerdem wäre dann auch noch die Summierung alleine auf der Folgeseite erschienen - alles Dinge, die durch das Makro ausgeschlossen werden sollen.

Einen Nachteil hat diese Konstruktion noch: Die zugrundeliegende Abfrage berücksichtigt noch nicht, dass eventuelle einzelne gleiche Waren mehrfach in einer Rechnung vorkommen. Hier erscheinen auf der oberen Seite zwei Ultrabooks und von der gleichen Kategorie gibt es dann noch einmal drei Ultrabooks. Zugegeben in dem Beispiel sich ein seltenerer Fall, aber bei Quittungen an Supermarktkassen schon häufiger zu sehen.

Die gleichen Waren fasst das folgende Formular in der Rechnung zusammen.

Rechnung_Textfelder_Uebertrag_Seriendruck



1	MainForm (Abfrage: *Rechnung_nicht_gedruckt*)
1	Listenfeld Kunde (Tabelle: *Anschrift*)
1.1	SubForm (Tabelle: *Verkauf*)
1.1	Listenfeld Ware (Tabelle: *Waren*)
1.2	Druck (Abfrage: *Rechnung_Textfelder*)

Makros in Feldeigenschaften		
1.2	Schaltfläche alle offenen Rechnungen drucken (Aktion ausführen) **Eigenschaften → Allgemein → Zusatzinformation:** «Serien-druck»	Module1.*Direktdruck_Start*
1.2	Schaltfläche einzelne Rechnung drucken (Aktion ausführen) **Eigenschaften → Allgemein → Zusatzinformation:** «Einzel-druck»	Module1.*Direktdruck_Start*

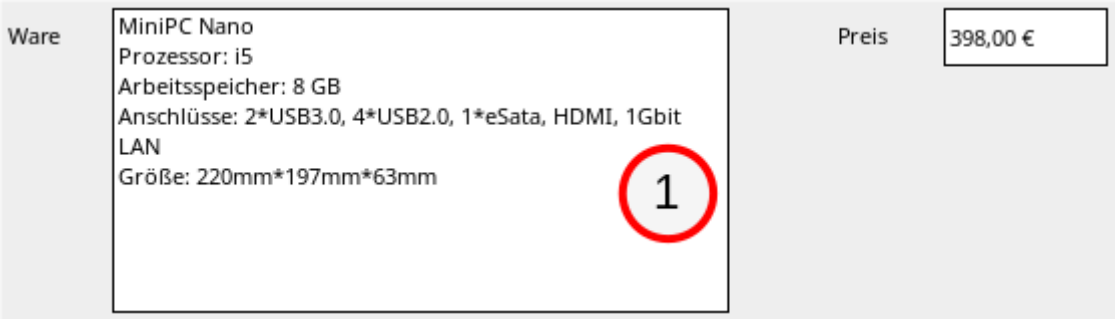
Das Prinzip in diesem Formular ist gleich dem des Formulars «Rechnung_Textfelder_Uebertrag». Hier wird dann allerdings statt der Darstellung des Inhaltes in einem Writerdokument eine *.pdf-Datei für jede Rechnung erstellt, für die noch kein Ausdruck erfolgt ist. Das Formular hat hier lediglich einen zusätzlichen Button, über den der Seriendruck angesteuert wird.

Die Rechnung berücksichtigt dabei außerdem, dass gleiche Waren in einem Feld zusammengefasst werden. Es tauchen also keine «Doppler» mehr auf.

Im Formular werden hier nur die Rechnungen angezeigt, die noch nicht ausgedruckt wurden. Dies wird über Abfragen für das Formular geregelt, die sich lediglich dadurch zusätzlich auszeichnet, dass das "Druckdatum" aus der Tabelle "Rechnung" leer sein muss.

Über den Button «Alle offenen Rechnungen drucken» werden alle Rechnungen, zu denen noch kein Druckdatum existiert, in einzelne PDF-Dateien geschrieben. Das Druckdatum wird dabei in die Tabelle "Rechnung" eingetragen und «MainForm» nach jedem Eintrag aktualisiert, so dass bei Betätigung des Buttons nach und nach alle Einträge aus dem Formular verschwinden und die Dateien anschließend im Verzeichnis der Datenbankdatei liegen - fertig zum Absenden per Mail, zum Ausdruck o.ä.

Waren



1	MainForm (Tabelle: *Waren*)

Das Formular «Waren» wurde nur benötigt, um für die Waren auch eine mehrzeilige Eingabe zu ermöglichen. Es bietet hier lediglich ein mehrzeiliges Textfeld, damit auch Absätze gespeichert werden. Für den Druck hat dieses Formular keine weitere Bedeutung.

Berichte

Diese Beispieldatenbank enthält keine Berichte, die in dem Berichtsordner angezeigt werden. Alle Ausdrucke werden über Makros direkt an Writer weitergeleitet. Die Berichte werden in diesem Fall über Textfelder und Writer-Tabellen erstellt.

Makros

Rechnung

Aufruf aus
Formular: *Rechnung*

Benötigt	
Makro:	*Rechnungsinhalt_zusammenstellen, Output_MailMerge*
Vorlage:	Beispiel_Rechnung.odt
Abfrage:	*Rechnung_einzeilig*

```
001 SUB Rechnung
002    DIM oForm AS OBJECT
003    DIM loFeldID AS LONG
004    DIM loID AS LONG
005    oForm = thisComponent.Drawpage.Forms.MainForm
006    loFeldID = oForm.findColumn("ID")
007    loID = oForm.getLong(loFeldID)
008    Rechnungsinhalt_zusammenstellen(loID)
```

```
009     Output_MailMerge(loID, "Beispiel_Rechnung.odt", 1, "Rechnung_einzeilig")
010 END SUB
```

Aus dem Formular heraus wird der Wert für das Primärschlüsselfeld "ID" ermittelt (Zeile 5 - 7). Anschließend wird die Prozedur «Rechnungsinhalt_zusammenstellen» für den Datensatz mit diesem Feldinhalt gestartet.

Nachdem der Rechnungsinhalt zusammengestellt wurde, wird dieser über die Prozedur «Output_MailMerge» als Serienbrief ausgegeben. Dabei dient das Dokument «Beispiel_Rechnung.odt» als Inhalt für den Serienbrief, dessen Datenbankfelder gefüllt werden. Basis für den Serienbrief ist eine Abfrage (Zahl '1'). Name der Abfrage ist «Rechnung_einzeilig».

Für FIREBIRD muss statt der Abfrage eine Ansicht (Zahl '0') erstellt werden. Sonst stört sich Firebird an den in der Abfrage enthaltenen Tabellen, die alle ein Feld "ID" haben - auch wenn die Abfrage selbst anstandslos ausgeführt wird. Also:

```
011     Output_MailMerge(loID, "Beispiel_Rechnung.odt", 0, "Rechnung_einzeilig")
```

Rechnungsinhalt_zusammenstellen

Aufruf aus
Makro: *Rechnung*

Benötigt
Tabelle: *Verkauf, Waren, Rechnungsinhalt*

```
001 SUB Rechnungsinhalt_zusammenstellen(loID AS LONG)
002    DIM oDatenquelle AS OBJECT
003    DIM oVerbindung AS OBJECT
004    DIM oSQL_Anweisung AS OBJECT
005    DIM oAbfrageergebnis AS OBJECT
006    DIM stSql AS STRING
007    DIM stText AS STRING
008    oDatenquelle = ThisComponent.Parent.CurrentController
009    If NOT (oDatenquelle.isConnected()) THEN
010       oDatenquelle.connect()
011    END IF
012    oVerbindung = oDatenquelle.ActiveConnection()
013    oSQL_Anweisung = oVerbindung.createStatement()
014    stSql = "SELECT ""Verkauf"".""Anzahl"" || CHAR( 9 ) || ""Waren"".""Ware"" ||
          CHAR( 9 ) || REPLACE( ""Waren"".""Preis"" || ' €', '.', ',' ) || CHAR( 9 ) ||
          REPLACE( ""Verkauf"".""Anzahl"" * ""Preis"" || ' €', '.', ',' ) ||CHAR( 13 )
          ||CHAR( 10 ) FROM ""Verkauf"", ""Waren"" WHERE ""Verkauf"".""Waren_ID"" =
          ""Waren"".""ID"" AND ""Verkauf"".""Rechnung_ID"" = "+loID+""
015    oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
016    WHILE oAbfrageergebnis.next
017       stText = stText + oAbfrageergebnis.getString(1)
018    WEND
019    stSql = "DELETE FROM ""Rechnungsinhalt"" WHERE ""ID"" = "+loID+""
020    oSQL_Anweisung.executeUpdate(stSql)
021    stSql = "INSERT INTO ""Rechnungsinhalt"" (""ID"",""Rechnungsinhalt"") VALUES
          ("+loID+", '"+stText+"')"
022    oSQL_Anweisung.executeUpdate(stSql)
023 END SUB
```

Die HSQLDB bietet keine Funktion wie GroupConcat (MySQL) oder List (Firebird). Deswegen werden in dieser Prozedur alle Inhalte, die in der Rechnung auftauchen sollen, zusammen in ein Feld der Tabelle "Rechnungsinhalt" geschrieben.

Der Kontakt zur Datenbank wird in den Zeilen 8 - 12 erstellt. Anschließend wird ein einem SQL-String der gesamte Inhalt, der aus einem Datensatz in der Rechnung erscheinen soll, in einem Feld zusammengefasst. Mit **`CHAR(9)`** wird ein Tabulator zwischen die Feldinhalte gesetzt. Bei Preisangaben muss außerdem das Eurozeichen ergänzt sowie aus dem Dezimalpunkt der

Datenbank ein Dezimalkomma gemacht werden. **CHAR(13)||CHAR(10)** sorgt bei allen Systemen dafür, dass zum Schluss ein Zeilenumbruch erfolgt.

Bei FIREBIRD müssen hier wieder die **CHAR( )** - Funktionen durch **ASCII_CHAR( )**- Funktionen ersetzt werden. Außerdem muss die Multiplikation **(""Verkauf"".""Anzahl"" * ""Preis"")** in Klammern gesetzt werden, da sie sonst als String interpretiert wird.

Serienbrief

Aufruf aus
Formular: *Anschrift*

Benötigt	
Makro:	*Output_MailMerge*
Vorlage:	Beispiel_Serienbrief.odt

```
001 SUB Serienbrief
002    DIM oForm AS OBJECT
003    DIM loFeldID AS LONG
004    DIM loID AS LONG
005    oForm = thisComponent.Drawpage.Forms.MainForm
006    loFeldID = oForm.findColumn("ID")
007    loID = oForm.getLong(loFeldID)
008    Output_MailMerge(loID, "Beispiel_Serienbrief.odt", 0, "Anschrift")
009 END SUB
```

Diese Prozedur ist in vielen Teilen gleich der Prozedur «Rechnung».

Es wird zuerst der Wert des Primärschlüssels "ID" über das Formular ausgelesen. Danach braucht beim Serienbrief allerdings nicht mehr eine Rechnung zusammengestellt zu werden. Stattdessen wird direkt MailMerge gestartet.

An die Prozedur «Output_MailMerge» wird an Variablen zuerst der Primärschlüsselwert und das Dokument mit den Datenbankfeldern «Beispiel_Serienbrief.odt» weitergegeben. Mit '0' wird vermittelt, dass es sich bei dem Eintrag «Anschrift» um eine Tabelle als Datenquelle handelt.

Output_MailMerge

Aufruf aus
Makro: *Rechnung, Serienbrief*

```
001 SUB Output_MailMerge(loID AS LONG, stFile AS STRING, inCommandType AS INTEGER,
       stCommand AS STRING)
002    DIM oDB AS OBJECT
003    DIM stDir AS STRING
004    DIM arProps()
005    MailMerge = CreateUnoService("com.sun.star.text.MailMerge")
006    oDB = ThisComponent.Parent
007    stDir = Left(oDB.Location,Len(oDB.Location)-Len(oDB.Title))
008    WITH MailMerge
009       .DataSourceName = "Beispiel_Datenbank_Serienbrief_direkt"
010       .DocumentURL = stDir + stFile
011       .CommandType = inCommandType
012       .Command = stCommand
013       .Filter = "ID ="+loID
014       .SaveAsSingleFile = true
015       .FileNameFromColumn = true
016       .FileNamePrefix = "Nachname"
017       .OutputType = 2
018       .OutputUrl = stDir
```

```
019     END WITH
020     MailMerge.execute(arProps())
021 END SUB
```

Die Prozedur «Output_MailMerge» wird mit verschiedenen Parametern über die Startprozeduren versorgt. In der Variablen **loID** steckt der Primärschlüssel des Datensatzes, der gedruckt werden soll. Die Variable **stFile** gibt an, wie die Datei heißt, in der sich die Datenbankfelder für den Druck befinden. Als numerischer Wert wird mit **inCommandType** weiter gegeben, ob es sich bei dem SQL-Kommando um eine Tabelle (0), eine Abfrage (1) oder um SQL-Code (2) handelt. **stCommand** schließlich beinhaltet entweder den Namen der Tabelle oder Abfrage oder den SQL-Code.

Zuerst wird der **UnoService MailMerge** mit einer Variablen erstellt, an die alle weiteren Parameter weitergegeben werden.

In Zeile 6 und 7 wird der Pfad zu der aufrufenden Datenbank ausgelesen. In diesem Pfad befinden sich auch die Dateien, die mit Inhalt gefüllt werden sollen.

Anschließend werden von Zeile 8 bis Zeile 19 die verschiedenen Variablen für **MailMerge** aufgelistet, die zum Schluss bei der Ausführung von **MailMerge** (Zeile 20) die Ausgabe bestimmen.

Zeile 9 enthält den Datenbanknamen, wie er in LibreOffice unter **Extras → Optionen → LibreOffice Base → Datenbanken → Registrierter Name** aufgeführt ist.

Zeile 10 gibt den Pfad zu der Datei weiter, die als **stFile** der Prozedur «Output_MailMerge» übergeben wurde.

Die Zeile 11 enthält die Information, ob es sich um eine Tabelle, Abfrage oder einen reinen SQL-Code handelt. In Zeile 12 ist dann entweder der Name für die Tabelle oder Abfrage enthalten oder eben der SQL-Code.

Aus der Datenquelle soll nur ein bestimmter Datensatz gedruckt werden. Deswegen ist in Zeile 13 ein Filter nach dem Primärschlüssel eingebaut.

Das «Druckergebnis» soll nicht über den Drucker direkt herausgehen, sondern als einzelne Datei gespeichert werden (Zeile 14). Der Dateiname soll dabei anhand eines Feldes der Datenquelle (Zeile 15), in diesem Fall des Feldes "Nachname" (Zeile 16) erstellt werden.

Zeile 17 gibt dann den Ausgabetyp von MailMerge wieder. Hier stehen für den Drucker die '1', für die Ausgabe in eine Datei '2' und für die Weitergabe als E-Mail eine '3'. Da in der Prozedur bereits festgelegt wurde, wie der Dateiname heißen soll, steht hier natürlich eine '2'. Bisher ist noch nicht festgelegt, welcher Pfad zu der Datei führen sollte. Der Einfachheit halber ist hier der Pfad zu der Datenbankdatei auch als Pfad für die abzuspeichernde Datei gewählt worden (Zeile 18). In der Praxis wird hier vielleicht eher ein Unterverzeichnis der Datenbankdatei angegeben, so dass alle Briefausgaben von anderen Inhalten getrennt aufbewahrt werden.

Textfelder_Fuellen

Aufruf aus
Formular: *Anschrift_Textfelder*

Benötigt	
Makro:	*Textfelder_Fuellen_ID*
Vorlage:	Beispiel_Textfelder.ott

```
001 SUB Textfelder_Fuellen(oEvent AS OBJECT)
002     DIM oForm AS OBJECT
003     DIM oDB AS OBJECT
004     DIM oNewDoc AS OBJECT
005     DIM stDir AS STRING
006     oForm = oEvent.Source.Model.Parent
007     oDB = ThisComponent.Parent
```

```
008     stDir = Left(oDB.Location,Len(oDB.Location)-Len(oDB.Title))
009     stDir = stDir & "Beispiel_Textfelder.ott"
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)
014     Textfelder_fuellen_ID(oNewDoc, oForm)
015 END SUB
```

Der Kontakt zum Formular wird über den auslösenden Button hergestellt (Zeile 6). Der Pfad zur Datenbank wird ausgelesen, der Dateiname davon abgetrennt und dann mit der Vorlagedatei «Beispiel_Textfelder.ott» verknüpft (Zeile 7 bis 9).

Anschließend wird eine neue Datei auf Basis dieser Vorlage erstellt, die dann bis zur Speicherung die Bezeichnung «Unbenannt.odt» hat (Zeile 10 bis 13).

In Zeile 14 wird die Funktion «Textfelder_Fuellen_ID» mit den Parametern für die neue Datei und das Formular aufgerufen. Der Rückgabewert dieser Funktion wird für die Ausfüllung der Textfelder nicht benötigt. Würde nur die Funktionalität des Formulars «Anschrift_Textfelder» benötigt, so könnte die Funktion «Textfelder_Fuellen_ID» auch direkt hier integriert werden.

Textfelder_Fuellen_ID

Aufruf aus	
Makro:	*Textfelder_Fuellen, Rechnungsinhalt_zusammenstellen_Tabelle, Rechnungsinhalt_zusammenstellen_Tabelle_Uebertrag, Rechnungsinhalt_zusammenstellen_Tabelle_Uebertrag_Direktdruck*

```
001 FUNCTION Textfelder_Fuellen_ID(oNewDoc AS OBJECT, oForm AS OBJECT) AS LONG
002    DIM oColumns AS OBJECT
003    DIM oTextfields AS OBJECT
004    DIM oTextfield AS OBJECT
005    DIM stColumnname AS STRING
006    DIM stDir AS STRING
007    DIM inIndex AS INTEGER
008    DIM loID AS LONG
009    oColumns = oForm.Columns
010    oTextfields = oNewDoc.Textfields.createEnumeration
011    DO WHILE oTextfields.hasMoreElements
012       oTextfield = oTextfields.nextElement
013       IF oTextfield.supportsService("com.sun.star.text.TextField.JumpEdit") THEN
014          stColumnname = oTextfield.PlaceHolder
015          IF oColumns.hasByName(stColumnname) THEN
016             inIndex = oForm.findColumn(stColumnname)
017             oTextfield.Anchor.String = oForm.getString(inIndex)
018          END IF
019       END IF
020    LOOP
021    loID = oForm.getLong(oForm.findColumn("ID"))
022    Textfelder_fuellen_ID = loID
023 END SUB
```

Die Funktion «Textfelder_Fuellen_ID» ist aus den anderen Prozeduren ausgegliedert, da sie sonst gleich mehrfach vorkäme und Nachbesserungen an so einer Prozedur gegebenenfalls in allen anderen Prozeduren auf gleiche Weise erfolgen müssten.

Die Spalten der Datenquelle des Formulars werden in der Variablen **oColumns** gespeichert (Zeile 9)

Die Vorlage mit den Textfeldern ist in ein neues Dokument **oNewDoc** geladen worden. Die Textfelder aus dem Dokument werden ausgelesen (Zeile 10). Zu beachten ist, dass hier für die Gesamtzahl der Felder die Variable **oTextfields** (mit einem «s»), für das einzelne Textfeld **oTextfield** gewählt wurde.

In einer Schleife von Zeile 11 bis 20 werden alle Textfelder des Dokuments angesteuert. Die Textfelder haben dabei eine Bezeichnung, den **Placeholder**. Ist dieser gleich der Bezeichnung einer der Spalten der Datenquelle des Formulars, so liegen dafür auch Daten vor (Zeile 15). Bei gleichen Bezeichnungen wird aus der Datenquelle des Formulars der entsprechende String ausgelesen und als String in das Textfeld eingefügt (Zeile 17).

Nachdem die Schleife durchlaufen worden ist wird noch der Primärschlüssel "ID" aus dem aktuellen Datensatz ausgelesen (Zeile 21) und als Wert der Funktion an die auslösende Prozedur zurückgegeben (Zeile 22). Dies wird in den folgenden Prozeduren für den Ausdruck von Rechnungen benötigt, die in Abhängigkeit von diesem Schlüsselwert die Rechnungsposten zusammenstellen.

Tabellen_fuellen

Aufruf aus
Makro: *Rechnungsinhalt_zusammenstellen_Tabelle*

Benötigt
Tabelle: *Rechnungsinhalt*

```
001 SUB Tabellen_fuellen(oNewDoc AS OBJECT, stSql AS STRING)
002    DIM oDatenquelle AS OBJECT
003    DIM oVerbindung AS OBJECT
004    DIM oSQL_Anweisung AS OBJECT
005    DIM oAbfrageergebnis AS OBJECT
006    DIM oTabelle AS OBJECT
007    DIM oTabellen AS OBJECT
008    DIM oRows AS OBJECT
009    DIM i AS INTEGER
010    oTabellen = oNewDoc.getTextTables
011    oTabelle = oTabellen.getByName("Rechnungsinhalt")
012    oDatenquelle = ThisComponent.Parent.CurrentController
013    If NOT (oDatenquelle.isConnected()) THEN
014       oDatenquelle.connect()
015    END IF
016    oVerbindung = oDatenquelle.ActiveConnection()
017    oSQL_Anweisung = oVerbindung.createStatement()
018    oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
019    WHILE oAbfrageergebnis.next
020       oTabelle.getCellByPosition(0,i).setValue(oAbfrageergebnis.getInt(1))
021       oTabelle.getCellByPosition(1,i).setString(oAbfrageergebnis.getString(2))
022       oTabelle.getCellByPosition(2,i).setValue(oAbfrageergebnis.getDouble(3))
023       oTabelle.getCellByPosition(3,i).setValue(oAbfrageergebnis.getDouble(4))
024       oRows = oTabelle.getrows()
025       oRows.insertByIndex(oRows.getCount(),1)
026       i = i + 1
027    WEND
028    oRows.removeByIndex(oRows.getCount()-1,1)
029 END SUB
```

In den Vorlagen für die Textfelder existiert für die Erfassung der Rechnungsinhalte eine Tabelle. Diese Tabelle ist über **Tabelleneigenschaften → Name** mit dem Namen «Rechnungsinhalt» versehen worden, so dass sie per Makro sicher angesteuert werden kann (Zeile 11).

Dieser Prozedur wird beim Aufruf aus einer anderen Prozedur über **stSql** ein SQL-Code für den Inhalt der Rechnungstabelle mitgegeben. Dabei enthält immer das erste Feld die Anzahl der Gegenstände, das zweite Feld die Benennung der Ware, das dritte Feld den Einzelpreis und das vierte Feld den Gesamtpreis für alle gleichen Waren.

Diese Abfrage wird in Zeile 18 ausgeführt und anschließend in die Tabelle Zeile für Zeile in einer Schleife von Zeile 19 bis 27 eingefügt.

Da bereits eine Zeile in der Tabelle existiert, können zuerst die Feldinhalte in die jeweiligen Zellen übertragen werden. Die Zellbezeichnungen geben mit **`getCellByPosition()`** zuerst die Spalte und dann die Zeile an. Links oben in der Tabelle befindet sich also das Feld mit der Position (0,0).

Sind alle Inhalte einer Datenzeile eingefügt, so wird zunächst gezählt, wie viele Zeilen die Tabelle bereits hat (Zeile 24). Anschließend wird unterhalb der letzten Tabellenzeile eine neue Zeile eingefügt und der Zähler für die Schleife um '1' erhöht.

Nachdem alle Inhalte in die Tabelle eingetragen wurden ist aufgrund der Schleifenkonstruktion in die Tabelle eine Tabellenzeile zu viel eingetragen wurden. Diese Tabellenzeile wird anschließend in Zeile 28 wieder entfernt.

Rechnungsinhalt_zusammenstellen_Tabelle

Aufruf aus
Formular: *Rechnung_Textfelder*

Benötigt	
Tabelle:	*Verkauf, Waren*
Makro:	*Textfelder_Fuellen_ID, Tabellen_fuellen*
Vorlage:	Beispiel_Rechnung_Textfelder.ott

```
001 SUB Rechnungsinhalt_zusammenstellen_Tabelle(oEvent AS OBJECT)
002    DIM oForm AS OBJECT
003    DIM oDB AS OBJECT
004    DIM oNewDoc AS OBJECT
005    DIM stDir AS STRING
006    DIM stSql AS STRING
007    oForm = oEvent.Source.Model.Parent
008    oForm.reload()
009    oForm.last()
010    oDB = ThisComponent.Parent
011    stDir = Left(oDB.Location,Len(oDB.Location)-Len(oDB.Title))
012    stDir = stDir & "Beispiel_Rechnung_Textfelder.ott"
013    DIM args(0) AS NEW com.sun.star.beans.PropertyValue
014    args(0).Name = "AsTemplate"
015    args(0).Value = True
016    oNewDoc = StarDesktop.loadComponentFromURL(stDir,"_blank",0,args)
017    loID = Textfelder_fuellen_ID(oNewDoc, oForm)
018    stSql = "SELECT ""Verkauf"".""Anzahl"", ""Waren"".""Ware"", ""Waren"".""Preis"",
          ""Verkauf"".""Anzahl"" * ""Waren"".""Preis"" FROM ""Verkauf"", ""Waren""
          WHERE ""Verkauf"".""Waren_ID"" = ""Waren"".""ID""
          AND ""Verkauf"".""Rechnung_ID"" = "+loID+""
019    Tabellen_fuellen(oNewDoc, stSql)
020 END SUB
```

Über das auslösende Ereignis des Buttons wird das Formular ermittelt (Zeile 7). Das Formular ist Unterformular für die eigentliche Eingabe der Rechnungsposten. Nach der letzten Eingabe muss in diesem Unterformular der Inhalt aktualisiert werden, da sonst die angezeigten Summen nicht stimmen könnten (Zeile 8 und 9).

Der Pfad zu der Vorlage wird über den Pfad zu der Datenbankdatei bestimmt (Zeile 10 bis 12). Danach wird ein neues Dokument «Unbenannt.odt» aufgrund dieser Vorlage erstellt (Zeile 13 bis 16).

In dem neuen Dokument werden zuerst die Textfelder über die Funktion «Textfelder_fuellen_ID» gefüllt. Die Funktion gibt hier den Primärschlüssel für die entsprechende Rechnung zurück (Zeile 17).

Zum Schluss wird der SQL-Code für die Rechnungsinhalte erstellt. Dafür ist der Primärschlüssel aus der vorhergehenden Funktion wichtig, damit nur die Rechnungsposten zugeordnet werden, die mit der aktuellen Rechnung zu tun haben (Zeile 18). Dieser Code wird zusammen mit dem Objekt des Dokumentes «Unbenannt.odt» an die Prozedur «Tabellen_fuellen» weitergeleitet. Die Rechnung ist danach erstellt und kann ausgedruckt oder abgespeichert werden.

Tabellen_fuellen_Uebertrag

Aufruf aus
Makro: *Rechnungsinhalt_zusammenstellen_Tabelle_Uebertrag, Rechnungsinhalt_zusammenstellen_Tabelle_Uebertrag_Direktdruck*

Benötigt
Tabelle: *Rechnungsinhalt*

```
001 SUB Tabellen_fuellen_Uebertrag(oNewDoc AS OBJECT, stSql AS STRING)
002    DIM oTabellen AS OBJECT
003    DIM oTabelle AS OBJECT
004    DIM oDatenquelle AS OBJECT
005    DIM oVerbindung AS OBJECT
006    DIM oSQL_Anweisung AS OBJECT
007    DIM oAbfrageergebnis AS OBJECT
008    DIM oCursor AS OBJECT
009    DIM oTxtCursor AS OBJECT
010    DIM oRows AS OBJECT
011    DIM i AS INTEGER
012    DIM ink AS INTEGER
013    DIM inStartSeite AS INTEGER
014    DIM inFolgezeilen AS INTEGER
015    DIM doUebertrag AS DOUBLE
016    oTabellen = oNewDoc.getTextTables
017    oTabelle = oTabellen.getByName("Rechnungsinhalt")
018    oDatenquelle = ThisComponent.Parent.CurrentController
019    If NOT (oDatenquelle.isConnected()) THEN
020       oDatenquelle.connect()
021    END IF
022    oVerbindung = oDatenquelle.ActiveConnection()
023    oSQL_Anweisung = oVerbindung.createStatement()
024    oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
025    inStartSeite = 1
026    inFolgezeilen = 3
027    oCursor = oNewDoc.CurrentController.ViewCursor
028    WHILE oAbfrageergebnis.next
029       doAnzahlPreis = oAbfrageergebnis.getDouble(4)
030       oTabelle.getCellByPosition(0,i).setValue(oAbfrageergebnis.getInt(1))
031       oTabelle.getCellByPosition(1,i).setString(oAbfrageergebnis.getString(2))
032       oTabelle.getCellByPosition(2,i).setValue(oAbfrageergebnis.getDouble(3))
033       oTabelle.getCellByPosition(3,i).setValue(doAnzahlPreis)
034       doUebertrag = doUebertrag + doAnzahlPreis
035       oRows = oTabelle.getrows()
036       oRows.insertByIndex(oRows.getCount(),1)
037       oCursor.JumpToLastPage()
038       oCursor.JumpToEndOfPage()
039       oCursor.goUp(inFolgezeilen,False)
040       IF oCursor.Page > inStartSeite THEN
041          oCursor.goUp(1,False)
042          FOR ink = 1 TO 2
043             i = i + 1
044             oRows = oTabelle.getRows()
045             oRows.insertByIndex(oRows.getCount()-2,1)
046             oTabelle.getCellByPosition(2,i-1).setString("Übertrag:")
```

```
047             oTxtCursor = oTabelle.getCellByPosition(2,i-1
                   ).createTextCursorByRange(oTabelle.getCellByPosition(2,i-1).text)
048             oTxtCursor.CharPosture = com.sun.star.awt.FontSlant.ITALIC
049             oTabelle.getCellByPosition(3,i-1).setValue(doUebertrag - doAnzahlPreis)
050             oTxtCursor = oTabelle.getCellByPosition(3,i-1
                   ).createTextCursorByRange(oTabelle.getCellByPosition(3,i-1).text)
051             oTxtCursor.CharPosture = com.sun.star.awt.FontSlant.ITALIC
052          NEXT ink
053          oCursor.gotoRange(oTxtCursor,false)
054          Wait(100)
055          DO WHILE oCursor.Page = inStartSeite
056             oRows = oTabelle.getRows()
057             oRows.insertByIndex(oRows.getCount()-3,1)
058             oCursor.gotoRange(oTxtCursor,false)
059             i = i + 1
060          LOOP
061          inStartSeite = inStartSeite + 1
062       END IF
063       i = i + 1
064    WEND
065    oRows.removeByIndex(oRows.getCount()-1,1)
066    oCursor.JumpToLastPage()
067    oCursor.JumpToEndOfPage()
068 END SUB
```

Der grundlegende Aufbau dieser Prozedur ist wie der Aufbau der Prozedur «Tabellen_fuellen». Auch hie wird die Tabelle «Rechnungsinhalt» in dem neuen Dokument aufgesucht und mit Inhalten gefüllt, die aus der Abfrage hervorgehen, die beim Aufruf dieser Prozedur mit übermittelt wird.

Zusätzlich muss hier allerdings mit dem **ViewCursor** gearbeitet werden, damit klar wird, wann ein Seitenumbruch stattfindet. Ein Objekt dieses Cursors wird in Zeile 27 erstellt. Dieser Cursor wird nach dem Einfügen von Daten in die Tabelle immer wieder auf die letzte Zeile der letzten Seite eingestellt (Zeile 37 und 38). Anschließend wird er um die Zeilenanzahl zurückgesetzt, die für Elemente außerhalb der Tabelle vorgesehen sind (Zeile 39). Das sind in diesem Fall «Nettobetrag:», «Umsatzsteuer 19%:» und «Insgesamt zu zahlen:», also 3 Zeilen. Steht der Cursor nach diesen Bewegungen auf einer neuen Seite (Zeile 40), so wird das Einfügen eines Übertrags notwendig. Dies wird in der Verzweigung von Zeile 40 bis Zeile 62 erledigt.

Der **ViewCursor** wird zuerst noch eine Zeile weiter nach oben, oberhalb des letzten Eintrags, geschoben. Dann wird der «Übertrag» in einer Schleife von Zeile 42 bis Zeile 52 doppelt eingetragen. Um den Eintrag kursiv zu formatieren ist neben dem **ViewCursor** hier der **TextCursor** notwendig.

Nach der Schleife wird der **ViewCursor** auf die letzte Position des **TextCursors** ausgerichtet. Dies ist notwendig, damit wieder klar wird, auf welcher Seite sich die zweite Zeile des Übertags befindet. Manchmal funktionierte die Positionierung hier nicht einwandfrei. Dies ließ sich durch einen **Wait(100)**-Befehl regulieren, so dass der bei der Abfrage nach der aktuellen Seite in Zeile 55 tatsächlich die aktuelle Seite des Cursors und damit des zweiten Eintrags zum Übertrag mitgeteilt wurde.

Solange der 2. Eintrag des Übertags nicht auf der Folgeseite erscheint wird immer wieder eine Zeile dazwischen geschoben und neu überprüft (Zeile 55 bis 60). Anschließend wird der interne Seitenzähler um '1' erhöht (Zeile 61) und der Zähler für die aktuelle zu befüllende Zeile in der Tabelle ebenfalls um '1' erhöht (Zeile 63). Weitere Einträge in die Rechnungstabelle können vorgenommen werden.

In Zeile 65 wird wieder die letzte Zeile der Tabelle entfernt, da sie ja leer ist. Dann wird der **ViewCursor** an die letzte Position gesetzt, so dass das Dokument immer so erscheint, dass direkt der Rechnungsbetrag sichtbar ist. Hier könnte natürlich auch ein Sprung direkt auf die erste Seite in die erste Zeile erfolgen, wenn dies gewünscht wird.

Rechnungsinhalt_zusammenstellen_Tabelle_Uebertrag

Aufruf aus
Formular: *Rechnung_Textfelder_Uebertrag*

Benötigt	
Tabelle:	*Verkauf, Waren*
Makro:	*Textfelder_Fuellen_ID, Tabellen_fuellen_Uebertrag*
Vorlage:	Beispiel_Rechnung_Textfelder.ott

```
001  SUB Rechnungsinhalt_zusammenstellen_Tabelle_Uebertrag(oEvent AS OBJECT)
002     DIM oForm AS OBJECT
003     DIM oDB AS OBJECT
004     DIM oNewDoc AS OBJECT
005     DIM stDir AS STRING
006     DIM stSql AS STRING
007     DIM loID AS LONG
008     oForm = oEvent.Source.Model.Parent
009     oForm.reload()
010     oForm.last()
011     oDB = ThisComponent.Parent
012     stDir = Left(oDB.Location,Len(oDB.Location)-Len(oDB.Title))
013     stDir = stDir & "Beispiel_Rechnung_Textfelder.ott"
014     DIM args(0) AS NEW com.sun.star.beans.PropertyValue
015     args(0).Name = "AsTemplate"
016     args(0).Value = True
017     oNewDoc = StarDesktop.loadComponentFromURL(stDir,"_blank",0,args)
018     loID = Textfelder_fuellen_ID(oNewDoc, oForm)
019     stSql = "SELECT ""Verkauf"".""Anzahl"", ""Waren"".""Ware"", ""Waren"".""Preis"",
           ""Verkauf"".""Anzahl"" * ""Waren"".""Preis"" FROM ""Verkauf"", ""Waren""
           WHERE ""Verkauf"".""Waren_ID"" = ""Waren"".""ID""
           AND ""Verkauf"".""Rechnung_ID"" = "+loID+""
020     Tabellen_fuellen_Uebertrag(oNewDoc, stSql)
021  END SUB
```

Das Formular wird wieder über das auslösende Ereignis (Betätigung des Druck-Buttons) bestimmt (Zeile 8). Der Pfad wird über die Datenbank ermittelt. Statt der Datenbankdatei wird die Vorlage «Beispiel_Rechnung_Textfelder.ott» an den Pfad angehängt (Zeile 11 bis 13).

Anschließend wird das neue Dokument auf Basis dieser Vorlage geöffnet und die Textfelder mit dem entsprechenden Inhalt über die Funktion «Textfelder_fuellen_ID» versehen. Die Funktion gibt dabei den Primärschlüsselwert für die Rechnung zurück (Zeilen 17 und 18).

Schließlich wird der SQL-Befehl für das Füllen des Dokuments mit den Rechnungsposten erstellt und über die Prozedur «Tabelle_fuellen_Uebertrag» weiter verarbeitet, so dass zum Schluss ein fertiges Writer-Dokument vorliegt.

Rechnungsinhalt_zusammenstellen_Tabelle_Uebertrag_Direktdruck

Aufruf aus
Makro: *Direktdruck_Start*

Benötigt	
Tabelle:	*Verkauf, Waren*
Makro:	*Textfelder_Fuellen_ID, Tabellen_fuellen_Uebertrag, PDF_Export*
Vorlage:	Beispiel_Rechnung_Textfelder.ott

```
001  SUB Rechnungsinhalt_zusammenstellen_Tabelle_Uebertrag_Direktdruck(oEvent AS OBJECT)
```

```
002     DIM oForm AS OBJECT
003     DIM oDB AS OBJECT
004     DIM oNewDoc AS OBJECT
005     DIM stDir AS STRING
006     DIM stPrintDir AS STRING
007     DIM stSql AS STRING
008     DIM loID AS LONG
009     oForm = oEvent.Source.Model.Parent
010     oForm.reload()
011     oForm.last()
012     oDB = ThisComponent.Parent
013     stDir = Left(oDB.Location,Len(oDB.Location)-Len(oDB.Title))
014     stPrintDir = stDir
015     stDir = stDir & "Beispiel_Rechnung_Textfelder.ott"
016     DIM args(1) AS NEW com.sun.star.beans.PropertyValue
017     args(0).Name = "AsTemplate"
018     args(0).Value = True
019     args(1).Name = "Hidden"
020     args(1).Value = True
021     oNewDoc = StarDesktop.loadComponentFromURL(stDir,"_blank",0,args)
022     loID = Textfelder_fuellen_ID(oNewDoc, oForm)
023     stSql = "SELECT SUM(""Verkauf"".""Anzahl"") AS ""SAnzahl"", ""Waren"".""Ware"",
           ""Waren"".""Preis"", SUM(""Verkauf"".""Anzahl"" * ""Waren"".""Preis"") AS
           ""SGesamt"" FROM ""Verkauf"", ""Waren"" WHERE ""Verkauf"".""Waren_ID"" =
           ""Waren"".""ID"" AND ""Verkauf"".""Rechnung_ID"" = "+loID+" GROUP BY
           ""Waren"".""Ware"", ""Waren"".""Preis"""
024     Tabellen_fuellen_Uebertrag(oNewDoc, stSql)
025     PDF_Export(oNewDoc, loID, stPrintDir)
026 END SUB
```

Diese Prozedur unterschiedet sich in zwei Punkten von der Prozedur «Rechnungsinhalt_zusammenstellen_Tabelle_Uebertrag».

- Die Inhalte der Rechnung werden vor dem Übertrag in das Dokument zusammengefasst, so dass gleiche Rechnungsposten nicht mehrfach in der Rechnung auftauchen. Dies wird durch die Gruppierung in der Abfrage (Zeile 23) erreicht.
- Die Rechnung selbst wird im Hintergrund erstellt (Zeile 19 und 20) und anschließend an die Prozedur «PDF_Export» weitergeleitet. Dem Export-Befehl wird ein Pfad mitgegeben, unter dem die PDF-Dateien abgespeichert werden sollen. Es ist hier der gleiche Pfad wie der, in dem sich die Datenbankdatei befindet.

PDF_Export

Aufruf aus
Makro: *Rechnungsinhalt_zusammenstellen_Tabelle_Uebertrag_Direktdruck*

Benötigt
Tabelle: *Rechnung*

```
001 SUB PDF_Export(oNewDoc AS OBJECT, loID AS LONG, stPrintDir AS STRING)
002     DIM oDatenquelle AS OBJECT
003     DIM oVerbindung AS OBJECT
004     DIM oSQL_Anweisung AS OBJECT
005     DIM stDatei AS STRING
006     DIM stSql AS STRING
007     stDatei = "RE_" & Right("0000000" & loID, 8) & ".pdf"
008     stPrintDir = stPrintdir & stDatei
009     DIM arg(0) AS NEW com.sun.star.beans.PropertyValue
010     arg(0).name = "FilterName"
011     arg(0).value = "writer_pdf_Export"
012     oNewDoc.storeToURL(stPrintDir, arg())
013     oNewDoc.close(true)
```

```
014     oDatenquelle = thisDatabaseDocument.CurrentController
015     IF NOT (oDatenquelle.isConnected()) THEN oDatenquelle.connect()
016     oVerbindung = oDatenquelle.ActiveConnection()
017     oSQL_Anweisung = oVerbindung.createStatement()
018     stSql = "UPDATE ""Rechnung"" SET ""Druckdatum"" = CURRENT_DATE WHERE ""ID"" =
           "+loID
019     oSQL_Anweisung.executeUpdate(stSql)
020 END SUB
```

Zuerst wird der abzuspeichernden Datei im Dateinamen die Rechnungsnummer zugewiesen Zeile 7). Damit sind alle Dateien unterscheidbar.

Die Datei wird mittels des *.pdf-Exports (Zeile 11) erstellt und in dem entsprechenden Pfad abgespeichert sowie geschlossen (Zeile 12).

Damit eine Rechnung nicht mehrmals ausgedruckt wird, wird das aktuelle Druckdatum anschließend in der Tabelle "Rechnung" eingetragen. So können dann auch alle offenen Rechnungen über eine Schleife gedruckt werden. Es wird dann einfach jede Rechnung gedruckt, die noch nicht mit einem Druckdatum versehen ist.

Direktdruck_Start

Aufruf aus
Formular: *Rechnung_Textfelder_Uebertrag_Seriendruck*

Benötigt
Makro: *Rechnungsinhalt_zusammenstellen_Tabelle_Uebertrag_Direktdruck*

```
001 SUB Direktdruck_Start(oEvent AS OBJECT)
002    DIM oMainForm AS OBJECT
003    DIM stSerie AS STRING
004    DIM stIDField AS STRING
005    DIM stID AS STRING
006    stSerie = oEvent.Source.Model.Tag
007    oMainForm = oEvent.Source.Model.Parent.Parent
008    stIDField = oMainForm.findColumn("ID")
009    stID = oMainForm.getString(stIDField)
010    IF stSerie = "Seriendruck" THEN
011       DO WHILE stID <> ""
012          Rechnungsinhalt_zusammenstellen_Tabelle_Uebertrag_Direktdruck
013          oMainForm.reload()
014          stID = oMainForm.getString(stIDField)
015       LOOP
016    ELSE
017       IF stID <> "" THEN
018          Rechnungsinhalt_zusammenstellen_Tabelle_Uebertrag_Direktdruck
019          oMainForm.reload()
020       END IF
021    END IF
022 END SUB
```

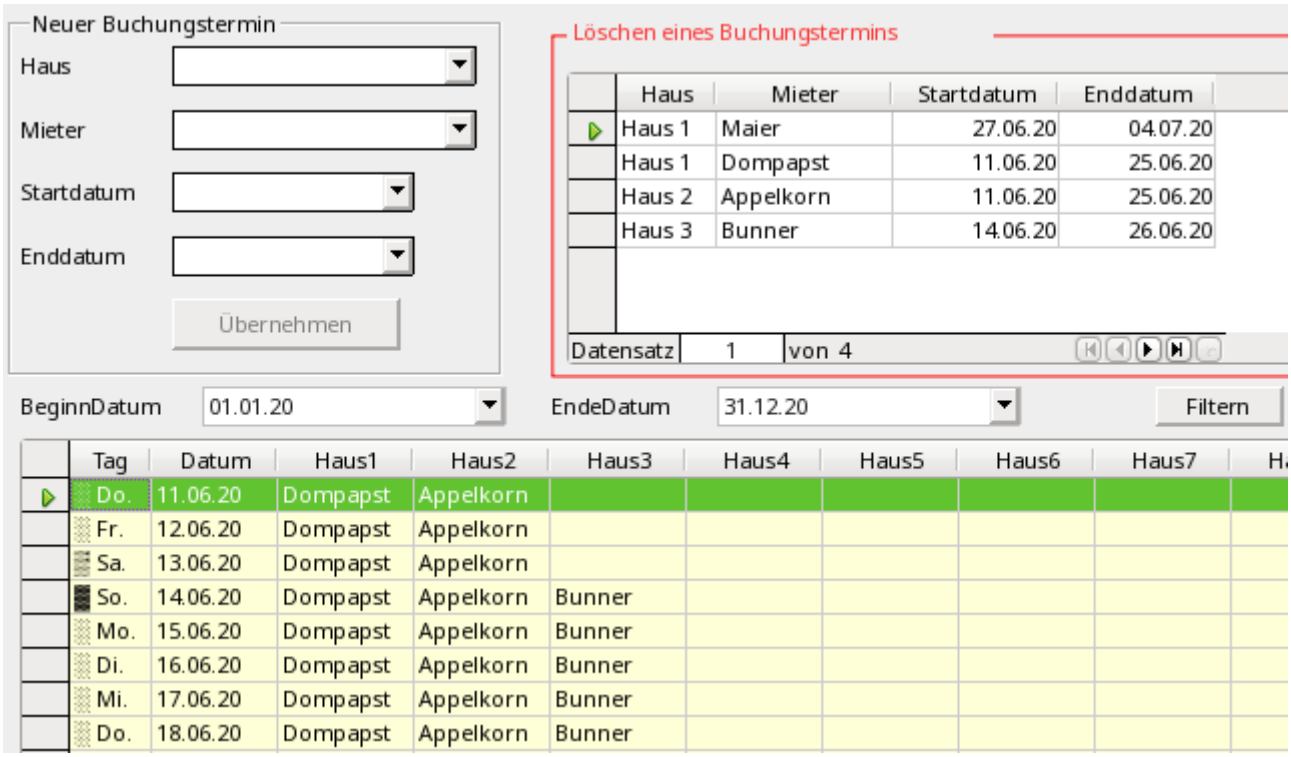
Der Direktdruck wird über diese separate Prozedur gestartet. Dies ist erforderlich, weil das Formular neben dem Druck einer einzelnen Rechnung auch den Druck aller offenen Rechnungen ermöglichen soll.

Steht in dem auslösenden Button in den Zusatzinformationen (Zeile 6) der Begriff «Seriendruck» (Zeile 10), so wird zuerst der Druck gestartet, dann das darüber liegende Formular (**Parent.Parent** Zeile 7) wieder neu geladen und der Primärschlüsselwert aus dem jetzt geladenen Formular wieder ausgelesen. Das Neuladen ist notwendig, da in dem Formular nur die Rechnungen angezeigt werden, bei denen das Druckdatum noch leer ist.

Handelt es sich nicht um einen Seriendruck und ist das Feld für den Primärschlüssel nicht leer, so wird stattdessen nur die Rechnung zum aktuellen Datensatz gedruckt (Zeile 18).

Terminübersicht Ferienhäuser

Einführung



Die einfache Eingabe von Daten für die Belegung eines Ferienhauses ist in Base problemlos möglich. Wie aber erhalte ich einen Überblick, zu welchen Terminen das Ferienhaus noch frei ist? Und wie erst, wenn es sich gleich um mehrere Häuser handelt? Dazu muss eine Jahresübersicht erstellt werden, die am besten alle Datumswerte des Jahres enthält. Wie gebe ich jetzt sicher ein, dass ein Ferienhaus nicht doppelt belegt wird. Auch hierzu bietet das Formular dieser Datenbank einen Lösungsvorschlag[5].

Tabellen

Die Tabellen decken nur den mindestens nötigen Inhalt ab, um die Organisation der Vermietungstermine zu demonstrieren. Eine Adressverwaltung fehlt ebenso wie eine Auflistung der Kosten o.ä.

Mieter

Datenziel
Abfrage: *Belegung, Bericht*
Formular, Ansicht, Bericht, Makro: keine direkt

Feldname	***Feldtyp***	***Beschreibung***
ID	Integer	Primärschlüssel der Tabelle. Der Primärschlüssel muss eindeutig sein. Das Feld kann als Auto-Wert-Feld gesetzt werden.
Name	Text	Name des Mieters. Eingabe erforderlich: Ja

5 Beispieldatenbank Beispiel_Terminuebersicht_Ferienhausvermietung.odb

Diese Tabelle enthält in diesem Fall lediglich die Namen der Mieter. Ein Formular, das diese Tabelle beschickt, ist nicht vorgesehen. Die wenigen Mieter, die zu Demonstrationszwecken benötigt werden, wurden direkt in die Tabelle eingegeben. Deswegen enthält die Tabelle auch kein Auto-Wert-Feld als Primärschlüssel.

Haus

Datenziel
Abfrage: *Bericht*
Formular, Ansicht, Bericht, Makro: keine direkt

Feldname	***Feldtyp***	***Beschreibung***
ID	Integer	Primärschlüssel der Tabelle. Der Primärschlüssel muss eindeutig sein. Das Feld kann als Auto-Wert-Feld gesetzt werden.
Haus	Text	Bezeichnung des Hauses. Eingabe erforderlich: Ja

Auch die Tabelle "Haus" ist nur mit der Mindestinformation, der Bezeichnung für das Haus, versehen. Da die vorgesehenen Häuser begrenzt sind ist auch hier keine Auto-Wert-Feld für den Primärschlüssel vorgesehen.

Reservierung

Datenziel
Formular: *Haus*
Abfrage: *Belegung, Bericht*
Makro: *Startdatum, Enddatum*
Ansicht, Bericht: keine direkt

Feldname	***Feldtyp***	***Beschreibung***
Haus_ID	Integer	Fremdschlüssel der Tabelle "Haus". Gleichzeitig einer der drei Primärschlüssel dieser Tabelle.
Mieter_ID	Integer	Fremdschlüssel der Tabelle "Mieter". Gleichzeitig einer der drei Primärschlüssel dieser Tabelle.
Startdatum	Datum	Nimmt den Beginn der Mietperiode auf. Muss zusammen mit den vorhergehenden Feldern ein Primärschlüssel sein, da sonst ein Mieter nicht zweimal das gleiche Haus mieten kann.
Enddatum	Datum	Hier wird das Ende der Mietperiode verzeichnet. Eingabe erforderlich: Ja

Die Tabelle "Reservierung" verbindet die Tabelle "Mieter" mit der Tabelle "Haus".

Das Ende der Mietperiode könnte noch mit einer Bedingung so versehen werden, dass das eingegebene Datum auf jeden Fall nach dem Startdatum liegt. Hier wurde darauf verzichtet, da die Eingabe komplett über das Formular gesteuert wird.

Die drei oben genannten Tabellen sind die Tabellen, in denen beständig Eingaben erfolgen sollen. Daneben existiert noch wieder eine Tabelle "Filter", die dazu dient, nach bestimmten Zeiträumen die Eingaben im Formular zu filtern.

Filter

Datenziel
Formular: *Haus*
Ansicht: *Ansicht_Datum*
Abfrage: *Belegung*
Bericht, Makro: keine direkt

Feldname	***Feldtyp***	***Beschreibung***
ID	Ja/Nein	Primärschlüssel der Tabelle, wird zum Start auf 'Ja' gesetzt und danach nicht mehr verändert.
BeginnDatum	Datum	Startdatum für Anzeige in Formular und Bericht
EndeDatum	Datum	Enddatum für Anzeige in Formular und Bericht

Diese Tabelle filtert in dem Formular die Zeitspanne, für die die Termine angezeigt werden.

NrBis31

Datenziel
Ansicht: *Ansicht_Datum*
Formular, Abfrage, Bericht, Makro: keine direkt

Feldname	***Feldtyp***	***Beschreibung***
Nr	TinyInteger	Fortlaufende Nummerierung von 1 bis 31 für die Erstellung von Datumsübersichten, kein Primärschlüssel, dadurch nicht beschreibbar.

Aus der Tabelle "NrBis31" wurde der Primärschlüssel nach der Eingabe der Ziffern entfernt, damit nicht irrtümlich eine der Zahlen gelöscht wird. Solch eine Tabelle kann auch einfach über Calc und die automatische Nummerierung erzeugt werden. Mit Hilfe dieser Tabelle wird eine fortlaufende Datumsübersicht erzeugt.

Für FIREBIRD ist eine andere Abfragetechnik und daher auch eine andere Hilfstabelle notwendig. Die Tabelle «NrBis1100» speichert hier alle Zahlen bis 1100, deckt also 3 Jahre und einige wenige Tage ab.

Ansichten

Ansicht_Datum

Datenquelle
Tabelle: *NrBis31, Filter*

Datenziel
Ansicht: *Ansicht_Datum_lfdNr*
Abfrage: *Belegung*
Makro: *Enddatum*

```
001 SELECT DISTINCT
002 CAST(
    "Y"."Nr" + (SELECT EXTRACT( YEAR FROM
    COALESCE("BeginnDatum",CURRENT_DATE)) FROM "Filter" WHERE "ID" = True) - 2
    || '-' ||
    RIGHT('0'||"M"."Nr",2)|| '-' ||
    RIGHT('0'||"D"."Nr",2)
    AS DATE ) AS "Datum"
003 FROM "NrBis31" AS "D", "NrBis31" AS "M", "NrBis31" AS "Y"
004 WHERE "Y"."Nr" <= 3
005 AND "M"."Nr" <= 12
006 AND "D"."Nr" <= 31
```

Die Tabelle "NrBis31" enthält lediglich die Zahlen von 1 bis 31. Aus diesen Zahlen wird ein Datumswert zusammengesetzt. Das Datum muss dabei in der Reihenfolge YYYY-MM-DD vorliegen. Erst dann kann es durch die HSQLDB in einen Datumswert umgewandelt werden.

Die Tabelle "NrBis31" wird dreimal durch einen Alias eingebunden (Zeile 3). Da zwischen den verschiedenen Alias-Bezeichnungen keine Beziehung erstellt wird ermittelt die Abfrage alle Kombinationen der Werte, die aus der Tabelle zu entnehmen sind.

Das Jahr beginnt mit Werten, die ein Jahr vor dem Jahr liegen, in dem das "BeginnDatum" der Tabelle "Filter" liegt. Bei den Monats- und Tagesbezeichnungen wird jeweils eine '0' vorangestellt und dann die letzten beiden Ziffern für die Verarbeitung zum Datum genutzt. So bleibt die Monats- und die Tageszahl immer zweistellig. Insgesamt ergibt sich daraus ein Datum in Textform, der jetzt noch in einen Datumswert über **`CAST … AS DATE`** umgewandelt wird (Zeile 2).

Es werden die Datumswerte für 3 Jahre ermittelt (Zeile 4). Für den Monat werden die Werte von 1 bis 12 ausgelesen (Zeile 5) und für den Tag die Werte von 1 bis 31 (Zeile 6).

Das Startjahr wird aus dem "BeginnDatum" der Tabelle "Filter" ausgelesen. Das Startjahr wird zu dem jeweiligen Datensatz aus "NrBis31" addiert. Da die Zahlenwerte aus der Tabelle "NrBis31" mit 1 beginnen und die Jahreszahlen 1 Jahr vor dem Jahr beginnen sollen, das aktuell angezeigt wird, wird von dem Betrag des Startjahres 2 subtrahiert.

Durch die Umwandlung erstellt die HSQLDB aus einem Datumswert wie z.B. 31.02.2015 stattdessen den 3.03.2015. Schließlich hat der Februar nicht 31 Tage. Tageswerte über 31 akzeptiert die HSQLDB allerdings nicht. Sonst wäre über diesen Weg auch leicht eine Addition von Tagen wie mit **`DATEADD`** in anderen Datenbanken zu einem Datum möglich. Diese Umwandlung erzeugt jetzt bei einigen Datumswerten doppelte Einträge wie eben bei dem 3.03.2015. Diese Doppler sollen unterbunden werden. Das geschieht schließlich mit dem Zusatz **`DISTINCT`** (Zeile 1).

Die Ansicht wird von der HSQLDB direkt in der korrekten Datumsreihenfolge ausgegeben.

Für FIREBIRD ist diese Ansicht nicht möglich (Fehlermeldung bei fehlerhaften Datumswerten wie '29.02.2019'). Die Ansicht ist aber auch nicht nötig, da Tage zu Datumswerten addiert werden können. Stattdessen wird direkt die Ansicht «Ansicht_Datum_lfdNr» erstellt.

Ansicht_Datum_lfdNr

Datenquelle	
Ansicht:	*Ansicht_Datum*
Abfrage:	*Bericht*
Makro:	*Startdatum*

```
001 SELECT "a"."Datum",
002    (SELECT COUNT(*) FROM "Ansicht_Datum" WHERE "Datum" <= "a"."Datum")
          AS "lfdNr"
003 FROM "Ansicht_Datum" AS "a"
```

Diese Ansicht greift auf die vorher erstellte Ansicht zu. Sie fügt allerdings noch eine Spalte hinzu, die eine fortlaufende Nummerierung der Datumswerte erstellt. Es werden durch eine korrelierende Unterabfrage alle Datumswerte gezählt, die kleiner oder gleich dem aktuellen Datumswert des angezeigten Datensatzes sind.

Eine solche Auflistung mit fortlaufender Nummerierung macht es möglich, zu einem Datum eine bestimmte Anzahl von Tagen zu addieren oder zu subtrahieren und als Ergebnis wieder ein Datum zu erhalten. Hierdurch kann also in Grenzen die Funktion **DATEADD** nachgestellt werden, die es in der alten Version der HSQLDB nicht gibt.

Mit FIREBIRD wird diese Ansicht direkt erstellt:

```
001 SELECT CAST( ( (
        SELECT EXTRACT( YEAR FROM COALESCE ( "BeginnDatum", CURRENT_DATE ) )
        FROM "Filter" WHERE "ID" = TRUE
        ) - 2 ) || '-12-31' AS DATE ) + "Nr" AS "Datum",
002 "Nr" AS "lfdNr"
003 FROM "NrBis1100"
```

Es wird hier das "BeginnDatum" aus der Tabelle "Filter" genommen und davon dann das Jahr extrahiert. Gestartet wird mit dem 31.12. 2 Jahre vor dem "BeginnDatum". Zu diesem datum wird '1' addiert, so dass das erste angezeigte Datum der 1.1. im Jahr vor dem "BeginnDatum" ist (Zeile 1). In Zeile 2 wird einfach die fortlaufende Nummer aus der Tabelle "NrBis1100" übernommen. Die Ansicht in FIREBIRD entspricht dann bis auf ein paar zusätzliche Tage genau der Ansicht, die über Umwege mit der HSQLDB konstruiert wurde.

Abfragen

Die Abfragen greifen auf zwei unterschiedliche Ansichten zu. Die Ansichten dienen dazu, laufende Datumswerte wieder zu geben bzw. auch noch mit einer laufenden Nummerierung zu versehen.

Belegung

Datenquelle	
Tabelle:	*Mieter, Reservierung, Filter*
Ansicht:	*Ansicht_Datum*

Datenziel

Formular: *Haus*

```
001 SELECT CASE WHEN DAYOFWEEK( "a"."Datum" ) = 1 THEN '▓ ' ||
    TO_CHAR( "a"."Datum", 'D' ) WHEN DAYOFWEEK( "a"."Datum" ) = 7 THEN '▒ ' ||
    TO_CHAR( "a"."Datum", 'D' ) ELSE '░ ' || TO_CHAR( "a"."Datum", 'D' ) END
    AS "Wochentag",
002 "a"."Datum",
003 ( SELECT "Mieter"."Name" FROM "Mieter", "Reservierung" WHERE "Mieter"."ID"
    = "Reservierung"."Mieter_ID" AND "Reservierung"."Haus_ID" = 1 AND
    "a"."Datum" BETWEEN "Reservierung"."Startdatum" AND
    "Reservierung"."Enddatum" ) AS "Haus1",
004 ( SELECT "Mieter"."Name" FROM "Mieter", "Reservierung" WHERE "Mieter"."ID"
    = "Reservierung"."Mieter_ID" AND "Reservierung"."Haus_ID" = 2 AND
    "a"."Datum" BETWEEN "Reservierung"."Startdatum" AND
    "Reservierung"."Enddatum" ) AS "Haus2",
005 ...
006 ... AS "Haus8"
007 FROM "Ansicht_Datum" AS "a"
008 WHERE "a"."Datum" BETWEEN
009 COALESCE( ( SELECT "BeginnDatum" FROM "Filter" WHERE "ID" = TRUE ),
    CURRENT_DATE ) AND COALESCE( ( SELECT "EndeDatum" FROM "Filter" WHERE "ID"
    = TRUE ), CURRENT_DATE )
```

In der ersten Spalte (Zeile 1) wird der Wochentag in Kurzschreibweise zusammen mit einem grafischen Element dargestellt, das die Wochenenden kennzeichnen soll. Da diese Abfrage innerhalb eines Tabellenkontrollfeldes genutzt werden soll sind hier die Möglichkeiten zur Kennzeichnung von Wochenenden für eine bessere Übersicht sehr begrenzt.

In der zweiten Spalte wird das Datum aus der "Ansicht_Datum" ausgelesen (Zeile 2). Dabei ist durch Einträge in der Tabelle "Filter" festgelegt, von welchem bis zu welchem Datum die Anzeige erfolgen soll (Zeile 9). Die gesamte Abfrage bezieht sich nur auf "Ansicht_Datum" (Zeile 7).

In einer korrelierenden Unterabfrage wird dann für jedes Haus (Zeilen 3 bis 6) abgefragt, ob das Datum der ersten Abfragespalte Zwischen "Startdatum" und "Enddatum" eines Eintrags zu finden ist, der in der Tabelle "Reservierung" existiert. Dann wird der jeweilige Name des Mieters herausgesucht und in die Übersicht aufgenommen.

Die Abfrage zu Belegung muss in FIREBIRD anders gestellt, weil der Befehl **DAYOFWEEK** unbekannt ist und Firebird zur Zeit in der internen Version bei der Verwendung von **BETWEEN** den Kontakt zur internen Datenbank abbricht.

```
001 SELECT CASE
002 WHEN EXTRACT( WEEKDAY FROM "a"."Datum" ) = 0 THEN '▓ ' || 'So.'
003 WHEN EXTRACT( WEEKDAY FROM "a"."Datum" ) = 6 THEN '▒ ' || 'Sa.'
004 WHEN EXTRACT( WEEKDAY FROM "a"."Datum" ) = 1 THEN '░ ' || 'Mo.'
005 ...
006 END AS "Wochentag", "a"."Datum",
007 ( SELECT "Mieter"."Name" FROM "Mieter", "Reservierung" WHERE "Mieter"."ID"
    = "Reservierung"."Mieter_ID" AND "Reservierung"."Haus_ID" = 1 AND
    "a"."Datum" >= "Reservierung"."Startdatum" AND "a"."Datum" <=
    "Reservierung"."Enddatum" ) "Haus1",
008 ( SELECT "Mieter"."Name" FROM "Mieter", "Reservierung" WHERE "Mieter"."ID"
    = "Reservierung"."Mieter_ID" AND "Reservierung"."Haus_ID" = 2 AND
    "a"."Datum" >= "Reservierung"."Startdatum" AND "a"."Datum" <=
    "Reservierung"."Enddatum" ) "Haus2",
009 ...
010 ... "Haus8"
011 FROM "Ansicht_Datum_lfdNr" AS "a" WHERE "a"."Datum" >= COALESCE ( ( SELECT
    "BeginnDatum" FROM "Filter" WHERE "ID" = TRUE ), CURRENT_DATE ) AND
```

```
"a"."Datum" <= COALESCE ( ( SELECT "EndeDatum" FROM "Filter" WHERE "ID" =
TRUE ), CURRENT_DATE )
```

Zuerst werden wieder die Tagesbezeichnungen zusammen mit der Kennzeichnung für die Tage erstellt. FIREBIRD nutzt für die Tagesnummern den Befehl **`EXTRACT( WEEKDAY FROM ...)`**, wobei die Tageszählung beim Sonntag mit '0' beginnt. Leider ist intern nicht irgendwo ein Kürzel für die Wochentage vorhanden, so dass jeder Wochentag einzeln deklariert werden muss. Hier gezeigt an den Zeile `2` bis `4`.

Um die Mieter einzubelenden muss der Befehl **`"Datum" BETWEEN "Startdatum" AND "Enddatum"`** geändert werden zu **`"Datum" >= "Startdatum" AND "Datum <= "Enddatum"`** (Ab Zeile `7`, entsprechend auch in Zeile `11`).

Da der Befehl **`WEEKDAY`** nicht in der GUI läuft muss die Abfrage auf **SQL-Befehl direkt ausführen** eingestellt werden. Sobald aber eine Abfrage mit dieser Einstellung läuft kann sie nicht mehr in einem Formular gefiltert und sortiert werden. Bestimmte Makroeigenschaften stehen damit nicht mehr zur Verfügung. Außerdem ergab sich bei einem ersten Test, dass das Tabellenkontrollfeld außer den Datumswerten nicht weiter anzeigte, obwohl die Abfrage einwandfrei lief.

Aus diesem Grunde wurde die Abfrage «Belegung» in eine «Ansicht_Belegung» umgewandelt. Sie behebt für FIREBIRD die beiden oben geschilderten Probleme.

Bericht

Datenquelle	
Tabelle:	*Mieter, Reservierung, Haus*
Ansicht:	*Ansicht_Datum_lfdNr*

Datenziel	
Bericht:	*Hausbelegung*

```
001 SELECT "Haus"."Haus", "Mieter"."Name",
002    "a"."Startdatum", "a"."Enddatum",
003    DATEDIFF( 'dd', "a"."Startdatum", "a"."Enddatum" ) AS "Tage",
004    ( SELECT "Datum" FROM "Ansicht_Datum_lfdNr" WHERE "lfdNr" =
          ( SELECT "lfdNr" FROM "Ansicht_Datum_lfdNr" WHERE "Datum" =
          "a"."Startdatum" ) - 30 )
          AS "AnzahlungBis"
005 FROM "Reservierung" AS "a", "Haus", "Mieter"
006 WHERE "a"."Haus_ID" = "Haus"."ID"
007    AND "a"."Mieter_ID" = "Mieter"."ID"
008 ORDER BY "Haus"."Haus" ASC, "a"."Startdatum" ASC
```

Diese Abfrage dient als Basis für den Bericht. Hier sollen Hausbezeichnung und Name des Mieters neben den Informationen aus der Tabelle "Reservierung" erscheinen. Außerdem kommen noch Informationen hinzu, die die Mietdauer und den Termin betreffen, zu dem eine Anzahlung zu erfolgen hat.

Da auf die Tabelle "Reservierung" wegen einer korrelierenden Unterabfrage durch ein Alias zugegriffen werden muss, werden alle damit verbundenen Felder über das Alias "a" angesprochen. Die Verbindung der Tabellen entspricht der Verbindung innerhalb der erklärten Beziehungen dieser Datenbank. Die ersten vier Felder dieser Abfrage sind also leicht über die GUI zusammen zu klicken. Die Tage, die ein Mieter in einem Haus ist, werden über die Funktion **`DATEDIFF`** berechnet. **`'dd'`** berechnet hier den Unterschied in Tagen zwischen den folgenden beiden Datumswerten.

Um das Datum zu berechnen, bis zu dem die Anzahlung fällig ist, wird auf die "Ansicht_Datum_lfdNr" zugegriffen. Durch diese Unterabfrage wird die in der HSQLDB fehlende Funktion **`DATEADD`** in diesem Falle ersetzt (Zeile `4`). Zu dem "Startdatum" des aktuellen Daten-

satzes wird die entsprechende "lfdNr" aus "Ansicht_Datum_lfdNr" ermittelt. Von dieser Nummer wird 30 als Anzahl der Tage subtrahiert, die das Anzahlungsdatum vor dem Startdatum liegen soll. Dann wird in der äußeren Abfrage zu dieser neuen "lfdNr" das entsprechende Datum aus der "Ansicht_Datum_lfdnr" ermittelt. Diese Abfragetechnik funktioniert leider nur solange, wie beide Datumswerte in der Ansicht vertreten sind.

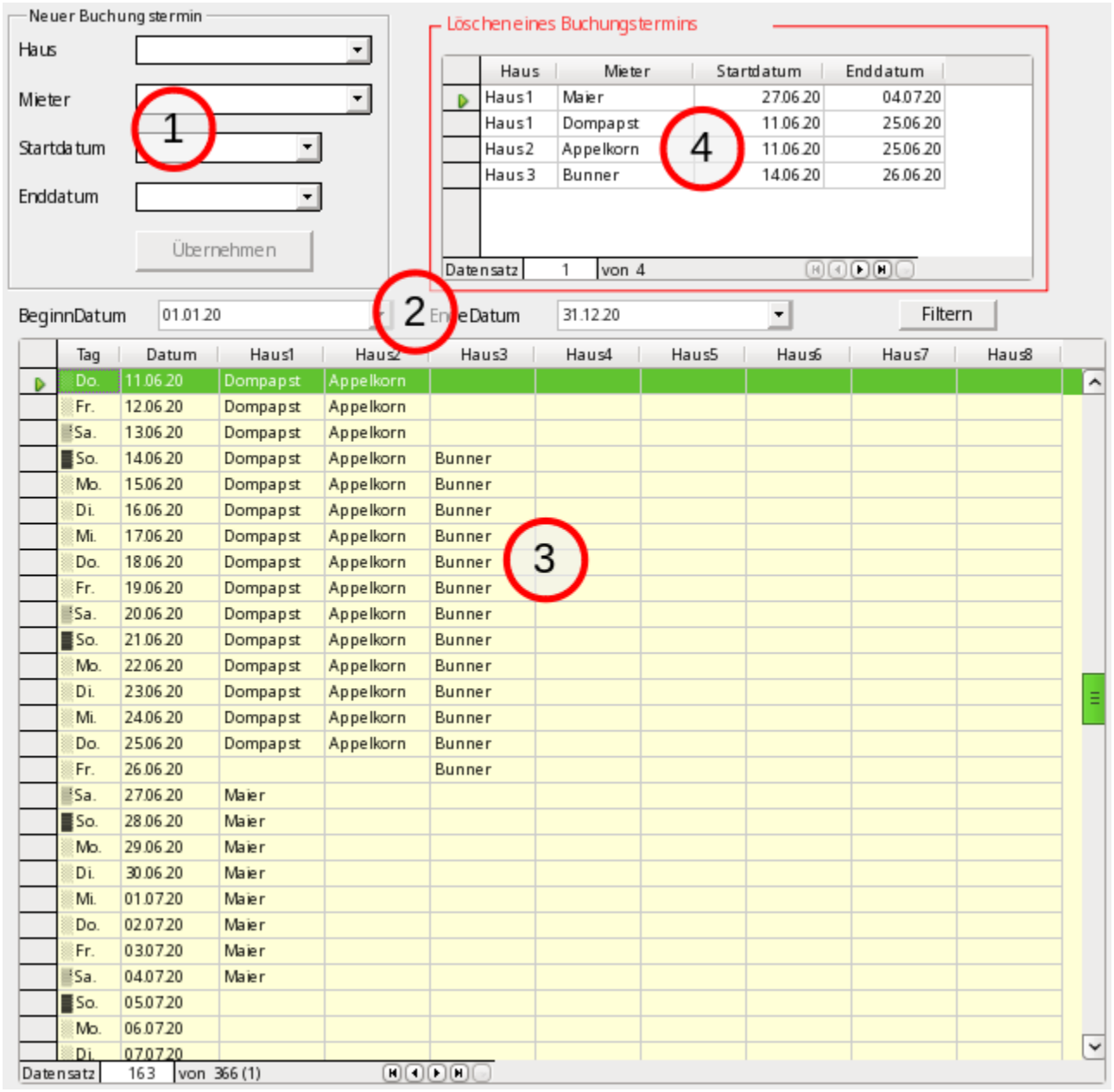
In FIREBIRD ließe sich eine Abfrage mit ähnlichem Code nur mit direktem SQL ausführen. Dafür stellt Firebird aber mehr Möglichkeiten bei der Datumsverarbeitung zur Verfügung, so dass das Ergebnis wesentlich einfacher zu erreichen ist:

```
001 SELECT "Haus"."Haus", "Mieter"."Name",
002    "a"."Startdatum", "a"."Enddatum",
003    "a"."Enddatum" - "a"."Startdatum" "Tage",
004    "a"."Startdatum" - 30 "AnzahlungBis"
005 FROM "Reservierung" AS "a", "Haus", "Mieter"
006 WHERE "a"."Haus_ID" = "Haus"."ID"
007    AND "a"."Mieter_ID" = "Mieter"."ID"
008 ORDER BY "Haus"."Haus" ASC, "a"."Startdatum" ASC
```

Hier können Datumswerte voneinander subtrahiert werden und das Ergebnis ist eine Tagesausgabe (Zeile 3). Außerdem können von Datumswerten ganze Zahlen subtrahiert werden. Das Ergebnis ist eine Datumsausgabe (Zeile 4).

Formulare

Diese Datenbank enthält nur ein Formular, das Formular Haus. Es gliedert sich in einen Eingabebereich für neue Buchungstermine, eine Auswahl der bestehenden Buchungstermine und eine Anzeige zur Übersicht über Buchungstermine in einem bestimmten Zeitraum.



1	Formular (Tabelle: *Reservierung*)
2	Filter (Tabelle: *Filter*)
3	Uebersicht (Abfrage: *Belegung*)
4	BuchungLoeschen (Tabelle: *Reservierung*)

Makros in Formulareigenschaften		
1	Nach der Datensatzaktion	Module1.*NeueBuchung*
4	Nach der Datensatzaktion	Module1.*BuchungsAenderung*
Makros in Feldeigenschaften		
1	Listenfeld Haus (Modifiziert)	Module1.*Startdatum*
1	Listenfeld Startdatum (Modifiziert)	Module1.*Enddatum*
4	Startdatum (Maustaste gedrückt)	Module1.*DatumsAenderung*

Der größte Aufwand wird betrieben, um bei der Auswahl des Startdatums bzw. des Enddatums nur tatsächlich mögliche Werte bereit zu stellen. Aus diesem Grunde sind die Felder auch nicht als Datumsfelder, sondern als Listenfelder vorgegeben. Für die Listenfelder kann über den SQL-Code die Auswahlmöglichkeit des Inhaltes gesteuert werden. Beide Listenfelder sind darauf eingestellt, dass sie im SQL-Code den Wert des Feldes, den sie anzeigen, anschließend auch in die darunterliegende Tabelle übertragen: **Eigenschaften → Daten → gebundenes Feld → 0**.

Würde in obigem Beispiel über das Listenfeld für das Haus 'Haus 4' gewählt, dann sind für beide Datumsfelder sämtliche Eingaben offen. Das Enddatums-Feld darf allerdings nur eine Eingabe möglich machen, die mindestens einen Tag hinter dem Startdatum liegt.

Schwieriger wird es z.B. bei der Auswahl von 'Haus 1'. Dort sind im Listenfeld für das Startdatum nur die Werte möglich, die eben bisher nicht mit 'belegt' gekennzeichnet sind. Würde aber jetzt z.B. der 26.6.20 für dieses Haus gewählt, dann wäre kein Enddatum mehr möglich. Das Listenfeld darf dieses Datum also nicht anbieten, obwohl es nicht belegt ist. Nur noch die Datumswerte vom 5.7. an dürfen angeboten werden, da sie in der Zukunft liegen. Die aktuellen Einträge wurden nämlich am 11.6. testweise erstellt. Termine für die Vergangenheit zu buchen wird also direkt ausgeschlossen.

Es muss also zuerst das Haus, dann das Startdatum und erst danach das Enddatum gewählt werden. Entsprechend sind die Makros auch mit den Listenfeldern für das Haus und das Startdatum verbunden.

Sobald das Startdatum gewählt wird stellt sich die Übersicht auf dieses Startdatum ein. Das Datum wird in der ersten Zeile der Übersicht angezeigt, die in dem Screenshot grün markiert ist. Jetzt muss noch ein Enddatum gewählt werden, das schon so weit vorgegeben ist, dass es auf jeden Fall hinter dem Startdatum liegt und auf keinen Fall die folgende Belegung beeinflusst.

Sobald der Datensatz über die Schaltfläche Speichern abgespeichert wurde wird das Formular für die Eingabe auf einen neuen Datensatz eingestellt. Die Schaltfläche Speichern ist in den Eigenschaften mit der **Aktion → Neuer Datensatz** verbunden. Gleichzeitig wird über das Formular die Übersicht aktualisiert, so dass die eingegebene Belegung mit dem entsprechenden Mieter in der Liste sichtbar erscheint.

Das Tabellenkontrollfeld rechts oben im Formular dient lediglich zum Löschen von Eingaben. Eine Änderung von Eingaben ist nicht vorgesehen, da dies auch schnell über die Neueingabe erfolgen kann. Auch bei diesem Tabellenkontrollfeld ist die Möglichkeit gegeben, den Datensatz vorher in der Übersicht über einen Doppelklick auf das Feld «Startdatum» noch einmal anzusehen. Um dies zu ermöglichen darf das Formular selbst nicht die Datenänderung verbieten. Stattdessen sind alle Felder in dem Formular auf **Nur Lesen → Ja** eingestellt. Würde die Datenänderung im Formular verboten oder das Datumsfeld inaktiv geschaltet, so ließe sich keine Aktion von diesem Feld mehr auslösen.

Berichte

Diese Beispieldatenbank enthält lediglich einen Bericht, den Bericht Hausbelegung.

Haus: **Haus 1**

Name	*Startdatum*	*Enddatum*	*Tage*	*Anzahlung bis*
Dompapst	11.06.20	25.06.20	14	12.05.20
Maier	27.06.20	04.07.20	7	28.05.20

Tage insgesamt: **21**

Haus: **Haus 2**

Name	*Startdatum*	*Enddatum*	*Tage*	*Anzahlung bis*
Appelkorn	11.06.20	25.06.20	14	12.05.20

Tage insgesamt: **14**

Haus: **Haus 3**

Name	*Startdatum*	*Enddatum*	*Tage*	*Anzahlung bis*
Bunner	14.06.20	26.06.20	12	15.05.20

Tage insgesamt: **12**

Datenquelle
Abfrage: *Bericht*

Der Bericht gibt eine Übersicht über die Belegung der Häuser zurück. Er ist gruppiert nach den Hausbezeichnungen. Im Gruppenfuß steht zusätzlich, wie viele Tage das Haus insgesamt belegt ist. Es gibt also so etwas wie eine Auslastung des Hauses in der angegebenen Zeitspanne wieder. Die Funktion für die Summierung ist eine der eingebauten Funktionen. Das Textfeld wurde aufgezogen und anschließend in **Eigenschaften → Daten** die folgenden Angaben gemacht:

- Datenfeld-Typ: Funktion
- Datenfeld: Tage
- Funktion: Summe
- Geltungsbereich: Gruppe: Haus

Makros

Startdatum

Aufruf aus
Formular: *Haus*

Benötigt
Tabelle: *Reservierung*
Ansicht: *Ansicht_Datum*

```
001  SUB Startdatum(oEvent AS OBJECT)
002     DIM oFeldStart AS OBJECT
003     DIM oForm AS OBJECT
004     DIM oFeld AS OBJECT
```

```
005     DIM inID AS INTEGER
006     DIM stSql(0) AS STRING
007     DIM stTeilsql AS STRING
008     oFeldStart = oEvent.Source.Model
009     oForm = oFeldStart.Parent
010     oFeld = oForm.getByName("Startdatum")
011     inID = oFeldStart.CurrentValue
012     stTeilsql = " ( SELECT ""a"".""Datum"", ""a"".""lfdNr"",
           ( SELECT 'belegt' FROM ""Reservierung"" WHERE ""Haus_ID"" = '"+inID+"' AND
           ""a"".""Datum"" BETWEEN ""Startdatum"" AND ""Enddatum"" ) AS ""Haus""
           FROM ""Ansicht_Datum_lfdNr"" AS ""a"" WHERE ""a"".""Datum"" >= CURRENT_DATE )
           WHERE ""Haus"" IS NULL "
013     stSql(0) = "SELECT ""Datum"" FROM "+stTeilsql+" AND ""lfdNr""+1 IN
           (SELECT ""lfdNr"" FROM "+stTeilsql+")"
014     oFeld.ListSource = stSql
015     oFeld.refresh
016 END SUB
```

Diese Makro soll das Listenfeld mit Inhalt versorgen, das das Startdatum abspeichern soll. Das Makro liest zuerst die Position für das Listenfeld in dem Formular über das auslösende Ereignis aus (Zeile 8 bis 11). Der Wert für inID stellt den Wert dar, der von dem Listenfeld für die Hausauswahl bereitgestellt wird. Die Hausauswahl bestimmt schließlich, welche Datumswerte noch frei sind.

Das Listenfeld selbst wird mit einem **`Array of String`** über **`stSql(0)`** beschickt. Der SQL-Code wird in Zeile 12 und 13 erstellt, in Zeile 14 als Code an das Feld weitergegeben und in Zeile 15 schließlich durch den **`Refresh`** auch in dem Feld angezeigt.

Der SQL-Code verlangt hier eine etwas ausführlichere Erklärung, da mehrere verschachtelte Unterabfragen enthalten sind.

Eine für 2 Elemente gleichlautende Unterabfrage ist als **`stTeilsql`** aus der äußeren Abfrage abgetrennt worden (Zeile 12).

Mit **`( SELECT 'belegt' ... ) AS "Haus"`** wird zuerst einmal für alle Datumswerte ein 'belegt' eingefügt, wenn das betreffende Haus für die Vermietung nicht mehr frei ist. Diese Unterabfrage ist gleich doppelt vertreten und beinhaltet jedes Mal den aus dem Feld für das Haus ermittelten Wert inID.

Um diese Abfrage herum ist eine weitere Abfrage konstruiert: **`( SELECT "a"."Datum" ... WHERE "a"."Datum" >= CURRENT_DATE)`**. Dadurch wird jetzt eine Liste erstellt, die die folgenden Kriterien erfüllt: Alle Datumswerte werden aus der Ansicht "Ansicht_Datum_lfdNr" übernommen, die gleich oder größer als das aktuelle Datum sind, also in der Zukunft liegen. Daneben ist für Berechnungszwecke die "lfdNr" enthalten sowie eine Spalte "Haus", in der an bestimmten Stellen 'belegt' steht. Diese gesamte Abfrage ist ebenfalls doppelt enthalten. Einmal dient sie dazu, direkt die Datumswerte anzugeben, bei denen nicht das Wort 'belegt' steht, die also leer sind: **`"Haus" IS NULL`**.

In Zeile 13 wird dann diese Unterabfrage in die entsprechende Rahmenabfrage eingebaut. Benötigt wird aus der Unterabfrage nur das Datum. Allerdings soll zu dem Datum auch ein direkter Nachfolger existieren, damit neben dem Startdatum auch ein Enddatum möglich ist - eine Buchung also von einem Tag zum nächsten. Deswegen wird das Datum nur ausgegeben, wenn die **`"lfdNr" + 1 IN`** der Menge der Datensätze der Unterabfrage enthalten ist. Die Arbeit mit der **`lfdNr`** ist hier wieder notwendig, da die HSQLDB die Addition von Tagen zu einem Datum nicht beherrscht.

Enddatum

Aufruf aus
Formular: *Haus*

Benötigt	
Tabelle:	*Reservierung*
Ansicht:	*Ansicht_Datum*
Makro:	*DatumsAenderung*

```
001  SUB Enddatum(oEvent AS OBJECT)
002     DIM oDatField AS OBJECT
003     DIM oForm AS OBJECT
004     DIM oFeld AS OBJECT
005     DIM inID AS INTEGER
006     DIM stSql(0) AS STRING
007     DIM stTeilsql AS STRING
008     DIM stDatum AS STRING
009     oDatField = oEvent.Source.Model
010     stDatum = oDatField.CurrentValue.Year & "-" &
           Right("0" & oDatField.CurrentValue.Month , 2) & "-" &
           Right("0" & oDatField.CurrentValue.Day , 2)
011     oForm = oDatField.Parent
012     oFeld = oForm.getByName("Enddatum")
013     inID = oForm.getByName("Haus_ID").CurrentValue
014     stTeilsql = "( SELECT ""a"".""Datum"", ( SELECT 'belegt' FROM ""Reservierung""
           WHERE ""Haus_ID"" = '"+inID+"' AND ""a"".""Datum"" BETWEEN ""Startdatum""
           AND ""Enddatum"" ) AS ""Haus"" FROM ""Ansicht_Datum"" AS ""a""
           WHERE ""a"".""Datum"" > '"+stDatum+"')"
015     stSql(0) = "SELECT ""Datum"" FROM "+stTeilsql+" WHERE ""Datum"" <
           COALESCE((SELECT MIN(""Datum"") FROM "+stTeilsql+"
           WHERE ""Haus"" = 'belegt'),(SELECT MAX (""Datum"") FROM ""Ansicht_Datum""))"
016     oFeld.ListSource = stSql
017     oFeld.refresh
018     DatumsAenderung(oEvent)
019  END SUB
```

Das Enddatum wird in Abhängigkeit von dem Startdatum und dem Haus in dem Listenfeld zur Verfügung gestellt.

Aus dem Feld für das Startdatum wird der Datumswert in SQL-Form ausgelesen (Zeile 9 und 10). Dann wird über das Formular die Verbindung zum Feld für das Enddatum sowie zu dem Feld für das Haus Bezug genommen (Zeile 11 bis 13). Aus dem Feld für das Haus wird der Schlüsselwert inID ausgelesen, der später für die Erstellung des SQL-Codes benötigt wird.

Der SQL-Code wird erstellt, an das Listenfeld weitergegeben (Zeile 16), das Listenfeld neu eingelesen (Zeile 17) und anschließend die Datumsänderung an die entsprechende Prozedur zur Steuerung der Übersicht weitergegeben, damit auch in der Übersicht alle passenden Eingaben für das Enddatum ersichtlich sind.

Der SQL-Code ist zur besseren Erklärbarkeit wieder in zwei Teile aufgesplittet. Die innere Abfrage stTeilsql gibt vom Prinzip her die gleiche Datenmenge wie in der Prozedur Startdatum wieder (Zeile 14). Einzige zusätzliche Einschränkung ist hier, dass das ausgewählte Datum größer sein muss als das Datum, das als Startdatum ausgesucht wurde. Außerdem werden alle Datumswerte ermittelt - auch die Werte, bei denen das Haus belegt ist.

Mit dem an das Listenfeld weitergegebenen Code (Zeile 15) wird wieder das Datum übermittelt. Bedingungen für dieses Datum sind, dass

- es größer ist als das Startdatum (stTeilsql)
- und kleiner ist als das kleinste dann noch in der stTeilsql vorhandene Datum, an dem das Haus belegt ist
- oder, sofern keine Belegung folgt, so groß ist wie das maximal verfügbare Datum der Ansicht "Ansicht_Datum".

Damit werden in dem Listenfeld nur die möglichen auswählbaren Datumswerte angezeigt. Es ist nicht möglich, ein Datum auszuwählen, das in eine Belegungszeit fällt. Es ist auch nicht

möglich, ein Datum auszuwählen, bei der das Startdatum vor einer Belegungszeit, das Enddatum aber nach dieser Belegungszeit liegt.

Für FIREBIRD existiert «Ansicht_Datum» nicht, so dass stattdessen «Ansicht_Datum_lfdNr» eingesetzt werden muss.

BuchungsAenderung

Aufruf aus
Formular: *Haus*
Makro: *NeueBuchung*

```
001  SUB BuchungsAenderung
002     DIM oDoc AS OBJECT
003     DIM oDrawpage AS OBJECT
004     DIM oForm AS OBJECT
005     DIM iRow AS INTEGER
006     oDoc = thisComponent
007     oDrawpage = oDoc.drawpage
008     oForm = oDrawpage.forms.getByName("Uebersicht")
009     iRow = oForm.getRow
010     oForm.reload
011     oForm.last
012     oForm.absolute(iRow)
013  END SUB
```

Wird an der Buchung etwas geändert, also eine Buchung hinzugefügt oder gelöscht, so muss die Übersicht für die Buchungen neu eingelesen werden. Damit die Übersicht dann nicht einfach zum ersten Datensatz zurückspringt wird in Zeile 9 die aktuelle Zeile der Datenmenge ausgelesen. Nach dem **Reload** in Zeile 10 wird dann das Formular auf den letzten Datensatz gesetzt (Zeile 11) um anschließend auf den aktuellen Datensatz zurückzuspringen (Zeile 12). Das hat den Vorteil, dass der aktuelle Datensatz immer die erste Zeile im Tabellenkontrollfeld ist und nebenbei auch noch die Angabe der maximal verfügbaren Datensätze in der Navigationsleiste des Tabellenkontrollfeldes genau stimmig gehalten wird.

NeueBuchung

Aufruf aus
Formular: *Haus*

Benötigt
Makro: *BuchungsAenderung*

```
001  SUB NeueBuchung
002     BuchungsAenderung
003     DIM oDoc AS OBJECT
004     DIM oDrawpage AS OBJECT
005     DIM oForm AS OBJECT
006     oDoc = thisComponent
007     oDrawpage = oDoc.drawpage
008     oForm = oDrawpage.forms.getByName("BuchungLoeschen")
009     oForm.reload
010  END SUB
```

Ist eine neue Buchung erfolgt, so wird zuerst die Buchungsänderung an die Übersicht im unteren Tabellenkontrollfeld weiter gegeben. Zusätzlich muss dann noch der Datensatz in den möglicherweise löschbaren Datensätzen erscheinen. Deswegen wird das Formular «BuchungLoeschen» ebenfalls neu geladen.

DatumsAenderung

Aufruf aus
Formular: *Haus*
Makro: *Enddatum*

```
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 einer Datumsauswahl des Startdatums soll die Übersicht auf das aktuelle Datum so eingestellt werden, dass es am besten als erste Feld in der Übersicht erscheint.

Zuerst wird das Datumsfeld ermittelt und das darin enthaltene Datum in SQL-Schreibweise ausgelesen (Zeile 12 und 13). Anschließend wird das Formular «Übersicht» angesteuert. Aus dem Formular wird die Datenbankverbindung ausgelesen (Zeile 17).

In dem Objekt des Formulars ist nicht nur das eigentliche SQL-Kommando enthalten, das beim Start eines Formulars den Inhalt bestimmt. Es kann ja passieren, dass die Daten mit einem Filter gefiltert werden oder sortiert werden. Der genaue Befehl hierfür liegt etwas versteckt in **`oForm.SingleSelectQueryComposer.Query`** (Zeile 19). Nur über diesen aktuellen SQL-Befehl lässt sich genau die Datenzeile ermitteln, die für das gewünschte Datum anzusteuern ist. Dieser Composer steht nur zur Verfügung, wenn die Abfrage nicht als direkte SQL-Abfrage gestellt wird. Direkte SQL-Abfragen können nicht richtig gefiltert und sortiert werden.

Die Abfrage, die auch den Inhalt des Tabellenkontrollfeldes bestimmt, wird hier noch einmal genutzt. In Zeile 20 wird die Abfrage gestellt. Von Zeile 21 bis 25 läuft eine Schleife so lange ab, bis das Datum aus der Abfrage dem Datum entspricht, das aus dem Datumsfeld in Zeile 13 ausgelesen wurde. Sobald also die Abfrage eine Ergebniszeile liefert (Zeile 22), wird die Zeilenanzahl um 1 erhöht (Zeile 23). Sollte die letzte Zeile erreicht sein ohne dass das Datum gefunden wurde, so muss aus der Schleife herausgesprungen werden (Zeile 24). Dies kann z.B. bei einer Filterung der Daten passieren, wenn z.B. der Filter nur auf den Samstag eingestellt ist, aber das Datum an einem Freitag liegt.

Die genaue Zeilennummer ist jetzt ermittelt. Es wird in Zeile 26 zum letzten Datensatz des Formulars gesprungen und in Zeile 27 dann wieder zurück zur ermittelten Zeilennummer. So erscheint die Zeile schließlich als erste in der Übersicht des Tabellenkontrollfeldes.

Leider wird für Firebird **isLast** (Zeile 24) nicht unterstützt. Deshalb muss der Code zum Schluss etwas anders gestaltet werden:

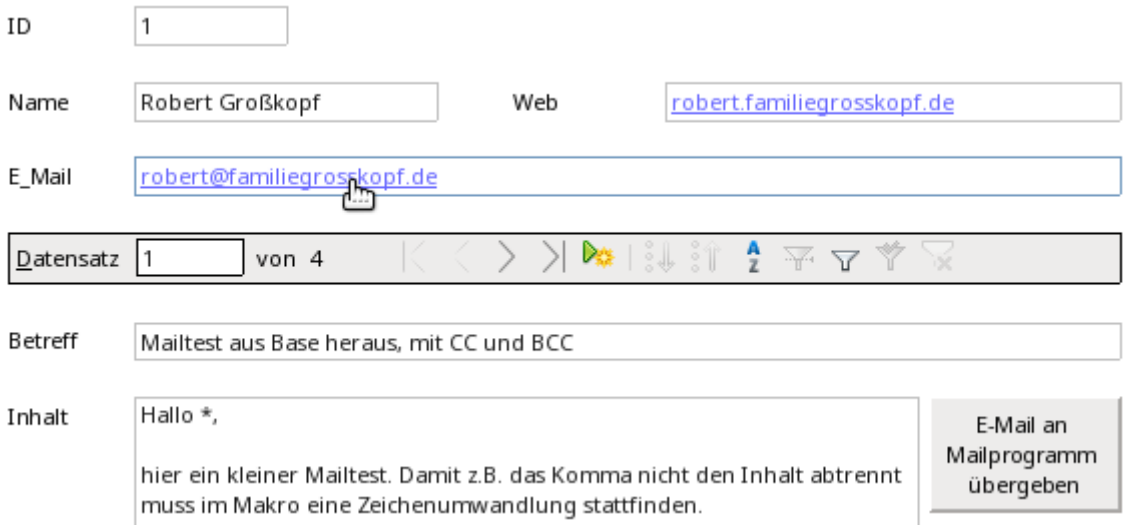
```
020     oResult = oSQL_Statement.executeQuery(stSql)
021     oForm.last
022     iLastRow = oForm.getRow
023     Do
024        oResult.next
025        iRow = iRow + 1
026        IF iRow = iLastRow THEN Exit Do
027     LOOP UNTIL stDatum = oResult.getString(2)
028     oForm.absolute(iRow)
029  END SUB
```

Hier wird jetzt für Firebird der Sprung ans Ende des Formular vorgezogen und die letzte Datensatznummer ausgelesen (Zeile 21 und 22). Sind die aktuell ermittelte Datensatznummer und die letzte Datensatznummer gleich, so wird die anschließende Schleife verlassen. Ansonsten springt das Formular zur aktuell ermittelten Datensatznummer (Zeile 28).

Mailaufruf

Einführung

In Foren kam immer wieder der Wunsch auf, von einer Datenbank aus die in einer Adressverwaltung gespeicherten Mailadressen auch direkt zum Mailversand nutzen zu können.



Die Datenbank «Mailaufruf»[6] erlaubt sowohl den Direktstart des Mailprogramms mit vorgegebener Mailadresse als auch den Direktaufruf einer vorgegebenen Website über den Browser.

Zum Versand kompletter Mails wird in dieser Datenbank immer das Mailprogramm genutzt, das über das Betriebssystem zur Verfügung gestellt wird. In der Datenbank kann aber der gesamte Mailverkehr abgelegt werden. Nur die letzte Kontrolle, ob denn nun die E-Mail tatsächlich vom Mailprogramm abgeschickt wurde, ist von Base heraus nicht möglich.

Diese Datenbank funktioniert mit der HSQLDB und mit der internen FIREBIRD gleich. Dies liegt an der einfachen Struktur und daran, dass nur in den Makros einfache Abfragen erfolgen, die sich vom Code her in den Datenbanken nicht unterscheiden.

Tabellen

Die Datenbank besteht aus vier Tabellen.

Kontakte

Datenziel
Formular: *Kontakte_Mail_Web, Mail_mit_Anhang*
Makro: *Mail_Aufruf, Mail_Attachment*
Abfrage, Bericht, Makro: keine

6 Beispieldatenbank Beispiel_Mailaufruf.odb

Feldname	Feldtyp	Beschreibung
ID	Integer	Primärschlüssel der Tabelle. Der Primärschlüssel muss eindeutig sein. Das Feld kann auch als Auto-Wert-Feld gesetzt werden.
Name	Text	Der Name der Person, an die die Mail verschickt werden soll und zu der auch die Webadresse gehört.
E_Mail	Text	Die E-Mail-Adresse der Person
Web	Text	Falls die Person eine Website führt, so kann z.B. auch aus Base heraus die Website aufgerufen werden. Eingabe erforderlich: Nein

Diese Tabelle könnte natürlich für die Nutzung von Serienbriefen entsprechend geändert und erweitert werden. So fehlt hier die Möglichkeit der Eingabe des Geschlechts, das dann eine entsprechende Anrede ermöglicht. Vorname und Nachname werden außerdem nicht getrennt und eine Adresse ist auch nicht vorgesehen. Würden diese Felder mit aufgenommen, so ließe sich auch von Base heraus ein Serienbrief starten, der sowohl einen personalisierten Inhalt im Mailtext als auch im Mailanhang hat.

Mails

Datenziel
Formular: *Kontakte_Mail_Web, Mail_mit_Anhang*
Abfrage, Bericht, Makro: keine

Feldname	Feldtyp	Beschreibung
ID	Integer	Primärschlüssel der Tabelle. Der Primärschlüssel muss eindeutig sein. Das Feld sollte als Auto-Wert-Feld gesetzt werden.
Betreff	Text	Betreff der zu versendenden E-Mail
Inhalt	Text	Der Inhalt der Mail kann viel Platz einnehmen. Die Standardeinstellung des Feldes mit 100 Zeichen dürfte also in den wenigsten Fällen ausreichen. In dem Beispiel ist dies auf 1000 Zeichen erweitert worden
Kontakte_ID	Integer	Falls die Person eine Website führt, so kann z.B. auch aus Base heraus die Website aufgerufen werden.

Mail_CC

Datenziel
Formular: *Kontakte_Mail_Web, Mail_mit_Anhang*
Makro: *Mail_Aufruf, Mail_Attachment*
Abfrage, Bericht: keine

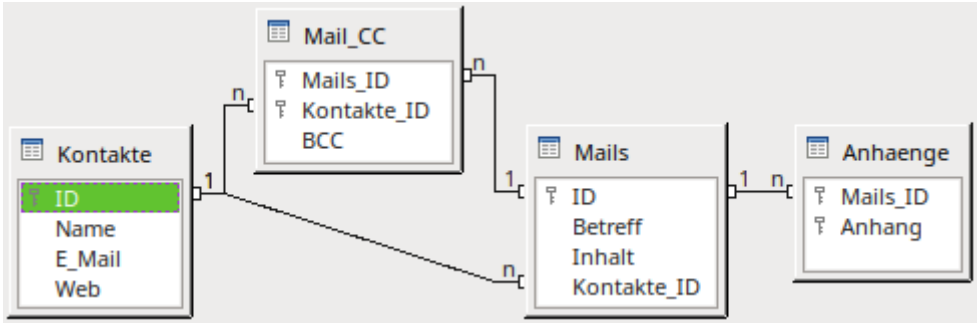
Feldname	Feldtyp	Beschreibung
Mails_ID	Integer	Die Mails_ID wird als Fremdschlüssel aus der Tabelle "Mails" bezogen. Dadurch können mehrere Personen eine Mailkopie erhalten.
Kontakte_ID	Integer	Die Kontakte_ID dienst dazu, die Mailadresse von anderen Personen aus der Tabelle "Kontakte" zu ermitteln. Mails_ID und Kontakte_ID bilden zusammen den Primärschlüssel der Tabelle "Mail_CC"
BCC	Ja/Nein	Muss nur angeklickt werden, wenn die Person eine Blindkopie erhalten soll.

Anhaenge

Datenziel
Formular: *Mail_mit_Anhang*
Makro: *Mail_Attachment*
Abfrage, Bericht: keine

Feldname	Feldtyp	Beschreibung
Mails_ID	Integer	Die Mails_ID wird als Fremdschlüssel aus der Tabelle "Mails" bezogen. Dadurch können mehrere Anhänge erzeugt werden
Anhang	Text	Hier wird der Pfad zum Anhang als relativer Pfad gespeichert. Mails_ID und Anhang bilden zusammen den Primräschlüssel der Tabelle "Anhaenge".

Alle vier Tabellen werden in den Beziehungen noch miteinander verbunden.



Wechselt eine Person im Laufe der Zeit die E-Mail-Adresse, so hat das keinen weiteren Einfluss auf die gespeicherten E-Mails. Nur der Inhalt sowie die Verbindung zu entsprechenden Personen wird gespeichert. Einmal überschriebene Mailadressen erscheinen nicht mehr in der Datenbank, sondern nur noch in der Liste der versandten Mails des Mailprogramms.

Einem Kontakt können beliebig viele Mails zugeordnet werden. Das Verhältnis von "Kontakte" zu "Mails" ist ein Verhältnis 1:n.

Einer Mail können beliebig viele Kopieempfänger zugeordnet werden. Das Verhältnis von "Mails" zu "Mail_CC" ist 1:n.

Ein Kontakt kann auch mehrmals als Kopieempfänger zugeordnet werden. Das Verhältnis von "Kontakte" zu "Mail_CC" ist 1:n.

Ausgeschlossen ist aber, dass eine Person mehrere Kopien der gleichen Mail erhält. "Mails_ID" und "Kontakte_ID" sind zusammen Primärschlüssel und müssen einzigartig sein.

Ansichten

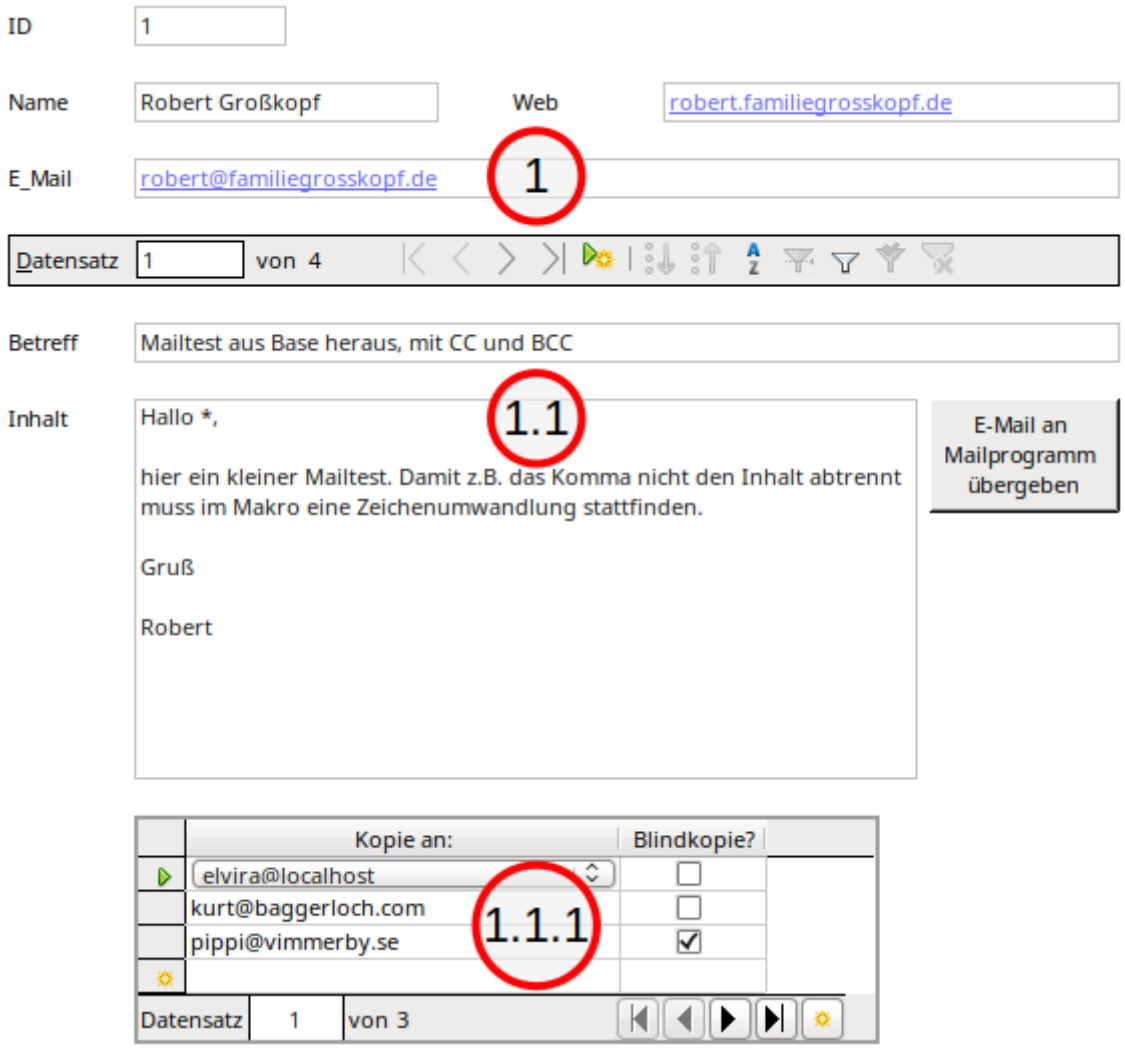
Diese Datenbank enthält keine Ansichten.

Abfragen

Die Datenbank enthält keine Abfragen.

Formulare

Kontakte_Mail_Web



1	Formular (Tabelle: *Kontakte*)	**1.1**	Unterformular (Tabelle: *Mails*)	**1.1.1**	FormMailsCC (Tabelle: *Mail_CC*)

Makros in Feldeigenschaften		
1	Feld «Web» (Maus innerhalb)	Module1.*Mauszeiger*
1	Feld «Web» (Maustaste gedrückt)	Module1.*Website_Aufruf*

1	Feld «E-Mail» (Maus innerhalb)	Module1.*Mauszeiger*
1	Feld «E-Mail» (Maustaste gedrückt)	Module1.*Website_Aufruf*
1.1	Button *E-Mail an Mailprogramm übergeben* (Aktion ausführen)	Module1.*Mail_Aufruf*

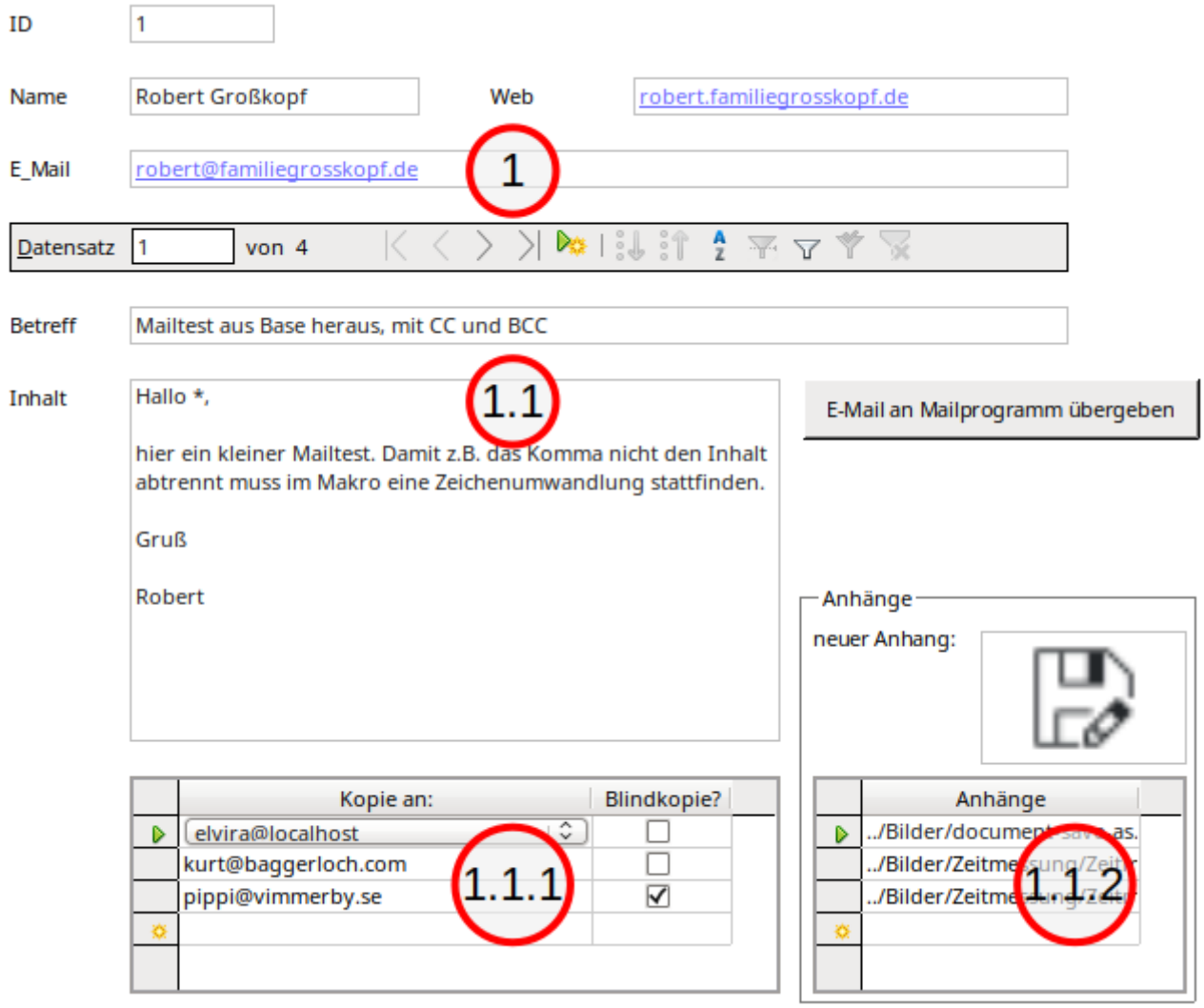
Im Hauptformular wird der Kontakt ausgesucht oder ein neuer Kontakt eingegeben. Die Eingabe der Mailadresse und gegebenenfalls einer Website erscheinen in blauer Farbe und unterstrichen. Wird die Maus auf diese Elemente bewegt, so ändert sich der Mauszeiger. Mit einem Klick wird direkt das Mailprogramm mit dieser Mailadresse oder der Browser mit dieser Webadresse gestartet.

Im Unterformular werden Mails an die im Hauptformular ausgesuchte Person verfasst. In diesem Formular ist nur die Eingabe des Betreffs und des Mailinhaltes möglich, da das Makro, mit dem der Mailversand gestartet wird, nicht bei jedem Mailprogramm sicher das Versenden von Anhängen erlaubt. Der Parameter «attachment» ist beim Versenden über die Kommandozeile nicht einheitlich anerkannt.

Soll die so verfasste Mail auch noch an weitere Personen gehen, so können deren Mailadressen im Unter-Unterformular «FormMailsCC» ausgesucht werden. Blindkopien, die für die anderen Empfänger nicht sichtbar sind, können hier über ein Ja/Nein-Feld gekennzeichnet werden.

Der Inhalt des Formulars «FormMailsCC» wird bei Betätigung des Buttons in «Unterformular» abgespeichert. Dies liegt daran, dass ein Verlassen eines Formulars auf einer gemeinsamen Formularoberfläche mit anderen Formularen so eine Speicherung automatisch ausführt.

Mail_mit_Anhang



1	Formular (Tabelle: *Kontakte*)	**1.1**	Unterformular (Tabelle: *Mails*)	**1.1.1**	FormMailsCC (Tabelle: *Mail_CC*)
				1.1.2	FormAnhaenge (Tabelle: *Anhaenge*)

Makros in Feldeigenschaften		
1	Feld «Web» (Maus innerhalb)	Module1.*Mauszeiger*
1	Feld «Web» (Maustaste gedrückt)	Module1.*Website_Aufruf*
1	Feld «E-Mail» (Maus innerhalb)	Module1.*Mauszeiger*
1	Feld «E-Mail» (Maustaste gedrückt)	Module1.*Website_Aufruf*
1.1	Button *E-Mail an Mailprogramm übergeben* (Aktion ausführen)	Module1.*Mail_Attachment*
1.1.2	Tabellenkontrollfeld «Anhänge» (Text modifiziert)	Module1.*Attachment*

Dieses Formular erlaubt zusätzlich zu dem Formular «Kontakte_Mail_Web» die Auswahl und Linkspeicherung von Anhängen. Die Links werden über das grafische Kontrollfeld ausgesucht. Zum einen erlaubt dieses Kontrollfeld seit LO 5.0 die Abspeicherung von Links nicht nur zu grafischen Dateien, sondern auch zu allen anderen Dateien. Zum anderen müsste ein Dateiaus-

wahlfeld erst umständlich mit Hilfe von Makros an die Datenbank angebunden werden, da es nicht mit einem Datenfeld verbunden werden kann. Letztlich hat dann das grafische Kontrollfeld auch noch den Vorteil, dass es eben Bilddateien, die sich im Anhang befinden, direkt anzeigen kann.

Mit jedem neu ausgesuchten Element für den Anhang wird ein neuer Datensatz hinzugefügt. Soll ein Anhang wieder entfernt werden, so muss das unter dem grafischen Kontrollfeld befindliche Tabellen-Kontrollfeld aufgesucht werden. Über einen rechten Mausklick auf den Zeilenkopf steht hier das Löschen des jeweiligen Inhaltes zur Verfügung.

Hinweis

Für die einwandfreie Funktion dieses Formulars ist es unter Linux notwendig, unter **Extras → Optionen → Internet → E-Mail** das E-Mail-Programm auszuwählen. Sonst gibt es Probleme mit Kommas und Zeilenumbrüchen.

Berichte

Die Datenbank enthält keine Berichte.

Makros

Mauszeiger

Aufruf aus
Formular: *Kontakte_Mail_Web, Mail_mit_Anhang*

```
001  SUB Mauszeiger(oEvent AS OBJECT)
002     DIM oPointer AS OBJECT
003     oPointer = createUnoService("com.sun.star.awt.Pointer")
004     oPointer.setType(28)
005     oEvent.Source.Peer.SetPointer(oPointer)
006  END SUB
```

Der Mauszeiger wird über den entsprechenden **UnoService** beeinflusst. Die Typen für den Mauszeiger sind in **com.sun.star.awt.SystemPointer** verzeichnet. Die 28 verweist hier auf das Aussehen des Zeigers als Hand mit einem zeigenden Finger. Dieses Makro wird dann ausgelöst, wenn sich die Maus innerhalb des Formularfeldes befindet: **Textfeld → Eigenschaften → Ereignisse → Maus innerhalb**. Da das Ereignis direkt von dem Formularfeld heraus gesteuert wird (und nicht durch einen Button von außerhalb) muss hier nicht erst das Formularfeld aufgesucht werden. Das Formularfeld ist Urheber des Ereignisses, **oEvent.Source**. Darüber wird schließlich beim Formularfeld der Mauszeigen neu eingestellt (Zeile 5).

FormSave

Aufruf aus
Makro: *Mail_Aufruf, Attachment, Mail_Attachment*

```
001  SUB FormSave(oForm AS OBJECT)
002     IF oForm.IsModified THEN
003        IF oForm.isNew THEN
004           oForm.InsertRow
005        ELSE
006           oForm.UpdateRow
```

```
007         END IF
008      END IF
009   END SUB
```

Das Speichern eines Formulars funktioniert intern über zwei unterschiedliche Befehle. Ist der Inhalt des Formulars neu (Zeile 4), dann muss ein **Insert** erfolgen. Existierte der Datensatz schon und wurde nur geändert, so muss ein **Update** erfolgen (Zeile 6). Diese kleine Prozedur wurde ausgelagert, da sie gleich in mehreren anderen Prozeduren sonst wiedrholt hätte geschrieben werden müssen.

Website_Aufruf

Aufruf aus
Formular: *Kontakte_Mail_Web, Mail_mit_Anhang*

```
001  SUB Website_Aufruf(oEvent AS OBJECT)
002     DIM oFeld AS OBJECT
003     DIM oShell AS OBJECT
004     DIM stFeld AS STRING
005     oFeld = oEvent.Source.Model
006     stFeld = oFeld.Text
007     IF stFeld = "" THEN
008        EXIT SUB
009     END IF
010     IF InStr(stFeld,"@") THEN
011        stFeld = "mailto:"+stFeld
012     ELSEIF InStr(stFeld,"http://") OR InStr(stFeld,"https://") THEN
013        stFeld = convertToUrl(stFeld)
014     ELSE
015        stFeld = "http://"+stFeld
016        stFeld = convertToUrl(stFeld)
017     END IF
018     oShell = createUnoService("com.sun.star.system.SystemShellExecute")
019     oShell.execute(stFeld,,0)
020  END SUB
```

Mit dieser Prozedur wird eine Mailadresse als Empfängeradresse im E-Mail-Programm der grafischen Benutzeroberfläche des Betriebssystems oder eine Website im Browser geöffnet. Die Art, eine Datei über das damit verknüpfte Programm zu öffnen, lässt sich auf alle möglichen Dateitypen übertragen, für die eben ein Programm zur Verfügung steht.

Ausgangspunkt ist direkt das Feld, in dem der Klick mit der Maus erfolgt (Zeile 5). Der Text, den das Feld anzeigt, kann ausgelesen werden (Zeile 6). Enthält das Feld keinen Text, so braucht auch kein Browser oder Mailprogramm gestartet zu werden (Zeile 7 bis 9). Die Prozedur wird also bei einem leeren Feld abgebrochen.

Enthält der Text ein «@», so handelt es sich um eine E-Mail-Adresse. Die Gültigkeit der Adresse wird hier nicht weiter überprüft. Für eine E-Mail-Adresse startet der Aufruf des Mailprogramms mit **mailto:** (Zeile 10 und 11).

Beginnt der Text mit **http://** oder **https://**, so handelt es sich um eine vollständige Internetadresse (Zeile 12). Solch eine Adresse kann direkt mit **convertToUrl** in eine Schreibweise umgewandelt werden, die auch URL-konform ist. Das sonst übliche **file:///** wird durch die Funktion nicht vor die Adresse gesetzt.

Ist in dem Text weder @ noch **http://** enthalten, so wird hier davon ausgegangen, dass es sich um eine Internetadresse ohne den entsprechenden Vorspann handelt (Zeile 15 und 16). Entsprechend wird die Adresse mit einem **http://** ergänzt. Würde dies nicht erfolgen, so würde die Adresse wie ein Dateipfad auf dem eigenen Rechner interpretiert. Bei einem tatsächlich existierenden Pfad zu z. B. einer *.png-Datei würde dann ein Bildbetrachtungsprogramm geöffnet.

Auch für die Internetadressen gilt, dass sie innerhalb des Makros nicht auf Gültigkeit überprüft werden. Es wird in Zeile 19 lediglich die entsprechende Adresse über die Shell an die Benutzeroberfläche weiter gegeben, die dann ein dafür entsprechendes Programm sucht und öffnet.

Mail_Aufruf

Aufruf aus
Formular: *Kontakte_Mail_Web, Mail_mit_Anhang*

Benötigt
Tabelle: *Kontakte, Mail_CC*
Makro: *FormSave*

```
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 oColumns AS OBJECT
011    DIM oDatenquelle AS OBJECT
012    DIM oVerbindung AS OBJECT
013    DIM oSQL_Anweisung AS OBJECT
014    DIM oAbfrageergebnis AS OBJECT
015    DIM inIndex AS INTEGER
016    DIM i AS INTEGER
017    DIM loMails_ID AS LONG
018    DIM stFeld1 AS STRING
019    DIM stFeld2 AS STRING
020    DIM stFeld3 AS STRING
021    DIM stFeld4 AS STRING
022    DIM stSql AS STRING
023    DIM stMail_CC AS STRING
024    DIM stMail_BCC AS STRING
025    DIM st_CC AS STRING
026    DIM st_BCC AS STRING
027    DIM arSo_Zeichen()
028    DIM arSo_Ersatz()
029    oDoc = thisComponent
030    oDrawpage = oDoc.Drawpage
031    oForm = oDrawpage.Forms.getByName("Formular")
032    oFeld1 = oForm.getByName("E_Mail")
033    oSubForm = oForm.getByName("UnterFormular")
034    oFeld2 = oSubForm.getByName("Betreff")
035    oFeld3 = oSubForm.getByName("Inhalt")
036    stFeld1 = oFeld1.Text
037    IF stFeld1 = "" THEN
038       msgbox "Keine Mailadresse vorhanden." & CHR(13) &
          "Das Mailprogramm wird nicht aufgerufen" , 48, "Mail senden"
039       EXIT SUB
040    END IF
041    FormSave(oSubForm)
042    oColumns = oSubForm.Columns
043    inIndex = oSubForm.findColumn("ID")
044    loMails_ID = oSubForm.getLong(inIndex)
045    stFeld2 = Mid(ConvertToUrl(oFeld2.Text),9)
046    stFeld3 = Mid(ConvertToUrl(oFeld3.Text),9)
047    arSo_Zeichen = Array("\","^","~","[","]"," ","&","+",",",";",":","=","?","@")
```

```
048     arSo_Ersatz = Array("5C","5E","7E","5B","5D","20","26","2B","2C","3B",
           "3A","3D","3F","40")
049     FOR i = LBound(arSo_Zeichen()) TO UBound(arSo_Zeichen())
050        stFeld3 = Join(Split(stFeld3,arSo_Zeichen(i)),"%" & arSo_Ersatz(i))
051     NEXT
052     stMail_CC = ""
053     stMail_BCC = ""
054     st_CC = ""
055     st_BCC = ""
056     oDatenquelle = ThisComponent.Parent.CurrentController
057     IF NOT (oDatenquelle.isConnected()) THEN
058        oDatenquelle.connect()
059     END IF
060     oVerbindung = oDatenquelle.ActiveConnection()
061     oSQL_Anweisung = oVerbindung.createStatement()
062     stSql = "SELECT ""Kontakte"".""E_Mail"", ""Mail_CC"".""BCC"" FROM ""Mail_CC"",
           ""Kontakte"" WHERE ""Mail_CC"".""Kontakte_ID"" = ""Kontakte"".""ID"" AND
           ""Mail_CC"".""Mails_ID"" = '"+ loMails_ID +"'"
063     oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
064     WHILE oAbfrageergebnis.next
065        IF oAbfrageergebnis.getBoolean(2) = TRUE THEN
066           stMail_BCC = stMail_BCC + oAbfrageergebnis.getString(1) + ","
067        ELSE
068           stMail_CC = stMail_CC + oAbfrageergebnis.getString(1) + ","
069        END IF
070     WEND
071     IF stMail_BCC <> "" THEN
072        st_BCC = Left(stMail_BCC,Len(stMail_BCC)-1)
073        stMail_BCC = "&bcc="+st_BCC
074        st_BCC = ",bcc='"+st_BCC+"'"
075     END IF
076     IF stMail_CC <> "" THEN
077        st_CC = Left(stMail_CC,Len(stMail_CC)-1)
078        stMail_CC = "&cc="+st_CC
079        st_CC = ",cc='"+st_CC+"'"
080     END IF
081     oShell = createUnoService("com.sun.star.system.SystemShellExecute")
082     oShell.execute("mailto:"+stFeld1+"?subject="+stFeld2+"&body="+stFeld3
           +stMail_CC+stMail_BCC,,0)
083     ' Shell("thunderbird -compose ""to="+stFeld1+st_CC+st_BCC+",subject="+stFeld2+
           ",body="+stFeld3+",attachment='file:///home/user/Testdatei.pdf'""")
084 END SUB
```

Über den Druck auf den Button E-Mail an Mailprogramm übergeben werden Adresse, Betreff, Inhalt, Kopieadressen und Blindkopieadressen an das Mailprogramm weiter gereicht. Der Mailaufruf erfolgt mit **`mailto:Empfänger?subject=...&body=...&cc=...&bcc=...`** (Zeile 83). Anhänge sind laut Definition von **`mailto`** nicht definiert. Manchmal funktioniert allerdings trotzdem **`attachment=...`** . Bei dem Mailprogramm Thunderbird ist **`attachment=...`** aus Sicherheitsgründen vom Betrieb mit **`mailto`** ausgenommen. Am Beispiel von Thunderbird wird noch gezeigt, wie auch der Versand mit Mailanhängen auf diesem Weg gelingen kann (Zeile 84). Das Formular müsste dazu aber sinnvollerweise noch mindestens mit einem Dateiauswahlfeld ergänzt werden, da vermutlich nicht jede Person den gleichlautenden Mailanhang erhalten soll. Das Anhangsproblem ist mit dem Formular *Mail_mit_Anhang* und der entsprechenden Prozedur *Mail_Attachment* gelöst worden.

Nachdem alle Variablen deklariert sind wird der Pfad zu den verschiedenen Feldern geklärt (Zeilen 29 bis 35). Enthält der Datensatz aus der Tabelle "Kontakte" keinen Eintrag (E-Mail-Adresse), so wird die Prozedur direkt abgebrochen (Zeile 36 bis 39).

Anschließend wird das «UnterFormular», dem die Tabelle "Mails" zugrunde liegt, noch einmal abgespeichert (Zeile 41) und das Feld "ID" aufgesucht. Ein über **`oSubForm.Columns`** angesprochenes Feld muss dafür nicht in dem Formular als Feld hinterlegt sein. Es ist in der Datenquelle des Formulars enthalten und kann deshalb für den aktuellen Datensatz ausgelesen werden (Zeile 44).

Der Text im «Betreff» und im «Inhalt» muss in einen URL-konformen Text umgewandelt, da sonst Sonderzeichen und Zeilenumbrüche nicht übernommen werden. Der Eintrag **file:///** wird dabei automatisch hinzugefügt und muss wieder entfernt werden. Der eigentliche Text beginnt also ab dem 9. Zeichen des URL-konformen Textes (Zeile 45 und 46).

Leider greift die Konvertierung zur URL bei einigen Sonderzeichen nicht. Nach Kommas bricht so z.B. der Text ab. Deswegen muss der bereits über die Funktion **ConverToUrl** bearbeitete Text noch einmal auf verschiedene Zeichen durchsucht werden. Über zwei Arrays werden hier noch diverse Sonderzeichen zusätzlich getestet und ersetzt (Zeile 47 und 48). Das erste Array enthält solche Sonderzeichen, das zweite Array an genau der gleichen Arrayposition die URL-konforme Schreibweise im Hexadezimalcode. Der Text für den Mailinhalt wird hier an jeder Stelle, an der ein Zeichen aus **arSo_Zeichen** vorkommt, in Einzelteile zerschnitten. Anschließend wird das Ergebnis wieder mit dem entsprechenden Zeichen aus **arSo_Ersatz** zusammengefügt.

Die Variablen für das CC(Carbon Copy - Kopie) bzw. BCC (Blind Carbon Copy - Blindkopie) werden als leere Textvariablen vordefiniert. Die Variablen **st_CC** und **st_BCC** sind hier nur eingefügt worden, da die Behandlung dieser Felder direkt über den Shellaufruf von Thunderbird (Zeile 83) anders funktioniert als mittels **mailto**.

Über eine Abfrage werden die Mailadressen ermittelt, an die ein Kopie bzw. Blindkopie weitergereicht werden soll (Zeile 62 bis 70). Eine Abfrage ist hier notwendig, da über das Formular nur auf genau einen Datensatz zugegriffen werden kann. Anschließend werden die ausgelesenen Werte für das abschließende Kommando zusammengestellt, mit dem das Mailprogramm gestartet werden soll.

Die Variablen **stMail_CC** sowie **stMail_BCC** sind nicht immer mit Inhalt belegt. Durch die in der vorhergehenden Zeile erfolgte Formulierung (Zeile 71 und 76) wird vermieden, dass ein «&cc=» ohne irgendeinen Inhalt in der Ausführung erscheint.

Für Thunderbird wird in Zeile 83 noch gezeigt, wie der Aufruf mit Parametern erfolgen kann. Dieser Aufruf ermöglicht es schließlich auch zusätzlich noch Anhänge an das Mailprogramm weiter zu geben. Der Aufruf von Thunderbird in den verschiedenen Betriebssystemen ist unterschiedlich. Hier wird nur der funktionierende Aufruf aus einer Linux-Umgebung gezeigt. Details zu dem Aufruf mit Parametern siehe auch: *http://www.thunderbird-mail.de/wiki/Aufrufparameter_von_Thunderbird*

Statt des **UnoServices** wird hier direkt mit dem Kommando **Shell()** die Kommandozeile des Systems angesprochen. Die angehängte Datei ist hier direkt im Code enthalten und müsste bei Nutzung dieser Funktion natürlich über das Formular ermittelt werden. Für nur einen Anhang reicht ein Feld in der Tabelle "Mails", für mehrere Anhänge müsste aber ähnlich wie bei der Tabelle "Mail_CC" eine Tabelle "Anhang" erstellt werden. Erst so lassen sich mehrere Anhänge mit einer Mail verschicken.

Der Shell-Befehl ist in dem der Beispieldatenbank beigefügten Makro natürlich als Kommentar geschrieben, so dass das Makro mit jedem in dem Betriebssystem verankerten Mailprogramm ohne Fehlermeldungen zu Ende laufen müsste.

Attachment

Aufruf aus
Formular: *Mail_mit_Anhang*

Benötigt
Makro: *FormSave*

```
001  SUB Attachment(oEvent AS OBJECT)
002     DIM oForm AS OBJECT
003     oForm = oEvent.Source.Model.Parent.Parent
```

```
004     FormSave(oForm)
005  END SUB
```

Die Anhänge werden über ein grafisches Kontrollfeld eingefügt. Eine Änderung in dem zugehörogen Tabellen-Kontrollfeld erzeugt über dieses Makro immer eine Änderung des aktuellen Datensatzes. Dass der Pfad innerhalb eines Tabellenkontrollfeldes liegt zeigt auch die Beziehung zwischen auslösendem Feld und Formular in Zeile 3: Hier wird nicht nur das direkte Elternelement (**Parent**) aufgesucht. Das direkte Elternelement für das Textfeld in dem Tabellen-Kontrollfeld ist nämlich das Tabellen-Kontrollfeld. Erst das Elternelenment des Tabellen-Kontrollfeldes ist das Formular, in dem dieses Feld liegt.

Mail_Attachment

Aufruf aus
Formular: *Mail_mit_Anhang*

Benötigt	
Tabelle:	*Kontakte, Mail_CC, Anhaenge*
Makro:	*FormSave*

```
001  SUB Mail_Attachment
002     DIM arAttach(0)
003     DIM arCC(0)
004     DIM arBCC(0)
005     DIM arURL_Start()
006     DIM oDoc AS OBJECT
007     DIM oDrawpage AS OBJECT
008     DIM oForm AS OBJECT
009     DIM oSubForm AS OBJECT
010     DIM oFeld1 AS OBJECT
011     DIM oFeld2 AS OBJECT
012     DIM oFeld3 AS OBJECT
013     DIM oColumns AS OBJECT
014     DIM oDatenquelle AS OBJECT
015     DIM oVerbindung AS OBJECT
016     DIM oSQL_Anweisung AS OBJECT
017     DIM oAbfrageergebnis AS OBJECT
018     DIM oMailProgram AS OBJECT
019     DIM oNewMessage AS OBJECT
020     DIM inIndex AS INTEGER
021     DIM i AS INTEGER
022     DIM k AS INTEGER
023     DIM loMails_ID AS LONG
024     DIM stFeld1 AS STRING
025     DIM stFeld2 AS STRING
026     DIM stFeld3 AS STRING
027     DIM stSql AS STRING
028     oDoc = thisComponent
029     oDrawpage = oDoc.Drawpage
030     oForm = oDrawpage.Forms.getByName("Formular")
031     oFeld1 = oForm.getByName("E_Mail")
032     oSubForm = oForm.getByName("UnterFormular")
033     oFeld2 = oSubForm.getByName("Betreff")
034     oFeld3 = oSubForm.getByName("Inhalt")
035     stFeld1 = oFeld1.Text
036     IF stFeld1 = "" THEN
037        msgbox "Keine Mailadresse vorhanden." & CHR(13) & "Das Mailprogramm wird
              nicht aufgerufen" , 48, "Mail senden"
038        EXIT SUB
039     END IF
040     FormSave(oForm)
041     stFeld2 = oFeld2.Text
```

```
042     stFeld3 = oFeld3.Text
043     oColumns = oSubForm.Columns
044     inIndex = oSubForm.findColumn("ID")
045     loMails_ID = oSubForm.getLong(inIndex)
046     oDatenquelle = ThisComponent.Parent.CurrentController
047     IF NOT (oDatenquelle.isConnected()) THEN
048        oDatenquelle.connect()
049     END IF
050     oVerbindung = oDatenquelle.ActiveConnection()
051     oSQL_Anweisung = oVerbindung.createStatement()
052     stSql = "SELECT ""Kontakte"".""E_Mail"", ""Mail_CC"".""BCC"" FROM ""Mail_CC"",
           ""Kontakte"" WHERE ""Mail_CC"".""Kontakte_ID"" = ""Kontakte"".""ID"" AND
           ""Mail_CC"".""Mails_ID"" = '"+ loMails_ID +"'"
053     oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
054     i = 0
055     k = 0
056     WHILE oAbfrageergebnis.next
057        IF oAbfrageergebnis.getBoolean(2) = TRUE THEN
058           IF i > 0 THEN
059              ReDim Preserve arBCC(i)
060           END IF
061           arBCC(i) = oAbfrageergebnis.getString(1)
062           i = i + 1
063        ELSE
064           IF k > 0 THEN
065              ReDim Preserve arCC(k)
066           END IF
067           arCC(k) = oAbfrageergebnis.getString(1)
068           k = k + 1
069        END IF
070     WEND
071     stSql = "SELECT ""Anhang"" FROM ""Anhaenge"" WHERE ""Mails_ID"" =
           '"+ loMails_ID +"'"
072     oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
073     arUrl_Start = split(oDoc.Parent.Url,oDoc.Parent.Title)
074     i = 0
075     WHILE oAbfrageergebnis.next
076        IF i > 0 THEN
077           ReDim Preserve arAttach(i)
078        END IF
079        arAttach(i) = ConvertToUrl(arUrl_Start(0) & oAbfrageergebnis.getString(1))
080        i = i + 1
081     WEND
082     IF GetGuiType() = 1 THEN
083        oMailer = createUnoService("com.sun.star.system.SimpleSystemMail")
           ' Sonst Linux/Mac
084     ELSE
085        oMailer = createUnoService("com.sun.star.system.SimpleCommandMail")
086     END IF
087     oMailProgram = oMailer.querySimpleMailClient()
088     oNewMessage = oMailProgram.createSimpleMailMessage()
089     oNewMessage.Recipient = stFeld1
090     oNewMessage.CcRecipient = arCC()
091     oNewMessage.BccRecipient = arBCC()
092     oNewMessage.Subject = stFeld2
093     oNewMessage.Body = stFeld3
094     oNewMessage.setAttachement(arAttach())
095     oMailProgram.sendSimpleMailMessage(oNewMessage, 0 )
096  END SUB
```

Diese Prozedur hat erst einmal sehr viele Ähnlichkeiten mit der Prozedur *Mail_Aufruf*. Zu Beginn unterscheiden sich die Variablen lediglich dadurch, dass die Attachments, CCs und BCCs in Arrays zusammengefasst werden (Zeile 2 bis 4).

Unter Linux funktioniert diese Prozedur nur dann einwandfrei, wenn in **Extras → Optionen → Internet → E-Mail** ein Mailprogramm ausgesucht wird. Ohne Angabe des Mailprogramm bereiten Kommas und Zeilenumbrüche Probleme.

Nach dem Auslesen der Inhalte aus den Formularfeldern erfolgt durch den SQL-Code in Zeile 52 das Auslesen der Einträge für das CC bzw. das BCC. In einer Schleife von Zeile 56 bis 70 wird wieder zwischen BCC und CC unterschieden. Beim ersten Eintrag wird der Wert direkt in das passende Array geschrieben (Zeile 61 bzw. Zeile 67). Der Zähler für das jeweilige Array wird um 1 erhöht. Muss ein neuer Wert eingetragen werden, so muss erst einmal das Array neu dimensioniert werden (Zeile 59 und 65). Diese Dimenionierung erfolgt so, dass der alte Inhalt erhalten bleibt. Erst nach der Dimensionierung kann auch ein weiterer Wert in das, jetzt größere, Array eingefügt werden.

Anschließend erfolgt ein ähnliches Vorgehen für die Liste der Anhänge (Zeile 71 bis 81). Hier muss lediglich zusätzlich beachtet werden, dass die Abspeicherung in der Base-Datei mit relativen Pfaden erfolgt. Das Mailprogramm braucht aber zum Auffinden des Anhangs den absoluten Pfad. So wird in Zeile 73 über **`Split`** der absolute Pfad der Basedatei in den Pfad zur Datei und den Dateinamen aufgeteilt. Der erste Teil dieses Arrays wird anschließend vor die jeweiligen relativen Pfade zu den Anlagen geschrieben (Zeile 79).

In Abhängigkeit vom Betriebssystem (Zeile 82) wird entweder SimpleSystemMail (Windows) oder SimpleCommandMail (Linux, Mac) für die Zusammenstellung der Mail genutzt.

Über Zeile 87 wird die Verbindung zum Mailprogramm hergestellt. Zeile 88 liefert dann den Code für die Mail, die versandt werden soll. Zu beachten ist hier, dass eben die Einträge zu CC, BCC und Attachment jeweils Arrays sind. Der Inhalt für die eigentliche Mail wird in Body (Zeile 93) weitergegeben. Diese Variable ist erst seit LO 4.2 enthalten und hat nicht, wie die anderen Variablen, auch noch den Zugang z.B. über setSubject oder setAttachment. Zu den möglichen Parametern siehe: *https://api.libreoffice.org/docs/idl/ref/interfacecom_1_1sun_1_1star_1_1system_1_1XSimpleMailMessage.html*.

In Zeile 95 wird die Mail schließlich an das Mailprogramm zur Absendung übergeben. Zu den möglichen Absendeparametern siehe *https://api.libreoffice.org/docs/idl/ref/interfacecom_1_1sun_1_1star_1_1system_1_1XSimpleMailClient.html*.

Es ist möglich, diese Mail auch unter Umgehung des Mailprogramms zu senden. Die Parameter zu sendSimpleMailMessage: **`DEFAULTS = 0; NO_USER_INTERFACE = 1; NO_LOGON_DIALOG = 2`**
Ein reibungsloser Ablauf bedingt bei 1: Der Empfänger muss angegeben sein
Ein reibungsloser Ablauf bedingt bei 2: Es darf kein Dialog zum Einloggen erforderlich sein
Woher bei 1 und 2 die Zugangsdaten, das Passwort und eventuell eine Verschlüsselung kommen sollen bleibt unklar.

Suchen und Filtern

Einführung

Base selbst bietet sowohl eine Suchfunktion als auch eine Möglichkeit, Daten zu filtern.

Beim Suchen wird schrittweise der gesamte Datenbestand einer Tabelle durchlaufen und die Treffer nacheinander markiert. Die Anzahl der Ergebnisdatensätze wird nicht auf Ergebnisse eingeschränkt, auf die das Suchkriterium zutrifft.

Beim Filtern wird durch einen oder mehrere Filterbegriffe direkt die Ergebnismenge aus der Tabelle heraus beeinflusst. Taucht das gewünschte Kriterium nur bei 3 von 100 Datensätzen auf, so werden eben nur 3 Datensätze angezeigt. Die Suchfunktion hingegen würde die 100 Datensätze durchlaufen und die einzelnen Treffer nacheinander markieren.

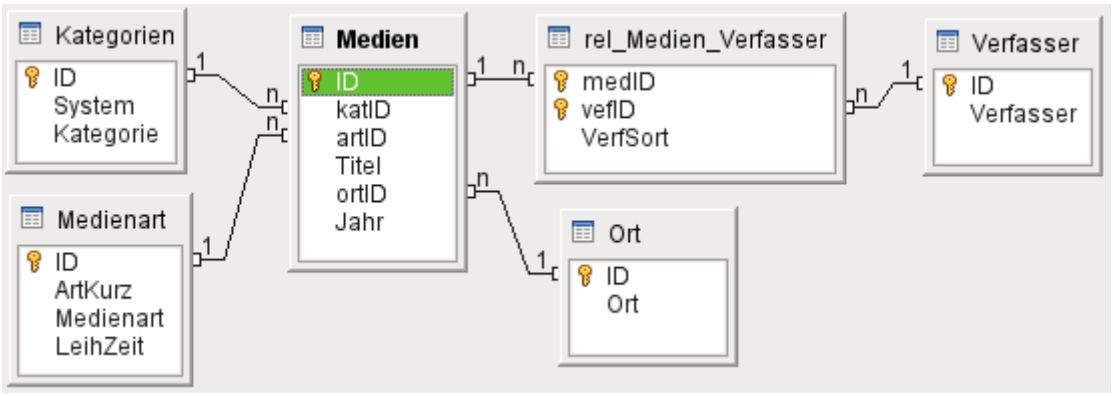
Die eingebaute Suchfunktion ist allerdings recht langsam, die Datenfilterung erfordert jeweils Einstellungen unterschiedlicher Werte und Zuordnungen. Die Datenfilterung ist sehr gut universell nutzbar, sofern sich sämtliche zu durchsuchenden Daten in einer Tabelle oder Abfrage befinden. Bei immer wiederkehrenden ähnlichen Filterungen führt sie aber nicht so schnell zum Erfolg, wie dies vor allem in Formularen erwünscht wird.

Als praktische Beispiel[7] soll ein Teil einer Mediendatenbank dienen, die in verschiedene Tabellen aufgeteilt ist. Gerade eine Verteilung auf mehrere Tabellen erfordert für ein ordnungsgemäßes Filtern von Daten mehr als nur die Nutzung der internen Funktionen von Base. Schließlich ist die eingebaute Funktionalität auf das Filtern innerhalb einer Tabelle bzw. eines Formulars ohne Unterformular beschränkt.

Der Begriff «Suche» wird in den folgenden Beispielen allerdings so genutzt, wie er auch im Internet üblich ist: Die Eingabe eines beliebigen Suchbegriffs soll zu Ergebnissen führen.

Die Reihenfolge in der Beschreibung entspricht nicht der Reihenfolge, die Abfragen und Tabellen in der Datenbank haben. Dies liegt daran, dass die inhaltliche Reihenfolge vom Einfacheren zum Umfassenderen nicht mit der alphabetischen Reihenfolge in der Datenbank übereinstimmt.

Tabellen



Über **Extras → Beziehungen** ist eine Übersicht der Tabellen zusammengefügt, durch die hindurch die Datensuche und Datenfilterung vorgenommen werden soll.

Zentrale Tabelle ist die Tabelle "Medien". Die Tabelle "Kategorien", "Medienart" und "Ort" sind mit der Tabelle über Fremdschlüssel verbunden. Für einen Datensatz aus "Medien" kann es mehrere Datensätze aus "Verfasser" geben. Ebenso kann es für einen Datensatz aus "Verfasser" mehrere Datensätze aus "Medien" geben. Aus dem Grunde ist über die Tabelle "rel_Medien_Verfasser" eine n:m-Beziehung erstellt worden.

7 Beispieldatenbank Beispiel_Suchen_Filtern.odb

Neben diesen Tabellen existiert in der Datenbank noch eine Tabelle "Filter", die zur Filterung und Suche genutzt wird und sonst keine Beziehung zu den anderen Tabellen aufweist.

Filter

Datenziel	
Formular:	*Filter_Formular, Filter_Formular_Nullwerte, Filter_Formular_Subformular, Filter_Formular_Subformular_3Filter, Makro_Filter_Formular, Filter_Formular_Listfeldbegrenzung, Makro_Filter_Formular_Subformular, Filter_Formular_Subformular_bedingende_Filter, Filter_Formular_Subformular_bedingende_Filter_allround, Filter_Formular_Subformular_bedingende_Filter_Datensaetze, Suche_Filter_Formular_Subformular_ohne_Makros, Suche_Filter_hierarchisch_mit_Makros, Suche_Filter_hierarchisch_allround_Datensaetze_mit_Makros, Parameter_Suche_Formular_Subformular, Suche_Formular, Suche_Formular_Subformular, Makro_Suche_Formular, Makro_Suche_Formular_Subformular*
Abfrage:	*Filter_Form, Filter_Form_Subform, Filter_Form_Subform_3Filter, Filter_Ansicht, Suche_Filter_Form_Subform_3Filter, Filter_Leerfeld, Filter_Such_Ansicht, Suche_Form, Suche_Form_Subform*
Makro:	*Filter_Suche_hierarchisch_Start*
Ansicht, Bericht: keine	

Feldname	***Feld-typ***	***Beschreibung***
ID	Ja/Nein	Primärschlüssel der Tabelle. Dieses Feld wird zum Start einmal auf 'Ja' (**True**) gesetzt. Damit steht der erste und einzige Datensatz fest.
Suche	Text	Texteingabe für eine freie Textsuche Eingabe erforderlich: Nein
Filter_Kategorie	Integer	Eingabe des Fremdschlüsselwertes für die Tabelle "Kategorie" Eingabe erforderlich: Nein
Filter_Verfasser	Integer	Eingabe des Fremdschlüsselwertes für die Tabelle "Verfasser" Eingabe erforderlich: Nein
Filter_Medienart	Integer	Eingabe des Fremdschlüsselwertes für die Tabelle "Medienart" Eingabe erforderlich: Nein
pTitel	Text	Zwischenspeicher für eine Parameterabfrage über ein Formular, Feld "Titel" Eingabe erforderlich: Nein
pVerfasser	Text	Zwischenspeicher für eine Parameterabfrage über ein Formular, Feld "Verfasser" Eingabe erforderlich: Nein
pOrt	Text	Zwischenspeicher für eine Parameterabfrage über ein Formular, Feld "Ort" Eingabe erforderlich: Nein
pMedienart	Text	Zwischenspeicher für eine Parameterabfrage über ein Formular, Feld "Medienart" Eingabe erforderlich: Nein

Die Filtertabelle wird nur mit einem Datensatz versehen. Hier werden alle Filter- und Suchergebnisse zwischengespeichert. Wird der Inhalt nicht gelöscht, so bleiben die letzten Suchergebnisse gespeichert.

Kategorien

Datenziel
Formular: Listenfeld «Kategorie» in jedem Formular, *Filter_Formular, Makro_Filter_Formular*
Ansicht: *Ansicht_Suche*
Abfrage: *Listenfeld_Kategorie_bedingt, Listenfeld_Kategorie_bedingt_Datensaetze, Listenfeld_Kategorie_ohne_Eintrag, Listenfeld_Kategorie_bedingt_Suche_Filter, Listenfeld_Kategorie_bedingt_Suche_Filter_Datensaetze*
Bericht, Makro: keine

Feldname	***Feldtyp***	***Beschreibung***
ID	Integer	Primärschlüssel der Tabelle. Der Primärschlüssel muss eindeutig sein. Das Feld kann ist als Auto-Wert-Feld gesetzt.
System	Text	Kürzel für die systematische Einordnung Eingabe erforderlich: Ja
Kategorie	Text	Ausführliche Beschreibung der Kategorie Eingabe erforderlich: Ja

Medien

Datenziel
Formular: *Filter_Formular, Standardfilter_Formular_Subformular, Standardfilter_Formular_Subformular_Datensatzkorrektur, Makro_Filter_Formular*
Ansicht: *Ansicht_Suche*
Abfrage: *Filter_Form, Parametersuche_Medien, Filter_Form_Subform, Filter_Form_Subform_3Filter, Suche_Filter_Form_Subform_3Filter, Filter_Form_Nullwerte, Listenfeld_Kategorie_ohne_Eintrag, Suche_Form, Suche_Form_Subform*
Bericht, Makro: keine

Feldname	***Feldtyp***	***Beschreibung***
ID	Integer	Primärschlüssel der Tabelle. Der Primärschlüssel muss eindeutig sein. Das Feld kann auch als Auto-Wert-Feld gesetzt werden.
katID	Integer	Fremdschlüssel zur Tabelle "Kategorie"."ID" Eingabe erforderlich: Nein
artID	Integer	Fremdschlüssel zur Tabelle "Medienart"."ID" Eingabe erforderlich: Nein
Titel	Text	Titel des Mediums Eingabe erforderlich: Ja
ortID	Integer	Fremdschlüssel zur Tabelle "Ort"."ID" Eingabe erforderlich: Nein
Jahr	Small Integer	Erscheinungsjahr des Mediums Eingabe erforderlich: Nein

Medienart

Datenziel	
Formular:	Listenfeld «Medienart» in jedem Formular
Ansicht:	*Ansicht_Suche*
Abfrage:	*Listenfeld_Medienart_bedingt, Listenfeld_Medienart_bedingt_Datensaetze, Listenfeld_Medienart_bedingt_Suche_Filter, Listenfeld_Medienart_bedingt_Suche_Filter_Datensaetze*
Bericht, Makro: keine	

Feldname	***Feldtyp***	***Beschreibung***
ID	Integer	Primärschlüssel der Tabelle. Der Primärschlüssel muss eindeutig sein. Das Feld ist als Auto-Wert-Feld gesetzt.
ArtKurz	Text(fix)	Kurzform der Medienart, auf 2 Zeichen festgelegt Eingabe erforderlich: Ja
Medienart	Text	Medienart komplett beschrieben, auf 25 Zeichen festgelegt Eingabe erforderlich: Ja
Leihzeit	Tiny Integer	Ausleihzeit für das Medium in Tagen Eingabe erforderlich: Nein

Ort

Datenziel	
Formular:	Listenfeld «Ort» in jedem Formular
Ansicht:	*Ansicht_Suche*
Abfrage, Bericht, Makro: keine	

Feldname	***Feldtyp***	***Beschreibung***
ID	Integer	Primärschlüssel der Tabelle. Der Primärschlüssel muss eindeutig sein. Das Feld ist als Auto-Wert-Feld gesetzt.
Ort	Text	Erscheinungsort des Mediums Eingabe erforderlich: Ja

rel_Medien_Verfasser

Datenziel	
Formular:	*Filter_Formular, Filter_Formular_Nullwerte, Filter_Formular_Subformular, Filter_Formular_Subformular_3Filter, Standardfilter_Formular_Subformular, Standardfilter_Formular_Subformular_Datensatzkorrektur, Makro_Filter_Formular, Filter_Formular_direkt_Nullwerte, Filter_Formular_Listfeldbegrenzung, Makro_Filter_Formular_Subformular, Filter_Formular_Subformular_bedingende_Filter, Filter_Formular_Subformular_bedingende_Filter_allround, Filter_Formular_Subformular_bedingende_Filter_Datensaetze, Suche_Filter_hierarchisch_mit_Makros, Suche_Filter_hierarchisch_allround_Datensaetze_mit_Makros, Parameter_Suche_Formular_Subformular, Suche_Formular, Suche_Formular_Subformular, Makro_Suche_Formular, Makro_Suche_Formular_Subformular*
Ansicht:	*Ansicht_Suche*
Abfrage:	*Listenfeld_Verfasser_bedingt, Listenfeld_Verfasser_bedingt_Suche_Filter*

Datenziel
Bericht, Makro: keine

Feldname	Feldtyp	Beschreibung
medID	Integer	Primärschlüssel der Tabelle zusammen mit vefID. Fremdschlüssel zur Tabelle "Medien"."ID"
vefID	Integer	Primärschlüssel der Tabelle zusammen mit medID. Fremdschlüssel zur Tabelle "Verfasser"."ID"
VerfSort	Tiny Integer	Sortierreihenfolge für die Verfassereinträge, damit die wichtigsten Personen nachher auch am Anfang erscheinen. Eingabe erforderlich: Nein

Verfasser

Datenziel
Formular: Listenfeld «Verfasser» in jedem Formular
Ansicht: *Ansicht_Suche*
Abfrage: *Listenfeld_Verfasser_bedingt, Listenfeld_Verfasser_bedingt_Suche_Filter*
Bericht, Makro: keine

Feldname	Feldtyp	Beschreibung
ID	Integer	Primärschlüssel der Tabelle. Der Primärschlüssel muss eindeutig sein. Das Feld ist als Auto-Wert-Feld gesetzt.
Verfasser	Text	Der Name der Person, die Autor des Mediums ist. Über die Tabelle "rel_Medien_Verfasser" können beliebig viele Verfasser einem Medium zugeordnet werden.

Ansichten

Ansicht_Suche

Datenquelle
Tabelle: *Kategorien, Medien, Medienart, Ort, rel_Medien_Verfasser, Verfasser*

Datenziel
Formular: *Standardfilter_Formular_Subformular*
Abfrage: *Parametersuche_Medien, Filter_Form_Subform, Filter_Form_Subform_3Filter, Filter_Ansicht, Suche_Filter_Form_Subform_3Filter, Filter_Such_Ansicht, Suche_Form_Subform, Listenfeld_Verfasser_bedingt_Datensaetze_langsam*

```
001 SELECT "Medien"."ID", "Medien"."katID", "Medien"."artID",
    "Medien"."Titel", "Medien"."ortID", "Medien"."Jahr",
    "Kategorien"."System", "Kategorien"."Kategorie", "Medienart"."ArtKurz",
    "Medienart"."Medienart", "Medienart"."LeihZeit", "Ort"."Ort",
    "Verfasser"."Verfasser", "rel_Medien_Verfasser"."VerfSort",
    "rel_Medien_Verfasser"."vefID",
002 COALESCE("rel_Medien_Verfasser"."vefID", 0 ) AS "VerfasserID",
003 COALESCE("Medien"."katID", 0 ) AS "KategorieID",
```

```
004 COALESCE("Medien"."artID", 0 ) AS "MedienartID"
005 FROM "Medien"
006 LEFT JOIN "Kategorien" ON "Medien"."katID" = "Kategorien"."ID"
007 LEFT JOIN "Medienart" ON "Medien"."artID" = "Medienart"."ID"
008 LEFT JOIN "Ort" ON "Medien"."ortID" = "Ort"."ID"
009 LEFT JOIN "rel_Medien_Verfasser" ON "rel_Medien_Verfasser"."medID" =
    "Medien"."ID"
010 LEFT JOIN "Verfasser" ON "rel_Medien_Verfasser"."vefID" = "Verfasser"."ID"
```

Mit dieser Ansicht werden alle Inhalte aus den Tabellen der Datenbank in einer Ansicht zusammengefasst. Dabei wird von der Tabelle "Medien" als zentraler Tabelle ausgegangen (Zeile 5). Alle anderen Tabellen werden über einen **LEFT JOIN** mit "Medien" verbunden.

Die Zeilen 2 bis 4 weisen den Feldern, die nicht über einen Fremdschlüssel mit den entsprechenden Tabellen verbunden sind, für eine entsprechende Abfrage den Wert '0' zu. Die Zählung des Primärschlüssel in den Tabellen geht in dieser Beispieldatenbank erst ab '1' los, so dass eine '0' dort nicht vorkommt.

Ansicht_Filter_Start

Datenquelle
Ansicht: *Ansicht_Suche*

Datenziel
Ansicht: *Ansicht_Standardfilter*

```
001 SELECT "ID", "System", "Kategorie", "Titel", "Jahr", "ArtKurz",
    "Medienart", "LeihZeit", "Ort",
002 ( SELECT COUNT( "ID" ) FROM "Ansicht_Suche" WHERE "ID" = "a"."ID" AND
    "vefID" <= "a"."vefID" ) AS "GruppenNr",
003 "Verfasser"
004 FROM "Ansicht_Suche" AS "a"
```

Diese Ansicht stellt neben einigen Inhalten aus der "Ansicht_Suche" eine fortlaufende Nummerierung für die Verfasser jedes einzelnen Mediums zur Verfügung (Zeile 2). Die interne HSQLDB ist nicht in der Lage, die Inhalte des Feldes "Verfasser" so zusammen zu fassen, dass pro Medium die Verfasser in einem Feld erscheinen. Die ursprüngliche Ansicht zeigt dadurch deutlich mehr Medien an als wirklich vorhanden sind, da eben einige Medien mehrere Verfasser haben.

Leider ist diese Ansicht recht langsam.

Ansicht_Standardfilter

Datenquelle
Ansicht: *Ansicht_Filter_Start*

Datenziel
Formular: *Standardfilter_Formular_Subformular_Datensatzkorrektur*

```
001 SELECT DISTINCT "ID", "System", "Kategorie", "Titel", "Jahr", "ArtKurz",
    "Medienart", "LeihZeit", "Ort",
002 ( SELECT "Verfasser" FROM "Ansicht_Filter_Start" WHERE "ID" = "a"."ID" AND
    "GruppenNr" = 1 ) ||
003 COALESCE( '; ' || ( SELECT "Verfasser" FROM "Ansicht_Filter_Start" WHERE
    "ID" = "a"."ID" AND "GruppenNr" = 2 ), '' ) ||
```

```
004 COALESCE( '; ' || ( SELECT "Verfasser" FROM "Ansicht_Filter_Start" WHERE
    "ID" = "a"."ID" AND "GruppenNr" = 3 ), '' ) ||
005 COALESCE( '; ' || ( SELECT "Verfasser" FROM "Ansicht_Filter_Start" WHERE
    "ID" = "a"."ID" AND "GruppenNr" = 4 ), '' ) ||
006 COALESCE( '; ' || ( SELECT "Verfasser" FROM "Ansicht_Filter_Start" WHERE
    "ID" = "a"."ID" AND "GruppenNr" = 5 ), '' ) ||
007 COALESCE( '; ' || ( SELECT "Verfasser" FROM "Ansicht_Filter_Start" WHERE
    "ID" = "a"."ID" AND "GruppenNr" = 6 ), '' ) AS "Verfasser"
008 FROM "Ansicht_Filter_Start" AS "a"
```

Über die Gruppennummer werden die einzelnen Verfasser aus der "Ansicht_Filter_Start" in einem Feld "Verfasser" zusammengefasst. Die einzelnen Einträge werden hier über ein Semikolon voneinander getrennt. Ergibt sich bei einer "GruppenNr" ein leerer Wert, so wird stattdessen ein leerer Text weitergegeben. Sonst würden bei dieser Zusammenfassung im Feld "Verfasser" nur die Felder Inhalt aufweisen, bei denen 6 Verfasser vorhanden sind.

Nachteil dieser für die HSQLDB notwendigen Variante ist, dass die Anzahl der zusammengefassten Verfasser durch den Code begrenzt ist und dass leider die Ansicht durch die vorhergehende Ansicht recht langsam ist.

Standardfilter_Firebird

```
001 SELECT "ID", "Titel", "Jahr", "System", "Kategorie", "ArtKurz",
    "Medienart", "LeihZeit", "Ort",
002 LIST("Verfasser",';') AS "Verfasser"
003 FROM "Ansicht_Suche"
004 GROUP BY "Ansicht_Suche"."ID", "Ansicht_Suche"."Titel",
    "Ansicht_Suche"."Jahr", "Ansicht_Suche"."System",
    "Ansicht_Suche"."Kategorie", "Ansicht_Suche"."ArtKurz",
    "Ansicht_Suche"."Medienart", "Ansicht_Suche"."LeihZeit",
    "Ansicht_Suche"."Ort"
```

Dieser Code könnte in FIREBIRD statt der Kombination von *Ansicht_Filter_Start* und *Ansicht_Standardfilter* genutzt werden. Er ist wesentlich schneller und erfüllt den gleichen Zweck: Die Einträge für die Verfasser werden hier durch ein Semikolon getrennt zu jedem Medium einzeln aufgelistet (Zeile 2). Die Anzahl der Datensätze stimmt mit der Anzahl der Medien überein. Dieser Code ist in der Firebird-Variante der Beispieldatenbank unter den Abfragen gespeichert.

Abfragen

Zur besseren Übersicht sind die Abfragen in verschiedene Bereiche untergliedert. Neben den Abfragen, die nur für die Filterung oder die Suche von Daten gedacht sind existieren auch viele Abfragen, die für Versorgung von Listenfeldern mit entsprechendem SQL-Code gedacht sind.

Filter

Diese Abfragen werden nur für die Filterung von Daten genutzt.

Filter_Form

Datenquelle
Tabelle: *Medien, Filter*

Datenziel
Formular: *Filter_Formular, Filter_Formular_Nullwerte, Makro_Filter_Formular, Filter_Formular_Listfeldbegrenzung*

```
001 SELECT "Medien".*,
002 COALESCE( "katID", 0 ) AS "kat"
003 FROM "Medien"
004 WHERE "kat" = COALESCE( ( SELECT "Filter_Kategorie" FROM "Filter" WHERE
    "ID" = TRUE ), "kat" )
```

Alle Datensätze der Tabelle "Medien" werden angezeigt. Zusätzlich wird ein Feld mit dem Alias "kat" erzeugt, das bei einer leeren "katID" stattdessen eine '0' setzt (Zeile 2). Die Daten werden durch eine Abfrage des Wertes von "Filter_Kategorie" in der Tabelle "Filter" eingeschränkt (Zeile 4). Ist dieses Feld leer, so wird stattdessen "kat" direkt mit "kat" verglichen. Und da "kat" grundsätzlich einen Eintrag hat (Zeile 2) werden durch diese Bedingung bei einem leeren Feld "Filter_Kategorie" alle Datensätze aus "Medien" angezeigt.

In FIREBIRD führt diese Abfrage zu einer Fehlermeldung. FIREBIRD bekommt hier die Verbindung des Alias "kat" aus Zeile 2 mit dem "kat" in Zeile 4 nicht geregelt. Deswegen wurde dort der Code geändert:

```
001 SELECT *
002 FROM "Medien"
003 WHERE COALESCE ( "katID", 0 ) = COALESCE ( ( SELECT "Filter_Kategorie"
    FROM "Filter" WHERE "ID" = TRUE ), COALESCE ( "katID", 0 ) )
```

Der Inhalt aus der oberen Abfrage für Zeile 2 wurde einfach in die Bedingung direkt geschrieben. Da jetzt nur die Datensätze aus "Medien" angegeben werden entfällt der Hinweis zu "Medien" in Zeile 1. Statt **"Medien".*** reicht einfach *.

Filter_Form_Subform

Datenquelle	
Tabelle:	*Medien, Filter*
Ansicht:	*Ansicht_Suche*

Datenziel
Formular: *Filter_Formular_Subformular, Makro_Filter_Formular_Subformular*

```
001 SELECT * FROM "Medien"
002 WHERE "ID" IN (
003 SELECT DISTINCT "ID" FROM "Ansicht_Suche" WHERE "VerfasserID" =
    COALESCE( ( SELECT "Filter_Verfasser" FROM "Filter" WHERE "ID" = TRUE ),
    "VerfasserID" )
004 )
```

Die Angaben für die Verfasser befinden sich im Unterformular der jeweiligen Formulare. Deswegen ist hier eine gesonderte Vorgehensweise erforderlich.

Alle Datensätze aus der Tabelle "Medien" werden angezeigt (Zeile 1). Voraussetzung ist, dass der Primärschlüssel "ID" des jeweiligen Mediums in der Liste auftaucht, die durch die Unterabfrage in Zeile 3 erstellt wird. Die Unterabfrage listet nur die Werte auf, zu denen eine dem Feld "Filter_Verfasser" entsprechender Datensatz in "Ansicht_Suche"."VerfasserID" existiert. Wird kein Wert im Feld "Filter_Verfasser" eingegeben, so wird stattdessen "VerfasserID" gleich "VerfasserID" gesetzt. Alle Datensätze werden angezeigt.

Abfragen, die mit dem Vergleich von Datenmengen funktionieren (**IN**, Zeile 2), beanspruchen grundsätzlich etwas mehr Ressourcen. In der Praxis macht sich das bei dieser Abfrage allerdings nicht bemerkbar.

Filter_Form_Subform_3Filter

Datenquelle	
Tabelle:	*Medien, Filter*
Ansicht:	*Ansicht_Suche*

Datenziel	
Formular:	*Filter_Formular_Subformular_3Filter, Filter_Formular_Subformular_bedingende_Filter, Filter_Formular_Subformular_bedingende_Filter_allround, Filter_Formular_Subformular_bedingende_Filter_Datensaetze*
Abfrage:	*Listenfeld_Kategorie_bedingt, Listenfeld_Medienart_bedingt, Listenfeld_Verfasser_bedingt, Listenfeld_Kategorie_bedingt_Datensaetze, Listenfeld_Medienart_bedingt_Datensaetze, Listenfeld_Verfasser_bedingt_Datensaetze_langsam*

```
001 SELECT *
002 FROM "Medien"
003 WHERE "ID" IN (
004 SELECT DISTINCT "ID" FROM "Ansicht_Suche" WHERE
005 "VerfasserID" = COALESCE( ( SELECT "Filter_Verfasser" FROM "Filter" WHERE
    "ID" = TRUE ), "VerfasserID" )
006 AND "MedienartID" = COALESCE( ( SELECT "Filter_Medienart" FROM "Filter"
    WHERE "ID" = TRUE ), "MedienartID" )
007 AND "KategorieID" = COALESCE( ( SELECT "Filter_Kategorie" FROM "Filter"
    WHERE "ID" = TRUE ), "KategorieID" )
008 )
```

Diese Abfrage funktioniert nach dem gleichen Prinzip wie *Filter_Form_Subform*. Nur wird hier nicht nur nach den Datensätzen für die "VerfasserID" (Zeile 5) sondern auch nach denen für die "MedienartID" (Zeile 6) und die "KategorieID" (Zeile 7) gefiltert. Dabei ergänzen sich die Filter über **AND**. Das bedeutet, dass die Filterungen einander einschränken. Je mehr Filter gesetzt werden, desto kleiner ist die Ergebnismenge.

Filter_Ansicht

Datenquelle	
Tabelle:	*Filter*
Ansicht:	*Ansicht_Suche*

Datenziel	
Abfrage:	*Listenfeld_Verfasser_bedingt_Datensaetze_optimiert*

```
001 SELECT *
002 FROM "Ansicht_Suche"
003 WHERE "ID" IN (
004 SELECT DISTINCT "ID" FROM "Ansicht_Suche" WHERE
005 "VerfasserID" = COALESCE( ( SELECT "Filter_Verfasser" FROM "Filter" WHERE
    "ID" = TRUE ), "VerfasserID" )
006 AND "MedienartID" = COALESCE( ( SELECT "Filter_Medienart" FROM "Filter"
    WHERE "ID" = TRUE ), "MedienartID" )
007 AND "KategorieID" = COALESCE( ( SELECT "Filter_Kategorie" FROM "Filter"
    WHERE "ID" = TRUE ), "KategorieID" )
008 )
```

Der Code gleicht dem aus *Filter_Form_Subform_3Filter*. Allerdings wird hier nicht direkt die Tabelle "Medien", sondern die Ansicht "Ansicht_Suche" angezeigt. Diese Fassung ist für die Lis-

tenfeldabfrage deutlich besser nutzbar als die Abfrage von "Medien", da in der Ansicht bereits die Verbindung der Verfasser zu den Medien (n:m-Beziehung) vorher abgefragt wird.

Filter_Form_Nullwerte

Datenquelle
Tabelle: *Medien*

Datenziel
Formular: *Filter_Formular_direkt_Nullwerte*

```
001 SELECT "Medien".*, COALESCE( "katID", 0 ) AS "kat" FROM "Medien"
```

Damit bei leeren Felder für "katID" auch eine Auswahl dieser Felder möglich gemacht wird, wird in dieser Abfrage "katID" gegebenenfalls '0' zugewiesen. So kann in dem Formular nach den Datensätzen gefiltert werden, für die kein Eintrag existiert. Gleichzeitig ist aber auch eine Änderung der anderen Daten möglich.

Diese Abfrage ist für die direkte Filterung über den Formularfilter notwendig.

Filter_Leerfeld

Datenquelle
Tabelle: *Filter*

Datenziel
Formular: *Filter_Formular_direkt_Nullwerte,*

```
001 SELECT "Filter".*, ( SELECT NULL FROM "Filter" ) AS "Leer"
002 FROM "Filter" WHERE "ID" = TRUE
```

Der Tabelle "Filter" wird ein leeres Feld hinzugefügt. Dem Feld wird der Alias "Leer" zugewiesen. Dies ist notwenig als Datenbasis für das Listenfeld im Formular *Filter_Formular_direkt_Nullwerte*, da ein Listenfeld nur funktioniert, wenn es mit einer Datenquelle verbunden ist. Das Listenfeld soll aber keine Daten speichern.

Suche

Die Suche arbeitet grundsätzlich mit frei eingebbarem Text. Hier sind also keine Fremdschlüsselwerte vorgesehen wie bei der Filterung.

Suche_Form

Datenquelle
Tabelle: *Filter, Medien*

Datenziel
Formular: *Suche_Formular, Makro_Suche_Formular*

```
001 SELECT *
002 FROM "Medien"
003 WHERE LOWER ( "Titel" ) LIKE COALESCE( '%' || LOWER ( ( SELECT "Suche"
    FROM "Filter" WHERE "ID" = TRUE ) ) || '%', LOWER ( "Titel" ) )
```

Aus der Tabelle "Medien" werden alle Datensätze angezeigt (Zeile 1 und 2), bei denen die Kleinschreibung von "Titel" **LOWER("Titel")** den klein geschriebenen Inhalt aus dem Feld "Suche"

der Tabelle "Filter" an irgendeiner Stelle enthält. Durch die voran- und nachgestellten **'%'** darf dieses Element an jeder Stelle des Feldes "Titel" auftauchen. Enthält das Feld "Suche" keinen Eintrag, so wird über **COALESCE** stattdessen der Eintrag in **LOWER("Titel")** mit sich selbst verglichen. Das bedeutet, dass alle Datensätze angezeigt werden.

Durch die Anzeige aller Datensätze aus "Medien", auch des Primärschlüssels, ist diese Abfrage zur Eingabe von Daten im Formular geeignet.

Suche_Form_Subform

Datenquelle	
Tabelle:	*Filter, Medien*
Ansicht:	*Ansicht_Suche*

Datenziel
Formular: *Suche_Formular_Subformular, Makro_Suche_Formular_Subformular*

```
001 SELECT *
002 FROM "Medien"
003 WHERE "ID" IN (
004 SELECT DISTINCT "ID" FROM "Ansicht_Suche"
005 WHERE LOWER ( "Titel" ) LIKE COALESCE( '%' || LOWER ( ( SELECT "Suche"
    FROM "Filter" WHERE "ID" = TRUE ) ) || '%', LOWER ( "Titel" ) )
006 OR LOWER ( "Kategorie" ) LIKE '%' || LOWER ( ( SELECT "Suche" FROM
    "Filter" WHERE "ID" = TRUE ) ) || '%'
007 OR LOWER ( "Medienart" ) LIKE '%' || LOWER ( ( SELECT "Suche" FROM
    "Filter" WHERE "ID" = TRUE ) ) || '%'
008 OR LOWER ( "Ort" ) LIKE '%' || LOWER ( ( SELECT "Suche" FROM "Filter"
    WHERE "ID" = TRUE ) ) || '%'
009 OR LOWER ( "Verfasser" ) LIKE '%' || LOWER ( ( SELECT "Suche" FROM
    "Filter" WHERE "ID" = TRUE ) ) || '%'
010 )
```

Diese Abfrage erweitert die Abfrage *Suche_Form* so, dass in mehreren Elementen gleichzeitig mit dem Suchbegriff nach Übereinstimmungen gesucht werden kann. Die Textbestandteile von "Kategorie", "Medienart", "Ort" und "Verfasser" sind allerdings nicht Bestandteil der Tabelle "Medien". Daher muss hier innerhalb von "Ansicht_Suche" nach den Elementen gesucht werden.

Zuerst werden alle Felder der Tabelle "Medien" angezeigt (Zeile 1 und 2), so dass die Abfrage für die Dateneingabe im Formular geeignet ist.. Von "Medien" werden nur die Datensätze angezeigt, deren Wert für "ID" **IN** einer bestimmten Datenmenge enthalten ist (Zeile 3). Diese Datenmenge wird von Zeile 4 bis Zeile 10 definiert.

Die kleingeschriebene Variante von "Titel" soll an irgendeiner Stelle den Inhalt des Feldes "Suche" aus der Tabelle "Filter" enthalten. Enthält das Feld "Suche" keinen Eintrag, so soll stattdessen die keingeschriebne Variante von "Titel" mit sich selbst verglichen werden. Es werden dann also alle Datensätze angezeigt (Zeile 5). Entsprechend wird für alle anderen zu durchsuchenden Feldern in "Ansicht_Suche" (Zeile 4) vorgegangen. Dabei muss nur eines der Felder den Inhalt enthalten. Dies wird durch die Verknüpfung mit **OR** erreicht.

Damit der Wert für "ID" nicht mehrmals als Treffer auftaucht ("Ansicht_Suche" enthält durch die Verknüpfung zu "Verfasser" pro "Verfasser" jeweils einen Datensatz aus "Medien"), wird die zu vergleichende Menge durch **DISTINCT** auf einzelne, von einander unterschiedliche Werte eingeschränkt (Zeile 4).

Parametersuche_Medien

Datenquelle
Tabelle: *Medien, Ansicht_Suche*

Datenziel
Formular: *Parameter_Suche_Formular_Subformular*

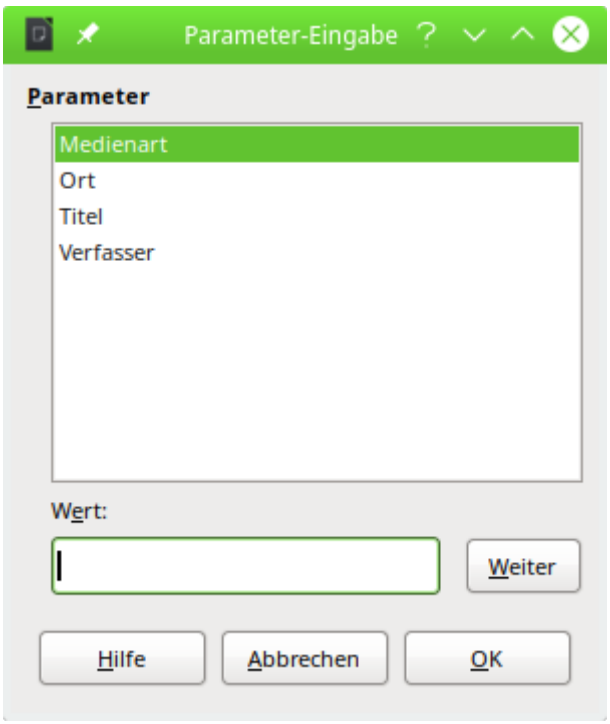
```
001 SELECT *
002 FROM "Medien"
003 WHERE "ID" IN (
004    SELECT "ID" FROM "Ansicht_Suche" WHERE
005       COALESCE( LOWER ( "Titel" ), '' ) LIKE '%' || COALESCE( LOWER
          ( :Titel ), '' ) || '%' AND
006       COALESCE( LOWER ( "Medienart" ), '' ) LIKE '%' || COALESCE( LOWER
          ( :Medienart ), '' ) || '%' AND
007       COALESCE( LOWER ( "Ort" ), '' ) LIKE '%' || COALESCE( LOWER
          ( :Ort ), '' ) || '%' AND
008       COALESCE( LOWER ( "Verfasser" ), '' ) LIKE '%' || COALESCE( LOWER
          ( :Verfasser ), '' ) || '%'
009 )
```

In der "Ansicht_Suche" befinden sich die Verfasser, Orte usw. in ausgeschriebener Form. Sie können also nach bestimmten Textelementen untersucht werden. Sämtliche Parameter werden durch **'%'** vor und hinter dem Parameter so abgefragt, dass die Textlemente an beliebiger Stelle in dem jeweiligen Feld stehen können. Außerdem wird sowohl der Feldinhalt als auch der Parameterinhalt in Kleinbuchstaben verglichen, so dass Groß- und Kleinschreibweise keine Rolle spielen.

Enthält ein Feld keinen Inhalt, so wir stattdessen ein leerer Text als Inhalt angenommen. So wird aus dem leeren Feld "Titel" gegebenenfalls ein leerer Text. Auch ein lererer Parameter wird als leerer Text angenommen. So wird aus **:Titel** ohne Inhalt ein leerer Text. Und leere Texte miteinander verglichen sind gleich. Würde hingegen einer der beiden Teile **NULL** sein, also tatsächlich ohne Inhalt, so wäre kein Vergleich möglich.

Alle Elemete aus "Medien", die dem Suchergebnis entsprechen, werden angezeigt. Durch die Anzeige aller Element, auch des Primärschlüssels, ist diese Abfrage für die weitere Dateneingabe geeignet.

Diese Parametersuche funktioniert prinzipiell auch ohne Formular. In dem Falle erscheint der folgende Dialog:



Nach Angabe der Parameter wird die Abfrage gestartet. Ein leerer Parameter wird dank der Abfrage so übersetzt, dass aus dem entsprechenden Feld alle Datensätze dem Inhalt entsprechen.

Filter und Suche kombiniert

Suche_Filter_Form_Subform_3Filter

Datenquelle	
Tabelle:	*Medien, Filter*
Ansicht:	*Ansicht_Suche*

Datenziel	
Formular:	*Suche_Filter_Formular_Subformular_ohne_Makros, Suche_Filter_hierarchisch_mit_Makros, Suche_Filter_hierarchisch_allround_Datensaetze_mit_Makros*
Abfrage:	*Listenfeld_Medienart_bedingt_Suche_Filter, Listenfeld_Kategorie_bedingt_Suche_Filter, Listenfeld_Verfasser_bedingt_Suche_Filter, Listenfeld_Medienart_bedingt_Suche_Filter_Datensaetze, Listenfeld_Kategorie_bedingt_Suche_Filter_Datensaetze*

```
001 SELECT *
002 FROM "Medien"
003 WHERE "ID" IN (
004 SELECT DISTINCT "ID" FROM "Ansicht_Suche"
005 WHERE "VerfasserID" = COALESCE( ( SELECT "Filter_Verfasser" FROM "Filter"
    WHERE "ID" = TRUE ), "VerfasserID" )
006 AND "MedienartID" = COALESCE( ( SELECT "Filter_Medienart" FROM "Filter"
    WHERE "ID" = TRUE ), "MedienartID" )
007 AND "KategorieID" = COALESCE( ( SELECT "Filter_Kategorie" FROM "Filter"
    WHERE "ID" = TRUE ), "KategorieID" )
008 AND (
```

```
009 LOWER ( "Titel" ) LIKE COALESCE( '%' || LOWER ( ( SELECT "Suche" FROM
    "Filter" WHERE "ID" = TRUE ) ) || '%', LOWER ( "Titel" ) )
010 OR LOWER ( "Kategorie" ) LIKE '%' || LOWER ( ( SELECT "Suche" FROM
    "Filter" WHERE "ID" = TRUE ) ) || '%'
011 OR LOWER ( "Medienart" ) LIKE '%' || LOWER ( ( SELECT "Suche" FROM
    "Filter" WHERE "ID" = TRUE ) ) || '%'
012 OR LOWER ( "Ort" ) LIKE '%' || LOWER ( ( SELECT "Suche" FROM "Filter"
    WHERE "ID" = TRUE ) ) || '%'
013 OR LOWER ( "Verfasser" ) LIKE '%' || LOWER ( ( SELECT "Suche" FROM
    "Filter" WHERE "ID" = TRUE ) ) || '%'
014 )
015 )
```

Diese Abfrage ist eine Kombination der Abfragen *Filter_Form_Subform_3Filter* und *Suche_Form_Subform*. Aus der Tabelle "Medien" werden alle Datensätze angezeigt, für die der Primärschlüssel "ID" **IN** der von Zeile 4 bis 14 definierten Ergebnismenge enthalten ist. Untersucht wird hier wieder "Ansicht_Suche", da nur in dieser Ansicht neben dem Feld "Titel" auch die anderen Felder nicht nur durch einen Fremdschlüssel enthalten sind. Außerdem ist "Ansicht_Suche" auch erforderlich, um den Fremdschlüssel zum "Verfasser" zu ermitteln, da "Verfasser" und "Medien" über die Tabelle "rel_Medien_Verfasser" in einer n:m-Beziehung stehen.

Von Zeile 5 bis 7 wird die Filterung für die Fremdschlüsselfelder vorgenommen. Von Zeile 9 bis 13 wird die Suche durch die angegebenen Felder realisiert. Beide Bedingungen sind durch ein **AND** in Zeile 8 verknüpft, wobei eine Klammerung um die zu durchsuchenden Felder (Zeile 8 und Zeile 14) bewirkt, dass die Verknüpfungen mit **OR** innerhalb der Suche nicht zu unerwarteten Ergebnissen führen. So könnte ohne die Klammerung z. B. ein Datensatz angezeigt werden, wenn zwar der Suchbegriff allein in "Verfasser" vorkommt, aber gleichzeitig ein Verfasser über das Filterfeld ausgesucht wurde, der dem Suchbegriff gar nicht entspricht.

Diese Abfrage musste für FIREBIRD umbenannt werden, da dort eine Grenze von 30 Zeichen für Bezeichnungen von Feldern und Tabellen vorgesehen ist. Die Benennung der Abfrage selbst macht dabei nichts aus. Aber andere Abfragen können nicht auf diese Abfrage zugreifen. Die Abfrage hat unter Firebird deswegen die Bezeichnung «Suche_Filter_Form_Subform_3F» Alle sich darauf beziehenden Abfragen und Formulare mussten entsprechend angepasst werden.

Filter_Such_Ansicht

Datenquelle	
Tabelle:	*Filter*
Ansicht:	*Ansicht_Suche*

Datenziel	
Abfrage:	*Listenfeld_Verfasser_bedingt_Suche_Filter_Datensaetze*

```
001 SELECT *
002 FROM "Ansicht_Suche"
003 WHERE "ID" IN (
004 SELECT DISTINCT "ID" FROM "Ansicht_Suche"
005 WHERE "VerfasserID" = COALESCE( ( SELECT "Filter_Verfasser" FROM "Filter"
    WHERE "ID" = TRUE ), "VerfasserID" )
006 AND "MedienartID" = COALESCE( ( SELECT "Filter_Medienart" FROM "Filter"
    WHERE "ID" = TRUE ), "MedienartID" )
007 AND "KategorieID" = COALESCE( ( SELECT "Filter_Kategorie" FROM "Filter"
    WHERE "ID" = TRUE ), "KategorieID" )
008 AND (
009 LOWER ( "Titel" ) LIKE COALESCE( '%' || LOWER ( ( SELECT "Suche" FROM
    "Filter" WHERE "ID" = TRUE ) ) || '%', LOWER ( "Titel" ) )
```

```
010 OR LOWER ( "Kategorie" ) LIKE '%' || LOWER ( ( SELECT "Suche" FROM
    "Filter" WHERE "ID" = TRUE ) ) || '%'
011 OR LOWER ( "Medienart" ) LIKE '%' || LOWER ( ( SELECT "Suche" FROM
    "Filter" WHERE "ID" = TRUE ) ) || '%'
012 OR LOWER ( "Ort" ) LIKE '%' || LOWER ( ( SELECT "Suche" FROM "Filter"
    WHERE "ID" = TRUE ) ) || '%'
013 OR LOWER ( "Verfasser" ) LIKE '%' || LOWER ( ( SELECT "Suche" FROM
    "Filter" WHERE "ID" = TRUE ) ) || '%'
014 )
015 )
```

Diese Abfrage ist auf den ersten Blick identisch mit der Abfrage *Suche_Filter_Form_Subform_3Filter*. Ein kleiner Unterschied besteht allerdings in der Datenquelle, aus der alle Datensätze angezeigt werden sollen. In dieser Abfrage handelt es sich nämlich um alle Datensätze aus "Ansicht_Suche", nicht aus "Medien". Der Grund dafür ist, dass diese Abfrage alle Datensätze zu dem Verfasser anzeigen soll, damit diese als Grundlagen für ein Listenfeld genommen werden können, das der Suchbedingung entspricht.

Listenfeldabfragen

Prinzipiell könnten viele dieser Abfragen auch gut direkt bei den Listenfeldern als SQL-Code eingetragen werden. Über die Abfragen wird lediglich einfacher der SQL-Code erreichbar und für den Nutzer einsehbar.

Listenfeld_Kategorie_ohne_Eintrag

Datenquelle
Tabelle: *Kategorien, Medien*

Datenziel
Formular: *Filter_Formular_Nullwerte, Filter_Formular_Subformular_3Filter, Filter_Formular_direkt_Nullwerte, Suche_Filter_Formular_Subformular_ohne_Makros*

```
001 SELECT "Kat", "ID"
002 FROM (
003 SELECT '1' AS "C", 'ohne Eintrag' AS "Kat", 0 AS "ID" FROM "Kategorien"
004 UNION
005 SELECT "Kategorien"."System",
006    LEFT( "Kategorien"."System" || '       ', 7 )
007    || ' - ' ||
008    LEFT("Kategorien"."Kategorie",50) ||
009    CASE WHEN CHAR_LENGTH("Kategorien"."Kategorie") > 50 THEN '…' ELSE ''
    END
010    AS "Kat",
011    "Kategorien"."ID"
012    FROM "Kategorien", "Medien" WHERE "Kategorien"."ID" = "Medien"."katID"
013 UNION
014 SELECT '0' AS "C", '' AS "Kat", NULL AS "ID" FROM "Kategorien")
```

Das Listenfeld soll auch die Möglichkeit bieten die Datensätze heraus zu suchen, die keinen Eintrag in dem entsprechenden Feld haben. Diesen Feldern ist über eine Abfrage der Wert '0' zugewiesen worden. Da die Abfrage mit dem Befehl **UNION** arbeitet ist sie nur mit direktem SQL ausführbar.

In Zeile 3 wird die erste Unterabfrage gestellt. Das erste Feld dieser Abfrage dient nur zur Sortierung innerhalb des **UNION**-Verbundes. Als zweites Feld wird lediglich 'ohne Eintrag' geschrieben. Als drittes Feld erscheint dann die '0', die über das Listenfeld weitergegeben wird.

In Zeile 5 beginnt die zweite Unterabfrage, die an die vorherige Abfrage durch **UNION** angehängt wird. Als Ordnungsmerkmal wird hier der Eintrag aus "Kategorien"."System" genutzt. Die Kategorien werden für das Listenfeld so dargestellt, dass der Systemeintrag komplett erscheinen soll, Leerstellen aber gegebenenfalls aufgefüllt werden. Der längste Systematikeintrag beträgt hier 7 Zeichen, so dass in Zeile 6 auf insgesamt 7 Zeichen von links aus ausgelesen wird. Zeile 7 enthält lediglich das Verbindungszeichen zwischen dem Systematikeintrag und der ausführlicheren Kategorienbeschreibung. In Zeile 8 wird die Kategorienbeschreibung auf 50 Zeichen begrenzt, da sonst das Listenfeld viel zu breit wird. Beträgt die Länge des Eintrags mehr als 50 Zeichen, so wird in Zeile 9 noch ein '...' angehängt.
Die Unterabfrage schließt mit dem Primärschlüsseleintrag aus der Tabelle "Kategorien" ab.

In Zeile 14 wird noch eine Unterabfrage angehängt, die notwendig ist, damit das Listenfeld anschließend auch eine leere Zeile darstellt. Diese Zeile ist notwendig, damit die Filterung auch wieder ausgeschaltet werden kann. Sie wird standardmäßig bei Listenfeldern mit eingeblendet, sofern der SQL-Code nicht direkt ausgeführt werden muss.
Sortierungsmerkmal ist hier die '0'. Damit erscheint der Eintag an erster Stelle. In dem Listenfeld soll nicht angezeigt werden - ein leerer String. Von dem Listenfeld soll der Wert **NULL** weitergegeben werden.
Wird diese Unterabfrage an erster Position gestellt, so versteht die HSQLDB die gesamte Kombination als eine Kombination von Werten mit **NULL**. Die gesamten Einträge im Feld "ID" bleiben dann leer.

Auf diese Unterabfragen wird nun durch eine äußere Abfrage für das Listenfeld zugegriffen, die lediglich die Felder "Kat" und "ID" anzeigt (Zeile 1).

Listenfeld_Kategorie_bedingt

Datenquelle	
Tabelle:	*Kategorien*
Abfrage:	*Filter_Form_Subform_3Filter*

Datenziel	
Formular:	*Filter_Formular_Subformular_bedingende_Filter, Filter_Formular_Subformular_bedingende_Filter_allround*

```
001 SELECT "System" || ' - ' || "Kategorie" AS "Kat", "ID"
002 FROM "Kategorien"
003 WHERE "ID" IN
004 ( SELECT "katID" FROM "Filter_Form_Subform_3Filter" )
005 ORDER BY "Kat" ASC
```

In Zeile 1 werden für die Anzeige die Inhalte aus "System" und "Kategorie" zusammengefasst. Diese Zusammenfassung erhält den Aliasbegriff "Kat", auf den sich dann in Zeile 5 die Sortierung beziehen kann.

Die Anzeige der Inhalte wird durch die vorhandenen Einträge für "katID" in der Abfrage "Filter_Form_Subform_3Filter" eingeschränkt (Zeile 4). Dadurch werden nur noch die Daten angezeigt, die in der zu filternden Menge auch noch zur Verfügung stehen. Das vermeidet, dass bei einer Auswahl über das Listenfeld plötzlich gar keine Ergebnisse mehr in der Ergebnismenge vorkommen.

Listenfelder mit einem Inhalt wie diesem müssen nach jeder Eingabe in einem anderen Listenfeld aufgefrischt werden. Dies kann über die Tastatur erfolgen, ist aber einfacher über Makros zu bedienen.

Listenfeld_Kategorie_bedingt_Datensaetze

Datenquelle	
Tabelle:	*Kategorien*
Abfrage:	*Filter_Form_Subform_3Filter*

Datenziel	
Formular:	*Filter_Formular_Subformular_bedingende_Filter_Datensaetze*

```
001 SELECT "System" || ' - ' || LEFT("Kategorie",40) ||
    CASE WHEN CHAR_LENGTH("Kategorie") > 40 THEN '…' ELSE '' END || ' → ' || (
002 SELECT COUNT( "katID" ) FROM "Filter_Form_Subform_3Filter" WHERE "katID" =
    "a"."ID"
003 ) || ' Treffer' AS "Kat", "ID"
004 FROM "Kategorien" AS "a"
005 WHERE "ID" IN
006 ( SELECT "katID" FROM "Filter_Form_Subform_3Filter" )
007 ORDER BY "Kat" ASC
```

Neben der vorgefilterten Anzeige der Kategorien (siehe *Listenfeld_Kategorie_bedingt*) soll hier noch angezeigt werden, wie viele Ergebnisse bei der Auswahl der entsprechend Kategorie zu erwarten sind. Dies wird durch eine korrelierende Unterabfrage in Zeile `2` bewerkstelligt. Durch diese Unterabfrage werden alle Vorkommen des Feldes "katID" in "Filter_Form_Subform_3Filter" zusammengezählt, die dem Wert der "ID" des aktuellen Datensatzes entsprechen. Auf den aktuellen Datensatz bezieht sich die Abfrage dadurch, dass der Tabelle "Kategorien" in Zeile `4` ein Alias "a" zugewiesen wird und in Zeile `2` dann genau dieses "a"."ID" auftaucht.

Die Anzeige in dem Listenfeld wird für "Kategorie" auf 40 Buchstaben begrenzt, da sonst beim Aufklappen ein sehr breites Feld entsteht. Enthält "Kategorie" mehr als 40 Buchstaben, so soll ein **'…'** angehängt werden (Zeile `1`).

Dieses Listenfeld zeigt nur die Kategorien an, zu denen noch Filterergebnisse möglich sind. Felder ohne Treffer werden durch die Vorfilterung in Zeile `6` unterbunden.

Listenfeld_Kategorie_bedingt_Suche_Filter

Datenquelle	
Tabelle:	*Kategorien*
Abfrage:	*Suche_Filter_Form_Subform_3Filter*

Datenziel	
Formular:	*Suche_Filter_hierarchisch_mit_Makros*

```
001 SELECT "System" || ' - ' || "Kategorie" AS "Kat", "ID"
002 FROM "Kategorien"
003 WHERE "ID" IN
004 ( SELECT "katID" FROM "Suche_Filter_Form_Subform_3Filter" )
005 ORDER BY "Kat" ASC
```

Diese Abfrage entspricht nahezu komplett der Abfrage *Listenfeld_Kategorie_bedingt*. Lediglich die Abfrage, durch die die anzuzeigenden Datensätze eingeschränkt werden, ändert sich hier auf "Suche_Filter_Form_Subform_3Filter" (Zeile `4`).

Listenfeld_Kategorie_bedingt_Suche_Filter_Datensaetze

Datenquelle	
Tabelle:	*Kategorien*
Abfrage:	*Suche_Filter_Form_Subform_3Filter*

Datenziel	
Formular:	*Suche_Filter_hierarchisch_allround_Datensaetze_mit_Makros*

```
001 SELECT "System" || ' - ' || "LEFT("Kategorie",40) ||
    CASE WHEN CHAR_LENGTH("Kategorie") > 40 THEN '…' ELSE '' END || ' → ' || (
002 SELECT COUNT( "katID" ) FROM "Suche_Filter_Form_Subform_3Filter" WHERE
    "katID" = "a"."ID"
003 ) || ' Treffer' AS "Kat", "ID"
004 FROM "Kategorien" AS "a"
005 WHERE "ID" IN
006 ( SELECT "katID" FROM "Suche_Filter_Form_Subform_3Filter" )
007 ORDER BY "Kat" ASC
```

Diese Abfrage entspricht nahezu komplett der Abfrage *Listenfeld_Kategorie_bedingt_Datensaetze*. Lediglich die Abfrage, durch die die anzuzeigenden Datensätze eingeschränkt werden, ändert sich hier auf "Suche_Filter_Form_Subform_3Filter" (Zeile 6). Natürlich wird auch die Trefferzahl nach dieser Abfrage ermittelt (Zeile 2).

Listenfeld_Medienart_bedingt

Datenquelle	
Tabelle:	*Medienart*
Abfrage:	*Filter_Form_Subform_3Filter*

Datenziel	
Formular:	*Filter_Formular_Subformular_bedingende_Filter, Filter_Formular_Subformular_bedingende_Filter_allround*

```
001 SELECT "Medienart", "ID"
002 FROM "Medienart"
003 WHERE "ID" IN
004 ( SELECT "artID" FROM "Filter_Form_Subform_3Filter" )
005 ORDER BY "Medienart" ASC
```

Alle Datensätze aus "Medienart" werden angezeigt, zu deren Primärschlüssel "ID" es in der Datenmenge von "Filter_Form_Subform_3Filter" einen entsprechenden Eintrag "artID" gibt.

Listenfeld_Medienart_bedingt_Datensaetze

Datenquelle	
Tabelle:	*Medienart*
Abfrage:	*Filter_Form_Subform_3Filter*

Datenziel	
Formular:	*Filter_Formular_Subformular_bedingende_Filter_Datensaetze*

```
001 SELECT "Medienart" || ' → ' || (
002 SELECT COUNT( "artID" ) FROM "Filter_Form_Subform_3Filter" WHERE "artID" =
    "a"."ID"
```

```
003 ) || ' Treffer' AS "Med", "ID"
004 FROM "Medienart" AS "a"
005 WHERE "ID" IN ( SELECT "artID" FROM "Filter_Form_Subform_3Filter" )
006 ORDER BY "Med" ASC
```

Diese Abfrage entspricht in den Grundzügen *Listenfeld_Kategorie_bedingt_Datensaetze*. Allerdings enthält das Feld "Medienart" nicht so lange Texte, so dass der Inhalt hier nicht begrenzt. wird.

Listenfeld_Medienart_bedingt_Suche_Filter

Datenquelle
Tabelle: *Medienart*
Abfrage: *Suche_Filter_Form_Subform_3Filter*

Datenziel
Formular: *Suche_Filter_hierarchisch_mit_Makros*

```
001 SELECT "Medienart", "ID"
002 FROM "Medienart"
003 WHERE "ID" IN
004 ( SELECT "artID" FROM "Suche_Filter_Form_Subform_3Filter" )
005 ORDER BY "Medienart" ASC
```

Im Unterschied zu *Listenfeld_Medienart_bedingt* bezieht sich diese Abfrage auf die Abfrage "Suche_Filter_Form_Subform_3Filter". Ansonsten ist die Funktion gleich.

Listenfeld_Medienart_bedingt_Suche_Filter_Datensaetze

Datenquelle
Tabelle: *Medienart*
Abfrage: *Suche_Filter_Form_Subform_3Filter*

Datenziel
Formular: *Suche_Filter_hierarchisch_allround_Datensaetze_mit_Makros*

```
001 SELECT "Medienart" || ' → ' || (
002 SELECT COUNT( "artID" ) FROM "Suche_Filter_Form_Subform_3Filter" WHERE
    "artID" = "a"."ID"
003 ) || ' Treffer' AS "Med", "ID"
004 FROM "Medienart" AS "a" WHERE "ID" IN
005 ( SELECT "artID" FROM "Suche_Filter_Form_Subform_3Filter" )
006 ORDER BY "Med" ASC
```

Im Unterschied zu *Listenfeld_Medienart_bedingt_Datensaetze* bezieht sich diese Abfrage auf die Abfrage "Suche_Filter_Form_Subform_3Filter". Ansonsten ist die Funktion gleich.

Listenfeld_Verfasser_bedingt

Datenquelle
Tabelle: *rel_Medien_Verfasser, Verfasser*
Abfrage: *Filter_Form_Subform_3Filter*

Datenziel
Formular: *Filter_Formular_Subformular_bedingende_Filter, Filter_Formular_Subformular_bedingende_Filter_allround*

```
001 SELECT DISTINCT "Verfasser"."Verfasser", "Verfasser"."ID"
002 FROM "rel_Medien_Verfasser", "Verfasser"
003 WHERE "rel_Medien_Verfasser"."vefID" = "Verfasser"."ID"
004 AND "rel_Medien_Verfasser"."VerfSort" < 99
005 AND "rel_Medien_Verfasser"."medID" IN
006 ( SELECT "ID" FROM "Filter_Form_Subform_3Filter" )
007 ORDER BY "Verfasser"."Verfasser" ASC
```

Bei den Listenfeldabfragen für den Verasser muss immer wieder ein Sinderweg genutzt werden. Schließlich steht die Tabelle "Verfasser" nicht in einer direkten Beziehung zur Tabelle "Medien". Deshalb ist hier die Beziehung auf 2 Tabellen notwendig (Zeile 2).

Aus der Tabelle "rel_Medien_Verfasser" werden nur die Fremdschlüssel "vefID" übernommen, bei denen "VerfSort" kleiner als '99' ist. Diese Werte sind Formulierungen wie '(Hrsg.)', '(u.a.)' oder anderen vorbehalten. Die Eingabe der entsprechenden Sortierungen ist in dieser Datenbank nicht vorgesehen. Die Datenbank dient schließlich in der Hauptsache als Quelle für die Demonstration von Filter- und Suchfunktionen.

Die Verfasser, die angezeigt werden, sollen auch in den Medien verfügbar sein, die in der Abfrage "Filter_Form_Subform_3Filter" (Zeile 6) enthalten sind.

Listenfeld_Verfasser_bedingt_Datensaetze_langsam

Datenquelle
Ansicht: *Ansicht_Suche*
Abfrage: *Filter_Form_Subform_3Filter*

Datenziel
keins, da die Abfrage zu langsam läuft

```
001 SELECT DISTINCT "a"."Verfasser" || ' → ' || (
002 SELECT COUNT( "vefID" ) FROM "Ansicht_Suche" WHERE "vefID" = "a"."vefID"
    AND "ID" IN ( SELECT "ID" FROM "Filter_Form_Subform_3Filter" )
003 ) || ' Treffer' AS "Ver", "a"."vefID"
004 FROM "Ansicht_Suche" AS "a"
005 WHERE "a"."VerfSort" < 99 AND "a"."ID" IN ( SELECT "ID" FROM
    "Filter_Form_Subform_3Filter" )
006 ORDER BY "Ver" ASC
```

Diese Abfrage liefert die gleichen Ergebnisse wie *Listenfeld_Verfasser_bedingt_Datensaetze_optimiert*, hat sich aber als zu langsam erwiesen. Dies liegt daran, dass sich die Abfrage nicht nur auf eine Datenquelle mit sehr vielen Datensätzen bezieht ("Ansicht_Suche"), sondern auch noch über die Abfrage "Filter_Form_Subform_3Filter" noch einmal den Bezug auf die große Tabelle "Medien" nimmt. Auch solche misslungenen Ansätze sollten dokumentiert werden. Schließlich funktioniert die Abfrage bei wenigen Datensätzen erst einmal problemlos. Das Erwachen kommt dann, wenn die Datenbank entsprechend gefüllt ist.

Unter FIREBIRD bringt diese Abfrage LO insgesamt zum Absturz. Sie wurde daher direkt entfernt.

Listenfeld_Verfasser_bedingt_Datensaetze_optimiert

Datenquelle
Abfrage: *Filter_Ansicht*

Datenziel
Formular: *Filter_Formular_Subformular_bedingende_Filter_Datensaetze*

```
001 SELECT DISTINCT "Verfasser" || ' → ' || (
002 SELECT COUNT( "vefID" ) FROM "Filter_Ansicht" WHERE "vefID" = "a"."vefID"
003 ) || ' Treffer' AS "Ver", "vefID"
004 FROM "Filter_Ansicht" AS "a"
005 WHERE "VerfSort" < 99
006 ORDER BY "Ver" ASC
```

Im Gegensatz zur Abfrage *Listenfeld_Verfasser_bedingt_Datensaetze_langsam* bezieht sich diese Abfrage über "Filter_Ansicht" nur auf eine große Datenquelle, nämlich *Ansicht_Suche*. Dadurch erhöht sich die Geschwindigkeit deutlich.

Die "Filter_Ansicht" selektiert die Daten bereits so weit vor, dass nur die passenden Datensätze zur Verfügung stehen, die bei der Auswahl über das entsprechende Filter-Listenfeld auch noch zu Treffern führen. In Zeile 2 werden diese Treffer über eine korrelierende Unterabfrage ermittelt. Diese Unterabfrage bezieht sich über "a"."vefID" auf die äußere Abfrage, weil der Datenquelle "Filter_Ansicht" das Alias "a" zugewiesen wurde.

Wie in *Listenfeld_Verfasser_bedingt* werden auch hier alle Datensätze mit "VerfSort" '99' von der Anzeige ausgeschlossen (Zeile 5).

Listenfeld_Verfasser_bedingt_Suche_Filter

Datenquelle	
Tabelle:	*rel_Medien_Verfasser, Verfasser*
Abfrage:	*Suche_Filter_Form_Subform_3Filter*

Datenziel
Formular: *Suche_Filter_hierarchisch_mit_Makros*

```
001 SELECT DISTINCT "Verfasser"."Verfasser", "Verfasser"."ID"
002 FROM "rel_Medien_Verfasser", "Verfasser"
003 WHERE "rel_Medien_Verfasser"."vefID" = "Verfasser"."ID"
004 AND "rel_Medien_Verfasser"."VerfSort" < 99
005 AND "rel_Medien_Verfasser"."medID" IN
006 ( SELECT "ID" FROM "Suche_Filter_Form_Subform_3Filter" )
007 ORDER BY "Verfasser"."Verfasser" ASC
```

Im Unterschied zu *Listenfeld_Verfasser_bedingt* bezieht sich diese Abfrage auf die Abfrage "Suche_Filter_Form_Subform_3Filter". Ansonsten ist die Funktion gleich.

Listenfeld_Verfasser_bedingt_Suche_Filter_Datensaetze

Datenquelle	
Abfrage:	*Filter_Such_Ansicht*

Datenziel
Formular: *Suche_Filter_hierarchisch_allround_Datensaetze_mit_Makros*

```
001 SELECT DISTINCT "Verfasser" || ' → ' ||
002 ( SELECT COUNT( "vefID" ) FROM "Filter_Such_Ansicht" WHERE "vefID" =
    "a"."vefID" )
003 || ' Treffer' AS "Ver", "vefID"
004 FROM "Filter_Such_Ansicht" AS "a"
```

```
005 WHERE "VerfSort" < 99
006 ORDER BY "Ver" ASC
```

Im Unterschied zu *Listenfeld_Verfasser_bedingt_Datensaetze_optimiert* bezieht sich diese Abfrage auf die Abfrage "Filter_Such_Ansicht". Ansonsten ist die Funktion gleich.

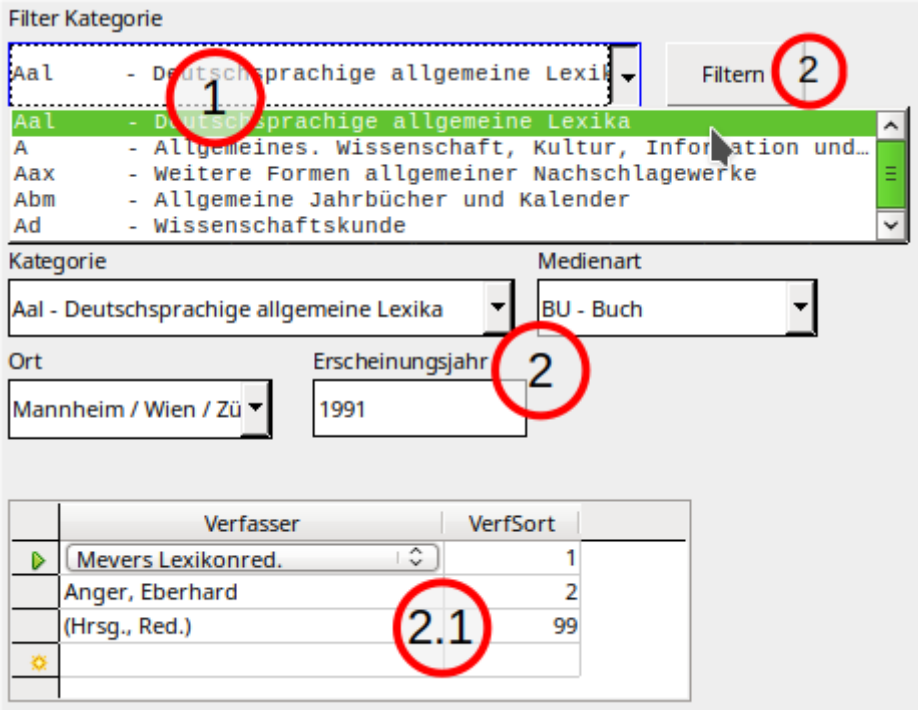
Formulare

Zur besseren Übersicht sind die Formulare teilweise in Unterordner verschoben worden. Die entsprechenden Unterordner sind hier in den Kapiteln abgebildet.

Filter_ohne_Makros

Die hierin enthaltenen Formulare arbeiten komplett ohne Makros. Sie Filtern die Daten über Listenfelder nach den entsprechenden Fremdschlüsselwerten in der Tabelle "Medien".

Filter_Formular



1	Filter (Tabelle: *Filter*)		
2	MainForm (Abfrage: *Filter_Form*)	**2.1**	SubForm (Tabelle: *rel_Medien_Verfasser*)

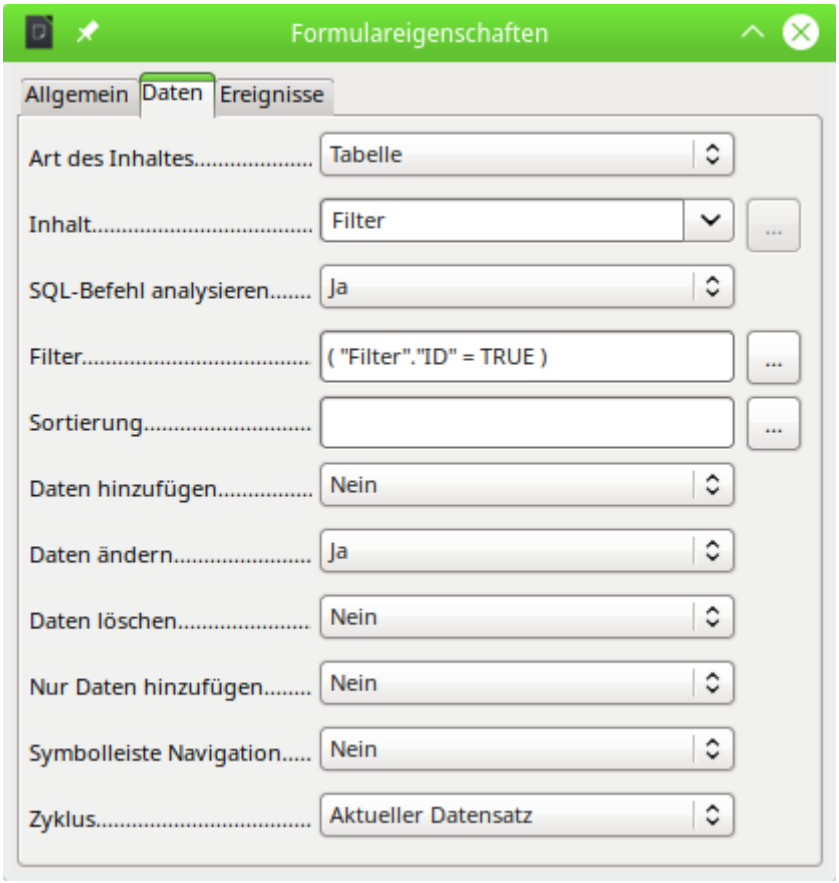
Listenfeld «Filter Kategorie» (Tabelle: *Kategorien, Medien*)

```
001 SELECT DISTINCT LEFT( "Kategorien"."System" || '       ', 7 ) || ' - ' ||
        LEFT("Kategorien"."Kategorie", 50)|| CASE WHEN
        CHAR_LENGTH("Kategorien"."Kategorie") > 50 THEN '…' ELSE '' END
        AS "Kat",
002 "Kategorien"."ID"
003 FROM "Kategorien", "Medien"
004 WHERE "Kategorien"."ID" = "Medien"."katID"
005 ORDER BY "Kat" ASC
```

Schriftart für dieses Listenfeld: Liberation Mono, Standard, 8. Durch die Schriftart mit gleichmäßiger Breite für jeden Buchstaben lässt sich ein Spalteneffekt in dem Listenfeld erzielen.

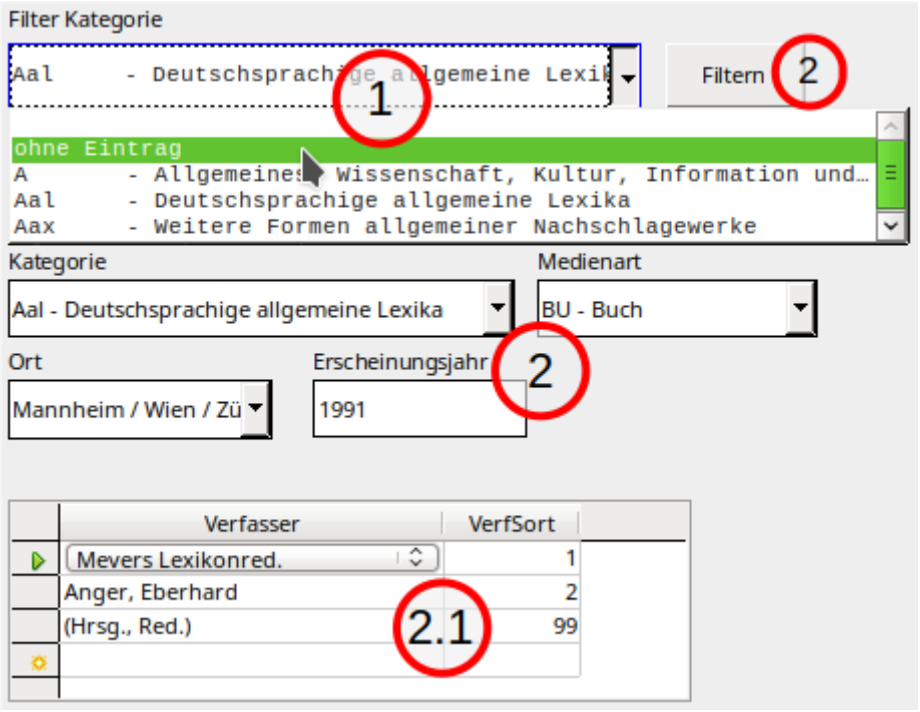
Der Button Filtern ist mit der Aktion «Formular aktualisieren» verbunden. Durch die Betätigung des Buttons wird das Formular 1 abgespeichert und das Formular 2 aktualisiert.

Die Filtertabelle soll jeweils nur mit neuen Daten in dem aktuellen Datensatz beschrieben werden. Die Formulareigenschaften sind deshalb wie im folgenden Screenshot eingestellt:



Formulareigenschaften des Formulars «Filter»

Filter_Formular_Nullwerte

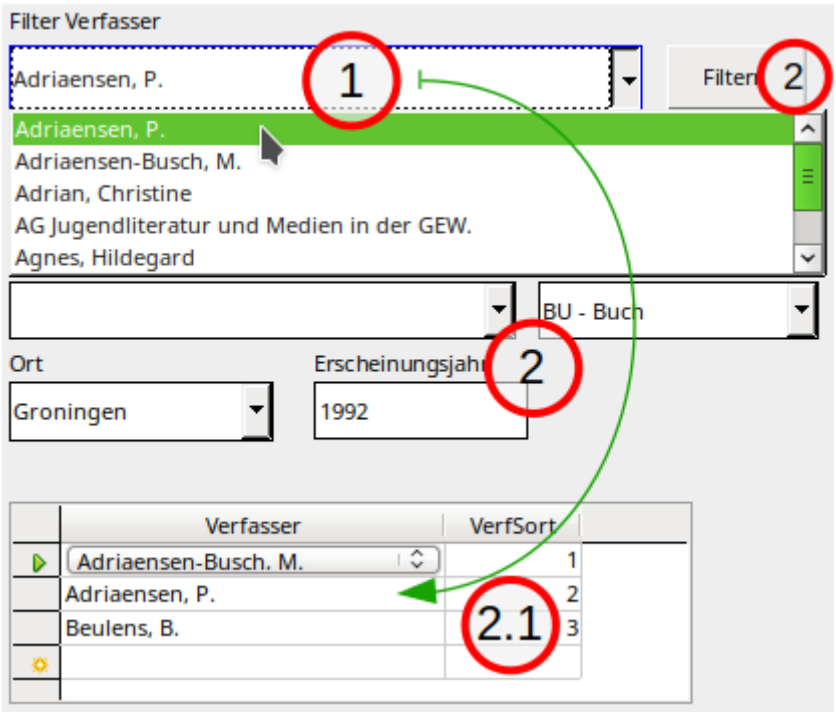


1	Filter (Tabelle: *Filter*)		
2	MainForm (Abfrage: *Filter_Form*)	**2.1**	SubForm (Tabelle: *rel_Medien_Verfasser*)

Listenfeld «Filter Kategorie» (Abfrage: *Listenfeld_Kategorie_ohne_Eintrag*)

Dieses Formular unterscheidet sich nur in Bezug auf das Listenfeld «Filter Kategorie» vom *Filter_Formular*.

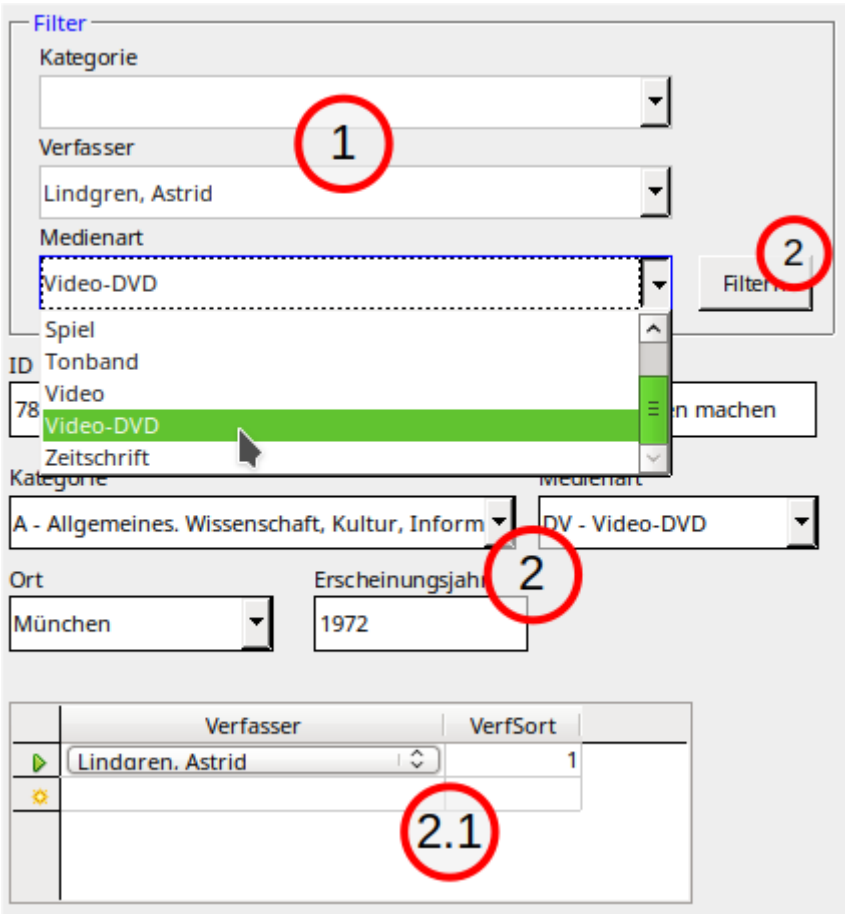
Filter_Formular_Subformular



1	Filter (Tabelle: *Filter*)		
2	MainForm (Abfrage: *Filter_Form_Subform*)	**2.1**	SubForm (Tabelle: *rel_Medien_Verfasser*)

Mit Hilfe der Abfrage in Formular 2 ist es möglich, auch Angaben für das Feld «Verfasser» in dem Unterformular 2.1 zu suchen.

Filter_Formular_Subformular_3Filter

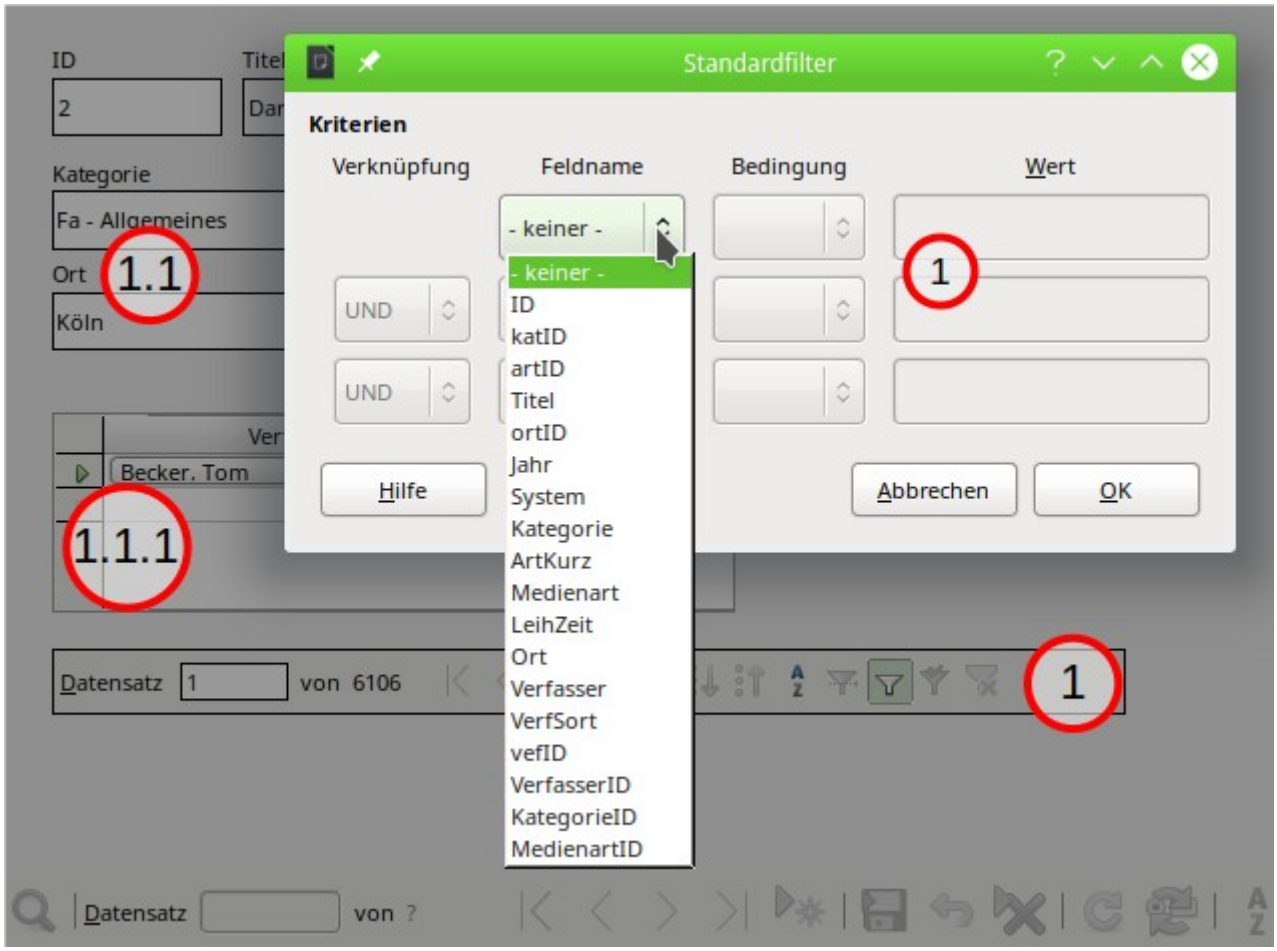


1	Filter (Tabelle: *Filter*)		
2	MainForm (Abfrage: *Filter_Form_Subform_3Filter*)	**2.1**	SubForm (Tabelle: *rel_Medien_Verfasser*)

Listenfeld «Filter Kategorie» (Abfrage: *Listenfeld_Kategorie_ohne_Eintrag*)

Auch die Filterung über mehrere Filterfelder ist ohne Makros möglich. Nachteil ist allerdings, dass andere Filter nicht an den bereits ausgewählten Filter angepasst werden. Das kann schnell dazu führen, dass gar kein Datensatz mehr anzeigbar ist. So existieren beispielsweise zu der oben ausgewählten Verfasserin nur Bücher und eine Video-DVD in dem Bestand. Die zur Auswahl stehenden anderen Medienarten führen zu einem leeren Suchergebnis.

Standardfilter_Formular_Subformular



1	Filter (Ansicht: *Ansicht_Suche*)	**1.1**	SubForm (Tabelle: *Medien*)	**1.1.1**	SubSubForm (Tabelle: *rel_Medien_Verfasser*)

Über den Standardfilter wird «Ansicht_Suche» im Formular 1 gefiltert. Das Ergebnis dieser Filterung wird an das Formular 1.1 weiter gegeben.

An der unten eingeblendeten Symbolleiste ist zu sehen, dass diese Filterung für die GUI nicht so gut handhabbar ist. Die gesamte Symbolleiste ist ausgegraut. Dieses Formular kann nicht zur Eingabe neuer Daten genutzt werden. Bestehende Daten lassen sich aber sehr wohl ändern.

Das Formular hat außerdem den Nachteil, dass die zugrundeliegende Ansicht für jeden Verfasser eines Mediums einen neuen Datensatz erstellt. Obwohl die beigefügte Tabelle "Medien" nur 4992 Datensätze enthält werden vor der Filterung 6106 Datensätze angegeben. Es gibt eben Medien mit mehreren Verfassern, so dass auch bei der Filterung ein Medium mehrmals vorkommen kann.

Standardfilter_Formular_Subformular_Datensatzkorrektur

1	Filter (Ansicht: *Ansicht_Standardfilter*)	**1.1**	SubForm (Tabelle: *Medien*)	**1.1.1**	SubSubForm (Tabelle: *rel_Medien_Verfasser*)

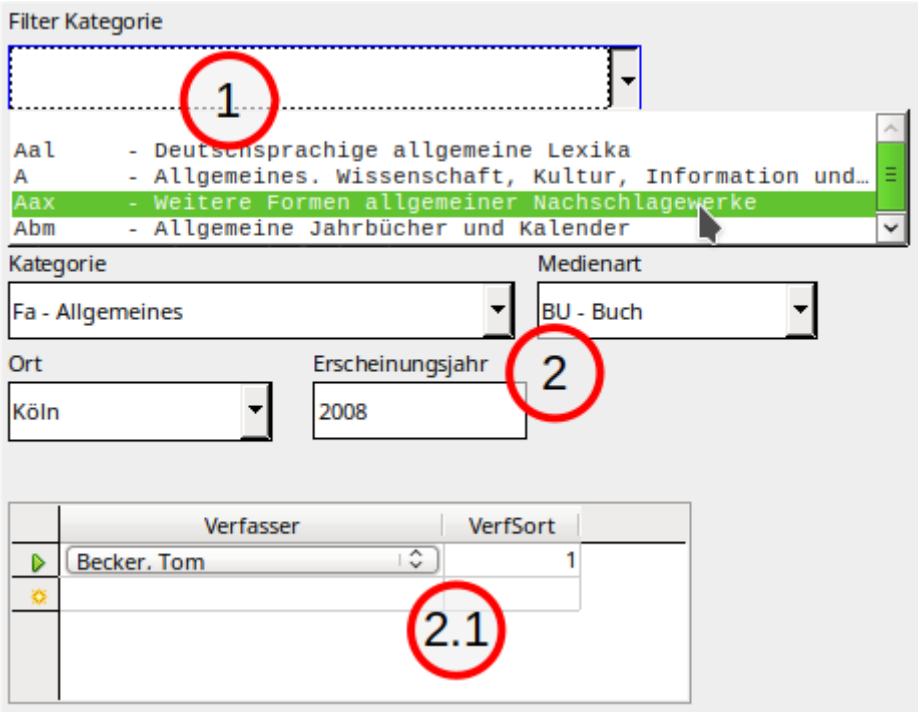
Dieses Formular startet bei der internen Hsqldb deutlich langsamer als das Formular *Standardfilter_Formular_Subformular*. Dafür zeigt die Ansicht aber die korrekte Anzahl der Datensätze an. Dies liegt daran, dass die Verfasser, die zu einem Datensatz in der Tabelle "Medien" gehören, in einem Feld zusammengefasst wurden.

Auch bei diesem Formular gilt allerdings: Eine Datenänderung ist möglich, eine Neueingabe von Daten nicht.

Filter_mit_Makros

In diesen Formularen läuft die Filterung direkt nach der Auswahl in den Listenfeldern ab. Die Betätigung eines Buttons ist nicht mehr notwendig.

Filter_Formular



1	Filter (Tabelle: *Filter*)		
2	MainForm (Abfrage: *Filter_Form*)	**2.1**	SubForm (Tabelle: *rel_Medien_Verfasser*)

Makros in Feldeigenschaften		
1	Feld «Filter Kategorie» (Modifiziert)	Module1.*Mauszeiger*

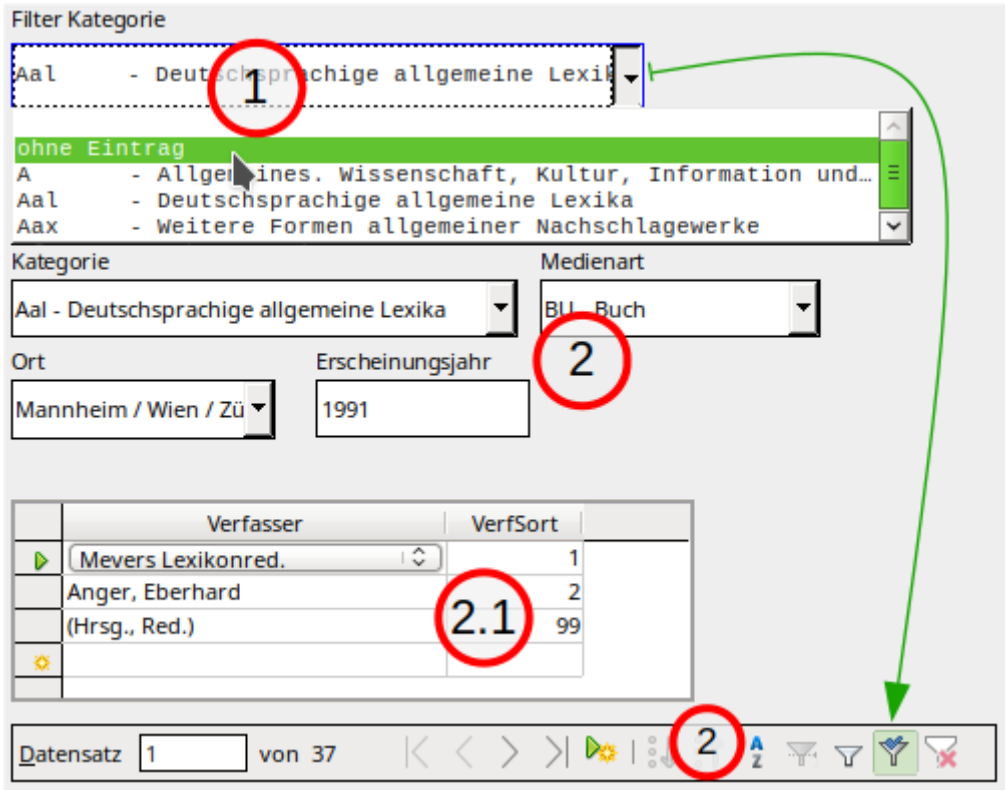
Listenfeld «Filter Kategorie» (Tabelle: *Kategorien, Medien*)

```
001 SELECT DISTINCT LEFT( "Kategorien"."System" || '       ', 7 ) || ' - ' ||
       LEFT("Kategorien"."Kategorie", 50)|| CASE WHEN
       CHAR_LENGTH("Kategorien"."Kategorie") > 50 THEN '…' ELSE '' END AS
    "Kat",
002 "Kategorien"."ID"
003 FROM "Kategorien", "Medien"
004 WHERE "Kategorien"."ID" = "Medien"."katID"
005 ORDER BY "Kat" ASC
```

Schriftart für dieses Listenfeld: Liberation Mono, Standard, 8. Durch die Schriftart mit gleichmäßiger Breite für jeden Buchstaben lässt sich ein Spalteneffekt in dem Listenfeld erzielen.

Wesentlicher Unterschied zu den Formularen ohne Makros ist erst einmal, dass der Button zum Auslösen der Aktion fehlt Nach dem Verlassen des

Filter_Formular_direkt_Nullwerte



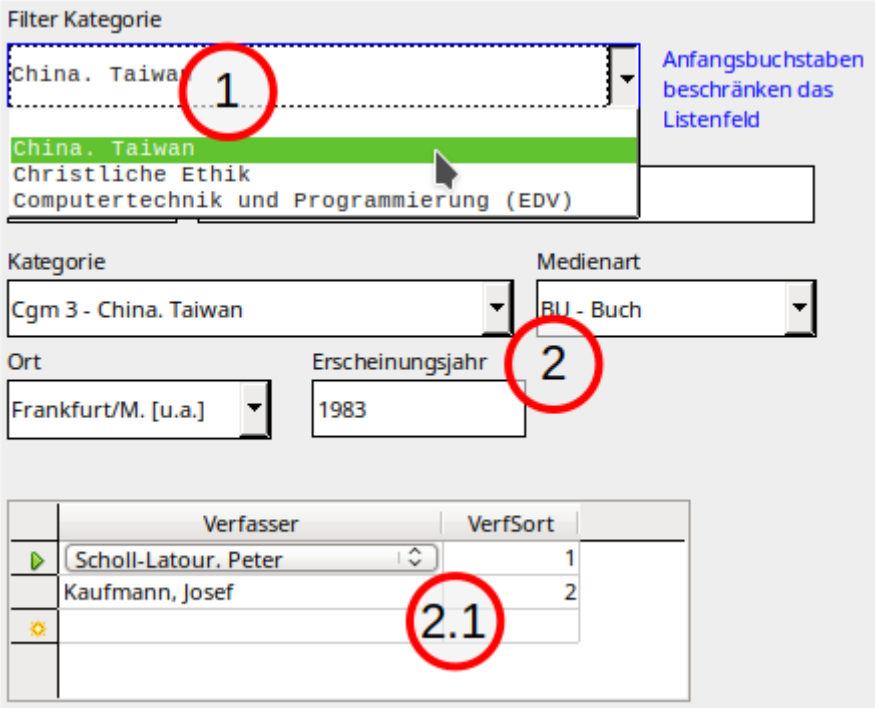
1	Filter (Abfrage: *Filter_Leerfeld)*		
2	MainForm (Abfrage: *Filter_Form_Nullwerte*)	**2.1**	SubForm (Tabelle: *rel_Medien_Verfasser*)

Makros in Feldeigenschaften		
1	Feld «Filter Kategorie» (Modifiziert)	Module1.*Filter_Formular_direkt*

Listenfeld «Filter Kategorie» (Abfrage: *Listenfeld_Kategorie_ohne_Eintrag*)

Bei diesem Formular können nicht nur die Felder, die keinen Eintrag haben, zusätzlich ausgefiltert werden. Hier wird der Filter für das Formular 2 «MainForm» direkt in das Formular geschrieben. Deswegen ist der Filter auch über die Navigationsleiste direkt ein- und ausschaltbar. In dem obigen Screenshot ist der Filter für den Kategorieeintrag 'Aal' gerade aktiv. Würde jetzt das aktive Filtersymbol gedrückt, dann wären statt der 37 aktiv erreichbaren Datensätze wieder alle Datensätze erreichbar.

Filter_Formular_Listfeldbegrenzung



1	Filter (Tabelle: *Filter)*		
2	MainForm (Abfrage: *Filter_Form*)	**2.1**	SubForm (Tabelle: *rel_Medien_Verfasser*)

Makros in Feldeigenschaften		
1	Feld «Filter Kategorie» (Modifiziert)	Module1.*Mauszeiger*
1	Feld «Filter Kategorie» (Taste losgelassen)	Module2.*ListFilter*

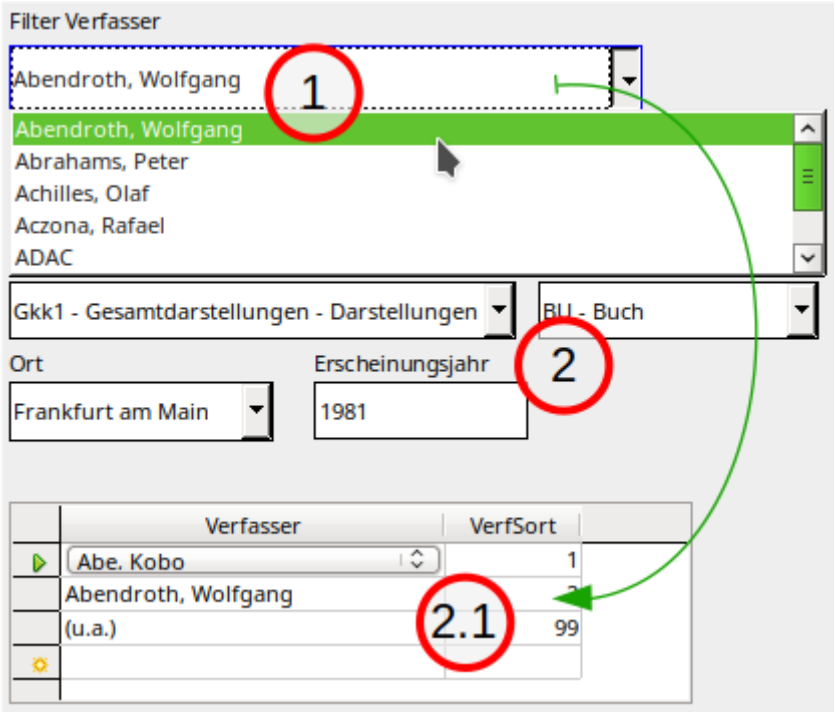
Manchmal ist ein Listenfeld einfach zu umfangreich für die Anzeige aller Daten. In diesem Formular kann deshalb über die Eingabe von Anfangsbuchstaben der Bereich für die angezeigten Kategorien eingeschränkt werden. Hier werden von den 2482 Kategorien nur die 3 Kategorien angezeigt, die mit dem Anfangsbuchstaben 'C' beginnen. Dabei ist es unerheblich, ob mit Kleinbuchstaben oder Großbuchstaben gefiltert wird. Auch die Eingabe von mehreren Buchstaben ist möglich, führt aber schnell zu einem komplett leeren Listenfeld für die Filterung. Aus dieser misslichen Lage geht es über *ESC* wieder zurück.

Der Code für das Listenfeld «Filter Kategorie» ist hier so gehalten, dass er anschließend möglichst einfach über das Makro zur Filterung ergänzt werden kann:

```
001 SELECT "Tab".* FROM
002    (SELECT DISTINCT "Kategorien"."Kategorie" AS "Field",
003       "Kategorien"."ID",
004       "Kategorien"."Kategorie" AS "Sort"
005    FROM "Kategorien", "Medien"
006    WHERE "Kategorien"."ID" = "Medien"."katID")
007 AS "Tab"
008 ORDER BY "Sort" ASC
```

Die eigentliche Abfrage zur Darstellung der Listeninhalte befindet sich in einer Unterabfrage. So können dort entsprechende Bedingungen wie hier z.B. die Beziehungen der Tabellen unter gebracht werden ohne die Bedingung der Filterung zu beeinflussen. Für das Makro werden zwei Felder mit festliegenden Alias benannt: «Field» ist das Feld, das gefiltert werden soll und «Sort» das Feld, nach dem die Sortierung erfolgt. Im Makro kann dann einfach der Teil, der vor dem Tabellenalias «AS "Tab"» steht, ermittelt und gefiltert werden.

Filter_Formular_Subformular

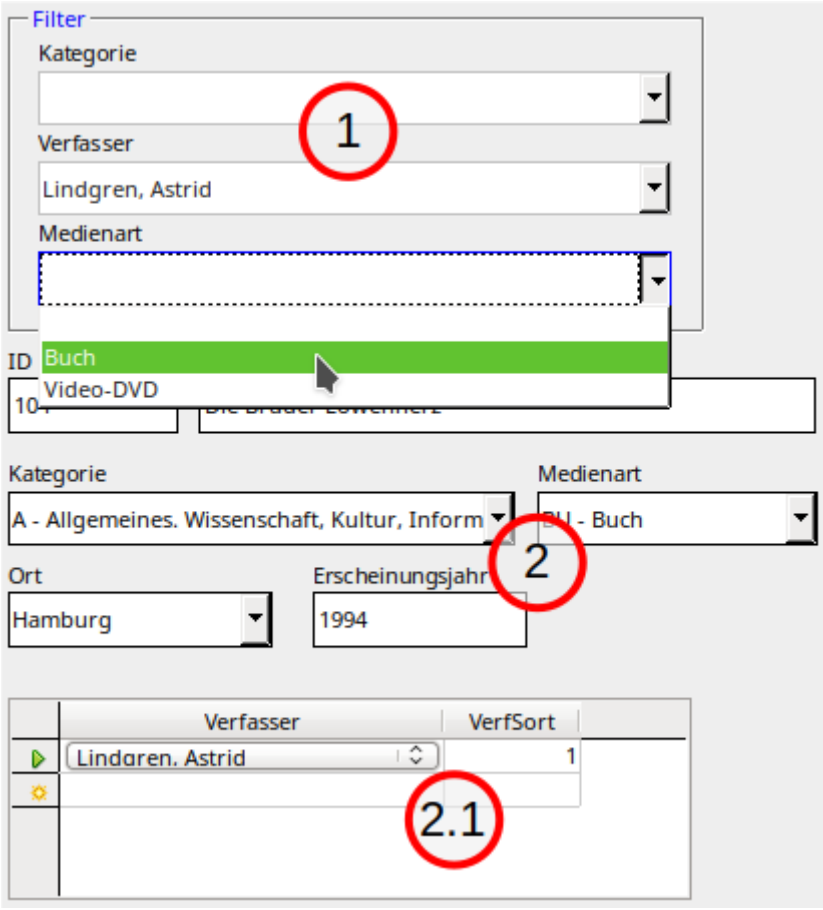


1	Filter (Tabelle: *Filter)*		
2	MainForm (Abfrage: *Filter_Form_Subform*)	**2.1**	SubForm (Tabelle: *rel_Medien_Verfasser*)

Makros in Feldeigenschaften		
1	Feld «Filter Verfasser» (Modifiziert)	Module1.*Mauszeiger*

Wie im entsprechenden Formular ohne Makros lässt sich natürlich auch in diesem Formular mit Makroeinsatz der Inhalt aus dem Unterformular gleich mit filtern. Hier also gerade eine Filterung des Formulars nach dem Verfasser-Eintrag 'Abendroth, Wolfgang'.

Filter_Formular_Subformular_bedingende_Filter



1	Filter (Tabelle: *Filter)*		
2	MainForm (Abfrage: *Filter_Form_Subform_3Filter*)	**2.1**	SubForm (Tabelle: *rel_Medien_Verfasser*)

Makros in Feldeigenschaften		
1	Feld «Filter Kategorie» (Modifiziert)	Module1.*Filter_bedingt*
1	Feld «Filter Verfasser» (Modifiziert)	Module1.*Filter_bedingt*
1	Feld «Filter Medienart» (Modifiziert)	Module1.*Filter_bedingt*

Listenfeld «Filter Kategorie» (Abfrage: *Listenfeld_Kategorie_bedingt*)
Listenfeld «Filter Verfasser» (Abfrage: *Listenfeld_Verfasser_bedingt*)
Listenfeld «Filter Medienart» (Abfrage: *Listenfeld_Medienart_bedingt*)

Bei der Filterauswahl kann mit einem beliebigen Listenfeld gestartet werden. Jedes Listenfeld löst zum einen eine Filterung des Formulars 2 aus, zum anderen auch eine Auffrischung der anderen Listenfelder. Dadurch stehen in den weiteren Listenfeldern nur noch Filtereinträge zur Auswahl, zu denen auch wirklich ein Datensatz existiert. So gibt es denn in dieser Datenbank zu der Verfasserin 'Lindgren, Astrid' nur Bücher und Video-DVDs, nicht aber Zeitschriften oder Spiele, wie sie in dem Formular ohne Makros, *Filter_Formular_Subformular_3Filter*, noch zur Auswahl angeboten werden.

Filter_Formular_Subformular_bedingende_Filter_allround

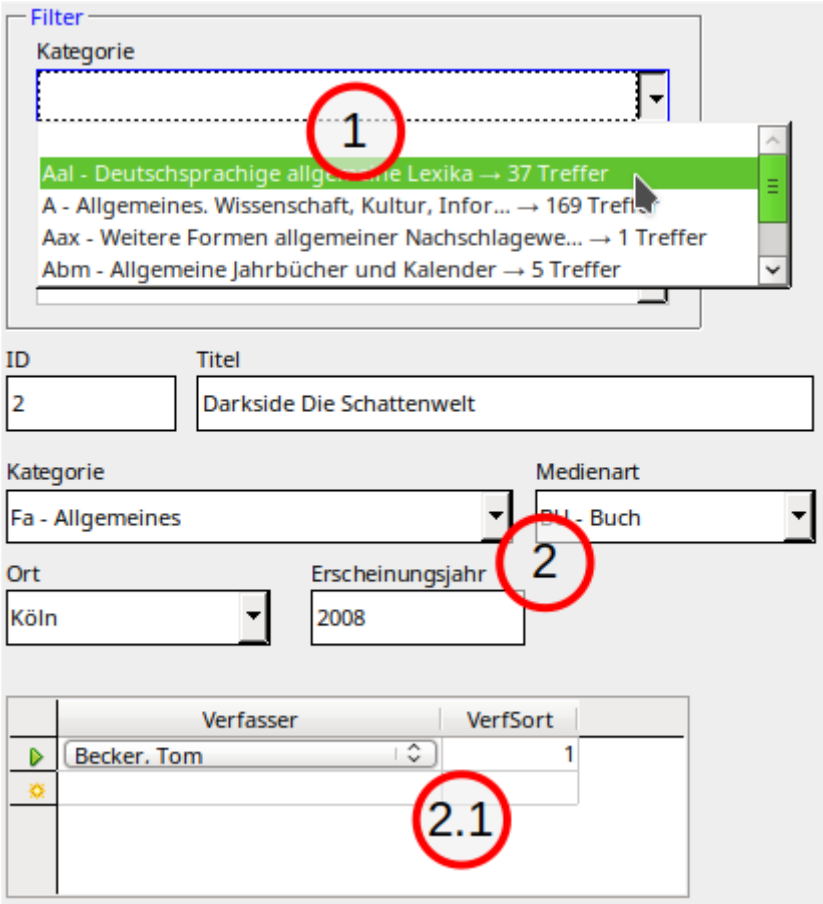
Makros in Feldeigenschaften		
1	Feld «Filter Kategorie» (Modifiziert)	Module1.*Filter_bedingt_allround*
1	Feld «Filter Verfasser» (Modifiziert)	Module1.*Filter_bedingt_allround*
1	Feld «Filter Medienart» (Modifiziert)	Module1.*Filter_bedingt_allround*

Dieses Formular unterscheidet sich von *Filter_Formular_Subformular_bedingende_Filter* durch das verarbeitende Makro. Um eine allgemeine Verarbeitung der Filterung zu erledigen wird in den Listenfeldern in den Zusatzinformationen ein Text hinterlegt:

Listenfeld «Filter_Kategorie» **Allgemein → Zusatzinformation**: lboFilter2,lboFilter3
Listenfeld «Filter_Verfasser» **Allgemein → Zusatzinformation**: lboFilter1,lboFilter3
Listenfeld «Filter_Medienart» **Allgemein → Zusatzinformation**: lboFilter1,lboFilter2

In den Zusatzinformationen befindet sich also lediglich die Information, unter welchem Namen die anderen beiden Listenfelder im Formular zu erreichen sind.

Filter_Formular_Subformular_bedingende_Filter_Datensaetze



1	Filter (Tabelle: *Filter)*		
2	MainForm (Abfrage: *Filter_Form_Subform_3Filter*)	**2.1**	SubForm (Tabelle: *rel_Medien_Verfasser*)

Makros in Feldeigenschaften		
1	Feld «Filter Kategorie» (Modifiziert)	Module1.*Filter_bedingt*
1	Feld «Filter Verfasser» (Modifiziert)	Module1.*Filter_bedingt*
1	Feld «Filter Medienart» (Modifiziert)	Module1.*Filter_bedingt*

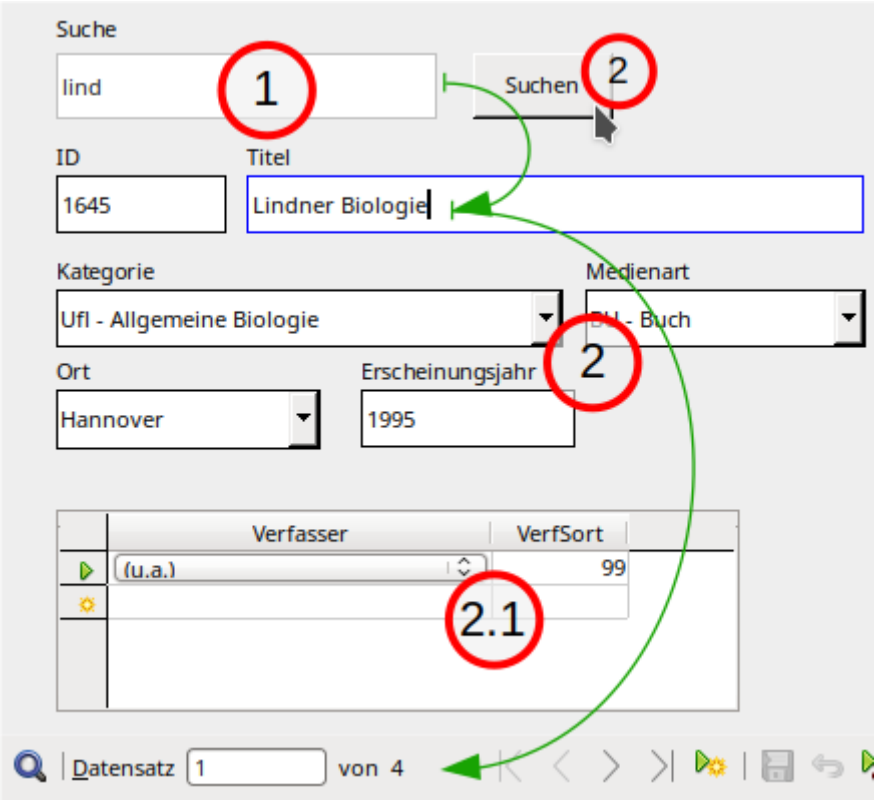
Listenfeld «Filter Kategorie» (Abfrage: *Listenfeld_Kategorie_bedingt_Datensaetze*)
Listenfeld «Filter Verfasser» (Abfrage: *Listenfeld_Verfasser_bedingt_Datensaetze_optimiert*)
Listenfeld «Filter Medienart» (Abfrage: *Listenfeld_Medienart_bedingt_Datensaetze*)

Zusätzlich zu den Funktionen von *Filter_Formular_Subformular_bedingende_Filter* bietet dieses Formular eine Übersicht über die jeweilige Anzahl an vorhandenen Datensätze für die aktuelle Filterung an.

Suche_ohne_Makros

In das Suchfeld wird ein Begriff eingegeben. Groß- und Kleinschreibung wird, wie bei Suchmaschinen, nicht unterschieden. Der Suchbegriff kann an irgendeiner Stelle des zu durchsuchenden Feldes stehen. Die Suche wird über den entsprechenden Button gestartet.

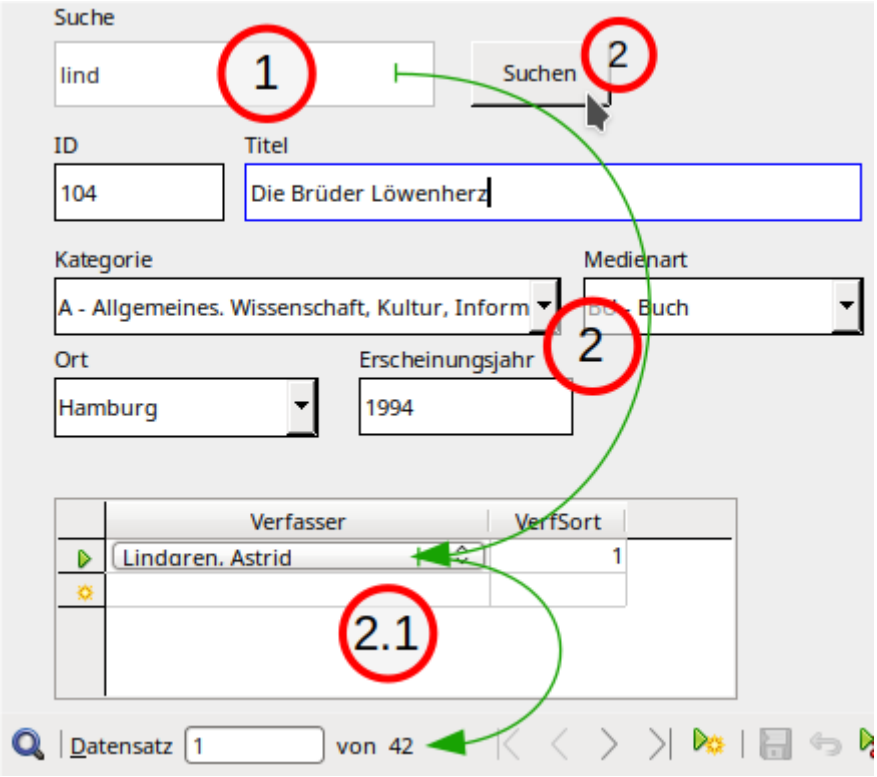
Suche_Formular



1	Suche (Tabelle: *Filter)*		
2	MainForm (Abfrage: *Suche_Form*)	**2.1**	SubForm (Tabelle: *rel_Medien_Verfasser*)

In das Textfeld «Suche» wird ein Begriff eingegeben. Der Button Suchen speichert diesen Begriff durch Verlassen des Formulars «Suche» und aktualisiert das Formular «MainForm», in dem er liegt. Die Suche beschränkt sich auf das Feld «Titel», ergibt also in dem obigen Beispiel insgesamt 4 Datensätze wieder, bei denen im Titel an irgendeiner Stelle der Begriff 'lind' unabhängig von Groß- und Kleinschreibung vorkommt.

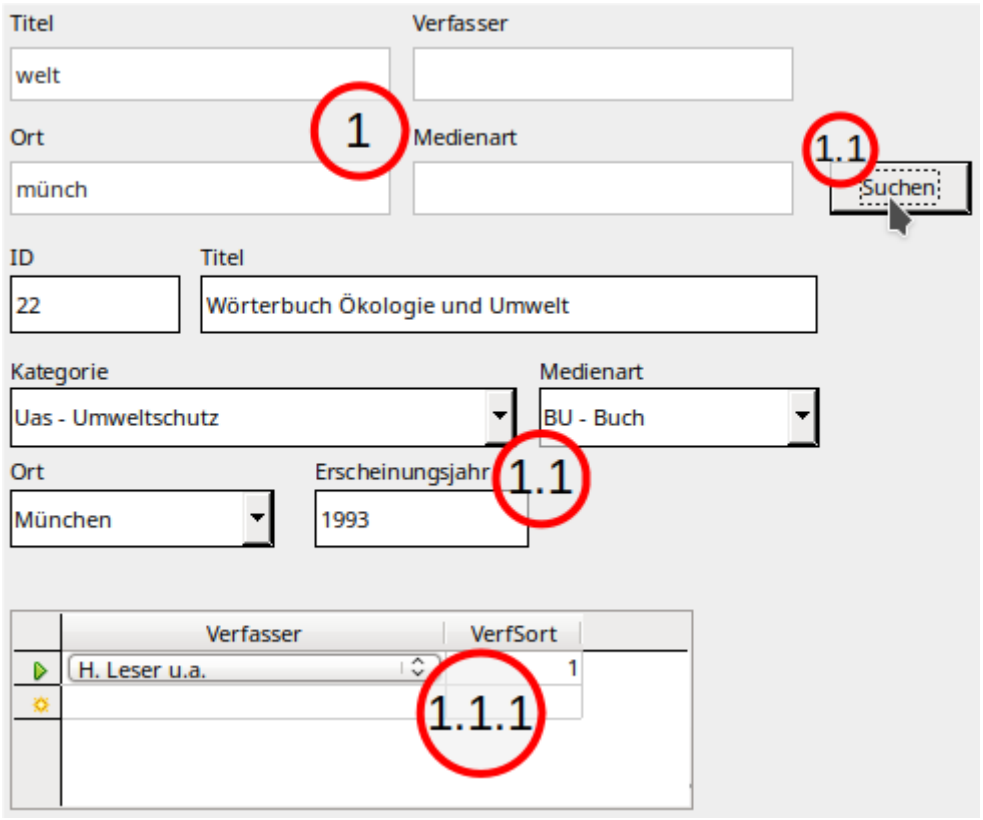
Suche_Formular_Subformular



1	Suche (Tabelle: *Filter)*		
2	MainForm (Abfrage: *Suche_Form_Subform*)	**2.1**	SubForm (Tabelle: *rel_Medien_Verfasser*)

Gegenüber dem Formular *Suche_Formular* ist die Suchmöglichkeit auf weitere Felder ausgedehnt worden. Die Suche findet jetzt durch die entsprechende Abfrage auch Begriffe im Unterformular, die zu dem Suchbegriff 'lind' passen. Die Datensatzzahl steigt dadurch auf insgesamt 42.

Parameter_Suche_Formular_Subformular



1	Suche (Tabelle: *Filter*)	**1.1**	SubForm (Abfrage: *Parametersuche_Medien*)	**1.1.1**	SubSubForm (Tabelle: *rel_Medien_Verfasser*)

Die Verknüpfung des Formulars «Suche» zum Formular «SubForm» erfolgt über alle Parameter, die in der Parameterabfrage vertreten sind.

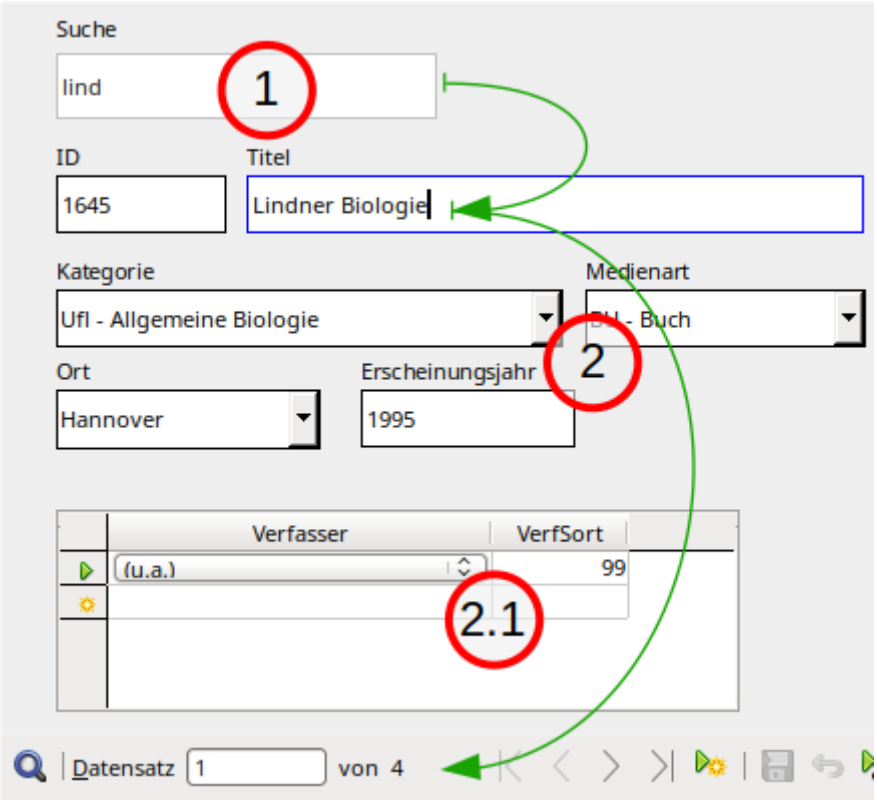
Die freien Texteingaben suchen nur in dem jeweils angegebenen Feld. Die Suche wird über *Suchen* abgeschickt.

Grundsätzlich ist es möglich, auch alle Felder mit einem Parameter zu versorgen. Auch könnte eine Parameterabfrage über entsprechende Listenfelder zum Filtern genutzt werden. Beide Ziele sind hier nicht weiter verfolgt worden, da die Parameterabfrage gegenüber anderen Abfragen nur dann ihre Vorteile ausspielen kann, wenn sie tatsächlich direkt als Abfrage genutzt wird und die Parameter in den entsprechenden Dialog eingegeben werden.

Suche_mit_Makros

Die Suche mit Makros läuft wieder ohne einen Button ab. Sobald mehr als 2 Zeichen Text angegeben werden, wird in dem Formular das Suchergebnis eingeblendet. Dies kann dann durch weitere Eingaben noch verfeinert werden. Soll die Suche wieder beim vollen Datenbestand beginnen, so muss nur das Feld geleert werden.

Suche_Formular

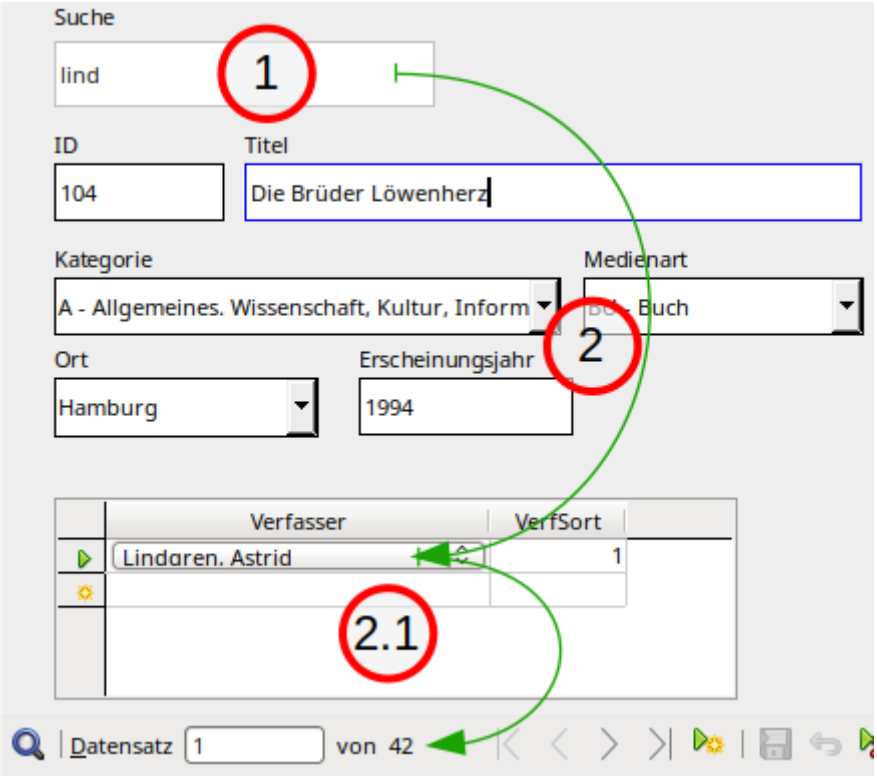


1	Suche (Tabelle: *Filter)*		
2	MainForm (Abfrage: *Suche_Form*)	**2.1**	SubForm (Tabelle: *rel_Medien_Verfasser*)

Makros in Feldeigenschaften		
1	Feld «Suche» (Text modifiziert)	Module1.*Filter_Suche*

Der einzige Unterschied zu *Suche_Formular* ohne Makros ist der, dass hier der Einsatz eine Buttons nicht nötig ist. Es reicht, in das Eingabefeld für «Suche» 3 Zeichen einzugeben. Danach wird das Ergebnis sofort in dem Formular sichtbar und kann weiter verfeinert werden.

Suche_Formular_Subformular

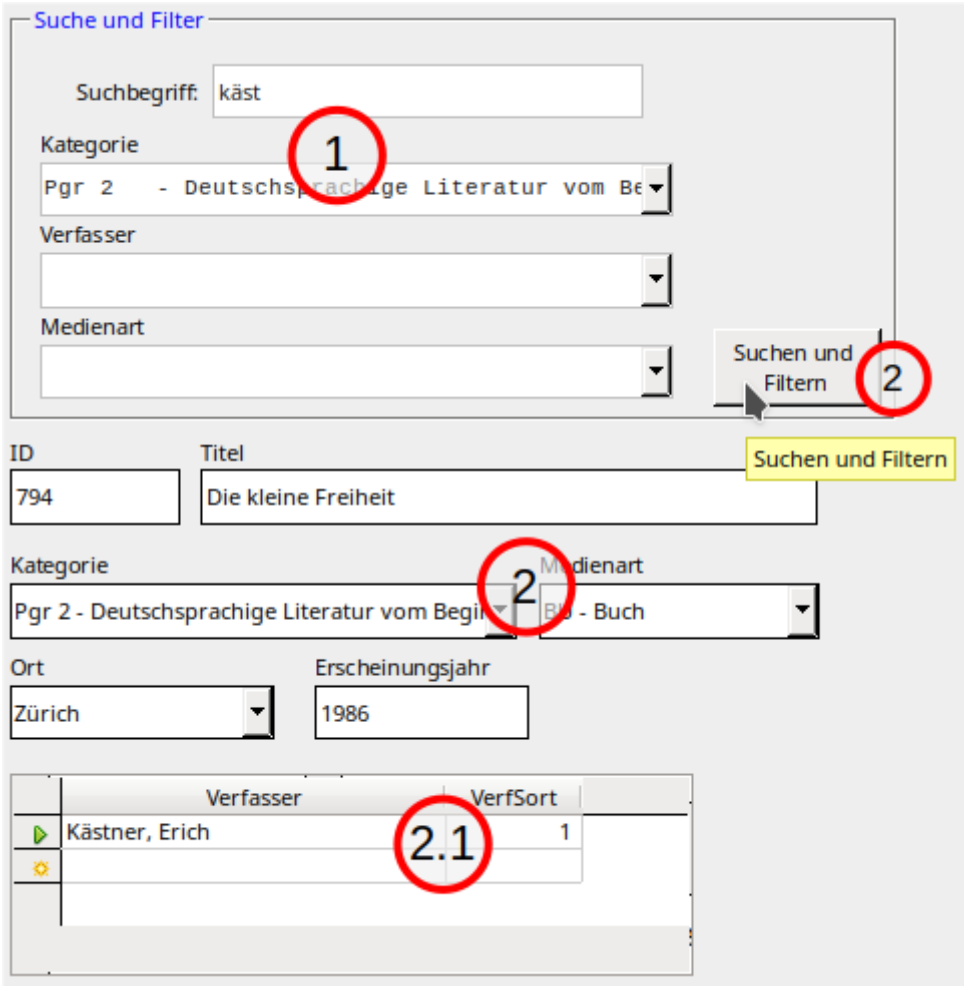


1	Suche (Tabelle: *Filter)*		
2	MainForm (Abfrage: *Suche_Form_Subform*)	**2.1**	SubForm (Tabelle: *rel_Medien_Verfasser*)

Makros in Feldeigenschaften		
1	Feld «Suche» (Text modifiziert)	Module1.*Filter_Suche*

Auch bei diesem Formular gilt: Das Formular unterscheidet sich von *Suche_Formular_Subformular* in der Variante ohne Makros nur durch die Möglichkeit, ohne einen Button das entsprechende Suchergebnis anzuzeigen. Dabei wird ein Suchergebnis erst dann angezeigt, wenn mindestens 3 Buchstaben in das Feld eingegeben wurden. Das Ergebnis wird während der Eingabe präsentiert.

Suche_Filter_Formular_Subformular_ohne_Makros

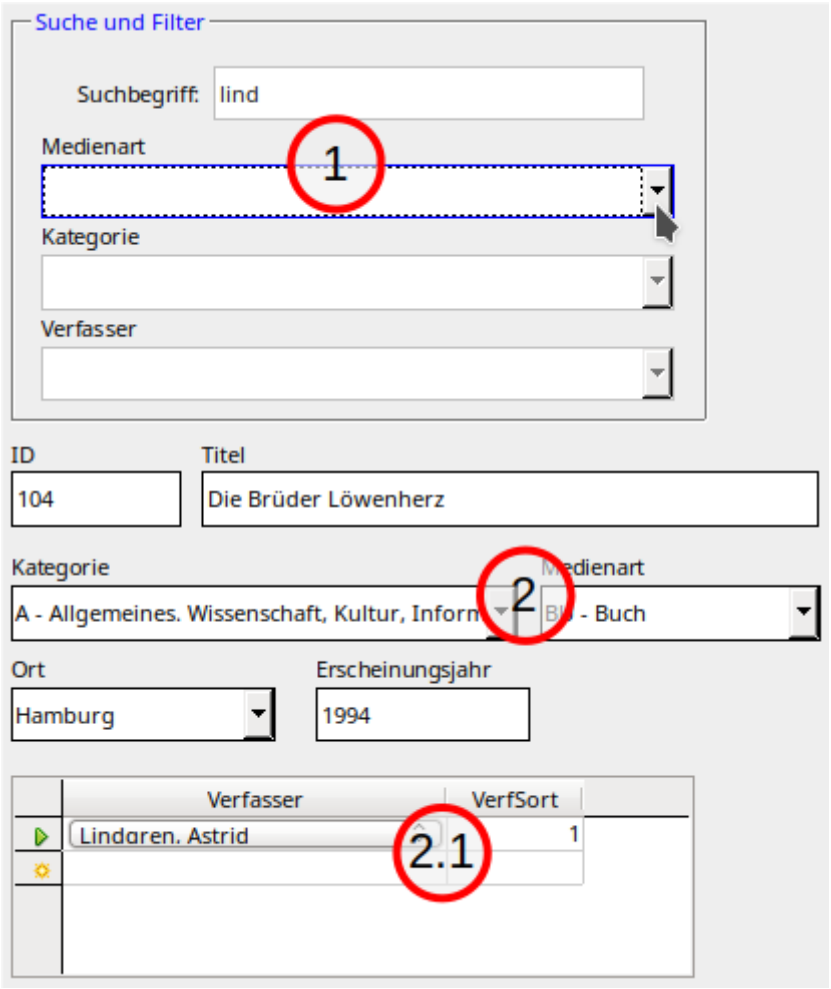


1	Filter (Tabelle: *Filter)*		
2	MainForm (Abfrage: *Suche_Filter_Form_Subform_3Filter*)	**2.1**	SubForm (Tabelle: *rel_Medien_Verfasser*)

Listenfeld «Filter Kategorie» (Abfrage: *Listenfeld_Kategorie_ohne_Eintrag*)

Hier wird die freie Inhaltssuche mit der Filterung durch feste Vorgaben aus den Listenfeldern kombiniert. Ohne Makrozusatz zeigen allerdings die Listenfelder immer den gesamten verfügbaren Inhalt an. So kann eine entsprechende Recherche schnell dazu führen, dass keine Datensätze angezeigt werden. Im aktuellen Fall wurde im Unterformular 2.1 der Verfasser 'Kästner, Erich' gefunden. Bei der Kategorie wurden dann dessen Werke auf die Kategorie 'Pgr 2 ...' eingegrenzt.

Suche_Filter_hierarchisch_mit_Makros



1	Filter (Tabelle: *Filter)*		
2	MainForm (Abfrage: *Suche_Filter_Form_Subform_3Filter*)	**2.1**	SubForm (Tabelle: *rel_Medien_Verfasser*)

Makros im Formulardokument		
Extras → Anpassen → Ereignisse → Ansicht wurde erzeugt		Module1.*Filter_Suche_hierarchisch_Start*
Makros in Feldeigenschaften		
1	Feld «Suchbegriff» (Bei Fokusverlust)	Module1.*Filter_Suche_hierarchisch*
1	Feld «Filter Medienart» (Modifiziert)	Module1.*Filter_Suche_hierarchisch*
1	Feld «Filter Kategorie» (Modifiziert)	Module1.*Filter_Suche_hierarchisch*
1	Feld «Filter Verfasser» (Modifiziert)	Module1.*Filter_Suche_hierarchisch*

Listenfeld «Filter Medienart» (Abfrage: *Listenfeld_Medienart_bedingt_Suche_Filter*)
Listenfeld «Filter Kategorie» (Abfrage: *Listenfeld_Kategorie_bedingt_Suche_Filter*)
Listenfeld «Filter Verfasser» (Abfrage: *Listenfeld_Verfasser_bedingt_Suche_Filter*)

Um eine Aktivierung der Filterfelder zu erledigen wird in den Listenfeldern in den Zusatzinformationen ein Text hinterlegt:

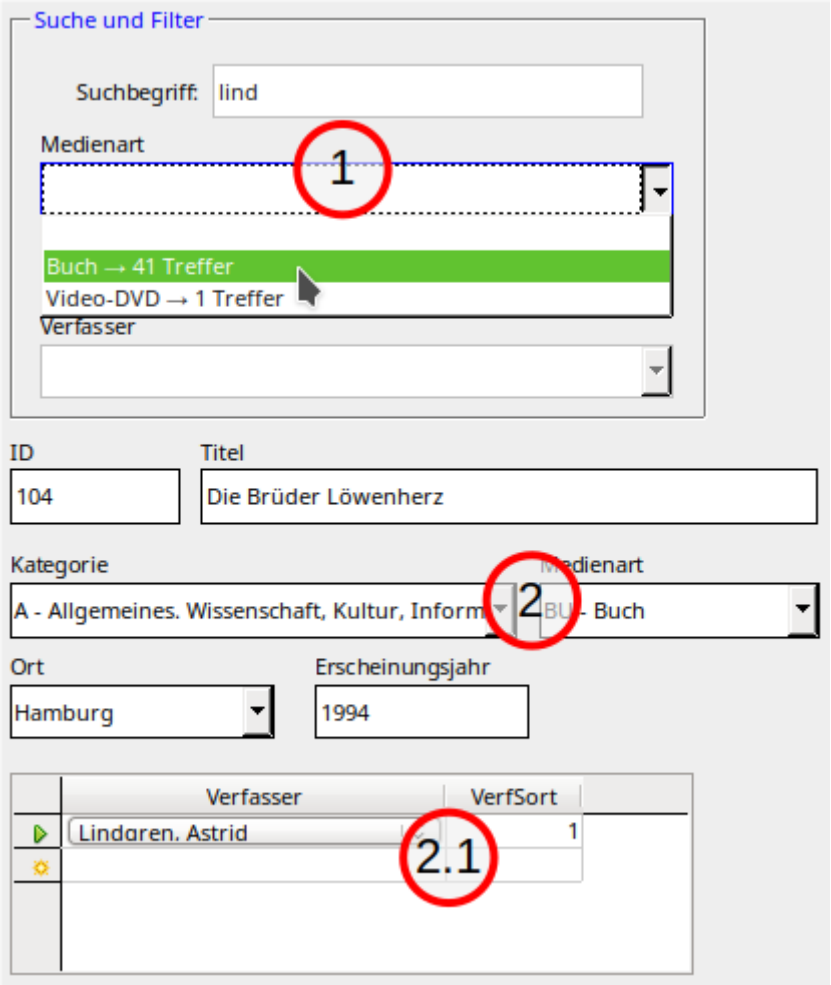
Textfeld «Suchbegriff» **Allgemein → Zusatzinformation**: lboFilter1,lboFilter2
Listenfeld «Filter_Medienart» **Allgemein → Zusatzinformation**: lboFilter2,lboFilter3

Listenfeld «Filter_Kategorie» **Allgemein → Zusatzinformation**: lboFilter3
Listenfeld «Filter_Verfasser» **Allgemein → Zusatzinformation**: kein Eintrag

Das Formular unterscheidet sich in den folgenden Punkten vom Formular *Suche_Filter_Formular_Subformular_ohne_Makros*:

- Die Suche und Filterung läuft automatisch ab und ist nicht an einen Button gebunden.
- Jede Eingabe in dem Such- und Filterblock löst sofort die Filterung auf.
- Jede Eingabe schränkt die weiteren Eingaben ein. Dies gilt allerdings nicht für die freie Eingabe von Text in dem Textfeld «Suchbegriff».
- Die Listenfelder sind hierarchisch aufgebaut. Das bedeutet, dass nur von oben nach unten gewählt werden kann. Die Folgefelder sind anfangs inaktiv. Erst nach einer Auswahl im Listenfeld «Medienart» wird das Listenfeld «Kategorie» für eine Eingabe aktiviert.

Suche_Filter_hierarchisch_allround_Datensaetze_mit_Makros



1	Filter (Tabelle: *Filter)*		
2	MainForm (Abfrage: *Suche_Filter_Form_Subform_3Filter*)	**2.1**	SubForm (Tabelle: *rel_Medien_Verfasser*)

Makros im Formulardokument	
Extras → Anpassen → Ereignisse → Ansicht wurde erzeugt	Module1.*Filter_Suche_hierarchisch_Start*

Makros in Feldeigenschaften		
1	Feld «Suchbegriff» (Bei Fokusverlust)	Module1.*Filter_Suche_hierarchisch_allround*
1	Feld «Filter Medienart» (Modifiziert)	Module1.*Filter_Suche_hierarchisch_allround*
1	Feld «Filter Kategorie» (Modifiziert)	Module1.*Filter_Suche_hierarchisch_allround*
1	Feld «Filter Verfasser» (Modifiziert)	Module1.*Filter_Suche_hierarchisch_allround*

Listenfeld «Filter Medienart» (Abfrage: *Listenfeld_Medienart_bedingt_Suche_Filter_Datensaetze*).

Die folgenden Listenfelder beziehen sich zwar auf die entsprechenden Abfragen. Zum Formularstart ist die Abfrage aber leer. Dadurch wird das Feld erst geladen, wenn es aktiv wird:
Listenfeld «Filter Kategorie» (Abfrage: *Listenfeld_Kategorie_bedingt_Suche_Filter_Datensaetze*)
Listenfeld «Filter Verfasser» (Abfrage: *Listenfeld_Verfasser_bedingt_Suche_Filter_Datensaetze*)

Um eine Aktivierung der Filterfelder zu erledigen und die nicht aktivierten Felder mit einer Datenquelle zu verbinden wird in den Listenfeldern in den Zusatzinformationen ein Text hinterlegt:

Textfeld «Suchbegriff» **Allgemein → Zusatzinformation**:
lboFilter1>Listenfeld_Medienart_bedingt_Suche_Filter_Datensaetze,lboFilter2
Listenfeld «Filter_Medienart» **Allgemein → Zusatzinformation**:
lboFilter2>Listenfeld_Kategorie_bedingt_Suche_Filter_Datensaetze,lboFilter3
Listenfeld «Filter_Kategorie» **Allgemein → Zusatzinformation**:
lboFilter3>Listenfeld_Verfasser_bedingt_Suche_Filter_Datensaetze
Listenfeld «Filter_Verfasser» **Allgemein → Zusatzinformation**: kein Eintrag

Das Formular unterscheidet sich in den folgenden Punkten vom Formular *Suche_Filter_hierarchisch_mit_Makros*:

- Neben den normalen Informationen in den Listenfeldern werden auch die möglichen Treffer angezeigt.
- Die inaktiven Listenfelder werden erst dann mit einer Datenquelle versehen, wenn sie aktiviert werden. Dadurch muss zum Start des Formulars weniger an Daten geladen werden.

Berichte

Die Datenbank enthält keine Berichte.

Makros

Die Makros sind in 2 Modulen untergebracht. In Modul 1 befinden sich alle Makros zum Filtern und Suchen im Formular. Modul 2 enthält nur die Prozedur, die bei einem Listenfeld die Anzahl der anzuzeigenden Datensätze bei der Eingabe begrenzt.

Filter

Aufruf aus
Formular: *Filter_Formular, Filter_Formular_Listfeldbegrenzung, Filter_Formular_Subformular*

```
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 oFeldList AS OBJECT
```

```
007     oDoc = thisComponent
008     oDrawpage = oDoc.drawpage
009     oForm1 = oDrawpage.forms.getByName("Filter")
010     oForm2 = oDrawpage.forms.getByName("MainForm")
011     oFeldList = oForm1.getByName("lboFilter")
012     oFeldList.commit()
013     oForm1.updateRow()
014     oForm2.reload()
015 END SUB
```

Die Zugriffe zu den beiden nebeneinander stehenden Formularen werden definiert (Zeile 9 und 10). Das Listenfeld für die Filterung mit dem Namen «lboFilter» liegt im Formular «Filter». Der eingestellte Wert des Listenfeldes wird in dem aktuellen Datensatz des Formulars durch **commit**() abgelegt (Zeile 12). Er muss jetzt noch in den Datensatz geschrieben werden. Dies geschieht durch **updateRow**() im entsprechenden Formular. Es handelt sich hier um ein Update, da ja keine neuen Datensätze geschrieben werden, sondern nur der (einzige) Datensatz mit dem Primärschlüssel **"ID" = TRUE** geändert wird.

Nach dem Abspeichern wird das Formular, das das Ergebnis der Filterung anzeigen soll, mit **reload**() neu eingelesen (Zeile 14).

Filter_Formular_direkt

Aufruf aus
Formular: *Filter_Formular_direkt_Nullwerte*

```
001 SUB Filter_Formular_direkt
002     DIM oDoc AS OBJECT
003     DIM oDrawpage AS OBJECT
004     DIM oForm1 AS OBJECT
005     DIM oForm2 AS OBJECT
006     DIM oFeldList AS OBJECT
007     DIM stListValue AS STRING
008     oDoc = thisComponent
009     oDrawpage = oDoc.drawpage
010     oForm1 = oDrawpage.forms.getByName("Filter")
011     oForm2 = oDrawpage.forms.getByName("MainForm")
012     oFeldList = oForm1.getByName("lboFilter")
013     stListValue = oFeldList.getCurrentValue()
014     IF stListValue = "" THEN
015        oForm2.Filter = ""
016     ELSE
017        oForm2.Filter = "kat = '" + stListValue + "'"
018     END IF
019     oForm2.reload()
020 END SUB
```

Durch die Erstellung eines direkten Filters in den Formulareigenschaften wird es möglich, den Filter auch über die Navigationsleiste des Formulars an- und auszustellen.

Auch hier wird wieder zuerst der Zugriff auf die Formulare und das Listenfeld erstellt. Anschließend wird der Wert des Listenfeldes ausgelesen. Dies ist nicht der angezeigte Text, sondern (seit LO 4.1) der Inhalt, der an die dem Formular zugrundeliegende Tabelle weiter gegeben werden soll (Zeile 13). In diesem Falle handelt es sich also um den Primärschlüssel der Tabelle "Kategorie".

Wird im Listenfeld das leere Feld gewählt (Listenfeld ganz oben), so ist auch der weitergegebene Inhalt leer. Da die Variable als Text deklariert wurde reicht hier ein **stListValue = ""**, um diesen Zustand zu erkennen. Bei einem leeren Filterwert soll der Filter ausgeschaltet werden: **oForm2.Filter = ""**.

Wird im Listenfeld ein Eintrag gewählt, so wird der entsprechende Schlüssel an die Filterformulierung weiter gegeben. Der gesamte Filter lautet dann z.B. **"kat = '13'"**. Nach Einstellung

des Filters wird das Hauptformular wieder neu geladen (Zeile 19). Damit wird der Filter übernommen.

Filter_bedingt

Aufruf aus
Formular: *Filter_Formular_Subformular_bedingende_Filter, Filter_Formular_Subformular_bedingende_Filter_Datensaetze*

```
001 SUB Filter_bedingt
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    DIM oFeldList3 AS OBJECT
009    oDoc = thisComponent
010    oDrawpage = oDoc.drawpage
011    oForm1 = oDrawpage.forms.getByName("Filter")
012    oForm2 = oDrawpage.forms.getByName("MainForm")
013    oFeldList1 = oForm1.getByName("lboFilter1")
014    oFeldList2 = oForm1.getByName("lboFilter2")
015    oFeldList3 = oForm1.getByName("lboFilter3")
016    oFeldList1.commit()
017    oFeldList2.commit()
018    oFeldList3.commit()
019    oForm1.updateRow()
020    oFeldList1.refresh()
021    oFeldList2.refresh()
022    oFeldList3.refresh()
023    oForm2.reload()
024 END SUB
```

Dieses Makro ersetzt den Aktualisierungsbutton in dem zu filternden Formular. Die Einstellung eines Filters beeinflusst gleichzeitig die anzuzeigenden Inhalte der anderen Filter. Dadurch wird die Auswahl der anderen Filter auf die möglichen Treffer reduziert.

Die Variablen werden deklariert und der Zugang zum Filter-Formular, jedem einzelnen Listenfeld sowie dem Formular, das das Filterergebnis darstellen soll erstellt (Zeile 9 bis 15). Unabhängig davon, welcher Filter ausgelöst wurde, werden von allen Filtern die Werte in die dem Formular zugrundeliegende Filtertabelle übertragen: **`commit()`**. Anschließend wird der geänderte Datensatz abgespeichert: **`updateRow()`**. Danach können die Listenfelder neu eingelesen werden, da ihr Inhalt auf die Einträge in der Filtertabelle zugreift: **`refresh()`**. Schließlich muss nur noch das Formular zur Darstellung des Filterergebnisses mit **`reload()`** neu geladen werden.

Filter_bedingt_allround

Aufruf aus
Formular: *Filter_Formular_Subformular_bedingende_Filter_allround*

```
001 SUB Filter_bedingt_allround(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 oFeld AS OBJECT
007    DIM stTag AS String
008    DIM arList()
009    DIM i AS INTEGER
010    oFeld = oEvent.Source.Model
```

```
011     oForm1 = oFeld.Parent
012     stTag = oFeld.Tag
013     arList = Split(stTag, ",")
014     oDoc = thisComponent
015     oDrawpage = oDoc.drawpage
016     oForm2 = oDrawpage.forms.getByName("MainForm")
017     oFeld.commit()
018     oForm1.updateRow()
019     FOR i = LBound(arList()) TO UBound(arList())
020        oForm1.getByName(arList(i)).refresh()
021     NEXT
022     IF IsEmpty(oFeld.currentValue) THEN
023        oFeld.refresh()
024     END IF
025     oForm2.reload()
026  END SUB
```

Mit diesem Makro werden die anderen Listenfelder vom Inhalt her neu eingelesen, wenn ein Listenfeld von der Auswahl her geändert wurde. Das Listenfeld, dessen Auswahl geändert wurde, wird nur dann neu eingelesen, wenn das leere Feld ausgewählt wurde, also nur eine Auswahl komplett zurück genommen wurde.

Bereits der Aufruf des Makros berücksichtigt, von welchem Listenfeld aus das Signal kam: **`oEvent AS OBJECT`**. Anschließend werden die Variablen deklariert. Besonders hinzuweisen ist hier auf die Deklaration des Arrays, die im voraus erfolgen muss: **`arList()`**.

Das Feld wird über das auslösende Ereignis ermittelt (Zeile 10). Das entsprechende Formular ist direkt dem Feld übergeordnet: **`oFeld.Parent`**. Entsprechend muss auch der Name des Formulars nicht mehr gesondert im Makro erscheinen.

Aus den Zusatzinformationen sollen die Bezeichnungen für die Listenfelder ausgelesen werden, die aktualisiert werden sollen. Die Zusatzinformationen sind gespeichert in **`oFeld.Tag`**. Die Bezeichnungen der Listenfelder werden anhand des Kommas voneinander getrennt und in das bereits definierte Array eingelesen: **`arList = Split(stTag, ",")`**. Hier wird davon ausgegangen, dass die Listenfeldbezeichnungen nur durch ein Komma getrennt werden. Sonst müssten gegebenenfalls noch Leerstellen mit Hilfe von **`Trim()`** entfernt werden.

Des weiteren erfolgt noch ein Zugriff auf das Formular, das die gefilterten Daten darstellen soll (Zeile 14 bis 16). Dieses Formular muss zum Schluss neu geladen werden (Zeile 25).

Das Listenfeld gibt seinen Wert ab (**`commit()`**). Die Änderung in dem Formular, in dem das Listenfeld steht, wird abgespeichert.

Anschließend wird in einer Schleife das Array durchgegangen (Zeile 19 bis 21). **`LBound(arList())`** zeigt den unteren Wert '0' des Arrays an. **`Ubound(arList())`** verweist auf den höchsten Wert, im praktischen Beispiel '1'.

Jede Bezeichnung für ein Listenfeld wird eingelesen. So präsentiert **`arList(0)`** das erste Listenfeld aus den Zusatzinformationen, aus dem obigen Screenshot also lboFilter2, **`arList(1)`** ist gleich lboFilter3. Das entsprechende Feld wird durch **`refresh()`** neu eingelesen.

Schließlich wird noch nachgesehen, ob der Wert für das aktuelle Listenfeld, den Auslöser für das Ereignis, jetzt leer ist. Nur dann soll auch das aktuelle Listenfeld neu eingelesen werden (Zeile 22 bis 24). Ansonsten kann einfach ein anderer Wert aus der alten Liste gewählt werden, falls der erste Wert nicht zum Erfolg führte.

Filter_Suche

Aufruf aus
Formular: *Suche_Formular, Suche_Formular_Subformular*

```
001  SUB Filter_Suche(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 stFeld AS STRING
007     oDoc = thisComponent
008     oDrawpage = oDoc.drawpage
009     oForm2 = oDrawpage.forms.getByName("MainForm")
010     oFeld = oEvent.Source.Model
011     stFeld = oFeld.CurrentValue
012     IF Len(stFeld) > 2 OR Len(stFeld) > 0
           OR oFeld.ImplementationName = "com.sun.star.form.OListBoxModel" THEN
013        oForm1 = oFeld.Parent
014        oFeld.commit()
015        oForm1.updateRow()
016        oForm2.reload()
017     END IF
018  END SUB
```

Mit diesem Makro wird aus einem Suchfeld heraus eine Suche vorgenommen, wenn entsprechend viele Buchstaben in dem Feld vorhanden sind. Die Aktualisierung ist fortlaufend. Die Verzweigung zum Ablauf der Aktualisierung der Suche läuft dann ab, wenn das Suchfeld mehr als 2 Buchstaben enthält oder die enthaltenen Buchstaben alle entfernt wurden (Zeile 12 bis 17). Damit das Makro auch mit Listenfeldern zur Filterung genutzt werden kann werden die Listenfelder von dieser Begrenzung ausgenommen (Zeile 12).

Der Kontakt zum Formular, das das Suchergebnis darstellen soll, wird auf die übliche Weise über die **Drawpage** erstellt.

Der Kontakt zum Suchformular und zu dem Eingabefeld entsteht durch das auslösende Ereignis. Dadurch wird das Feld und das Formular ermittelt. Mit **commit()** wird der Wert in die Tabelle "Filter" übertragen, mit **updateRow()** wird der geänderte Datensatz gespeichert. Anschließend wird das Formular zur Darstellung des Suchergebnisses über **reload()** neu geladen.

Filter_Suche_hierarchisch

Aufruf aus
Formular: *Suche_Filter_hierarchisch_mit_Makros*

```
001  SUB Filter_Suche_hierarchisch(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 oFeld AS OBJECT
007     DIM stTag AS String
008     DIM arList()
009     DIM i AS INTEGER
010     oDoc = thisComponent
011     oDrawpage = oDoc.drawpage
012     oForm2 = oDrawpage.forms.getByName("MainForm")
013     oFeld = oEvent.Source.Model
014     oFeld.commit()
015     oForm1 = oFeld.Parent
016     oForm1.updateRow()
017     stTag = oFeld.Tag
018     arList = Split(stTag, ",")
019     FOR i = LBound(arList()) TO UBound(arList())
020        IF IsEmpty(oFeld.currentValue) THEN
021           oForm1.getByName(arList(i)).Enabled = FALSE
022        ELSE
023           oForm1.getByName(arList(0)).Enabled = TRUE
024        END IF
025        oForm1.getByName(arList(i)).BoundField.UpdateNULL()
026        oForm1.updateRow()
```

```
027         oForm1.getByName(arList(i)).refresh()
028      NEXT
029      oForm2.reload()
030   END SUB
```

Die Verbindungen zu den Formularen und zum aktuellen Formularfeld werden wie bereits in den vorhergehenden Makros erstellt. Das aktuelle Feld wird abgespeichert (Zeile 14 bis 16). Der Inhalt aus den Zusatzinformationen (**Tag**) wird in das Array «arList» eingelesen. Anschließend wird das Array in einer Schleife durchlaufen (Zeile 19 bis 28).

Wenn das leere Feld gewählt wurde, dann wird für jeden Eintrag in der Arrayliste das Listenfeld deaktiviert: **Enabled = FALSE**. Wenn das gewählte Feld nicht leer ist wird das erste Feld aus dem Array aktiviert: **Enabled = TRUE**. Das zweite Feld bleibt hingegen deaktiviert.

Für alle Listenfelder aus dem Array wird der Datenbestand aus dem Formular entfernt: **UpdateNull()**. Danach wird der Datensatz gespeichert und das Listenfeld neu eingelesen, also ohne Wert dargestellt (Zeile 26 und 27). So kann mit dem aktuellen Listenfeld auch eine gerade erstellte Wahl wieder geändert werden, ohne dass dadurch plötzlich kein Datensatz mehr in der Ergebnismenge vorhanden ist.

Schließlich wird das zu filternde Formular neu geladen.

Filter_Suche_hierarchisch_Start

Aufruf aus
Formular: *Suche_Filter_hierarchisch_mit_Makros,* *Suche_Filter_hierarchisch_allround_Datensaetze_mit_Makros*

Benötigt
Tabelle: *Filter*

```
001   SUB Filter_Suche_hierarchisch_Start
002      DIM oDatenquelle AS OBJECT
003      DIM oVerbindung AS OBJECT
004      DIM oSQL_Anweisung AS OBJECT
005      DIM stSql AS STRING
006      oDatenquelle = ThisComponent.Parent.CurrentController
007      If NOT (oDatenquelle.isConnected()) THEN
008         oDatenquelle.connect()
009      END IF
010      oVerbindung = oDatenquelle.ActiveConnection()
011      oSQL_Anweisung = oVerbindung.createStatement()
012      stSql = "DELETE FROM ""Filter"""
013      oSQL_Anweisung.executeUpdate(stSql)
014      stSql = "INSERT INTO ""Filter"" (""ID"") VALUES (True)"
015      oSQL_Anweisung.executeUpdate(stSql)
016   END SUB
```

Die Einstellungen in der Tabelle "Filter" werden standardmäßig gespeichert und stehen beim nächsten Start des Formulars wieder zur Verfügung. Mit diesem Makro werden die Filtereinstellung beim Start eines Formulars gelöscht, damit die Filterung komplett neu beginnt.

Zuerst wird die Datenquelle für das aktuelle Datenbankdokument aufgesucht. Interne Formulare einer Base-Dokumentes erzeugen beim Öffnen automatisch eine Verbindung zu der Datenquelle. Besteht bisher keine Verbindung, so wird anschließend eine Verbindung zur Datenbank hergestellt (Zeile 6 bis 9).

Jede SQL-Anweisung muss über **createStatement()** zuerst vorbereitet werden. Der Inhalt für die Anweisung wird anschließend in einer Variablen gespeichert. Anschließend wird die Variable mit dem Befehl **executeUpdate()** an die Datenbank weiter gegeben. Eine Ausgabe der Daten wird nicht erwartet.

Mit **DELETE FROM "Filter"** werden alle Datensätze in der Tabelle "Filter" gelöscht. Im Makro sind hier doppelte Anführungszeichen um den Tabellennamen gesetzt, damit die einfachen Anführungszeichen in dem Kommando weiter gegeben werden. Anführungszeichen werden mit Anführungszeichen maskiert.

Mit **INSERT INTO "Filter" ("ID") VALUES (TRUE)** wird der für die Filterung benötigte Datensatz wieder erstellt. Einziger Eintrag ist hier der Wert für das Primärschlüsselfeld, der bei der Filterung existieren muss.

Filter_Suche_hierarchisch_allround

Aufruf aus
Formular: *Suche_Filter_hierarchisch_allround_Datensaetze_mit_Makros*

```
001 SUB Filter_Suche_hierarchisch_allround(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 oFeld AS OBJECT
007    DIM stTag AS String
008    DIM arList()
009    DIM arTmp()
010    DIM stSql(0) AS STRING
011    DIM i AS INTEGER
012    oDoc = thisComponent
013    oDrawpage = oDoc.drawpage
014    oForm2 = oDrawpage.forms.getByName("MainForm")
015    oFeld = oEvent.Source.Model
016    oFeld.commit()
017    oForm1 = oFeld.Parent
018    oForm1.updateRow()
019    stTag = oFeld.Tag
020    arList = Split(stTag, ",")
021    FOR i = LBound(arList()) TO UBound(arList())
022       IF i = 0 THEN
023          arTmp = Split(arList(0), ">")
024          arList(0) = arTmp(0)
025          stSql(0) = arTmp(1)
026       END IF
027       IF IsEmpty(oFeld.currentValue) THEN
028          stSql(0) = ""
029          oForm1.getByName(arList(i)).Enabled = FALSE
030          oForm1.getByName(arList(i)).ListSource = stSql
031       ELSE
032          oForm1.getByName(arList(0)).Enabled = TRUE
033          oForm1.getByName(arList(0)).ListSource = stSql
034       END IF
035       oForm1.getByName(arList(i)).BoundField.UpdateNULL()
036       oForm1.updateRow()
037       oForm1.getByName(arList(i)).refresh()
038    NEXT
039    oForm2.reload()
040 END SUB
```

Der Grundaufbau dieser Prozedur entspricht erst einmal der Prozedur «Filter_Suche_hierarchisch». Bereits bei der Deklaration kommen zwei Arrays als Variablen hinzu. Das Array **arTmp()** nimmt nur zwischendurch das Ergebnis der Funktion **Split()** auf und gibt die Teile entsprechend weiter. Das Array **stSql()** wird von vornherein als ein Array definiert, das nur einen Datensatz enthält, der außerdem ein **STRING** sein muss (Zeile 9 und 10).

Die Inhalte aus den Zusatzinformationen werden, wie bei «Filter_Suche_hierarchisch», ausgelesen und in einer Schleife abgearbeitet. Nur beim ersten Durchgang der Schleife, wenn also **i =**

0, wird das erste Element des Arrays **arList()** erneut gesplittet (Zeile 22 bis 26), jetzt aber bei dem Zeichen «>». Der erste Teil enthält dann den Namen des Listenfeldes, z.B. 'lboFilter2'. Der zweite Teil enthält den Namen der Abfrage, die das Listenfeld mit den auszuwählenden Daten versehen soll. Der zweite Teil ist der einzige Eintrag, der in das Array **stSql** als **stSql(0)** eingetragen wird.

Ist der ausgewählte Inhalt aus dem Listenfeld leer, so soll auch der Name der Datenquelle wieder gelöscht werden. Daher steht hier **stSql(0) = ""**. Ein Listenfeld, das anschließend deaktiviert wird, braucht nicht den gesamten Datenbestand wieder einzulesen. Mit **ListSource = stSql** wird das gesamte Array als Datenquelle angegeben. **ListSource** erwartet ein Array vom Typ **STRING** – als Hinweis auf eine Abfrage, als Abfrage direkt, als Datensammlung ...

Ist ein Inhalt in dem aktuellen Listenfeld ausgewählt worden (**oFeld.currentValue** ist nicht leer), so wird das in der Hierarchie folgende Listenfeld aktiviert sowie mit einer Datenquelle versehen (Zeile 32 und 33).

Standardmäßig werden die Formularfelder für alle dem aktuellen Listenfeld in der Hierarchie folgenden Listenfelder auf **NULL** gesetzt, der Filter mit **updateRow()** auf die entsprechenden Werte eingestellt und die folgenden Listenfelder neu eingelesen (Zeile 35 bis 37).

Zum Schluss wird der entsprechende Filterwert auf das Formular, das das Filterergebnis anzeigen soll, übertragen: **oForm2.reload()**.

ListFilter

Aufruf aus
Formular: *Filter_Formular_Listfeldbegrenzung*

```
001 GLOBAL stListStart AS STRING
002 GLOBAL lZeit AS LONG
```

```
001 SUB ListFilter(oEvent AS OBJECT)
002    oFeld = oEvent.Source.Model
003    IF oEvent.KeyCode < 538 OR oEvent.KeyCode = 1283 OR oEvent.KeyCode = 1284
       OR oEvent.KeyCode = 1281 THEN
004       DIM stSql(0) AS STRING
005       DIM stText AS STRING
006       DIM stFeld AS STRING
007       DIM stQuery AS STRING
008       ar() = Split(oFeld.ListSource(0)," AS ""Tab""")
009       stQuery = ar(0) & " AS ""Tab"" WHERE"
010       IF oEvent.KeyCode = 1281 THEN  'Taste ESC
011          stListStart = "" 'Bei ESC wieder den gesamten Inhalt anzeigen
012       ELSEIF lZeit > 0 AND Timer() - lZeit < 5 THEN
013          stListStart = stListStart & oEvent.KeyChar
014       ELSE
015          stListStart = oEvent.KeyChar
016       END IF
017       lZeit = Timer()
018       stText = LCase( stListStart & "%")
019       stSql(0) = stQuery + "LOWER(""Field"") LIKE '"+stText+"'
             ORDER BY ""Sort"""
020       oFeld.ListSource = stSql
021       oFeld.refresh
022    END IF
023 END SUB
```

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.

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.

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. Er lässt sich aber auch einfach über **msgbox oEvent.KeyCode** bei der Eingabe in ein Formularfeld erfahren. 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 die Backspace-Taste. 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 (Zeile 3).

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.

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.

Der SQL-Code wird nach der Deklaration der Variablen für die weitere Verwendung aufgesplittet. Für das Feld, das gefiltert werden soll, wird nach dem vordefinierten Alias **AS "Tab"** der Code abgetrennt (Zeile 8).

Anschließend wird an den Code der Alias zusammen mit dem Start für die Filterbedingung (WHERE) wieder angefügt.

In der folgenden Verzweigung wird zuerst wird die Bedingung abgefragt, ob *ESC* gedrückt wurde (Zeile 10). 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 (Zeile 12 und 13). Anderenfalls wird der eingegebene Buchstabe als einzige (neue) Eingabe verstanden (Zeile 15). Das Listenfeld wird dann einfach neu nach dem entsprechenden Buchstaben gefiltert. Anschließend wird die aktuelle Zeit wieder in der globalen Variablen **lZeit** gespeichert.

Der SQL-Code wird schließlich zusammengefügt (Zeile 18 und 19). 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.

Haushaltsbuch

Einführung

Wo ist im Haushalt bloß das Geld geblieben. Das Nachverfolgen des Finanzstromes lässt sich mit entsprechenden Abfragetechniken direkt innerhalb einer Abfrage auch berechnen und dann noch zusätzlich in Formularen übersichtlich geordnet nach Konten und Kategorien darstellen. Die Datenbank[8] lässt sich mit unterschiedlichen Formularen, gegebenenfalls auch ganz ohne Makrozusatz, bedienen.

Tabellen

Kasse

Datenziel	
Ansicht:	*Ansicht_Kasse_Kategorie, Ansicht_Kasse_mit_Umbuchungen, Ansicht_Bericht_Kategorie*
Abfrage:	*Kasse_Saldo_Konto_Kategorie, Kasse_Saldo_Konto_Kategorie_Umbuch, Fehler: Verweis nicht gefunden, Saldo_Kategorie, Saldo_Kategorie_ungesplittet*
Formular:	*Konto_Salden_komplett_Umbuchung_eM, Konto_Salden_komplett_Umbuchung*
Makro:	*periodische_Buchung, SplitRest*
Bericht: keiner direkt	

Feldname	***Feldtyp***	***Beschreibung***
ID	Integer	Primärschlüssel der Tabelle. Der Primärschlüssel muss eindeutig sein. Das Feld ist als Auto-Wert-Feld gesetzt.
Betrag	Zahl	Der Betrag, der aus dem folgenden Konto abgebucht oder hinzu gebucht wird. Bei der Felderstellung ist auf 2 Nachkommastellen zu achten. Eingabe erforderlich: Ja
Konto_ID	Tiny Integer	Fremdschlüssel aus der Tabelle "Konto". Eingabe erforderlich: Ja
Datum	Datum	Hier soll das Datum gespeichert werden, zu dem die Einnahme/Ausgabe erfolgt ist. Eingabe erforderlich: Ja
Adressat_ID	Integer	Fremdschlüssel aus der Tabelle "Adressat". Eingabe erforderlich: Nein
Verwendungszweck	Text	Wofür ist das Geld verwandt worden? Dieses Feld speichert Zusatzinformationen zu den Kategorien. Eingabe erforderlich: Nein
Umbuch_Konto_ID	Tiny Integer	Wird Geld z.B. vom Girokonto abgehoben, so wechselt das Geld lediglich das Konto von «Giro» zu «Bargeld», ist aber noch nicht anderweitig ausgegeben. Eingabe erforderlich: Nein
periodisch	Ja/Nein	Wenn eine Buchung häufiger in ähnlicher Art und Weise vorkommt, so soll sie in diesem Feld gekennzeichnet werden. Diese Buchungen können in einem Formular als neue Datensätze bis auf das Datum übernommen werden. Eingabe erforderlich: Nein

8 Beispieldatenbank Beispiel_Haushaltsbuch.odb

Feldname	Feldtyp	Beschreibung
Kategorie_ID	Tiny Integer	Zuweisung der Kategorie, wenn Splitbuchungen nicht geplant sind. Ermöglicht es, Formulare ohne Unterformulare zu erstellen. Eingabe erforderlich: Nein

Um die Umbuchung von einem Konto zum anderen Konto so abzusichern, dass tatsächlich von einem Konto zum anderen gebucht wird und nicht das abgebende Konto gleich dem aufgebenden Konto ist, wird innerhalb der Tabelle ein **CONSTRAINT** definiert. Über **Extras → SQL** wird eingegeben:

```
001 ALTER TABLE "Kasse" ADD CONSTRAINT "Konto ungleich Umbuchungskonto" CHECK
    ("Umbuch_Konto_ID" <> "Konto_ID");
```

Sofern ein Eintrag im Feld "Umbuch_Konto_ID" erstellt wird wird dieser Vergleich vorgenommen. Das Feld "Umbuch_Konto_ID" darf also sehr wohl leer sein, darf eben nur nicht den gleichen Wert wie "Konto_ID" haben.

Das Feld für die Eingabe der Kategorie soll standardmäßig mit '0' beschrieben werden, wenn von dem Formular aus keine Eingabe erfolgt. Dafür muss vorher in der Tabelle *Kategorie* der entsprechende Datensatz mit der "ID" = '0' eingefügt worden sein.

```
001 ALTER TABLE "Kasse" ALTER COLUMN "Kategorie_ID" SET DEFAULT 0
```

Den Primärschlüsselwert '0' hat in der Tabelle "Kategorie" der Eintrag 'ohne Zuweisung'. Wird also keine Angabe im Feld "Kategorie_ID" gemacht, so wird der Betrag 'ohne Zuweisung' geführt. Dass dieses Feld in der Tabelle nicht mit einer erforderlichen Eingabe versehen worden ist liegt daran, dass vorher bereits Eingaben ohne Zuweisung der Kategorie, vor allem bei Umbuchungen, getätigt wurden. Außerdem ist das Feld "Kategorie_ID" bei den Formularen, die die Splitbuchung benutzen, gar nicht belegt. Sollte also die Splitbuchung nicht benötigt werden, so ließe sich hier mit einer klaren Anweisung von **Eingabe erforderlich → Ja** und dem Setzen des SQL-Default-Wertes der Code in einigen Abfragen vereinfachen.

Adressat

Datenziel	
Ansicht:	*Ansicht_Kasse_Kategorie, Ansicht_Kasse_mit_Umbuchungen*
Makro:	*Adressat_neu*
Abfrage, Formular, Bericht: keine direkt	

Feldname	Feldtyp	Beschreibung
ID	Integer	Primärschlüssel der Tabelle. Der Primärschlüssel muss eindeutig sein. Das Feld ist als Auto-Wert-Feld gesetzt.
Adressat	Text	Woher stammt die Rechnung, wer erhält das Geld? Eingabe erforderlich: Ja

Diese Tabelle kann natürlich beliebig erweitert werden, wenn z.B. mittels Kontoverbindung entsprechende Überweisungen gemacht werden sollten. Hier ist nur die einfache Variante mit einem Namen erstellt worden. Diese Tabelle wird entweder direkt oder über eine Input-Box mittels Makro beschrieben.

Konto

Datenziel	
Ansicht:	*Ansicht_Kasse_Kategorie, Ansicht_Kasse_mit_Umbuchungen*
Abfrage:	*Kasse_Saldo_Konto_Kategorie, Kasse_Saldo_Konto_Kategorie_Umbuch*
Formular:	*Konto_oM, Konto_Salden_oM, Konto_Salden_separat_oM, Konto_Salden_komplett_eM, Konto_Salden_komplett_Umbuchung_eM, Konto, Konto_Salden, Konto_Salden_komplett, Konto_Salden_komplett_Umbuchung, Buchung_Umbuchung_Salden*
Bericht, Makro: keine direkt	

Feldname	Feldtyp	Beschreibung
ID	Tiny Integer	Primärschlüssel der Tabelle. Der Primärschlüssel muss eindeutig sein. Die Kontoverbindungen bleiben überschaubar, so dass lediglich eine kleine Zahl ohne Auto-Wert notwendig ist.
Konto	Text	Bezeichnung für das Konto, wie es auch in den Listenfeldern erscheint. Eingabe erforderlich: Ja
IBAN	Text	IBAN-Kontobezeichnung Eingabe erforderlich: Nein
Bank	Text	Bezeichnung der Bankverbindung Eingabe erforderlich: Nein
BIC	Text	BIC-Bankbezeichnung für internationalen Zahlungsverkehr Eingabe erforderlich: Nein

Die einfache Variante wäre hier die alleinige Bezeichnung des Kontos. In dem Beispiel existieren lediglich 3 Konten: Giro, Bargeld und Sparbuch. Die Einträge neben dem Konto dienen nur zur zusätzlichen Information, werden sonst aber nicht weiter genutzt.

Kategorie

Datenziel	
Ansicht:	*Ansicht_Kasse_Kategorie, Ansicht_Bericht_Kategorie*
Abfrage:	*Saldo_Kategorie, Saldo_Kategorie_ungesplittet*
Formular, Bericht, Makro: keine direkt	

Feldname	Feldtyp	Beschreibung
ID	Tiny Integer	Primärschlüssel der Tabelle. Der Primärschlüssel muss eindeutig sein. Die Kategorien bleiben überschaubar, so dass lediglich eine kleine Zahl ohne Auto-Wert notwendig ist.
Kategorie	Text	Wofür ist die Ausgabe erfolgt oder woher kommt die Einnahme? Eingabe erforderlich: Ja

```
001 INSERT INTO "Kategorie" VALUES ( 0 , 'ohne Zuweisung')
```

Das Feld mit der "ID" = '0' wird zum Beginn ausgefüllt. Sonst kann in den Tabellen Kasse und rel_Kasse_Kategorie nicht der entsprechende Defaultwert zugewiesen werden. Das Ausfüllen des Datensatzes mit der "ID" = '0' kann natürlich auch über die grafische Benutzeroberfläche erfolgen.

rel_Kasse_Kategorie

Datenziel	
Ansicht:	*Ansicht_Kasse_Kategorie*
Formular:	*Kasse, Kasse_Umbuchung, Konto, Konto_Salden, Konto_Salden_komplett, Konto_Salden_komplett_Umbuchung, Buchung_Umbuchung_Salden*
Makro:	*SplitRest*
Abfrage, Bericht:	keine direkt

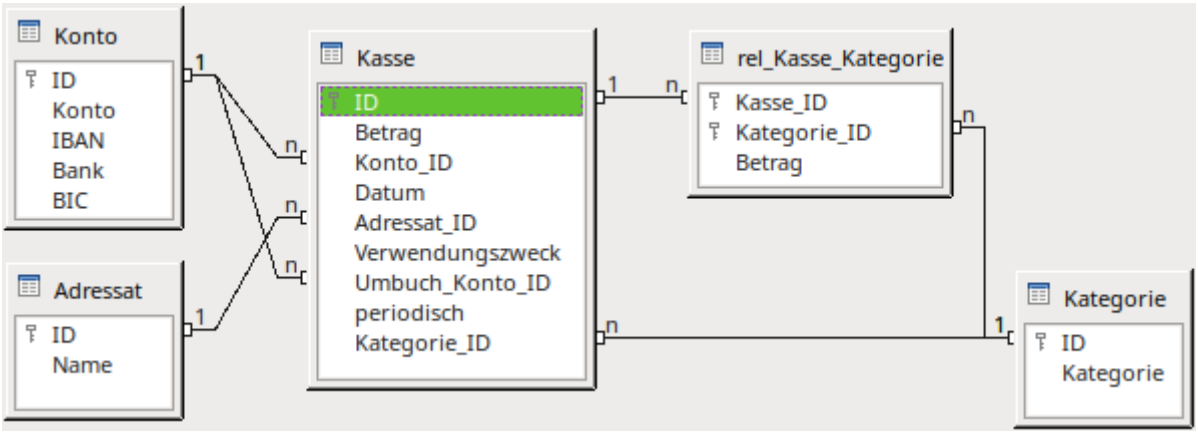
Feldname	***Feldtyp***	***Beschreibung***
Kasse_ID	Integer	Zusammen mit "Kategorie_ID" Primärschlüssel der Tabelle. Gleichzeitig Fremdschlüssel für die Verbindung zur Tabelle "Kasse".
Kategorie_ID	Tiny Integer	Zusammen mit "Kasse_ID" Primärschlüssel der Tabelle. Gleichzeitig Fremdschlüssel für die Verbindung zur Tabelle "Kategorie".
Betrag	Zahl	Der Betrag, der aus dem folgenden Konto abgebucht oder hinzu gebucht wird. Bei der Felderstellung ist auf 2 Nachkommastellen zu achten. Eingabe erforderlich: Ja

Das Feld für die Eingabe der Kategorie soll standardmäßig mit '0' beschrieben werden, wenn von dem Formular aus keine Eingabe erfolgt. Dafür muss vorher in der Tabelle *Kategorie* der entsprechende datensatz mit der "ID" = '0' eingefügt worden sein.

```
001 ALTER TABLE "rel_Kasse_Kategorie" ALTER COLUMN "Kategorie_ID"
    SET DEFAULT 0
```

Den Primärschlüsselwert '0' hat in der Tabelle "Kategorie" der Eintrag 'ohne Zuweisung'. Wird also keine Angabe im Feld "Ketegorie_ID" gemacht, so wird der Betrag 'ohne Zuweisung' geführt.

Diese Tabellen werden unter **Extras → Beziehungen** wie folgt verknüpft:



Je nach Absicht kann entweder die Kategorie direkt in die Tabelle "Kasse" eingegeben werden oder eben über die n:m-Verbindung in die Tabelle "rel_Kasse_Kategorie". Die erste Variante ermöglicht keine Splitbuchungen beim gleichen Datensatz von "Kasse". Es müsste dann also der Inhalt des Beleges vorher in einzelne Einträge in "Kasse" gesplittet werden.

Schließlich gibt es noch die Tabelle "Filter", die allerdings nicht zu den anderen Tabelle in Beziehung steht.

Filter

Datenziel	
Ansicht:	*Ansicht_Filter*
Formular:	*Buchung_Umbuchung_Salden*
Makro:	*Aktuelles_Buchungsdatum, periodische_Buchung*
Abfrage, Bericht:	keine direkt

Feldname	Feldtyp	Beschreibung
ID	Ja/Nein	Primärschlüssel der Tabelle. Diese Tabelle nimmt nur einen Datensatz auf.
Kasse_ID	Integer	Zwischenspeicherfeld für den Wert des Primärschlüssel aus "Kasse". Dieses Feld wird für die periodische Buchung benötigt. Eingabe erforderlich: Nein
StartDatum	Datum	Zwischenspeicher für das Datum, von dem ab der Bericht erstellt werden soll. Eingabe erforderlich: Nein
EndDatum	Datum	Zwischenspeicher für das Datum, bis zu dem der Bericht erstellt werden soll Eingabe erforderlich: Nein
DatensatzDatum	Datum	Zwischenspeicher für das jeweilige Datum des Datensatzes, der im Formular gerade bearbeitet wird. Wird bei einem neuen Datensatz mit dem aktuellen Datum zur Zeit der Neueingabe versehen. Eingabe erforderlich: Nein

Diese Tabelle dient vor allem zur Eingrenzung der anzuzeigenden Datenmengen. Sie enthält nur eine Zeile, in der die aktuellen Daten eingegeben werden. Auf diese Tabelle wird vor allem über die «Ansicht_Filter» Bezug genommen. In dieser Ansicht werden Standardwerte vordefiniert, falls die Werte in der Tabelle «Filter» leer sind.

Ansichten

Ansichten arbeiten grundsätzlich schneller als Abfragen, da der Code nicht noch durch die GUI übersetzt werden muss. Bei laufend wiederkehrenden Elementen in Abfragen ist daher eine Ansicht als Quelle oft hilfreich. Ansichten können allerdings nicht, wie viele Abfragen, für die Eingabe von Daten über die GUI genutzt werden.

Auch bei Berichten sollte der Ansicht der Vorzug gegeben werden, wenn nicht gerade der Bericht mit einer Parameterabfrage gestartet werden soll. Der Report-Buildder muss für die Gruppierungsfunktion gegebenenfalls den SQL-Code interpretieren können und macht da wesentlich eher Fehler als dies z.B. bei Formularen der Fall ist. Ansichten sind für den Report-Builder nur einfache Tabellen und somit einfacher handhabbar.

Ansicht_Kasse_Kategorie

Datenquelle	
Tabelle:	*Kasse, Adressat, Konto, Kategorie, rel_Kasse_Kategorie*

Datenziel
Ansicht: *Ansicht_Kategorie_Diagramm*
Abfrage: *Kategorieverlauf, Fehler: Verweis nicht gefunden, Saldo_Kategorie*

```
001 SELECT "Kasse"."ID", "Kasse"."Betrag" AS "Gesamtbetrag",
002 COALESCE( "rel_Kasse_Kategorie"."Betrag", "Kasse"."Betrag" ) AS
    "Splitbetrag",
003 "Konto"."Konto", "Kasse"."Datum", "Adressat"."Name",
    "Kasse"."Verwendungszweck",
004 "rel_Kasse_Kategorie"."Kategorie_ID",
005 COALESCE( "Kategorie"."Kategorie",
    (SELECT "Kategorie" FROM "Kategorie" WHERE "ID" = 0)) AS "Kategorie",
006 EXTRACT( YEAR FROM "Datum" ) || '-' || RIGHT( '0' || EXTRACT( MONTH FROM
    "Datum" ), 2 ) AS "JahrMonat"
007 FROM "Kasse"
008 LEFT JOIN "Konto" ON "Kasse"."Konto_ID" = "Konto"."ID"
009 LEFT JOIN "Adressat" ON "Kasse"."Adressat_ID" = "Adressat"."ID"
010 LEFT JOIN "rel_Kasse_Kategorie" ON "Kasse"."ID" =
    "rel_Kasse_Kategorie"."Kasse_ID"
011 LEFT JOIN "Kategorie" ON "rel_Kasse_Kategorie"."Kategorie_ID" =
    "Kategorie"."ID"
012 WHERE "Kasse"."Umbuch_Konto_ID" IS NULL
013 UNION
014 SELECT "a"."ID", "a"."Betrag" AS "Gesamtbetrag",
015 "a"."Betrag" - COALESCE((SELECT SUM( "Betrag") FROM "rel_Kasse_Kategorie"
    WHERE "Kasse_ID" = "a"."ID" ),0) AS "Splitbetrag",
016 "Konto"."Konto", "a"."Datum", "Adressat"."Name", "a"."Verwendungszweck",
017 0 AS "Kategorie_ID",
018 (SELECT "Kategorie" FROM "Kategorie" WHERE "ID" = 0) AS "Kategorie",
019 EXTRACT( YEAR FROM "a"."Datum" ) || '-' || RIGHT( '0' || EXTRACT( MONTH
    FROM "a"."Datum" ), 2 ) AS "JahrMonat"
020 FROM "Kasse" AS "a"
021 LEFT JOIN "Konto" ON "a"."Konto_ID" = "Konto"."ID"
022 LEFT JOIN "Adressat" ON "a"."Adressat_ID" = "Adressat"."ID"
023 LEFT JOIN "rel_Kasse_Kategorie" ON "a"."ID" =
    "rel_Kasse_Kategorie"."Kasse_ID"
024 LEFT JOIN "Kategorie" ON "rel_Kasse_Kategorie"."Kategorie_ID" =
    "Kategorie"."ID"
025 WHERE "a"."Umbuch_Konto_ID" IS NULL AND "Splitbetrag" <> 0
```

Für FIREBIRD muss in der letzten Zeile eine Änderung vorgenommen werden. FIREBIRD versteht es nicht, wenn sich die Sortierung einer Abfrage auf einen Alias bezieht, der in dieser Abfrage steht. Der Begriff "Splitbetrag" ist FIREBIRD also nicht bekannt. Stattdessen muss der Originalcode aus Zeile 15 wieder aufgerufen werden:

```
025 WHERE "a"."Umbuch_Konto_ID" IS NULL AND ("a"."Betrag" - COALESCE((SELECT
    SUM( "Betrag") FROM "rel_Kasse_Kategorie" WHERE "Kasse_ID" =
    "a"."ID" ),0)) <> 0
```

Diese Ansicht führt alle Tabellen zusammen: "Kasse", "Konto", "Adressat", "rel_Kasse_Kategorie" und "Kategorie" (Zeile 7 bis 11). Diese Daten werden nicht vor gefiltert, so dass die Tabelle "Filter" hier nicht auftaucht. An die Tabelle "Kasse" werden alle anderen Tabellen mit einem **LEFT JOIN** angebunden, so dass auch Datensätze existieren können, für die es z.B. keinen Adressaten gibt. Es werden nur Datensätze übernommen, die keinen Eintrag in der Tabelle "Kasse" im Feld "Umbuch_Konto_ID" haben (Zeile 12). Einträge von "Kasse"."Betrag" werden durch die Umbuchung wieder aufgehoben. Der Betrag wird dem Umbuchkonto gut geschrieben. Würde wie vorgesehen mit der Umbuchung gerechnet, so würde der Verlust zwar z.B. auf dem Girokonto berücksichtigt, das Geld aber nicht vom Kassenautomaten im Portemonnaie landen.

Aus "Kasse" werden die Felder "ID" und "Betrag" direkt übernommen, wobei dem Feld "Betrag" der Alias "Gesamtbetrag" zugewiesen wird. In dem nächsten Feld wird dann der "Splitbetrag"

angegeben (Zeile 2). Gibt es keinen entsprechenden Eintrag (**COALESCE**)im Bereich der Kategorien, so wird stattdessen als "Splitbetrag" der Betrag aus der Tabelle "Kasse" übernommen.

Die Felder in Zeile 3 und 4 werden wieder direkt übernommen. In Zeile 5 wird dann zu allen Feldern eine Kategoriezuweisung vorgenommen. Existiert diese Zuweisung nicht, so wird der Inhalt genommen, der für "ID" = '0' in der Tabelle "Kategorie" steht.

Zeile 6 enthält für die monatlichen Zusammenfassung einen Eintrag, der aus der Jahreszahl un der Monatszahl zusammengesetzt wird. Auf diese Art können die Ergebnisse auch leicht monatlich gruppiert ausgelesen werden.

An diese erste SELECT-Anweisung wird über UNION (Zeile 13) eine zweite SELECT-Anweisung angehängt. Diese Anweisung enthält vom Aufbau her die gleichen Felder, soll aber die Beträge erfassen, für die bisher keine Zuweisungen bei der Kategorie erfolgt sind.

In Zeile 15 wird aus diesem Grunde der Betrag ausgerechnet, der sich ergibt, wenn von einem Datensatz aus "Kasse" sämtliche passenden Beträge aus der Tabelle "rel_Kasse_Kategorie" subtrahiert werden. Ist dieser Betrag ungleich '0' (Zeile 25), so wird der entsprechende Datensatz aufgenommen. Schließlich fehlt für einen Betrag oder Teilbetrag eine Kategoriezuweisung. Der Bezug zwischen innerer Abfrage und äußerer "Kasse"."ID" wird darüber sichergestellt, dass der Tabelle "Kasse" ein Alias "a" mitgegeben wird (Zeile 20). Dadurch wird die Abfrage in Zeile 15 zu einer korrelierenden Unterabfrage. In Zeile 17 wird diesem Fehlbetrag die "Kategorie_ID" '0' zugewiesen. In Zeile 18 wird schließlich die Bezeichnung für "ID" = '0' aus der Tabelle "Kategorie" ausgelesen.

Alle anderen Felder sind gleich den Feldern in dem ersten Abfrageteil. Durch die "ID" der Tabelle "Kasse" wird der Betrag ohne Kategorie dann richtig zugeordnet.

Ansicht_Kasse_mit_Umbuchungen

Datenquelle	
Tabelle:	*Kasse, Adressat, Konto*

Datenziel	
Ansicht:	*Ansicht_Konto_Diagramm*
Abfrage:	*Kontoverlauf, Fehler: Verweis nicht gefunden, Saldo_Konto*

```
001 SELECT "Kasse"."ID", "Kasse"."Betrag",
    "Kasse"."Konto_ID", "Konto"."Konto", "Kasse"."Datum",
    "Kasse"."Adressat_ID", "Adressat"."Name", "Kasse"."Verwendungszweck",
    EXTRACT( YEAR FROM "Kasse"."Datum" ) || '-' || RIGHT( '0' ||
    EXTRACT( MONTH FROM "Kasse"."Datum" ), 2 ) AS "JahrMonat"
002 FROM "Kasse"
003 LEFT JOIN "Konto" ON "Kasse"."Konto_ID" = "Konto"."ID"
004 LEFT JOIN "Adressat" ON "Kasse"."Adressat_ID" = "Adressat"."ID"
005 UNION
006 SELECT "Kasse"."ID", "Kasse"."Betrag" * -1 AS "Betrag",
    "Kasse"."Umbuch_Konto_ID", "Konto"."Konto", "Kasse"."Datum",
    "Kasse"."Adressat_ID", "Adressat"."Name", "Kasse"."Verwendungszweck",
    EXTRACT( YEAR FROM "Kasse"."Datum" ) || '-' || RIGHT( '0' ||
    EXTRACT( MONTH FROM "Kasse"."Datum" ), 2 ) AS "JahrMonat"
007 FROM "Kasse"
008 LEFT JOIN "Konto" ON "Kasse"."Umbuch_Konto_ID" = "Konto"."ID"
009 LEFT JOIN "Adressat" ON "Kasse"."Adressat_ID" = "Adressat"."ID"
010 WHERE NOT "Kasse"."Umbuch_Konto_ID" IS NULL
```

Diese Ansicht dient als Vorstufe dazu, eine monatliche Übersicht über den Kassenstand zu erhalten. Wieder handelt es sich um zwei Teilabfragen, die durch **UNION** miteinander verbunden sind. Solche Abfragen sind nur in direktem SQL möglich. Zusammen mit einer Ansicht macht

das aber keinen wahrnehmbaren Unterschied, da ja die Dateninhalte in Ansichten sowieso schreibgeschützt sind.

Die obere Abfrage ist ein Zusammenschluss von den Tabellen "Kasse", "Konto" und "Adressat" (Zeile 2 bis 4). Es werden durch **LEFT JOIN** alle Datensätze aus "Kasse" angezeigt, auch wenn dafür kein "Adressat" existiert. Lediglich das letzte Feld in der Abfrage wird nicht direkt ausgelesen sondern als ein Zusammenhang von Jahr und Monat des jeweiligen Datums erstellt. Das Jahr wird über **EXTRACT(YEAR FROM ...)** ausgelesen. Beim Monat ist hier zusätzlich notwendig, die führende Null zu ergänzen, da sonst eine Sortierung nach "JahrMonat" durcheinander kommen würde. Die Formulierung mit **EXTRACT** ist hier gegenüber Kurzformen der HSQLDB gewählt, weil sie auch in FIREBIRD funktioniert.

Die untere Abfrage enthält wieder die gleichen Felder. Nur werden hier allein die Datensätze ausgelesen, bei denen der Eintrag im Feld "Umbuch_Konto_ID" nicht leer ist (Zeile 10). Diese Konten erhalten ja schließlich einen entsprechenden Betrag aus den abgebenden Konten, der aber sonst nirgendwo auftaucht. Entsprechend wird auch die "Umbuch_Konto_ID" aus der Tabelle "Kasse" mit der "ID" aus "Konto" verbunden, so dass die entsprechende Kontenbezeichnung auslesbar wird (Zeile 8). Einziger weiterer Unterschied zur oberen Abfrage ist, dass in Zeile 6 nicht einfach der "Betrag" übernommen wird, sondern der "Betrag" mit '-1' multipliziert wird. Was dem bei dem einen Konto ein Minus ist, ist dadurch bei dem anderen Konto ein Plus. Für die Umbuchung selbst ist es also egal, ob der Nutzer die Umbuchung auf dem Konto aufschreibt, das den Betrag abgibt oder auf dem Konto aufschreibt, das den Betrag erhält.

Ansicht_Filter

Datenquelle	
Tabelle:	*Filter*

Datenziel	
Ansicht:	*Ansicht_Kategorie_Diagramm, Ansicht_Konto_Diagramm, Ansicht_Bericht_Kategorie*
Formular:	*Konto_Salden_komplett_eM, Konto_Salden_komplett_Umbuchung_eM, Konto_Salden_komplett, Konto_Salden_komplett_Umbuchung, Buchung_Umbuchung_Salden*
Abfrage:	*Kategorieverlauf, Kontoverlauf, Fehler: Verweis nicht gefunden, Fehler: Verweis nicht gefunden*

```
001 SELECT "Filter"."ID", "Filter"."Kasse_ID",
002 COALESCE("Filter"."StartDatum",CAST( EXTRACT( YEAR FROM CURRENT_DATE ) ||
    '-01-01' AS DATE )) AS "StartDatum",
003 COALESCE("Filter"."EndDatum",CAST( EXTRACT( YEAR FROM CURRENT_DATE ) || '-
    12-31' AS DATE )) AS "EndDatum",
004 "Filter"."DatensatzDatum",
005 CASE WHEN (SELECT MAX( "Datum" ) FROM "Kasse") > CURRENT_DATE THEN (SELECT
    MAX( "Datum" ) FROM "Kasse") ELSE CURRENT_DATE END AS "Datum_Max"
006 FROM "Filter"
007 WHERE "ID" = TRUE
```

Diese Ansicht gibt alle Werte aus "Filter" wieder. Sie ergänzt diese Werte um Standardwerte, die sonst in vielen Abfragen wieder vorkommen müssten. So wird in Zeile 2 das Startdatum des aktuellen Jahres hinterlegt, das alternativ zu einem vielleicht fehlenden "StartDatum" abgerufen werden kann. Dabei wird das Jahr aus dem aktuellen Datum ausgelesen und mit dem 1. Januar als Text verbunden. Um als Datum weiter genutzt werden zu können wird dieser so entstandene Text über **CAST(... AS DATE)** in ein Datum umgewandelt. In Zeile 3 wird das entsprechend mit dem Enddatum des aktuellen Jahres gemacht. In Zeile 5 wird schließlich das Datum herausgesucht, das höher ist: Entweder das aktuelle Datum oder ein Datum, zu dem eine bestimmte Buchung vorgesehen ist.

Ansicht_Kategorie_Diagramm

Datenquelle
Ansicht: *Ansicht_Kasse_Kategorie, Ansicht_Filter*

Datenziel
Formular: *Kategorie_Konto_Auswertung*

```
001 SELECT "R"."RowDescription", "C"."ColumnDescription",
002 (SELECT SUM( "Splitbetrag" ) FROM "Ansicht_Kasse_Kategorie" WHERE
    "Kategorie" = "R"."RowDescription" AND "JahrMonat" =
    "C"."ColumnDescription") AS "Data",
003 'BC' AS "Type"
004 FROM
005 (SELECT DISTINCT "Kategorie" AS "RowDescription"
    FROM "Ansicht_Kasse_Kategorie"
    WHERE "Datum" BETWEEN (SELECT "StartDatum" FROM "Ansicht_Filter") AND
    (SELECT "EndDatum" FROM "Ansicht_Filter")) AS "R",
006 (SELECT DISTINCT "JahrMonat" AS "ColumnDescription"
    FROM "Ansicht_Kasse_Kategorie"
    WHERE "Datum" BETWEEN (SELECT "StartDatum" FROM "Ansicht_Filter") AND
    (SELECT "EndDatum" FROM "Ansicht_Filter")) AS "C"
007 ORDER BY "R"."RowDescription", "C"."ColumnDescription"
```

Mit dieser Ansicht soll ein Säulendiagramm beschickt werden, das für jede vorkommende Kategorie und jeden Monat eine Säule zeichnet.

In dieser Ansicht muss jede Kategorie vertreten sein, die in dem abgefragten Zeitraum einen Eintrag hat. Diese Kategorien werden in Zeile 5 zusammengestellt und mit einem Alias "R" in der äußeren Abfrage verfügbar gemacht. Der Zeitraum wird dabei über "Ansicht_Filter" und die entsprechenden Datumswerte bestimmt.

Diese Ansicht muss für jeden Monat, der in dem Zeitraum vorkommt und zu dem irgendwie ein Eintrag existiert, Datensätze zusammenstellen. Sämtliche "JahrMonat"-Felder werden in Zeile 6 zusammengestellt, wieder vom Zeitraum her eingeschränkt und mit dem Alias "C" versehen.

Beide Unterabfragen werden mit **DISTINCT** darauf eingestellt, dass sie nur die unterschiedlichen Werte auflisten.

Die Unterabfragen werden nicht miteinander durch eine Bedingung verknüpft, so dass alle Datensätze der einen Unterabfrage mit der anderen kombiniert werden. Für jede Kategorie stehen also alle Monate zur Verfügung, sofern auch nur in einer Kategorie dazu eine Eingabe gemacht wurde. In Zeile 1 wird diese Kombination aus "Kategorie" ("RowDescription") und "JahrMonat" ("ColumnDescription") aufgelistet. Die Begriffe für die Spalten sollen nur darauf hindeuten, was in dem Diagramm dahinter steckt: Zeilenbeschriftungen und Spaltenbeschriftungen der Tabelle, die als Basis für die Daten genommen wird.

In Zeile 2 wird ermittelt, wie groß die Summe der Beträge für die aktuelle Kategorie in dem aktuellen Monat ist. Die Unterabfrage bezieht sich dabei auf die beiden Felder in Zeile 1.

Zeile 3 enthält durchgängig die Bezeichnung 'BC' als Diagrammtyp. Diese Bezeichnung ist nur dann von Bedeutung, wenn es sich um ein XY-Diagramm handelt, da dies im Makro etwas anders angesprochen werden muss.

Ansicht_Konto_Diagramm

Datenquelle
Ansicht: *Ansicht_Kasse_mit_Umbuchungen, Ansicht_Filter*

Datenziel
Formular: *Kategorie_Konto_Auswertung*

```
001 SELECT "Konto" AS "RowDescription", '' AS "ColumnDescription",
002 SUM( "Betrag" ) AS "Data",
003 'CC' AS "Type"
004 FROM "Ansicht_Kasse_mit_Umbuchungen"
005 WHERE "Datum" BETWEEN ( SELECT "StartDatum" FROM "Ansicht_Filter" ) AND
    ( SELECT "EndDatum" FROM "Ansicht_Filter" )
006 GROUP BY "Konto"
007 ORDER BY "Konto" ASC
```

Das Diagramm zur Darstellung des Kontobestandes soll nur eine Übersicht im Kreisdiagramm sein. Dafür ist keine Ermittlung der Monatsbeträge wie beim Säulendiagramm für die Kategorien notwendig. Spaltenbeschriftungen gibt es beim Kreisdiagramm nicht. Der Punkt bleibt also leer: `''`, Zeile `1`. Vom Betrag wird für jedes Konto (siehe **`GROUP BY`** - Anweisung Zeile `6`) die Summe des Feldes "Betrag" erstellt (Zeile `2`). Der Type des Diagramms wird mit 'CC' gekennzeichnet.

Ansicht_Bericht_Kategorie

Datenquelle
Tabelle: *Kasse, Kategorie*
Ansicht: *Ansicht_Filter*

Datenziel
Bericht: *Bericht_Kategorie*

```
001 SELECT "Kasse"."ID", "Kasse"."Betrag", "Kasse"."Datum" AS "Datum",
    "Kasse"."Verwendungszweck",
002 COALESCE("Kategorie"."Kategorie", (SELECT "Kategorie" FROM "Kategorie"
    WHERE "ID" = 0)) AS "Kategorie",
003 ( SELECT "StartDatum" FROM "Ansicht_Filter" ) AS "StartDatum",
004 ( SELECT "EndDatum" FROM "Ansicht_Filter" ) AS "EndDatum"
005 FROM "Kasse"
006 LEFT JOIN "Kategorie" ON "Kasse"."Kategorie_ID" = "Kategorie"."ID"
007 WHERE "Kasse"."Umbuch_Konto_ID" IS NULL
008 AND "Kasse"."Datum" BETWEEN ( SELECT "StartDatum" FROM "Ansicht_Filter" )
    AND ( SELECT "EndDatum" FROM "Ansicht_Filter" )
009 ORDER BY "Datum" ASC
```

Diese Ansicht wird für die Erstellung des einen Berichtes benötigt. Grundsätzlich ist es besser, Ansichten für Berichte zu nutzen. Ansichten kann der Report-Builder wie Tabellen nutzen. Er braucht nicht noch zusätzlich den SQL-Code zu interpretieren.

Die Abfrage stellt alle Datensätze zu den Kategorien zusammen, die direkt mit der Tabelle "Kasse" über den Fremdschlüssel "Kategorie_ID" verknüpft sind (Zeile `6`). Sie ist also nicht für die Funktion der Splitbuchung ausgelegt. Es werden nur die Datensätze übernommen, die innerhalb der Zeitspanne liegen, die in "Ansicht_Filter" durch "StartDatum" und "EndDatum" festgelegt sind (Zeile `8`).

Da die Tabelle "Kasse" über einen **`LEFT JOIN`** mit der Tabelle "Kategorie" verbunden ist, werden alle Datensätze aus "Kasse" angezeigt. Es kann also sein, dass "Kategorie_ID" leer ist und somit keine "Kategorie" für einen Eintrag existiert. Hier wird in Zeile `2` dann statt des leeren Feldes der Wert für "ID" = '0' übernommen: 'ohne Zuweisung'.

Abfragen

Kategorieverlauf

Datenquelle
Ansicht: *Ansicht_Kasse_Kategorie, Ansicht_Filter*

Datenziel
Formular: *Kategorie_Konto_Auswertung*

```
001 SELECT "a"."Kategorie", "a"."Datum", "a"."Splitbetrag" "Betrag",
002 ( SELECT SUM( "Splitbetrag" ) FROM "Ansicht_Kasse_Kategorie" WHERE
    "Kategorie" = "a"."Kategorie" AND ( "Datum" < "a"."Datum" OR ( "Datum" =
    "a"."Datum" AND "ID" <= "a"."ID" ) ) ) AS "Kategorieverlauf",
003 ( SELECT "StartDatum" FROM "Ansicht_Filter" ) AS "StartDatum",
004 ( SELECT "EndDatum" FROM "Ansicht_Filter" ) AS "EndDatum"
005 FROM "Ansicht_Kasse_Kategorie" AS "a"
006 WHERE "a"."Datum" BETWEEN "StartDatum" AND "EndDatum"
007 ORDER BY "a"."Kategorie" ASC, "a"."Datum" ASC, "a"."ID" ASC
```

Hier wird eine Übersicht über sämtliche Buchungen sortiert nach der "Kategorie", dem "Datum" und der "ID"erstellt (Zeile 7). Die "ID" ist hier wichtig, damit eine eindeutige Sortierung bei gleichem Datum erfolgen kann.

In der Spalte "Kategorieverlauf" wird der Betrag für die jeweilige Kategorie abhängig vom "Datum" und der "ID"aufgelistet (Zeile 2). Alle "Splitbeträge" zu der aktuellen "Kategorie" werden aufgelistet, deren "Datum" kleiner ist als das Datum des aktuellen Datensatzes: **"Datum" < "a"."Datum"**. Ist das Datum gleich dem Datum des aktuellen Datensatzes, so muss die "ID" kleiner oder gleich dem aktuellen Datensatz sein: **"Datum" = "a"."Datum" AND "ID" <= "a"."ID"**.

Es werden nur die Daten angezeigt, die in dem in der "Ansicht_Filter" beschriebenen Zeitraum zwischen "StartDatum" und "EndDatum" liegen.

Auch hier muss bei FIREBIRD nachgearbeitet werden, da FIREBIRD die Nutzung des Alias in Zeile 6 nicht versteht:

```
006 WHERE "a"."Datum" BETWEEN ( SELECT "StartDatum" FROM "Ansicht_Filter" )
    AND ( SELECT "EndDatum" FROM "Ansicht_Filter" )
```

Die Unterabfragen aus Zeile 3 und Zeile 4 müssen also erneut durchgeführt werden.

Kasse_Saldo_Konto_Kategorie

Datenquelle
Tabelle: *Kasse, Konto*

Datenziel
Formular: *Kasse_oM, Konto_oM, Konto_Salden_oM, Konto_Salden_separat_oM, Konto_Salden_komplett_eM, Kasse, Konto, Konto_Salden, Konto_Salden_komplett*

```
001 SELECT "a".*,
002 CASE WHEN "a"."Betrag" > 0 THEN "a"."Betrag" ELSE 0 END AS "Einnahme",
003 CASE WHEN "a"."Betrag" < 0 THEN "a"."Betrag" ELSE 0 END AS "Ausgabe",
004 ( SELECT "Konto" FROM "Konto" WHERE "ID" = "a"."Konto_ID" ) AS
    "Kontoname",
```

```
005 ( SELECT SUM( "Betrag" ) FROM "Kasse" WHERE "Konto_ID" = "a"."Konto_ID"
    AND "Umbuch_Konto_ID" IS NULL AND ( "Datum" < "a"."Datum" OR ( "Datum" =
    "a"."Datum" AND "ID" <= "a"."ID" ) ) ) AS "Saldo_Konto",
006 ( SELECT SUM( "Betrag" ) FROM "Kasse" WHERE "Kategorie_ID" =
    "a"."Kategorie_ID" AND "Umbuch_Konto_ID" IS NULL AND ( "Datum" <
    "a"."Datum" OR ( "Datum" = "a"."Datum" AND "ID" <= "a"."ID" ) ) ) AS
    "Saldo_Kategorie",
007 ( SELECT SUM( "Betrag" ) FROM "Kasse" WHERE "Umbuch_Konto_ID" IS NULL AND
    ("Datum" < "a"."Datum" OR ( "Datum" = "a"."Datum" AND "ID" <=
    "a"."ID" ) )) AS "Saldo_gesamt"
008 FROM "Kasse" AS "a"
009 WHERE "a"."Umbuch_Konto_ID" IS NULL
010 ORDER BY "Datum", "ID" ASC
```

Die Salden werden in dieser Abfrage in Abhängigkeit vom Konto und in Abhängigkeit von der Kategorie ermittelt. Die Ermittlung des Saldos zur Kategorie bezieht sich auf die Einträge zur Kategorie direkt in der Tabelle "Kasse". Zeile 6 ist nur dann notwendig, wenn die Kategorie nicht zur Splitbuchung genutzt wird.

Aus der Tabelle "Kasse" (Zeile 9) werden alle Datensätze ausgelesen, bei denen die "Umbuch_Konto_ID" leer ist. Umbuchungen beeinflussen die Salden. Umbuchungen müssten stattdessen separat in zwei unterschiedlichen Datensätzen erscheinen.

Alle Felder aus "Kasse" werden angezeigt. Da "Kasse" die einzige Tabelle ist, die in der Auflistung erscheint, ist damit die Abfrage editierbar. Schließlich ist auch der Primärschlüssel enthalten.

In Zeile 2 werden alle Beträge > 0 als Einnahmen aufgelistet. Sonst sind die Einnahmen 0. In Zeile 3 sind entsprechend die Ausgaben alle Beträge, die < 0 sind.

Der Kontoname wird in Zeile 4 über eine korrelierende Unterabfrage eingelesen. Zu diesem Konto wird die fortlaufende Entwicklung des Saldos in Zeile 5 ermittelt.

Auch Zeile 6 bezieht sich auf den jeweils aktuellen Datensatz, hier aber auf die Kategorie. Hier wird dann die Entwicklung des Saldos für die Kategorie in Abhängigkeit von "Datum" und "ID" ermittelt.

In Zeile 7 wird schließlich die fortlaufende Entwicklung des Saldos unabhängig von Konto und Kategorie ermittelt. Dies ist dann sozusagen das Gesamtsaldo von allem, was in der Datenbank verzeichnet ist.

Kasse_Saldo_Konto_Kategorie_Umbuch

Datenquelle
Tabelle: *Kasse, Konto*

Datenziel
Formular: *Kasse_Umbuchung*

```
001 SELECT "a".*,
002 CASE WHEN "a"."Betrag" > 0 AND "Umbuch_Konto_ID" IS NULL THEN "a"."Betrag"
    ELSE 0 END AS "Einnahme",
003 CASE WHEN "a"."Betrag" < 0 AND "Umbuch_Konto_ID" IS NULL THEN "a"."Betrag"
    ELSE 0 END AS "Ausgabe",
004 ( SELECT "Konto" FROM "Konto" WHERE "ID" = "a"."Konto_ID" ) AS
    "Kontoname",
005 ( SELECT SUM( "Betrag" ) FROM "Kasse" WHERE "Konto_ID" = "a"."Konto_ID"
    AND ( "Datum" < "a"."Datum" OR ( "Datum" = "a"."Datum" AND "ID" <=
    "a"."ID" ) ) ) -
    COALESCE( ( SELECT SUM( "Betrag" ) FROM "Kasse" WHERE "Umbuch_Konto_ID" =
```

```
     "a"."Konto_ID" AND ( "Datum" < "a"."Datum" OR ( "Datum" = "a"."Datum" AND
     "ID" <= "a"."ID" ) ) ), 0 ) AS "Saldo_Konto",
006  ( SELECT SUM( "Betrag" ) FROM "Kasse" WHERE "Kategorie_ID" =
     "a"."Kategorie_ID" AND "Umbuch_Konto_ID" IS NULL AND ( "Datum" <
     "a"."Datum" OR ( "Datum" = "a"."Datum" AND "ID" <= "a"."ID" ) ) ) AS
     "Saldo_Kategorie",
007  ( SELECT "Konto" FROM "Konto" WHERE "ID" = "a"."Umbuch_Konto_ID" ) AS
     "Umbuch_Kontoname",
008  COALESCE( ( SELECT SUM( "Betrag" ) FROM "Kasse" WHERE "Konto_ID" =
     "a"."Umbuch_Konto_ID" AND ( "Datum" < "a"."Datum" OR ( "Datum" =
     "a"."Datum" AND "ID" <= "a"."ID" ) ) ), 0 ) -
     COALESCE( ( SELECT SUM( "Betrag" ) FROM "Kasse" WHERE "Umbuch_Konto_ID" =
     "a"."Umbuch_Konto_ID" AND ( "Datum" < "a"."Datum" OR ( "Datum" =
     "a"."Datum" AND "ID" <= "a"."ID" ) ) ), 0 ) AS "Saldo_Umbuchkonto",
009  ( SELECT SUM( "Betrag" ) FROM "Kasse" WHERE "Umbuch_Konto_ID" IS NULL AND
     ( "Datum" < "a"."Datum" OR ( "Datum" = "a"."Datum" AND "ID" <=
     "a"."ID" ) ) ) AS "Saldo_gesamt"
010  FROM "Kasse" AS "a"
011  ORDER BY "Datum", "ID" ASC
```

Diese Abfrage berücksichtigt im Gegensatz zu *Kasse_Saldo_Konto_Kategorie* auch die Umbuchungen. In Zeile 2 und 3 werden allerdings bei Umbuchungseinträgen die Beträge auf '0' gesetzt.

Das Saldo des Kontos in Zeile 5 muss jetzt berücksichtigen, dass die Nennung des Fremdschlüssels "Umbuch_Konto_ID" dazu führt, dass das Saldo des jeweiligen Kontos um das Gegenteil des Betrages in der Spalte "Betrag" vermindert wird. Wird z.B. vom Girokonto -200,- € auf das Bargeldkonto umgebucht, so muss beim Bargeldkonto - -200,- € gerechnet werden. Dieser Rechenschritt erfolgt im zweiten Teil der Rechnung von Zeile 5.

Die Kategorie wird durch die Umbuchung nicht beeinflusst. Daher wird in Zeile 6 nur die Summierung in Abhängigkeit von "Kategorie_ID", "Datum" und "ID" des aktuellen Datensatzes vorgenommen.

In Zeile 7 taucht dann die Bezeichnung für das Umbuchkonto auf. Zu diesem Konto wird jetzt in Zeile 8 auf die gleiche Art wie in Zeile 5 berechnet, wie hoch der aktuelle Kontostand ist. Das Saldo des Betrages, bei dem die "Umbuch_Konto_ID" mit der "Konto_ID" übereinstimmt, wird aufsummiert. Anschließend wird das Saldo des Betrages in Abhängigkeit von der "Umbuch_Konto_ID" wird ermittelt. Der Betrag ist natürlich genau entgegengesetzt dem Betrag, der dem Konto gutgeschrieben werden muss. Also muss diese Summe von der vorherigen abgezogen werden.

Kontoverlauf

Datenquelle
Ansicht: *Ansicht_Kasse_mit_Umbuchungen, Ansicht_Filter*

Datenziel
Formular: *Kategorie_Konto_Auswertung*

```
001  SELECT "a"."Konto", "a"."Datum", "a"."Betrag",
002  ( SELECT SUM( "Betrag" ) FROM "Ansicht_Kasse_mit_Umbuchungen" WHERE
     "Konto" = "a"."Konto" AND ( "Datum" < "a"."Datum" OR ( "Datum" =
     "a"."Datum" AND "ID" <= "a"."ID" ) ) ) AS "Kontoverlauf",
003  ( SELECT "StartDatum" FROM "Ansicht_Filter" ) AS "StartDatum",
004  ( SELECT "EndDatum" FROM "Ansicht_Filter" ) AS "EndDatum"
005  FROM "Ansicht_Kasse_mit_Umbuchungen" AS "a"
006  WHERE "a"."Datum" BETWEEN "StartDatum" AND "EndDatum"
007  ORDER BY "a"."Konto" ASC, "a"."Datum" ASC, "a"."ID" ASC
```

Hier wird der Verlauf der Einnahmen und Ausgaben sortiert nach "Konto", "Datum" und "ID" (Zeile 7) mit laufender Summierung dargestellt. Dabei werden nur Datensätze berücksichtigt, die zwischen dem in "Ansicht_Filter" stehenden "StartDatum" und "EndDatum" liegen (Zeile 3 und 4 sowie 6).

Auch hier muss wieder bei FIREBIRD nachgearbeitet werden, da FIREBIRD die Nutzung des Alias in Zeile 6 nicht versteht:

```
008 WHERE "a"."Datum" BETWEEN ( SELECT "StartDatum" FROM "Ansicht_Filter" ) AND
    ( SELECT "EndDatum" FROM "Ansicht_Filter" )
```

Die Unterabfragen aus Zeile 3 und Zeile 4 müssen also erneut durchgeführt werden.

Die fortlaufende Summierung des Betrages erfolgt wieder über eine korrelierende Unterabfrage (Zeile 2). Die Summe wird aus den Beträgen ermittelt, die zu dem Konto des aktuellen Datensatzes "a"."Konto" passen. Sie müssen außerdem vom Datum her vor dem Datum des aktuellen Datensatzes "a"."Datum" liegen oder, bei gleichem Datum, eine kleinere oder gleiche "a"."ID" haben. Der Bezug zur "ID" ist wichtig, weil nur so die Datensätze sicher unterschieden werden können und bei mehreren Buchungen auf einem Konto sonst hinter jeder Buchung bereits das Resultat für den ganzen Tag erscheinen würde.

Für die Anzeige solcher Abfragen sollte natürlich die Sortierung der Abfrage mit der Sortierung aus der Berechnung übereinstimmen. Erst dann wird der fortlaufende Charakter sichtbar.

Saldo_Kategorie

Datenquelle	
Tabelle:	*Kasse, Kategorie*
Ansicht:	*Ansicht_Kasse_Kategorie*

Datenziel	
Formular:	*Konto_Salden, Konto_Salden_komplett, Konto_Salden_komplett_Umbuchung, Buchung_Umbuchung_Salden*

```
001 SELECT "a"."ID", "a"."Kategorie",
002 CASE WHEN "a"."ID" = 0 THEN
    ( SELECT SUM( "Betrag" ) FROM "Kasse" WHERE "Umbuch_Konto_ID" IS NULL AND
    "Datum" <= :qDatum ) - COALESCE ( ( SELECT SUM( "Splitbetrag" ) FROM
    "Ansicht_Kasse_Kategorie" WHERE "Kategorie_ID" > 0 AND "Datum"
    <= :qDatum ), 0 )
    ELSE
    ( SELECT SUM( "Splitbetrag" ) FROM "Ansicht_Kasse_Kategorie" WHERE
    "Kategorie_ID" = "a"."ID" AND "Datum" <= :qDatum )
    END AS "Saldo_Kategorie",
003 ( SELECT SUM( "Betrag" ) FROM "Kasse" WHERE "Umbuch_Konto_ID" IS NULL AND
    "Datum" <= :qDatum ) AS "Saldo_gesamt"
004 FROM "Kategorie" AS "a"
005 WHERE "a"."ID" IN ( SELECT DISTINCT "Kategorie_ID" FROM "Kasse" WHERE
    "Datum" <= :qDatum )
006 OR "a"."ID" = 0
007 ORDER BY "a"."Kategorie" ASC
```

Diese Abfrage stellt die Salden in der jeweiligen Kategorie bis zu einem über den Parameter **:qDatum** festgelegten Datum dar. Daneben wird noch das Gesamtsaldo aller Kategorien eingeblendet. Die Abfrage wird auf die Kategorien eingeschränkt, die in dem besagten Zeitrahmen auch über einen Eintrag verfügen (Zeile 5). Bei FIREBIRD muss statt **:qDatum** in dieser Abfrage jedes Mal der Parameter in den Typ **VARCHAR** umgewandelt werden. **:qDatum** wird dann zu **CAST(:qDatum AS VARCHAR(50))** - obwohl die Eingabe selbst ja mit einem Datum verglichen wird.

Am meisten Aufwand muss um Buchungen getrieben werden, die nicht (komplett) einer Kategorie zugewiesen wurden. Deshalb wurde in der Tabelle "Kategorie" mit "ID" = '0' die "Kategorie" = 'ohne Eintrag' erstellt. Dieser Kategorie sollen alle Beträge zugewiesen werden, die nicht in anderen Kategorien verbucht sind.

Für die Kategorie '0' wird zuerst in Zeile 2 die Summe aller Beträge von "Kasse" berechnet, die keinen Eintrag bei "Umbuch_Konto_ID" haben und kleiner oder gleich dem aktuellen Datum sind. Dieser Teil der Berechnung ist gleich dem in Zeile 3. Von dieser Summe soll die Summe der Einträge aus "Splitbetrag" in "Ansicht_Kasse_Kategorie" abgezogen werden, bei denen die "Kategorie_ID" > '0' ist. Diese Beträge haben schließlich eine eindeutige Zuweisung zu einer Kategorie. Falls hier gar keine Beträge verzeichnet sind, also der Teil dieser Unterabfrage leer ist, muss für die korrekte Berechnung stattdessen über **COALESCE** der Wert '0' zugewiesen werden. Jede Kombination eines Feldes mit einem leeren Feld, auch eine Berechnung, bleibt sonst immer leer. Statt also den Gesamtbetrag als 'ohne Eintrag' auszuweisen würde dann in dem Ergebnis nichts stehen.

Ist die "ID" der Tabelle "Kategorie" ungleich '0', dann wird nur der "Splitbetrag" aus "Ansicht_Kasse_Kategorie" zu dieser "Kategorie_ID" abgefragt.

Diese Parameterabfrage funktioniert mit dem Formular problemlos. Bei der direkten Bedienung muss aber beachtet werden, dass die Eingabe bei der HSQLDB für **qDatum** in der Form YYYY-MM-DD, also z.B. 2020-07-24, erfolgen muss. Parameter für Datumseingaben wandelt die Abfrage-GUI nicht um, sofern die Parameter nur in Unterabfragen auftauchen (*https://bugs.documentfoundation.org/show_bug.cgi?id=87737*). In FIREBIRD hingegen funktioniert die Umwandlung problemlos.

Saldo_Kategorie_ungesplittet

Datenquelle
Tabelle: *Kasse, Kategorie*

Datenziel
Formular: *Konto_Salden_oM, Konto_Salden_komplett_eM, Konto_Salden_komplett_Umbuchung_eM*

```
001 SELECT "a"."ID", "a"."Kategorie",
002 CASE WHEN "a"."ID" = 0 THEN ( SELECT SUM( "Betrag" ) FROM "Kasse" WHERE
    "Umbuch_Konto_ID" IS NULL AND "Datum" <= :qDatum ) - COALESCE ( ( SELECT
    SUM( "Betrag" ) FROM "Kasse" WHERE "Umbuch_Konto_ID" IS NULL AND
    "Kategorie_ID" > 0 AND "Datum" <= :qDatum ), 0 )
    ELSE ( SELECT SUM( "Betrag" ) FROM "Kasse" WHERE "Kategorie_ID" = "a"."ID"
    AND "Umbuch_Konto_ID" IS NULL AND "Datum" <= :qDatum )
    END AS "Saldo_Kategorie",
003 ( SELECT SUM( "Betrag" ) FROM "Kasse" WHERE "Umbuch_Konto_ID" IS NULL AND
    "Datum" <= :qDatum ) AS "Saldo_gesamt"
004 FROM "Kategorie" AS "a"
005 WHERE "a"."ID" IN ( SELECT DISTINCT "Kategorie_ID" FROM "Kasse" WHERE
    "Datum" <= :qDatum )
006 OR "a"."ID" = 0
007 ORDER BY "a"."Kategorie" ASC
```

Die Kategorien werden zusammen mit den aufsummierten Beträgen aufgelistet. Fehlt eine Zuweisung der Kategorie, so werden diese Beträge als 'ohne Zuweisung' angegeben.

Diese Abfrage ist vom Aufbau gleich der Abfrage *Saldo_Kategorie*. Allerdings wird hier die Berechnung für die innerhalb der Tabelle "Kasse" eingetragenen Kategorien vollzogen. Jeder Eintrag in "Betrag" hat hier maximal einen Eintrag in "Kategorie_ID".

Die "Kategorie_ID" '0' erhält hier wieder grundsätzlich alle Zuweisungen, zu denen keine "Kategorie_ID" größer als '0' verzeichnet sind. Es muss also nicht unbedingt ein Eintrag '0' in "Kategorie_ID" erfolgen.

Für die Parameterabfrage gilt wie bei *Saldo_Kategorie*: Parameter funktionieren bei der HSQLDB mit der SQL-Vorgabe der Datumsschreibweise, nicht aber mit der ortsüblichen Schreibweise, weil der Parameter nur in Unterabfragen vorkommt. Dafür muss bei FIREBIRD der Parameter **:qDatum** über **CAST(:qDatum AS VARCHAR(50))** in einen String umgewandelt werden.

Saldo_Konto

Datenquelle
Ansicht: *Ansicht_Kasse_mit_Umbuchungen*

Datenziel
Formular: *Konto_Salden_oM, Konto_Salden_komplett_eM, Konto_Salden_komplett_Umbuchung_eM, Konto_Salden, Konto_Salden_komplett, Konto_Salden_komplett_Umbuchung, Buchung_Umbuchung_Salden*

```
001 SELECT "Konto_ID", "Konto" AS "Kontoname", SUM( "Betrag" ) AS
    "Saldo_Konto",
    ( SELECT SUM( "Betrag" ) FROM "Ansicht_Kasse_mit_Umbuchungen" WHERE
    "Datum" <= :qDatum ) AS "Saldo_gesamt"
002 FROM "Ansicht_Kasse_mit_Umbuchungen"
003 WHERE "Datum" <= :qDatum
004 GROUP BY "Konto_ID", "Konto"
005 ORDER BY "Kontoname"
```

Die Berechnung des Saldos beim Konto ist deutlich einfacher als bei den Kategorien. Dies liegt daran, dass das Konto ein Pflichtfeld ist und damit immer genau ein Eintrag vorhanden ist.

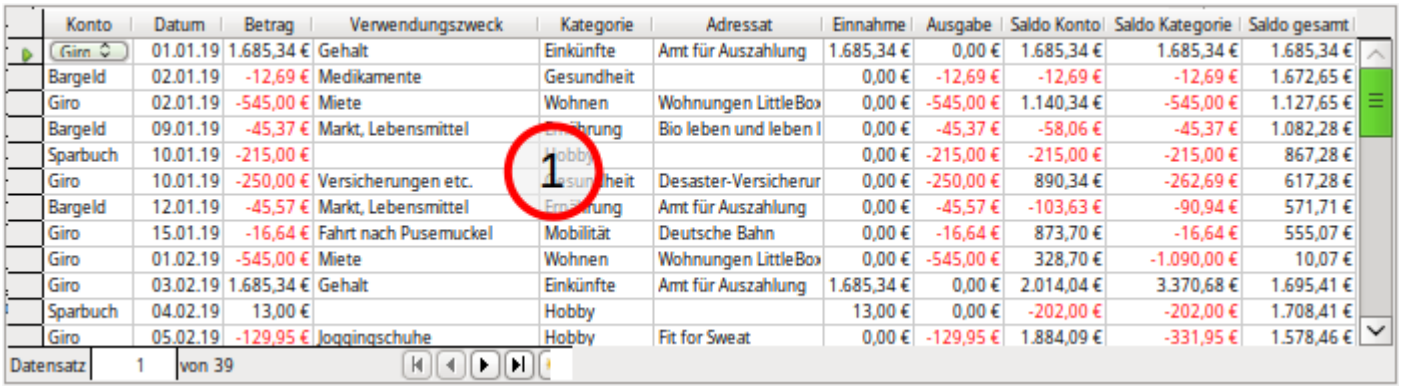
Die Abfrage ist nach den Einträgen in "Konto_ID" und "Konto" gruppiert (Zeile 4). Abhängig von dieser Gruppierung wird die Summe des Betrages berechnet (Zeile 1). Zusätzlich wird über eine Unterabfrage die Summe des Gesamtbetrages aus "Ansicht_Kasse_mit_Umbuchungen" ermittelt. Diese Abfrage ist keine korrelierende Unterabfrage, da sie ja nur ein einziges Ergebnis liefert - den Gesamtkassenstand.

Wie bei der Abfrage *Saldo_Kategorie* gilt auch hier, dass die Eingabe des Parameters in der HSQLDB bei dem direkten Start nach dem SQL-Schema für Datumsangaben erfolgen muss: YYYY-MM-DD. Dafür muss bei FIREBIRD der Parameter **:qDatum** über **CAST(:qDatum AS VARCHAR(50))** in einen String umgewandelt werden.

Formulare

Formulare ohne Makros

Kasse_oM



Konto	Datum	Betrag	Verwendungszweck	Kategorie	Adressat	Einnahme	Ausgabe	Saldo Konto	Saldo Kategorie	Saldo gesamt
Giro	01.01.19	1.685,34 €	Gehalt	Einkünfte	Amt für Auszahlung	1.685,34 €	0,00 €	1.685,34 €	1.685,34 €	1.685,34 €
Bargeld	02.01.19	-12,69 €	Medikamente	Gesundheit		0,00 €	-12,69 €	-12,69 €	-12,69 €	1.672,65 €
Giro	02.01.19	-545,00 €	Miete	Wohnen	Wohnungen LittleBo›	0,00 €	-545,00 €	1.140,34 €	-545,00 €	1.127,65 €
Bargeld	09.01.19	-45,37 €	Markt, Lebensmittel	Ernährung	Bio leben und leben l	0,00 €	-45,37 €	-58,06 €	-45,37 €	1.082,28 €
Sparbuch	10.01.19	-215,00 €		Hobby		0,00 €	-215,00 €	-215,00 €	-215,00 €	867,28 €
Giro	10.01.19	-250,00 €	Versicherungen etc.	Gesundheit	Desaster-Versicherur	0,00 €	-250,00 €	890,34 €	-262,69 €	617,28 €
Bargeld	12.01.19	-45,57 €	Markt, Lebensmittel	Ernährung	Amt für Auszahlung	0,00 €	-45,57 €	-103,63 €	-90,94 €	571,71 €
Giro	15.01.19	-16,64 €	Fahrt nach Pusemuckel	Mobilität	Deutsche Bahn	0,00 €	-16,64 €	873,70 €	-16,64 €	555,07 €
Giro	01.02.19	-545,00 €	Miete	Wohnen	Wohnungen LittleBo›	0,00 €	-545,00 €	328,70 €	-1.090,00 €	10,07 €
Giro	03.02.19	1.685,34 €	Gehalt	Einkünfte	Amt für Auszahlung	1.685,34 €	0,00 €	2.014,04 €	3.370,68 €	1.695,41 €
Sparbuch	04.02.19	13,00 €		Hobby		13,00 €	0,00 €	-202,00 €	-202,00 €	1.708,41 €
Giro	05.02.19	-129,95 €	Joggingschuhe	Hobby	Fit for Sweat	0,00 €	-129,95 €	1.884,09 €	-331,95 €	1.578,46 €

Datensatz 1 von 39

1	MainForm (Abfrage: *Kasse_Saldo_Konto_Kategorie*)

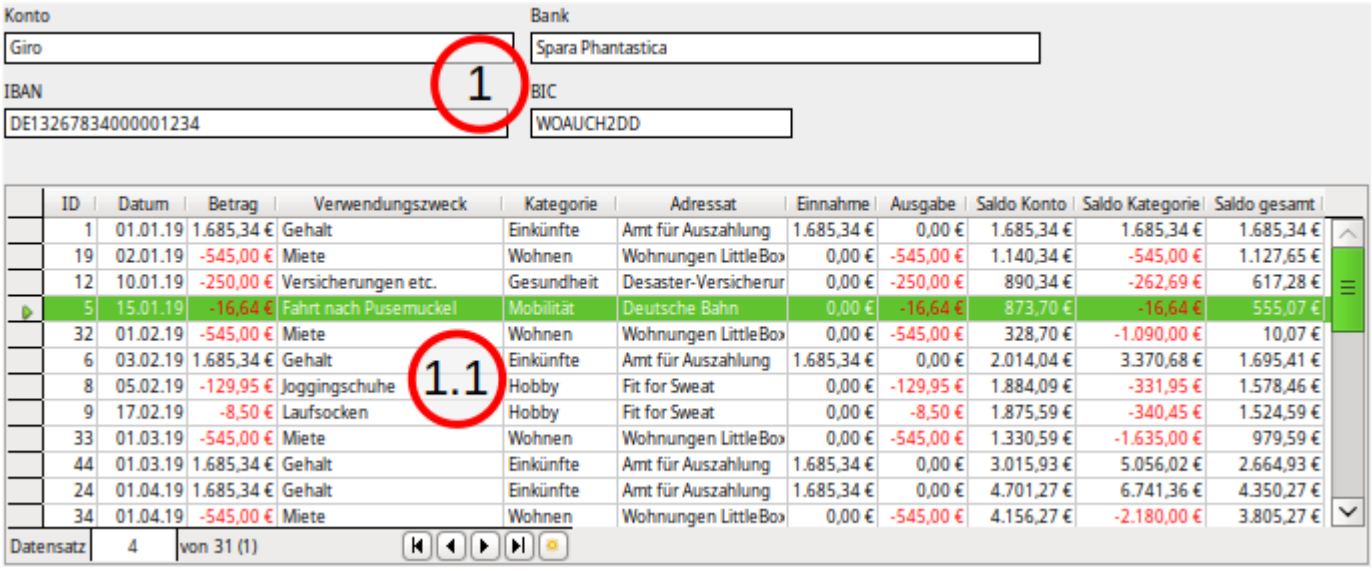
Dieses Formular stellt die einfachste Variante dar. Ein einziges Tabellenkontrollfeld, in dem alle Information der Datenquelle enthalten sind.

Eingaben in dieses Formular sind nur in den ersten 6 Spalten möglich. Die letzten 5 Spalten sind reine Berechnungen.

Die Spalten für «Konto», «Kategorie» und «Adressat» sind mit Listenfeldern versehen. Hier erfolgt eine Auswahl des Begriffes, für den dann der entsprechende Primärschlüssel als Fremdschlüssel in die Abfrage eingefügt wird. Sollen hier neue Einträge z.B. beim Adressaten erfolgen, die zur Zeit nicht auswählbar sind, so muss ein solcher Eintrag direkt in der Tabelle getätigt werden.

Die Berechnungen in den letzten Spalten erscheinen grundsätzlich erst nach der Abspeicherung des Datensatzes. Deswegen kann diese einfache Übersicht auch nur mit einem Tabellenkontrollfeld sinnvoll gelingen.

Konto_oM



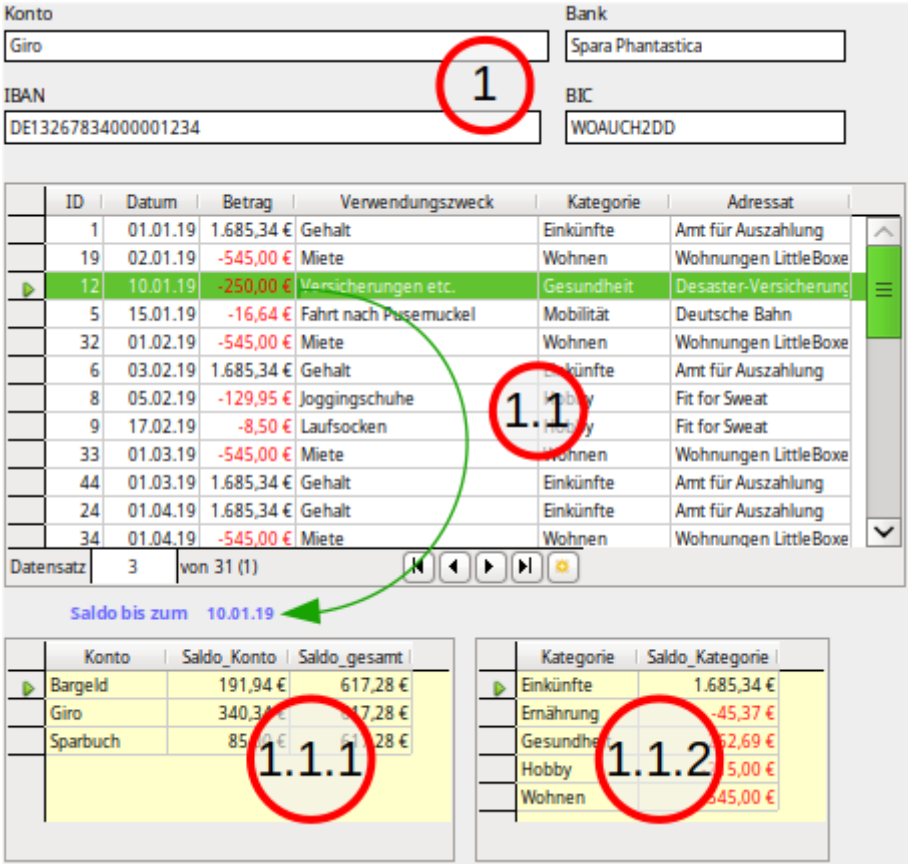
ID	Datum	Betrag	Verwendungszweck	Kategorie	Adressat	Einnahme	Ausgabe	Saldo Konto	Saldo Kategorie	Saldo gesamt
1	01.01.19	1.685,34 €	Gehalt	Einkünfte	Amt für Auszahlung	1.685,34 €	0,00 €	1.685,34 €	1.685,34 €	1.685,34 €
19	02.01.19	-545,00 €	Miete	Wohnen	Wohnungen LittleBo›	0,00 €	-545,00 €	1.140,34 €	-545,00 €	1.127,65 €
12	10.01.19	-250,00 €	Versicherungen etc.	Gesundheit	Desaster-Versicherur	0,00 €	-250,00 €	890,34 €	-262,69 €	617,28 €
5	15.01.19	-16,64 €	Fahrt nach Pusemuckel	Mobilität	Deutsche Bahn	0,00 €	-16,64 €	873,70 €	-16,64 €	555,07 €
32	01.02.19	-545,00 €	Miete	Wohnen	Wohnungen LittleBo›	0,00 €	-545,00 €	328,70 €	-1.090,00 €	10,07 €
6	03.02.19	1.685,34 €	Gehalt	Einkünfte	Amt für Auszahlung	1.685,34 €	0,00 €	2.014,04 €	3.370,68 €	1.695,41 €
8	05.02.19	-129,95 €	Joggingschuhe	Hobby	Fit for Sweat	0,00 €	-129,95 €	1.884,09 €	-331,95 €	1.578,46 €
9	17.02.19	-8,50 €	Laufsocken	Hobby	Fit for Sweat	0,00 €	-8,50 €	1.875,59 €	-340,45 €	1.524,59 €
33	01.03.19	-545,00 €	Miete	Wohnen	Wohnungen LittleBo›	0,00 €	-545,00 €	1.330,59 €	-1.635,00 €	979,59 €
44	01.03.19	1.685,34 €	Gehalt	Einkünfte	Amt für Auszahlung	1.685,34 €	0,00 €	3.015,93 €	5.056,02 €	2.664,93 €
24	01.04.19	1.685,34 €	Gehalt	Einkünfte	Amt für Auszahlung	1.685,34 €	0,00 €	4.701,27 €	6.741,36 €	4.350,27 €
34	01.04.19	-545,00 €	Miete	Wohnen	Wohnungen LittleBo›	0,00 €	-545,00 €	4.156,27 €	-2.180,00 €	3.805,27 €

Datensatz 4 von 31 (1)

1	MainForm (Tabelle: *Konto*)	1.1	SubForm (Abfrage: *Kasse_Saldo_Konto_Kategorie*)

Bei diesem Formular ist im Vergleich zu Kasse_oM lediglich ein Hauptformular hinzugekommen. Im Hauptformular wird das Konto vorausgewählt hier kann dann auch ein neues Konto hinzugefügt werden oder die alten Konten können bearbeitet werden. Das Unterformular zeigt dann nur Datensätze an, die zu dem Konto im Hauptformular passen. Die Salden stimmen weiterhin, da ja nur die anderen Konten ausgeblendet sind, die Salden aber für alle Konten zusammen nach Datum und ID sortiert berechnet werden.

Konto_Salden_oM

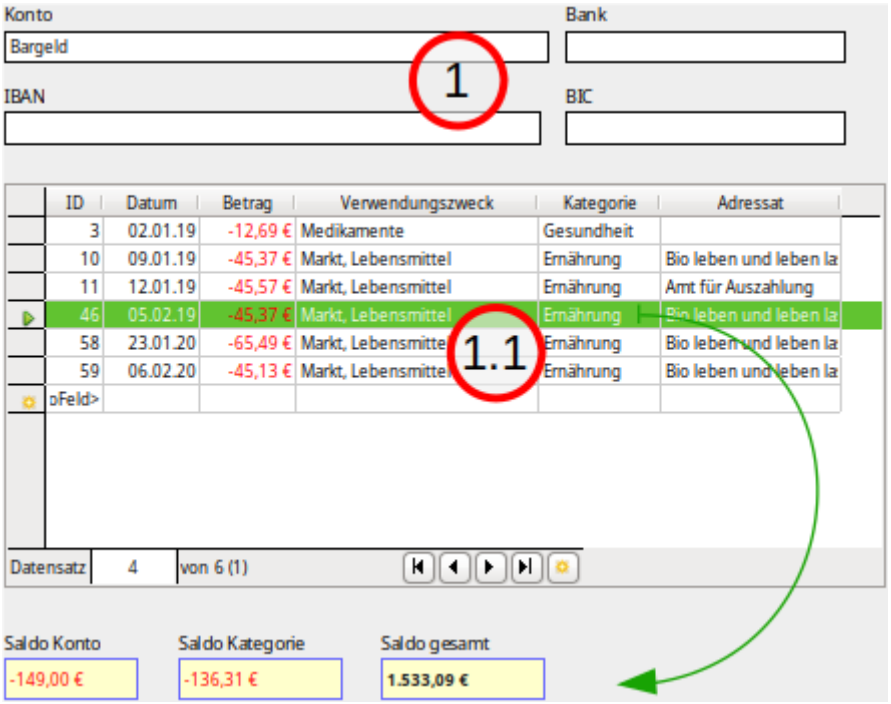


1	MainForm (Tabelle: *Konto*)	1.1	SubForm (Abfrage: *Kasse_Saldo_Konto_Kategorie*)	1.1.1	SaldoKonto (Abfrage: *Saldo_Konto*)
				1.1.2	SaldoKategorie (Abfrage: *Saldo_Kategorie_ungesplittet*)

Mit diesem Formular wird das Tabellenkontrollfeld des Unterformulars 1.1 nur noch mit Feldern bestückt, die für die Eingabe gedacht sind. Trotzdem nutzt das Formular als Datenbasis die Abfrage mit den Summierungen. Diese Abfrage schließt nämlich gleichzeitig die Einträge aus "Kasse" aus, bei denen ein Eintrag in der "Umbuch_Konto_ID" erfolgt ist. Ohne diese Art der Umbuchung in anderen Beispielformularen könnte also auch direkt die Tabelle "Kasse" genutzt werden.

Die Salden werden jetzt in Abhängigkeit vom Datum des aktuellen Datensatzes in UnterUnterformularen angezeigt. Der Screenshot weist hier z.B. vom markierten Datensatz mit dem Datum '10.01.19' auf das Saldo bis zum 10.01.19 hin. Die Salden geben also nicht den aktuellen Kontostand wieder, sondern den Kassenstand zum Ende des Buchungstages.

Konto_Salden_separat_oM

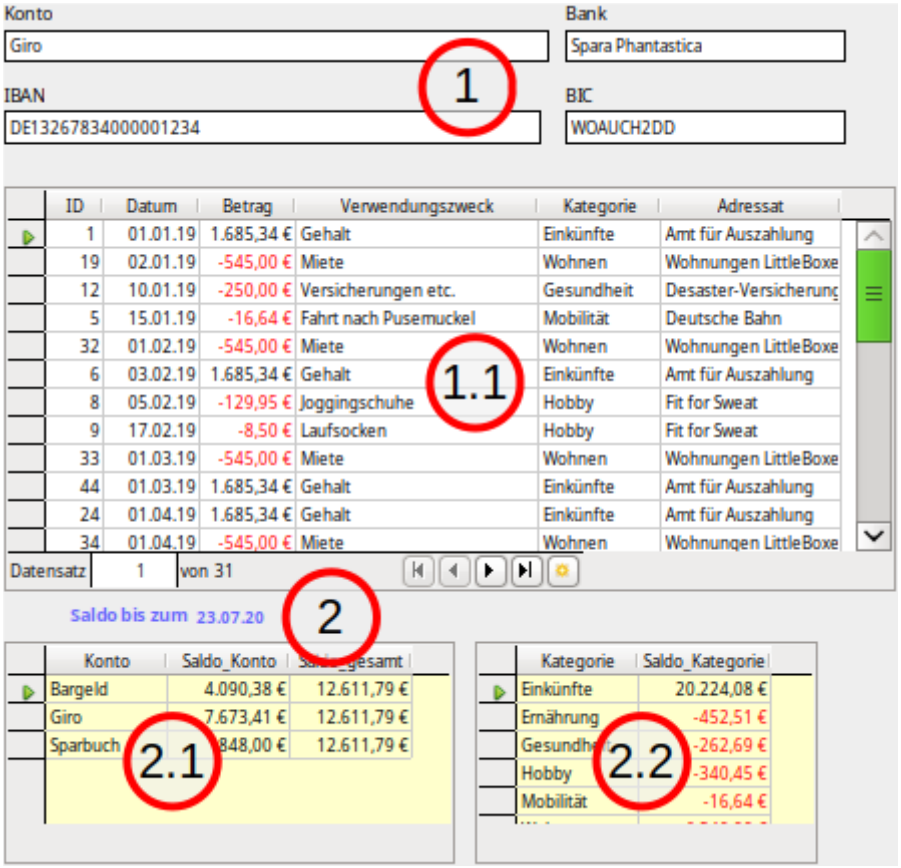


1	MainForm (Tabelle: *Konto*)	**1.1**	SubForm (Abfrage: *Kasse_Saldo_Konto_Kategorie*)

Ist der Platz geringer als in *Konto_Salden_oM* durch die Übersicht der Salden benötigt, so reicht hier auch eine einfache Auswertung der Salden, die sich jetzt wieder auf die Werte bezieht, die direkt in der entsprechenden Zeile der Abfrage des Unterformulars stehen. Die drei Felder für die Beträge sind nur aus dem Tabellenkontrollfeld ausgelagert und als Einzelfelder unter das Tabellenkontrollfeld eingebunden worden. Einzelfelder im gleichen Formular wie das Tabellenkontrollfeld zeigen immer den Inhalt zu dem aktuellen Datensatz an, sind aber für die Navigation mit dem Tabulator innerhalb des Tabellenkontrollfeldes nicht direkt erreichbar und damit auch nicht direkt störend.

Formulare mit einfachem Aktualisierungsmakro

Konto_Salden_komplett_eM



1	MainForm (Tabelle: *Konto*)	**1.1**	SubForm (Abfrage: *Kasse_Saldo_Konto_Kategorie*)
2	MaxDatum (Ansicht: *Ansicht_Filter*)	**2.1**	SaldoKonto (Abfrage: *Saldo_Konto*)
		2.2	SaldoKategorie (Abfrage: *Saldo_Kategorie_ungesplittet*)

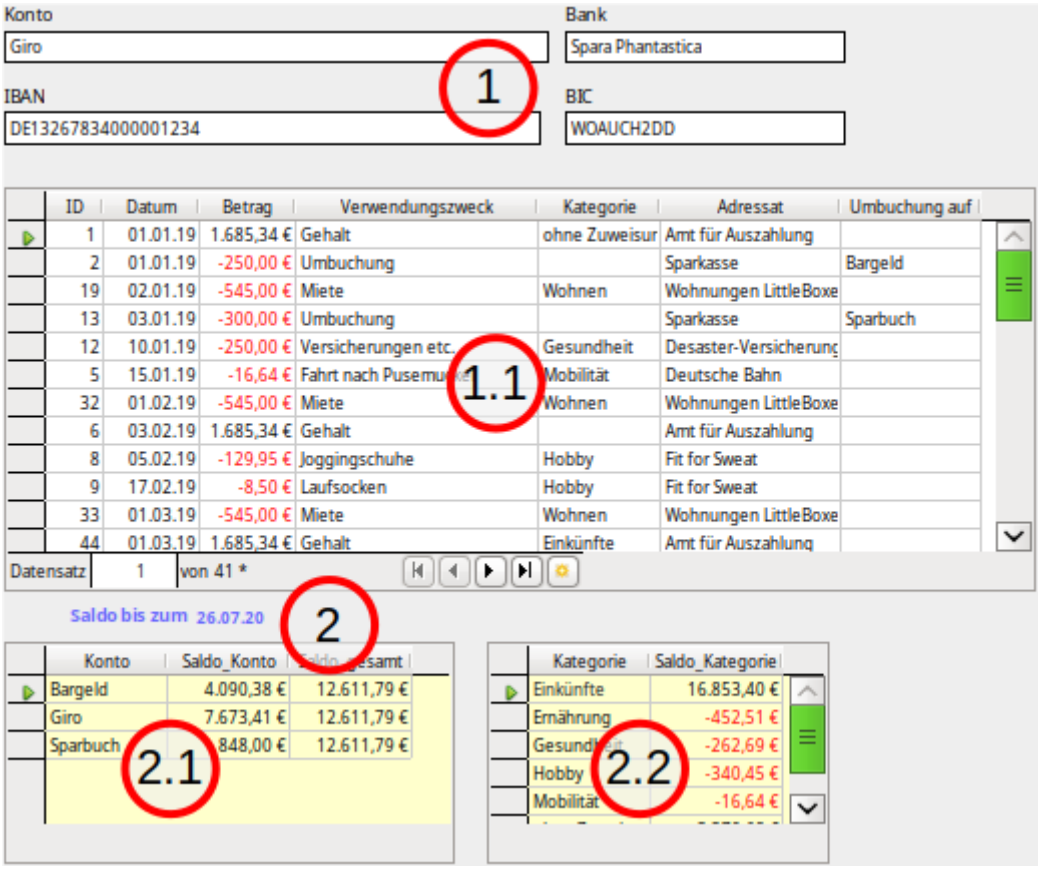
Makros in Formulareigenschaften		
1.1	Nach der Datensatzaktion	Module1.*Aktualisieren_einfach*

In diesem Formular wird die Anzeige der Salden nicht an das Eingabeformular 1.1 gekoppelt. Die Abfrage zu den Formularen 2.1 und 2.2 wird stattdessen direkt beim Öffnen der Formulars 2 gestellt. Der Inhalt dort würde also bestehen bleiben, auch wenn in dem Eingabeformular etwas geändert würde.

Vorteil dieser Konstruktion ist natürlich, dass dadurch der Kontostand unter Berücksichtigung aller eingetragenen Buchungen angegeben wird. Das höchste Datum ist hier nicht automatisch das aktuelle Datum. Wenn bereits für einen der kommenden Tage eine Buchung mit Datum eingetragen wurde, dann wird auch die noch berücksichtigt und das Datum eben dieser Buchung als maximales Datum genommen.

Damit die Auswertung nicht nur beim Öffnen des Formulars stimmt, muss sie beim Ändern von Eingaben in dem Eingabeformular nach der Abspeicherung wieder neu eingelesen werden. Dies wird hier durch ein kleines Makro erledigt, dass nach jeder Datensatzaktion kurz das Formular 2 und damit auch die Unterformulare 2.1 und 2.2 neu einliest.

Konto_Salden_komplett_Umbuchung_eM



1	MainForm (Tabelle: *Konto*)	**1.1**	SubForm (Tabelle: *Kasse*)
2	MaxDatum (Ansicht: *Ansicht_Filter*)	**2.1**	SaldoKonto (Abfrage: *Saldo_Konto*)
		2.2	SaldoKategorie (Abfrage: *Saldo_Kategorie_ungesplittet*)

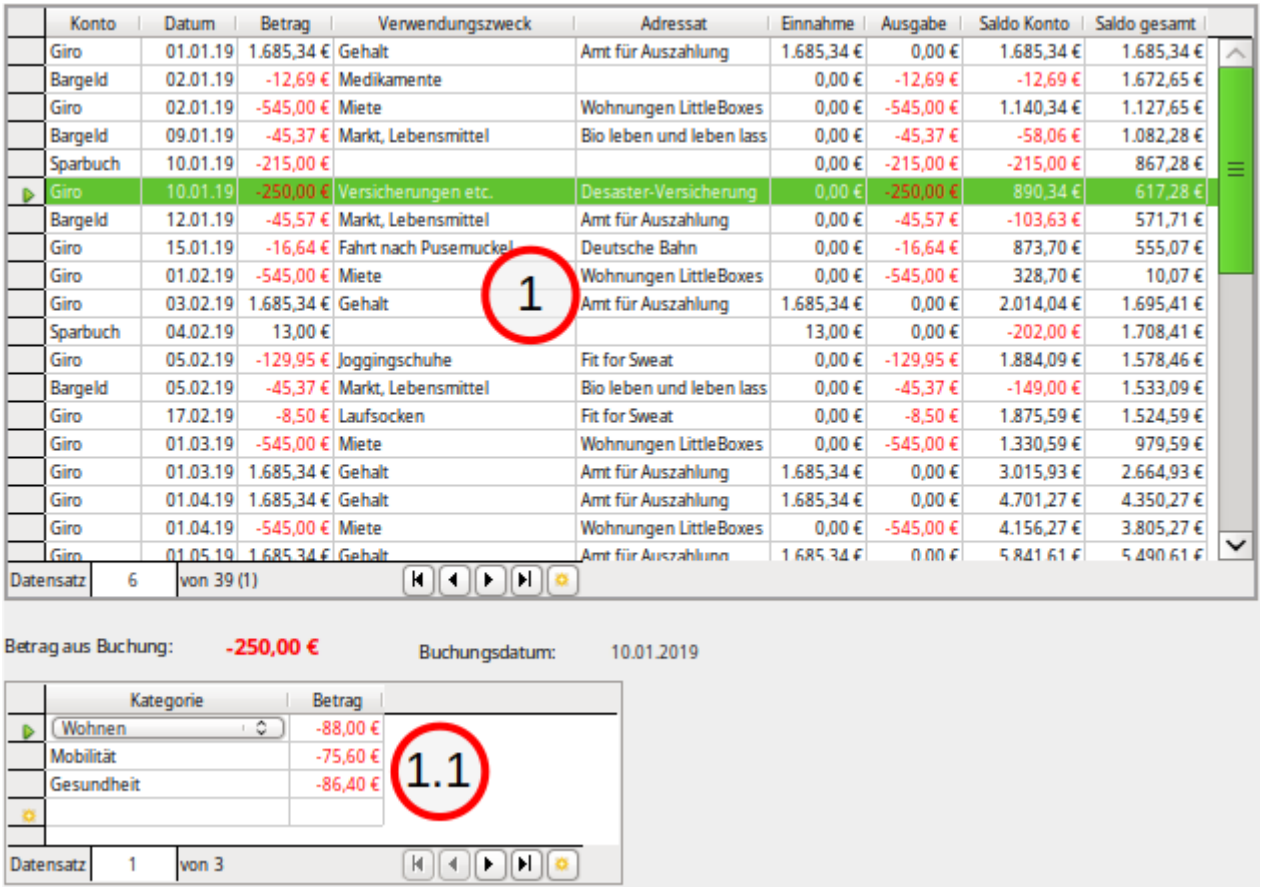
Makros in Formulareigenschaften		
1.1	Nach der Datensatzaktion	Module1.*Aktualisieren_einfach*

Gegenüber dem Formular *Konto_Salden_komplett_eM* wird hier lediglich noch die Möglichkeit hinzugefügt, auch die Umbuchungsfunktion zu nutzen. Damit kann dann auch die Tabelle "Kasse" direkt benutzt werden. Allerdings muss innerhalb der Formulareigenschaft von 1.1 noch die Sortierung so eingestellt werden, dass zuerst nach dem Datum und dann nach der ID sortiert wird. Eine solche Einstellung in den Formulareigenschaften kann gegebenenfalls auch wieder bei der Eingabe über die Navigationsleiste vorübergehend rückgängig gemacht werden.

Der Screenshot zeigt darüber hinaus, dass es zur Zeit möglich ist, dem Feld "Kategorie_ID" in der Tabelle "Kasse" gar keine Kategorie zuzuweisen. Gerade bei den Umbuchung fehlen natürlich solche Zuweisungen. Auch der Datensatz mit der "ID" = '6' hat keine Zuweisung der Kategorie. Beim Datensatz mit der "ID" = '1' ist hingegen in der Tabelle "Kasse" für die "Kategorie_ID" der Wert '0' eingetragen. Deswegen wird dort die Bezeichnung 'ohne Zuweisung' ausgegeben.

Formulare mit mehrfachem Makroeinsatz

Kasse



1	MainForm (Abfrage: *Kasse_Saldo_Konto_Kategorie*)	**1.1**	SplitKategorie (Tabelle: *rel_Kasse_Kategorie)*

Makros in Formulareigenschaften		
1	Vor der Datensatzaktion	Module1.*ZeileZwischenspeichern*
1	Nach der Datensatzaktion	Module1.*SplitRest*
1.1	Nach der Datensatzaktion	Module1.*SplitRest*

In diesem Formular sind nur die ersten 5 Felder des Tabellenkontrollelementes im Hauptformular beschreibbar. Die anderen Feldwerte werden aus der Abfrage ausgelesen.

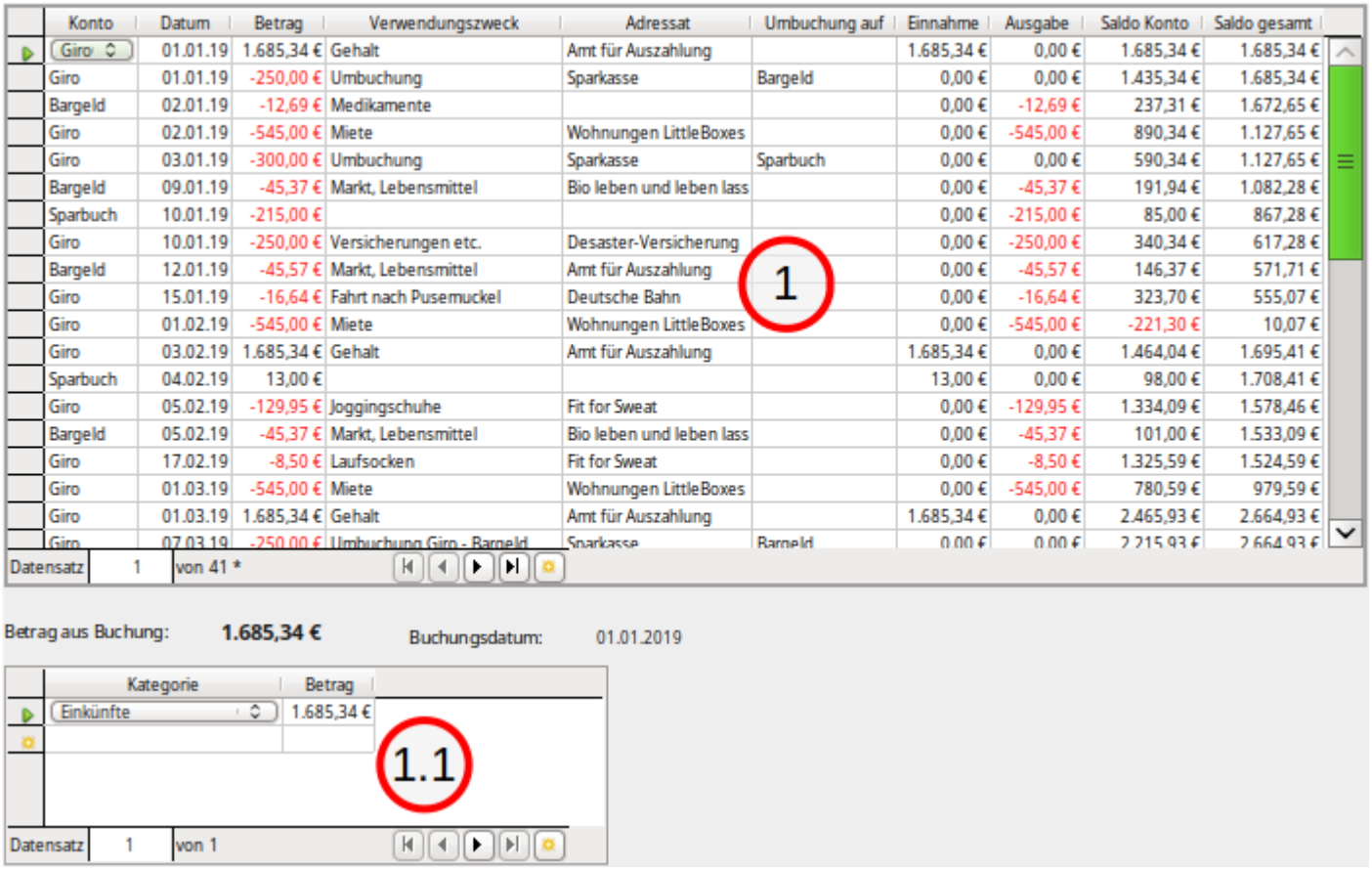
Im Unterformular wird die Kategorie für die jeweilige Buchung ausgesucht. Sollen Buchungen auf verschiedene Kategorien aufgeteilt werden, so können sie dort gesplittet werden. Dabei wird beständig der Rest ausgerechnet, der noch zum Splitten zur Verfügung steht.

Die Abfrage als Grundlage des Formulars berücksichtigt nicht die Umbuchungen, die über andere Formulare in dieser Beispieldatenbank gemacht worden sind. Für eine Umbuchung müssten grundsätzlich zwei Datensätze eingegeben werden: Die Abbuchung aus einem Konto und die Einzahlung auf dem anderen Konto. Das ist dann fehleranfälliger, weil

- der Abbuchungswert und der Einzahlungswert eventuelle unterschiedlich eingegeben werden oder
- einer der beiden Datensätze gelöscht werden kann und damit die Abbuchung oder Einzahlung fehlt.

Aus diesem Grunde wird im folgenden Formular die Umbuchung im gleichen Datensatz vorgenommen und der Buchungsbetrag für die Umbuchung automatisch geschrieben.

Kasse_Umbuchung



1	MainForm (Abfrage: *Kasse_Saldo_Konto_Kategorie_Umbuch*)	**1.1**	SplitKategorie (Tabelle: *rel_Kasse_Kategorie)*

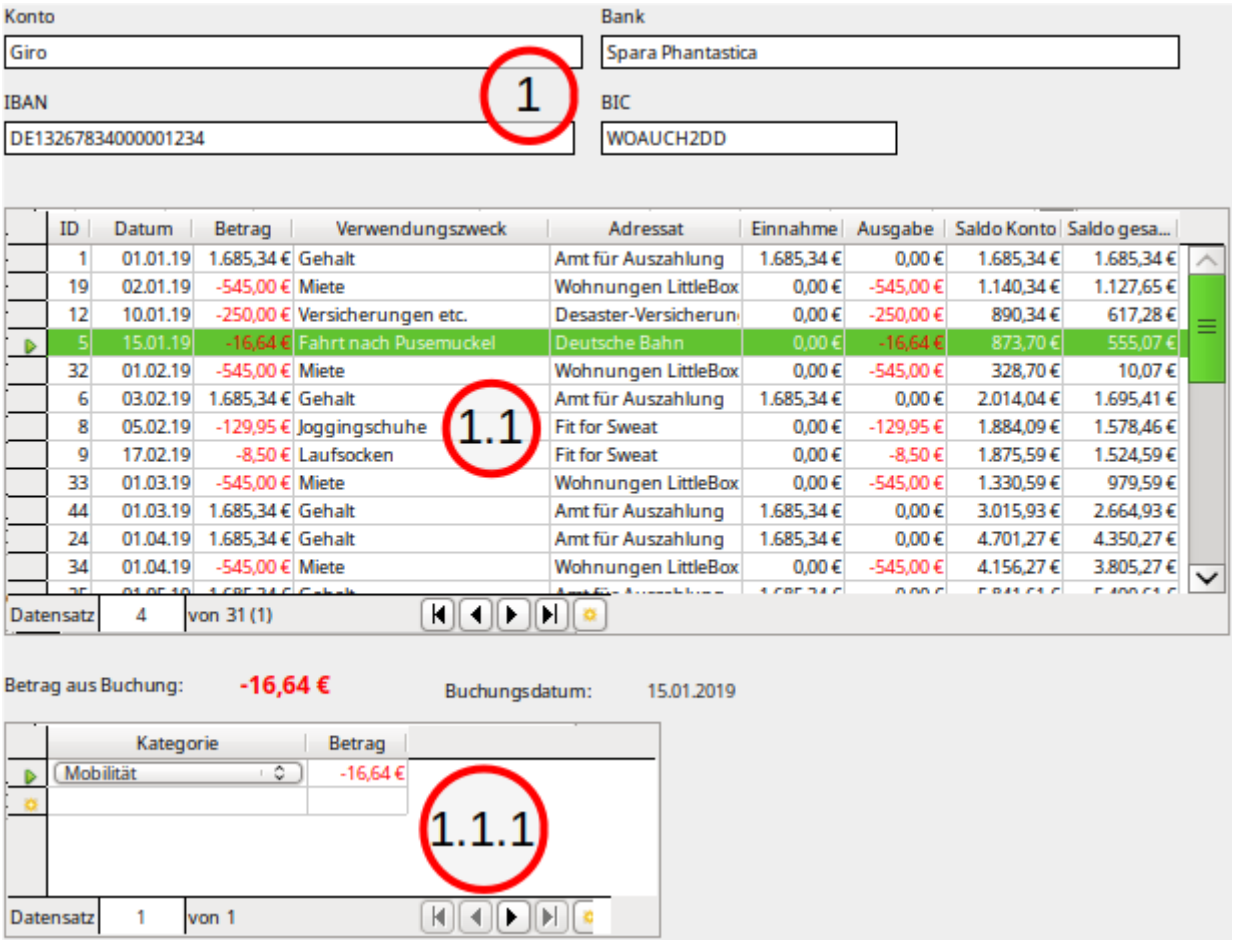
Makros in Formulareigenschaften		
1	Vor der Datensatzaktion	Module1.*ZeileZwischenspeichern*
1	Nach der Datensatzaktion	Module1.*SplitRest*
1.1	Nach der Datensatzaktion	Module1.*SplitRest*

In diesem Formular ist auch die Umbuchung erreichbar. Die entsprechend geänderte Abfrage berücksichtigt die Umbuchung bei Einnahme und Ausgabe (beide bleiben 0,00 €) und auch beim «Saldo gesamt». Lediglich das «Saldo Konto» weist diese Änderung auf, die im Feld «Betrag» gemacht wurde. Aber eben auch nur für das entsprechende Konto. Der Betrag wird nach der Umbuchung in dem anderen Konto mit dem entgegengesetzten Vorzeichen geführt.

Im Unterformular wird die Kategorie für die jeweilige Buchung ausgesucht. Sollen Buchungen auf verschiedene Kategorien aufgeteilt werden, so können sie dort gesplittet werden. Dabei wird beständig der Rest ausgerechnet, der noch zum Splitten zur Verfügung steht.

Dieses Formular erfüllt alle Möglichkeiten der Umbuchung, der Splitbuchung und des gleichzeitigen Blicks auf den Kontostand mit lediglich einem Formular und einem Unterformular.

Konto

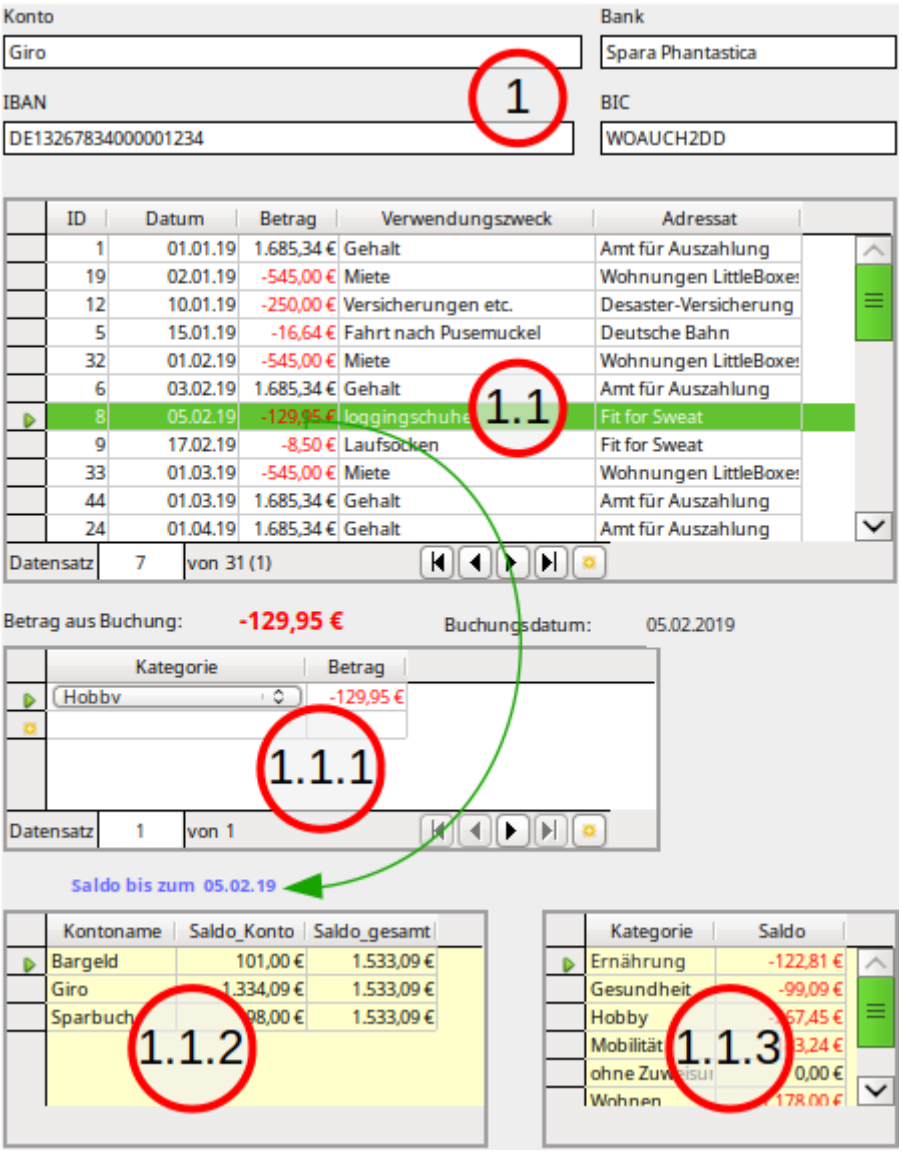


1	MainForm (Tabelle: *Konto*)	**1.1**	SubForm (Abfrage: *Kasse_Saldo_Konto_Kategorie*)	**1.1.1**	SplitKategorie (Tabelle: *rel_Kasse_Kategorie*)

Makros in Formulareigenschaften		
1.1	Vor der Datensatzaktion	Module1.*ZeileZwischenspeichern*
1.1	Nach der Datensatzaktion	Module1.*SplitRest*
1.1.1	Nach der Datensatzaktion	Module1.*SplitRest*

Die ist eine Erweiterung des Formulars *Kasse*. Aus dem Formular wurde lediglich der Eintrag für das Konto herausgenommen, so dass in dem Eingabeformular 1.1 als Unterformular nur die Datensätze vorhanden sind, die zu dem im Hauptformular liegenden Formular passen.

Konto_Salden



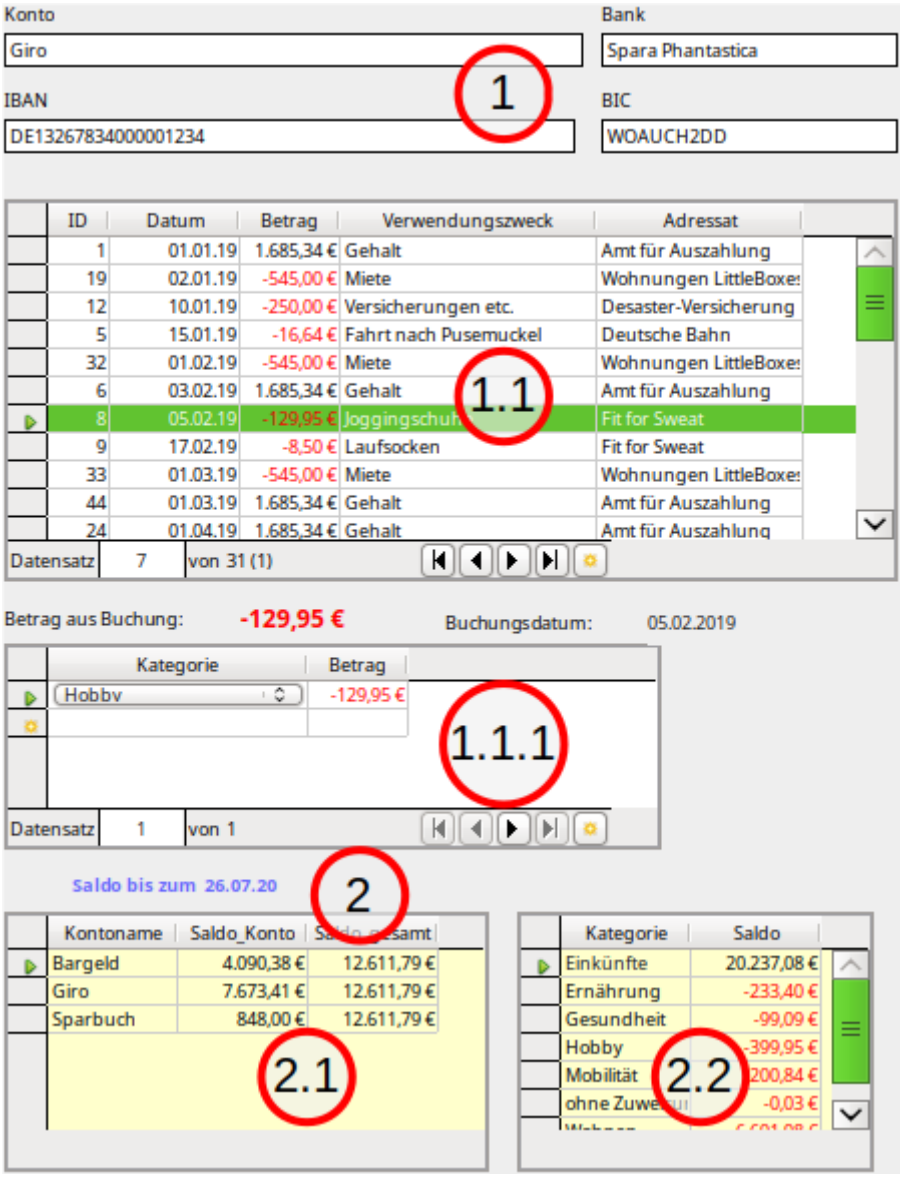
1	MainForm (Tabelle: *Konto*)	**1.1**	SubForm (Abfrage: *Kasse_Saldo_Konto_Kategorie*)	**1.1.1**	SplitKategorie (Tabelle: *rel_Kasse_Kategorie*)
				1.1.2	SaldoKonto (Abfrage: *Saldo_Konto*)
				1.1.3	SaldoKategorie (Abfrage: *Saldo_Kategorie*)

Makros in Formulareigenschaften		
1.1	Vor der Datensatzaktion	Module1.*ZeileZwischenspeichern*
1.1	Nach der Datensatzaktion	Module1.*SplitRest*
1.1.1	Nach der Datensatzaktion	Module1.*SplitRest*

Hier wird die Angabe der Salden gegenüber dem Formular *Konto* aus dem Eingabeformular herausgenommen und in separate Unterformulare ausgelagert. Die Unterformulare 1.1.2 und 1.1.3 beziehen dabei das Datum, zu dem sie die Salden berechnen sollen, aus dem Formular 1.1. Es wird also der Kontostand zum Zeitpunkt des markierten Datensatzes angegeben.

Als Datenbasis dient für 1.1 weiter eine Abfrage, weil diese Abfrage die Umbuchungen der dafür gesondert erstellten Formulare ausblendet. Umbuchungen müssten hier also, wie unter *Kasse* angegeben, als zwei Datensätze geführt werden: Abbuchung und Einzahlung.

Konto_Salden_komplett

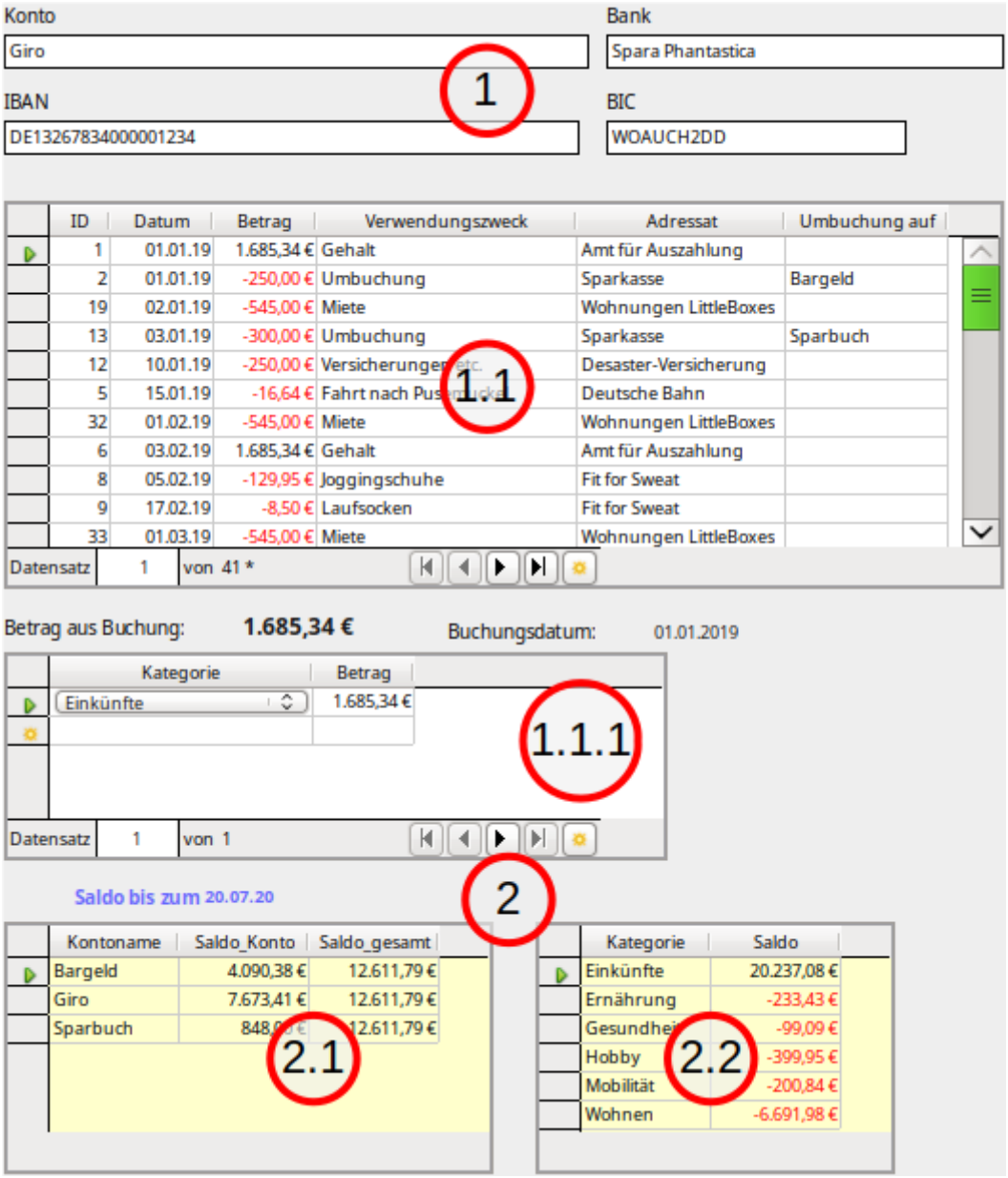


1	MainForm (Tabelle: *Konto*)	**1.1**	SubForm (Abfrage: *Kasse_Saldo_Konto_Kategorie*)	**1.1.1**	SplitKategorie (Tabelle: *rel_Kasse_Kategorie*)
2	MaxDatum (Ansicht: *Ansicht_Filter*)	**2.1**	SaldoKonto (Abfrage: *Saldo_Konto*)		
		2.2	SaldoKategorie (Abfrage: *Saldo_Kategorie*)		

Makros in Formulareigenschaften		
1.1	Vor der Datensatzaktion	Module1.*ZeileZwischenspeichern*
1.1	Nach der Datensatzaktion	Module1.*Aktualisieren*
1.1.1	Nach der Datensatzaktion	Module1.*Aktualisieren*

Hier ist gegenüber *Konto_Salden* die Ausgabe der Salden nicht von dem ausgesuchten Datensatz abhängig. Es wird immer der Kontostand angegeben, der sich aus allen in der Datenbank befindlichen Buchungen ergibt. Eine kleine Besonderheit im Screenshot rechts unten: Hier wurde einfach einmal testweise aus der Splitbuchungstabelle "rel_Kasse_Kategorie" ein Betrag um 3 Ct vermindert. Die Abfrage gibt dies jetzt als einen Betrag von 0,03 € der Kategorie 'ohne Zuweisung' aus.

Konto_Salden_komplett_Umbuchung



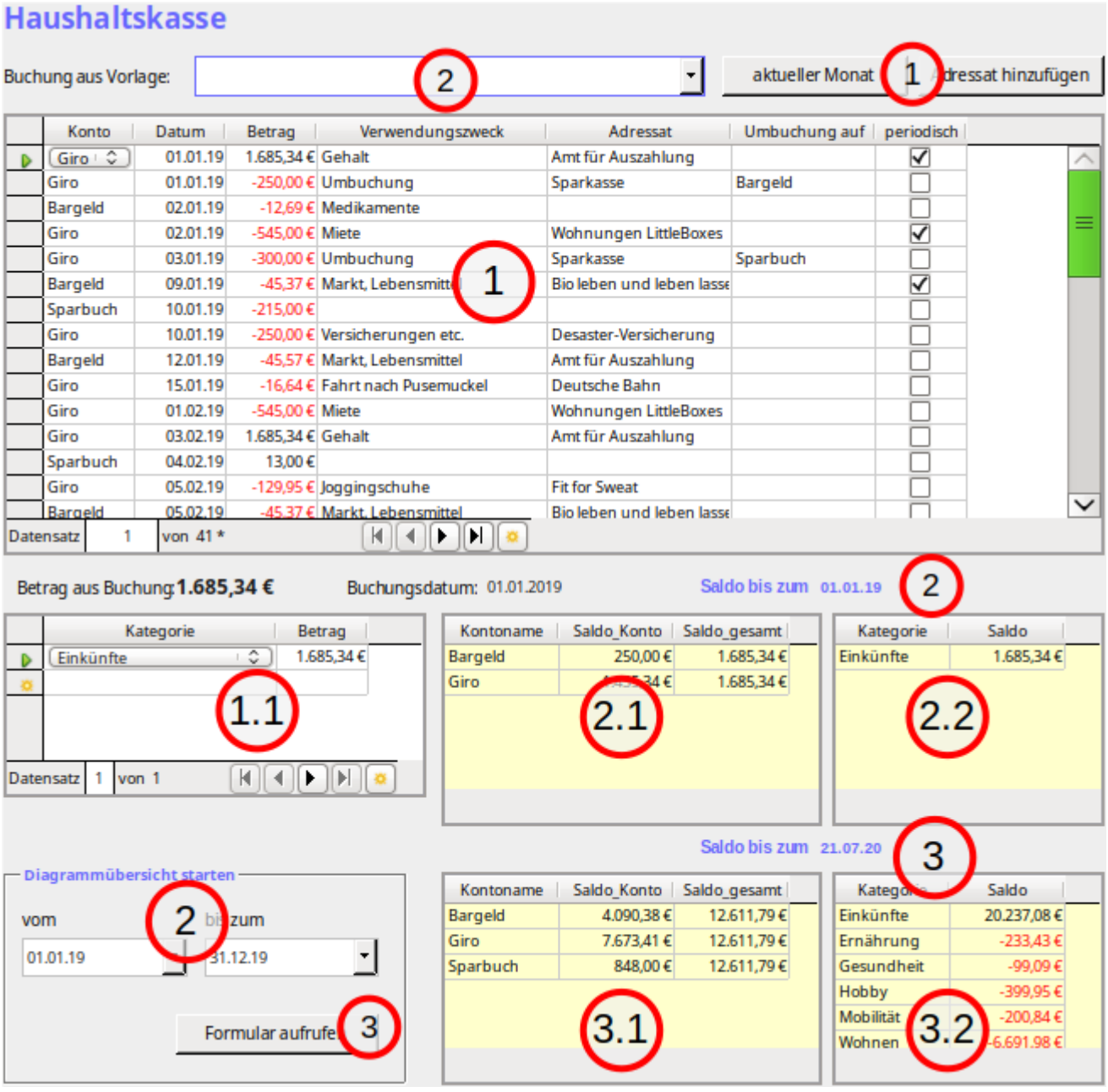
1	MainForm (Tabelle: *Konto*)	**1.1**	SubForm (Tabelle: *Kasse*)	**1.1.1**	SplitKategorie (Tabelle: *rel_Kasse_Kategorie*)
2	MaxDatum (Ansicht: *Ansicht_Filter*)	**2.1**	SaldoKonto (Abfrage: *Saldo_Konto*)		
		2.2	SaldoKategorie (Abfrage: *Saldo_Kategorie*)		

Makros in Formulareigenschaften		
1.1	Vor der Datensatzaktion	Module1.*ZeileZwischenspeichern*
1.1	Nach der Datensatzaktion	Module1.*Aktualisieren*
1.1	Nach dem Datensatzwechsel	Module1.*Kontofilter_Formstart*
1.1.1	Nach der Datensatzaktion	Module1.*Aktualisieren*

Hier kommen jetzt zusätzlich zu *Konto_Salden_komplett* die Einträge der Umbuchung ins Spiel. Deswegen kann hier jetzt auch direkt auf die Tabelle "Kasse" zugegriffen werden.

Zur Umbuchung stehen über ein Makro nur die Konten zur Verfügung, die nicht dem Konto im Hauptformular 1 entsprechen.

Buchung_Umbuchung_Salden



1	MainForm (Tabelle: *Kasse*)	**1.1**	SplitKategorie (Tabelle: *rel_Kasse_Kategorie*)
2	Filter (Tabelle: *Filter*)	**2.1**	SaldoKonto (Abfrage: *Saldo_Konto*)
		2.2	SaldoKategorie (Abfrage: *Saldo_Kategorie*)
3	MaxDatum (Ansicht: *Ansicht_Filter*)	**3.1**	SaldoKonto (Abfrage: *Saldo_Konto*)
		3.2	SaldoKategorie (Abfrage: *Saldo_Kategorie*)

Makros in Formulareigenschaften		
1	Vor der Datensatzaktion	Module1.*ZeileZwischenspeichern*
1	Nach der Datensatzaktion	Module1.*Aktualisieren*
1	Nach dem Datensatzwechsel	Module1.*Aktuelles_Buchungs-datum*
1.1	Nach der Datensatzaktion	Module1.*Aktualisieren*
Makros in Feldeigenschaften		
1	Schaltfläche Aktueller Monat (Aktion Ausführen)	Module1.*Monat_aktuell*
1	Schaltfläche Adressat hinzufügen (Aktion Ausführen)	Module1.*Adressat_neu*
1	Listenfeld «Konto» (Status geändert)	Module1.*Kontofilter_Feldstart*
2	Listenfeld «Buchung aus Vorlage» (Modifiziert)	Module1.*periodische_Buchung*
3	Schaltfläche Formularaufrufen (Aktion Ausführen)	Module1.*Formular_starten*

In diesem Formular werden schließlich alle Register gezogen. Schon allein die Zuordnung von einzelnen Schaltflächen und des Listenfeldes für die Buchung aus einer Vorlage zu verschiedenen Formularen muss intensiver erklärt werden.

Doch zuerst einmal die Funktionalität:

Wird über das Listenfeld «Buchung aus Vorlage» ein vorheriger Datensatz ausgesucht, so wird dieser Datensatz mit aktuellem Datum am Schluss der Tabelle von Formular 1 eingefügt. Das Listenfeld speichert hierfür die ID der Vorlage in der Tabelle "Filter" ab. Deswegen auch die Zuordnung zu Formular 2.

Ein Klick auf den Button Aktueller Monat führt dazu, dass die Datensätze in 1 nur für den aktuellen Monat angezeigt werden. Bei vielen Daten und einer Neueingabe muss sonst immer erst nach unten gescrollt werden.

Fehlt ein Adressat in dem Listenfeld in 1, so kann dieser über den Button Adressat hinzufügen in eine einfache Input-Box eingegeben werden. Hier ist nur der Name des Adressaten vorgesehen. Hätte die Tabelle für den Adressaten mehr Felder als nur den Namen, so würde so etwas über einen Dialog realisiert.

Die Angabe des Kontos ist jetzt wieder in das Formular integriert. In Abhängigkeit von dem Buchungskonto wird die Liste der Umbuchungskonten mit einem Makro erstellt. Das Buchungskonto kommt in der Liste der Umbuchungskonten nicht vor.

Die Eingabe in das Formular weist jetzt auch die Möglichkeit auf, bestimmte Datensätze zu periodisch wiederkehrenden Datensätzen zu erklären. Die Funktion bedeutet nicht, dass die Daten automatisch eingegeben werden. Sie erscheinen vielmehr in der Auswahl des Listenfeldes «Buchung aus Vorlage».

Die Auswertungen in Formular 2 sind von dem aktuelle ausgewählten Datensatz abhängig. Sie zeigen auch die Umbuchungen auf.

Die Auswertungen in Formular 3 beziehen sich auf alle eingetragenen Datensätze.

In Formular 2 wird auch der Anfangsmonat und der Endmonat eingegeben, für den eine Übersicht ausgegeben werden soll. Die beiden Datumsfelder gehören zur Tabelle "Filter". Die einzige

Auswertung erfolgt in Form eines Formulars. Der Button dafür wurde in das Formular 3 platziert, da durch das Verlassen von Formular 2 der Datensatz mit der den Datumsangaben gespeichert wird.

Der Code für das Listenfeld «Buchung aus Vorlage» wird hier noch gesondert aufgeführt, weil er sich von der HSQLDB zu FIREBIRD unterscheidet.

HSQLDB:

```
001 SELECT LEFT("Verwendungszweck"|| SPACE((SELECT
    MAX(CHAR_LENGTH("Verwendungszweck")) FROM "Kasse" WHERE "periodisch" =
    TRUE)), (SELECT MAX(CHAR_LENGTH("Verwendungszweck")) FROM "Kasse" WHERE
    "periodisch" = TRUE))
    || '  ' ||
    REPLACE( RIGHT( SPACE(8) ||"Betrag", 8),'.',',') || ' €  ' ||
    RIGHT( '0' || EXTRACT(DAY FROM "Datum" ), 2 ) || '.' || RIGHT( '0' ||
    EXTRACT( MONTH FROM "Datum" ), 2 ) || '.' || EXTRACT( YEAR FROM "Datum" )
    AS "Periode",
002 "ID"
003 FROM "Kasse" WHERE "periodisch" = TRUE
004 ORDER BY "Periode" ASC
```

FIREBIRD:

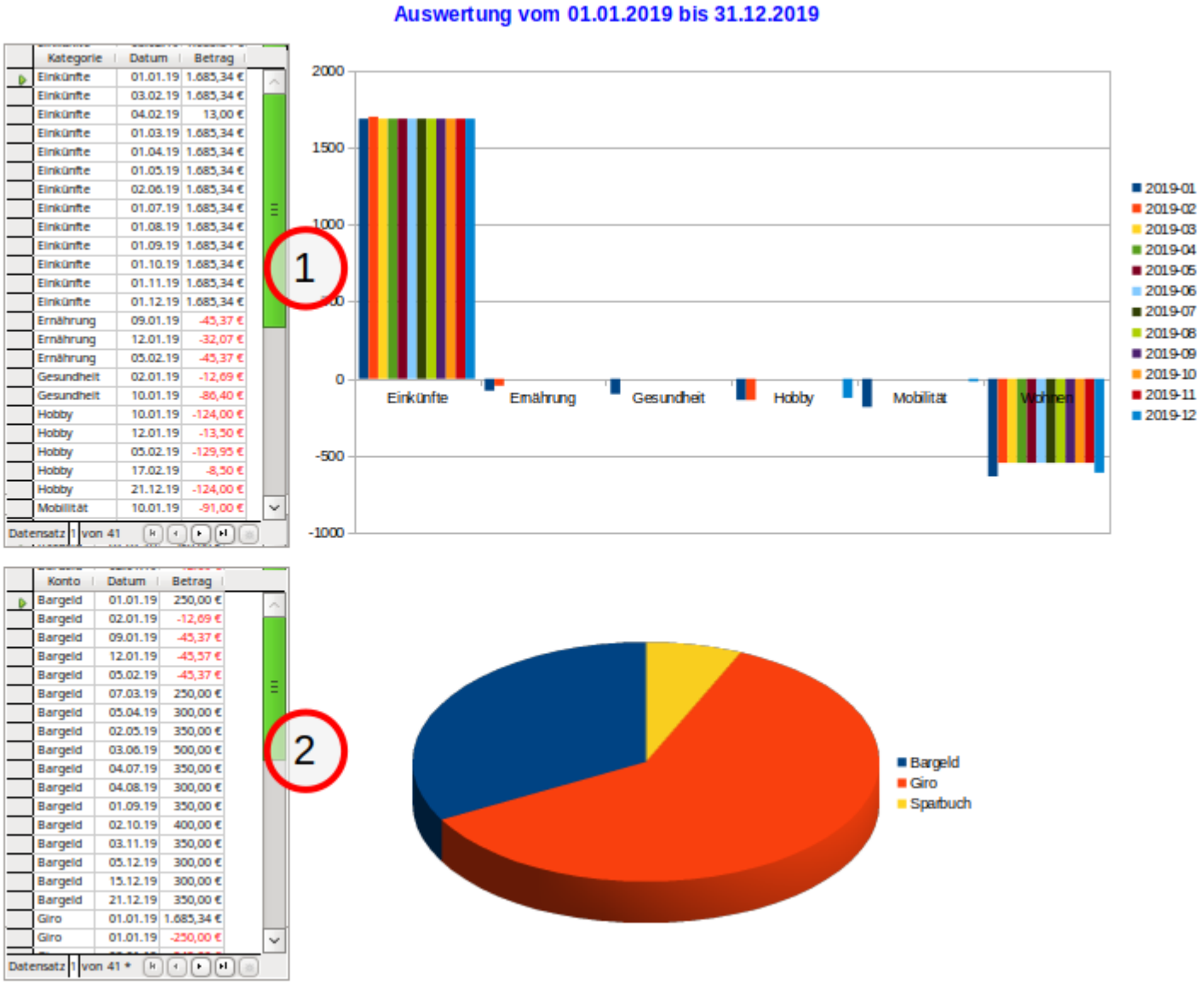
```
001 SELECT RPAD("Verwendungszweck", (SELECT
    MAX(CHAR_LENGTH("Verwendungszweck")) FROM "Kasse" WHERE "periodisch" =
    TRUE))
    || '  ' ||
    REPLACE( LPAD("Betrag", 8),'.',',') || ' €  ' ||
    RIGHT( '0' || EXTRACT(DAY FROM "Datum" ), 2 ) || '.' || RIGHT( '0' ||
    EXTRACT( MONTH FROM "Datum" ), 2 ) || '.' || EXTRACT( YEAR FROM "Datum" )
    AS "Periode",
002 "ID"
003 FROM "Kasse" WHERE "periodisch" = TRUE
004 ORDER BY "Periode" ASC
```

Der Code für die HSQLDB wurde vor dem Export so weit wie möglich angepasst. Die Länge des Feldes "Verwendungszweck" wird zuerst für alle betroffenen Felder, bei denen "periodisch" = TRUE ist, ermittelt. Damit kann dann ein entsprechend breiter Platz für diesen Eintrag bei fester Schriftart erzeugt werden. In der HSQLDB wird ein Leerzeichen so oft eingefügt, wie es diese Länge erfordert. Danach wird der gesamte String genommen und auf die beabsichtigte Länge abgeschnitten.

FIREBIRD kennt die Funktion **SPACE** nicht, hat dafür aber Funktionen wie **RPAD** und **LPAD**. Diese Funktionen füllen auf die beabsichtigte Länge automatisch mit der entsprechenden Anzahl an Leerzeichen von rechts aus (**RPAD**) oder von links aus (**LPAD**) auf. Es könnte sogar noch ein entsprechendes Ausfüllzeichen statt des Leerzeichens mit angegeben werden. Hierfür würde in der HSQLDB die Funktion **REPEAT** benötigt.

Das Listenfeld ist unter **Eigenschaften → Allgemein → Schrift** auf die Schriftart «Liberation Mono» eingestellt. Eine Schriftart mit fester Breite ist für die tabellarische Darstellung erforderlich.

Kategorie_Konto_Auswertung



1	FormKategorie (Abfrage: *Kategorieverlauf*)
1	Objekt1 Diagramm oben (Ansicht: *Ansicht_Kategorie_Diagramm*)
1	Objekt2 Diagramm unten (Ansicht: *Ansicht_Konto_Diagramm*)
2	FormKonto (Abfrage: *Kontoverlauf*)

Makros in Formulareigenschaften		
1	Beim Laden	Module2.*ChangeData*

In dem oberen Diagramm wird der monatliche Verlauf für jede Kategorie angezeigt. Das Diagramm macht natürlich deutlich, dass es sich bei den Daten nicht um einen tatsächlichen Verlauf handelt. Die Ausgaben für die Ernährung sind so gering, dass die Person vermutlich längst gestorben wäre. Wie sie dann aber noch etwas für das Hobby zum Jahresschluss ausgeben kann ist natürlich fraglich.

Im zweiten Diagramm wird die prozentuale Verteilung der Summen auf die verschiedenen Konten im Auswertungszeitraum angezeigt.

Beide Diagramme liegen übrigens im gleichen Formular und werden über ein Makro nacheinander mit den aktuellen Daten versorgt. In dem Formular befindet sich ein verstecktes Feld, das die Informationen über die Namen der Diagramme und die Bezeichnung für die Ansichten enthält, die den Datenbestand für die Diagramme enthalten.

Die Auswertung der Finanzen mit Diagrammen war ursprünglich in Berichten vorgesehen. Das Verfahren, Diagramme auch in Formulare einzubinden, ist nicht wesentlich aufwendiger, wenn das entsprechende Makro erst einmal existiert.

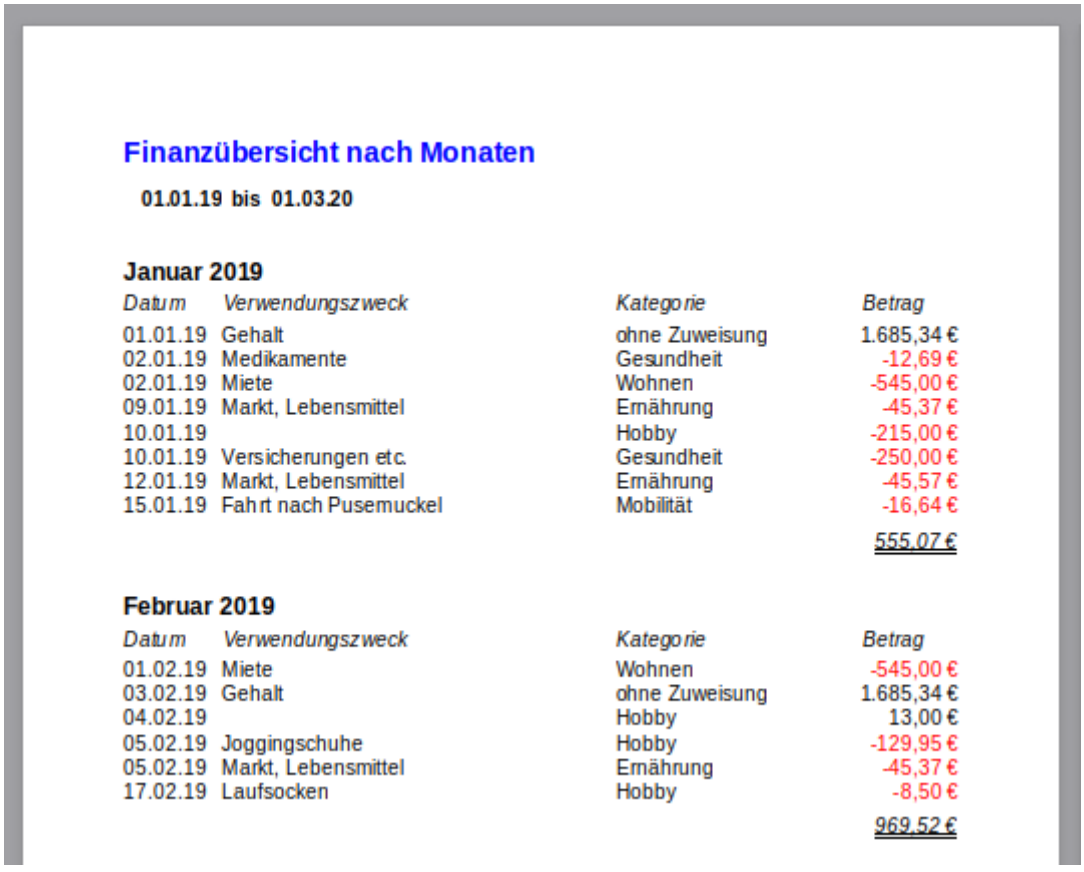
Vom Prinzip her geht das Ganze folgendermaßen:

4. Writer öffnen und "Diagramm einfügen" aufrufen. Dort den Diagrammtyp auswählen.
5. Das so erstellte Dummydiagramm kopieren und in das Formular einfügen.
6. Eine Ansicht zur Versorgung des Diagramms mit Daten erstellen, wie sie oben in den Ansichten aufgezeigt wurde.
7. In einem versteckten Feld im Formular, das für das Makro die Bezeichnung "Chart" haben muss, folgende Eingaben machen: Wert → Objekt1 und Zusatzinformation → Ansicht_Kategorie_Diagramm.
8. Beim Laden des Formulars das Makro *ChangeData* ausführen lassen.

Berichte

Die ursprüngliche Fassung dieser Datenbank enthielt Berichte, die eine Auswertung mit Diagrammen anzeigten. Da die Diagramme aber seit LO 5.3 nicht mehr angezeigt wurden, das Löschen der alten Diagramme sogar reproduzierbar zum Absturz von LO führt, ist statt der Berichte jetzt ein entsprechendes Formular eingebaut worden, das hoffentlich sicherer läuft.

Lediglich ein Bericht für die Fassung ohne Makros, der «Bericht_Kategorie», ist Bestandteil der Beispieldatenbank:



Finanzübersicht nach Monaten

01.01.19 bis 01.03.20

Januar 2019

Datum	*Verwendungszweck*	*Kategorie*	*Betrag*
01.01.19	Gehalt	ohne Zuweisung	1.685,34 €
02.01.19	Medikamente	Gesundheit	-12,69 €
02.01.19	Miete	Wohnen	-545,00 €
09.01.19	Markt, Lebensmittel	Ernährung	-45,37 €
10.01.19		Hobby	-215,00 €
10.01.19	Versicherungen etc.	Gesundheit	-250,00 €
12.01.19	Markt, Lebensmittel	Ernährung	-45,57 €
15.01.19	Fahrt nach Pusemuckel	Mobilität	-16,64 €
			555,07 €

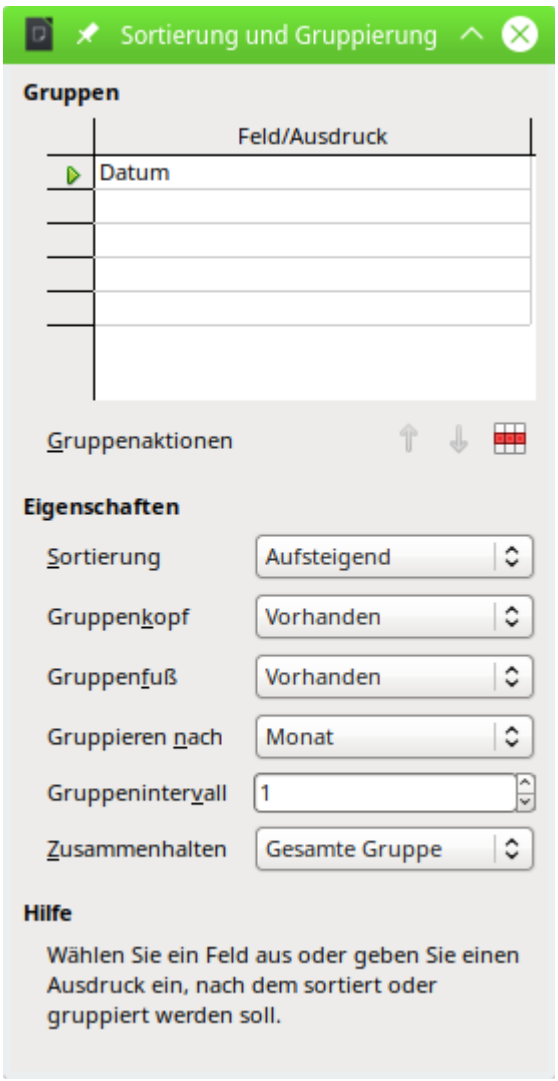
Februar 2019

Datum	*Verwendungszweck*	*Kategorie*	*Betrag*
01.02.19	Miete	Wohnen	-545,00 €
03.02.19	Gehalt	ohne Zuweisung	1.685,34 €
04.02.19		Hobby	13,00 €
05.02.19	Joggingschuhe	Hobby	-129,95 €
05.02.19	Markt, Lebensmittel	Ernährung	-45,37 €
17.02.19	Laufsocken	Hobby	-8,50 €
			969,52 €

Benötigt
Ansicht: *Ansicht_Bericht_Kategorie*

Der Bericht gibt eine Übersicht über die monatlichen Ausgaben. Die entsprechenden Summen dazu werden im Bericht erstellt.

Der Bericht enthält einen Berichtskopf, in dem neben dem Titel das Startdatum und das Enddatum, des Berichtes aufgelistet ist. Dann folgt der Gruppenkopf, der für die jeweiligen Monatsdarstellungen gilt.



Die Sortierung erfolgt nach dem Datum. Gruppenkopf und Gruppenfuß sind vorhanden. Sobald ein Monat zuende ist wird ein Gruppenfuß erstellt. Sobald ein neuer Monat anfängt wird ein Gruppenkopf erstellt. Das wird durch die monatliche Gruppierung erreicht. Das Gruppenintervall mit '1' gibt an, dass es sich hier um Schritte von einem Monat handelt. Es könnte auch mehrere Monate zusammengefasst werden.

Die letzte Einstellung **Zusammenhalten → Gesamte Gruppe** benötigt hier ein gesondertes Augenmerk. Dies bedeutet, dass versucht wird, sämtliche Einträge von Gruppenkopf bis einschließlich Gruppenfuß auf einer Seite zusammen zu fassen. Bei vielen Einträgen z.B. zum Monat 'Januar 2019' kann dies dazu führen, dass die Gruppe nicht mehr komplett auf die erste Seite passt. Der erste Lösungsweg ist hier der Beginn auf der zweiten Seite. Schließlich enthält die zweite Seite keinen Berichtskopf und damit mehr Platz. Reicht auch die zweite Seite nicht aus, dann muss innerhalb der Gruppe ein Umbruch stattfinden. Die Folge aber ist: Auf der ersten Seite steht der Gruppenkopf dann alleine.

Da der Bericht zuerst als ein Writer-Dokument erstellt wird kann gegebenenfalls noch nachgebessert werden. Das in oben genanntem Sonderfall ungünstige Zusammenhalten könnte dann in den Tabelleneigenschaften zurückgestellt werden.

Der Inhalt vom Gruppenkopf wird über das Datumsfeld des ersten Datensatzes erstellt. Das Datum ist hier einfach entsprechend formatiert worden, so dass nur der Monat und das Jahr erscheinen.

Im Gruppenfuß werden die Beträge aus der gesamten Gruppe über die Funktion **Summe** aufsummiert. Das gleiche Verfahren, nur für den ganzen Bericht, folgt im Berichtsfuß.

Manche kleinen Fehler tauchen erst bei der Erstellung von Berichten auf. Hier z.B. die Summierung für den Monat Februar 2020. Die doppelte Unterstreichung wurde nicht angezeigt, wenn die Farbe automatisch gesetzt werden sollte. Stattdessen musste eine Farbe vorbestimmt werden, so dass die Unterstreichung des negativen Wertes leider schwarz erfolgt.

Unter dem gesamten Bericht im Berichtsfuß taucht schließlich noch der Gesamtbetrag des Berichtes auf.

Eine Splittung des Gesamtbetrages nach Kategorien würde einen erneuten Bericht erfordern, da es nicht möglich ist, die Gruppierung einmal nach den Monaten ablaufen zu lassen und dann eine separate Berechnung nach den Kategorien aufzustellen. Die einzige Form, hier eine Übersicht hinein zu bekommen, wäre das Diagramm, das aber innerhalb von Berichten immer wieder Probleme bereitet.

Makros

Aktualisieren_einfach

Aufruf aus
Formular: *Konto_Salden_komplett_eM, Konto_Salden_komplett_Umbuchung_eM*
Makro: *Aktualisieren*

```
001  SUB Aktualisieren_einfach
002     DIM oDoc AS OBJECT
003     DIM oDrawpage AS OBJECT
004     DIM oForm AS OBJECT
005     oDoc = thisComponent
006     oDrawpage = oDoc.drawpage
007     oForm = oDrawpage.forms.getByName("MaxDatum")
008     oForm.reload
009  END SUB
```

Auf dem Weg über das Dokument und die Zeichnungsoberfläche wird das Formular mit dem Namen 'MaxDatum' aufgesucht (Zeile 5 bis 7). Das Formular wird neu geladen (Zeile 8).

Aktualisieren

Aufruf aus
Formular: *Konto_Salden_komplett, Konto_Salden_komplett_Umbuchung, Buchung_Umbuchung_Salden*

Benötigt
Makro: *SplitRest, Aktualisieren_einfach*

```
001  SUB Aktualisieren(oEvent AS OBJECT)
002     SplitRest(oEvent)
003     Aktualisieren_einfach
004  END SUB
```

Dieses Makro ist lediglich deshalb erforderlich, weil das Nebenformular sonst von einer Änderung der Daten im Hauptformular nichts mit bekommt. Zuerst wird die eventuelle Splitbuchung nach einer Datensatzänderung durchgeführt (Zeile 2). Anschließend wird dann des Nebenformular aktualisiert.

ZeileZwischenspeichern

Aufruf aus
Formular: *Kasse, Kasse_Umbuchung, Konto, Konto_Salden, Konto_Salden_komplett, Konto_Salden_komplett_Umbuchung, Buchung_Umbuchung_Salden*

```
001  SUB ZeileZwischenspeichern(oEvent AS OBJECT)
002     DIM oForm AS OBJECT
003     DIM oHiddenFeld AS OBJECT
004     DIM inUmbuchID AS INTEGER
005     DIM stUmbuch AS STRING
006     oForm = oEvent.Source
007     IF InStr(oForm.ImplementationName, "ODatabaseForm") THEN
008        oHiddenFeld = oForm.getByname("Hidden")
009        inUmbuchID = oForm.findColumn("Umbuch_Konto_ID")
010        stUmbuch = oForm.getString(inUmbuchID)
011        IF oEvent.Action = 3 OR stUmbuch <>"" THEN
012           oHiddenFeld.HiddenValue = "no"
013        ELSEIF oEvent.Action = 1 THEN
014           oHiddenFeld.HiddenValue = oForm.RowCount + 1
015        ELSE
016           oHiddenFeld.HiddenValue = oForm.Bookmark()
017        END IF
018     END IF
019  END SUB
```

Die Prozedur «ZeileZwischenspeichern» ist notwendig, damit auf den Datensatz zugegriffen werden kann, für den Eintragungen bei den Kategorien erfolgen sollen. Bei der normalen Bewegung durch ein Formular wird der Speichermodus z.B. durch das Verlassen einer Eingabezeile im Tabellenkontrollfeld ausgelöst. Damit das Unterformular aber mit den korrekten Werten bestückt werden kann muss nach dem Verlassen der Zeile wieder auf die Zeile zurückgesprungen werden. Erst dann sind die Formulare «SubForm» und «SubSubForm» korrekt miteinander verbunden.

Die Prozedur wird durch das Ereignis vor der Datensatzaktion ausgelöst. Mit dem Ereignis sind zwei Implementationen verbunden, so dass das Ereignis jedes normale Makro also doppelt durchlaufen würde. Über den Implementationsnamen wird hier die Implementation heraus gesucht, die einen Zugriff auf die Formularfelder bietet:
`com.sun.star.comp.forms.ODatabaseForm` (Zeile 7).

Das versteckte Feld wird bekannt gemacht (Zeile 8). Außerdem wird das Feld herausgesucht, das in der dem Formular zugrundeliegenden Abfrage «Umbuch_Konto_ID» heißt. Dies geht hier über die Funktion **`oForm.findColumn`**. Die Funktion gibt nur eine Zahl wieder, und zwar die Zahl der entsprechenden Tabellenspalte. Über **`oForm.getString`** wird der Wert aus dieser Spalte als Text ausgelesen (Zeile 9 und 10). Das Auslesen erfolgt hier als Text, da über einen Text leichter leere Felder erkennbar sind.

Wenn das Feld «Umbuch_Konto_ID» nicht leer ist, dann handelt es sich um eine Umbuchung. Folglich wird dort keine Zuweisung der Kategorie gebraucht. Gleiches gilt, wenn das auslösende Ereignis eine Löschung des Datensatzes bewirken soll. Das Löschen wird über **`oEvent.Action = 3`** abgefragt. Beim Löschen wird schließlich jede Verbindung zu den Kategorien entfernt, eine

Eingabe dort ist auch gar nicht mehr möglich. Für diese beiden genannten Fälle wird in dem versteckten Feld die Bezeichnung 'no' gespeichert (Zeile 11 und 12).

Wenn es sich bei der Eingabe um einen noch nicht existierenden neuen Datensatz handelt (**oEvent.Action = 1**), dann kann auf den Datensatz nicht über die Funktion Bookmark zugegriffen werden. Stattdessen wird zu der Anzahl der Zeilen der Tabelle eine Zeile hinzugezählt und der Wert entsprechend in dem versteckten Feld gespeichert (Zeile 14). Damit kann anschließend auf die Zeile zurückgesprungen werden.

Handelt es sich um eine Datensatzänderung, dann wird über **oForm.Bookmark()** die entsprechende Zeile gespeichert.

Nach Ablauf dieser Prozedur erfolgt die Speicherung des aktuellen Datensatzes in dem Tabellenkontrollfeld. Nach der Datensatzaktion startet dann die Prozedur *SplitRest*.

SplitRest

Aufruf aus
Formular: *Kasse, Kasse_Umbuchung, Konto, Konto_Salden*
Makro: *Aktualisieren*

Benötigt
Tabelle: *Kasse, rel_Kasse_Kategorie*

```
001 SUB Splitrest(oEvent AS OBJECT)
002    DIM oDoc AS OBJECT
003    DIM oForm AS OBJECT
004    DIM oSubForm AS OBJECT
005    DIM oFeld AS OBJECT
006    DIM oHiddenFeld AS OBJECT
007    DIM oController AS OBJECT
008    DIM oView AS OBJECT
009    DIM oDatenquelle AS OBJECT
010    DIM oVerbindung AS OBJECT
011    DIM oSQL_Anweisung AS OBJECT
012    DIM stSql AS STRING
013    DIM oSQL_Anweisung2 AS OBJECT
014    DIM stSql2 AS STRING
015    DIM oAbfrageergebnis AS OBJECT
016    DIM inID AS INTEGER
017    DIM loID AS LONG
018    DIM inBetrag AS INTEGER
019    DIM doBetrag AS DOUBLE
020    DIM doRest AS DOUBLE
021    oDoc = thisComponent
022    IF oEvent.Source.hasByName("Hidden") THEN
023       oForm = oEvent.Source
024    ELSE
025       oForm = oEvent.Source.Parent
026    END IF
027    oHiddenFeld = oForm.getByname("Hidden")
028    IF oHiddenFeld.HiddenValue = "no" THEN
029       EXIT SUB
030    END IF
031    IF oHiddenFeld.HiddenValue <> "" THEN
032       oForm.absolute(oHiddenFeld.HiddenValue)
033       oHiddenFeld.HiddenValue = ""
034    END IF
035    oDatenquelle = ThisComponent.Parent.CurrentController
036    If NOT (oDatenquelle.isConnected()) THEN
037       oDatenquelle.connect()
038    END IF
```

```
039     oVerbindung = oDatenquelle.ActiveConnection()
040     oSQL_Anweisung = oVerbindung.createStatement()
041     oSQL_Anweisung2 = oVerbindung.createStatement()
042     inID = oForm.findColumn("ID")
043     loID = oForm.getLong(inID)
044     IF oHiddenFeld.Tag <> "" THEN
045        stSql = "SELECT ""Kategorie_ID"", ""Betrag"" FROM ""rel_Kasse_Kategorie""
             WHERE ""Kasse_ID"" ='"+oHiddenFeld.Tag+"'"
046        oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
047        oHiddenFeld.Tag = ""
048        WHILE oAbfrageergebnis.next
049           doBetrag = oAbfrageergebnis.getDouble(2)
050           stSql2 = "INSERT INTO ""rel_Kasse_Kategorie"" (""Kasse_ID"",
                ""Kategorie_ID"", ""Betrag"") VALUES
                ('"+loID+"','"+oAbfrageergebnis.getShort(1)+"','"+str(doBetrag)+"')"
051           oSQL_Anweisung2.executeUpdate(stSql2)
052        WEND
053     END IF
054     stSql = "SELECT ""a"".""Betrag"" - COALESCE((SELECT SUM(""Betrag"")
          FROM ""rel_Kasse_Kategorie"" WHERE ""Kasse_ID"" = ""a"".""ID""),0.00)
          AS ""Wert"" FROM ""Kasse"" AS ""a"" WHERE ""a"".""ID"" = '"+loID+"'"
055     oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
056     WHILE oAbfrageergebnis.next
057        doRest = oAbfrageergebnis.getDouble(1)
058     WEND
059     oSubForm = oForm.getByname("SplitKategorie")
060     oFeld = oSubForm.getByName("tblKategorieBetrag")
061     oController = oDoc.getCurrentController()
062     oView = oController.getControl(oFeld)
063     oView.setFocus
064     wait 500
065     IF doRest <> 0 THEN
066        inBetrag = oSubForm.findColumn("Betrag")
067        oSubForm.MoveToInsertRow
068        oSubForm.UpdateDouble(inBetrag,doRest)
069     END IF
070  END SUB
```

Mit dieser Prozedur soll berechnet werden, welcher Betrag in der Splittabelle für die Kategorien zur Verfügung steht. Der Betrag soll direkt in den neuen Datensatz eingegeben werden. Es ist also nur noch die Wahl einer Kategorie erforderlich.

Wird der Betrag nur zum Teil einer Kategorie zugeordnet, so wird das Makro aus SubSubForm heraus erneut aufgerufen und dem nächsten Datensatz in der Splittabelle wird der verbleibende Rest zugewiesen – so lange, bis kein Rest mehr bleibt.

Da die Prozedur aus Formularen mit unterschiedliche Formularstruktur (Formular, Unterformular usw.) ausgelöst wird muss zuerst geklärt werden, ob das Formular auch das gewünschte Formular ist. In dem Formular soll sich das Feld mit der Bezeichnung «Hidden» befinden (Zeile 22 bis 26).

Ist in dem Feld «Hidden» der **`HiddenValue = "no"`**, so wird die Prozedur nicht weiter ausgeführt (Zeile 28 und 29). Entweder handelt es sich in diesem Fall um eine Umbuchung oder der auslösende Datensatz aus dem Formular wurde gelöscht. Eine Zuweisung von Kategorien ist also nicht erforderlich oder sogar unmöglich.

Der Wert aus dem versteckten Feld wird über **`oForm.absolute()`** zum Sprung auf den Datensatz aus dem Eingabeformular für die Buchung genutzt. Das versteckte Feld muss nur einmal ausgelesen werden. Es sollte auf jeden Fall geleert werden, wenn die Prozedur durchlaufen wurde (Zeile 31 bis 34). Wird grundsätzlich das Makro erst vom Formular ausgelöst und dann vom Unterformular für die Kategorien eine Eingabe getätigt, so ist die Zeile auf jeden Fall korrekt. Es könnte aber passieren, dass bei einer Auslösung des Makros direkt aus dem Tabellenkontrollfeld für die Zuweisung der Kategorie ohne vorherige Änderung des Buchungsdatensatzes ein Sprung zu einem Buchungsdatensatz erfolgt, der zuletzt geändert wurde.

In den Zeilen 35 bis 39 wird erst noch einmal die Datenbankverbindung überprüft und als Objekt **oVerbindung** zur Verfügung gestellt. Anschließend wird für zwei SQL-Anweisungen ein Objekt bereitgestellt.

Im Buchungsformular wird nach dem Wert des Primärschlüsselfeldes gesucht (Zeile 42 und 43). Die anschließende Bedingungsabfrage von Zeile 44 bis 53 trifft nur zu, wenn es sich um eine periodische Buchung handelt. Diese Erweiterung ist für das Formular *Buchung_Umbuchung_Salden* notwendig. Die Prozedur *periodische_Buchung* schreibt einen Wert in **oHiddenFeld.Tag**.

Bei einer periodischen Buchung soll auch die Zuweisung der Kategorie automatisch erfolgen. Eine periodische Buchung wird aber nicht vollautomatisch ausgeführt sondern liest nur zuerst einmal die Werte der vorhergehenden Buchung in das Buchungsformular ein. Deshalb kann nicht der komplette Datensatz direkt erstellt werden. Über **oHiddenFeld.Tag** wird der Schlüsselwert des Datensatzes ausgelesen, aus dem die periodische Buchung erfolgt ist (Zeile 45). Über diesen Schlüsselwert können alle Einträge aus der Tabelle «rel_Kasse_Kategorie» ausgelesen werden, die zu dem Buchungsdatensatz passen. Sie werden nacheinander (Zeile 48 bis 52) als neue Datensätze mit der aktuellen Schlüsselnummer **loID** in die Tabelle «rel_Kasse_Kategorie» neu eingefügt. Dabei erweist sich das Einfügen des Betrages als etwas problematisch. Wird der Wert nicht in einen String umgewandelt (**str(doBetrag)**), so gibt Basic statt eines Wertes mit Dezimalpunkt einen Wert mit der Formatierung des eingestellten Landes der GUI wieder, also in diesem Fall ein Dezimalkomma. Das führt unweigerlich dazu, dass der Datensatz nicht eingefügt wird.

Anschließend wird über eine Abfrage festgestellt, wie hoch die Differenz zwischen dem Betrag zu der passenden Buchungsnummer und der Summe der Beträge zu der entsprechenden Buchungsnummer in der Tabelle «rel_Kasse_Kategorie» ist (Zeile 54 bis 58). Danach wird das Unterformular bestimmt (Zeile 59) und der Cursor in das Tabellenkontrollfeld gesetzt (Zeile 60 bis 63).

In den Test hat es sich als notwendig erwiesen, vor weiteren Aktionen mit dem Unterformular für die Eingabe der Kategorie eine kurze Wartezeit einzulegen (Zeile 64).

Ist nach der Abfrage noch ein Rest zu verzeichnen, so wird eine neue Zeile in dem Unterformular für die Kategorien erstellt. In diese Zeile wird der Restbetrag eingetragen. Ist hingegen kein Rest mehr vorhanden, so erübrigt sich ein Eintrag. Die Prozedur wird beendet.

periodische_Buchung

Aufruf aus
Formular: *Buchung_Umbuchung_Salden*

Benötigt
Tabelle: *Kasse, Filter*

```
001  SUB periodische_Buchung(oEvent AS OBJECT)
002     DIM oDoc AS OBJECT
003     DIM oDrawpage AS OBJECT
004     DIM oForm AS OBJECT
005     DIM oFeld AS OBJECT
006     DIM oHiddenFeld AS OBJECT
007     DIM oDatenquelle AS OBJECT
008     DIM oVerbindung AS OBJECT
009     DIM oSQL_Anweisung AS OBJECT
010     DIM stSql AS STRING
011     DIM oAbfrageergebnis AS OBJECT
012     DIM loID AS LONG
013     DIM loIDNew AS LONG
014     DIM inSave AS INTEGER
015     DIM unoDate
```

```
016     oFeld = oEvent.Source.Model
017     IF oFeld.CurrentValue = "" THEN
018        EXIT SUB
019     END IF
020     loID = oFeld.CurrentValue
021     oDoc = thisComponent
022     oDrawpage = oDoc.drawpage
023     oForm = oDrawpage.forms.getByName("MainForm")
024     oForm.moveToInsertRow
025     oHiddenFeld = oForm.getByName("Hidden")
026     oHiddenFeld.Tag = loID
027     oDatenquelle = ThisComponent.Parent.CurrentController
028     If NOT (oDatenquelle.isConnected()) THEN
029        oDatenquelle.connect()
030     END IF
031     oVerbindung = oDatenquelle.ActiveConnection()
032     oSQL_Anweisung = oVerbindung.createStatement()
033     stSql = "SELECT ""Betrag"", ""Konto_ID"", ""Adressat_ID"", ""Verwendungszweck"",
           ""Umbuch_Konto_ID"" FROM ""Kasse"" WHERE ""ID"" ='"+loID+"'"
034     oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
035     WHILE oAbfrageergebnis.next
036        oForm.UpdateDouble(oForm.findColumn("Betrag"),oAbfrageergebnis.getDouble(1))
037        oForm.UpdateLong(oForm.findColumn("Konto_ID"),oAbfrageergebnis.getLong(2))
038        oForm.UpdateLong(oForm.findColumn("Adressat_ID"),oAbfrageergebnis.getLong(3))
039        IF oAbfrageergebnis.wasNull() THEN
040           oForm.UpdateNULL(oForm.findColumn("Adressat_ID"),NULL)
041        END IF
042        oForm.UpdateString(oForm.findColumn("Verwendungszweck"),
              oAbfrageergebnis.getString(4))
043        oForm.UpdateLong(oForm.findColumn("Umbuch_Konto_ID"),
              oAbfrageergebnis.getLong(5))
044        IF oAbfrageergebnis.wasNull() THEN
045           oForm.UpdateNULL(oForm.findColumn("Umbuch_Konto_ID"),NULL)
046        END IF
047     WEND
048     inSave = MsgBox("Soll der Datensatz mit aktuellem Datum abgespeichert und
           die Kategorien hinzugefügt werden?", 36 ,"Buchungsübernahme")
049     IF inSave = 6 THEN
050        unoDate = createUnoStruct("com.sun.star.util.Date")
051        unoDate.Year = Year(Now())
052        unoDate.Month = Month(Now())
053        unoDate.Day = Day(Now())
054        oForm.UpdateDate(oForm.findColumn("Datum"),unoDate)
055        IF oForm.isNew THEN
056           oForm.insertRow()
057        ELSE
058           oForm.updateRow()
059        END IF
060     END IF
061     oSQL_Anweisung.executeUpdate("UPDATE ""Filter"" SET ""Kasse_ID"" = NULL WHERE
           ""ID"" = TRUE")
062     oFeld.Parent.reload
063  END SUB
```

Im Hauptformular des Formulars *Buchung_Umbuchung_Salden* können Datensätze als periodische Datensätze gekennzeichnet werden. Diese Datensätze werden dann als Vorlage für andere Datensätze über das Listenfeld ganz oben im Formular abrufbar. Der Inhalt dieser Datensätze wird, bis auf das Buchungsdatum, als neuer Datensatz eingefügt und muss nur noch abgespeichert werden.

Der Wert des Listenfeldes wird ermittelt (Zeile 16 bis 20). Es handelt sich dabei nicht um den angezeigten Wert, sondern um den Inhalt des Schlüsselfeldes, das damit verbunden ist. In diesem Fall ist das der Primärschlüsselinhalt der Tabelle «Kasse».

Dieser Wert wird zur weiteren Nutzung in dem bereits in der Prozedur *SplitRest* vorgestellten versteckten Feld, in diesem Fall in den Zusatzinformationen (Tag), zwischengelagert (Zeile 26). Er wird dort später für die Splitbuchungen bei den Kategorien benötigt.

Die Datenbankverbindung wird überprüft und die Verbindung zur als Objekt zur Verfügung gestellt (Zeile 27 bis 31). Der Datensatz zu dem entsprechenden Primärschlüssel aus der Tabelle "Kasse" wird ausgelesen (Zeile 33 und 34). Die Felder "ID" und "Datum" werden nicht benötigt. "Datum" wird sowieso neu eingestellt und "ID" ist bekannt – außerdem soll ja ein neuer Datensatz erstellt wird, so dass "ID" über den Autowert neu festgelegt wird.

Sämtliche Werte werden direkt als neue Werte in das Formular übertragen (Zeile 35 bis 49). Dabei wäre es nicht notwendig, eine **`WHILE - WEND`** - Schleife zu verwenden, da die Abfrage sowieso nur einen Datensatz als Ergebnis hat. **`oAbfrageergbnis.next`** würde für das eine Ergebnis in Zeile 35 ausreichen. Mit **`wasNull()`** wird das letzte Ergebnis jeweils getestet. Damit wird ausgeschlossen, dass stattdessen '0' für die "Adressat_ID" weiter gegeben wird, obwohl dort vielleicht kein Wert enthalten ist. "Betrag" und "Konto_ID" hingegen sind Felder, die nach Tabellendefinition nie **`NULL`** sein dürfen.

Der folgende "Verwendungszweck" darf zwar auch **`NULL`** sein. Wenn dort allerdings ein leerer String eingetragen wird, so schadet dies nicht der Integrität der Dateneingabe. Bei der "Umbuch_Konto_ID" muss hingegen wieder darauf geachtet werden, dass tatsächlich **`NULL`** übermittelt wird.

In Zeile 50 wird eine Messagebox ausgegeben. Die Zahl '36' besagt hier, dass ein Ja- und ein Nein-Button angezeigt werden (Wert: 4) und ein Fragezeichen-Symbol erscheint (Wert: 32). Wird auf die Nachfrage mit Ja geantwortet, so gibt die Messagebox den Wert 6 zurück. Nur dann wird in den Zeile 52 bis 55 das aktuelle Datum zusammengestellt und in Zeile 56 in das entsprechende Formularfeld geschrieben.

Zeile 57 bis 61 dienen der Speicherung der eingegebenen Elemente. Hier wurde die sichere Variante genutzt, die zuerst abklärt, ob die Datenzeile neu ist, um dann den entsprechenden Befehl abzusetzen. Da in dem Formular gleichzeitig andere Makros mitlaufen kann es sein, dass die Zeile schon teilweise abgespeichert wurde und deswegen ein **`insertRow()`** zu einem Fehler in der Funktionsreihenfolge führt.

Nach dem Abspeichern wird automatisch die Prozedur *SplitRest* aufgerufen, die dann aus dem alten Eintrag der Tabelle "Kasse" den Primärschlüssel in dem versteckten Feld (Zeile 26) vorfindet, den neuen Primärschlüssel aus dem Formular ausliest und damit auch die Inhalte für die Kategorien von der einen Buchung zu der anderen Buchung überträgt.

Zum Schluss wird der Wert des Listenfeldes über den entsprechenden Wert in der Tabelle "Filter" wieder zurück gesetzt (Zeile 64).

Adressat_neu

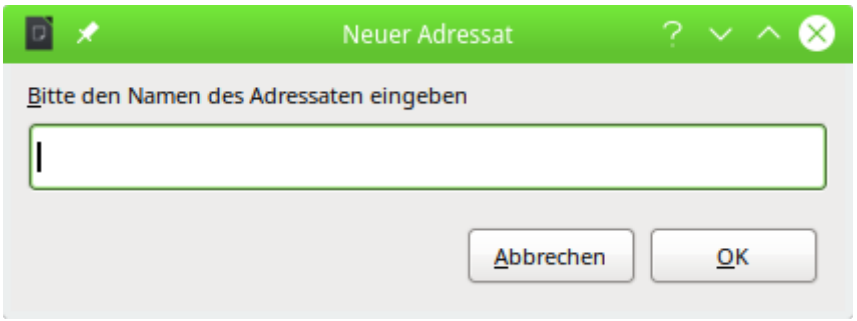
Aufruf aus
Formular: *Buchung_Umbuchung_Salden*

Benötigt
Tabelle: *Adressat*

```
001  SUB Adressat_neu(oEvent AS OBJECT)
002     DIM oForm AS OBJECT
003     DIM oListbox AS OBJECT
004     DIM stInput AS STRING
005     DIM oDatenquelle AS OBJECT
006     DIM oVerbindung AS OBJECT
007     DIM oSQL_Anweisung AS OBJECT
008     DIM oSQL_Anweisung2 AS OBJECT
009     DIM stSql AS STRING
```

```
010     DIM oAbfrageergebnis AS OBJECT
011     stInput = InputBox("Bitte den Namen des Adressaten eingeben","Neuer Adressat")
012     stInput = Trim(stInput)
013     IF stInput = "" THEN
014        msgbox "Der Eintrag für den Adressaten ist leer."
015        EXIT SUB
016     END IF
017     oDatenquelle = ThisComponent.Parent.CurrentController
018     If NOT (oDatenquelle.isConnected()) THEN
019        oDatenquelle.connect()
020     END IF
021     oVerbindung = oDatenquelle.ActiveConnection()
022     oSQL_Anweisung = oVerbindung.createStatement()
023     oSQL_Anweisung2 = oVerbindung.createStatement()
024     stSql = "SELECT ""Name"" FROM ""Adressat"" WHERE ""Name"" ='"+stInput+"'"
025     oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
026     IF oAbfrageergebnis.Next THEN
027        msgbox "Der Adressat """+stInput+""" ist schon vorhanden."
028     ELSE
029        oSQL_Anweisung2.executeUpdate("INSERT INTO ""Adressat"" (""Name"") VALUES
           ('" + stInput + "')")
030        oForm = oEvent.Source.Model.Parent
031        oListbox = oForm.getByName("MainForm_Grid").getByName("lboAdressat")
032        oListbox.refresh
033     END IF
034 END SUB
```

Am Anfang fehlen noch viele Adressaten. Hier soll auf einfach Weise wenigstens ein Name für den Adressaten abgefragt werden, damit eine Eingabe zügig erfolgen kann.



Solch ein Eingabefeld wird in der Zeilen 11 gebildet. Der Inhalt wird eingelesen, auf mögliche gleiche Inhalte hin überprüft und ansonsten in die Datenbank übertragen. Der Inhalt der **InputBox** wird als Text in **stInput** gespeichert. Um unnötige Leertasten auszuschließen wird **stInput** mit **Trim(stInput)** von Leertasten vor und nach der sichtbaren Eingabe befreit (Zeile 12).

Ist der Inhalt leer, so wird das Makro nach einer kurzen Meldung beendet (Zeile 13 bis 16). Ansonsten wird eine Verbindung zur Datenquelle aufgebaut, sofern noch nicht vorhanden. Da eine SQL-Anweisung abgesetzt wird, während die andere noch im Speicher liegt, werden für die SQL-Anweisungen zwei Objekte erstellt (Zeile 17 bis 23).

In der Datenbank wird jetzt nachgefragt, ob der "Name" in genau der Schreibweise schon vorhanden ist. Bei dieser Nachfrage wird nach Groß- und Kleinschreibung unterschieden. Wenn klar ist, dass der "Name" nicht bereits vor kommt (Zeile 26 und 27) wird die zweite Anweisung, nämlich das Einfügen des neuen Datensatzes, abgearbeitet.

Nachdem ein neuer Datensatz eingefügt wurde ist es erforderlich, dass das Listenfeld seinen Datenbestand neu einliest. Da der Button, der das Makro ausgelöst hat, genau im gleichen Formular liegt wie das Tabellenkontrollfeld für die Buchungseingabe, kann das Formular über **oEvent.Source.Model.Parent** erreicht werden (Zeile 30). Das Listenfeld liegt im Tabellenkontrollfeld. Das Tabellenkontrollfeld hat den Namen «MainForm_Grid», das Listenfeld den Namen «lboAdressat». Über **oListbox.refresh** wird der Inhalt des Listenfeldes erneuert.

Monat_aktuell

Aufruf aus
Formular: *Buchung_Umbuchung_Salden*

```
001  SUB Monat_aktuell(oEvent AS OBJECT)
002     DIM oForm AS OBJECT
003     oForm = oEvent.Source.Model.Parent
004     oForm.Filter = "EXTRACT( YEAR FROM ""Datum"") =
           EXTRACT( YEAR FROM CURRENT_DATE )
           AND EXTRACT( MONTH FROM ""Datum"") = EXTRACT( MONTH FROM CURRENT_DATE )"
005     oForm.ApplyFilter = True
006     oForm.reload
007  END SUB
```

Der Button für den Filter liegt in dem Formular, dessen Filter entsprechend eingestellt werden soll. Der Filtertext in Zeile 4 enthält den SQL-Befehl, mit dem auch eine Tabelle entsprechend gefiltert würde. Er wird über den Formularfilter an **SELECT * FROM "Kasse"** mit **WHERE** angehängt.

Der Filter wird mit **oForm.Filter** zugewiesen. Mit **oForm.ApplyFilter** wird der Filter eingestellt. Er kann also auch später über die Navigationsleiste wieder abgestellt und erneut eingestellt werden. Zum Schluss wird das Formular neu eingelesen.

Formular_starten

Aufruf aus
Formular: *Buchung_Umbuchung_Salden*

```
001  SUB Formular_starten(oEvent AS OBJECT)
002     DIM stFormular AS STRING
003     stFormular = oEvent.Source.Model.Tag
004     ThisDatabaseDocument.FormDocuments.getByName(stFormular).open
005  END SUB
```

Diese Prozedur startet über einen Button ein Formulardokument. Der Name des entsprechenden Formulardokuments ist in den Zusatzinformationen (**Tag**) des Buttons enthalten.

Aktuelles_Buchungsdatum

Aufruf aus
Formular: *Buchung_Umbuchung_Salden*

Benötigt	
Tabelle:	*Filter*
Makro:	*Kontoliste_filtern*

```
001  SUB Aktuelles_Buchungsdatum(oEvent AS OBJECT)
002     DIM oForm AS OBJECT
003     DIM oDatenquelle AS OBJECT
004     DIM oVerbindung AS OBJECT
005     DIM oSQL_Anweisung AS OBJECT
006     DIM stSql AS STRING
007     DIM oDoc AS OBJECT
008     DIM oDrawpage AS OBJECT
009     DIM oForm2 AS OBJECT
010     DIM oFeld AS OBJECT
011     DIM inID AS INTEGER
012     oForm = oEvent.Source
```

```
013     IF oForm.IsNew THEN
014        stSql = "UPDATE ""Filter"" SET ""DatensatzDatum"" = CURRENT_DATE
              WHERE ""ID"" = TRUE"
015     ELSE
016        stSql = "UPDATE ""Filter"" SET ""DatensatzDatum"" = '" +
              oForm.getString(oForm.findColumn("Datum")) + "' WHERE ""ID"" = TRUE"
017     END IF
018     oDatenquelle = ThisComponent.Parent.CurrentController
019     If NOT (oDatenquelle.isConnected()) THEN
020        oDatenquelle.connect()
021     END IF
022     oVerbindung = oDatenquelle.ActiveConnection()
023     oSQL_Anweisung = oVerbindung.createStatement()
024     oSQL_Anweisung.executeUpdate(stSql)
025     oDoc = thisComponent
026     oDrawpage = oDoc.drawpage
027     oForm2 = oDrawpage.forms.getByName("Filter")
028     oForm2.reload
029     inID = oForm.getInt(oForm.findColumn("Konto_ID"))
030     oFeld = oForm.getByname("Kasse_Grid").getByName("lboUmbuchID")
031     Kontoliste_filtern(inID,oFeld)
032 END SUB
```

Das Makro sorgt dafür, dass die Salden für das Konto und die Kategorie abhängig von dem aktuellen Datensatz ermittelt werden können und, wenn gerade eine Neueingabe gemacht wird, abhängig vom aktuellen Datum dargestellt werden.

Nach dem Datensatzwechsel im Hauptformular wird ermittelt, ob ein neuer Datensatz eingefügt werden soll: **`oForm.IsNew`** (Zeile 13). Handelt es sich um einen neuen Datensatz, so wird in der Tabelle "Filter" das Feld "DatensatzDatum" auf das aktuelle Datum **`CURRENT_DATE`** eingestellt. Handelt es sich nicht um ein einen neuen Datensatz, so wird stattdessen in der Tabelle "Filter" das Feld "DatensatzDatum" auf den Wert des aktuellen Datums des gerade erreichten Datensatzes eingestellt: **`oForm.getString(oForm.findColumn("Datum"))`**.

Nachdem die Verbindung zur Datenbank überprüft und Die SQL-Anweisung vorbereitet wurde (Zeile 18 bis 23) wird der SQL-Befehl an die Datenbank verschickt und das Nebenformular «Filter» auf den aktuellen Wert entsprechend eingestellt (Zeile 27 und 28).

Nach dem Datensatzwechsel muss außerdem das Listenfeld für das Umbuchkonto neu eingestellt werden. Dies erfolgt in den Zeilen 29 bis 31.

Kontofilter_Formstart

Aufruf aus
Formular: *Konto_Salden_komplett_Umbuchung*

Benötigt
Makro: *Kontoliste_filtern*

```
001 SUB Kontofilter_Formstart(oEvent AS OBJECT)
002     DIM oForm AS OBJECT
003     DIM oFeld AS OBJECT
004     DIM inID AS INTEGER
005     oForm = oEvent.Source
006     inID = oForm.getInt(oForm.findColumn("Konto_ID"))
007     oFeld = oForm.getByname("Kasse_Grid").getByName("lboUmbuchID")
008     Kontoliste_filtern(inID,oFeld)
009 END SUB
```

Wenn der Datensatz im Formular gewechselt wird soll das Listenfeld für das Umbuchungs-Konto neu eingelesen werden. So kann, zusätzlich zu der Beschränkung in der Tabelle *Kasse*, ausgeschlossen werden, dass das abgebende Konto und das Umbuchungs-Konto gleich sein können.

In Zeile 6 wird der aktuelle Wert für die "Konto_ID" über das Formular ausgelesen. Zeile 7 ermittelt anschließend das Objekt des Listenfeldes für das Umbuchungs-Konto. Hier sind zwei Ebenen mit **getByName** eingebaut, da dieses Feld innerhalb eines Tabellenkontrollfeldes liegt. Der ermittelte Wert für das abgebende Konto wird zusammen mit dem Objekt für das Listenfeld des Umbuchungs-Kontos an das Makro «Kontoliste_Filtern» weitergegeben.

Kontofilter_Feldstart

Aufruf aus
Formular: *Buchung_Umbuchung_Salden*

Benötigt
Makro: *Kontoliste_filtern*

```
001 SUB Kontofilter_Feldstart(oEvent AS OBJECT)
002    DIM oFeldStart AS OBJECT
003    DIM oFeld AS OBJECT
004    DIM inID AS INTEGER
005    oFeldStart = oEvent.Source.Model
006    inID = oFeldStart.ValueItemList(oEvent.Selected)
007    oFeld = oFeldStart.Parent.getByName("lboUmbuchID")
008    Kontoliste_filtern(inID,oFeld)
009 END SUB
```

Das Makro soll dann den korrekten Wert für das Listenfeld auslesen, wenn das Listenfeld gerade geändert wurde. Der Wert ist also noch nicht in dem Formular gespeichert und auch nicht über **CurrentValue** erreichbar.

In Zeile 6 wird deswegen der Wert über die Position in der Werteliste bestimmt. Direkt nach dem Auslösen wird unter **oEvent.Selected** diese Position angegeben. Hier könnte auch stattdessen **oFeldStart.SelectedItemPos** stehen.

Über das auslösende Feld **oFeldStart** wird in Zeile 7 das Feld für das Umbuchung-Konto ermittelt. Dabei wird einfach eine Ebene nach oben gegangen und von dort aus der Name des Umbuchung-Kontos aufgerufen.

Der ermittelte Wert für das ausgewählte abgebende Konto wird zusammen mit dem Objekt für das Listenfeld des Umbuchungs-Kontos an das Makro «Kontoliste_Filtern» weitergegeben.

Kontoliste_filtern

Aufruf aus
Makro: *Aktuelles_Buchungsdatum, Kontofilter_Formstart, Kontofilter_Feldstart*

```
001 SUB Kontoliste_filtern(inID,oFeld)
002    DIM stSql(0) AS STRING
003    stSql(0) = "SELECT ""Konto"", ""ID"" FROM ""Konto"" WHERE ""ID"" <> "+inID+"
          ORDER BY ""Konto"""
004    oFeld.ListSource = stSql
005    oFeld.refresh
006 END SUB
```

Der SQL-Code für die Liste der Umbuchungskonten wird neu zusammengestellt. In dem Listenfeld soll der Wert nicht mehr angezeigt werden, der bereits für das Konto vergeben ist.

Das Listenfeld wird mit einem Array bestückt. Dieses Array enthält lediglich einen Wert. Deshalb die entsprechende Definition in Zeile 2. In Zeile 3 wird dann der aktuelle Wert des abgebenden Kontos von der Liste ausgeschlossen. Anschließend wird dieses Array dem Listenfeld zugewiesen und das Listenfeld neu eingelesen.

ChangeData

Aufruf aus
Formular: *Kategorie_Konto_Auswertung*

```
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       stSql = "SELECT * FROM """+arView(n)+""""
029       oDatasource = ThisComponent.Parent.CurrentController
030       IF NOT (oDatasource.isConnected()) THEN
031          oDatasource.connect()
032       END IF
033       oConnection = oDatasource.ActiveConnection()
034       oSQL_Command = oConnection.createStatement()
035       oResult = oSQL_Command.executeQuery(stSql)
036       i = 0
037       k = 0
038       x = 0
039       WHILE oResult.next
040          stRow = oResult.getString(1)
041          stType = oResult.getString(4)
042          IF aNewRowDescription(i) = stRow THEN
043             ReDim Preserve aNewColumnDescription(k)
044             ReDim Preserve aTmp(k)
045          ELSE
046             IF x > 0 THEN
047                i = i + 1
048             ELSE
049                x = 1
050             END IF
051             ReDim Preserve aNewRowDescription(i)
052             ReDim Preserve aNewData(i)
053             k = 0
054             ReDim Preserve aNewColumnDescription(k)
055             ReDim aTmp(k)
056          END IF
057          aNewRowDescription(i) = stRow
058          aNewColumnDescription(k) = oResult.getString(2)
059          aTmp(k) = oResult.getDouble(3)
060          aNewData(i) = aTmp()
```

```
061             k = k + 1
062         WEND
063         oDiag = thisComponent.EmbeddedObjects.getByName(arDiag(n))
064         oXCOEO = oDiag.ExtendedControlOverEmbeddedObject
065         oXCOEO.changeState(4)
066         IF stType <> "XY" THEN
067             oDiag.model.Data.setData(aNewData)
068         END IF
069         oDiag.model.DataProvider.setData(aNewData)
070         oDiag.model.DataProvider.setRowDescriptions(aNewRowDescription)
071         oDiag.model.DataProvider.setColumnDescriptions(aNewColumnDescription)
072         oDiag.Component.setmodified(true)
073         oDiag.Component.update()
074         oXCOEO.changeState(1)
075     NEXT
076 END SUB
```

Nach der Deklaration der Variablen wird zuerst aus den Zusatzinformationen (**Tag**) des versteckten Kontrollfeldes der Name für die Ansicht ausgelesen (Zeile 25). 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 (Zeile 26). Anschließend läuft eine Schleife (Zeile 27 bis 74) 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 (Zeile 28), die Verbindung zur Datenbank, falls erforderlich, hergestellt (Zeile 29 bis 32) und der SQL-Code an die Datenbank weitergeleitet (Zeile 33 bis 35). 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 (Zeile 39 bis 62) 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 (Zeile 40). Hier handelt es sich um Zeilenbeschriftungen, die eben einfach Text sind, in der vorliegenden Datenbank z.B. die Bezeichnungen der Kategorien. 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 (Zeile 42 bis 44).

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 (Zeile 46 und 47). 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** (Zeile 51 und 52). **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** (Zeile 54 und 55). 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)** (Zeile 59).

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 (Zeile 63). 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 (Zeile 66 bis 69). 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.

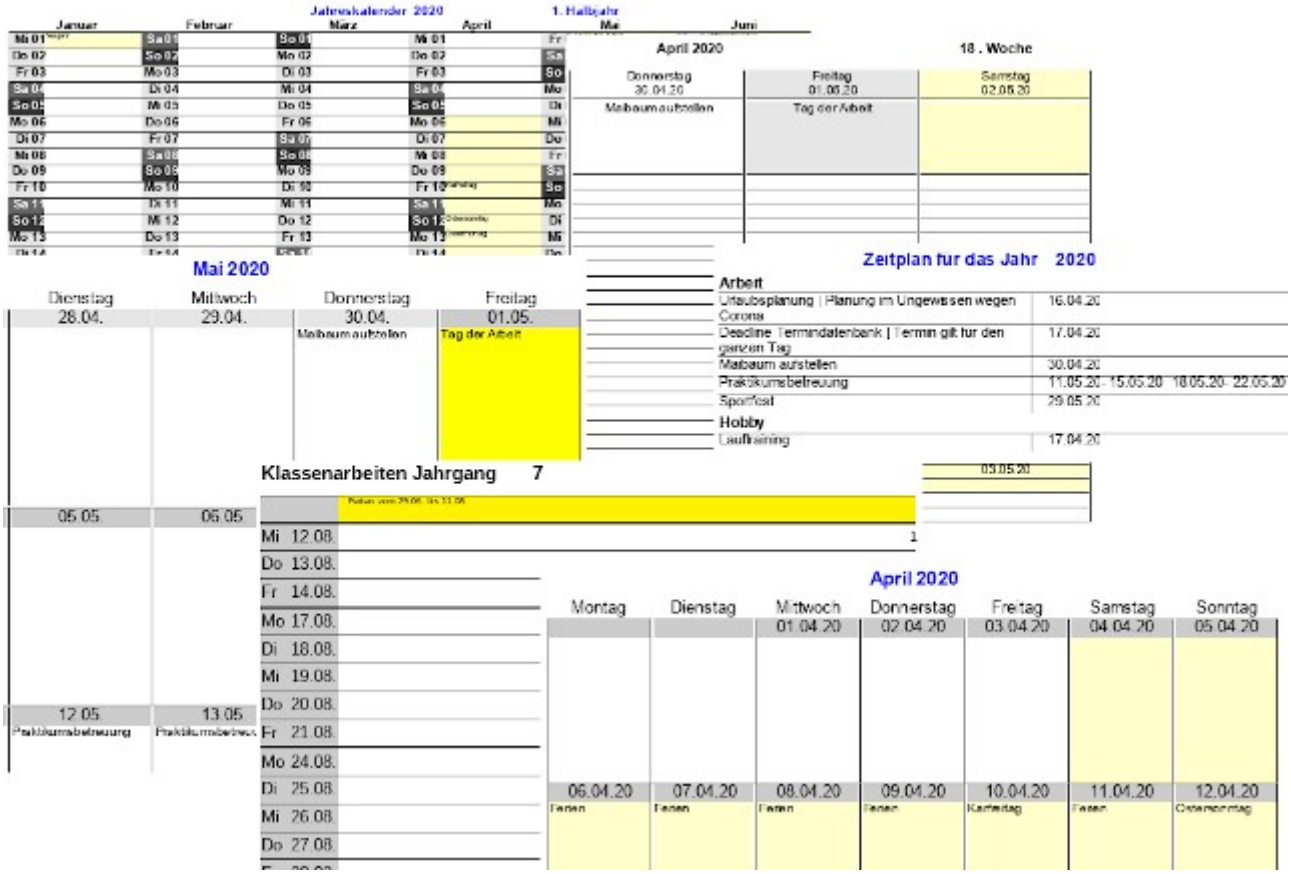
Nach der Änderung wird der Status des Diagramms auf '1' gesetzt, da ansonsten das Diagramm die ganze Zeit im Bearbeitungsmodus erscheint.

Eine Übersicht über die möglichen Diagrammtypen liefert: *https://api.libreoffice.org/docs/idl/ref/servicecom_1_1sun_1_1star_1_1chart_1_1Diagram.html*

Termine

Einführung

Die Datenbank «Termine»[9] dient nicht dazu, einen einfachen Terminplaner zu ersetzen. Terminplaner sind für die Verwaltung privater Termine wesentlich besser geeignet. Die Datenbank ist entstanden, um an einer Schule die über das ganze Schuljahr verstreuten Termine zusammenzufassen und in Form von verschiedenen Berichten auf unterschiedliche Art verfügbar zu machen. So gibt es einen Bericht, aus dem das schuljährliche Terminheft erstellt wird (jede Seite ein kompletter Monat mit allen Tagen außer Samstag und Sonntag), einen anderen Bericht, der unter den Terminen viele Einträge ermöglicht und so für Organisationszwecke wie Vertretungsunterricht herangezogen werden kann.



Ausdrucke über Berichten sind die zentrale Anwendung dieser Termindatenbank.

Diese Datenbank ist vor allem auch deswegen entstanden, weil es eben im Schulalltag viele verschiedene Ausgabemöglichkeiten für Termine gibt und es immer problematisch ist, Termine per Hand von einem Formular in ein anderes einzutippen. Fehler beim Abgleich sind vorprogrammiert.

Diese Datenbank wird beständig erweitert. So ist ein Feld für eine Vielfachgruppe hinzu gekommen. Es bestand plötzlich das Problem, dass doch die Jahresübersichten für Klassenarbeiten abhängig von den Terminen für bestimmte Jahrgänge erstellt werden sollten. Eine Eingabe für bestimmte Klassen war aber gar nicht vorgesehen.

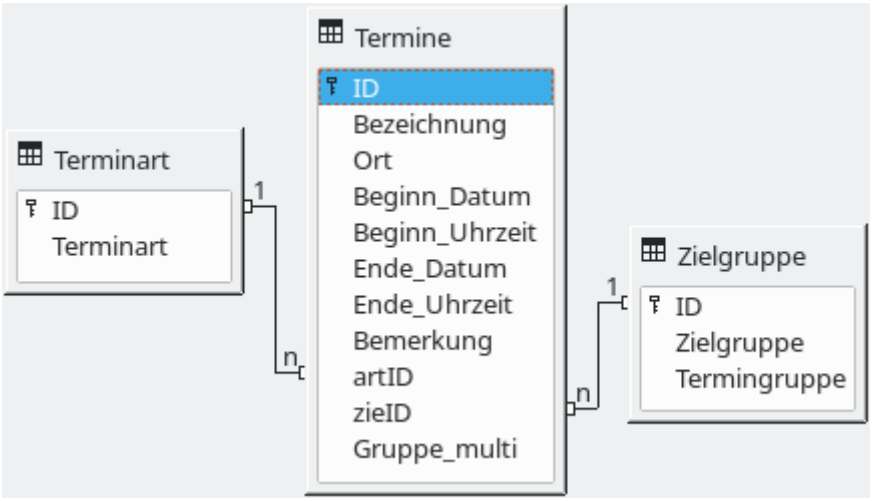
Aktuellste Erweiterung ist die Möglichkeit, Termine für einen bestimmten Zeitraum in eine iCalendar-Datei (*.ics) zu exportieren, so dass die Termine für KollegInnen nicht nur in gedruckter Form, sondern auch direkt in digitaler Form in deren Kalender übernommen werden können. Auch der Import aus iCalendar-Daten ist in einem Formular mit integriert.

9 Beispieldatenbank Beispiel_Termine.odb

Tabellen

Die Datenbank besteht aus insgesamt 3 Tabellen, die miteinander verknüpft sind. Die restlichen Tabellen sind temporär durch Makros erstellte Tabellen, eine Filtertabelle und eine Tabelle, in der die Termine der Ferien direkt eingetragen werden. Daneben tauchen in dem Tabellenordner noch einige Ansichten auf.

Tabellen zur Dateneingabe über Formulare oder direkt



Termine

Datenziel	
Ansicht:	*Ansicht_Kalender_Export, Ansicht_Kalender_ohne_Ferien_Export, Ansicht_Termine_Monat_Jahre*
Abfrage:	*Termine_Gruppe_multi, Terminuebersicht*
Makro:	*Kalender, Zeitplan, Monat, GruppeMultiUebersicht, Termindaten_uebertragen, Tabellenstart, Ical_Import*
Formular, Bericht::	keine direkt

Feldname	***Feld-typ***	***Beschreibung***
ID	Integer	Primärschlüssel, als Autowert ausgelegt. Basis für iCalendar-Eintrag **UID**
Bezeichnung	Text	Terminbeschreibung in Kurzform Eingabe erforderlich: Ja Basis für iCalendar-Eintrag **SUMMARY**
Ort	Text	Wo findet der Termin statt? Eingabe erforderlich: Nein Basis für iCalendar-Eintrag **LOCATION**
Beginn_Datum	Datum	Starttag des Termins Eingabe erforderlich: Ja Datumsbasis für iCalendar-Eintrag **DTSTART**
Beginn_Uhrzeit	Zeit	Startuhrzeit des Termins Eingabe erforderlich: Nein Zeitbasis für iCalendar-Eintrag **DTSTART**

Feldname	Feld-typ	Beschreibung
Ende_Datum	Datum	Endtag des Termins, kann auch frei bleiben. Dann ist der Starttag und der Endtag in der Auswertung gleichgesetzt. Eingabe erforderlich: Nein Datumsbasis für iCalendar-Eintrag **DTEND**
Ende_Uhrzeit	Zeit	Enduhrzeit des Termins, kann auch frei bleiben. Dann ist der Termin für einen ganzen Tag ausgelegt. Eingabe erforderlich: Nein Zeitbasis für iCalendar-Eintrag **DTEND**
Bemerkung	Text	Detailliertere Beschreibung Eingabe erforderlich: Nein Basis für iCalendar-Eintrag **DESCRITION**
artID	Integer	Fremdschlüssel für die Art des Termins, bei schulischen Belangen so etwas wie Fachkonferenz, Informationsveranstaltung usw. Eingabe erforderlich: Nein Wird nicht nach iCalendar exportiert
zieID	Integer	Fremdschlüssel für die Zielgruppe des Termins, bei schulischen Belangen so etwas wie LehrerInnen, Eltern, SchülerInnen oder Öffentlichkeit. Eingabe erforderlich: Ja Wird nicht nach iCalendar exportiert
Gruppe_multi	Integer	Mehrfacheintrag für gleichartige Gruppen, bei schulischen Belangen alle Jahrgänge der Schule. Der Eintrag erfolgt hier Bitweise, d.h. den Gruppen werden feste Werte im Zweiersystem zugeordnet: 20 = 1 | 21 = 2 | 22 = 4 | 23 = 8 usw. So können mehrere Gruppen in einer Zahl wiedergegeben werden. Wird z.B '7' eingetragen, so sind die ersten 3 Gruppen betroffen, weil 1 + 2 + 4 = 7. Solch ein Eintrag kann auch durch eine n:m-Beziehung von Tabellen erreicht werden. Dieses Feld wurde aber im Nachhinein hinzugefügt und konnte einfacher in die komplette Datenbank mit allen Abfragen usw. übernommen werden. Eingabe erforderlich: Nein

Terminart

Datenziel
Formular: *Terminart_Termin* (MainForm)
Makro: *Monat, Tabellenstart*
Ansicht, Abfrage, Bericht: keine direkt

Feld-name	Feld-typ	Beschreibung
ID	Integer	Primärschlüssel, als Autowert ausgelegt. Wird mit der Tabelle "Termine" verknüpft
Terminart	Text	Art des Termins, bei schulischen Belangen so etwas wie Fachkonferenz, Informationsveranstaltung usw. Eingabe erforderlich: Ja

Die Tabelle "Terminart" wird nur in einem der Formulare zum direkten Editieren genutzt. Die anderen Formulare greifen auf diese Tabelle nur lesend zu.

Zielgruppe

Datenziel
Makro: *Bericht_Start, Tabellenstart*
Ansicht, Abfrage, Formular, Bericht: keine direkt

Feldname	Feld-typ	Beschreibung
ID	Integer	Primärschlüssel, als Autowert ausgelegt. Wird mit der Tabelle "Termine" verknüpft
Zielgruppe	Text	Zielgruppe des Termins, bei schulischen Belangen so etwas wie LehrerInnen, Eltern, SchülerInnen oder Öffentlichkeit. Eingabe erforderlich: Ja
Termin-gruppe	Text	Auf welche Termine soll die Gruppe Zugriff haben. Lehrerkonferenzen z.B. brauchen nicht in der Terminübersicht für die Eltern zu erscheinen. Hier stehen, durch Kommas getrennt, die eigene ID sowie die IDs anderer Zielgruppen Eingabe erforderlich: Nein

Die Tabelle "Zielgruppe" wird nur direkt mit Daten gefüllt. Die Formulare greifen auf die Zielgruppe mit Listenfeldern zu. Die Termingruppe entscheidet, welche Termine später in den Berichten enthalten sind. Wenn eine Termingruppe mit allen Einträgen aus "Termingruppe"."ID" durch Komma getrennt bestückt ist, dann werden alle Termine aus dem Terminkalender beim Ausdruck für diese Gruppe berücksichtigt.

Diese oben genannten 3 Tabellen "Termine", "Terminart" und "Zielgruppe" wurden über **Extras → Beziehungen** miteinander verknüpft.

Ferien

Datenziel
Ansicht: *Ansicht_Gruppe_multi, Ansicht_Kalender_Export, Ansicht_Termine_Monat_Jahre*
Makro: *Kalender, GruppeMultiUebersicht, Tabellenstart*
Abfrage, Formular, Bericht: keine direkt

Feldname	Feld-typ	Beschreibung
ID	Integer	Primärschlüssel, als Autowert ausgelegt.
Beginn_Datum	Datum	Beginn von Ferien. Kann auch für den Beginn anderer Zeiten genutzt werden, bei denen kein Eintrag eines Termins erfolgt. Hier eingetragene Termine werden im Kalender als Ferien gekennzeichnet. Eingabe erforderlich: Ja
Ende_Datum	Datum	Ende der Ferien, bei einem einzelnen Ferientag gleich dem Beginndatum. Eingabe erforderlich: Nein

Diese Tabelle wird direkt editiert. Sie wird über Formulare nicht beeinflusst und nur bei den Berichten berücksichtigt.

Filtertabelle

Datenziel	
Ansicht:	*Ansicht_Gruppe_multi, Ansicht_Kalender_Export, Ansicht_Kalender_ohne_Ferien_Export, Ansicht_Termine_Monat_Jahre*
Formular:	*Terminart_Termin* (Berichte, Export), *Termine_Kalenderübersicht* (Monat, Berichte, Export), *Termine_Monat_Terminart* (Berichte, Export), *Termine_Tabelleneingabe* (Monat, Berichte, Export)
Makro:	*Bericht_Start, Monatsansicht, Monatsansicht_aktualisieren, DialogEnde, Ical_Import*
Abfrage, Bericht:	keiner direkt

In der Tabelle "Filter" werden folgende Felder definiert:

Feldname	***Feldtyp***	***Beschreibung***
ID	Ja/Nein	Primärschlüssel, erster Datensatz 'Ja' bereits eingegeben.
StartDatum	Datum	Startdatum für den Export nach iCalendar Eingabe erforderlich: Nein
EndDatum	Datum	Enddatum für den Export nach iCalendar Eingabe erforderlich: Nein
Jahr_Monat	Text	Einhält eine Kombination aus Jahreszahl und Monatszahl, durch Komma getrennt (z.B. '2020,04') Diese Eingabe ist erforderlich, damit die Monatsdarstellung einwandfrei funktioniert. Zur Not kann hier ein Dummy wie oben genannt eingesetzt werden. Eingabe erforderlich: Ja
Startjahr	Small Integer	Enthält die Jahreszahl, mit der der Kalenderausdruck startet Eingabe erforderlich: Nein
Halbjahrende	Datum	Nur für Schulen: Das Ende des ersten Halbjahres, damit Ausdrucke auch halbjahresweise erfolgen können. Eingabe erforderlich: Nein
Ferien	Ja/Nein	Sollen Ferien beim Export mit exportiert werden? Eingabe erforderlich: Nein
BerichtZielgruppeID	Integer	Fremdschlüsselfeld für die Zielgruppe beim Berichtsausdruck Nur zur Speicherung der Formulareingabe Eingabe erforderlich: Nein
BerichtStartDat	Datum	Startdatum für den Berichtsausdruck Nur zur Speicherung der Formulareingabe Eingabe erforderlich: Nein
BerichtEndDat	Datum	Enddatum für den Berichtsausdruck Nur zur Speicherung der Formulareingabe Eingabe erforderlich: Nein
BerichtName	Text	Berichtsname aus dem Listenfeld des Formulars – nicht unbedingt identisch mit dem Berichtsname in Base selbst Nur zur Speicherung der Formulareingabe Eingabe erforderlich: Nein

Feldname	Feldtyp	Beschreibung
BerichtGuppeMulti	Small Integer	Kennziffer für die gewünschte Gruppe beim Bericht "Liste_Gruppe_multi" Nur zur Speicherung der Formulareingabe Eingabe erforderlich: Nein
StartIDImport	Integer	Beim Import von iCalendar-Daten wird hier die erste neue ID aus der Tabelle "Termine" gespeichert. Dies dient zum Anzeigen der neu importierten Inhalte im Formular. Eingabe erforderlich: nein
EndIDImport	Integer	Beim Import von iCalendar-Daten wird hier die letzte neue ID aus der Tabelle "Termine" gespeichert. Wird nur ein Datensatz eingefügt, so ist die EndIDImport gleich der StartIDImport. Dies dient zum Anzeigen der neu importierten Inhalte im Formular. Eingabe erforderlich: nein

Diese Tabelle wird zum Filtern für den Export, für die Erstellung der Tabellen für die Berichte usw. genutzt. Sie enthält nur einen Datensatz. Der Datensatz muss durch die Eingabe des Primärschlüssels 'Ja' erstellt sein, bevor die Tabelle im Formular genutzt werden kann.

Durch Makros gefüllt Tabellen

Die weiteren Tabellen sind Tabellen, die jeweils durch Makros für den folgenden Ausdruck gefüllt werden. Sie starten alle mit der Benennung "tmp_".

tmp_Halbjahreskalender

Datenziel
Bericht: *Halbjahreskalender_A4_quer*
Makro: *Halbjahreskalender*
Ansicht, Abfrage, Formular: keine direkt

Feldname	Feldtyp	Beschreibung
ID	Integer	Primärschlüssel, als Autowert ausgelegt.
Jahr	Small Integer	Jahreszahl
T1 ... T6	Text	Insgesamt 6 Felder für Termine, siehe "Termine"."Terminbezeichnung" – 250 Zeichen
D1 ... D6	Datum	Zu den 6 Terminfeldern passende 6 Datumsfelder
Sort_Zahl	Tiny Integer	Sortierzahl, '1' für das erste Halbjahr, '2' für das 2. Halbjahr
F1 ... F6	Text	Ferienanzeige, wenn '1', dann wird dies als "Ferien" im Kalender vermerkt. 1 Zeichen

Die Termine werde in dieser Tabelle so aufgenommen, dass jeweils 6 Monatsverläufe nebeneinander präsentiert werden. In D1 beginnt die Zählung z.B. mit dem '1.1.2020', in D2 mit dem '1.2.2020' usw.

Da ein Monat maximal 31 Tage hat enthält diese Tabelle insgesamt für ein Jahr 62 Tabellenzeilen – 31 Tabellenzeilen für das erste Halbjahr und 31 Tabellenzeilen für das zweite Halbjahr.

tmp_Gruppe_multi

Datenziel
Ansicht: *Ansicht_Gruppe_multi*
Makro: *GruppeMultiUebersicht*
Abfrage, Formular, Bericht:: keine direkt

Feldname	Feld-typ	Beschreibung
ID	Integer	Primärschlüssel, als Autowert ausgelegt.
Datum	Datum	Komplette Datumsreihenfolge für ein (Schul)jahr
Bezeichnung	Text	Siehe "Termine"."Terminbezeichnung", auch Feiertage ..
Format	Text	Ferienanzeige, wenn '1', dann wird dies als "Ferien" im Kalender vermerkt. 1 Zeichen
Gruppe_multi	Text	Zuordnung des Termins zu einer Gruppe, siehe "Termine"."Gruppe_multi". 5 Zeichen

tmp_Kalender

Datenziel
Ansicht: *Ansicht_Wochentage*
Makro: *Kalender, Monatskalender, Halbjahreskalender*
Abfrage, Formular, Bericht: keine direkt

Feldname	Feld-typ	Beschreibung
ID	Integer	Primärschlüssel, als Autowert ausgelegt.
Datum	Datum	Komplette Datumsreihenfolge für ein (Schul)jahr
Bezeichnung	Text	Siehe "Termine"."Terminbezeichnung", auch Feiertage ..
Format	Text	Ferienanzeige, wenn '1', dann wird dies als "Ferien" im Kalender vermerkt. 1 Zeichen

tmp_Monat

Datenziel
Formular: *Termine_Kalenderübersicht* (Ansicht)
Makro: *Monat*
Ansicht, Abfrage, Bericht: keine direkt

Feldname	Feld-typ	Beschreibung
ID	Tiny Integer	Primärschlüssel
T1 ... T7	Text	Insgesamt 7 Felder für Termine (7 Tage), siehe "Termine"."Beginn_Datum", "Termine"."Terminbezeichnung" Alle Termine für einen Tag des Monats in einem Feld, durch Zeilenumbrüche voneinander abgesetzt, nach Wochentagen sortiert. – 500 Zeichen

tmp_Terminuebersicht

Datenziel
Bericht: *Zeitplan*
Makro: *Zeitplan*
Ansicht, Abfrage, Formular: keine direkt

Feldname	Feldtyp	Beschreibung
ID	Integer	Primärschlüssel, als Autowert ausgelegt.
Beginn_Datum	Datum	Startdatum für den Termin. Wenn der Termin nur an einem Tag liegt ist dies der einzige Datumseintrag.
Bezeichnung	Text	Siehe "Termine"."Terminbezeichnung"
Terminart	Text	Siehe "Terminart"."Terminart"
Monat	Tiny Integer	Monatszahl
Jahr	Small Integer	Jahreszahl
D2 ... D7	Text	Wiederholungstermine oder Terminfortführung bis zu dem entsprechenden Datum.

tmp_Wochenkalender

Datenziel
Ansicht: *Ansicht_Wochen_einzeln*
Abfrage: *Wochenkalender_Mo_Fr*
Bericht: *Monatskalender_Mo_So*
Makro: *Monatskalender*
Formular: keine direkt

Feldname	Feldtyp	Beschreibung
ID	Integer	Primärschlüssel, als Autowert ausgelegt.
Jahr	Small Integer	Jahreszahl
Monat_Text	Text	Monatsname, 20 Zeichen
Woche	Tiny Integer	Wochenzahl
T1 ... T7	Text	250 Zeichen
D1 ... D7	Datum	
Monat_Zahl	Tiny Integer	Monat, Zahlwert

Feldname	Feldtyp	Beschreibung
F1 ... F7	Text	Formatierung (Ferien, Feiertage) , 1 Zeichen
Monat_Sort	Tiny Integer	Sortierung bei Jahresübergängen sicherstellen

tmp_Wochenkalender_halb

Datenziel
Bericht: *Wochenkalender_halb*
Makro: *Wochenkalender_halb*
Ansicht, Abfrage, Formular: keine direkt

Feldname	Feld-typ	Beschreibung
ID	Integer	Primärschlüssel, als Autowert ausgelegt.
Jahr	Small Integer	Jahreszahl
Monat_Text	Text	Monatsname, 20 Zeichen
Woche	Tiny Integer	Wochenzahl
T1 ... T4	Text	Siehe "Termine"."Terminbezeichnung", Termine der Tage Montag – Mittwoch und Donnerstag – Sonntag werden im Wochenkalender in jeweils einer Zeile zusammengefasst – 250 Zeichen
D1 ... D4	Datum	Datum zu den Terminen
Monat_Zahl	Tiny Integer	Monat, Zahlwert
F1 ... F4	Text	Formatierung (Ferien, Feiertage) , 1 Zeichen

Ansichten

Ansicht_Gruppe_multi

Datenquelle
Tabelle: *tmp_Gruppe_multi, Filter, Ferien*

Datenziel
Abfrage: *Liste_Gruppe_Multi*
Ansicht, Formular, Bericht: keine direkt

```
001 SELECT "Datum" AS "Sort",
002 CASE WHEN DAYOFWEEK("Datum")=2 THEN 'Mo'
003 WHEN DAYOFWEEK("Datum")=3 THEN 'Di'
004 WHEN DAYOFWEEK("Datum")=4 THEN 'Mi'
005 WHEN DAYOFWEEK("Datum")=5 THEN 'Do'
006 WHEN DAYOFWEEK("Datum")=6 THEN 'Fr'
007 END AS "Tag",
008 RIGHT('0'||EXTRACT(DAY FROM "Datum"),2)||'.'||RIGHT('0'||EXTRACT (MONTH
    FROM "Datum"),2)||'.' AS "Dat",
```

```
 009 "Bezeichnung", "Format", EXTRACT (YEAR FROM "Datum")||RIGHT(0||
     WEEK("Datum"),2) AS "W1", "Gruppe_multi"
 010 FROM "tmp_Gruppe_multi"
 011 WHERE "Datum" BETWEEN ( SELECT "Startjahr" FROM "Filter" WHERE "ID" = TRUE
     ) || '-08-01' AND (SELECT "Startjahr" FROM "Filter" WHERE "ID" = TRUE ) +
     1 || '-08-01'
 012 AND DAYOFWEEK("Datum") BETWEEN 2 AND 6
 013 AND ( NOT "Bezeichnung" = 'Ferien' OR "Bezeichnung" IS NULL )
 014 UNION
 015 SELECT "Beginn_Datum",
 016 NULL AS "Tag",
 017 NULL AS "Dat",
 018 'Ferien vom '||RIGHT('0'||EXTRACT (DAY FROM "Beginn_Datum"),2)||'.'||
     RIGHT('0'||EXTRACT (MONTH FROM "Beginn_Datum"),2)||'.'||' bis '||
     RIGHT('0'||EXTRACT(DAY FROM "Ende_Datum"),2)||'.'||RIGHT('0'||EXTRACT
     (MONTH FROM "Ende_Datum"),2)||'.',
 019 '1',
 020 NULL,
 021 (SELECT DISTINCT "Gruppe_multi" FROM "tmp_Gruppe_multi")
 022 FROM "Ferien"
 023 WHERE ("Beginn_Datum" BETWEEN ( SELECT "Startjahr" FROM "Filter" WHERE
     "ID" = TRUE ) || '-08-01' AND ( SELECT "Startjahr" FROM "Filter" WHERE
     "ID" = TRUE ) + 1 || '-08-01' )
 024 OR ("Ende_Datum" BETWEEN ( SELECT "Startjahr" FROM "Filter" WHERE "ID"
     =TRUE ) || '-08-01' AND ( SELECT "Startjahr" FROM "Filter" WHERE "ID" =
     TRUE ) + 1 || '-08-01' )
```

Die Ansicht dient als Basis für einen Listenausdruck, der nach den Gruppen sortiert angefertigt wird. Dieser Listenausdruck wird nach Erstellung der Tabelle "tmp_Gruppe_multi" ermöglicht.

Die erste Spalte zeigt das Datum als Sortierelement an (Zeile 1), die zweite Spalte bezeichnet den Wochentag (Zeile 2 bis 7). Hier sind nur 5 Wochentage aufgeführt, da die Liste nur für montags bis freitags erstellt wird.

In Zeile 8 wird das Datum in Kurzform übernommen: '21.04', also zweistellige Tageszahl, getrennt mit einem Punkt von der zweistelligen Monatszahl. Die Zweistelligkeit wird erreicht, indem einfach eine Null mit der jeweiligen Zahl kombiniert wird und dann die beiden hinteren Zeichen durch **`RIGHT('String', 2 )`** übernommen werden.

Zeile 9 speichert dann noch das Jahr, gekoppelt mit der Kalenderwoche, so dass auch hierüber eine Sortierung erfolgen kann. Aus diesem Wert werden nachher in einer Abfrage die Schulwochen abgeleitet, die ja mitten im Jahr beginnen und auch die Ferien nicht berücksichtigen.

Die ausgelesenen Werte werden nach den folgenden Bedingungen zusammengestellt:
In Zeile 11 wird nach dem Startjahr aus der Tabelle "Filter" geschaut. Die Datumswerte für die Liste werden danach auf die Zeit vom 1.8. des Startjahres bis zum 31.7. des Folgejahres erlaubt. Der Schuljahresbeginn findet immer Anfang August statt, auch wenn dort z.B. noch Ferien sind.
Zeile 12 begrenzt die ausgelesenen Datumseinträge auf die Wochentage 2 bis 6, was den Wochentagen von Montag bis Freitag entspricht.
Zeile 13 und Zeile schließen dann noch aus, dass die "Bezeichnung" der Tabelle "tmp_Gruppe_multi" den Begriff 'Ferien' enthält. Diese Bedingung alleine würde aber die Felder, bei denen die "Bezeichnung" leer ist, auch mit aus der Ansicht heraus werfen. Sie müssen deshalb noch extra wieder mit ausgenommen werden. Entweder also nicht 'Ferien' oder ein leeres Feld.

Daran angehängt wird jetzt noch ein Auszug der Tabelle "Ferien". Für jeden Eintrag in der Tabelle "Ferien" wird hier ein Eintrag eingefügt, der dann anschließend in der auszudruckenden Übersicht als eine eingefärbte Zeile mit dem Verweis darauf existiert, dass die Tage fehlen, weil da eben Ferien liegen. Dieser Verweis wird in Zeile 18 zusammengestellt, die spezielle Formatierung verbirgt sich hinter der Ziffer '1', die in Zeile 19 hinzugefügt wird.

Ansicht_Kalender_Export

Datenquelle
Tabelle: *Termine, Filter, Ferien*

Datenziel
Makro: *Export_Start*
Ansicht, Abfrage, Formular, Bericht: keine direkt

```
001 SELECT RIGHT('0000000'||"ID",8) AS "UID",
002 "Ort" AS "Location", "Bezeichnung" AS "Summary", "Bemerkung" AS
    "Description",
003 COALESCE(
004 EXTRACT( YEAR FROM "Beginn_Datum" ) || RIGHT( '0' || EXTRACT( MONTH FROM
    "Beginn_Datum" ), 2 ) || RIGHT( '0' || EXTRACT( DAY FROM "Beginn_Datum" ),
    2 ) || 'T' || RIGHT( '0' || EXTRACT( HOUR FROM "Beginn_Uhrzeit" ), 2 ) ||
    RIGHT( '0' || EXTRACT( MINUTE FROM "Beginn_Uhrzeit" ), 2 ) || RIGHT( '0'
    || EXTRACT( SECOND FROM "Beginn_Uhrzeit" ), 2 ),
005 EXTRACT( YEAR FROM "Beginn_Datum" ) || RIGHT( '0' || EXTRACT( MONTH FROM
    "Beginn_Datum" ), 2 ) || RIGHT( '0' || EXTRACT( DAY FROM "Beginn_Datum" ),
    2 )) AS "DtStart",
006 COALESCE (
007 EXTRACT( YEAR FROM "Ende_Datum" ) || RIGHT( '0' || EXTRACT( MONTH FROM
    "Ende_Datum" ), 2 ) || RIGHT( '0' || EXTRACT( DAY FROM "Ende_Datum" ), 2 )
    || 'T' || RIGHT( '0' || EXTRACT( HOUR FROM "Ende_Uhrzeit" ), 2 ) || RIGHT(
    '0' || EXTRACT( MINUTE FROM "Ende_Uhrzeit" ), 2 ) || RIGHT( '0' ||
    EXTRACT( SECOND FROM "Ende_Uhrzeit" ), 2 ),
008 EXTRACT( YEAR FROM "Beginn_Datum" ) || RIGHT( '0' || EXTRACT( MONTH FROM
    "Beginn_Datum" ), 2 ) || RIGHT( '0' || EXTRACT( DAY FROM "Beginn_Datum" ),
    2 ) || 'T' || RIGHT( '0' || EXTRACT( HOUR FROM "Ende_Uhrzeit" ), 2 ) ||
    RIGHT( '0' || EXTRACT( MINUTE FROM "Ende_Uhrzeit" ), 2 ) || RIGHT( '0' ||
    EXTRACT( SECOND FROM "Ende_Uhrzeit" ), 2 ),
009 EXTRACT( YEAR FROM "Ende_Datum" ) || RIGHT( '0' || EXTRACT( MONTH FROM
    "Ende_Datum" ), 2 ) || RIGHT( '0' || EXTRACT( DAY FROM "Ende_Datum" )+1, 2
    ),
010 EXTRACT( YEAR FROM "Beginn_Datum" ) || RIGHT( '0' || EXTRACT( MONTH FROM
    "Beginn_Datum" ), 2 ) || RIGHT( '0' || EXTRACT( DAY FROM "Beginn_Datum" )
    +1, 2 )) AS "DtEnd",
011 EXTRACT( YEAR FROM CURRENT_DATE ) || RIGHT( '0' || EXTRACT( MONTH FROM
    CURRENT_DATE ), 2 ) || RIGHT( '0' || EXTRACT( DAY FROM CURRENT_DATE ), 2 )
    || 'T' || RIGHT( '0' || EXTRACT( HOUR FROM CURRENT_TIME ), 2 ) ||
    RIGHT( '0' || EXTRACT( MINUTE FROM CURRENT_TIME ), 2 ) || RIGHT( '0' ||
    EXTRACT( SECOND FROM CURRENT_TIME ), 2 ) AS "DtStamp"
012 FROM "Termine"
013 WHERE "Beginn_Datum" BETWEEN COALESCE((SELECT "StartDatum" FROM "Filter"
    WHERE "ID" = TRUE),"Beginn_Datum") AND COALESCE((SELECT "EndDatum" FROM
    "Filter" WHERE "ID" = TRUE),"Beginn_Datum")
014 OR "Ende_Datum" BETWEEN COALESCE((SELECT "StartDatum" FROM "Filter" WHERE
    "ID" = TRUE),"Ende_Datum") AND COALESCE((SELECT "EndDatum" FROM "Filter"
    WHERE "ID" = TRUE),"Ende_Datum")
015 UNION
016 SELECT RIGHT('000000'||"ID"||'F',8) AS "UID", '','Ferien','',
017 EXTRACT( YEAR FROM "Beginn_Datum" ) || RIGHT( '0' || EXTRACT( MONTH FROM
    "Beginn_Datum" ), 2 ) || RIGHT( '0' || EXTRACT( DAY FROM "Beginn_Datum" ),
    2 ) AS "DtStart",
018 COALESCE(
```

```
019 EXTRACT( YEAR FROM "Ende_Datum" ) || RIGHT( '0' || EXTRACT( MONTH FROM
    "Ende_Datum" ), 2 ) || RIGHT( '0' || EXTRACT( DAY FROM "Ende_Datum" )+1, 2
    ) ,
020 EXTRACT( YEAR FROM "Beginn_Datum" ) || RIGHT( '0' || EXTRACT( MONTH FROM
    "Beginn_Datum" ), 2 ) || RIGHT( '0' || EXTRACT( DAY FROM "Beginn_Datum" )
    +1, 2 ) ) AS "DtEnd",
021 EXTRACT( YEAR FROM CURRENT_DATE ) || RIGHT( '0' || EXTRACT( MONTH FROM
    CURRENT_DATE ), 2 ) || RIGHT( '0' || EXTRACT( DAY FROM CURRENT_DATE ), 2 )
    || 'T' || RIGHT( '0' || EXTRACT( HOUR FROM CURRENT_TIME ), 2 ) ||
    RIGHT( '0' || EXTRACT( MINUTE FROM CURRENT_TIME ), 2 ) || RIGHT( '0' ||
    EXTRACT( SECOND FROM CURRENT_TIME ), 2 ) AS "DtStamp"
022 FROM "Ferien"
023 WHERE "Beginn_Datum" BETWEEN COALESCE((SELECT "StartDatum" FROM "Filter"
    WHERE "ID" = TRUE),"Beginn_Datum") AND COALESCE((SELECT "EndDatum" FROM
    "Filter" WHERE "ID" = TRUE),"Beginn_Datum")
024 OR  "Ende_Datum" BETWEEN COALESCE((SELECT "StartDatum" FROM "Filter" WHERE
    "ID" = TRUE),"Ende_Datum") AND COALESCE((SELECT "EndDatum" FROM "Filter"
    WHERE "ID" = TRUE),"Ende_Datum")
025 ORDER BY "DtStart"
```

Aus dieser Ansicht findet der Export nach iCalendar statt. Die verschiedenen Felder sind durch einen Alias auch für den Nutzer sichtbar mit den Bezeichnungen des Exportformats vorbelegt.

Die Ansicht verbindet die Dateneinträge aus der Tabelle "Termine" (Zeile 11) mit denen aus der Tabelle "Ferien" (Zeile 21), so dass alle Termine exportiert werden. Die Termine werden lediglich durch die Filtertabelle vom Datum her eingegrenzt (Zeile 12 und 13 sowie Zeile 22 und 23).

Die UID aus iCalendar wird über den Primärschlüssel als eindeutiges Kennzeichen erstellt. Dabei wird die Integer-Zahl des Primärschlüssels mit führenden Nullen zu einem achtstelligen Text umgewandelt (Zeile 1).

Ort, Bezeichnung und Bemerkung werden hier durch den Alias nur schon einmal so gekennzeichnet, dass klar ist, auf welcher Felder des iCalendar-Exports sie sich beziehen.

Die Datums- und Zeitbehandlung erfordert hier ein besonderes Vorgehen. Reine Datumswerte werden im Format YYYYMMDD, also z.B. '20200421', erwartet. Kommt ein Zeitfeld hinzu, so wird dieses über ein 'T' angehängt. Zeiten werden in dem Format HHMMSS angegeben, also z.B. '113559'. Ein kompletter Zeitstempel wäre also '20200421T113549'.

Zeile 3 weist bereist mit dem Befehl COALESCE darauf hin, dass eventuell ein Inhalt so, wie er zusammengebaut wird, auch leer sein kann. Zuerst wird die Kombination von Datum und Zeit versucht. Ist diese Leer, so wird nur das Datum in den erforderlichen String umgewandelt.

Die Monatszahl und die Tageszahl können beim Auslesen einstellig sein. Deswegen wird immer eine führende '0' mit diesen Zahlen kombiniert und von rechts aus die beiden Zeichen weiter benutzt. Zweistellige Zahlen sind also von der Konstruktion **`RIGHT ( '0' || EXTRACT(MONTH FROM "Beginn_Datum"), 2 )`** (Zeile 4 und folgende) nicht berührt.

Die Kombination von "DtStart" ist damit abgeschlossen. Bei der Kombination zu "DtEnd" sind hingegen 4 verschiedene Kombinationen möglich (Zeile 7 bis Zeile 10). Zuerst wird die Kombination von "Ende_Datum" und "Ende_Uhrzeit" getestet. Klappt die nicht, dann wird geprüft, ob zwar eine "Ende_Uhrzeit" existiert, aber eben keine "Ende_Datum". Dann wird die Uhrzeit mit dem "Beginn_Datum" kombiniert, weil davon ausgegangen wird, dass der Termin genau am gleichen Tag zu Ende ist. Existiert zwar ein "Ende_Datum" aber keine "Ende_Uhrzeit", so wird nur das Datum übernommen, aber um einen Tag erhöht, da so der gesamte Tag erfasst wird. Existiert weder ein "Ende_Datum" noch eine "Ende_Uhrzeit", so wird nur das "Beginn_Datum" für diesen Zeitstempel übernommen, ebenfalls um einen Tag erhöht. Die iCalendar-Spezifikation sieht bei einer Angabe eines reinen Datumswertes für "DtStart" den Termin als Termin für einen ganzen Tag an, wenn der Tag für "DtEnd" auf dem Folgetag liegt.

In Zeile 11 wird schließlich der Zeitstempel für die Erstellung des Termins über das aktuelle Datum und die aktuelle Zeit zusammengestellt.

Die Daten aus "Termine" werden über den **UNION**-Befehl mit den Daten aus "Ferien" verbunden. Bei der Zusammenstellung der Ferien ab Zeile 16 ist das Verfahren etwas einfacher, da dort die Zeitfelder nicht vorkommen. Das "Beginn_Datum" wird hier direkt übernommen, für das "Ende_Datum" muss wieder ein Tag addiert werden, damit in iCalendar klar ist, dass der Termin für einen ganzen Tag gilt. Fehlt ein "Ende_Datum", so wird stattdessen das "Beginn_Datum" um einen Tag erhöht. Es wird dann von lediglich einem Ferientag ausgegangen.

Ansicht_Kalender_ohne_Ferien_Export

Datenquelle
Tabelle: *Termine, Filter*

Datenziel
Makro: *Export_Start*
Ansicht, Abfrage, Formular, Bericht: keine direkt

```
001 SELECT RIGHT('0000000'||"ID",8) AS "UID",
002 "Ort" AS "Location", "Bezeichnung" AS "Summary", "Bemerkung" AS
    "Description",
003 COALESCE(
004 EXTRACT( YEAR FROM "Beginn_Datum" ) || RIGHT( '0' || EXTRACT( MONTH FROM
    "Beginn_Datum" ), 2 ) || RIGHT( '0' || EXTRACT( DAY FROM "Beginn_Datum" ),
    2 ) || 'T' || RIGHT( '0' || EXTRACT( HOUR FROM "Beginn_Uhrzeit" ), 2 ) ||
    RIGHT( '0' || EXTRACT( MINUTE FROM "Beginn_Uhrzeit" ), 2 ) || RIGHT( '0'
    || EXTRACT( SECOND FROM "Beginn_Uhrzeit" ), 2 ),
005 EXTRACT( YEAR FROM "Beginn_Datum" ) || RIGHT( '0' || EXTRACT( MONTH FROM
    "Beginn_Datum" ), 2 ) || RIGHT( '0' || EXTRACT( DAY FROM "Beginn_Datum" ),
    2 )) AS "DtStart",
006 COALESCE (
007 EXTRACT( YEAR FROM "Ende_Datum" ) || RIGHT( '0' || EXTRACT( MONTH FROM
    "Ende_Datum" ), 2 ) || RIGHT( '0' || EXTRACT( DAY FROM "Ende_Datum" ), 2 )
    || 'T' || RIGHT( '0' || EXTRACT( HOUR FROM "Ende_Uhrzeit" ), 2 ) || RIGHT(
    '0' || EXTRACT( MINUTE FROM "Ende_Uhrzeit" ), 2 ) || RIGHT( '0' ||
    EXTRACT( SECOND FROM "Ende_Uhrzeit" ), 2 ),
008 EXTRACT( YEAR FROM "Beginn_Datum" ) || RIGHT( '0' || EXTRACT( MONTH FROM
    "Beginn_Datum" ), 2 ) || RIGHT( '0' || EXTRACT( DAY FROM "Beginn_Datum" ),
    2 ) || 'T' || RIGHT( '0' || EXTRACT( HOUR FROM "Ende_Uhrzeit" ), 2 ) ||
    RIGHT( '0' || EXTRACT( MINUTE FROM "Ende_Uhrzeit" ), 2 ) || RIGHT( '0' ||
    EXTRACT( SECOND FROM "Ende_Uhrzeit" ), 2 ),
009 EXTRACT( YEAR FROM "Beginn_Datum" ) || RIGHT( '0' || EXTRACT( MONTH FROM
    "Beginn_Datum" ), 2 ) || RIGHT( '0' || EXTRACT( DAY FROM "Beginn_Datum" )
    +1, 2 )) AS "DtEnd",
010 EXTRACT( YEAR FROM CURRENT_DATE ) || RIGHT( '0' || EXTRACT( MONTH FROM
    CURRENT_DATE ), 2 ) || RIGHT( '0' || EXTRACT( DAY FROM CURRENT_DATE ), 2 )
    || 'T' || RIGHT( '0' || EXTRACT( HOUR FROM CURRENT_TIME ), 2 ) ||
    RIGHT( '0' || EXTRACT( MINUTE FROM CURRENT_TIME ), 2 ) || RIGHT( '0' ||
    EXTRACT( SECOND FROM CURRENT_TIME ), 2 ) AS "DtStamp"
011 FROM "Termine"
012 WHERE "Beginn_Datum" BETWEEN COALESCE((SELECT "StartDatum" FROM "Filter"
    WHERE "ID" = TRUE),"Beginn_Datum") AND COALESCE((SELECT "EndDatum" FROM
    "Filter" WHERE "ID" = TRUE),"Beginn_Datum")
013 OR  "Ende_Datum" BETWEEN COALESCE((SELECT "StartDatum" FROM "Filter" WHERE
    "ID" = TRUE),"Ende_Datum") AND COALESCE((SELECT "EndDatum" FROM "Filter"
    WHERE "ID" = TRUE),"Ende_Datum")
014 ORDER BY "DtStart"
```

Diese Ansicht unterscheidet sich von der Ansicht "Ansicht_Kalender_Export" nur darin, dass nicht mit dem Befehl **UNION** eine zweite Abfrage zu den Ferien an die erste Abfrage angehängt wird.

Diese Ansicht dient also zum Export der Daten nach iCalendar ohne den gleichzeitigen Export der Ferien.

Ansicht_Termine_Monat_Jahre

Datenquelle
Tabelle: *Termine, Filter, Ferien*

Datenziel
Ansicht, Abfrage, Formular, Bericht: keine direkt

```
001 SELECT DISTINCT "Jahr", "Jahr"||','||RIGHT('0'||"Monat_Z",2) AS
    "Monat_Zahl",
002 CASE WHEN "Monat_Z"='1' THEN 'Januar' WHEN "Monat_Z"='2' THEN 'Februar'
    WHEN "Monat_Z"='3' THEN 'März' WHEN "Monat_Z"='4' THEN 'April'
    WHEN "Monat_Z"='5' THEN 'Mai' WHEN "Monat_Z"='6' THEN 'Juni'
    WHEN "Monat_Z"='7' THEN 'Juli' WHEN "Monat_Z"='8' THEN 'August'
    WHEN "Monat_Z"='9' THEN 'September' WHEN "Monat_Z"='10' THEN 'Oktober'
    WHEN "Monat_Z"='11' THEN 'November' WHEN "Monat_Z"='12' THEN 'Dezember'
    END AS "Monat_Text",
003 "Monat_Z" FROM
004 (SELECT EXTRACT (YEAR FROM "Beginn_Datum" ) AS "Jahr", EXTRACT (MONTH FROM
    "Beginn_Datum" ) AS "Monat_Z" FROM "Termine"
005 UNION ALL
006 SELECT EXTRACT (YEAR FROM "Ende_Datum" ) AS "Jahr", EXTRACT (MONTH FROM
    "Ende_Datum" ) AS "Monat_Z" FROM "Termine" WHERE NOT "Ende_Datum" IS NULL)
007 AS "Jahre_Monate" ORDER BY "Jahr" ASC, "Monat_Z" ASC
```

Diese Ansicht dient dazu, Listenfelder zur Auswahl der Monate zu beschicken. Nach dieser Auswahl wird in Formularen anschließend der Inhalt gefiltert (Formulare "Termine_Kalenderübersicht" und "Termine_Tabelleneingabe").

Ansicht_Wochen_einzeln

Datenquelle
Tabelle: *tmp_Wochenkalender*

Datenziel
Makro: *Wochenkalender_halb*
Ansicht, Abfrage, Formular, Bericht: keine direkt

```
001 SELECT "tmp_Wochenkalender".*
002 FROM ( SELECT "Woche", MIN( "ID" ) AS "ID_einzeln" FROM
    "tmp_Wochenkalender" AS "tmp_Wochenkalender" GROUP BY "Woche", "Jahr" ) AS
    "Woche_ID", "tmp_Wochenkalender"
003 WHERE "Woche_ID"."ID_einzeln" = "tmp_Wochenkalender"."ID"
```

In der Tabelle "tmp_Wochenkalender" sind für die entsprechenden Berichte Wiederholungen an den Monatsenden eingebaut, sofern eine Woche nicht genau mit dem Monat endet. Dadurch werden Wochen einmal im Vormonat und dann im aktuellen Monat angezeigt.

Für die Erstellung von "tmp_Wochenkalender_halb" müssen aus "tmp_Wochenkalender" alle Datensätze entfernt werden, die eine Verdoppelung der Wochen darstellen. Hier wird in der

Unterabfrage gruppiert nach "Woche" und "Jahr" der kleinere von den Primärschlüsselwerten "ID" über **MIN**("ID") herausgesucht. Wenn also eine Woche doppelt vorkommt, so wird der jeweils zweite Datensatz mit der größeren "ID" nicht mehr in dieser Ansicht angezeigt.

Ansicht_Wochentage

Datenquelle
Tabelle: *tmp_Kalender*

Datenziel
Bericht: *Wochentage_Mensaabo*
Ansicht, Abfrage, Formular: keine direkt

```
001 SELECT COUNT( "ID" ) AS "MoDiMiDoFr",
002 ( SELECT COUNT( "ID" ) FROM "tmp_Kalender" WHERE "Format" IS NULL AND
    DAYOFWEEK( "Datum" ) IN ( 2, 4, 5 ) AND EXTRACT (YEAR FROM "Datum" ) =
    EXTRACT (YEAR FROM "a"."Datum" ) AND EXTRACT (MONTH FROM "Datum" ) =
    EXTRACT (MONTH FROM "a"."Datum" ) ) AS "MoMiDo",
003 ( SELECT COUNT( "ID" ) FROM "tmp_Kalender" WHERE "Format" IS NULL AND
    DAYOFWEEK( "Datum" ) IN ( 2, 3, 4, 5 ) AND EXTRACT (YEAR FROM "Datum" ) =
    EXTRACT (YEAR FROM "a"."Datum" ) AND EXTRACT (MONTH FROM "Datum" ) =
    EXTRACT (MONTH FROM "a"."Datum" ) ) AS "MoDiMiDo",
004 ( SELECT COUNT( "ID" ) FROM "tmp_Kalender" WHERE "Format" IS NULL AND
    DAYOFWEEK( "Datum" ) IN ( 2, 4, 5, 6 ) AND EXTRACT (YEAR FROM "Datum" ) =
    EXTRACT (YEAR FROM "a"."Datum" ) AND EXTRACT (MONTH FROM "Datum" ) =
    EXTRACT (MONTH FROM "a"."Datum" ) ) AS "MoDiMiDoFr",
005 EXTRACT (YEAR FROM "Datum" ) AS "Jahr", EXTRACT (MONTH FROM "Datum" ) AS
    "Monat",
006 CASEWHEN( EXTRACT (MONTH FROM "Datum" ) < 8 ,EXTRACT (YEAR FROM "Datum" )-
    1||'/'|| EXTRACT (YEAR FROM "Datum" ),EXTRACT (YEAR FROM "Datum" )||'/'||
    EXTRACT (YEAR FROM "Datum" )+1) AS "Schuljahr"
007 FROM "tmp_Kalender" AS "a"
008 WHERE "Format" IS NULL AND DAYOFWEEK( "Datum" ) BETWEEN 2 AND 6 GROUP BY
    EXTRACT (YEAR FROM "Datum" ), EXTRACT (MONTH FROM "Datum" )
```

Mit dieser Ansicht wird die Anzahl bestimmter Wochentagskombinationen in einzelnen Monaten ermittelt. Danach soll in einem Bericht ermittelt werden, für wie viele Tage für ein entsprechendes Essensabo in der Schulmensa gezahlt werden muss. Zeile `1` zählt alle Wochentage von Montag bis Freitag, die in der Tabelle "tmp_Kalender" verzeichnet sind (Zeile `7`). Allerdings dürfen diese Tage keinen Formatvermerk haben (Zeile `8`). Dieser Formatvermerk wird bei allen Tagen gemacht, die Feiertage sind oder in den eingetragenen Ferien liegen. Die Wochentage Montag bis Freitag haben bei der internen HSQLDB die Nummern 2 bis 6. Die Ausgabe der Wochentage wird nach der Jahreszahl und der Monatszahl gruppiert.

Die Abfrage für die Wochentage von Montag bis Freitag ist die äußere Abfrage. In den Unterabfragen können nicht mehr Datenzeilen enthalten sein als in den äußeren Abfrage. Da die Unterabfragen nur einen Teil dieser äußeren Tage enthalten wird dies auch nicht passieren. Wenn von Montag bis Freitag kein einziger Tag regulärer Schultag war, dann ist dies auch nicht bei einem einzelnen der Tage der Fall.

In der äußeren Abfrage wird auch noch die Jahreszahl und die Monatszahl abgespeichert. Außerdem ist dort auch noch das Schuljahr in der entsprechenden Schreibweise '2020/2021' angegeben. Die Monate 1 – 7 gehören zum alten Schuljahr. Deswegen wird in Zeile `6` für diese der Jahreswert das Startjahr um ein Jahr heruntergesetzt. Der Monat 'April 2020' gehört schließlich zum Schuljahr '2019/2020'. Bei allen anderen Monaten wird stattdessen das Zieljahr des Schuljahres um 1 Jahr heraufgesetzt.

Auf diese äußere Abfrage greifen jetzt die korrelierenden Unterabfragen zu. Sie richten sich nach den Jahres- und Monatsangaben der äußeren Abfrage. Dazu wurde in der äußeren Abfrage dem Tabellennamen "tmp_Kalender" ein Alias "a" zugewiesen (Zeile 7). Die Unterabfrage in Zeile 2 z.B. ermittelt nur die Anzahl der Tage Montag, Mittwoch und Donnerstag. Nach diesen Tagen wird über den **COUNT**-Befehl gesucht. Das Jahr der Unterabfrage soll mit dem Jahr der äußeren Abfrage übereinstimmen. Der Verweis auf das aktuelle Jahr der äußeren Abfrage, das gerade in dieser Zeile stehen wird, ist das "a" vor dem Datum, aus dem das Jahr ermittelt wird. Auch der Monat der Unterabfrage soll mit dem Monat der äußeren Abfrage übereinstimmen. Auch hier wieder das "a" als Verweis auf die äußere Abfrage.

In Zeile 3 und Zeile 4 sind entsprechende korrelierende Unterabfragen für andere Tageskombinationen enthalten.

Abfragen

Liste_Gruppe_Multi

Datenquelle
Ansicht: *Ansicht_Gruppe_multi*

Datenziel
Bericht: *Liste_Gruppe_multi, Liste_Gruppe_multi_1HJ, Liste_Gruppe_multi_2HJ*

```
001 SELECT "a".*,
002 ( SELECT COUNT( DISTINCT ( "W1" ) ) FROM "Ansicht_Gruppe_multi" WHERE "W1"
    <= "a"."W1" ) AS "Woche"
003 FROM "Ansicht_Gruppe_multi" AS "a"
```

Der "Ansicht_Gruppe_multi" wird hier lediglich ein Zähler für die Wochen hinzugefügt. Damit können in der Übersicht der Berichte auch die jeweiligen Schulwochen gekennzeichnet werden, die ja unabhängig von Kalenderwochen sind. Zum einen beginnen die Schulwochen mitten im Jahr, zum anderen werden die Wochen, die komplett in den Ferien liegen, nicht mitgezählt.

Der laufende Zähler wird durch eine korrelierende Unterabfrage bewerkstelligt. Die "Ansicht_Gruppe_multi" wird mit einem Alias "a" versehen (Zeile 3). Dadurch kann sich der Zähler der Unterabfrage immer nach dem aktuellen Datensatz der äußeren Abfrage richten. "W1" in der "Ansicht_Gruppe_multi" ist dabei eine Kombination aus Jahreszahl und Wochenzahl. Über diese Kombination sind Wochen, die zum Anfang eines Folgejahres liegen, für den Zähler größer als Wochen, die im aktuellen Jahr liegen.

Termine_Gruppe_multi

Datenquelle
Tabelle: *Termine*

Datenziel
Formular: *Terminart_Termin* (SubTermine), *Termine_Kalenderübersicht* (MainForm), *Termine_Monat_Terminart* (MainForm), *Termine_Tabelleneingabe* (MainForm)

```
001 SELECT "Termine".*,
002 BITAND( "Gruppe_multi", 1 ) "Jg5", BITAND( "Gruppe_multi", 2 ) "Jg6",
    BITAND( "Gruppe_multi", 4 ) "Jg7", BITAND( "Gruppe_multi", 8 ) "Jg8",
    BITAND( "Gruppe_multi", 16 ) "Jg9", BITAND( "Gruppe_multi", 32 ) "Jg10",
```

```
    BITAND( "Gruppe_multi", 64 ) "Jg11", BITAND( "Gruppe_multi", 128 ) "Jg12",
    BITAND( "Gruppe_multi", 256 ) "Jg13",
003 EXTRACT( YEAR FROM "Beginn_Datum" ) AS "Jahr", EXTRACT( MONTH FROM
    "Beginn_Datum" ) AS "Monat"
004 FROM "Termine" ORDER BY "Beginn_Datum" ASC
```

Die einzelnen Checkboxen in den Formularen sind nicht direkt mit einem Feld in der Tabelle "Termine" verbunden. Stattdessen sind ihre Werte nach dem Verfahren 20, 21, 22, 23 usw. als Einserwerte des Binärsystems festgelegt. Alle Werte zusammen werden in einem Feld "Gruppe_multi" als Summe gespeichert. Die Abfrage soll dazu dienen, dass die Werte aber auch nach dem Abspeichern anschließend von den Checkboxen wieder angezeigt werden können.

Angenommen in dem Feld "Gruppe_multi" befindet sich die Zahl '7', dann soll ja hier klar werden, dass nur 1, 2 und 4 in diese Zahl hinein passen – die Zahlen also, die den Jahrgängen 5, 6 und 7 in den Formularen zugeordnet waren. Die Zahl 7 hat in der Binärschreibweise die Binärfolge 0111. Wird jetzt nach dem 7 Jahrgang gesucht, so muss die Funktion **BITAND** bei der Addition von 0111 + 0100 genau den Wert für den Jahrgang 7 ermitteln. 0100 ist dann genau 22. Und damit wird eben die Checkbox für den Jahrgang 7 als ausgewählt angezeigt.

Terminuebersicht

Datenquelle
Tabelle: *Termine*

Datenziel
Formular: *Terminart_Termin* (SubTerminMonat, SubTerminOrt), *Termine_Monat_Terminart* (SubTermineMonat, SubTerminart)

```
001 SELECT "Termine".*,
002 EXTRACT( MONTH FROM "Beginn_Datum" ) AS "Monat",
003 EXTRACT( YEAR FROM "Beginn_Datum" ) AS "Jahr",
004 FROM "Termine"
005 ORDER BY "Beginn_Datum" ASC, "Beginn_Uhrzeit" ASC
```

Die Verbindung zu den Unterformularen, vor allem dem Unterformular, das die Termine in Abhängigkeit vom Monat zeigt, wird über den Monatswert und den Jahreswert festgelegt. Deswegen wird hier neben allen Werten aus der Tabelle "Termine" die Monatszahl (Zeile 2) und die Jahreszahl (Zeile 3) ermittelt.

Wochenkalender_Mo_Fr

Datenquelle
Tabelle: *tmp_Wochenkalender*

Datenziel
Bericht: *Monatskalender_Mo_Fr*

Der Bericht soll nur die Wochentage Montag bis Freitag darstellen. Fällt der Monatsanfang auf einen Samstag oder Sonntag, so soll die entsprechende Woche nicht in dem neuen Monat auftauchen. Deshalb ist eine Abfrage nötig, die die Monatsübergänge anders regelt als beim Bericht für ganze Wochen.

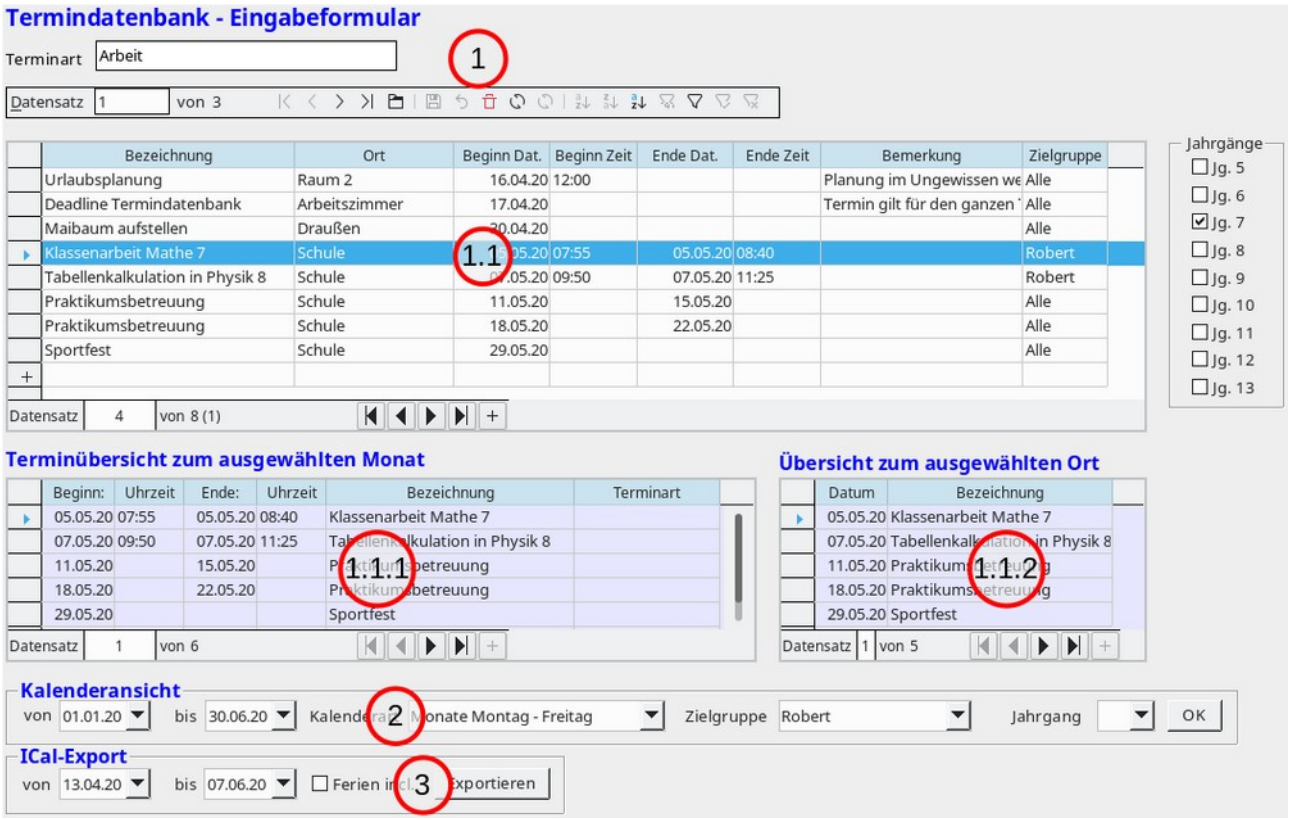
```
001 SELECT "tmp_Wochenkalender".*
002 FROM "tmp_Wochenkalender"
003 WHERE EXTRACT( MONTH FROM "D_1" ) = "Monat_Zahl"
004 OR EXTRACT( MONTH FROM "D_2" ) = "Monat_Zahl"
005 OR EXTRACT( MONTH FROM "D_3" ) = "Monat_Zahl"
```

```
006 OR EXTRACT( MONTH FROM "D_4" ) = "Monat_Zahl"
007 OR EXTRACT( MONTH FROM "D_5" ) = "Monat_Zahl"
```

Formulare

Die Formulare erfüllen grundsätzlich alle den gleichen Zweck: Befüllen der Tabelle "Termine" mit Inhalten, Übersicht über die Termine sowie Ausgabe der Termine in den verschiedenen Berichten. Sie zeigen nur, wie unterschiedlich Formulare für gleiche Zwecke gestaltet werden können.

Terminart_Termin



1	MainForm (Tabelle: *Terminart*)	**1.1**	SubTermine (Abfrage: *Termine_Gruppe_multi*)	**1.1.1**	SubTerminMonat (Abfrage: *Terminuebersicht*)
				1.1.2	SubTerminOrt (Abfrage: *Terminuebersicht*)
2	Berichte (Tabelle: *Filter*)				
3	Export (Tabelle: *Filter*)				

Makros in Formulareigenschaften		
keine		

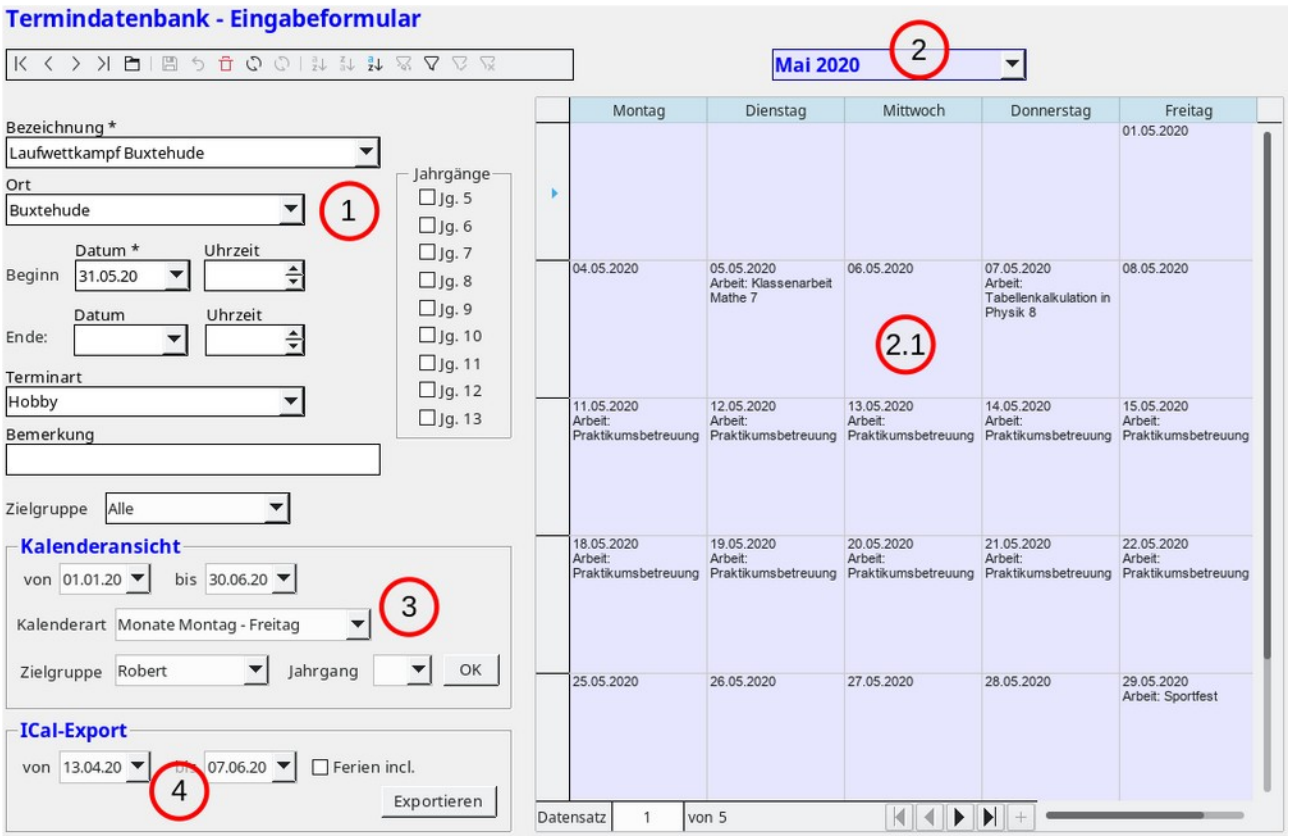
Makros in Feldeigenschaften		
1.1	Beginn Dat. (Text modifiziert)	Eingabe.*DatenAktualisieren_Terminort_Termine*
1.1	Checkboxen "Jahrgänge" (Status geändert)	Eingabe.*GruppeMultiSpeichern*
2	Schaltfläche OK (Aktion ausführen)	Abfragen.*Bericht_Start*
3	Schaltfläche Exportieren (Aktion ausführen)	Export.*Export_Start*

Das Formular ist so organisiert, dass die Terminart vorgewählt wird. Es zeigt im Unterformular "SubTermine" die zu der Terminart passenden Einträge an. Im Unterformular werden auch in der Hauptsache die Daten eingegeben. Die gruppierten Checkboxen "Jahrgänge" sind ebenfalls Teil dieses Unterformulars.

Die Formulare "SubTerminMonat" und "SubTerminOrt" zeigen zu dem ausgewählten Datensatz eine entsprechende Übersicht an.

Die Formulare "Berichte" und "Export" sind nicht für die Eingabe von Daten bestimmt. Sie starten lediglich die Berichtsausführung bzw. den Export der Daten.

Termine_Kalenderübersicht



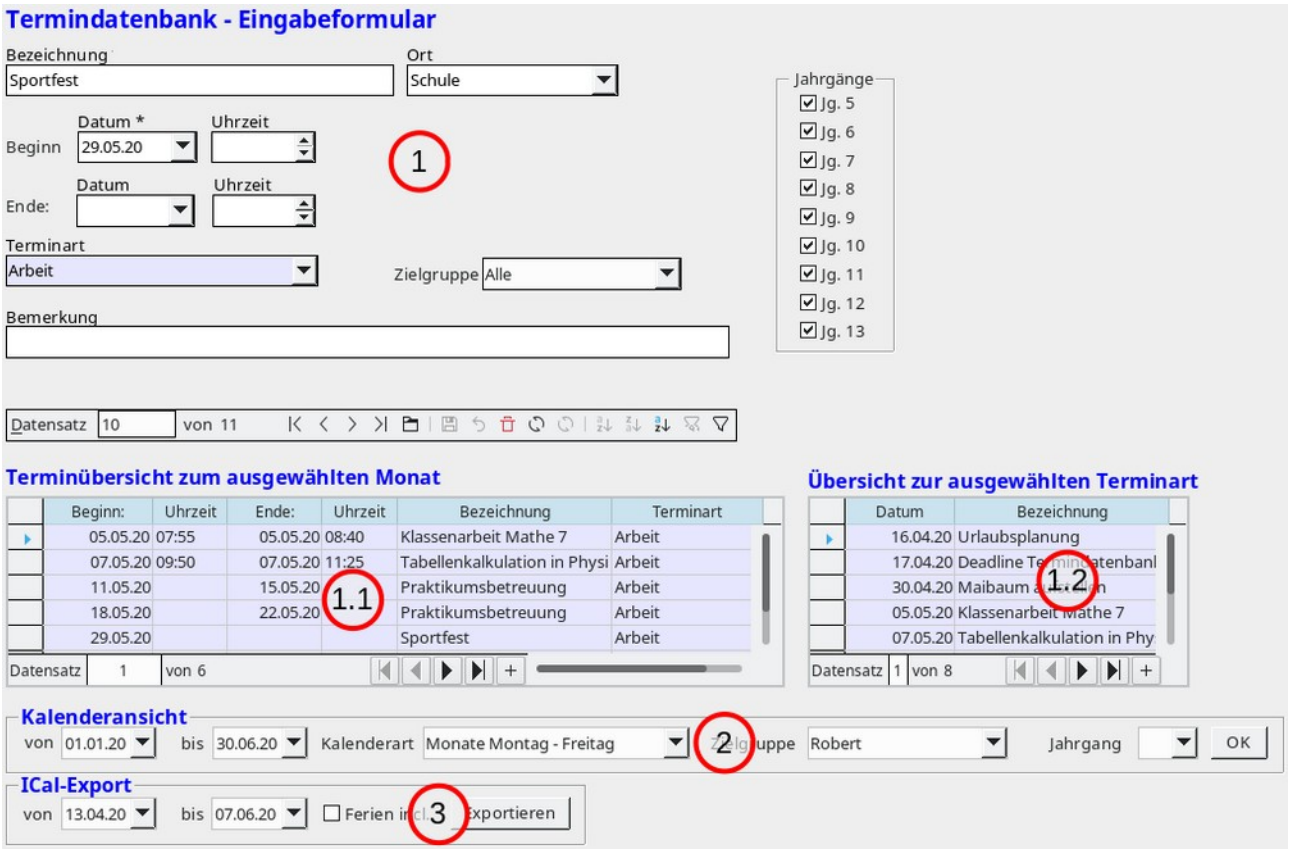
1	MainForm (Abfrage: *Termine_Gruppe_multi*)		
2	Monat (Tabelle: *Filter*)	**2.1**	Ansicht (Tabelle: *tmp_Monat*)
3	Berichte (Tabelle: *Filter*)		
4	Export (Tabelle: *Filter*)		

Makros in Formulareigenschaften		
1	Beim Laden	Eingabe.*LetzterDatensatz_Termine_Kalender*
1	Beim erneuten Laden	Eingabe.*Monatsansicht_aktualisieren*
1	Nach dem Datensatzwechsel	Eingabe.*Monatsansicht_aktualisieren*
Makros in Feldeigenschaften		
1	Kombinationsfeld "Bezeichnung" (Fokusverlust)	Eingabe.*Termindaten_uebertragen*
1	Beginn Datum (Fokuserhalt)	Eingabe.*Anfangswert_Datum*
1	Beginn Datum (Fokusverlust)	Eingabe.*Monatsansicht_aktualisieren*
1	Checkboxen Jahrgänge (Status geändert)	Eingabe.*GruppeMultiSpeichern*
2	Listenfeld Monat (Modifiziert)	Eingabe.*Monat_aktualisieren*
3	Schaltfläche OK (Aktion ausführen)	Abfragen.*Bericht_Start*
4	Schaltfläche Exportieren (Aktion ausführen)	Export.*Export_Start*

Dieses Formular bietet eine Eingabemöglichkeit in einfachen Eingabefeldern. Dadurch geht die Übersicht auf die Daten verloren. Deswegen ist neben den Eingabefeldern eine Übersicht für den jeweils aktuellen Monat eingeblendet. An der entsprechenden Makroübersicht wird deutlich, dass diese Darstellung ohne Makros nicht zu bewältigen ist. Schließlich soll der aktuelle Monat während der Eingabe nach der Auswahl des Datums zum Terminbeginn gegebenenfalls neu eingestellt werden. Ansonsten würde ja die Übersicht nur passen, wenn durch die Termine im Hauptformular gescrollt wird, nicht aber, wenn ein Termin in einem neuen Monat eingegeben wird.

Die Formulare "Berichte" und "Export" sind nicht für die Eingabe von Daten bestimmt. Sie starten lediglich die Berichtsausführung bzw. den Export der Daten.

Termine_Monat_Terminart



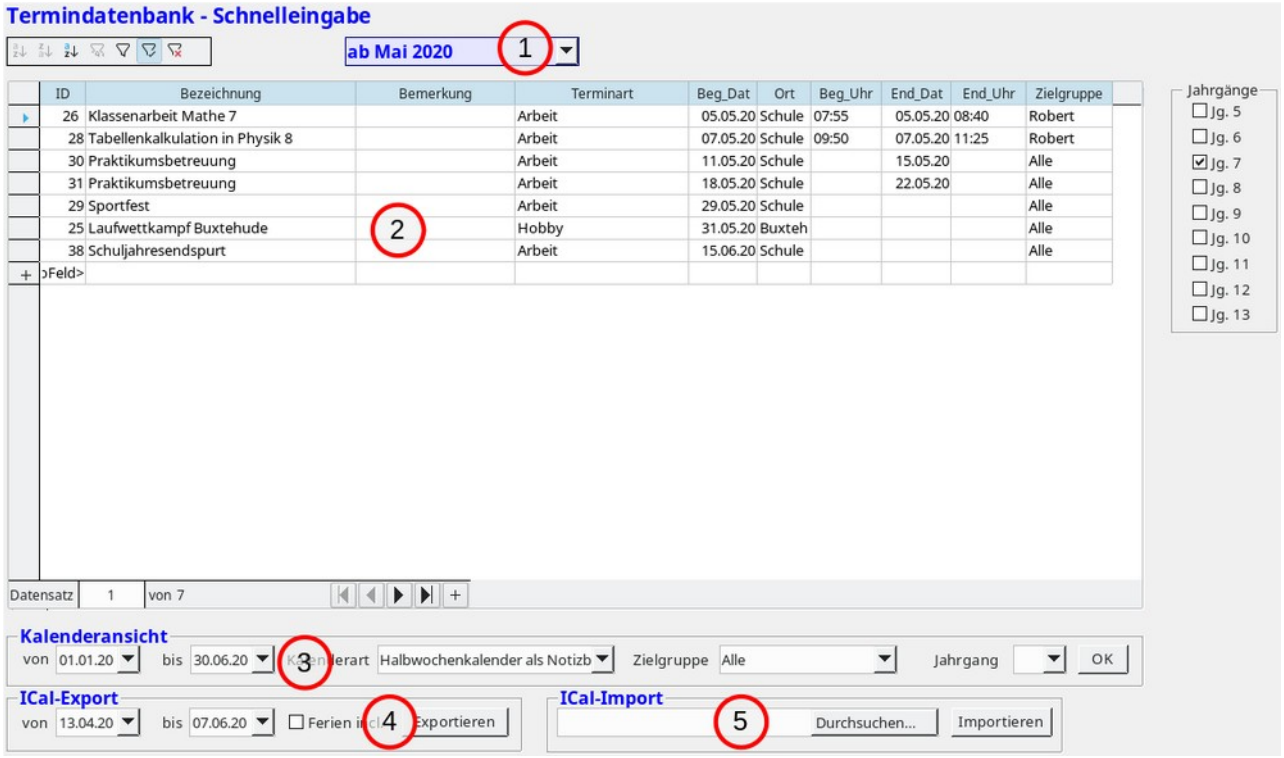
1	MainForm (Abfrage: *Termine_Gruppe_multi)*	**1.1**	SubTermineMonat (Abfrage: *Terminuebersicht*)
		1.2	SubTerminart (Abfrage: *Terminuebersicht*)
2	Berichte (Tabelle: *Filter*)		
3	Export (Tabelle: *Filter*)		

Makros in Formulareigenschaften		
1	Beim Laden	Eingabe.*LetzterDatensatz*
Makros in Feldeigenschaften		
1	Beginn Datum (Text modifiziert)	Eingabe.*DatenAktualisieren_Termine*
1	Listenfeld "Terminart" (Modifiziert)	Eingabe.*DatenAktualisieren_Termine*
1	Checkboxen Jahrgänge (Status geändert)	Eingabe.*GruppeMultiSpeichern*
2	Schaltfläche OK (Aktion ausführen)	Abfragen.*Bericht_Start*
3	Schaltfläche Exportieren (Aktion ausführen)	Export.*Export_Start*

Das Hauptformular "MainForm" dient hier wieder zur Dateneingabe in einfache Formularfelder. Die Übersicht zu den aktuell eingegebenen Daten vermitteln die Unterformulare "SubTermine-Monat" und "SubTerminart". Auch hier muss mehr mit Makros gearbeitet werden, da während der Eingabe in die Formularfelder des Hauptformular die Unterformulare ggf. neu eingestellt werden müssen. Sonst würde die Übersicht auf die Daten nur etwas bringen, wenn durch die Daten gescrollt wird, nicht aber, wenn neue Daten eingegeben werden.

Die Formulare "Berichte" und "Export" sind nicht für die Eingabe von Daten bestimmt. Sie starten lediglich die Berichtsausführung bzw. den Export der Daten.

Termine_Tabelleneingabe



1	Monat (Tabelle: *Filter*)
2	MainForm (Abfrage: *Termine_Gruppe_multi*)
3	Berichte (Tabelle: *Filter*)
4	Export (Tabelle: *Filter*)

Makros in Formulareigenschaften		
keine		
Makros in Feldeigenschaften		
1	Listenfeld Monat (Modifiziert)	Filter.*Filter*_Abgleich
2	Checkboxen Jahrgänge (Status geändert)	Eingabe.*GruppeMultiSpeichern*
3	Schaltfläche OK (Aktion ausführen)	Abfragen.*Bericht_Start*
4	Schaltfläche Exportieren (Aktion ausführen)	Export.*Export_Start*
5	Schaltfläche Importieren (Aktion ausführen)	Import.*Import_Start*

Dieses Formular hat sich in der Praxis als das herausgestellt, was überwiegend genutzt wird. Nach einer Vorauswahl des Monats zeigt das Formular "MainForm" alle Daten zu dem aktuellen Monat an. Die Daten können gegebenenfalls über eine kleine Navigationsleiste sortiert und neben der Filterung nach dem Monat noch zusätzlich gefiltert werden.

Die Filterung nach dem Monat erfolgt über den Formularfilter, der in den Eigenschaften von "MainForm" steckt:

```
001 (EXTRACT( YEAR FROM "Beginn_Datum" )||','|| RIGHT('0'||EXTRACT( MONTH FROM
    "Beginn_Datum" ),2) >= COALESCE((SELECT "Jahr_Monat" FROM "Filter" WHERE
    "ID" = TRUE),EXTRACT( YEAR FROM "Beginn_Datum" )||','|| RIGHT('0'||
    EXTRACT( MONTH FROM "Beginn_Datum" ),2)))
```

Wenn in der Tabelle "Filter" der Kombinationswert aus Jahr und Monat steht, dann wird dieser mit dem entsprechenden Eintrag in der Abfrage verglichen. Ist der Eintrag leer, so werden alle Datensätze angezeigt, die einen Eintrag für das "Beginn_Datum" haben. Und da dieser Eintrag nicht leer sein darf sind das dann tatsächlich alle Datensätze.

Der Einsatz dieses Formularfilters hat den Vorteil, dass der Filter gegebenenfalls auch ausgeschaltet werden kann. Ist dies nicht erforderlich, so kann der Filter auch direkt in die entsprechende Abfrage eingebaut werden.

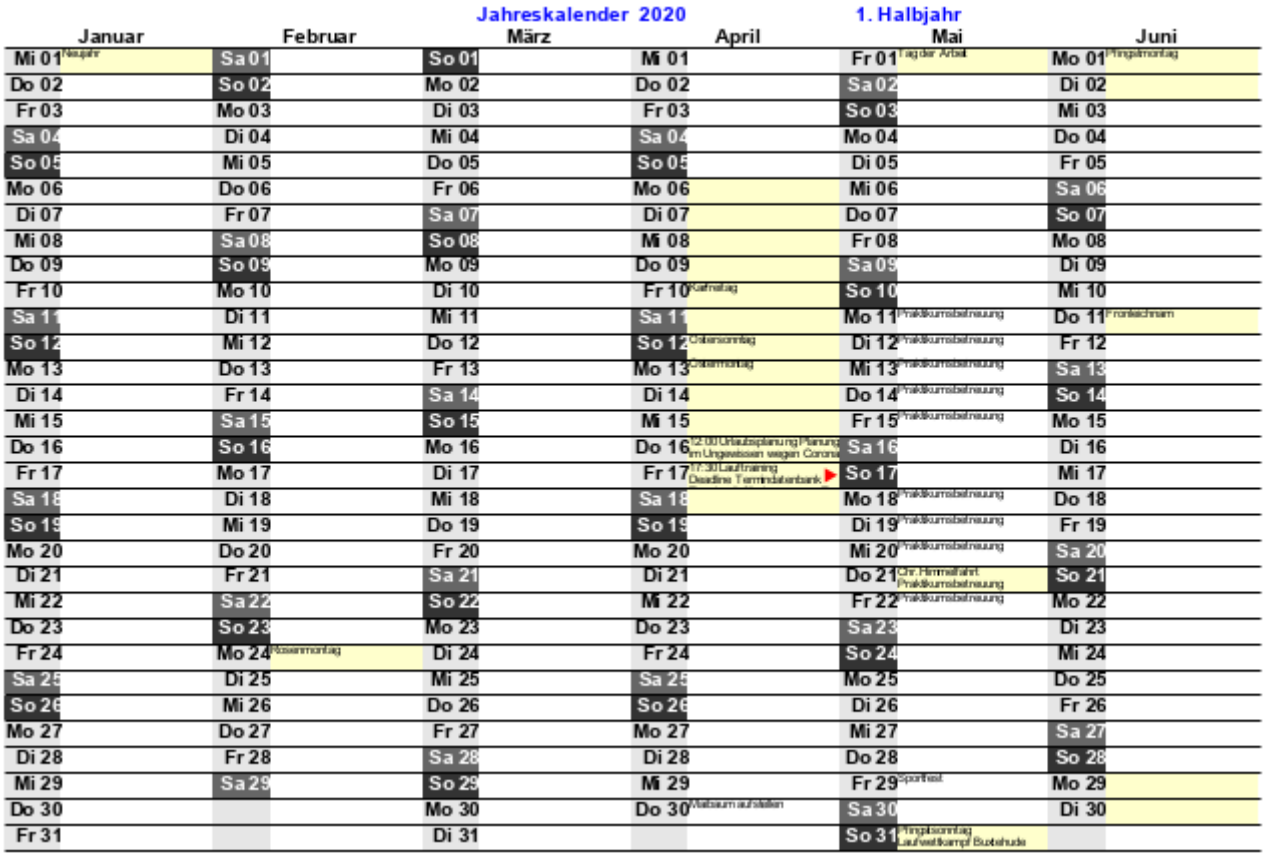
Die Formulare "Berichte" und "Export" sind nicht für die Eingabe von Daten bestimmt. Sie starten lediglich die Berichtsausführung bzw. den Export sowie in diesem Formular auch den Import der Daten.

Werden Daten importiert, so werden sie anschließend über den durch das Makro erstellten Formularfilter direkt in der Tabellenansicht des Formulars zum weiteren bearbeiten angezeigt.

Berichte

Die Berichte stellen den eigentlichen Einsatzpunkt dieser Datenbank dar. Es geht hier weniger um eine Terminverwaltung wie im privaten Terminkalender. Da sind die speziellen Programme weit bedienungsfreundlicher. Hier geht es darum, dass zu unterschiedlichen Anlässen unterschiedliche Kalenderlayouts benötigt und ausgedruckt werden können.

Halbjahreskalender_A4_quer



Jahreskalender 2020 — 1. Halbjahr

Januar	Februar	März	April	Mai	Juni
Mi 01 Neujahr	Sa 01	So 01	Mi 01	Fr 01 Tag der Arbeit	Mo 01 Pfingstmontag
Do 02	So 02	Mo 02	Do 02	Sa 02	Di 02
Fr 03	Mo 03	Di 03	Fr 03	So 03	Mi 03
Sa 04	Di 04	Mi 04	Sa 04	Mo 04	Do 04
So 05	Mi 05	Do 05	So 05	Di 05	Fr 05
Mo 06	Do 06	Fr 06	Mo 06	Mi 06	Sa 06
Di 07	Fr 07	Sa 07	Di 07	Do 07	So 07
Mi 08	Sa 08	So 08	Mi 08	Fr 08	Mo 08
Do 09	So 09	Mo 09	Do 09	Sa 09	Di 09
Fr 10	Mo 10	Di 10	Fr 10 Karfreitag	So 10	Mi 10
Sa 11	Di 11	Mi 11	Sa 11	Mo 11 Praktikumsbetreuung	Do 11 Fronleichnam
So 12	Mi 12	Do 12	So 12 Ostersonntag	Di 12 Praktikumsbetreuung	Fr 12
Mo 13	Do 13	Fr 13	Mo 13 Ostermontag	Mi 13 Praktikumsbetreuung	Sa 13
Di 14	Fr 14	Sa 14	Di 14	Do 14 Praktikumsbetreuung	So 14
Mi 15	Sa 15	So 15	Mi 15	Fr 15 Praktikumsbetreuung	Mo 15
Do 16	So 16	Mo 16	Do 16 12:00 Urlaubsplanung Planung im Ungewissen wegen Corona	Sa 16	Di 16
Fr 17	Mo 17	Di 17	Fr 17 17:30 Lauftraining Deadline Termindatenbank	So 17	Mi 17
Sa 18	Di 18	Mi 18	Sa 18	Mo 18 Praktikumsbetreuung	Do 18
So 19	Mi 19	Do 19	So 19	Di 19 Praktikumsbetreuung	Fr 19
Mo 20	Do 20	Fr 20	Mo 20	Mi 20 Praktikumsbetreuung	Sa 20
Di 21	Fr 21	Sa 21	Di 21	Do 21 Chr. Himmelfahrt Praktikumsbetreuung	So 21
Mi 22	Sa 22	So 22	Mi 22	Fr 22 Praktikumsbetreuung	Mo 22
Do 23	So 23	Mo 23	Do 23	Sa 23	Di 23
Fr 24	Mo 24 Rosenmontag	Di 24	Fr 24	So 24	Mi 24
Sa 25	Di 25	Mi 25	Sa 25	Mo 25	Do 25
So 26	Mi 26	Do 26	So 26	Di 26	Fr 26
Mo 27	Do 27	Fr 27	Mo 27	Mi 27	Sa 27
Di 28	Fr 28	Sa 28	Di 28	Do 28	So 28
Mi 29	Sa 29	So 29	Mi 29	Fr 29 Sportfest	Mo 29
Do 30		Mo 30	Do 30 Arbeitsamt aufsuchen	Sa 30	Di 30
Fr 31		Di 31		So 31 Pfingstsonntag Laufwettkampf Butzbhude	

Datenquelle
Tabelle *tmp_Halbjahreskalender*

Der Halbjahreskalender besteht aus 2 DIN A4-Blättern, auf denen jeweils ein halbes Kalenderjahr abgebildet wird. Unabhängig von der Vorauswahl des Datums beim Druck wird zu dem gewählten Startdatum der entsprechende Jahreskalender ausgegeben.

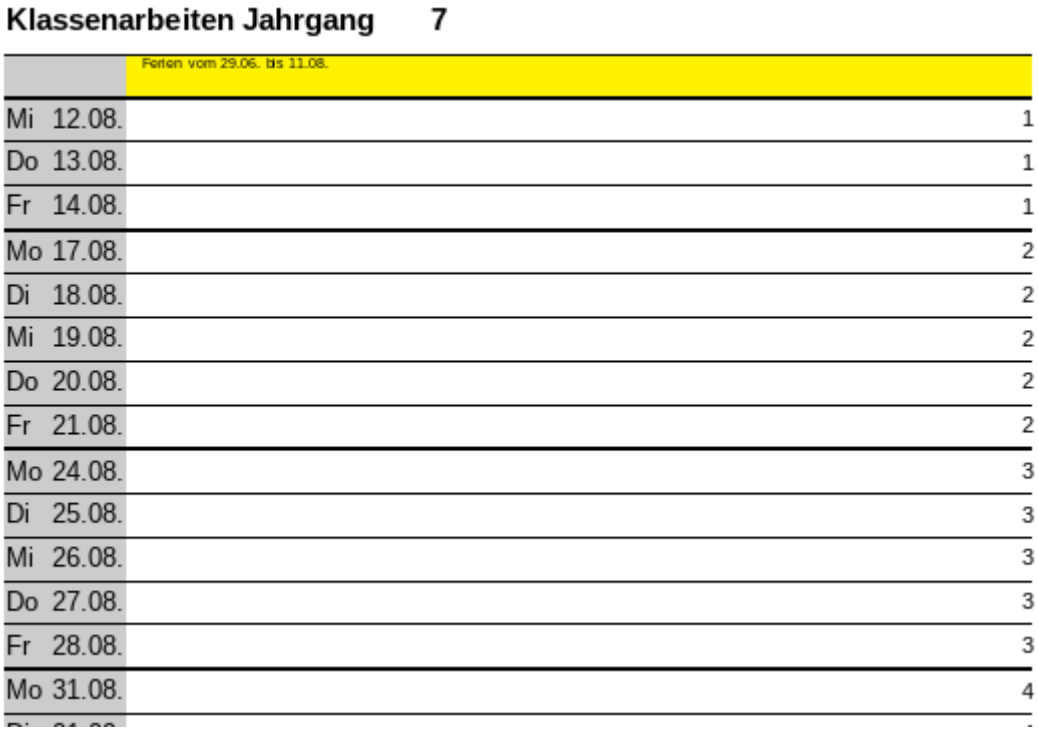
Auf dem Ausdruck ist zu erkennen, dass der Platz nicht allzu viele Einträge zulässt. Da aber die Berichte ja sowieso im Writer dargestellt werden kann z.B. der Textüberlauf am 17. April anschließend noch bearbeitet werden. Hier kann entweder der Text gekürzt oder die Schriftart verkleiner werden. Auf keinen Fall kann die Zeilenhöhe vergrößert werden, da sonst der Ausdruck von der ersten Seite auf der 2. Seite fortgesetzt würde.

In gelber Farbe sind die freien Tage gekennzeichnet. Dies sind entweder Ferientage oder Feiertage.

Liste_Gruppe_multi

Diese Liste bietet Platz für Termineinträge, die nicht in den Ferien liegen. Die Ferien werden nur als Anhaltspunkt vermerkt.

In der Praxis werden in dieser Liste die Klassenarbeiten innerhalb eines Jahrgangs koordiniert, so dass nicht zu viele Klassenarbeiten in einer Woche oder an direkt aufeinander folgenden Tagen geschrieben werden.



Klassenarbeiten Jahrgang 7

	Ferien vom 29.06. bis 11.08.
Mi 12.08.	1
Do 13.08.	1
Fr 14.08.	1
Mo 17.08.	2
Di 18.08.	2
Mi 19.08.	2
Do 20.08.	2
Fr 21.08.	2
Mo 24.08.	3
Di 25.08.	3
Mi 26.08.	3
Do 27.08.	3
Fr 28.08.	3
Mo 31.08.	4

Datenquelle
Abfrage *Liste_Gruppe_Multi*

Zusätzlich zu dieser Liste können noch separate Listen erstellt werden, die durch die Eingabe eines Datums getrennt werden. Der Grund für die Trennung war rein praktischer Natur: Wurden die Ausdrucke für die Listen untereinander gehängt, so war gerade einmal genug Platz für ein Halbjahr. Die Listen eines ganzen Schuljahres brauchten den Platz von 7 Seiten, was einer Gesamtlänge von ca. 210 cm entspricht.

Liste_Gruppe_multi_1HJ

Hiermit wird eine Liste für das erste Halbjahr ausgegeben. Der Trenner dafür wird aus "Filter"."Halbjahrende" ausgelesen.

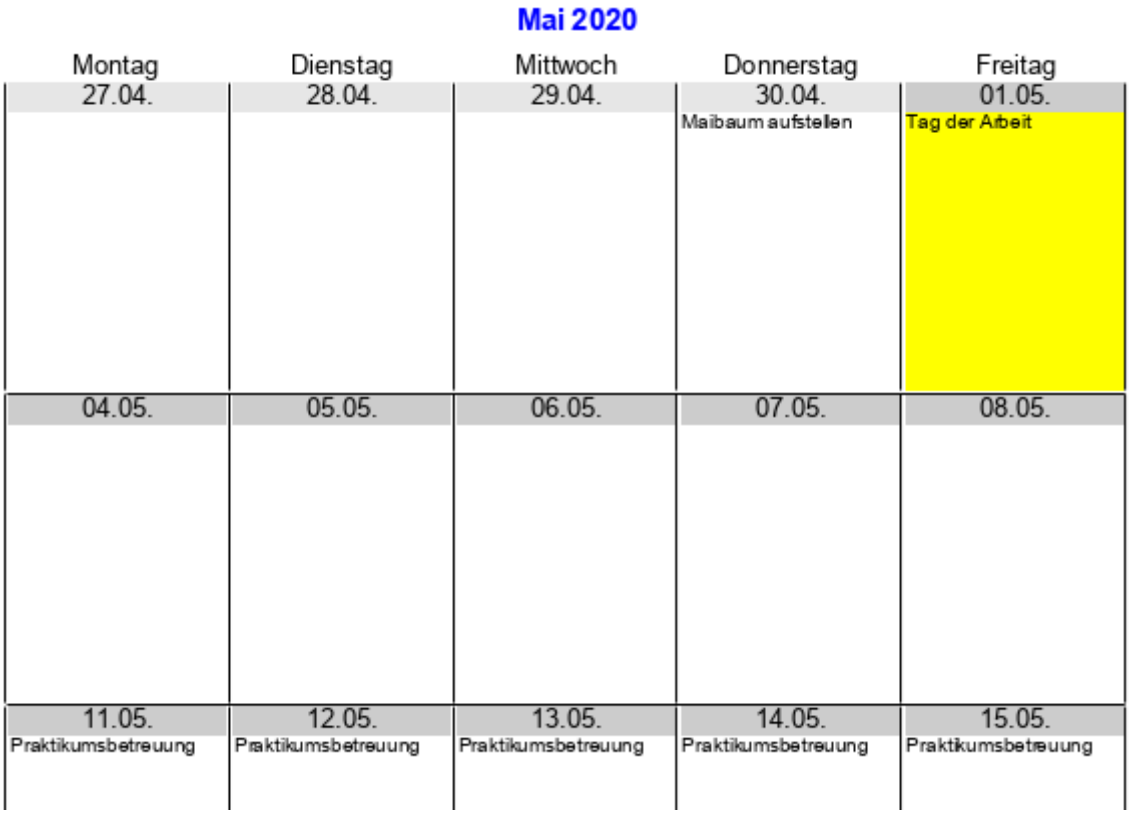
Datenquelle
Abfrage: *Liste_Gruppe_Multi*
Filter: `"Sort" <= (SELECT "Halbjahrende" FROM "Filter" WHERE "ID" = TRUE)`

Liste_Gruppe_multi_2HJ

Hiermit wird eine Liste für das zweite Halbjahr ausgegeben. Der Trenner dafür wird aus "Filter"."Halbjahrende" ausgelesen.

Datenquelle
Abfrage: *Liste_Gruppe_Multi*
Filter: `"Sort" > (SELECT "Halbjahrende" FROM "Filter" WHERE "ID" = TRUE)`

Monatskalender_Mo_Fr



Datenquelle
Abfrage: *Wochenkalender_Mo_Fr*

Dieser Kalender war der ursprüngliche Kalenderausdruck des Projekts. Die doch recht dunkele Farbgebung für den Feiertag war dem Drucker geschuldet. Die Drucke des Kalenders erfolgen nämlich über einen sogenannten Risographen, der von der Qualität der Graustufendarstellung nicht an Laserdrucker heran kommt.

Jedes Feld hat genügend Platz für mehrere Einträge am Tag. Durchgehende Einträge werden dabei immer nach oben gesetzt, so wie das hier bei den Einträgen ab dem 11.05. zu sehen ist.

Sollte einmal der Platz nicht ausreichen, so besteht nach der Erstellung durch den Report Bilder noch die Möglichkeit, die Einträge entsprechend anzupassen. Vielleicht wird etwas an den For-

mulierungen geändert oder einfach die Einträge für ein Feld mit einer etwas kleineren Schriftart ausgedruckt.

Monatskalender_Mo_So

April 2020

Montag	Dienstag	Mittwoch	Donnerstag	Freitag	Samstag	Sonntag
		01.04.20	02.04.20	03.04.20	04.04.20	05.04.20
06.04.20 Ferien	07.04.20 Ferien	08.04.20 Ferien	09.04.20 Ferien	10.04.20 Karfreitag	11.04.20 Ferien	12.04.20 Ostersonntag
13.04.20 Ostermontag	14.04.20 Ferien	15.04.20 Ferien	16.04.20 Ferien 12:00 Urlaubsplanung Planung im Ungewissen wegen Corona	17.04.20 Ferien 17:30 Lauftraining Deadline Termindatenbank Termin gilt für den ganzen Tag	18.04.20 Ferien	19.04.20
20.04.20	21.04.20	22.04.20	23.04.20	24.04.20	25.04.20	26.04.20

Datenquelle
Tabelle: *tmp_Wochenkalender*

Dieser Monatskalender stellt nur eine Erweiterung des vorherigen dar, falls eben auch die Sicht auf den Samstag und den Sonntag gewünscht sind. Dafür steht dann aber eben etwas weniger Platz für den Text zur Verfügung.

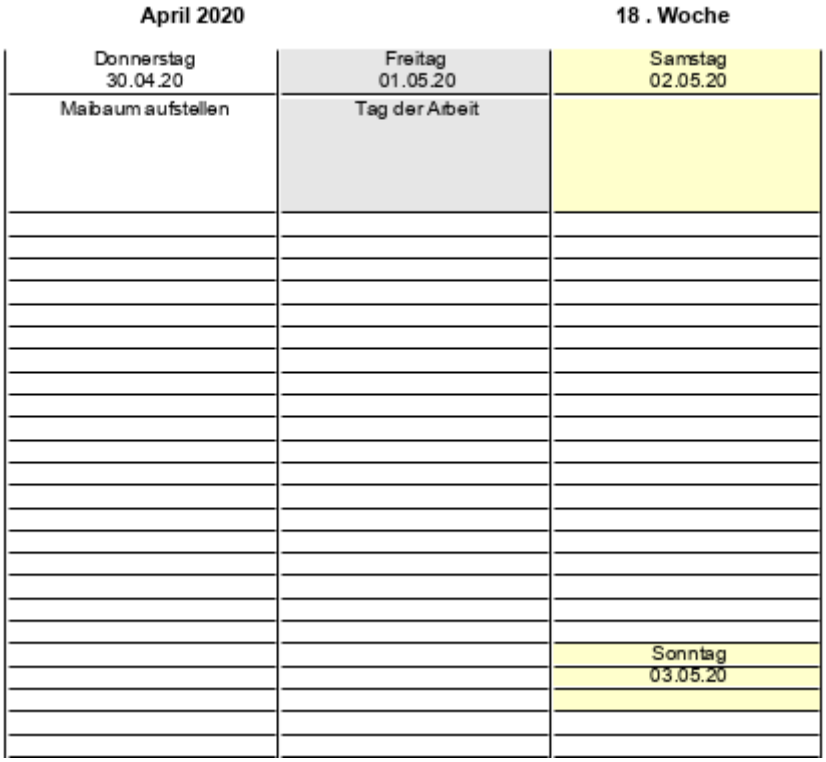
Wochenkalender_halb

April 2020 **18. Woche**

Montag 27.04.20	Dienstag 28.04.20	Mittwoch 29.04.20

Jede Seite in diesem Kalender stellt eine halbe Woche dar. Unterhalb der Datumsangabe werden in dem großen Feld die Termine eingetragen, die in der Datenbank stehen. Wie in der untenstehenden Ansicht zu sehen werden dort Feiertage und Wochenenden farbig noch besonders hervorgehoben.

Unterhalb der Termineinträge sind dann viele Zeilen zum Eintragen von irgendwelchen Ereignissen, die tagesaktuell verarbeitet werden müssen. In der täglichen Anwendung dient dieser Plan zur Vertretungs- und Raumplanung an der Schule. Die Einträge werden hier handschriftlich gemacht und anschließend in das Vertretungs- und Planungsprogramm übernommen.



April 2020		18 . Woche
Donnerstag 30.04.20	Freitag 01.05.20	Samstag 02.05.20
Maibaum aufstellen	Tag der Arbeit	
		Sonntag 03.05.20

Datenquelle
Tabelle: *tmp_Wochenkalender_halb*

Wochentage_Mensaabo

Schuljahr 2020/2021

Jahr	Monat	MoDiMiDoFr	MoDiMiDo	MoMiDoFr	MoMiDo
2020	8	14	11	12	9
2020	9	22	18	17	13
2020	10	12	9	10	7
2020	11	21	17	17	13
2020	12	16	13	12	9
2021	1	17	13	14	10
2021	2	19	15	15	11
2021	3	20	16	16	12
2021	4	15	12	12	9
2021	5	18	14	15	11
2021	6	21	17	16	12
2021	7	2	1	2	1
Summe Schuljahr:		**197**	**156**	**158**	**117**

Datenquelle
Ansicht: *Ansicht_Wochentage*

Die Mensa der Schule bietet unterschiedliche Abos an. Um den Preis vernünftig bestimmen zu können wird mit dem Programm einfach geschaut, wie viele Tage im ganzen Jahr denn in dieser Abozeit liegen. Dabei werden die Ferientage, Feiertage und Wochenenden nicht mit gezählt.

Zeitplan

Zeitplan für das Jahr 2020 **11.04.20**

Arbeit	
Urlaubsplanung | Planung im Ungewissen wegen Corona	16.04.20
Deadline Termindatenbank | Termin gilt für den ganzen Tag	17.04.20
Maibaum aufstellen	30.04.20
Praktikumsbetreuung	11.05.20- 15.05.20 18.05.20- 22.05.20
Sportfest	29.05.20
Hobby	
Lauftraining	17.04.20

Datenquelle
Tabelle: *tmp_Terminuebersicht*

Dieser Zeitplan ist eine Tischvorlage für Konferenzen. Da wird noch einmal kurz zusammengefasst, was denn wann im Jahr geplant ist.

Makros

AA_Variablen

In diesem Modul sind neben globalen Variablen, die hier nicht einzeln aufgeführt werden, einige allgemeine Funktionen enthalten.

Datenbankstart

Aufruf aus
Datenbankdatei: **Extras → Anpassen → Ereignisse → Dokument öffnen** oder: **→ Ansicht wurde erzeugt**

Benötigt	
Makro:	*Datenbankbackup*

Globale Variablen sind in allen Modulen verfügbar. Damit die Verbindung zur Datenbank nicht in allen möglichen Prozeduren erneut überprüft werden muss wird sie einmal für die gesamte Laufzeit in einer globalen Variablen abgespeichert. Bei unterschiedlichen LO-Versionen hat sich gezeigt, dass die Verbindung mit den Ereignissen des Datenbankdokumentes manchmal problematisch ist. **Ansicht wurde erzeugt** fand z.B. in LO 6.1.5.2 unter Kubuntu gar nicht statt. Stattdessen startete das Makro sehr wohl mit **Dokument öffnen**.

Vor der Definition der Datenquelle wird noch in Zeile 3 eine Kopie der Datenbank in den Back-up-Ordner geschrieben.

Außerdem wird noch global das Startdatum mit der ersten Formularöffnung festgelegt.

```
001  GLOBAL oVerbindung AS OBJECT
002  GLOBAL daStartDat AS DATE
```

```
001  SUB Datenbankstart
002     DIM oDatenquelle AS OBJECT
003     Datenbankbackup(10)
004     oDatenquelle = thisDatabaseDocument.CurrentController
005     IF NOT (oDatenquelle.isConnected()) THEN oDatenquelle.connect()
006     oVerbindung = oDatenquelle.ActiveConnection()
007  END SUB
```

OfficeVersion

Aufruf aus	
Makro:	*Datumswert, Listenfeld_Text, ID_Ermittlung*

Manche Funktionen ändern sich im Laufe der Zeit. So wird z.B. seit LO 4.1 anders auf die Werte eines Listenfeldes zugegriffen oder seit 4.1.2 Datumsfelder anders ausgelesen. Um die Datenbank kompatibel zu älteren Versionen zu halten muss also die Office-Version gegebenenfalls bestimmt werden können.

```
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
```

Der **nodepath** in Zeile 8 stellt den Einstieg in **Extras → Optionen → LibreOffice → Erweitert → Experteneinstellungen** dar. Der Wert aus Zeile 9 ist dann der Zugriff auf die in Zeile 11 genauer bezeichneten Elemente.

Mit **ooName** wird der Unterschied zwischen LO und AOO ermittelt. Der String, der über diesen Parameter ermittelt wird, heißt bei LO "LibreOffice".

Mit **ooSetupVersionAboutBox** wird der Wert zu der jeweiligen Version ermittelt, also z.B. 6.4.3.2 bei LibreOffice

Datumswert

Aufruf aus	
Makro:	*Bericht_Start, Anfangswert_Datum, Monatsansicht_aktualisieren*

Benötigt	
Makro:	*OfficeVersion*

Die Behandlung des Datumswertes hat sich in Abhängigkeit von der LO-Version geändert.

```
001 FUNCTION Datumswert(oFeld AS OBJECT) AS DATE
002    DIM b()
003    a() = OfficeVersion()
004    b = Split(a(1),".")
005    IF a(0) = "LibreOffice" AND (b(0) = 4 AND b(1) > 0) OR b(0) > 4 THEN
006       DIM stMonat AS STRING
007       DIM stTag AS STRING
008       stMonat = Right("0" & Trim(Str(oFeld.CurrentValue.Month)),2)
009       stTag =  Right("0" & Trim(Str(oFeld.CurrentValue.Day)),2)
010       Datumswert = CDateFromIso(oFeld.CurrentValue.Year & stMonat & stTag)
011    ELSE
012       Datumswert = CDateFromIso(oFeld.getCurrentValue)
013    END IF
014 END FUNCTION
```

In Zeile 4 wird der zweite Eintrag aus der Funktion **OfficeVersion()** in seine Bestandteile zerlegt. Die Zahlen für die Version sind durch Punkte getrennt und mit den einzelnen Ziffern kann die Version bestimmt werden.

In Zeile 5 wird zuerst nachgeschaut, ob bei dem aus der Funktion **OfficeVersion()** zurückgegebenen ersten Eintrag um "LibreOffice" handelt. "AOO" würde also direkt auf die Zeile 10 weiter verwiesen.

Anschließend wird in Zeile 5 nach dem zweiten Eintrag geschaut. Der erste Eintag des neuen Arrays b muss eine 4 und der zweite Eintrag eine Zahl größer als 0 enthalten. Damit sind alle Versionen LO 4.* abgedeckt. Kommt es zu höheren Versionen, so wird dies durch die Bedingung hinter dem **OR** abgedeckt. Der erste Wert des Arrays ist dann größer als 4.

Sind diese Bedingungen erfüllt, so wird das Datum nicht über den **CurrentValue** ausgelesen, sondern stattdessen als Struct aus einzelnen Feldern dieses Wertes. Ab LO 4.1 existiert hier der Eintag **Year**, **Month** und **Day**.

Listenfeld_Text

Aufruf aus	
Makro:	*Bericht_Start*

Benötigt	
Makro:	*OfficeVersion*

```
001 FUNCTION Listenfeld_Text(oFeld AS OBJECT) AS STRING
002    DIM b()
003    a() = OfficeVersion()
004    b = Split(a(1),".")
005    IF a(0) = "LibreOffice" AND (b(0) = 4 AND b(1) > 0) OR b(0) > 4 THEN
006       Listenfeld_Text = oFeld.StringItemList(oFeld.SelectedItems(0))
007    ELSE
```

```
008         Listenfeld_Text = oFeld.CurrentValue()
009      END IF
010   END FUNCTION
```

Der angezeigte Text eines Listenfeldes muss abhängig von der LO-Version ermittelt werden. Während er früher über **CurrentValue** wie in Zeile 8 zu erreichen war muss jetzt in Zeile 6 in der Liste des Listenfeldes der gerade angezeigte Wert ermittelt werden. Als **CurrentValue** wird in den aktuellen Fassungen der Wert gesehen, der schließlich in der Datenbank gespeichert werden soll.

ID_Ermittlung

Aufruf aus
Makro: *Bericht_Start*

Benötigt
Makro: *OfficeVersion*

```
001   FUNCTION ID_Ermittlung(oFeld AS OBJECT) AS INTEGER
002      DIM stInhalt AS STRING
003      DIM b()
004      a() = OfficeVersion()
005      b = Split(a(1),".")
006      IF a(0) = "LibreOffice" AND (b(0) = 4 AND b(1) > 0) OR b(0) > 4 THEN
007         stInhalt = oFeld.CurrentValue
008      ELSE
009         stInhalt = oFeld.ValueItemList(oFeld.SelectedItems(0))
010      END IF
011      IF IsEmpty(stInhalt) THEN
012         ID_Ermittlung = -1
013      ELSE
014         ID_Ermittlung = Cint(stInhalt)
015      END IF
016   END FUNCTION
```

Die Ermittlung des abzuspeichernden Inhaltes aus dem Listenfeld, meist der Primärschlüsselwertes des angezeigten Inhaltes, erfolgt neu über den **CurrentValue** in Zeile 6. Die ID-Felder in den Tabellen dieser Datenbank haben alle einen Zahlenwert. Deswegen gibt diese Funktion den Wert auch als Integer-Wert aus. Das hat bei einem leeren Feld allerdings den Nachteil, dass eine Zahlenvariable daraus dann '0' machen würde. Deshalb wird in Zeile 12 in diesem Fall der Wert '-1' als Funktionswert weitergegeben.

Abfragen

Kalender

Aufruf aus
Makro: *Bericht_Start*

Benötigt
Makro: *FeiertageArray, Date_2_SQLDate*
Tabelle: *tmp_Kalender, Termine, Ferien*

```
001   SUB Kalender(inJahr AS INTEGER, inFerien AS INTEGER, stTermin AS STRING)
002      DIM stDatum AS STRING
003      DIM stDat_Beginn AS STRING
004      DIM stDat_Ende AS STRING
005      DIM stUpdate AS STRING
```

```
006     DIM stZeit AS STRING
007     DIM i AS INTEGER
008     DIM d AS DATE
009     DIM d_neu AS DATE
010     DIM d_end AS DATE
011     DIM stBezeichnung AS STRING
012     DIM stBemerkung AS STRING
013     inJahresschleife = inJahr
014     oSQL_Anweisung = oVerbindung.createStatement()
015     oSQL_Anweisung1 = oVerbindung.createStatement()
016     oSQL_Anweisung2 = oVerbindung.createStatement()
017     stSql = "DELETE FROM ""tmp_Kalender"""
018     oSQL_Anweisung.executeUpdate(stSql)
019     oSQL_Anweisung.executeQuery("ALTER TABLE ""tmp_Kalender"" ALTER COLUMN ""ID""
           RESTART WITH 1")
020     FOR inJahr=inJahresschleife TO inJahresschleife+1
021        Feiertage = FeiertageArray(inJahr)
022        d = "01.01."+inJahr
023        d_end = "01.01."+(inJahr+1)
024        stDatum = Date_2_SQLDate(d)
025        d_neu=d
026        WHILE d_neu < d_end
027           stSql = "INSERT INTO ""tmp_Kalender"" (""Datum"") VALUES ('"+stDatum+"') "
028           oSQL_Anweisung.executeUpdate(stSql)
029           d_neu=DateAdd("d",1,d_neu)
030           stDatum = Date_2_SQLDate(d_neu)
031        WEND
032        stSql = "SELECT ""Beginn_Datum"", ""Ende_Datum"" FROM ""Ferien"" WHERE
              YEAR(""Beginn_Datum"") ='"+inJahr+"' OR YEAR(""Ende_Datum"")='"+inJahr+"'
              ORDER BY ""Beginn_Datum"" ASC, ""Ende_Datum"" DESC"
033        oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
034        WHILE oAbfrageergebnis.next
035           stDat_Beginn = oAbfrageergebnis.getString(1)
036           stDat_Ende = oAbfrageergebnis.getString(2)
037           IF inFerien = 1 THEN
038              stUpdate = """Bezeichnung"" ='Ferien', ""Format"" = '1'"
039           ELSE
040              stUpdate = """Format"" = '1'"
041           END IF
042           stSql1 = "UPDATE ""tmp_Kalender"" SET "+stUpdate+" WHERE ""Datum"" =
              '"+stDat_Beginn+"'"
043           oSQL_Anweisung1.executeUpdate(stSql1)
044           IF stDat_Ende <>"" AND stDat_Ende <> stDat_Beginn THEN
045              d_neu=DateAdd("d",1,CDate(stDat_Beginn))
046              WHILE d_neu <= CDate(stDat_Ende)
047                 stDatum = Date_2_SQLDate(d_neu)
048                 stSql1 = "UPDATE ""tmp_Kalender"" SET "+stUpdate+" WHERE ""Datum"" =
                    '"+stDatum+"'"
049                 oSQL_Anweisung1.executeUpdate(stSql1)
050                 d_neu=DateAdd("d",1,d_neu)
051              WEND
052           END IF
053        WEND
054        FOR i=0 TO 14
055           stDatum = Feiertage(i,1)
056           stBezeichnung = Feiertage(i,0)
057           stSql = "UPDATE ""tmp_Kalender"" SET ""Bezeichnung"" ='"+stBezeichnung+"',
              ""Format"" = '1' WHERE ""Datum"" = '"+stDatum+"'"
058           oSQL_Anweisung.executeUpdate(stSql)
059        NEXT
060        stSql = "SELECT ""Bezeichnung"", ""Beginn_Datum"", ""Ende_Datum"",
              ""Bemerkung"", ""Beginn_Uhrzeit"" FROM ""Termine"" WHERE ""zieID"" IN
              ("+stTermin+") AND (YEAR(""Beginn_Datum"")='"+inJahr+"' OR
              YEAR(""Ende_Datum"")='"+inJahr+"') ORDER BY ""Beginn_Datum"" ASC,
              ""Ende_Datum"" DESC, ""Beginn_Uhrzeit"" ASC"
061        oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
```

```
062         WHILE oAbfrageergebnis.next
063            stBezeichnung = oAbfrageergebnis.getString(1)
064            stBezeichnung_Dat_Ende = oAbfrageergebnis.getString(1)
065            stDat_Beginn = oAbfrageergebnis.getString(2)
066            stDat_Ende = oAbfrageergebnis.getString(3)
067            stBemerkung = oAbfrageergebnis.getString(4)
068            stZeit = oAbfrageergebnis.getString(5)
069            stZeit = left(stZeit,5)
070            IF stBemerkung <> "" THEN
071               stBezeichnung = stBezeichnung + " " + stBemerkung
072               stBezeichnung_Dat_Ende = stBezeichnung_Dat_Ende + " " + stBemerkung
073            END IF
074            IF stZeit <> "" THEN
075               stBezeichnung = stZeit + " " + stBezeichnung
076               stBezeichnung_Dat_Ende = stZeit + " " + stBezeichnung_Dat_Ende
077            END IF
078            stSql1 = "SELECT ""Bezeichnung"" FROM ""tmp_Kalender"" WHERE ""Datum"" =
                  '"+stDat_Beginn+"'"
079            oAbfrageergebnis1 = oSQL_Anweisung1.executeQuery(stSql1)
080            WHILE oAbfrageergebnis1.next
081               IF oAbfrageergebnis1.getString(1) <>"" THEN
082                  stBezeichnung = oAbfrageergebnis1.getString(1) + CHR(13) +
                        stBezeichnung
083               END IF
084            WEND
085            stSql1 = "UPDATE ""tmp_Kalender"" SET ""Bezeichnung"" ='"+stBezeichnung+"'
                  WHERE ""Datum"" = '"+stDat_Beginn+"'"
086            oSQL_Anweisung1.executeUpdate(stSql1)
087            IF stDat_Ende <>"" AND stDat_Ende <> stDat_Beginn THEN
088               d_neu=DateAdd("d",1,CDate(stDat_Beginn))
089               WHILE d_neu <= CDate(stDat_Ende)
090                  stDatum = Date_2_SQLDate(d_neu)
091                  stSql2 = "SELECT ""Bezeichnung"" FROM ""tmp_Kalender"" WHERE
                        ""Datum"" = '"+stDatum+"'"
092                  oAbfrageergebnis2 = oSQL_Anweisung2.executeQuery(stSql2)
093                  WHILE oAbfrageergebnis2.next
094                     IF oAbfrageergebnis2.getString(1) <>"" THEN
095                        stBezeichnung = oAbfrageergebnis2.getString(1) + CHR(13) +
                              stBezeichnung_Dat_Ende
096                     ELSE
097                        stBezeichnung = stBezeichnung_Dat_Ende
098                     END IF
099                  WEND
100                  stSql1 = "UPDATE ""tmp_Kalender"" SET ""Bezeichnung"" ='"
                        +stBezeichnung+ "' WHERE ""Datum"" = '"+stDatum+"'"
101                  oSQL_Anweisung1.executeUpdate(stSql1)
102                  d_neu=DateAdd("d",1,d_neu)
103               WEND
104            END IF
105         WEND
106      NEXT
107   END SUB
```

Diese Prozedur erwartet beim Aufruf eine Jahreszahl das Startjahr (**inJahr**), eine 1 oder 0 für den Schriftzug "Ferien" in den Feldern für die Terminbezeichnung (**inFerien**) sowie eine Benennung der Gruppe aus der Tabelle "Zielgruppe"."Termingruppe", für die der Termin gedacht ist (**stTermin**).

Die Kalenderzusammenstellungen verlaufen vom Prinzip her nach dem gleichen Schema.

9. Zuerst wird die bestehende Tabelle, hier "tmp_Kalender", von allen Daten geleert (Zeile 17).
10. Danach wird der automatisch hoch zählende Primärschlüsselwert wird auf den Startwert '1' eingestellt (Zeile 19).

11. Anschließend wird in einer Schleife für das angegebene Startjahr und das Folgejahr der geforderte Inhalt ausgelesen (ab Zeile 20 bis Zeile 106). Diese Schleife ist erforderlich, da nicht Jahreskalender für ein Kalenderjahr sondern Jahreskalender für ein Schuljahr erstellt werden.
12. In einer hierin enthaltenen Schleife wird zuerst die Tabelle komplett mit Datumswerten für das ganze Jahr versehen (Zeile 26 bis Zeile 31)
13. Abhängig von der Kalenderart werden zuerst die Ferien (Zeile 32 bis 53) oder die Feiertage (Zeile 54 bis Zeile 59) in die Tabelle übertragen. Die zweite Schleife überschreibt dabei gegebenenfalls Werte der ersten Schleife. Es steht dann eben der Name des Feiertags und nicht, falls gewünscht (Zeile 38), "Ferien" in den Feldern. Es gibt Berichte wie den Bericht "Liste_Gruppe_multi", bei dem die Auflistung von Feiertagen innerhalb von Ferien zu unerwünschten Nebeneffekten führt. Dort sind dann die Schleifen vertauscht.
14. Zum Schluss, hier ab Zeile 60, wird dann die jeweilige Tabelle mit Daten gefüllt.

Die Datenabfrage ab Zeile 60 hier im Einzelnen:

Die Abfrage in Zeile 60 wird eingegrenzt durch die Zielgruppe, für die die Daten zusammengestellt werden, sowie das "Beginn_Datum" bzw. das "Ende_Datum", das in dem entsprechenden Jahr liegen soll. Das Abfrageergebnis wird für das "Beginn_Datum" und die "Beginn_Uhrzeit" aufsteigend sortiert, für das "Ende_Datum" allerdings absteigend. Damit soll erreicht werden, dass bei gleichem "Beginn_Datum" der Datensatz zuerst ausgelesen wird, der über einen längeren Zeitraum läuft. Dadurch können Einträge z.B. im Wochenkalender auf gleicher Höhe positioniert werden, wenn sie mehrmals hintereinander vorkommen.

Damit auch die Bezeichnung für das "Ende_Datum" erhalten bleibt, wird das Feld "Bezeichnung" doppelt ausgelesen (Zeile 63 und Zeile 64). Alle anderen Felder werden als Strings ausgelesen, da die Formatierung direkt so wieder für das Einfügen in die Zieltabelle geeignet ist. Die ausgelesene Zeit wird dabei in Zeile 69 auf die Angabe von Stunden und Minuten zusammengekürzt.

Kommt eine Bemerkung vor, so wird sie mit der Bezeichnung gekoppelt (Zeile 70 bis Zeile 73). Der Eintrag für eine Zeit wird gegebenenfalls der Bemerkung vorangestellt (Zeile 74 bis Zeile 77).

Steht bereits ein Eintrag zu dem Datum in der Tabelle "tmp_Kalender", so wird dieser Eintrag ausgelesen und der neue Eintrag an den vorhergehenden mit einem Zeilenumbruch angehängt (Zeile 76 bis Zeile 86).

Nur wenn das "Ende_Datum" nicht leer und nicht gleich dem "Beginn_Datum" ist wird mit weiteren Einträgen für jedes dazwischenliegende Datum die Tabelle "tmp_Kalender" weiter aufgefüllt. In Zeile 88 muss dafür zuerst das Startdatum in eine für die Funktion **DateAdd** verarbeitbare Date-Variable mit der Funktion **CDate** umgewandelt werden. Dann können auch einzelne Tage addiert werden.

Die Datumsschleife läuft dann von Zeile 89 bis Zeile 103. Wieder muss zuerst geklärt werden, ob in "tmp_Kalender" schon ein Eintrag für das gewünschte Datum existiert. Unter den Umständen muss der neue Eintrag mit einem Zeilenumbruch an den vorhergehenden angehängt werden (Zeile 91 bis Zeile 99). Ansonsten wird nur einfach der aktuelle Eintrag mit einem Update in die Tabelle "tmp_Kalender" eingefügt (Zeile 100 bis Zeile 101). In Zeile 95 muss dann auch zum ersten Mal die in Zeile 64 erstellte zweite Variable **stBezeichnung_Dat_Ende** eingesetzt werden, da die Variable **stBezeichnung** laufend überschrieben wird.

Monatskalender

Aufruf aus
Makro: *Bericht_Start*

Benötigt
Makro: *Date_2_Monat, Date_2_SQLDate*
Tabelle: *tmp_Wochenkalender, tmp_Kalender*

```
001 SUB Monatskalender(daStart AS DATE, daEnde AS DATE)
002    DIM i AS INTEGER
003    DIM inMonat AS INTEGER
004    DIM inWoche AS INTEGER
005    DIM inMonat_Sort AS INTEGER
006    DIM inTag AS INTEGER
007    DIM inDd AS INTEGER
008    DIM inMd AS INTEGER
009    DIM inYd AS INTEGER
010    DIM inW AS INTEGER
011    DIM inID AS INTEGER
012    DIM inWcount AS INTEGER
013    DIM inArrayCount AS INTEGER
014    DIM inMonat_2 AS INTEGER
015    DIM inJahr AS INTEGER
016    DIM d AS DATE
017    DIM d_neu AS DATE
018    DIM d_end AS DATE
019    DIM stBezeichnung AS STRING
020    DIM stBezeichnung_2 AS STRING
021    DIM stDatum AS STRING
022    DIM stDat AS STRING
023    DIM stDat_2 AS STRING
024    DIM stMonat AS STRING
025    DIM stForm AS STRING
026    d = YEAR(daStart) & "-" & MONTH(daStart) & "-01"
027    inJahr = Year(d)
028    d_end = daEnde
029    d_neu = DateAdd("d",1,d)
030    stDatum = Date_2_SQLDate(d)
031    oSQL_Anweisung = oVerbindung.createStatement()
032    oSQL_Anweisung1 = oVerbindung.createStatement()
033    oSQL_Anweisung2 = oVerbindung.createStatement()
034    stSql = "DELETE FROM ""tmp_Wochenkalender"""
035    oSQL_Anweisung.executeUpdate(stSql)
036    oSQL_Anweisung.executeQuery("ALTER TABLE ""tmp_Wochenkalender"" ALTER COLUMN
          ""ID"" RESTART WITH 1")
037    stMonat = Date_2_Monat(d)
038    inTag = DatePart("w",d,2)
039    inWoche = DatePart("ww",d,2,2)
040    inMonat = Trim(Month(d))
041    inMonatSort = inMonat
042    WHILE d_neu < d_end
043       stSql1 = "INSERT INTO ""tmp_Wochenkalender"" (""Jahr"",""Woche"",
             ""D_"+inTag+""", ""Monat_Text"", ""Monat_Zahl"", ""Monat_Sort"") VALUES
             ('"+inJahr+"', '"+inWoche+"', '"+stDatum+"', '"+stMonat+"', '"+inMonat+"',
             '"+inMonatSort+"')"
044       oSQL_Anweisung1.executeUpdate(stSql1)
045       stSql = "CALL IDENTITY()"
046       oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
047       WHILE oAbfrageergebnis.next
048          inID = oAbfrageergebnis.getInt(1)
049       WEND
050       FOR i = inTag+1 TO 7
051          stDatum = Date_2_SQLDate(d_neu)
052          stSql2 = "UPDATE ""tmp_Wochenkalender"" SET ""D_"+i+""" ='"+stDatum+"'
                WHERE ""Woche"" = '"+inWoche+"' AND ""Jahr"" = '"+inJahr+"'"
053          oSQL_Anweisung2.executeUpdate(stSql2)
054          d_neu=DateAdd("d",1,d_neu)
055       NEXT
056       stDatum = Date_2_SQLDate(d_neu)
```

```
057         stMonat = Date_2_Monat(d_neu)
058         inTag = 1
059         inMonat_2 = Trim(Month(d_neu))
060         inMonatSort = inMonat_2
061         inJahr = Year(d_neu)
062         IF Year(d) < inJahr THEN
063            inMonatSort = inMonat_2+12
064         END IF
065         IF inMonat_2 <> inMonat AND d_neu < d_end THEN
066            stSql = "INSERT INTO ""tmp_Wochenkalender"" (""Jahr"",""Woche"",
                  ""Monat_Text"",""Monat_Zahl"",""Monat_Sort"",""D_1"",""D_2"",""D_3"",
                  ""D_4"",""D_5"",""D_6"",""D_7"") VALUES ('"+inJahr+"','"+inWoche+"',
                  '"+stMonat+"','"+inMonat_2+"','"+inMonatSort+"',"
067            FOR i = 1 TO 7
068               stSql = stSql + "(SELECT ""D_"+i+""" FROM ""tmp_Wochenkalender"" WHERE
                     ID = '"+inID+"'),"
069            NEXT
070            stSql = Left(stSql , Len(stSql) -1 ) &")"
071            oSQL_Anweisung.executeUpdate(stSql)
072         END IF
073         inWoche = DatePart("ww",d_neu,2,2)
074         inMonat = Trim(Month(d_neu))
075         inMonatSort = inMonat
076         IF Year(d) < inJahr  THEN
077            inMonatSort = inMonat + 12
078         END IF
079         d_neu=DateAdd("d",1,d_neu)
080      WEND
081      stSql = "SELECT ""Bezeichnung"", DAYOFWEEK( ""Datum""), ""Datum"", ""Format""
            FROM ""tmp_Kalender"""
082      oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
083      WHILE oAbfrageergebnis.next
084         stBezeichnung = oAbfrageergebnis.getString(1)
085         inTag = oAbfrageergebnis.getInt(2)
086         inID = inTag - 1
087         IF inID = 0 THEN
088            inID = 7
089         END IF
090         stDat = oAbfrageergebnis.getString(3)
091         stForm = oAbfrageergebnis.getString(4)
092         stSql1 = "UPDATE ""tmp_Wochenkalender"" SET ""T_"+inID+"""
               ='"+stBezeichnung+"', ""F_"+inID+""" ='"+stForm+"' WHERE ""D_"+inID+""" =
               '"+stDat+"'"
093         oSQL_Anweisung1.executeUpdate(stSql1)
094      WEND
095   END SUB
```

Diese Prozedur erwartet beim Aufruf lediglich eine Angabe des Startdatums **daStart** und eine Angabe des Enddatums **daEnde**.

In Zeile 26 wird direkt aus **daStart** lediglich Monat und Jahr herausgezogen und der Beginn auf den Monatsersten gelegt. Es ist also ohne Belang, ob da nun der 15.4.20 steht – daraus wird dann einfach der 1.4.20 zusammengesetzt – natürlich in der für Datenbanken üblichen Schreibweise '2020-04-01'.

Nach dem Leeren der Tabelle "tmp_Wochenkalender" werden zunächst einmal verschiedene zusätzliche Variablen ermittelt. Die ist in Zeile 37 der deutschsprachige Monatsname (**stMonat**), in Zeile 38 die Tageszahl (**inTag** mit Montag als dem ersten Wochentag (Parameter 2)), in Zeile 39 die Wochenzahl (**inWoche** mit Montag als dem ersten Wochentag und der ersten gezählten Woche, die 4 Tage im aktuellen Jahr hat, (Parameter 2,2)) und in Zeile 40 die Monatszahl (**inMonat**).

Von Zeile 42 bis Zeile 80 verläuft eine Schleife, die so lange neue Datumseinträge in die Tabelle "tmp_Wochenkalender" einfügt, bis das Enddatum erreicht wird. Dabei wird zum Startdatum zum Schluss der Schleife immer ein Tag addiert.

Zuerst wird in der Schleife ein Datensatz eingefügt, der alle Daten für den ersten Tag des Monats enthält (Zeile 43 und Zeile 44), der nicht unbedingt auch erster Tag der Woche ist. Neben dem Datum sind dies auch das Jahr, die Woche, der Monatsname und der Monat als Zahl. Die bei diesem Insert automatisch erstellte ID wird ermittelt (Zeile 45 bis Zeile 49) und im Nachfolgenden für die weiteren Dateneinträge der Wochentage bis zum 7. Wochentag genutzt (Zeile 50 bis Zeile 55). Damit ist die erste Zeile für den Monat erstellt.

Die Variablen werden auf die neuen Werte eingestellt (Zeile 56 bis Zeile 61), die sich aus der Datumsänderung ergeben haben. Die Sortiervariable wird dabei um 12 erhöht, wenn die Jahreszahl auf ein neues Jahr wechselt (Zeile 63). Nur bei einem Monatswechsel während der Woche wird die nächste Zeile durch eine SQL-Anweisung hinzugefügt, (Zeile 65 bis Zeile 72), deren SQL-Code teilweise durch eine Schleife für alle Tage der Woche erstellt wird (Zeile 67 bis Zeile 69). Diese Schleife liest einfach die vorher erstellten Werte für die Tage erneut ein und sorgt dafür, dass die Werte auch in der Folgezeile eingefügt werden. Dies ist erforderlich, damit eine Zeile einmal beim Auslesen der Werte in einem Monat und dann gleichlautend bei Auslesen im folgenden Monat erscheinen kann. So erscheinen für manche Wochen bei Monatsübergängen die Wochenzahlen doppelt.

Ansonsten werden von Zeile 73 bis Zeile 79 die Variablen auf die Werte für die nächste Woche eingestellt und die Schleife beginnt wieder von vorne.

Nachdem so die Datumswerte eingefügt wurden wird anschließend der Inhalt aus der Tabelle "tmp_Kalender" ausgelesen. In einer Schleife (Zeile 83 bis Zeile 94) werden die Datensätze ermittelt. Da die Funktion **DAYOFWEEK** (Zeile 81) bei der HSQLDB den Beginn der Woche mit dem Wert '1' auf den Sonntag legt, hier aber der Montag als erster Tag benutzt wird, muss der Wert nach der Abfrage erst einmal korrigiert werden (Zeile 86 bis Zeile 89). Ansonsten werden die Werte zum jeweiligen Datum direkt übernommen und in Zeile 92 und Zeile 93 in die Tabelle "tmp_Wochenkalender" übertragen.

Wochenkalender_halb

Aufruf aus	
Makro:	*Bericht_Start*

Benötigt	
Makro:	keins
Tabelle:	*tmp_Wochenkalender_halb*
Ansicht:	*Ansicht_Wochen_einzeln*

```
001  SUB Wochenkalender_halb(daStart AS DATE, daEnde AS DATE)
002     DIM inWoche AS INTEGER
003     DIM inJahr AS INTEGER
004     DIM i AS INTEGER
005     DIM d AS DATE
006     DIM d_end AS DATE
007     DIM arRow1()
008     DIM arRow2()
009     d = daStart
010     inJahr = Year(d)
011     d_end = daEnde
012     oSQL_Anweisung = oVerbindung.createStatement()
013     stSql = "DELETE FROM ""tmp_Wochenkalender_halb"""
014     oSQL_Anweisung.executeUpdate(stSql)
015     oSQL_Anweisung.executeQuery("ALTER TABLE ""tmp_Wochenkalender_halb""
           ALTER COLUMN ""ID"" RESTART WITH 1")
016     inWoche = DatePart("ww",d,2,2)
017     arRow1 = array("Monat_Text","Monat_Zahl","D_1","D_2","D_3","T_1","T_2","T_3",
           "F_1", "F_2","F_3")
```

```
018     arRow2 = array("Monat_Text","Monat_Zahl","D_4","D_5","D_6","D_7","T_4","T_5",
           "T_6","T_7","F_4","F_5","F_6","F_7")
019     WHILE d < d_end
020        stSql = "INSERT INTO ""tmp_Wochenkalender_halb"" (""Jahr"",""Woche"",
              ""Monat_Text"",""Monat_Zahl"",""D_1"",""D_2"",""D_3"",""T_1"",""T_2"",
              ""T_3"",""F_1"",""F_2"",""F_3"") VALUES ('"+inJahr+"','"+inWoche+"',"
021        FOR i = LBound(arRow1()) TO UBound(arRow1())
022           stSql = stSql + "(SELECT """+arRow1(i)+""" FROM ""Ansicht_Wochen_einzeln""
                 WHERE ""Woche"" = '"+inWoche+"' AND ""Jahr"" = '"+inJahr+"'),"
023        NEXT
024        stSql = Left(stSql , Len(stSql) -1 ) &")"
025        oSQL_Anweisung.executeUpdate(stSql)
026        stSql = "INSERT INTO ""tmp_Wochenkalender_halb"" (""Jahr"",""Woche"",
              ""Monat_Text"",""Monat_Zahl"",""D_1"",""D_2"",""D_3"",""D_4"",""T_1"",
              ""T_2"",""T_3"",""T_4"",""F_1"",""F_2"",""F_3"",""F_4"") VALUES
              ('"+inJahr+"','"+inWoche+"',"
027        FOR i = LBound(arRow2()) TO UBound(arRow2())
028           stSql = stSql + "(SELECT """+arRow2(i)+""" FROM ""Ansicht_Wochen_einzeln""
                 WHERE ""Woche"" = '"+inWoche+"' AND ""Jahr"" = '"+inJahr+"'),"
029        NEXT
030        stSql = Left(stSql , Len(stSql) -1 ) &")"
031        oSQL_Anweisung.executeUpdate(stSql)
032        d = DateAdd("d",7,d)
033        inJahr = Year(d)
034        inWoche = DatePart("ww",d,2,2)
035     WEND
036  END SUB
```

Für diese Prozedur bereitet die "Ansicht_Wochen_einzeln" bereits die Daten so weit auf, dass sie vom Prinzip her nur noch einmal neu sortiert in die Tabelle "tmp_Wochenkalender_halb" übertragen werden müssen.

Zuerst wird die Tabelle "tmp_Wochenkalender" geleert (Zeile 13 und 14) und der Primärschlüsselwert auf 1 zurückgestellt (Zeile 15). Anschließend wird von der Startwoche die Wochenzahl ermittelt (Zeile 16).

Die Inhalte aus "Ansicht_Wochen_einzeln" sollen in dem Kalender so verteilt werden, dass die Tage "D_1" bis "D_3" (Montag bis Mittwoch) auf einer (linken) Seite erscheinen, die Tage "D_4" bis "D_7" auf der anderen Seite erscheinen. Das Berichtsdesign packt hier den Sonntag in die gleiche Spalte wie den Samstag, da am Samstag und Sonntag sowieso nicht so viele dienstliche Termine vorkommen. In arRow1 werden die Felder für die Tage Montag bis Mittwoch zusammengefasst, in arRow2 die Felder für Donnerstag bis Sonntag (Zeile 17 und 18).

Von Zeile 19 bis Zeile 35 läuft die Schleife so lange ab, bis das vorgegebene Enddatum erreicht ist.

Der Code für die Inserts wird in 2 Etappen zusammengestellt. Zuerst werden die Felder aufgeführt, in die die Daten geschrieben werden sollen, ergänzt um die Werte für Jahr und Woche. Anschließend wird in einer Schleife abgefragt, was an Daten zu dem angegebenen Jahr und der angegebenen Woche in den jeweiligen Feldern aus dem Array arRow1 in der Ansicht "Ansicht_Wochen_einzeln" enthalten ist. Die Schleife wird solange durchlaufen, bis das komplette Array abgearbeitet ist (Zeile 21 bis 23). Anschließend endet der Code wegen der Schleife mit einer Klammer und einem Komma. Das Komma wird ersetzt durch die korrekte 2. schließende Klammer (Zeile 24).

Nach dem Insert für die erste Seite erfolgt der Insert für die zweite Seite mit den gleichen Schritten. Nachdem die Schritte durchlaufen sind wird das Datum um eine Woche heraufgesetzt (Zeile 32) und anschließend die Jahreszahl und die Wochenzahl aus diesem Datum für den nächsten Schleifendurchlauf ermittelt.

Halbjahreskalender

Aufruf aus	
Makro:	*Bericht_Start*

Benötigt	
Makro:	*Date_2_SQLDate*
Tabelle:	*tmp_Halbjahreskalender, tmp_Kalender*

```
001 SUB Halbjahreskalender(inJahr AS INTEGER)
002    DIM i AS INTEGER
003    DIM inMonat AS INTEGER
004    DIM inWoche AS INTEGER
005    DIM inTag AS INTEGER
006    DIM inDd AS INTEGER
007    DIM inMd AS INTEGER
008    DIM inYd AS INTEGER
009    DIM inW AS INTEGER
010    DIM inID AS INTEGER
011    DIM inWcount AS INTEGER
012    DIM inArrayCount AS INTEGER
013    DIM inMonat_2 AS INTEGER
014    DIM k AS INTEGER
015    DIM d AS DATE
016    DIM d_neu AS DATE
017    DIM d_end AS DATE
018    DIM stBezeichnung AS STRING
019    DIM stBezeichnung_2 AS STRING
020    DIM stDatum AS STRING
021    DIM stDat AS STRING
022    DIM stDat_2 AS STRING
023    DIM stMonat AS STRING
024    DIM stTag AS STRING
025    DIM stJahr AS STRING
026    DIM arMon()
027    d = "01.01."+inJahr
028    d_end = "01.01."+(inJahr+1)
029    d_neu=DateAdd("d",1,d)
030    stDatum = Date_2_SQLDate(d)
031    stJahr = Trim(Str(inJahr))
032    oSQL_Anweisung = oVerbindung.createStatement()
033    oSQL_Anweisung1 = oVerbindung.createStatement()
034    oSQL_Anweisung2 = oVerbindung.createStatement()
035    stSql = "DELETE FROM ""tmp_Halbjahreskalender"""
036    oSQL_Anweisung.executeUpdate(stSql)
037    oSQL_Anweisung.executeQuery("ALTER TABLE ""tmp_Halbjahreskalender""
          ALTER COLUMN ""ID"" RESTART WITH 1")
038    arMon = array("-01-","-02-","-03-","-04-","-05-","-06-","-07-","-08-","-09-",
          "-10-","-11-","-12-")
039    inID = 28
040    IF isDate("29.02."+stJahr) THEN
041       inID = 29
042    END IF
043    FOR i=1 TO inID
044       IF i < 10 THEN
045          stTag = "0"+Trim(Str(i))
046       ELSE
047          stTag = Trim(Str(i))
048       END IF
049       stSql = "INSERT INTO ""tmp_Halbjahreskalender"" (""Jahr"",""Sort_Zahl"",
             ""T_1"",""D_1"",""F_1"",""T_2"",""D_2"",""F_2"",""T_3"",""D_3"",""F_3"",
             ""T_4"",""D_4"",""F_4"",""T_5"",""D_5"",""F_5"",""T_6"",""D_6"",""F_6"")
             VALUES ('"+inJahr+"','1','"
```

```
050         FOR k = 0 TO 5
051            stSql = stSql + " (SELECT ""Bezeichnung"" FROM ""tmp_Kalender"" WHERE
                  ""Datum"" = '"+stJahr+arMon(k)+stTag+"'),'"+stJahr+arMon(k)+stTag+"',
                  (SELECT ""Format"" FROM ""tmp_Kalender"" WHERE ""Datum"" =
                  '"+stJahr+arMon(k)+stTag+"'),"
052         NEXT
053         stSql = Left(stSql , Len(stSql) -1 ) &")"
054         oSQL_Anweisung.executeUpdate(stSql)
055      NEXT
056      inID = inID + 1
057      FOR i = inID TO 30
058         stTag = Trim(Str(i))
059         stSql = "INSERT INTO ""tmp_Halbjahreskalender"" (""Jahr"",""Sort_Zahl"",
               ""T_1"",""D_1"",""F_1"",""T_3"",""D_3"",""F_3"",""T_4"",""D_4"",""F_4"",
               ""T_5"",""D_5"",""F_5"",""T_6"",""D_6"",""F_6"")
               VALUES ('"+inJahr+"','1',"
060         FOR k = 0 TO 5
061            IF k = 1 THEN
062            ELSE
063               stSql = stSql + " (SELECT ""Bezeichnung"" FROM ""tmp_Kalender"" WHERE
                     ""Datum""= '"+stJahr+arMon(k)+stTag+"'),'"+stJahr+arMon(k)+stTag+"',
                     (SELECT ""Format"" FROM ""tmp_Kalender"" WHERE ""Datum"" =
                     '"+stJahr+arMon(k)+stTag+"'),"
064            END IF
065         NEXT
066         stSql = Left(stSql , Len(stSql) -1 ) &")"
067         oSQL_Anweisung.executeUpdate(stSql)
068      NEXT
069      stTag = "31"
070      stSql = "INSERT INTO ""tmp_Halbjahreskalender"" (""Jahr"",""Sort_Zahl"",""T_1"",
            ""D_1"",""F_1"",""T_3"",""D_3"",""F_3"",""T_5"",""D_5"",""F_5"") VALUES
            ('"+inJahr+"','1',"
071      FOR k = 0 TO 4 STEP 2
072         stSql = stSql + " (SELECT ""Bezeichnung"" FROM ""tmp_Kalender"" WHERE
               ""Datum"" = '"+stJahr+arMon(k)+stTag+"'),'"+stJahr+arMon(k)+stTag+"',
               (SELECT ""Format"" FROM ""tmp_Kalender"" WHERE ""Datum"" =
               '"+stJahr+arMon(k)+stTag+"'),"
073      NEXT
074      stSql = Left(stSql , Len(stSql) -1 ) &")"
075      oSQL_Anweisung.executeUpdate(stSql)
076      FOR i=1 TO 30
077         IF i < 10 THEN
078            stTag = "0"+Trim(Str(i))
079         ELSE
080            stTag = Trim(Str(i))
081         END IF
082         stSql = "INSERT INTO ""tmp_Halbjahreskalender"" (""Jahr"",""Sort_Zahl"",
               ""T_1"",""D_1"",""F_1"",""T_2"",""D_2"",""F_2"",""T_3"",""D_3"",""F_3"",
               ""T_4"",""D_4"",""F_4"",""T_5"",""D_5"",""F_5"",""T_6"",""D_6"",""F_6"")
               VALUES ('"+inJahr+"','2',"
083         FOR k = 6 TO 11
084            stSql = stSql + " (SELECT ""Bezeichnung"" FROM ""tmp_Kalender"" WHERE
                  ""Datum"" = '"+stJahr+arMon(k)+stTag+"'),'"+stJahr+arMon(k)+stTag+"',
                  (SELECT ""Format"" FROM ""tmp_Kalender"" WHERE ""Datum"" =
                  '"+stJahr+arMon(k)+stTag+"'),"
085         NEXT
086         stSql = Left(stSql , Len(stSql) -1 ) &")"
087         oSQL_Anweisung.executeUpdate(stSql)
088      NEXT
089      stTag = "31"
090      stSql = "INSERT INTO ""tmp_Halbjahreskalender"" (""Jahr"",""Sort_Zahl"",""T_1"",
            ""D_1"",""F_1"",""T_2"",""D_2"",""F_2"",""T_4"",""D_4"",""F_4"",""T_6"",
            ""D_6"",""F_6"") VALUES ('"+inJahr+"','2',"
091      FOR k = 6 TO 11
092         IF k = 8 OR k = 10 THEN
093         ELSE
```

```
094             stSql = stSql + " (SELECT ""Bezeichnung"" FROM ""tmp_Kalender"" WHERE
                   ""Datum"" = '"+stJahr+arMon(k)+stTag+"'),'"+stJahr+arMon(k)+stTag+"',
                   (SELECT ""Format"" FROM ""tmp_Kalender"" WHERE ""Datum"" =
                   '"+stJahr+arMon(k)+stTag+"'),"
095         END IF
096       NEXT
097       stSql = Left(stSql , Len(stSql) -1 ) &")"
098       oSQL_Anweisung.executeUpdate(stSql)
099   END SUB
```

Die Prozedur "Halbjahreskalender" soll die Tabelle "tmp_Halbjahreskalender" mit Daten versehen, die anschließend den Bericht "Halbjahreskalender_A4_quer" versorgen. Dieser Halbjahreskalender stellt jeweils 6 komplette Monate auf einer Seite dar. Das bedeutet auch, dass für einen Bericht 6 Spalten für Datumswerte und 6 Spalten für Inhalte vorgehalten werden müssen. Und da die Ferien auch noch in den Bericht einfließen sollen sind hier für den Formatierungscode wieder 6 Felder notwendig. Zusammen mit der Jahreszahl, einer Sortierziffer für das 1. und 2. Halbjahr und dem Primärschlüssel sind das insgesamt 21 Felder.

Zuerst wird die Tabelle "tmp_Halbjahreskalender" geleert (Zeile 35) und der AutoWert wieder auf den Startwert 1 gesetzt (Zeile 37). In Zeile 38 werden dann noch die Monatszahlen vorformatiert für die Eingabe in ein Datum in ein Array geschrieben. In Zeile 40 bis Zeile 42 wird mit der Funktion **isDate** überprüft, ob in dem Jahr ein 29. Februar existiert. Ist dies der Fall, so wird der Variablen "inID" der Wert 29, sonst der Wert 28 zugeschrieben.

Die erste Schleife von Zeile 43 bis 54 läuft so lange, bis dieser Wert von "inID" erreicht ist. Dabei wird zuerst die Tageszahl als Text erstellt, indem den einstelligen Zahlen für "inID" eine '0' vorangestellt wird oder aber einfach aus der Zahl eine Textvariable erstellt wird (Zeilen 44 bis 48).

Der SQL-Code für den anschließenden Insert wird in zwei Teilen zusammengestellt. Zuerst erfolgt die Zusammenstellung der Felder, in die die Daten eingefügt werden. Außerdem steht in der Basis bereits die Variable für das Jahr und die Sortierzahl, die in diesem Fall für das erste Halbjahr eine '1' einsetzt (Zeile 49). Von Zeile 50 bis Zeile 52 erfolgt dann eine Schleife, die durch alle Monate hindurchgeht, die im ersten Halbjahr liegen. Das jeweilige Datum wird dabei mit dem bereits aus Zeile 38 beschriebenen Array variabel erstellt. Datenbasis für die Inhalte des Halbjahreskalenders ist die Tabelle "tmp_Kalender", die also vorher erstellt werden muss. Da die Schleife zum Schluss mit einer Klammer gefolgt von einem Komma endet muss hier noch nachgebessert werden. Das Komma wird entfernt und eine schließende Klammer gesetzt (Zeile 53). Anschließend wird die Datenzeile erstellt und die Schleife beginnt für die ersten 28 (oder 29) Tage wieder von vorne.

Von Zeile 57 bis Zeile 68 erfolgt die gleiche Schleife noch einmal, dieses Mal aber ohne den Februar. Die Schleife wird also maximal 2 Mal durchlaufen, denn dann sind auch die anderen 5 Monate des ersten Halbjahres auf 30 Tage aufgestockt.

Schließlich muss noch einmal ein Insert für all die Monate stattfinden, die 31 Tage haben. Dies wird in Zeile 69 bis Zeile 73 erledigt Die Schleife kann praktischerweise in Zweierschritten durchlaufen werden (Zeile 69), da im ersten Halbjahr jeder 2. Monat 31 Tage hat.

Ab Zeile 76 erfolgt dann ein entsprechendes Vorgehen für das 2. Halbjahr. Hier kann direkt bis zu 30 Tagen aufgefüllt werden, weil eben alle Monate mindestens 30 Tage haben. Nur die Reihenfolge der Monate für die 31 Tage ist nicht so günstig gelegen wie im ersten Halbjahr, so dass hier mit k=6 (Monat '-07-') bis k=11 (Monat '-12-') in Einserschritten gearbeitet wird (Zeile 91). Dafür werden dann in Zeile 92 die Monate September und November nicht mehr im SQL-Befehl berücksichtigt.

Zeitplan

Aufruf aus	
Makro:	*Bericht_Start*

Benötigt
Makro: *SQLDate_2_Datum, Date_2_SQLDate*
Tabelle: *tmp_Terminuebersicht, Termine*

```
001 SUB Zeitplan(daStart AS DATE, daEnde AS DATE, stTermin AS STRING)
002    DIM stStart AS STRING
003    DIM stEnde AS STRING
004    DIM stDat AS STRING
005    DIM stDatum AS STRING
006    DIM stBemerkung AS STRING
007    DIM stBezeichnung AS STRING
008    DIM stBezeichnung_alt AS STRING
009    DIM stBezeichNeu AS STRING
010    DIM stSpaltengrenze AS STRING
011    DIM inID AS INTEGER
012    DIM inID_alt AS INTEGER
013    DIM i AS INTEGER
014    DIM k AS INTEGER
015    stStart = Date_2_SQLDate(daStart)
016    stEnde = Date_2_SQLDate(daEnde)
017    oSQL_Anweisung = oVerbindung.createStatement()
018    oSQL_Anweisung1 = oVerbindung.createStatement()
019    oSQL_Anweisung2 = oVerbindung.createStatement()
020    stSql = "DELETE FROM ""tmp_Terminuebersicht"""
021    oSQL_Anweisung.executeUpdate(stSql)
022    stSql = "INSERT INTO ""tmp_Terminuebersicht"" (SELECT ""Termine"".""ID"",
          ""Termine"".""Beginn_Datum"",""Termine"".""Bezeichnung"",
          ""Terminart"".""Terminart"", EXTRACT(MONTH FROM ""Beginn_Datum"" ),
          EXTRACT(YEAR FROM ""Beginn_Datum"" ), NULL,NULL,NULL,NULL,NULL,NULL FROM
          ""Termine"" LEFT JOIN ""Terminart"" ON ""Termine"".""artID"" =
          ""Terminart"".""ID""  WHERE ""Termine"".""zieID"" IN ("+stTermin+")
          AND((""Termine"".""Beginn_Datum"" > '"+stStart+"' AND
          ""Termine"".""Beginn_Datum"" < '"+stEnde+"') OR (""Termine"".""Ende_Datum"" >
          '"+stStart+"' AND ""Termine"".""Ende_Datum"" < '"+stEnde+"')) ORDER BY
          ""Termine"".""Beginn_Datum"" ASC, ""Termine"".""Beginn_Uhrzeit"" ASC)"
023    oSql_Anweisung.executeUpdate(stSql)
024    stSql = "SELECT ""ID"", ""Ende_Datum"", ""Bemerkung"", ""Bezeichnung"" FROM
          ""Termine"" WHERE ""zieID"" IN ("+stTermin+") AND NOT((""Ende_Datum"" IS NULL
          OR ""Ende_Datum"" = ""Beginn_Datum"") AND ""Bemerkung"" IS NULL) "
025    oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
026    WHILE oAbfrageergebnis.next
027       inID = oAbfrageergebnis.getInt(1)
028       stDat = oAbfrageergebnis.getString(2)
029       stBemerkung = oAbfrageergebnis.getString(3)
030       stBezeichnung = oAbfrageergebnis.getString(4)
031       IF stDat <> "" THEN
032          stDatum = "- "+SQLDate_2_Datum(stDat)
033          stSql1 = "UPDATE ""tmp_Terminuebersicht"" SET ""D_2"" ='"+stDatum+"' WHERE
                ""ID"" = '"+inID+"'"
034          oSQL_Anweisung1.executeUpdate(stSql1)
035       END IF
036       IF stBemerkung <> "" THEN
037          stBezeichnung = stBezeichnung + " | " + stBemerkung
038          stSql1 = "UPDATE ""tmp_Terminuebersicht"" SET ""Bezeichnung""
                ='"+stBezeichnung+"' WHERE ""ID"" = '"+inID+"'"
039          oSQL_Anweisung1.executeUpdate(stSql1)
040       END IF
041    WEND
042    stSql = "SELECT ""ID"", ""Beginn_Datum"", ""Ende_Datum"", ""Bemerkung"",
          ""Bezeichnung"" FROM ""Termine"" WHERE ""zieID"" IN ("+stTermin+") AND
          (((""Beginn_Datum"" > '"+stStart+"' AND ""Beginn_Datum"" < '"+stEnde+"')  OR
          (""Ende_Datum"" > '"+stStart+"' AND ""Ende_Datum"" < '"+stEnde+"')) AND
          ""Bezeichnung"" IN (SELECT ""Bezeichnung"" FROM ""Termine"" GROUP BY
          ""Bezeichnung"" HAVING((COUNT(""ID"")>1))) ORDER BY ""Bezeichnung"",
          ""Beginn_Datum"""
```

```
043     oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
044     inID_alt = -1
045     stBezeichnung_alt = ""
046     k = 1
047     stSpaltengrenze = "Zu viele Termine in einer Zeile. Nicht alle Termine wurden
           übertragen."
048     WHILE oAbfrageergebnis.next
049        inID = oAbfrageergebnis.getInt(1)
050        stDatum = SQLDate_2_Datum(oAbfrageergebnis.getString(2))
051        stEnde = oAbfrageergebnis.getString(3)
052        stBemerkung = oAbfrageergebnis.getString(4)
053        stBezeichnung = oAbfrageergebnis.getString(5)
054        IF stBezeichnung = stBezeichnung_alt THEN
055           IF i>7 THEN
056              msgbox stSpaltengrenze
057           ELSE
058              stSql1 = "UPDATE ""tmp_Terminuebersicht"" SET ""D_"+i+"" ='"+stDatum+"'
                    WHERE ""ID"" = '"+inID_alt+"'"
059              oSQL_Anweisung1.executeUpdate(stSql1)
060              i = i + 1
061              IF stEnde <> "" THEN
062                 IF i>7 THEN
063                    msgbox stSpaltengrenze
064                 ELSE
065                    stEnde = "- "+SQLDate_2_Datum(stEnde)
066                    stSql1 = "UPDATE ""tmp_Terminuebersicht"" SET ""D_"+i+""
                       ='"+stEnde+"' WHERE ""ID"" = '"+inID_alt+"'"
067                    oSQL_Anweisung1.executeUpdate(stSql1)
068                    i = i + 1
069                 END IF
070              END IF
071              IF stBemerkung <> "" THEN
072                 stSql2 = "SELECT ""Bezeichnung"" FROM ""tmp_Terminuebersicht"" WHERE
                       ""ID"" = '"+inID_alt+"'"
073                 oAbfrageergebnis2 = oSQL_Anweisung2.executeQuery(stSql2)
074                 WHILE oAbfrageergebnis2.next
075                    stBezeichNeu = oAbfrageergebnis2.getString(1)
076                 WEND
077                 stBezeichNeu = stBezeichNeu + " | " + stBemerkung
078                 stSql1 = "UPDATE ""tmp_Terminuebersicht"" SET ""Bezeichnung""
                       ='"+stBezeichNeu+"' WHERE ""ID"" = '"+inID_alt+"'"
079                 oSQL_Anweisung1.executeUpdate(stSql1)
080              END IF
081              stSql1 = "DELETE FROM ""tmp_Terminuebersicht"" WHERE ""ID"" =
                    '"+inID+"'"
082              oSQL_Anweisung1.executeUpdate(stSql1)
083           END IF
084        ELSE
085           i = 2
086           stSql2 = "SELECT ""D_2"" FROM ""tmp_Terminuebersicht"" WHERE ""ID"" =
                 '"+inID+"' AND NOT ""D_2"" IS NULL"
087           oAbfrageergebnis2 = oSQL_Anweisung2.executeQuery(stSql2)
088           WHILE oAbfrageergebnis2.next
089              i = 3
090           WEND
091           stBezeichnung_alt = stBezeichnung
092           inID_alt = inID
093        END IF
094     WEND
095  END SUB
```

Mit dieser Prozedur wird der Zeitplan über die Tabelle "tmp_Terminuebersicht" mit Daten versorgt. Nachdem die Tabelle "tmp_Terminübersicht" geleert worden ist (Zeile 20 und 21) wird sie direkt über einen SQL-Befehl mit Daten aus der Tabelle "Termine" versorgt (Zeile 22 und 23). Da die Tabelle "tmp_Terminübersicht" mehr Felder hat als in diesem Insert direkt mit Inhalt gefüllt werden sollen, werden die anderen Felder einfach mit einem leeren Inhalt NULL gefüllt.

In den Bedingungen für die Inhalte des Inserts steht für das Feld "zieID" IN ("+stTermin+"). Die Variable stTermin besteht aus Ganzzahlen, getrennt durch Komma, die einen Verweis auf die Primärschlüssel der Tabelle "Zielgruppe" darstellen. So kann bei den Zielgruppen geregelt werden, welche Inhalte aus anderen Gruppen auch Kalender für die eigentliche Zielgruppe aufzunehmen sind.

In Zeile 24 und 25 werden die Inhalte aus der Tabelle "Termine" gesucht, die die Zielgruppe in ihrem Kalender finden soll und die ein Enddatum haben, das sich von dem Startdatum unterschiedet oder einen Eintrag in der Spalte "Bemerkung" vorweisen können. In der Abfrage ist die Bedingung so gestellt, dass nicht beide gleichzeitig NULL sein dürfen. Von Zeile 26 bis Zeile 41 wird dann an jedes Feld, zu dem ein Enddatum existiert, das Enddatum an das Anfangsdatum angehängt und zu jeder Zeile, zu der eine Bemerkung existiert, die Bemerkung an die Bezeichnung angehängt.

Mit der Abfrage in Zeile 42 werden die Termine gesucht, bei denen die Bezeichnung mehr als einmal vorkommt (**"Bezeichnung" HAVING((COUNT("ID")>1))**). Diese Termine sollen an die vorherigen gleichlautenden Termine mit Datumsvermerk angehängt werden. Da die Spaltenanzahl für Termine in dem Bericht vom Platz her begrenzt ist (das letzte Feld ist D7 - also maximal 7 Datumseinträge) wird schon einmal für eine Meldung zur Spaltengrenze in Zeile 47 der Text in eine Variable geschrieben.

Die in Zeile 48 beginnende Schleife springt nach dem Auslesen der Werte aus der ersten Abfragezeile zuerst nach Zeile 84, da die Bedingung von Zeile 54 nicht erfüllt ist (stBezeichnung_alt startet leer – Zeile 45). Die Variable i wird hier zuerst auf '2' gesetzt (für "D_2"). Dann wird durch Abfrage nachgesehen, ob in der Spalte D_2 bereits ein Eintrag ist. Ist dort ein Eintrag, so soll mit i=3 weiter fortgefahren werden. Ansonsten bleibt i=2 bestehen. In den Zeilen 91 und 92 werden die aktuellen Variablen für die Bezeichnung und die ID auf die Variablen mit dem Zusatz "_alt" kopiert, damit später klar ist, wann eine Änderung des Datensatzes erfolgen muss.

Anschließend geht es für den nächsten Datensatz weiter mit Zeile 54, da die Variablen für die Bezeichnung jetzt gleich sind und auch die anschließende Bedingung erfüllt ist, dass i nicht größer als 7 ist. In Zeile 58 bis Zeile 60 wird das Beginndatum in die aktuelle Datenzeile übertragen und der Zähler von i um eins erhöht, damit der folgende Eintrag im nächsten Tabellenfeld erfolgt. Ist i zu groß (Zeile 62), so wird kein weiterer Datumseintrag vorgenommen (Zeile 65 bis 68).

Enthält die Bemerkung Text (Zeile 71), so wird dieser Text an die bisherige Bezeichnung in der Tabelle mit angehängt. So könnte z.B. die ursprüngliche Bezeichnung "Lehrerkonferenz" lauten und in den Bemerkungen dann '1.', '2.' usw stehen. Dann werden alle Lehrerkonferenzen in eine Zeile geschrieben und zu Beginn steht dann 'Lehrerkonferenz | 1. | 2. | ...'

Die Tabelle "tmp_Terminuebersicht" enthält nach dem Insert alle Daten, auch wiederholende Werte für die Bezeichnung. Nachdem der Datensatz mit der gleichen Bezeichnung in den vorherigen Datensatz überführt wurde muss er gelöscht werden. Hierzu wird die aktuelle ID genutzt (Zeile 81).

GruppeMultiUebersicht

Aufruf aus	
Makro:	*Bericht_Start*

Benötigt	
Makro:	*FeiertageArray, Date_2_SQLDate*
Tabelle:	*tmp_Gruppe_multi, Ferien, Termine*

```
001  SUB GruppeMultiUebersicht(inJahr AS INTEGER, inFerien AS INTEGER, stTermin AS
     STRING, inJahrgang AS INTEGER, stJahrgang AS STRING)
002     DIM stDatum AS STRING
```

```
003     DIM stDat_Beginn AS STRING
004     DIM stDat_Ende AS STRING
005     DIM stUpdate AS STRING
006     DIM stZeit AS STRING
007     DIM i AS INTEGER
008     DIM d AS DATE
009     DIM d_neu AS DATE
010     DIM d_end AS DATE
011     DIM stBezeichnung AS STRING
012     DIM stBemerkung AS STRING
013     inJahresschleife = inJahr
014     oSQL_Anweisung = oVerbindung.createStatement()
015     oSQL_Anweisung1 = oVerbindung.createStatement()
016     oSQL_Anweisung2 = oVerbindung.createStatement()
017     stSql = "DELETE FROM ""tmp_Gruppe_multi"""
018     oSQL_Anweisung.executeUpdate(stSql)
019     oSQL_Anweisung.executeQuery("ALTER TABLE ""tmp_Gruppe_multi""
           ALTER COLUMN ""ID"" RESTART WITH 1")
020     FOR inJahr=inJahresschleife TO inJahresschleife+1
021        Feiertage = FeiertageArray(inJahr)
022        d = "01.01."+inJahr
023        d_end = "01.01."+(inJahr+1)
024        stDatum = Date_2_SQLDate(d)
025        d_neu=d
026        WHILE d_neu < d_end
027           stSql = "INSERT INTO ""tmp_Gruppe_multi"" (""Datum"") VALUES
                 ('"+stDatum+"') "
028           oSQL_Anweisung.executeUpdate(stSql)
029           d_neu=DateAdd("d",1,d_neu)
030           stDatum = Date_2_SQLDate(d_neu)
031        WEND
032        FOR i=0 TO 14
033           stDatum = Feiertage(i,1)
034           stBezeichnung = Feiertage(i,0)
035           stSql = "UPDATE ""tmp_Gruppe_multi"" SET ""Bezeichnung""
                 ='"+stBezeichnung+"', ""Format"" = '1' WHERE ""Datum"" = '"+stDatum+"'"
036           oSQL_Anweisung.executeUpdate(stSql)
037        NEXT
038        stSql = "SELECT ""Beginn_Datum"", ""Ende_Datum"" FROM ""Ferien"" WHERE
              YEAR(""Beginn_Datum"")='"+inJahr+"' OR YEAR(""Ende_Datum"")='"+inJahr+"'
              ORDER BY ""Beginn_Datum"" ASC, ""Ende_Datum"" DESC"
039        oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
040        WHILE oAbfrageergebnis.next
041            stDat_Beginn = oAbfrageergebnis.getString(1)
042           stDat_Ende = oAbfrageergebnis.getString(2)
043           IF inFerien = 1 THEN
044              stUpdate = """Bezeichnung"" ='Ferien', ""Format"" = '1'"
045           ELSE
046              stUpdate = """Format"" = '1'"
047           END IF
048           stSql1 = "UPDATE ""tmp_Gruppe_multi"" SET "+stUpdate+" WHERE ""Datum"" =
                 '"+stDat_Beginn+"'"
049           oSQL_Anweisung1.executeUpdate(stSql1)
050           IF stDat_Ende <>"" AND stDat_Ende <> stDat_Beginn THEN
051              d_neu=DateAdd("d",1,CDate(stDat_Beginn))
052              WHILE d_neu <= CDate(stDat_Ende)
053                 stDatum = Date_2_SQLDate(d_neu)
054                 stSql1 = "UPDATE ""tmp_Gruppe_multi"" SET "+stUpdate+" WHERE
                       ""Datum"" = '"+stDatum+"'"
055                 oSQL_Anweisung1.executeUpdate(stSql1)
056                 d_neu=DateAdd("d",1,d_neu)
057              WEND
058           END IF
059        WEND
060        stSql = "SELECT ""Bezeichnung"", ""Beginn_Datum"", ""Ende_Datum"",
              ""Bemerkung"", ""Beginn_Uhrzeit"", ""Gruppe_multi"" FROM ""Termine"" WHERE
```

```
            BITAND(""Gruppe_multi"",'"+inJahrgang+"') > 0 AND ""zieID"" IN
            ("+stTermin+") AND (YEAR(""Beginn_Datum"")='"+inJahr+"' OR
            YEAR(""Ende_Datum"")='"+inJahr+"') ORDER BY ""Beginn_Datum"" ASC,
            ""Ende_Datum"" DESC, ""Beginn_Uhrzeit"" ASC"
061         oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
062         WHILE oAbfrageergebnis.next
063             stBezeichnung = oAbfrageergebnis.getString(1)
064             stBezeichnung_Dat_Ende = oAbfrageergebnis.getString(1)
065             stDat_Beginn = oAbfrageergebnis.getString(2)
066             stDat_Ende = oAbfrageergebnis.getString(3)
067             stBemerkung = oAbfrageergebnis.getString(4)
068             stZeit = oAbfrageergebnis.getString(5)
069             stZeit = left(stZeit,5)
070             IF stBemerkung <> "" THEN
071                 stBezeichnung = stBezeichnung + " " + stBemerkung
072                 stBezeichnung_Dat_Ende = stBezeichnung_Dat_Ende + " " + stBemerkung
073             END IF
074             IF stZeit <> "" THEN
075                 stBezeichnung = stZeit + " " + stBezeichnung
076                 stBezeichnung_Dat_Ende = stZeit + " " + stBezeichnung_Dat_Ende
077             END IF
078             stSql1 = "SELECT ""Bezeichnung"" FROM ""tmp_Gruppe_multi"" WHERE ""Datum""
                = '"+stDat_Beginn+"'"
079             oAbfrageergebnis1 = oSQL_Anweisung1.executeQuery(stSql1)
080             WHILE oAbfrageergebnis1.next
081                 IF oAbfrageergebnis1.getString(1) <>"" THEN
082                     stBezeichnung = oAbfrageergebnis1.getString(1) + CHR(13) +
                        stBezeichnung
083                 END IF
084             WEND
085             stSql1 = "UPDATE ""tmp_Gruppe_multi"" SET ""Bezeichnung""
                ='"+stBezeichnung+"' WHERE ""Datum"" = '"+stDat_Beginn+"'"
086             oSQL_Anweisung1.executeUpdate(stSql1)
087             IF stDat_Ende <>"" AND stDat_Ende <> stDat_Beginn THEN
088                 d_neu=DateAdd("d",1,CDate(stDat_Beginn))
089                 WHILE d_neu <= CDate(stDat_Ende)
090                     stDatum = Date_2_SQLDate(d_neu)
091                     stSql2 = "SELECT ""Bezeichnung"" FROM ""tmp_Gruppe_multi"" WHERE
                        ""Datum"" = '"+stDatum+"'"
092                     oAbfrageergebnis2 = oSQL_Anweisung2.executeQuery(stSql2)
093                     WHILE oAbfrageergebnis2.next
094                         IF oAbfrageergebnis2.getString(1) <>"" THEN
095                             stBezeichnung = oAbfrageergebnis2.getString(1) + CHR(13) +
                                stBezeichnung_Dat_Ende
096                         ELSE
097                             stBezeichnung = stBezeichnung_Dat_Ende
098                         END IF
099                     WEND
100                     stSql1 = "UPDATE ""tmp_Gruppe_multi"" SET ""Bezeichnung""
                        ='"+stBezeichnung+"' WHERE ""Datum"" = '"+stDatum+"'"
101                     oSQL_Anweisung1.executeUpdate(stSql1)
102                     d_neu=DateAdd("d",1,d_neu)
103                 WEND
104             END IF
105         WEND
106     NEXT
107     stSql = "UPDATE ""tmp_Gruppe_multi"" SET ""Gruppe_multi"" ='"+stJahrgang+"'"
108     oSQL_Anweisung.executeUpdate(stSql)
109 END SUB
```

Der prinzipielle Ablauf ist wie der bei ähnlichen Prozeduren zur Erstellung von Tabellen, die anschließend als Basis für einen der berichte dienen. In Zeile 17 bis Zeile 19 wird die Tabelle von vorherigem Inhalt geleert und der Autowert auf '1' neu eingestellt. Von Zeile 20 bis Zeile 106 erfolgt dann eine Schleife, die für 2 Jahre durchlaufen wird. Erst ganz zum Schluss dieser Prozedur wird dann zu allen in der Tabelle eingetragenen Datensätzen der Jahrgang hinzugefügt (Zeile 107 und Zeile 108).

Die Datumseinträge in die Tabelle erfolgen von Zeile 26 bis Zeile 31. Anschließend werden von Zeile 32 bis Zeile 37 die Feiertage hinzugefügt. Dies geschieht hier vor den Ferien, weil die Ferien in dem Bericht nur mit einer Zeile erscheinen sollen und die Feiertage in den Ferien nicht angezeigt werden sollen.

In Zeile 38 und Zeile 39 werden die Ferien ausgelesen und anschließend ab Zeile 40 bis Zeile 59 in der Tabelle den jeweiligen Datumswerten zugeordnet.

In der Abfrage Zeile 60 werden jetzt die Termine herausgesucht, die zu dem entsprechenden Jahrgang passen. Wichtig ist hier die Funktion **BITAND**("Gruppe_multi", Jahrgangswert) > 0. Angenommen in dem Feld "Gruppe_multi" befindet sich die Zahl '7', dann soll ja hier klar werden, dass nur 1, 2 und 4 in diese Zahl hinein passen – die Zahlen also, die den Jahrgängen 5, 6 und 7 in den Formularen zugeordnet waren. Die Zahl 7 hat in der Binärschreibweise die Binärfolge 0111. Wird jetzt nach dem 7 Jahrgang gesucht, so muss die Funktion bei der Addition von 0111 + 0100 einen Wert größer als 0 ermitteln. Beim binären Addieren zählen nur die Felder als 1, bei denen beide binären Codes eine 1 stehen haben. 0111 + 0100 = 0100 und ist also größer als 0.

Alle Daten, die jetzt für diesen Jahrgang gelten, werden wie bei der Prozedur "Kalender", nach und nach abgearbeitet, gegebenenfalls mit der Zeit und den Bemerkungen ergänzt und bei mehreren Terminen am gleichen Tag auch noch mit einem Zeilenumbruch versehen in die Tabelle "tmp-Gruppe_multi" eingetragen.

Monat

Aufruf aus	
Makro:	*Monatsansicht*

Benötigt	
Makro:	*Date_2_Datum, Date_2_SQLDate*
Tabelle:	*tmp_Monat, Termine, Terminart*

```
001 SUB Monat(inMonat AS INTEGER, inJahr AS INTEGER)
002    DIM d AS DATE
003    DIM d_neu AS DATE
004    DIM d_end AS DATE
005    DIM stDatum AS STRING
006    DIM stDatumSql AS STRING
007    DIM inID AS INTEGER
008    DIM i AS INTEGER
009    oSQL_Anweisung = oVerbindung.createStatement()
010    oSQL_Anweisung1 = oVerbindung.createStatement()
011    stSql = "DELETE FROM ""tmp_Monat"""
012    oSQL_Anweisung.executeUpdate(stSql)
013    d = CDate("01."+inMonat+"."+inJahr)
014    d_end = DateAdd("m",1,d)
015    inID = 1
016    FOR d_neu = d TO d_end-1
017       stDatumSql = Date_2_SQLDate(d_neu)
018       stDatum = Date_2_Datum(d_neu)
019       inWochentag = DatePart("w",d_neu,2,2)
020       stSql = "INSERT INTO ""tmp_Monat"" (""ID"", ""T_"+inWochentag+""") VALUES
             ('"+inID+"','"+stDatum+"')"
021       oSql_Anweisung.executeUpdate(stSql)
022       stBezeichnung = stDatum
023       stSql ="SELECT ""Terminart"".""Terminart""||': '||""Termine"".""Bezeichnung""
             FROM ""Termine"" LEFT JOIN ""Terminart"" ON ""Terminart"".""ID"" =
             ""Termine"".""artID"" WHERE ""Termine"".""Beginn_Datum"" ='"+stDatumSql+"'
             OR (""Termine"".""Beginn_Datum"" < '"+stDatumSql+"' AND
             ""Termine"".""Ende_Datum"" >= '"+stDatumSql+"')"
024       oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
```

```
025         WHILE oAbfrageergebnis.next
026            stBezeichnung = stBezeichnung + CHR(13) + oAbfrageergebnis.getString(1)
027            stSql1 = "UPDATE ""tmp_Monat"" SET ""T_"+inWochentag+"""
                  ='"+stBezeichnung+"' WHERE ""ID""='"+inID+"'"
028            oSQL_Anweisung1.executeUpdate(stSql1)
029         WEND
030         FOR i = inWochentag+1 TO 7
031            d_neu = DateAdd("d",1,d_neu)
032            IF d_neu < d_end THEN
033               stDatumSql = Date_2_SQLDate(d_neu)
034               stDatum = Date_2_Datum(d_neu)
035               stBezeichnung = stDatum
036               stSql1 = "UPDATE ""tmp_Monat"" SET ""T_"+i+""" ='"+stBezeichnung+"'
                     WHERE ""ID""='"+inID+"'"
037               oSQL_Anweisung1.executeUpdate(stSql1)
038               stSql = "SELECT ""Terminart"".""Terminart""||':
                     '||""Termine"".""Bezeichnung"" FROM ""Termine"" LEFT JOIN
                     ""Terminart"" ON ""Terminart"".""ID"" = ""Termine"".""artID"" WHERE
                     ""Termine"".""Beginn_Datum"" = '"+stDatumSql+"' OR
                     (""Termine"".""Beginn_Datum"" < '"+stDatumSql+"' AND
                     ""Termine"".""Ende_Datum"" >= '"+stDatumSql+"')"
039               oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
040               WHILE oAbfrageergebnis.next
041                   stBezeichnung = stBezeichnung + CHR(13) +
                        oAbfrageergebnis.getString(1)
042                  stSql1 = "UPDATE ""tmp_Monat"" SET ""T_"+i+""" ='"+stBezeichnung+"'
                        WHERE ""ID""='"+inID+"'"
043                  oSQL_Anweisung1.executeUpdate(stSql1)
044               WEND
045            END IF
046         NEXT
047         inID = inID + 1
048      NEXT
049   END SUB
```

Die Prozedur "Monat" soll die Monatsübersicht für das Formular "Termine_Kalenderübersicht" erstellen.

Zuerst wird die alte Tabelle in Zeile 11 und 12 geleert. Eine Zurückstellung des AutoWertes ist nicht erforderlich, da der Primärschlüssel direkt eingegeben wird. Das Startdatum für die Tabelle wird auf den Monatsanfang gesetzt (Zeile 13), das Enddatum auf den Anfang des folgenden Monats (Zeile 14). Die anschließende Schleife wird so lange durchlaufen, bis durch Addition von Tagen zu dem Startdatum das Enddatum erreicht wird. Die Zunahme des Datums um jeweils einen Tag erfolgt in Zeile 31.

In Zeile 19 wird der Tag der Woche als Zahl ermittelt. Dieser Tag ist Bestandteil der Feldbezeichnung für die folgende Eingabe in die Tabelle. Dadurch wird gewährleistet, dass der erste Eintag nicht einfach in das erste Feld geschrieben wird und jeder Monat scheinbar mit einem Montag beginnt. Über das Schlüsselfeld und das Datum (als Text in deutschsprachiger Schreibweise) wird die erste Zeile in die Tabelle eingefügt (Zeile 20 und 21). Anschließend wird zu dem Datum nachgesehen, ob ein Eintrag in der Tabelle "Termine" vorhanden ist. Dieser wird dann gegebenenfalls an das bereits eingefügte Datum angehängt (Zeile 22 bis Zeile 29).

Von Zeile 30 bis Zeile 46 wird der gleiche Vorgang für alle folgenden Tage der Woche erneut durchgeführt. Anschließend wird in Zeile 47 der Wert des Schlüsselfeldes um 1 erhöht und die Eingabe für die nächste Woche kann ab Zeile 16 erfolgen.

Monatsansicht

Aufruf aus	
Makro:	*Monatsansicht_aktualisieren, Monat_aktualisieren*

Benötigt	
Makro:	*Monat*
Tabelle:	*Filter*

```
001  SUB Monatsansicht
002     DIM oForm AS OBJECT
003     DIM oSubForm AS OBJECT
004     DIM stBezeichnung AS STRING
005     DIM a()
006     oDoc = thisComponent
007     oDrawpage = oDoc.Drawpage
008     oForm = oDrawpage.Forms.getByName("Monat")
009     oSQL_Anweisung = oVerbindung.createStatement()
010     stSql = "SELECT ""Jahr_Monat"" FROM ""Filter"" WHERE ""ID"" = TRUE "
011     oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
012     WHILE oAbfrageergebnis.next
013         stBezeichnung = oAbfrageergebnis.getString(1)
014     WEND
015     a = Split(stBezeichnung,",")
016     Monat(CInt(a(1)),CInt(a(0)))
017     oSubForm = oForm.getByName("Ansicht")
018     oSubForm.reload()
019  END SUB
```

Diese Prozedur ist Teil der Prozedur, die die Monatsübersicht in dem Formular "Termine_Kalenderübersicht" darstellen soll.

Aus der Filtertabelle wird die Kombination "Jahr_Monat" ausgelesen. Hier sind die Jahreszahl und die Monatszahl durch ein Komma getrennt. Diese Trennung wird in Zeile 15 dazu genutzt, beide Zahlen in ein Array zu speichern. In Zeile 16 werden so die Monatszahl und die Jahreszahl an die Prozedur "Monat" weitergeleitet. Ist die Prozedur "Monat" abgelaufen, so wird das Formular, das die Ansicht beinhaltet, neu geladen.

Date_2_SQLDate

Aufruf aus	
Makro:	*Kalender, Monatskalender, Halbjahreskalender, Zeitplan, Monat, Bericht_Start, GruppeMultiUebersicht, FeiertageArray*

Benötigt	
Makro:	keins

```
001  FUNCTION Date_2_SQLDate(d AS DATE) AS STRING
002     DIM stMonth AS STRING
003     DIM stDay AS STRING
004     stMonth = Right("0" + Trim(Str(Month(d))),2)
005     stDay = Right("0" + Trim(Str(Day(d))),2)
006     Date_2_SQLDate = Trim(Str(Year(d))) +"-"+ stMonth +"-"+ stDay
007  END FUNCTION
```

Aus der internen Basic-Datumsangabe wird ein Datum in SQL-Schreibweise erzeugt. Sind die Monatsangaben oder Tagesangaben einstellig, so wird ihnen eine '0' vorangestellt. Dies geschieht einfach, indem eine '0' grundsätzlich vorangestellt wird, schließlich aber nur die letzten beiden Ziffern weitergegeben werden.

Date_2_Datum

Aufruf aus	
Makro:	*Monat*

Benötigt
Makro: keins

```
001 FUNCTION Date_2_Datum(d AS DATE) AS STRING
002    DIM stMonth AS STRING
003    DIM stDay AS STRING
004    stMonth = Right("0" + Trim(Str(Month(d))),2)
005    stDay = Right("0" + Trim(Str(Day(d))),2)
006    Date_2_Datum = stDay +"."+ stMonth +"."+ Trim(Str(Year(d)))
007 END FUNCTION
```

Aus der internen Basic-Datumsangabe wird ein Datum in deutschsprachiger Schreibweise mit vierstelliger Jahreszahl erzeugt. Sind die Monatsangaben oder Tagesangaben einstellig, so wird ihnen eine '0' vorangestellt. Dies geschieht einfach, indem eine '0' grundsätzlich vorangestellt wird, schließlich aber nur die letzten beiden Ziffern weitergegeben werden.

SQLDate_2_Datum

Aufruf aus
Makro: *Zeitplan*

Benötigt
Makro: keins

```
001 FUNCTION SQLDate_2_Datum(d AS STRING) AS STRING
002    DIM a()
003    DIM stDatum AS STRING
004    a = split(d,"-")
005    stDatum = a(2)+"."+a(1)+"."+Right(a(0),2)
006    SQLDate_2_Datum = stDatum
007 END FUNCTION
```

Das SQL-Datum hat den Aufbau YYYY-MM-DD. Für das Datum in deutschsprachiger Schreibweise wird hier eine Vertauschung von Jahreszahl und Tageszahl vorgenommen. Dann werden noch die Punkte als Trenner gesetzt und die Jahreszahl in zweistelliger Form ausgegeben.

Date_2_Monat

Aufruf aus
Makro: *Monatskalender*

Benötigt
Makro: keins

```
001 FUNCTION Date_2_Monat(d AS DATE) AS STRING
002    IF Month(d) = 1 THEN
003       Date_2_Monat = "Januar"
004    ELSEIF Month(d) = 2 THEN
005       Date_2_Monat = "Februar"
006    ELSEIF Month(d) = 3 THEN
007       Date_2_Monat = "März"
008    ELSEIF Month(d) = 4 THEN
009       Date_2_Monat = "April"
010    ELSEIF Month(d) = 5 THEN
011       Date_2_Monat = "Mai"
012    ELSEIF Month(d) = 6 THEN
013       Date_2_Monat = "Juni"
014    ELSEIF Month(d) = 7 THEN
015       Date_2_Monat = "Juli"
016    ELSEIF Month(d) = 8 THEN
```

```
017       Date_2_Monat = "August"
018    ELSEIF Month(d) = 9 THEN
019       Date_2_Monat = "September"
020    ELSEIF Month(d) = 10 THEN
021       Date_2_Monat = "Oktober"
022    ELSEIF Month(d) = 11 THEN
023       Date_2_Monat = "November"
024    ELSE
025       Date_2_Monat = "Dezember"
026    END IF
027 END FUNCTION
```

Mit Hilfe der Monatszahl wird hier jeweils ein entsprechender Monatsname ausgegeben. Diese Funktion könnte genauso gut durch eine globale Variable in Form eines Arrays ersetzt werden.

Datum_Ostersonntag

Aufruf aus
Makro: *FeiertageArray*

Benötigt
Makro: keins

```
001 FUNCTION Datum_Ostersonntag(inJahr AS INTEGER) AS LONG
002 REM Spencers Osterformel mit Ganzzahldivision \
003    DIM a AS INTEGER
004    DIM b AS INTEGER
005    DIM c AS INTEGER
006    DIM d AS INTEGER
007    DIM e AS INTEGER
008    DIM f AS INTEGER
009    DIM g AS INTEGER
010    DIM h AS INTEGER
011    DIM i AS INTEGER
012    DIM k AS INTEGER
013    DIM l AS INTEGER
014    DIM m AS INTEGER
015    DIM n AS INTEGER
016    DIM p AS INTEGER
017    DIM OS AS INTEGER
018    DIM loDatum AS LONG
019    a = inJahr MOD 19
020    b = inJahr \ 100
021    c = inJahr MOD 100
022    d = b \ 4
023    e = b MOD 4
024    f = (b + 8) \ 25
025    g = (b - f + 1) \ 3
026    h = (19*a + b - d - g + 15) MOD 30
027    i = c \ 4
028    k = c MOD 4
029    l = ( 32 + 2*e + 2*i - h - k ) MOD 7
030    m = ( a + 11*h + 22*l ) \ 451
031    n = ( h + l - 7*m + 114 ) \ 31 'Monat
032    p = ( h + l - 7*m + 114 ) MOD 31
033    Os = p + 1 'Tag
034    loDatum = DateSerial(inJahr, n, Os)
035    Datum_Ostersonntag() = loDatum
036 END FUNCTION
```

Diese Funktion ist eine direkte Umsetzung von *https://de.wikipedia.org/wiki/Spencers_Osterformel*. Aus der Jahresvorgabe wird so ein Datum für den Ostersonntag ermittelt, mit dem innerhalb von Basic aus weiter gerechnet werden kann. So können die Feiertage für ein Jahr bestimmt werden, die von der Lage des Ostersonntags abhängig sind.

FeiertageArray

Aufruf aus
Makro: *Kalender, GruppeMultiUebersicht*

Benötigt
Makro: *Datum_Ostersonntag, Date_2_SQLDate*

```
001 FUNCTION FeiertageArray(inJahr AS INTEGER) AS OBJECT
002    DIM d_Ostersonntag AS DATE
003    DIM Feiertage(14,1)
004    d_Ostersonntag = CDate(Datum_Ostersonntag(inJahr))
005    Feiertage(0,0) = "Neujahr"
006    Feiertage(0,1) = Trim(Str(inJahr)) + "-01-01"
007    Feiertage(1,0) = "Rosenmontag"
008    Feiertage(1,1) = Date_2_SQLDate(DateAdd("d",-48,d_Ostersonntag))
009    Feiertage(2,0) = "Karfreitag"
010    Feiertage(2,1) = Date_2_SQLDate(DateAdd("d",-2,d_Ostersonntag))
011    Feiertage(3,0) = "Ostersonntag"
012    Feiertage(3,1) = Date_2_SQLDate(d_Ostersonntag)
013    Feiertage(4,0) = "Ostermontag"
014    Feiertage(4,1) = Date_2_SQLDate(DateAdd("d",1,d_Ostersonntag))
015    Feiertage(5,0) = "Chr.Himmelfahrt"
016    Feiertage(5,1) = Date_2_SQLDate(DateAdd("d",39,d_Ostersonntag))
017    Feiertage(6,0) = "Pfingstsonntag"
018    Feiertage(6,1) = Date_2_SQLDate(DateAdd("d",49,d_Ostersonntag))
019    Feiertage(7,0) = "Pfingstmontag"
020    Feiertage(7,1) = Date_2_SQLDate(DateAdd("d",50,d_Ostersonntag))
021    Feiertage(8,0) = "Fronleichnam"
022    Feiertage(8,1) = Date_2_SQLDate(DateAdd("d",60,d_Ostersonntag))
023    Feiertage(9,0) = "Tag der dt. Einheit"
024    Feiertage(9,1) = Trim(Str(inJahr)) + "-10-03"
025    Feiertage(10,0) = "Allerheiligen"
026    Feiertage(10,1) = Trim(Str(inJahr)) + "-11-01"
027    Feiertage(11,0) = "Heiligabend"
028    Feiertage(11,1) = Trim(Str(inJahr)) + "-12-24"
029    Feiertage(12,0) = "1. Weihnachtstag"
030    Feiertage(12,1) = Trim(Str(inJahr)) + "-12-25"
031    Feiertage(13,0) = "2. Weihnachtstag"
032    Feiertage(13,1) = Trim(Str(inJahr)) + "-12-26"
033    Feiertage(14,0) = "Tag der Arbeit"
034    Feiertage(14,1) = Trim(Str(inJahr)) + "-05-01"
035    FeiertageArray = Feiertage
036 END FUNCTION
```

In Abhängigkeit vom Ostersonntag werden viele Feiertage berechnet. Die Feiertage werden mit Datum und Bezeichnung in einem Array gespeichert. Das Datum wird dabei in der für SQL notwendigen amerikanischen Schreibweise zwischengespeichert.

BerichtZeilenhoeheAuto

Aufruf aus
Makro: *Bericht_Start*

Benötigt
Makro: keins

```
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  END SUB
```

Diese Prozedur sucht in dem fertigen Bericht alle Tabellen auf, die die Namensbezeichnung "Detail" haben. Die Zeilen in diesem Bereich werden dann auf automatische Höhe eingestellt.

Ab LO 6.4 wäre dies nicht mehr nötig, da dort die AutoGrow-Eigenschaft in den ReportBuilder aufgenommen wurde.

Bericht_Start

Aufruf aus	
Formular:	*Terminart_Termin, Termine_Kalenderübersicht, Termine_Monat_Terminart, Termine_Tabelleneingabe*

Benötigt	
Makro:	*Kalender, Monatskalender, Wochenkalender_halb, Halbjahreskalender, Zeitplan, GruppeMultiUebersicht, BerichtZeilenhoeheAuto, DialogStart, Date_2_SQLDate*
Tabelle:	*Zielgruppe, Filter*

```
001  SUB Bericht_Start(oEvent AS OBJECT)
002     DIM oForm AS OBJECT
003     DIM oFeld_1 AS OBJECT
004     DIM oFeld_2 AS OBJECT
005     DIM oFeld_3 AS OBJECT
006     DIM oFeld_4 AS OBJECT
007     DIM oReport AS OBJECT
008     DIM daStart AS DATE
009     DIM daEnde AS DATE
010     DIM stKalender AS STRING
011     DIM inZielgruppe AS INTEGER
012     DIM stTermin AS STRING
013     DIM stGruppeMulti AS STRING
014     DIM stHalbjahr AS STRING
015     DIM inGruppeMulti AS INTEGER
016     oForm = oEvent.Source.Model.Parent
017     oForm.updateRow
018     Wait 1000
019     oFeld_1 = oForm.getByName("datStart")
020     oFeld_2 = oForm.getByName("datEnde")
021     oFeld_3 = oForm.getByName("lboKalenderart")
022     oFeld_4 = oForm.getByName("lboZielgruppe")
023     oFeld_5 = oForm.getByName("lboGruppeMulti")
024     stKalender = Listenfeld_Text(oFeld_3)
025     inZielgruppe = ID_Ermittlung(oFeld_4)
026     oSQL_Anweisung = oVerbindung.createStatement()
027     stSql = "SELECT ""Termingruppe"" FROM ""Zielgruppe"" WHERE ""ID""
           ='"+inZielgruppe+"' "
028     oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
029     WHILE oAbfrageergebnis.next
```

```
030         stTermin = oAbfrageergebnis.getString(1)
031     WEND
032     IF stKalender <>"" THEN
033        IF NOT isEmpty(oFeld_1.currentValue) THEN
034              daStart = Datumswert(oFeld_1)
035           IF NOT isEmpty(oFeld_2.currentValue)  THEN
036              daEnde = Datumswert(oFeld_2)
037           ELSE
038              daEnde = "1900-01-01"
039           END IF
040            IF daEnde < daStart THEN
041              daEnde = DateAdd("yyyy",1,daStart)
042           END IF
043        ELSE
044           daStart = now
045           daEnde = DateAdd("yyyy",1,daStart)
046        END IF
047        IF stKalender = "Terminübersicht" THEN
048           Zeitplan(daStart,daEnde,stTermin)
049           oReport = ThisDatabaseDocument.ReportDocuments.getByName("Zeitplan").open
050           BerichtZeilenhoeheAuto(oReport)
051        ELSEIF stKalender = "Halbjahreskalender" THEN
052           Kalender(YEAR(daStart),0,stTermin)
053           Halbjahreskalender(YEAR(daStart))
054           ThisDatabaseDocument.ReportDocuments.getByName(
                 "Halbjahreskalender_A4_quer").open
055        ELSEIF stKalender = "Monate Montag - Sonntag" THEN
056           Kalender(YEAR(daStart),1,stTermin)
057           Monatskalender(daStart,daEnde)
058          ThisDatabaseDocument.ReportDocuments.getByName("Monatskalender_Mo_So").open
059        ELSEIF stKalender = "Monate Montag - Freitag" THEN
060           Kalender(YEAR(daStart),1,stTermin)
061           Monatskalender(daStart,daEnde)
062          ThisDatabaseDocument.ReportDocuments.getByName("Monatskalender_Mo_Fr").open
063        ELSEIF stKalender = "Halbwochenkalender als Notizbuch" THEN
064           Kalender(YEAR(daStart),1,stTermin)
065           Monatskalender(daStart,daEnde)
066           Wochenkalender_halb(daStart,daEnde)
067           ThisDatabaseDocument.ReportDocuments.getByName("Wochenkalender_halb").open
068        ELSEIF stKalender = "Wochentage Mensaabo" THEN
069           Kalender(YEAR(daStart),1,stTermin)
070           ThisDatabaseDocument.ReportDocuments.getByName("Wochentage_Mensaabo").open
071        ELSEIF stKalender = "Klassenarbeitenübersicht" THEN
072           stSql = "UPDATE ""Filter"" SET ""Startjahr"" = '"+Year(daStart)+"' WHERE
                 ""ID"" = TRUE "
073           oSQL_Anweisung.executeUpdate(stSql)
074           IF NOT isEmpty(oFeld_5.currentValue) THEN
075              stGruppeMulti = Listenfeld_Text(oFeld_5)
076              inGruppeMulti = ID_Ermittlung(oFeld_5)
077              GruppeMultiUebersicht(YEAR(daStart),1,stTermin,inGruppeMulti,
                    stGruppeMulti)
078              ThisDatabaseDocument.ReportDocuments.getByName(
                    "Liste_Gruppe_multi").open
079           ELSE
080              msgbox "Ein Jahrgang muss ausgewählt werden."
081           END IF
082        ELSEIF stKalender = "Klassenarbeitenübersicht 1. HJ" THEN
083           stSql = "UPDATE ""Filter"" SET ""Startjahr"" = '"+Year(daStart)+"' WHERE
                 ""ID"" = TRUE "
084           oSQL_Anweisung.executeUpdate(stSql)
085           stSql = "SELECT ""Halbjahrende"" FROM ""Filter"" WHERE ""ID"" = TRUE AND
                 ""Halbjahrende"" BETWEEN '"+Date_2_SQLDate(daStart)+"' AND
                 '"+Date_2_SQLDate(daEnde)+"'"
086           oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
087           WHILE oAbfrageergebnis.next
088              stHalbjahr = oAbfrageergebnis.getString(1)
```

```
089           WEND
090           IF stHalbjahr = "" THEN
091              DialogStart
092           END IF
093           IF NOT isEmpty(oFeld_5.currentValue) THEN
094              stGruppeMulti = Listenfeld_Text(oFeld_5)
095              inGruppeMulti = ID_Ermittlung(oFeld_5)
096              GruppeMultiUebersicht(YEAR(daStart),1,stTermin,inGruppeMulti,
                    stGruppeMulti)
097              ThisDatabaseDocument.ReportDocuments.getByName(
                    "Liste_Gruppe_multi_1HJ").open
098           ELSE
099              msgbox "Ein Jahrgang muss ausgewählt werden."
100           END IF
101        ELSEIF stKalender = "Klassenarbeitenübersicht 2. HJ" THEN
102           stSql = "UPDATE ""Filter"" SET ""Startjahr"" = '"+Year(daStart)+"' WHERE
                 ""ID"" = TRUE "
103           oSQL_Anweisung.executeUpdate(stSql)
104           stSql = "SELECT ""Halbjahrende"" FROM ""Filter"" WHERE ""ID"" = TRUE AND
                 ""Halbjahrende"" BETWEEN '"+Date_2_SQLDate(daStart)+"' AND
                 '"+Date_2_SQLDate(daEnd)+"'"
105           oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
106           WHILE oAbfrageergebnis.next
107              stHalbjahr = oAbfrageergebnis.getString(1)
108           WEND
109           IF stHalbjahr = "" THEN
110              DialogStart
111           END IF
112           IF NOT isEmpty(oFeld_5.currentValue) THEN
113              stGruppeMulti = Listenfeld_Text(oFeld_5)
114              inGruppeMulti = ID_Ermittlung(oFeld_5)
115              GruppeMultiUebersicht(YEAR(daStart),1,stTermin,inGruppeMulti,
                    stGruppeMulti)
116              ThisDatabaseDocument.ReportDocuments.getByName(
                    "Liste_Gruppe_multi_2HJ").open
117           ELSE
118              msgbox "Ein Jahrgang muss ausgewählt werden."
119           END IF
120        ELSE
121        END IF
122     END IF
123  END SUB
```

Von der Prozedur "Bericht_Start" aus werden alle Berichte gestartet. Kein Bericht kann mit aktuellen Daten direkt gestartet werden, da die Daten erst einmal aus den Eingabetabellen so aufgearbeitet werden müssen, dass sie für den Bericht verwertbar sind. Das geschieht, indem in verschiedenen Prozeduren die mit "tmp_" beginnenden Tabelle mit Inhalten befüllt werden.

Zu Beginn wird der Inhalt der Formularfelder in die Tabelle "Filter" abgespeichert (Zeile 17). Hier ist anschließend eine Pause von 1 Sekunde eingebaut (Zeile 18), damit diese Speicherung auch tatsächlich einwandfrei funktioniert.

Von Zeile 19 bis Zeile 23 werden die Objekte für die verschiedenen Formularfelder erstellt. In Zeile 24 und Zeile 25 werden mittels bereits definierter Funktionen die entscheidenden Werte der Listenfelder ermittelt. Von Zeile 27 bis Zeile 31 wird die zu der Zielgruppe passende Termingruppe gesucht und in der Variablen **stTermin** abgespeichert.

Die weiteren Schritte werden nur durchlaufen, wenn in dem Listenfeld für den gewünschten Kalender auch ein Eintrag vorhanden war (Zeile 32).

Die Zeilen 33 bis 46 dienen dazu, auf jeden Fall ein brauchbares Datum zur weiteren Verarbeitung zu übermitteln. Wenn das Startdatum nicht leer ist wird noch nachgeschaut, ob das Enddatum auch nicht leer ist. Liegt das Enddatum vor dem Startdatum, dann wird der Wert ignoriert und einfach ein Jahr zu dem Startdatum hinzugezählt. Enthält das Startdatum keinen Wert, so wird hier einfach von dem aktuellen Datum ausgegangen und das Enddatum wieder ein Jahr später gesetzt.

In den folgenden Schritten wird jeweils überprüft, welcher Eintrag in **stKalender** steht. Zu dem jeweiligen Eintrag werden dann die verschiedenen Prozeduren aufgerufen bevor zum Schluss auch der dazu passende Bericht gestartet wird. Hier nur zwei Beispiele:

- Terminübersicht (Zeile 47 bis 50): Zuerst wird die Prozedur "Kalender" gestartet. Anschließend wird der Bericht aufgerufen, hier allerdings so, dass er für die Prozedur weiter als Variable zur Verfügung steht. Zum Schluss werden die Zeilen in dem Detailbereich des Berichtes auf automatische Höhe eingestellt.
- Klassenarbeiten Übersicht 2. HJ. (Zeilen 101 bis 119): Zuerst wird das Startjahr in die Tabelle "Filter" eingetragen. Danach wird nachgeschaut, ob in der Tabelle "Filter" ein Eintrag für das "Halbjahrende" existiert, der außerdem auch noch zwischen dem Startdatum und dem Enddatum liegt. Ist das nicht der Fall, dann wird der Dialog für die Eingabe des Halbjahresdatums gestartet. Als letzte Hürde wird überprüft, ob denn eine Auswahl eines Jahrgangs erfolgt ist. Ist dies der Fall, dann startet die Prozedur "GruppeMultiUebersicht" und anschließend dann der Bericht "Liste_Gruppe_multi_2HJ". Wurde kein Jahrgang ausgewählt, so erfolgt die Meldung, dass noch ein Jahrgang ausgewählt werden muss bevor der Bericht zusammengestellt werden kann.

Backup

Die Absicherung von Datenbankdateien sollte regelmäßig erfolgen. Eine Sicherheitskopie erstellt LibreOffice hiervon nicht automatisch, so dass dafür etwas nachgeholfen werden sollte.

Datenbankbackup

Aufruf aus
Makro: *Datenbankstart, Backup_sofort*

Benötigt
Makro: keins

```
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
017    oPath = createUnoService("com.sun.star.util.PathSettings")
018    FOR i = 1 TO inMax + 1
019       IF NOT FileExists(oPath.Backup & "/" & i & "_" & sTitel) THEN
020          IF i > inMax THEN
021             FOR k = inMax - 1 TO 1 STEP -1
022                IF FileDateTime(oPath.Backup & "/" & k & "_" & sTitel) <=
                      FileDateTime(oPath.Backup & "/" & k+1 & "_" & sTitel) THEN
023                   IF k = 1 THEN
024                      i = k
025                      EXIT FOR
026                   END IF
027                ELSE
028                   i = k+1
```

```
029                     EXIT FOR
030                 END IF
031             NEXT
032         END IF
033         EXIT FOR
034       END IF
035    NEXT
036    sUrl_Ziel = oPath.Backup & "/" & i &"_" & sTitel
037    FileCopy(sUrl_Start,sUrl_Ziel)
038 END SUB
```

Das Prinzip der Makros ist erst einmal, lediglich eine Kopie der Datenbankdatei in das Backup-Verzeichnis von LibreOffice zu befördern. Dort soll nicht sofort die vorhergehende Kopie gelöscht und die neue Kopie eingefügt werden. Stattdessen soll eine vorher festgelegte Anzahl an Kopien angelegt werden bevor die älteste dieser Kopien entfernt wird.

Zuerst wird ein Bezug zur Datenbankdatei hergestellt (Zeile 9). Von der Datei wird der Pfad ausgelesen. Falls das Makro nicht direkt von der Base-Datei aus gestartet wird, sondern vielleicht aus einem Formular ausgelöst wird, muss der Pfad aus dem nächst höheren Element **parent** ausgelesen werden (Zeile 12 bis Zeile 16). Anschließend wird der Titel der Datei (der hier nicht vorher durch ein Makro geändert werden darf) ausgelesen und in einer Schleife von Zeile 18 bis Zeile 35 laufend als neue Datei im Backupverzeichnis mit einem Zähler versehen ausgetestet (Zeile 19).

Nur wenn die Backupdatei nicht existiert und wenn der Zähler größer ist als das vorgegebene Maximum an Dateien geht es in die weitere Verschachtelung. Ist der Zähler kleiner und existiert die Datei nicht, so wird die Schleife mit **EXIT FOR** (Zeile 33) beendet.

Solange noch nicht geklärt ist, wie die Datei denn nun heißen darf, erfolgt rückwärts zählend ab der Kopie mit der höchsten Nummer ein Vergleich. Ist die Datei mit der aktuellen Nummer vom Zeitstempel her kleiner oder gleich der Datei, die die nächst höhere Nummer hat. Solange das der Fall ist wird weiter gesucht. Wird stattdessen eine neuere Datei mit größerem Zeitstempel gefunden, so muss die neu einzufügende Datei die Nummer bekommen, die um 1 größer ist als diese neuere Datei (Zeile 27 bis Zeile 30). Ist dies nie der Fall, so wird die Datei mit der Nummer 1 ersetzt (Zeile 23 bis Zeile 26).

Damit ist klar, welchen Zähler die Backupdatei erhalten soll. Die Ziel-URL wird erstellt (Zeile 36) und die Ausgangsdatei zur Ziel-URL hin kopiert (Zeile 37).

Backup_sofort

Aufruf aus
Datenbankdatei, direkt in die Symbolleiste neben dem Speicherbutton integriert

Benötigt
Makro: *Datenbankbackup*

```
001 SUB Backup_sofort
002    oVerbindung.flush
003    Datenbankbackup(10)
004 END SUB
```

Über einen Button z.B. in der Symbolleiste der Base-Datei kann diese Prozedur ausgelöst werden. Zuerst werden die Daten mit einem **flush** in die Tabelle übertragen, die in der Datenbankdatei steckt. Danach wird die Prozedur zum Backup der Datenbankdatei gestartet. Die '10' sagt hier lediglich, dass bis zu 10 Backups aufbewahrt werden und danach das älteste Backup überschrieben wird.

Eingabe

Die Prozeduren in diesem Modul beeinflussen die Eingabe von Daten über die Formulare.

DatenSpeichern

Aufruf aus	
Makro:	*DatenAktualisieren_Termine, DatenAktualisieren_Terminort_Termine*

Benötigt	
Makro:	keins

```
001  SUB DatenSpeichern
002     DIM oDocument AS OBJECT
003     DIM oDispatcher AS OBJECT
004     oDocument = ThisComponent.CurrentController.Frame
005     oDispatcher = createUnoService("com.sun.star.frame.DispatchHelper")
006     oDispatcher.executeDispatch(oDocument, ".uno:RecSave", "", 0, Array())
007  END SUB
```

Die ist ein allgemeiner Aufruf zum Speichern innerhalb eines Formulars. **UNO** (**Universal Network Object**) dient hier dazu, über den Dispatcher schnell an eine komplexere Funktion zu kommen. Der direkte Weg zu dieser Funktionalität würde bedeuten, dass zuerst abgeklärt werden muss, ob die Daten bereits abgespeichert sind. Dann muss ein "Update" erfolgen. Ansonsten ist ein "Insert" notwendig. **.uno:RecSave** fasst diese beiden Möglichkeiten in einem Objekt zusammen.

DatenAktualisieren_Termine

Aufruf aus	
Formular:	*Termine_Monat_Terminart*

Benötigt	
Makro:	*DatenSpeichern*

```
001  SUB DatenAktualisieren_Termine
002     DIM oForm AS OBJECT
003     DIM oSubForm_1 AS OBJECT
004     DIM oSubForm_2 AS OBJECT
005     DatenSpeichern
006     oDoc=thisComponent
007     oDrawpage=oDoc.drawpage
008     oForm = oDrawpage.Forms.getByName("MainForm")
009     oSubForm_1 = oForm.getByName("SubForm")
010     oSubForm_2 = oForm.getByName("SubTerminart")
011     oSubForm_1.reload()
012     oSubForm_2.reload()
013  END SUB
```

Abhängig von der Eingabe im Feld "BeginnDatum" oder im Feld "TerminArt" sollen die Inhalte der Unterformulare neu eingelesen werden.

In Zeile 5 wird die Prozedur "DatenSpeichern" aufgerufen, die unabhängig davon funktioniert, ob es sich um ein Update oder um einen Insert handelt. Anschließend werden die Unterformulare über den Formularstrang aufgerufen. Beide Formulare werden nacheinander (Zeile 11 und Zeile 12) neu geladen.

DatenAktualisieren_Terminort_Termine

Aufruf aus	
Formular:	*Terminart_Termin*

Benötigt
Makro: *DatenSpeichern*

```
001 SUB DatenAktualisieren_Terminart_Termine
002    DIM oForm AS OBJECT
003    DIM oSubForm AS OBJECT
004    DIM oSubSubForm_1 AS OBJECT
005    DIM oSubSubForm_2 AS OBJECT
006    DatenSpeichern
007    oDoc=thisComponent
008    oDrawpage=oDoc.drawpage
009    oForm = oDrawpage.Forms.getByName("MainForm")
010    oSubForm = oForm.getByName("SubTermine")
011    oSubSubForm_1 = oSubForm.getByName("SubTerminMonat")
012    oSubSubForm_2 = oSubForm.getByName("SubTerminOrt")
013    oSubSubForm_1.reload()
014    oSubSubForm_2.reload()
015 END SUB
```

Nach einer Datensatzänderung im Unterformular in dem Feld für das "BeginnDatum "sollen die UnterUnterformulare auf diese Änderung entsprechend eingestellt werden. Da zu dem Zeitpunkt auch der Ort bereits eingegeben wurde können sich Zeit und Ort, also die Parameter für die beiden Unterformulare, geändert haben.

In Zeile 6 wird die Prozedur "DatenSpeichern" aufgerufen, die unabhängig davon funktioniert, ob es sich um ein Update oder um einen Insert handelt. Anschließend werden die UnterUnterformulare über den gesamten Formularstrang aufgerufen. Beide Formulare werden nacheinander (Zeile 13 und Zeile 14) neu geladen.

LetzterDatensatz

Aufruf aus
Formular: *Termine_Monat_Terminart*
Makro: *LetzterDatensatz_Termine_Kalender*

Benötigt
Makro: keins

```
001 SUB LetzterDatensatz(oEvent AS OBJECT)
002    DIM oForm AS OBJECT
003    oForm = oEvent.Source
004    oForm.last()
005 END SUB
```

Beim Laden des Formulars wird das Formular direkt auf den letzten Datensatz gesetzt, damit anschließend mit einer Neueingabe begonnen werden kann.

Das Formular ist die Quelle für das auslösende Ereignis, **`oEvent.Source`**. Mit **`last()`** wird der letzte Datensatz aufgesucht.

LetzterDatensatz_Termine_Kalender

Aufruf aus
Formular: *Termine_Kalenderübersicht*

Benötigt
Makro: *LetzterDatensatz, Monatsansicht_aktualisieren*

```
001 SUB LetzterDatensatz_Termine_Kalender(oEvent AS OBJECT)
```

```
002     LetzterDatensatz(oEvent)
003     Monatsansicht_aktualisieren
004 END SUB
```

Beim Laden des Formulars soll der Nutzer nicht erst umständlich zum letzten Datensatz navigieren. Das Formular hat hier schließlich kein übersichtliches Tabellenkontrollfeld, wo das nur ein Klick mit der Maus ist. Deswegen wird zuerst das Formular mit der Prozedur "Letzter Datensatz" auf den letzten Datensatz eingestellt und danach die Prozedur "Monatsansicht_aktualisieren" aufgerufen, damit das Datum des letzten Datensatzes auch in der Monatsansicht zu finden ist.

Anfangswert_Datum

Aufruf aus
Formular: *Termine_Kalenderübersicht*

Benötigt
Makro: *Datumswert*

```
001 SUB Anfangswert_Datum(oEvent AS OBJECT)
002     DIM oFeld AS OBJECT
003     oFeld = oEvent.Source.Model
004     IF NOT ISEMPTY(oFeld.getCurrentValue()) THEN
005         daStartDat = Datumswert(oFeld)
006     END IF
007 END SUB
```

Diese Prozedur wird ausgelöst, wenn auf das Datumsfeld in dem Formular zugegriffen wird, das die komplette Monatsübersicht darstellt.

Zuerst wird lediglich das Feld aus dem auslösenden Ereignis abgeleitet (Zeile 3). Ist dieses Feld nicht leer (Zeile 4), so wird aus dem Feld der entsprechende Inhalt für das Datum ausgelesen, durch die Funktion Datumswert in eine **Cdate**-Variable überführt und in der globalen Variablen **daStartDat** abgespeichert (Zeile 5). Diese Abspeicherung ist für die Prozedur Monatsansicht_aktualisieren wichtig. Hier muss erkannt werden, ob beim Verlassen des Feldes eine Datumsänderung erfolgt ist oder nur mit dem Cursor einmal das Feld betreten und wieder verlassen wurde.

Monatsansicht_aktualisieren

Aufruf aus
Formular: *Termine_Kalenderübersicht*
Makro: *LetzterDatensatz_Termine_Kalender*

Benötigt
Makro: *Datumswert, Monatsansicht*
Tabelle: *Filter*

```
001 SUB Monatsansicht_aktualisieren
002     DIM oForm AS OBJECT
003     DIM oForm_2 AS OBJECT
004     DIM oFeld_0 AS OBJECT
005     DIM oFeld_1 AS OBJECT
006     DIM oFeld_2 AS OBJECT
007     DIM daStart AS DATE
008     DIM stMonat AS STRING
009     DIM stBezeichnung AS STRING
010     oDoc = thisComponent
```

```
011     oDrawpage = oDoc.Drawpage
012     oForm = oDrawpage.Forms.getByName("MainForm")
013     oFeld_1 = oForm.getByName("datBeginn_Datum")
014     IF NOT ISEMPTY(oFeld_1.getCurrentValue()) THEN
015        IF oForm.getString(oForm.findColumn("ID")) = "" THEN
016           oForm.insertRow()
017        ELSE
018           oForm.updateRow()
019        END IF
020        oSQL_Anweisung = oVerbindung.createStatement()
021        stSql = "SELECT ""Jahr_Monat"" FROM ""Filter"" WHERE ""ID"" = TRUE "
022        oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
023        WHILE oAbfrageergebnis.next
024           stBezeichnung = oAbfrageergebnis.getString(1)
025        WEND
026        daStart = Datumswert(oFeld_1)
027        IF daStartDat <> daStart OR stBezeichnung = "" THEN
028           IF isDate(daStart) THEN
029              stMonat = Trim(str(Year(daStart)))+","+
                    Right("0"+Trim(str(Month(daStart))),2)
030              stSql = "UPDATE ""Filter"" SET ""Jahr_Monat"" = '"+stMonat+"' WHERE
                    ""ID"" = TRUE "
031              oSQL_Anweisung.executeUpdate(stSql)
032              oForm_2 = oDrawpage.Forms.getByName("Monat")
033              oFeld_2=oForm_2.getByName("lboMonat_Jahr")
034              oFeld_2.refresh()
035              oForm_2.reload()
036              Monatsansicht
037              daStartDat = daStart
038           END IF
039        END IF
040     END IF
041  END SUB
```

In dem Formular mit der Monatsübersicht soll die Übersicht in dem Moment aktualisiert werden, wenn bei einer Neueingabe oder einer Datensatzänderung das Feld mit dem "BeginnDatum" verlassen wird. Bei einem Monatswechsel kann sonst während der Eingabe der Termin nicht richtig in seinem Umfeld eingeordnet werden.

In Zeile 13 wird das Monatsfeld aufgesucht und dann in Zeile 14 überprüft, ob das Feld nicht zufällig leer ist. Dann würde eine Neueinstellung ja keinen Sinn ergeben. Ist das Feld nicht leer, so läuft die Prozedur weiter. Ansonsten endet sie hier.

Zeile 15 dient zur Überprüfung, ob der Datensatz schon vorhanden ist. Enthält das Feld "ID" der zugrundeliegenden Datenquelle keinen Eintrag, dann muss der Datensatz neu eingefügt werden: **`insertRow()`**. Andernfalls ist ein **`updateRow()`** erforderlich. Das Feld "ID" wird hier mit der Methode **`getString`** abgefragt, damit auf einen leeren String hin überprüft werden kann. Wenn das Feld mit einem **`getLong`** oder **`getInt`** abgefragt würde, dann wäre das Ergebnis selbst bei einem leeren Feld '0' – schlecht zu überprüfen, wenn doch eine '0' auch als Primärschlüssel vorkommen darf.

Von Zeile 20 bis Zeile 25 wird ermittelt, welcher Eintrag in der Filtertabelle für das Feld "Jahr_Monat" vorhanden ist. Nach diesem Feld wird schließlich die Monatsansicht gefiltert.

Der Basic-Interne Datumswert für das auslösende Datumsfeld wird ermittelt. Anschließend wird dieser Wert mit der zu Beginn abgespeicherten globalen Variable **`daStartDat`** verglichen (Zeile 27). Wenn sich die Werte unterscheiden oder wenn in der Filter-Tabelle kein Eintrag für das Feld "Jahr_Monat" zu finden ist, dann läuft die Prozedur weiter. Ansonsten endet sie hier. Das Neueinlesen macht ja keinen Sinn, wenn die Übersicht bereits auf den aktuellen Monat eingestellt ist.

Das neu eingegebene Datum wird jetzt zu einer Kombination von Monat und Jahr umgeschrieben (Zeile 29) und in die Filtertabelle eingetragen (Zeile 30 und Zeile 31).

Das Listenfeld über der Datumsansicht, das dazu dient, die Datumsansicht umzustellen, wird neu eingelesen (Zeile 34). Es könnte ja sein, dass der neue Eintrag noch nicht mit vorherigen Einträgen in der Terminübersicht übereinstimmt und deshalb im Listenfeld noch nicht vorhanden ist. Anschließend wird das Formular, in dem sich das Listenfeld befindet, neu geladen (Zeile 35).

Anschließend wird die Prozedur "Monatsansicht" ausgelöst, die die Gesamtansicht bei der Monatsübersicht im Formular steuert.

Zum Abschluss wird die globale Variable **daStartDat** mit dem neuen Wert **daStart** überschieben, damit beim nächsten Eintrag wieder davon ausgegangen werden kann, dass die Datumsübersicht auf **daStartDat** eingestellt ist.

Monat_aktualisieren

Aufruf aus
Formular: *Termine_Kalenderübersicht*

Benötigt	
Makro:	*Monatsansicht*

```
001  SUB Monat_aktualisieren(oEvent AS OBJECT)
002     DIM oForm AS OBJECT
003     DIM oFeld AS OBJECT
004     oFeld = oEvent.Source.Model
005     oForm = oFeld.Parent
006     oFeld.commit()
007     oForm.updateRow()
008     Monatsansicht
009  END SUB
```

Diese Prozedur ist in recht allgemeingültiger Weise geschrieben. Sie wird ausgelöst durch ein Listenfeld im Formular. Das Feld wird über das auslösende Ereignis in Zeile 4 ermittelt. Das Feld steckt in einem Formular. Das Formular ist **Parent** für das Feld (Zeile 5).

Bei Änderung des Feldes wird der Inhalt an die darunterliegende Tabelle übertragen und über das Formular mit einem **updateRow()** abgespeichert. Hier gilt nur ein **updateRow()**, weil es sich um die Filtertabelle handelt, bei der der Datensatz immer nur ausgetauscht wird.

Anschließend wird die Prozedur "Monatsansicht" ausgelöst, die die Gesamtansicht bei der Monatsübersicht im Formular steuert.

Termindaten_uebertragen

Aufruf aus
Formular: *Termine_Kalenderübersicht*

Benötigt	
Makro:	keins
Tabelle:	*Termine*

```
001  SUB Termindaten_uebertragen
002     DIM oForm AS OBJECT
003     DIM oFeld_0 AS OBJECT
004     DIM oFeld_1 AS OBJECT
005     DIM oFeld_2 AS OBJECT
006     DIM oFeld_3 AS OBJECT
007     DIM oFeld_4 AS OBJECT
008     DIM oFeld_5 AS OBJECT
```

```
009     DIM oFeld_6 AS OBJECT
010     DIM stBezeichnung AS STRING
011     DIM stBeginn AS STRING
012     DIM stEnde AS STRING
013     DIM stOrt AS STRING
014     DIM stBemerkung AS STRING
015     DIM startID AS STRING
016     oDoc = thisComponent
017     oDrawpage = oDoc.Drawpage
018     oForm = oDrawpage.Forms.getByName("MainForm")
019     IF oForm.getString(oForm.findColumn("ID")) = "" THEN
020        oFeld_1=oForm.getByName("comBezeichnung")
021        stBezeichnung = oFeld_1.getCurrentValue()
022        oSQL_Anweisung = oVerbindung.createStatement()
023        stSql = "SELECT ""Beginn_Uhrzeit"", ""Ende_Uhrzeit"", ""Ort"", ""Bemerkung"",
              ""artID"" FROM ""Termine"" WHERE ""ID"" = ( SELCET MAX(""ID"") FROM
              ""Termine"" WHERE ""Bezeichnung"" = '"+stBezeichnung+"' )"
024        oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
025        WHILE oAbfrageergebnis.next
026            stBeginn = oAbfrageergebnis.getString(1)
027            stEnde = oAbfrageergebnis.getString(2)
028            stOrt = oAbfrageergebnis.getString(3)
029            stBemerkung = oAbfrageergebnis.getString(4)
030            startID = oAbfrageergebnis.getString(5)
031        WEND
032        oFeld_2=oForm.getByName("timBeginn_Uhrzeit")
033        oFeld_3=oForm.getByName("timEnde_Uhrzeit")
034        oFeld_4=oForm.getByName("comOrt")
035        oFeld_5=oForm.getByName("txtBemerkung")
036        oFeld_6=oForm.getByName("lboartID")
037        IF stBeginn <> "" THEN
038           oFeld_2.BoundField.updateString(stBeginn)
039        END IF
040        IF stEnde <> "" THEN
041           oFeld_3.BoundField.updateString(stEnde)
042        END IF
043        IF stOrt <> "" THEN
044           oFeld_4.BoundField.updateString(stOrt)
045        END IF
046        IF stBemerkung <> "" THEN
047           oFeld_5.BoundField.updateString(stBemerkung)
048        END IF
049        IF startID <> "" THEN
050           oFeld_6.BoundField.updateString(startID)
051        END IF
052     END IF
053  END SUB
```

Dieses Makro läuft nur ab, wenn ein neuer Datensatz erstellt wird, das Feld "ID" also noch keinen Wert enthält (Zeile 19). Es wird von dem Feld "comBezeichnung" (Zeile 20) ausgelöst und soll dazu führen, dass der letzte Eintrag mit der gleichen Bezeichnung herausgesucht wird und der Inhalt der anderen Felder ausgelesen und in die Formularfelder übertragen wird. Die Abfrage in Zeile 23 klärt in einer Unterabfrage den letzten Eintrag mit der gleichen "Bezeichnung" ab und übergibt an die äußere Abfrage nur einen einzigen Wert: die ID.

Die Inhalte werden als Strings ausgelesen (Zeile 26 bis Zeile 30). Leere Felder werden so zu NULL in den Eingabefeldern. Eine Umwandlung der Inhalte in Datumswerte, Zeitwerte oder Zahlenwerte sollte vermieden werden, da dies nur zu Formatierungsproblemen und fehlerhaften Einträgen führt. Eine leere Uhrzeit gibt es nicht, ebenso wenig eine leere Zahl und ein leeres Datum. Hier würden dann Default-Werte genommen – bei den Zahlen eben die '0', bei Uhrzeiten wohl ebenfalls '00:00:00' und bei Zeiten dann das Startdatum 01.01.1900.

In den Zeilen 37 bis 51 werden die Inhalte, sofern sie nicht leer sind, in die entsprechenden Formularfelder mit den entsprechenden **`BoundField.updateString`** – Kommandos geschrieben.

GruppeMultiSpeichern

Aufruf aus	
Formular:	*Terminart_Termin, Termine_Kalenderübersicht, Termine_Monat_Terminart, Termine_Tabelleneingabe*

Benötigt	
Makro:	keins

```
001  SUB GruppeMultiSpeichern(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","Check7",
           "Check8","Check9")
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("Gruppe_multi"),inValue)
015  END SUB
```

In den Formularen befinden sich 9 Checkboxen, die nicht direkt mit einem Feld der zugrundeliegenden Tabelle verbunden sind. Stattdessen existiert in der Tabelle das Feld mit der Bezeichnung "Gruppe_multi" (Zeile 14). Die Checkboxen werden in einem Array zusammengefasst (7). Anschließend wird eine Schleife durch alle Checkboxen gestartet (9) und die Werte der Checkboxen, die ausgewählt wurden (**State = 1**, Zeile 10), zusammen addiert . Der Gesamtwert wird in das Datenfeld "Gruppe_multi" übertragen.

DialogStart

Aufruf aus	
Makro:	*Bericht_Start*

Benötigt	
Dialog:	*Dialog Datumseingabe*

```
001  DIM oDialogDat AS OBJECT
```

```
001  SUB DialogStart
002     DialogLibraries.LoadLibrary("Standard")
003     oDialogDat = createUnoDialog(DialogLibraries.Standard.Datumseingabe)
004     oDialogDat.Execute()
005  END SUB
```

Für die Berichte, die eine Halbjahresübersicht anfertigen sollen, ist eine Datumseingabe des Halbjahres erforderlich. Statt diese Datumseingabe über ein Input-Feld zu erledigen und nachher erst einmal auf die korrekte Eingabe überprüfen zu müssen wird hier ein Dialog genommen und mit dieser Prozedur gestartet.

Der Dialog wird vorher als Variable für das ganze Modul deklariert, da er ja auch in der Prozedur zur Beendigung des Dialogs benötigt wird.

DialogEnde

Aufruf aus	
Dialog:	*Dialog Datumseingabe*

Benötigt	
Makro:	*DialogStart*
Tabelle:	*Filter*

```
001 SUB DialogEnde
002    DIM stDatum AS STRING
003    oField =  oDialogDat.getControl("datDatum")
004    stDatum = oField.Date.Year & "-" & Right("0" & Trim(Str(oField.Date.Month)),2) &
          "-" & Right("0" & Trim(Str(oField.Date.Day)),2)
005    oSQL_Anweisung = oVerbindung.createStatement()
006    stSql = "UPDATE ""Filter"" SET ""Halbjahrende"" = '"+stDatum+"' WHERE ""ID"" =
          TRUE "
007    oSQL_Anweisung.executeUpdate(stSql)
008    oDialogDat.EndExecute()
009 END SUB
```

Der Dialog besteht nur aus einem Datumsfeld und einem Button. Wird der Dialog über den Button beendet, so wird diese Prozedur aufgerufen.

Aus dem Datumsfeld heraus wird der Wert direkt so formatiert, dass er über ein SQL-Kommando als Datum erkannt wird (Zeile 4). Anschließend wird der Wert in die Filtertabelle geschrieben und danach der Dialog beendet.

Export

Ical_Export

Aufruf aus	
Makro:	*Export_Start*

Benötigt	
Makro:	keins

```
001 SUB Ical_Export(stView AS STRING)
002    DIM oUcb AS OBJECT
003    DIM oOutputStream AS OBJECT
004    DIM oFile AS OBJECT
005    DIM oSQL_Anweisung AS OBJECT
006    DIM oAbfrageergebnis AS OBJECT
007    DIM stBaseUrl AS STRING
008    DIM stFileUrl AS STRING
009    DIM stString AS STRING
010    DIM stSql AS STRING
011    DIM stUID AS STRING
012    DIM stLocation AS STRING
013    DIM stSummary AS STRING
014    DIM stDescription AS STRING
015    DIM stDtStart AS STRING
016    DIM stDtEnd AS STRING
017    DIM stDtStamp AS STRING
018    DIM i AS INTEGER
019    DIM ar()
020    oUcb = createUnoService("com.sun.star.ucb.SimpleFileAccess")
021    oOutputStream = createUnoService("com.sun.star.io.TextOutputStream")
```

```
022     ar = split(ThisDatabaseDocument.URL, "/")
023     stBaseUrl = ""
024     FOR i = LBound(ar()) TO UBound(ar()) - 1
025        stBaseUrl = stBaseUrl & ar(i) & "/"
026     NEXT
027     stBaseFile = ar(i)
028     stFileUrl = stBaseUrl & "BaseTermine.ics"
029     IF oUcb.exists(stFileUrl) THEN
030        oUcb.kill(stFileUrl)
031     END IF
032     oFile = oUcb.OpenFileReadWrite(stFileUrl)
033     oOutputStream.SetOutputStream(oFile.getOutputStream)
034     stString = "BEGIN:VCALENDAR" & CHR(13) & CHR(10)
035     stString = stString & "VERSION:2.0" & CHR(13) & CHR(10)
036     stString = stString & "https://www.familiegrosskopf.de/robert/" & CHR(13) &
           CHR(10)
037     stString = stString & "METHOD:PUBLISH" & CHR(13) & CHR(10)
038     oOutputStream.writeString(stString)
039     oSQL_Anweisung = oVerbindung.createStatement()
040     stSql = "SELECT * FROM """+stView+""""
041     oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
042     WHILE oAbfrageergebnis.next
043         stUID = oAbfrageergebnis.getString(1)
044        stLocation = oAbfrageergebnis.getString(2)
045        stSummary = oAbfrageergebnis.getString(3)
046        stDescription = oAbfrageergebnis.getString(4)
047        stDtStart = oAbfrageergebnis.getString(5)
048        IF Len(stDtStart) = 8 THEN
049           stDtStart = ";VALUE=DATE:"&stDtStart
050        ELSE
051           stDtStart = ":"&stDtStart
052        END IF
053        stDtEnd = oAbfrageergebnis.getString(6)
054        IF Len(stDtEnd) = 8 THEN
055           stDtEnd = ";VALUE=DATE:"&stDtEnd
056        ELSE
057           stDtEnd = ":"&stDtEnd
058        END IF
059        stDtStamp = oAbfrageergebnis.getString(7)
060        stString = "BEGIN:VEVENT" & CHR(13) & CHR(10)
061        stString = stString & "UID:" & stUID & stBaseFile & CHR(13) & CHR(10)
062        stString = stString & "LOCATION:" & stLocation & CHR(13) & CHR(10)
063        stString = stString & "SUMMARY:" & stSummary & CHR(13) & CHR(10)
064        stString = stString & "DESCRIPTION:" & stDescription & CHR(13) & CHR(10)
065        stString = stString & "CLASS:PUBLIC" & CHR(13) & CHR(10)
066        stString = stString & "DTSTART"  & stDtStart & CHR(13) & CHR(10)
067        stString = stString & "DTEND" & stDtEnd & CHR(13) & CHR(10)
068        stString = stString & "DTSTAMP:" & stDtStamp & CHR(13) & CHR(10)
069        stString = stString & "END:VEVENT" & CHR(13) & CHR(10)
070        oOutputStream.writeString(stString)
071     WEND
072     stString = "END:VCALENDAR"
073     oOutputStream.writeString(stString)
074     oOutputStream.closeOutput()
075  END SUB
```

In Zeile 22 bis Zeile 25 wird die BasisURL der aktuellen Base-Datei ermittelt. Der Dateiname wird separat in der Variablen stBaseFile gespeichert (27). Anschließend wird der Pfad und Name für die neu anzulegende Datei festgelegt. Existiert diese Datei bereits, so wird sie in Zeile 30 erst einmal gelöscht. Dies ist notwendig, da beim Schreiben eines Strings in die Datei neu am Anfang begonnen wird. Bestehende Zeilen werden überschrieben, doch wenn der neue Inhalt weniger Zeilen einnimmt als der alte Inhalt, dann enthält die Datei anschließend neben dem neuen Inhalt auch noch alten Inhalt.

Von Zeile 35 bis Zeile 37 werden die Kopfzeilen für die iCalendar-Datei zusammengestellt. Jede Zeile endet dabei mit einem Zeilenumbruch, der sowohl für Linux als auch für Windows gilt. Diese Kopfzeilen werden in Zeile 38 dann in die neue Datei geschrieben.

In Zeile 40 und Zeile 41 werden die einzufügende Inhalte aus der vorgewählten Ansicht ausgelesen. Die Inhalte werden Zeile für Zeile abgearbeitet und in das Schema für die iCalendar-Datei übertragen. Ist in den Zeitstempelfeldern nur ein Eintrag, der eine Länge von 8 Zeichen hat, dann handelt es sich um einen Eintrag, der nur aus einem Datum ohne Zeit besteht. Dies muss entsprechend in der iCalendar-Datei vermerkt werden. Deshalb werden diese Variablen in den Zeilen 48 bis 58 entsprechend ergänzt. Dadurch werden Einträge als Einträge für den ganzen Tag gesehen und erscheinen nicht in der Terminübersicht, die sich auf Termine mit Startzeit und Endzeit bezieht.

Besondere Aufmerksamkeit erfordert die korrekte Zeitangabe. Hier gibt es unterschiedliche Herangehensweisen, die davon geprägt sind, dass zwei Personen, die einen Termin zu einem Zeitpunkt ausmachen, sich auch zu dieser zeit tatsächlich treffen können, wenn so ein Kalenderausschnitt der anderen Person zugeschickt wird. Innerhalb eines Ortes ist das kein Problem, so dass in dieser Prozedur auch auf alle Angaben zu Zeitzonen verzichtet wird (sogenannte floatende Zeitangabe). Sobald die Personen aber in unterschiedlichen Zeitzonen arbeiten muss der Termin je nach Zeitzone unterschiedlich ausgelesen werden. Bei dem Makro führten am Anfang Einträge wie **`DTSTART;TZID=Europe/Berlin:`** oder ein **`Z`** am Ende des Zeitstempels dazu, dass die korrekt aus der Datenbank ausgelesenen Werte hier in unter Berücksichtigung der Sommerzeit um 2 Stunden versetzt in den Kalender eingelesen wurden.

Nachdem alle Datensätze ausgelesen wurden wird in Zeile 72 der Endeintrag vorgenommen und anschließend als letzter Eintrag in die Datei übertragen. Anschließend wird die Dateiausgabe geschlossen.

Die erstelle iCalendar-Datei kann jetzt in andere Kalenderprogramme übertragen werden. Sie sollte dazu aber vorher aus dem Verzeichnis der Base-Datei in ein anderes Verzeichnis kopiert werde, da sonst die Ausgabe neuer Daten in eine Datei mit dem gleichen Namen nicht möglich ist. Die Kalenderdatei blockiert dann den Schreibvorgang vom Makro aus.

Export_Start

Aufruf aus	
Formular:	*Terminart_Termin, Termine_Kalenderübersicht, Termine_Monat_Terminart, Termine_Tabelleneingabe*

Benötigt	
Makro:	*Ical_Export*
Ansicht:	*Ansicht_Kalender_Export, Ansicht_Kalender_ohne_Ferien_Export*

```
001 SUB Export_Start(oEvent AS OBJECT)
002    DIM stView AS STRING
003    oForm = oEvent.Source.Model.Parent
004    oForm.updateRow
005     IF oForm.getByName("chkFerien").State = 1 THEN
006        stView = "Ansicht_Kalender_Export"
007     ELSE
008        stView = "Ansicht_Kalender_ohne_Ferien_Export"
009    END IF
010    Ical_Export(stView)
011 END SUB
```

Es kann zwischen den beiden Exportmöglichkeiten mit und ohne Ferien gewählt werden. Der Unterschied besteht lediglich darin, welche Ansicht (**`View`**) als Datengrundlage genutzt wird.

Import

Dieses Modul sollte mit entsprechender Vorsicht gehandhabt werden. Die Zusammenstellung der Prozeduren beruht nur auf Tests mit einem einfachen Export eines Termins aus einer Kalendersoftware, der über Mail versendet wurde.

Mit den Prozeduren in diesem Modul ließ sich der Termin einwandfrei importieren. Ein doppelter Import wurde ausgeschlossen, da vorher nach gleichlautender Terminbezeichnung zusammen mit gleichlautendem Datum gesucht wird.

Ical_Import

Aufruf aus	
Makro:	*Import_Start*

Benötigt	
Makro:	*Ical_Timestamp, LetzterSonntag*
Tabelle:	*Termine, Filter*

```
001 FUNCTION Ical_Import(stFileUrl AS STRING) AS INTEGER
002    DIM oSQLAnweisung AS OBJECT
003    DIM oAbfrageergebnis AS OBJECT
004    DIM stSql AS STRING
005    DIM stRow AS STRING
006    DIM stBez AS STRING
007    DIM stBezeichnung AS STRING
008    DIM stBemerkung AS STRING
009    DIM stOrt AS STRING
010    DIM stDat AS STRING
011    DIM stStartDat AS STRING
012    DIM stStartZeit AS STRING
013    DIM stEndDat AS STRING
014    DIM stEndZeit AS STRING
015    DIM inBegin AS INTEGER
016    DIM inCount AS INTEGER
017    DIM daStart AS DATE
018    DIM daEnd AS DATE
019    DIM fn AS INTEGER
020    DIM ink AS INTEGER
021    DIM loID AS LONG
022    fn = freeFile
023    OPEN stFileUrl FOR INPUT AS #fn
024    DO WHILE NOT eof(#fn)
025       LINE INPUT #fn, stRow
026       IF stRow = "BEGIN:VEVENT" THEN
027          stBemerkung = "NULL,"
028          stOrt = "NULL,"
029          stEndDat = "NULL,"
030          stEndZeit = "NULL"
031          inBegin = 1
032       END IF
033       IF stRow = "END:VEVENT" THEN
034          oSQL_Anweisung = oVerbindung.createStatement()
035          stSql = "SELECT COUNT(""ID"") FROM ""Termine"" WHERE ""Bezeichnung"" =
                '"+stBez+"' AND ""Beginn_Datum"" = '"+stDat+"'"
036          oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
037          WHILE oAbfrageergebnis.next
038             inCount = oAbfrageergebnis.getInt(1)
039          WEND
040          IF inCount = 0 THEN
041             stSql = "INSERT INTO ""Termine"" (""zieID"",""Bezeichnung"",
                   ""Bemerkung"",""Ort"",""Beginn_Datum"",""Beginn_Uhrzeit"",
```

```
                  ""Ende_Datum"",""Ende_Uhrzeit"") VALUES ('0',"+stBezeichnung+
                  stBemerkung+stOrt+stStartDat+stStartZeit+stEndDat+stEndZeit+") "
042               oSQL_Anweisung.executeUpdate(stSql)
043               stSql = "CALL IDENTITY()"
044               oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
045               WHILE oAbfrageergebnis.next
046                  loID = oAbfrageergebnis.getLong(1)
047               WEND
048               IF ink = 0 THEN
049                  stSql = "UPDATE ""Filter"" SET ""StartIDImport"" = '"+loID+"',
                        ""EndIDImport"" = '"+loID+"' WHERE ""ID"" = TRUE"
050               ELSE
051                  stSql = "UPDATE ""Filter"" SET  ""EndIDImport"" = '"+loID+"' WHERE
                        ""ID"" = TRUE"
052               END IF
053               oSQL_Anweisung.executeUpdate(stSql)
054               ink = ink + 1
055            END IF
056            inBegin = 0
057         END IF
058         IF inBegin = 1 THEN
059            IF InStr(stRow,"SUMMARY") THEN
060               stBez = Mid(stRow, inStr(stRow,":")+1)
061               stBezeichnung = "'" & stBez & "',"
062            ELSEIF InStr(stRow,"DESCRIPTION:") THEN
063               stBemerkung = "'" & Mid(stRow, inStr(stRow,":")+1) & "',"
064            ELSEIF InStr(stRow,"LOCATION:") THEN
065               stOrt = "'" & Mid(stRow, inStr(stRow,":")+1) & "',"
066            ELSEIF InStr(stRow,"DTSTART") THEN
067               daStart = Ical_Timestamp(Mid(stRow, inStr(stRow,":")+1))
068               IF InStr(stRow,"UTC") OR Right(stRow,1) = "Z" THEN
069                  daStart = DateAdd("h",1,daStart)
070                  IF daStart > LetzterSonntag(DateValue(Year(daStart) & "-04-01")) AND
                        daStart < LetzterSonntag(DateValue(Year(daStart) & "-11-01"))
                        THEN
071                     daStart = DateAdd("h",1,daStart)
072                  END IF
073               END IF
074               stDat = Year(daStart) & "-" & Right("0" & Month(daStart),2) & "-" &
                     Right("0" & Day(daStart),2)
075               stStartDat = "'" & stDat & "',"
076               IF InStr(Mid(stRow, inStr(stRow,":")+1),"T") THEN
077                  stStartZeit = "'" & Right("0" & Hour(daStart),2) & ":" &
                        Right("0" & Minute(daStart),2) & ":" &
                        Right("0" & Second(daStart),2) & "',"
078               ELSE
079                  stStartZeit = "NULL,"
080               END IF
081            ELSEIF InStr(stRow,"DTEND") THEN
082               daEnd = Ical_Timestamp(Mid(stRow, inStr(stRow,":")+1))
083               IF InStr(stRow,"UTC") OR Right(stRow,1) = "Z" THEN
084                  daEnd = DateAdd("h",1,daEnd)
085                  IF daEnd > LetzterSonntag(DateValue(Year(daEnd) & "-04-01")) AND
                        daEnd < LetzterSonntag(DateValue(Year(daEnd) & "-11-01")) THEN
086                     daEnd = DateAdd("h",1,daEnd)
087                  END IF
088               END IF
089               stEndDat = "'" & Year(daEnd) & "-" & Right("0" & Month(daEnd),2)
                     & "-" & Right("0" & Day(daEnd),2) & "',"
090               IF InStr(Mid(stRow, inStr(stRow,":")+1),"T") THEN
091                  stEndZeit = "'" & Right("0" & Hour(daEnd),2) & ":" &
                        Right("0" & Minute(daEnd),2) & ":" &
                        Right("0" & Second(daEnd),2) & "'"
092               ELSE
093                  stEndZeit = "NULL"
094               END IF
```

```
095           ELSE
096           END IF
097        END IF
098     LOOP
099     CLOSE #fn
100     Ical_Import = ink
101  END FUNCTION
```

In Zeile 22 wird die nächste freie Datenkanalnummer nachgeschlagen und direkt danach genutzt. Die Datei wird für den INPUT geöffnet. Etwas verwirrend, aber es ist der Input in das ausführende Programm gemeint - also Lesezugriff. Solange der Inhalt der Datei nicht zu Ende ist wird der Inhalt Zeilenweise ausgelesen. Dies findet in der Schleife von Zeile 24 bis Zeile 98 statt.

Jeder Termin beginnt mit BEGIN:VEVENT. Das ist sozusagen der Start, jetzt darauf zu achten, wann die anderen Schlüsselbegriffe erscheinen (Zeile 26). Mit dem ersten Auftauchen des Schlüsselbegriffs werden noch 2 Variablen gesetzt. Zum einen die Variablen stBemerkung, stOrt, stStartZeit, stEndDat und stEndZeit, die für die Datenbank vordefiniert wird als NULL. Dieses Feld kann eventuell beim Export aus einem Kalenderprogramm fehlen. Und dann noch die Variable inBegin, die mit dem Wert '1' ermöglichen soll, dass mit dieser Zeile die Verzweigung ab Zeile 58 beschritten werden kann.

Ab Zeile 54 wird nach den Schlüsselbegriffen SUMMARY, DESCRITION, LOCATION, DTSTART und DTEND geschaut. Taucht einer dieser Begriffe auf, so wird die entsprechende Verzweigung betreten.

SUMMARY (Zeile 59): Hier wird die Bezeichnung des Termins ausgelesen. Daraus werden 2 Variablen gebildet. StBez ohne irgendeine zusätzliche Formatierung, damit nach der Bezeichnung in den bestehenden Daten gesucht werden kann. StBezeichnung wird gleich so formatiert, dass sie für den SQL-Code des INSERT geeignet ist. SUMMARY kann auch noch Hinweise auf die Sprache enthalten, so dass der Doppelpunkt hier nicht in den Suchbegriff aufgenommen wurde.

DESCRIPTION (Zeile 61) und LOCATION (Zeile 64): Die Bemerkung und der Ort sind so vorformatiert, dass sie direkt in den INSERT übernommen werden können. Dabei wird davon ausgegangen, dass immer eine Ortsangabe aus der Datenquelle vorliegt.

Komplizierter wird es mit der Umwandlung von Zeitstempeln aus iCalendar zu einer Trennung von Datum und Zeit, die zudem noch eventuelle Abweichungen zu der Sommerzeit berücksichtigt. Der Zeitstempel wird hierfür durch die Funktion Ical_Timestamp in eine in Basic verarbeitbare Datumsvariable mit Zeitanteil umgewandelt (Zeile 67). Enthält DTSTAT zusätzlich einen Vermerk wie 'UTC' oder ein 'Z' am Ende des Zeitstempels, so muss die Zeit auf die in dieser Datenbank gebräuchlichen Normalzeit umgewandelt werden Zeile 68 bis 73). Diese Umwandlung wird hier nur für den deutschsprachigen Raum nach Zeitzone Berlin gemacht (1 Stunde addieren bei Normalzeit (Zeile 69), 2 Stunden addieren bei Sommerzeit (Zeile 71 eine zusätzlich Stunde). Als Basis für die Unterschiede wird der letzte Sonntag vor dem 1.4 und der letzte Sonntag vor dem 1.11. herausgesucht. An diesen Sonntagen erfolgt momentan der Wechsel von Winterzeit zu Sommerzeit und umgekehrt.

Das Datum aus DTSTRAT wird in Zeile 74 zu einem in SQL gebräuchlichen Datumsstring umgewandelt. Dabei wird auf die zweistellige Monatszahl und die zweistellige Tageszahl Rücksicht genommen. In Zeile 75 wird nur der reine SQL-String erstellt (für die Abfrage nach Duplikaten), in Zeile 76 dann der String, der direkt im INSERT weiter verwendet werden kann. Enthält die Zeile innerhalb des Zeitstempels, also nicht in dem führenden DTSTART, ein 'T', dann liegt auch eine Zeitangabe vor. Diese Zeitangabe wird dann mit zweistelliger Stunden-, Minuten- und Sekundenangabe erstellt (Zeile 77). Ist kein 'T' vorhanden, so wird der Wert hier auf für den SQL-String auf NULL gesetzt.

Die Umwandlung von DTEND läuft identisch zu der von DTSTART ab. Einzige Ausnahme ist hier der SQL-Code für den INSERT, der für das letzte Element nicht mit einem Komma endet (Zeile 91 bzw. Zeile 93).

Taucht END:VEVENT auf, dann wird es zeit zum Abspeichern des Datensatzes (Zeile 33 bis Zeile 57). Von Zeile 35 bis Zeile 39 wird überprüft, ob ein Datensatz mit gleicher Bezeichnung und

gleichem Datum vorkommt. So ein Datensatz soll erst einmal nicht importiert werden. Erst wenn die Zählung hier '0' ergibt geht es weiter zum Abspeichern.

In Zeile 41 wird der durch die Variablen bereits gut vorbereitete Code zum Einfügen der Daten erstellt und abgeschickt. Zeile 43 ermittelt die durch den INSERT erstellten Primärschlüsselwert. Dieser wird in die Felder StartIDImport und EndIDImport übertragen (Zeile 49), wenn es der erste Datensatz dieses Importes ist. Beim zweiten Datensatz ist **ink=1** (Zeile 54) und damit wird nur EndIDImport auf den neuen Wert gesetzt (Zeile 51).

Der Wert für inBegin wird jetzt auf 0 gesetzt, damit die ganze folgende Untergliederung nicht noch durchsucht wird.

Mit Zeile 98 startet der Import für den nächsten Datensatz. Ist der Import zu Ende, dann wird in Zeile 99 der Zugriff auf die Datei wieder freigegeben und in Zeile 100 mit der Funktion Ical_Import die Anzahl der importierten Datensätze weitergegeben.

Ical_Timestamp

Aufruf aus
Makro: *Ical_Import*

Benötigt
Makro: keins

```
001  FUNCTION IcalTimestamp(d AS STRING) AS DATE
002     DIM daT AS DATE
003     IF InStr(d,"T") THEN
004        daT = DateSerial(Left(d,4), Mid(d,5,2), Mid(d,7,2)) + TimeSerial(Mid(d,10,2),
              Mid(d,12,2), Mid(d,14,2))
005     ELSE
006        daT = DateSerial(Left(d,4), Mid(d,5,2), Mid(d,7,2))
007     END IF
008     IcalTimestamp = daT
009  END FUNCTION
```

Der Timestamp in ICalendar ist nach dem Prinzip Jahreszahl, Monatszahl, Tageszahl, Trenner 'T', Stunden, Minuten und Sekunden angelegt. Es kann auch vorkommen, dass keine Zeit mit angegeben ist, dann entfällt der Inhalt ab 'T'. Da der Zeitstempel angepasst werden muss, wenn die Zeitzone berücksichtigt wird, muss der Timestamp in einen internen Zeitstempel umgewandelt werden.

In der Bedingung Zeile 3 wird nach diesem 'T' gefragt. Ist es vorhanden, so wird ein Zeitstempel aus Datum und Zeit angefertigt, mit dem weiter gerechnet werden kann. Während das Jahr noch über **Left** ausgelesen wird kommt bei den anderen Teilen des Strings die Funktion **Mid** zum Einsatz. Mit ihr lassen sich genau bemessene Teilstrings auslesen, mit **Mid(d,7,2)** sind dies die Zeichen ab dem 7. von der Variablen d und von dort aus 2 Zeichen. Aus dem Timestamp '20200423T154809' wird so '23' für den Tag ausgelesen.

Enthält der Timestamp aus Icalendar kein 'T', so wird nur ein Datum ausgelesen (Zeile 6).

Der in Basic verwertbare Zeitstempel wird dann an die Funktion als Rückgabewert übergeben (Zeile 8).

Import_Start

Aufruf aus
Formular: *Terminart_Termin, Termine_Kalenderübersicht, Termine_Monat_Terminart, Termine_Tabelleneingabe*

Benötigt
Makro: *Ical_Import*

```
001 SUB Import_Start(oEvent AS OBJECT)
002    DIM oFeld AS OBJECT
003    DIM oFeld1 AS OBJECT
004    DIM stFileUrl AS STRING
005    DIM inImport AS INTEGER
006    oFeld = oEvent.Source.Model
007    oFeld1 = oFeld.parent.getByName("Dateiauswahl")
008    IF oFeld1.Text <> "" THEN
009       stFileUrl = ConvertToUrl(oFeld1.Text)
010        inImport = Ical_Import(stFileUrl)
011    END IF
012    IF inImport > 0 THEN
013       oForm = oFeld.Parent.Parent.getByName("MainForm")
014       oForm.filter ="""ID"" BETWEEN (SELECT ""StartIDImport"" FROM ""Filter"" WHERE
             ""ID"" = True) AND (SELECT ""EndIDImport"" FROM ""Filter"" WHERE ""ID"" =
             True) "
015       oForm.ApplyFilter = TRUE
016       oForm.reload()
017       msgbox "Die importierten Datensätze werden im Formular angezeigt."
018    ELSE
019       msgbox "Es wurden keine Datensätze importiert."
020    END IF
021 END SUB
```

Es wird nicht direkt nach der Auswahl der Datei importiert, da dies bei eine eventuell falschen Auswahl dann sofort zu importieren Daten führt. Stattdessen wird über einen gesonderten Button der Import angestoßen.Das Feld für die Dateiauswahl wird über die Lage des auslösenden Buttons im Formular bestimmt (Zeile 6 und 7). Danach wird aus der Dateiauswahl der Pfad ausgelesen und in eine verwertbare URL umgewandelt. Diese URL wird an die Funktion Ical_Import weitergegeben Zeile 10). Diese Funktion gibt mit inImport zurück, wie viele Datensätze importiert wurden.

Wenn überhaupt ein Datensatz importiert wurde, so soll das Ergebnis des Imports vom Formular aus angezeigt werden. Dies wird dadurch erreicht, dass ein Formularfilter gesetzt wird (Zeile 14 und 15). Der Formularfilter liest dabei die in der Tabelle "Filter" abgespeicherten Werte für die erste und die letzte ID der anzuzeigenden Werte aus. Diese Werte wurden in der Funktion Ical_Import dort abgespeichert.

Sind Datensätze importiert worden, so kommt die Meldung, dass die importierten Datensätze jetzt angezeigt werden (Zeile 17). Worden keine importiert, so wird auch dies vermeldet (Zeile 19).

LetzterSonntag

Aufruf aus
Makro: *Ical_Import*

Benötigt
Makro: keins

```
001 FUNCTION LetzterSonntagVormonat(d AS DATE)
002    d = DateAdd("d", -1, d)
003    DO WHILE DatePart("w",d) <> 1
004       d = DateAdd("d", -1, d)
005    LOOP
006    d = DateValue(d) + TimeValue("02:00:00")
007    LetzterSonntag = d
008 END FUNCTION
```

Für den Wechsel von Winterzeit zu Sommerzeit und umgekehrt ist es notwendig, den letzten Sonntag im März und im Oktober zu ermitteln. Diese Funktion übernimmt einfach das Datum, was ihr mitgegeben wurde, zählt davon einen Tag herunter (Zeile 2), damit nicht bei der Weitergabe eines Sonntags als Datumswert das gleiche Datum zurückgegeben wird, und zählt von dort aus so lange je einen Tag herunter (Zeile 4), bis es wieder bei dem Wochentag '1' angelangt ist (Zeile 3). Das Datum wird dann mit dem Zeitwert von 2 Uhr ergänzt, da dies die Grenze für den Wechsel von Sommerzeit zu Winterzeit und umgekehrt darstellt. Das Datum wird an die aufrufende Prozedur zurückgegeben.

Filter

Aufruf aus
Formular: *Termine_Tabelleneingabe*

Benötigt
Makro: keins

```
001  SUB Filter_Abgleich(oEvent AS OBJECT)
002     DIM oFeld AS OBJECT
003     DIM oForm1 AS OBJECT
004     DIM oForm2 AS OBJECT
005     DIM stTag AS String
006     DIM atTag()
007     oFeld = oEvent.Source.Model
008     stTag = oFeld.Tag
009     oForm1 = oFeld.parent
010     oFeld.commit()
011     oForm1.updateRow()
012     oForm2 = oForm1.parent.getByName(aTag(0))
013     IF UBound(aTag) > 0 THEN
014        oForm2.filter = aTag(1)
015        oForm2.ApplyFilter = TRUE
016     END IF
017     oForm2.reload()
018  END SUB
```

In den Zusatzinformationen (8) des Listenfeldes für die Monatsvorgabe ist enthalten, wie das zweite Formular heißen soll, das nach Betätigung des Listenfeldes aktualisiert werden soll. Außerdem steht doch, durch einen '→' getrennt, gegebenenfalls eine Filterinformation für das zu aktualisierende Formular. Alle anderen Elemente, die angesprochen werden sollen, werden durch das auslösende Ereignis ermittel.

Das Listenfeld gibt in Zeile 10 den Wert an das Formular weiter, in dem es sich befindet. Anschließend wird das Formular abgespeichert und schließlich das Formular, das dadurch gefiltert werden soll, neu eingelesen. Dabei wird in Zeile 14 und 15 gegebenenfalls der Filter wieder zugewiesen, falls er durch andere Aktionen, z.B. den Import von Daten, entfernt wurde.

Wartung

Tabellenindex_runter

Aufruf aus
Makro: *Tabellenstart*

Benötigt
Makro: keins

```
001  Sub Tabellenindex_runter(stTabelle AS STRING)
002     DIM stAnzahl AS INTEGER
003     DIM inAnzahl AS INTEGER
004     DIM inSequence_Value AS INTEGER
005     oSQL_Anweisung = oVerbindung.createStatement()
006     stSql = "SELECT MAX(""ID"") FROM """+stTabelle+""""
007     oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
008     WHILE oAbfrageergebnis.next
009        stAnzahl = oAbfrageergebnis.getString(1)
010     WEND
011     IF stAnzahl = "" THEN
012        inAnzahl = -1
013     ELSE
014        inAnzahl = cInt(stAnzahl)
015     END IF
016     inSequence_Value = inAnzahl+1
017     oSQL_Anweisung1 = oVerbindung.createStatement()
018     oSQL_Anweisung1.executeQuery("ALTER TABLE """+tablename+""" ALTER COLUMN ""ID""
           RESTART WITH "+inSequence_Value+"")
019  End Sub
```

Mit dieser Prozedur wird das automatisch hoch geschriebene Primärschlüsselfeld mit der vorgegebenen Bezeichnung "ID" auf den niedrigst möglichen Wert eingestellt.

In Zeile 6 und Zeile 7 wird der höchste eingegebenen Wert für das Feld "ID" ermittelt. Der Wert wird als Text in Zeile 9 ausgelesen. Die Textform ist hier notwendig, weil sonst selbst bei einer leeren Tabelle nicht ein leerer Wert ausgegeben wird sondern die Integer-Zahl 0.

Ist das Ergebnis ein leerer Text (11), dann wird die Integer-Wert 1 gesetzt. Ansonsten wird der Integer-Wert angenommen, der als höchster Wert in der Tabelle steht (14). Zu diesem Wert wird jetzt 1 addiert (16) und dieser Wert als nächster einzufügende Wert für die Sequenz in die Tabellenbeschreibung eingefügt (18).

Tabellenstart

Aufruf aus	
Makro:	keins, wird direkt aufgerufen, wenn Daten zurückgesetzt wurden.

Benötigt	
Makro:	*Tabellenindex_runter*
Tabelle:	*Termine, Terminart, Zielgruppe, Ferien*

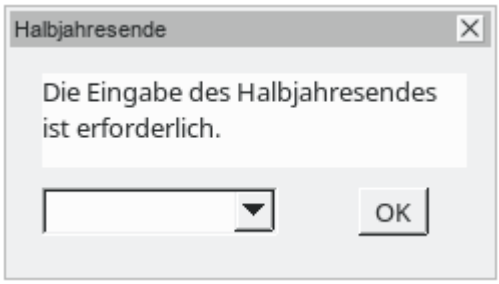
```
001  Sub Tabellenstart
002     Tabellenindex_runter("Termine")
003     Tabellenindex_runter("Terminart")
004     Tabellenindex_runter("Zielgruppe")
005     Tabellenindex_runter("Ferien")
006  End Sub
```

Die Prozedur gibt lediglich alle benannten Tabellen nacheinander an die Prozedur "Tabellenindex_runter" weiter. Dort wird dann die jeweilige Tabelle so eingestellt, dass der Auto-Wert mit der nächstfolgenden Zahl weiter zählt.

Dialog Datumseingabe

Aufruf aus	
Makro:	*DialogStart*

Benötigt
Makro: *DialogEnde*



Dieser Dialog ist notwendig, damit noch nach dem Aufruf eines Berichtes über das Formular die Eingabe eines Halbjahresdatums erledigt werden kann. Natürlich könnte das auch über ein Feld im Formular erfolgen. Nur ist die Eingabe für lediglich zwei der verschiedenen Berichte notwendig.

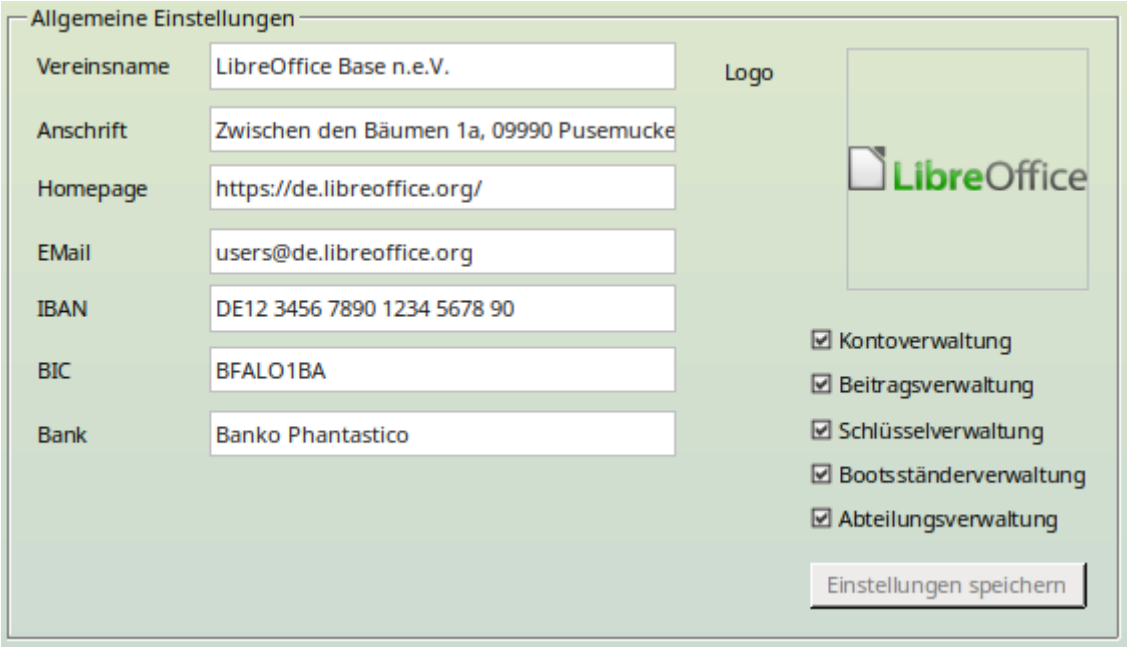
Alternativ dazu wäre auch über den Befehl **`InputBox`** eine Eingabe möglich gewesen. Da hätte dann aber anschließend wieder abgesichert werden müssen, dass die Eingabe wirklich einem Datum entspricht. Hier ist das Datumsfeld in einem Dialog, noch dazu mit übersichtlich aufklappbarer Monatsübersicht, klar im Vorteil.

Vereinsverwaltung

Einführung

Vereine benötigen eine Mitgliedsverwaltung, eine Übersicht über die aktuellen Vorstände und darüber hinaus oft noch verschiedene Module wie z.B. eine Kontoverwaltung, eine Beitragsverwaltung usw.

Die Datenbank «LO Verein»[10] stellt eine überarbeitete Fassung einer älteren Vereinssoftware dar, die für einen Kanusportverein entwickelt wurde. Neben der Mitgliedsverwaltung sind dort auch Beitragsverwaltungen, Abteilungsverwaltungen und die Verwaltung der Liegeplätze von Booten zu finden.



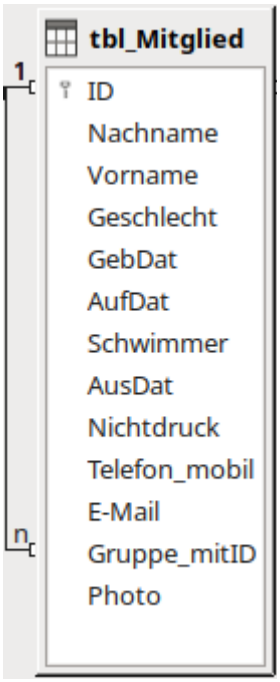
Der besondere Zusatz der aktuellen Version: Die verschiedenen Anwendungsgebiete sind einzeln in einer Einstellungstabelle zuschaltbar. Nicht jeder Verein braucht eine Liegeplatzverwaltung für Boote. Die Beitragsverwaltung wird manchmal auch lieber über eine Buchungssoftware der Banken erledigt. Da stören dann diese Module nur, wenn sie als Eintragungspunkte in den Formularen auftauchen.

Tabellen

In der Datenbank sind grundsätzlich die Tabellen enthalten, die für alle Module erforderlich sind. So befinden sich dort eben unabhängig davon, ob sie gebraucht werden, z. B. Tabellen zur Verwaltung von Bootsliegeplätzen («Bootsständerverwaltung»). Diese Tabellen sollten, auch wenn sie nicht benötigt werden, nicht gelöscht werden. Sie könnten stattdessen über **Extras → Tabellenfilter** unsichtbar geschaltet werden.

10 Beispieldatenbank Beispiel_LO_Verein202211.odb

Zentrale Mitgliedstabelle



In der "tbl_Mitglied" werden die Daten verwaltet, die sich auf die einzelne Person beziehen. Gibt es mehrere Personen, die z. B. eine gemeinsame Adresse haben, so wird in dem Feld "Gruppe_mitID" die Primärschlüsselnummer der Person gespeichert, für die zuerst diese Adresse eingegeben wurde. Deswegen in dem obigen Bild die Verbindung des Primärschlüssels "ID" mit dem Fremdschlüssel "Gruppe_mitID" innerhalb der Tabelle.

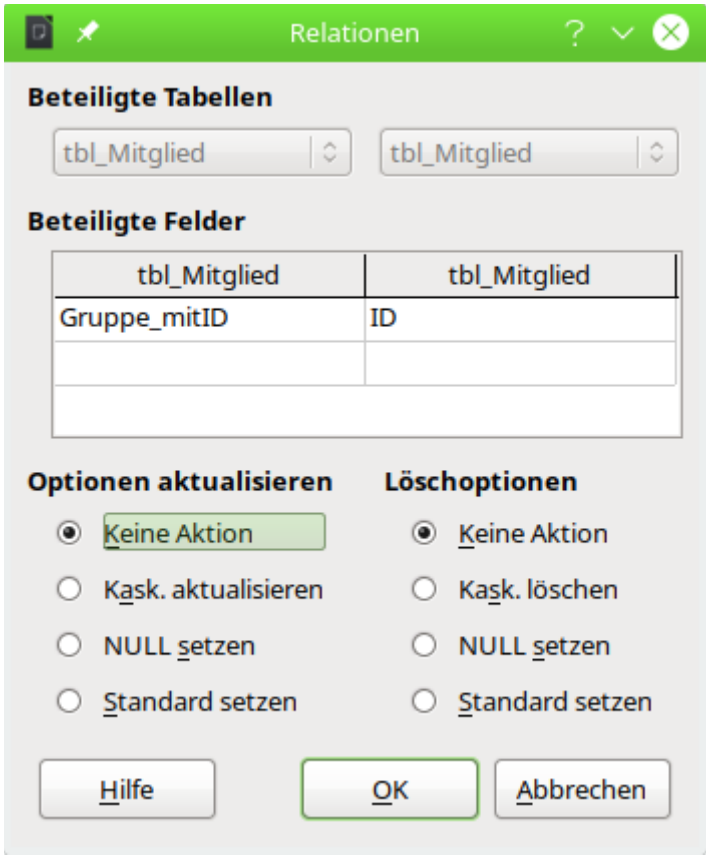


Abbildung 1: Die Beziehung ist so geregelt, dass weder eine Aktualisierung noch eine Löschung eines Mitglieds möglich ist, das mit anderen Mitgliedern über "Gruppe_mitID" verbunden ist.

	ID	Nachname	Vorname	Geschlecht	GebDat	AufDat	Schwimmer
	3	Müller	Karl	m	27.02.64	04.04.21	☑
	4	Müller-Mehl	Maria	w	13.11.69	01.01.22	☑
	5	Müller	Isabel	w	17.07.07	01.01.22	☑
	6	Müller	Janis	m	23.10.09	01.01.22	☑
	7	Schulze	Jolantha	w	17.09.01	03.02.22	☑
	8	Querulant	Gisbert	m	13.11.03	07.03.22	☐

AusDat	Nichtdruck	Telefon_mobil	E-Mail	Gruppe_mitID	Photo
	☐	0123456789			DB_Bilder/Gr
01.09.22	☐			3	
	☐			3	
	☐	0987654321	flitzepiepe	3	
	☐			3	
	☐		mailq@ex		

Datenziel	
Formular:	*Einzelmessung_System, frm_Berichte, frm_Boot, frm_Mitglied, frm_Schluessel, frm_Vorstand*
Makro:	*SelectGroup, NewBoatAllowed, Group*
Abfrage:	*Abfrage_Start, qry_Beitraege, qry_Bootsstaender, qry_Filter_Mitglied, qry_Jubilaeum, qry_Mitglied_Konto, qry_Mitgliedsdauer, qry_Schluessel_Uebersicht*
Ansicht:	*viw_Adresse_Num_Namen, viw_Beitrag, viw_Filter_Mitglieder_komplett_AusEin, viw_Gruppe, viw_Mitglieder_komplett, viw_Statistik_Alter_Detail*
Bericht:	keine direkt

Feldname	Feldtyp	Beschreibung
ID	Integer	Primärschlüssel der Tabelle. Der Primärschlüssel muss eindeutig sein. Das Feld ist als Auto-Wert-Feld gesetzt.
Nachname	Text	Nachname des Mitglieds Eingabe erforderlich: Ja
Vorname	Text	Vorname des Mitglieds
Geschlecht	Text	Kürzel ('d', 'm', 'w'), maximal ein Zeichen
GebDat	Datum	Geburtsdatum
AufDat	Datum	Aufnahmedatum
Schwimmer	Ja/Nein	Schwimmausweis liegt vor – wichtig für einen wassersporttreibenden Verein
AusDat	Datum	Austrittsdatum – Mitglieder werden beim Austritt nicht (direkt) gelöscht.
Nichtdruck	Ja/Nein	Bei Rundschreiben kein gedrucktes Schreiben an diese Person. Stimmt die Adresse nicht, so ist die Post unzustellbar. Das Porto kann hier gespart werden.
Telefon_mobil	Text	Mobiltelefonnummer wird direkt ans Mitglied gekoppelt
E-Mail	Text	E-Mail-Adresse z. B. Für Rundschreiben per Mail.
Gruppe_mitID	Integer	Zuweisung zu einem anderen Mitglied, bei dem die Adresse und die Kontoverbindung gespeichert wurde. Fremdschlüssel auf "tbl_Mitglied"."ID"
Photo	Text	Bilder werden in einem gesonderten Verzeichnis gespeichert. In der Datenbank selbst liegt nur der Pfad dazu. Im Formular kann das Bild so trotzdem angezeigt werden.

Zur Absicherung des Austrittsdatums wird unter **Extras → SQL** eingegeben:

```
001 ALTER TABLE "tbl_Mitglied"
002 ADD CONSTRAINT "Austrittsdatum >= Aufnahmedatum"
003 CHECK ( COALESCE ("AusDat", "AufDat")>="AufDat" )
```

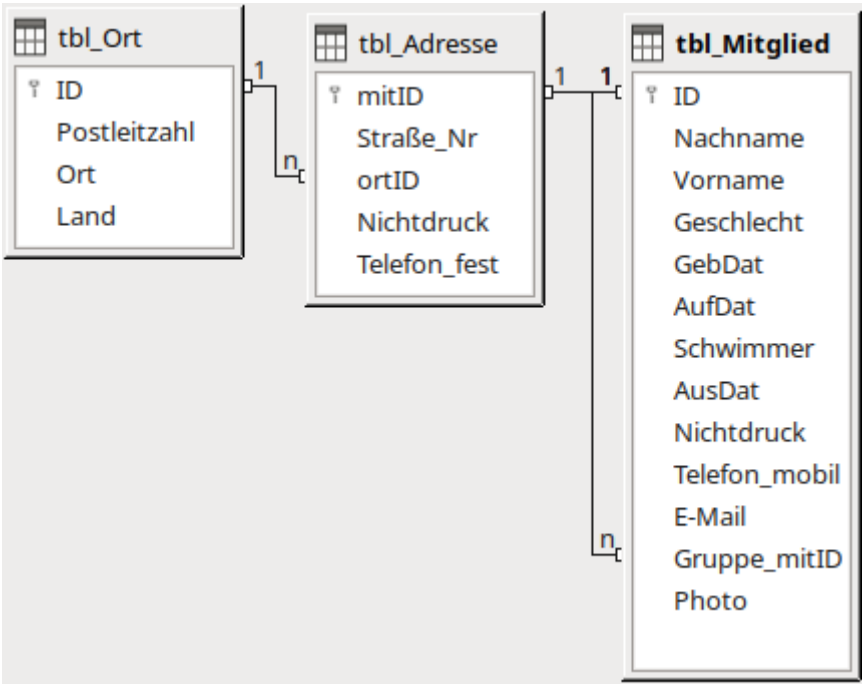
Hierdurch wird das Feld "AusDat" so festgelegt, dass es auf jeden Fall nicht vor dem Feld "AufDat" liegt. Ein Austritt vor dem Beitritt ist nicht möglich. Bei Fehlversuchen wird ausgegeben: «Check constraint violation Austrittsdatum => Aufnahmedatum table: tbl_Mitglied…»

Zur Absicherung des Geburtsdatums wird unter **Extras → SQL** eingegeben:

```
001 ALTER TABLE "tbl_Mitglied"
002 ADD CONSTRAINT "Aktuelles Datum >= Geburtsdatum"
003 CHECK ("GebDat"<=CURRENT_DATE)
```

Hierdurch wird das Feld "GebDat" so festgelegt, dass es auf jeden Fall nicht in der Zukunft liegt. Ein Beitritt vor der Geburt ist nicht möglich. Bei Fehlversuchen wird ausgegeben: «Check constraint violation Aktuelles Datum >= Geburtsdatum table: tbl_Mitglied…»

Adresse



Die Adresse wird nicht direkt beim Mitglied eingetragen. Sie kann über das Feld "Gruppe_mitID" auch anderen Mitgliedern zugeordnet werden, obwohl so die Beziehung zwischen "tbl_Mitglied" und "tbl_Adresse" eine 1:1-Beziehung ist.

Postleitzahl und Ort sind aus der Tabelle "tbl_Adresse" ausgelagert, da sie gerade bei Vereinen in der Regel sehr häufig mehrfach vorkommen.

tbl_Adresse

	mitID	Straße_Nr	ortID	Nichtdruck	Telefon_fest
	3	Unter dem Baum 42	1	☐	05971/424242
	7	Konsumrennbahn 1a	2	☐	
	8	An dem Bach 162c	2	☐	09090/123234
	9	Kirchweg 13	0	☐	
	10	Industriegebiet Nord 4711	1	☐	14567/89123
▶+				☐	

Datenziel	
Formular:	*frm_Mitglied*
Makro:	*SelectGroup, Group*
Abfrage:	*Abfrage_Start*
Ansicht:	*viw_Adresse_Num_Namen, viw_Anschrift, viw_Filter_Mitglieder_komplett_AusEin, viw_Mitglieder_komplett*
Bericht:	keine direkt

Feldname	***Feldtyp***	***Beschreibung***
mitID	Integer	Primärschlüssel und gleichzeitig Fremdschlüssel nach "tbl_Mitglied"."ID"

Feldname	Feldtyp	Beschreibung
Straße_Nr	Text	Straße und Hausnummer zusammen in einem Feld
ortID	Integer	Fremdschlüssel nach "tbl_Ort"."ID"
Nichtdruck	Ja/Nein	Diese Adresse soll vom Druck ausgenommen werden, da dort niemand aus der Gruppe mehr wohnt.
Telefon_fest	Text	Festnetztelefonnummer, wird mit der Adresse gespeichert, da in der Regel max. eine Festnetznummer angegeben wird.

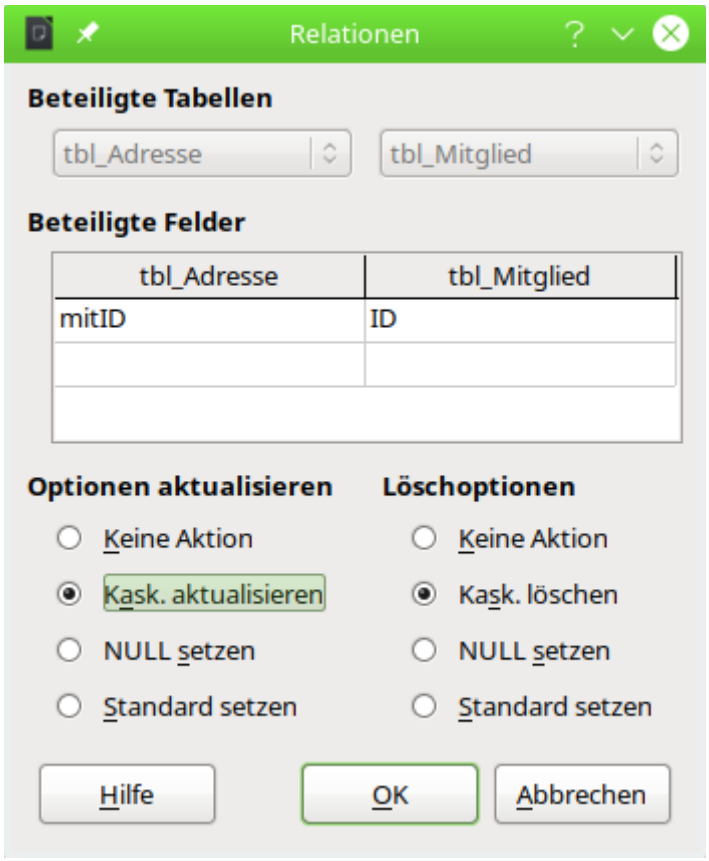


Abbildung 2: Die Beziehung ist so geregelt, dass bei einer Änderung der "ID" aus "tbl_Mitglied" die "mitID" aus "tbl_Adresse" angepasst wird: ***Kaskadierend aktualisieren****. Wird das mit der Adresse verbundene Mitglied aus der Tabelle gelöscht, so wird auch der Datensatz in der Adresse gelöscht:* ***Kaskadierend löschen****.*

In der Tabelle "tbl_Adresse" ist (bis auf den Primärschlüssel) keins der Felder als erforderlich gesetzt worden. Es ist also möglich, dort auch nur die Telefonnummer oder nur die Straße zu speichern.

tbl_Ort

	ID	Postleitzahl	Ort	Land
	0	42430	Heideland	DE
	1	98701	Bergdorf	DE
	2	02431	Platttown	DE
▸+	<Auto			

Datenziel	
Formular:	*frm_Mitglied*
Makro:	*SelectGroup*
Ansicht:	*viw_Anschrift, viw_Filter_Mitglieder_komplett_AusEin, viw_Mitglieder_komplett*
Bericht:	keine direkt

Feldname	***Feldtyp***	***Beschreibung***
ID	Integer	Primärschlüssel der Tabelle. Der Primärschlüssel muss eindeutig sein. Das Feld ist als Auto-Wert-Feld gesetzt.
Postleitzahl	Text	Postleitzahl
Ort	Text	Ort Eingabe erforderlich: Ja
Land	Text	Landeskürzel, 2 Zeichen, beide groß geschrieben

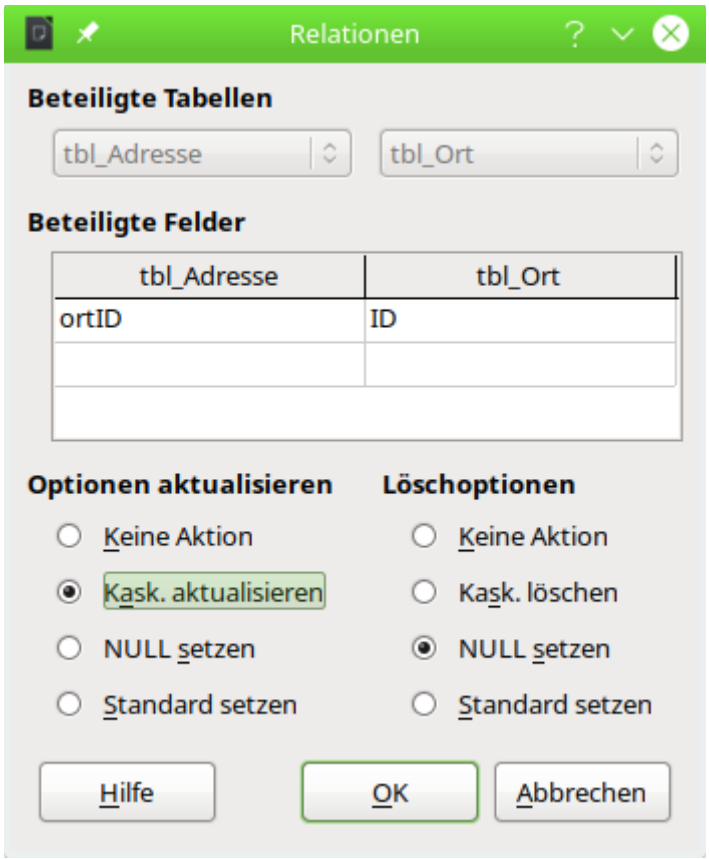
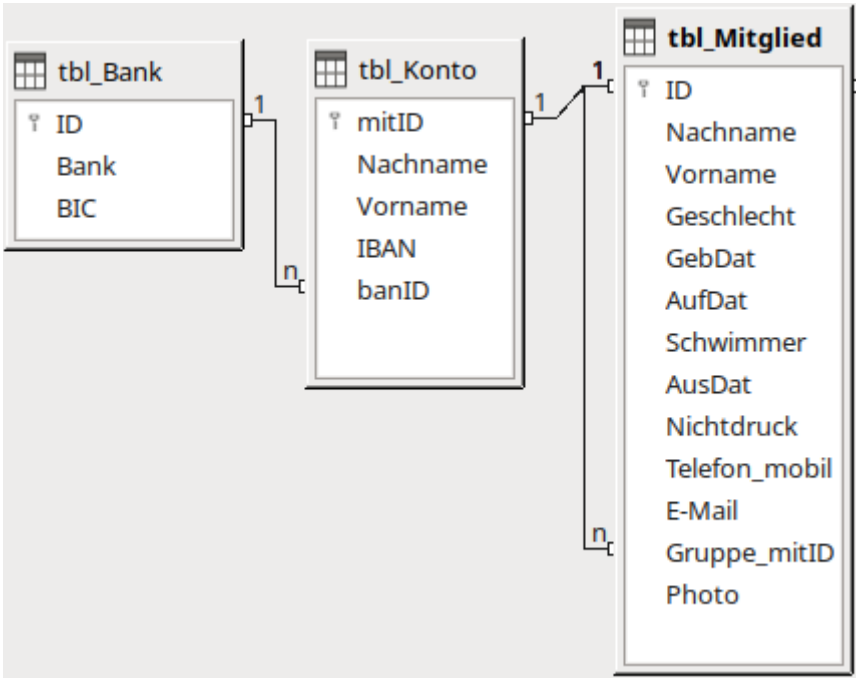


Abbildung 3: Die Beziehung ist so geregelt, dass bei einer Änderung der "ID" aus "tbl_Ort" die "ortID" aus "tbl_Adresse" angepasst wird: ***Kaskadierend aktualisieren****. Wird der mit der Adresse verbundene Ort aus der Tabelle gelöscht, so wird das Feld ortID als leeres Feld gesetzt:* ***NULL setzen****. Es kann also auch (vorübergehend) Adressen ohne Ortszuweisung geben.*

Konto



Das Konto wird nicht direkt beim Mitglied eingetragen. Es kann über das Feld "Gruppe_mitID" auch anderen Mitgliedern zugeordnet werden, obwohl so die Beziehung zwischen "tbl_Mitglied" und "tbl_Konto" eine 1:1-Beziehung ist.

Bank und BIC sind aus der Tabelle "tbl_Konto" ausgelagert, da sie gerade bei Vereinen in der Regel sehr häufig mehrfach vorkommen.

Die Kontoverwaltung kann durch die separaten Tabellen mit Hilfe der Einstellungen des Programms leichter abgewählt werden.

tbl_Konto

Datenziel	
Formular:	*frm_Mitglied*
Makro:	*SelectGroup*
Abfrage:	*qry_Beitraege, qry_Mitglied_Konto*
Ansicht:	*viw_Filter_Mitglieder_komplett_AusEin, viw_Mitglieder_komplett*
Bericht:	keine direkt

Feldname	***Feldtyp***	***Beschreibung***
mitID	Integer	Primärschlüssel und gleichzeitig Fremdschlüssel nach "tbl_Mitglied"."ID"
Nachname	Text	Nachname des Kontoinhabers / der Kontoinhaberin, falls nicht identisch mit dem aus "tbl_Mitglied"."ID"
Vorname	Text	Vorname des Kontoinhabers / der Kontoinhaberin, falls nicht identisch mit dem aus "tbl_Mitglied"."ID"
IBAN	Text	IBAN wird mit einem Leerzeichen nach 4 Zeichen eingegeben. Das erledigt ein maskiertes Feld, Länge des Feldes für maximale IBAN von 34 Zeichen deshalb 42 Zeichen. Eingabe erforderlich: Ja

Feldname	Feldtyp	Beschreibung
banID	Integer	Fremdschlüssel nach "tbl_Bank"."ID" Eingabe erforderlich: Ja

Die Verknüpfung zwischen "tbl_Konto" und "tbl_Mitglied" ist wie in *Abbildung 2* so geregelt, dass bei einer Änderung des Primärschlüssels von "tbl_Mitglied" der Fremdschlüssel angepasst wird. Bei einer Löschung in der "tbl_Mitglied" wird auch der Datensatz in "tbl_Konto" gelöscht.

tbl_Bank

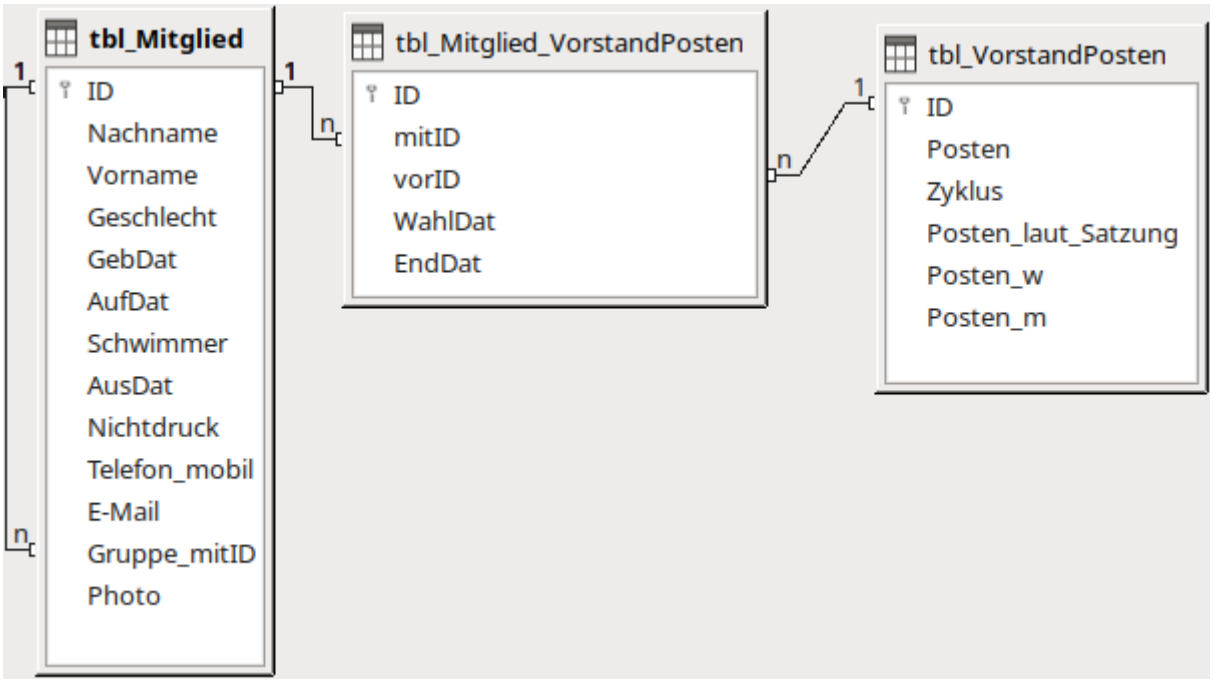
	ID	Bank	BIC
	1	Stadtsparkasse Rheine	WELADED1RHN
	4	Volksbank Münsterland Nord eG	GENODEM1IBB
▷+	<Auto		

Datenziel	
Formular:	*frm_Mitglied*
Makro:	*SelectGroup*
Abfrage:	*qry_Beitraege, qry_Mitglied_Konto*
Ansicht:	*viw_Filter_Mitglieder_komplett_AusEin, viw_Mitglieder_komplett*
Bericht:	keine direkt

Feldname	Feldtyp	Beschreibung
ID	Integer	Primärschlüssel der Tabelle. Der Primärschlüssel muss eindeutig sein. Das Feld ist als Auto-Wert-Feld gesetzt.
Bank	Text	Name der Bank Eingabe erforderlich: Ja
BIC	Text	BIC der Bank Eingabe erforderlich: Ja

Die Verknüpfung zwischen "tbl_Konto" und "tbl_Bank" ist wie in *Abbildung 3* so geregelt, dass bei einer Änderung des Primärschlüssels von "tbl_Bank" der Fremdschlüssel angepasst wird. Bei einer Löschung in der "tbl_Bank" wird der Fremdschlüssel "banID" in "tbl_Konto" auf **NULL** gesetzt.

Vorstand



Mitglieder können unterschiedliche Vorstandsposten besetzen. Gleiche Vorstandsposten werden nacheinander von unterschiedlichen Mitgliedern wahr genommen. Die 1:n-Beziehung zwischen "tbl_Mitglied" und "tbl_Mitglied_VorstandPosten" sowie die 1:n-Beziehung zwischen "tbl_VorstandPosten" und "tbl_Mitglied_VorstandPosten" ergeben so zusammen eine n:m-Beziehung zwischen "tbl_Mitglied" und "tbl_VorstandPosten".

tbl_Mitglied_VorstandPosten

	ID	mitID	vorID	WahlDat	EndDat
	2	6	1	31.01.21	
	3	7	2	31.01.22	
	8	3	4	31.01.22	
▸+	<Auto				

Datenziel	
Formular:	*frm_Berichte*
Makro:	*SelectBoard*
Abfrage:	*qry_Filter_Vorstand, qry_Jubilaeum, qry_Vorstand_aktuell, qry_Vorstand_aktuell_lt_Satzung*
Ansicht, Bericht:	keine direkt

Feldname	***Feldtyp***	***Beschreibung***
ID	Integer	Primärschlüssel der Tabelle. Der Primärschlüssel muss eindeutig sein. Das Feld ist als Auto-Wert-Feld gesetzt.
mitID	Integer	Fremdschlüssel nach "tbl_Mitglied"."ID" Eingabe erforderlich: Ja
vorID	Integer	Fremdschlüssel nach "tbl_VorstandPosten"."ID" Eingabe erforderlich: Ja

Feldname	Feldtyp	Beschreibung
WahlDat	Datum	Wahldatum Eingabe erforderlich: Ja
EndDat	Daum	Beendigung der Vorstandstätigkeit.

Die Verknüpfung zwischen "tbl_Mitglied_VorstandPosten" und "tbl_Mitglied" ist wie in *Abbildung 3* so geregelt, dass bei einer Änderung des Primärschlüssels von "tbl_Mitglied" der Fremdschlüssel angepasst wird. Bei einer Löschung in der "tbl_Mitglied" wird der Fremdschlüssel "mitID" in "tbl_Mitglied_VorstandPosten" auf **NULL** gesetzt. So bleibt zumindest die Belegung des Postens erhalten, auch wenn das entsprechende Mitglied nicht mehr in der Datenbank verzeichnet ist. Über Berichte sollte deswegen der Vorstand auch außerhalb der Datenbank gesichert werden.

Zur Absicherung des Enddatums/Rücktrittsdatums wird unter **Extras → SQL** eingegeben:

```
001 ALTER TABLE "tbl_Mitglied_VorstandPosten"
002 ADD CONSTRAINT "Rücktrittsdatum >= Wahldatum"
003 CHECK ( COALESCE("EndDat","WahlDat")>="WahlDat" )
```

Hierdurch wird das Feld "EndDat" so festgelegt, dass es auf jeden Fall nicht vor dem "WahlDat" liegt. Ein Rücktritt vor der Wahl ist nicht möglich. Bei Fehlversuchen wird ausgegeben: «Check constraint violation Rücktrittsdatum >= Wahldatum table: tbl_Mitglied_VorstandPosten...»

tbl_VorstandPosten

	ID	Posten	Zyklus	Posten_laut_Satzung	Posten_w	Posten_m
	1	Vorsitzende(r)	u	☑	Vorsitzende	Vorsitzender
	2	Geschäftsführer(in)	g	☑	Geschäftsführerin	Geschäftsführer
	3	Schatzmeister(in)	u	☑	Schatzmeisterin	Schatzmeister
	4	stellv. Vorsitzende(r)	g	☑	stellv. Vorsitzende	stellv. Vorsitzender
	5	1. Jugendwart(in)	u	☑	1. Jugenwartin	1. Jugendwart
	6	2. Jugendwart(in)	g	☑	2. Jugendwartin	2. Jugendwart
	7	Sportwart(in)	u	☑	Sportwartin	Sportwart
	8	stellv. Sportwart(in)	g	☑	stellv. Sportwartin	stellv. Sportwart
	9	Protokollführer(in)	u	☑	Protokollführerin	Protokollführer

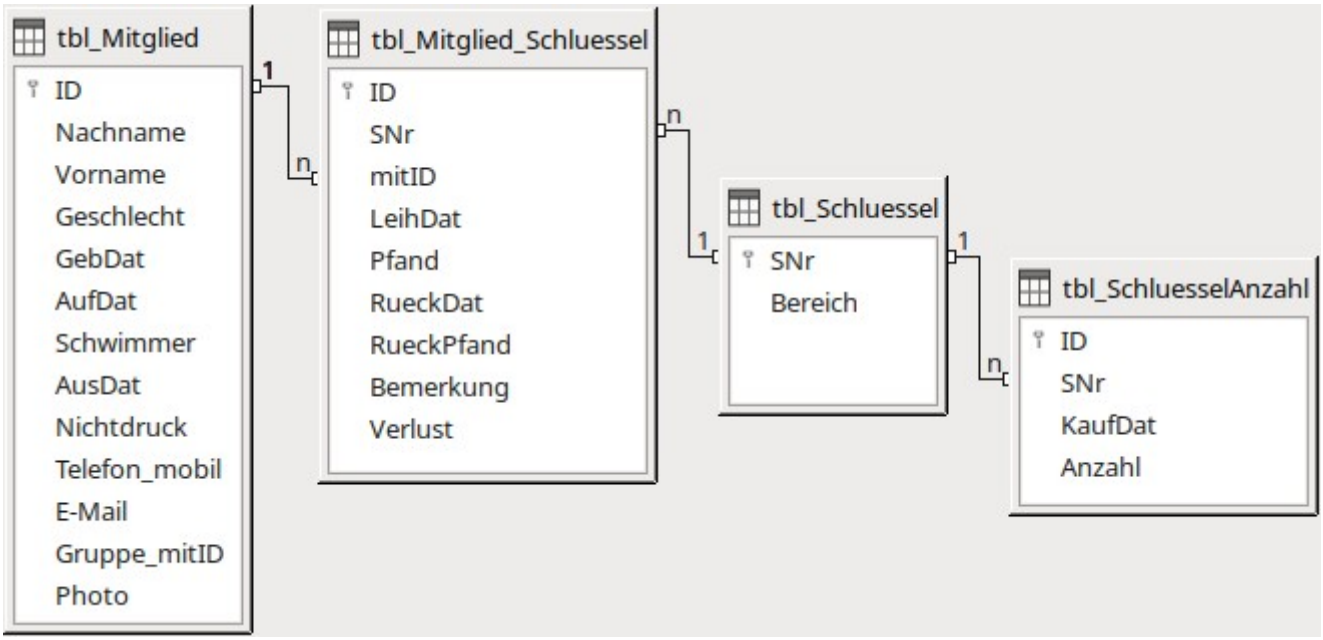
Datenziel	
Formular:	*frm_Berichte, frm_Vorstand*
Makro:	*SelectBoard*
Abfrage:	*qry_Jubilaeum, qry_Vorstand_aktuell, qry_Vorstand_aktuell_lt_Satzung*
Ansicht, Bericht:	keine direkt

Feldname	Feldtyp	Beschreibung
ID	Integer	Primärschlüssel der Tabelle. Der Primärschlüssel muss eindeutig sein. Das Feld ist als Auto-Wert-Feld gesetzt.
Posten	Text	Beschreibung des Postens Eingabe erforderlich: Ja
Zyklus	Text	Wahlen in geraden ('g') oder ungeraden ('u') Jahren? Wenn nicht durch Satzung vorgegeben: '-'

Feldname	Feldtyp	Beschreibung
Posten_laut_Satzung	Ja/Nein	Ist der Posten in der Satzung festgelegt? Dann hier: Ja
Posten_w	Text	Für Unterschriftsfelder: Beschreibung des Postens in weiblicher Form
Posten_m	Text	Für Unterschriftsfelder: Beschreibung des Postens in männlicher Form

Die Verknüpfung zwischen "tbl_Mitglied_VorstandPosten" und "tbl_VorstandPosten" ist wie in *Abbildung 3* so geregelt, dass bei einer Änderung des Primärschlüssels von "tbl_VorstandPosten" der Fremdschlüssel angepasst wird. Bei einer Löschung in der "tbl_VorstandPosten" wird der Fremdschlüssel "vorID" in "tbl_Mitglied_VorstandPosten" auf **NULL** gesetzt.

Schlüssel



Mitglieder können unterschiedliche Schlüssel entleihen. Schlüssel für gleiche Schließbereiche können von mehreren Mitgliedern entliehen werden. Die 1:n-Beziehung zwischen "tbl_Mitglied" und "tbl_Mitglied_Schluessel" sowie die 1:n-Beziehung zwischen "tbl_Schluessel" und "tbl_Mitglied_MitgliedSchluessel" ergeben so zusammen eine n:m-Beziehung zwischen "tbl_Mitglied" und "tbl_Schluessel".

Manchmal sind zu wenig Schlüssel verfügbar. So muss beständig der Bestand kontrollierbar sein und ggf. neue Schlüssel nachgekauft werden. Deshalb die Bestandserfassung in "tbl_SchluesselAnzahl".

Die Schlüsselverwaltung kann mit Hilfe der Einstellungen des Programms abgewählt werden.

tbl_MitgliedSchluessel

	ID	SNr	mitID	LeihDat	Pfand	RueckDat	RueckPfand	Bemerkung	Verlust
	0	GS 2	4	28.07.22	15,00 €				☐
	1	GS 2	12	17.08.22	15,00 €				☐
	2	GS 1	3	17.08.22	15,00 €				☐
	3	GS 3	3	17.08.22	15,00 €				☐
▷+	<Auto								☐

Datenziel	
Formular:	*frm_Schluessel*
Makro:	*SelectKey*
Abfrage:	*qry_Schluessel_Uebersicht*
Ansicht:	*viw_Schluessel_Mitglied*
Bericht:	keine direkt

Feldname	***Feldtyp***	***Beschreibung***
ID	Integer	Primärschlüssel der Tabelle. Der Primärschlüssel muss eindeutig sein. Das Feld ist als Auto-Wert-Feld gesetzt.
SNr	Text	Fremdschlüssel zur Tabelle "tbl_Schluessel", wie dort ein Textfeld mit maximal 5 Zeichen Eingabe erforderlich: Ja
mitID	Integer	Fremdschlüssel zur Tabelle "tbl_Mitglied" Eingabe erforderlich: Ja
LeihDat	Datum	Ausleihdatum Eingabe erforderlich: Ja
Pfand	Dezimal	Pfand für die Ausleihe Eingabe erforderlich: Ja
RueckDat	Datum	Rückgabedatum
RueckPfand	Dezimal	Rückgabe des Pfandes, eventuell bei verbogenem Schlüssel mit Abzug
Bemerkung	Text	Grund für die Zuweisung zu einem besonderen Schließkreis o. ä.
Verlust	Ja/Nein	Ist der Schlüssel verloren, so wird hier mit 'Ja' der Schlüssel komplett aus dem System genommen. Der Pfand wird nicht zurück gezahlt.

Die Verknüpfung zwischen "tbl_MitgliedSchluessel" und "tbl_Mitglied" ist wie in *Abbildung 3* so geregelt, dass bei einer Änderung des Primärschlüssels von "tbl_Mitglied" der Fremdschlüssel angepasst wird. Bei einer Löschung in der "tbl_Mitglied" wird der Fremdschlüssel "mitID" in "tbl_MitgliedSchluessel" auf **NULL** gesetzt.

Zur Absicherung des Rückgabedatums wird unter **Extras → SQL** eingegeben:

```
001 ALTER TABLE "tbl_Mitglied_Schluessel"
002 ADD CONSTRAINT "Rückgabedatum >= Ausleihdatum"
003 CHECK ( COALESCE("RueckDat","LeihDat")>="LeihDat" )
```

Hierdurch wird das Feld "RueckDat" so festgelegt, dass es auf jeden Fall nicht vor dem "LeihDat" liegt. Eine Rückgabe vor der Ausleihe ist nicht möglich. Bei Fehlversuchen wird ausgegeben:
«Check constraint violation Rückgabedatum >= Ausleihdatum table: tbl_Mitglied_Schluessel...»

Zur Absicherung des Rückgabepfandes wird unter **Extras → SQL** eingegeben:

```
001 ALTER TABLE "tbl_Mitglied_Schluessel"
002 ADD CONSTRAINT "Rückgabepfand <= Pfand"
003 CHECK ( COALESCE("RueckPfand","Pfand")<="Pfand" )
```

Hierdurch wird das Feld "RueckPfand" so festgelegt, dass es auf jeden Fall nicht größer als "Pfand" ist. Es soll nicht mehr Pfand zurückgezahlt werden als tatsächlich gezahlt wurde. Bei Fehlversuchen wird ausgegeben:
«Check constraint violation Rückgabepfand <= Pfand table: tbl_Mitglied_Schluessel...»

tbl_Schluessel

	SNr	Bereich
	E 001	Werkstatt
	E 002	Küche
	GS 1	Vorstand
	GS 2	Mitglied einfach
	GS 3	Rennboote
	GS 4	Kraftraum
▷+		

Datenziel	
Formular:	*frm_Schluessel*
Makro:	*SelectKey*
Ansicht, Bericht:	keine direkt

Feldname	***Feldtyp***	***Beschreibung***
SNr	Text	Primärschlüssel, Text mit 5 Zeichen
Bereich	Text	Schließbereich des Schlüssels Eingabe erforderlich: Ja

Die Verknüpfung zwischen "tbl_MitgliedSchluessel" und "tbl_Schluessel" ist wie in *Abbildung 3* so geregelt, dass bei einer Änderung des Primärschlüssels von "tbl_Schluessel" der Fremdschlüssel angepasst wird. Bei einer Löschung in der "tbl_Schluessel" wird der Fremdschlüssel "SNr" in "tbl_MitgliedSchluessel" auf **NULL** gesetzt.

tbl_SchluesselAnzahl

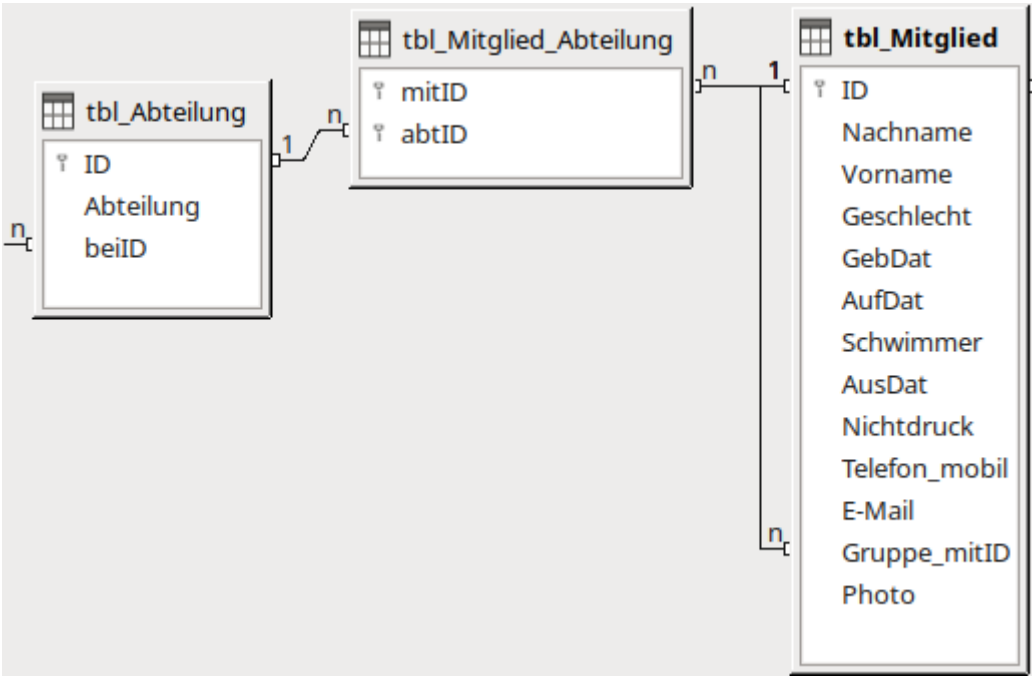
	ID	SNr	KaufDat	Anzahl
	0	GS 1	01.01.22	10
	1	GS 2	01.01.22	20
	2	GS 3	01.01.22	10
▷+	<Auto			

Datenziel	
Makro:	*SelectKey*
Ansicht, Bericht:	keine direkt

Feldname	Feldtyp	Beschreibung
ID	Integer	Primärschlüssel der Tabelle. Der Primärschlüssel muss eindeutig sein. Das Feld ist als Auto-Wert-Feld gesetzt.
SNr	Text	Fremdschlüssel zur "tbl_Schluessel", wie dort Text mit 5 Zeichen Eingabe erforderlich: Ja
KaufDat	Datum	Kaufdatum für den Schlüssel Eingabe erforderlich: Ja
Anzahl	Small Integer	Anzahl der gekauften Schlüssel für den entsprechenden Schließbereich. Small Integer wurde hier als Feldtyp gewählt, da direkt bei der Einrichtung eventuell mehr Schlüssel benötigt werden als mit Tiny Integer (bis +127 bei der HSQLDB) möglich wären. Eingabe erforderlich: Ja

Die Verknüpfung zwischen "tbl_SchluesselAnzahl" und "tbl_Schluessel" ist wie in *Abbildung 3* so geregelt, dass bei einer Änderung des Primärschlüssels von "tbl_Schluessel" der Fremdschlüssel angepasst wird. Bei einer Löschung in der "tbl_Schluessel" wird der Fremdschlüssel "SNr" in "tbl_SchluesselAnzahl" auf **NULL** gesetzt.

Abteilung



Ein Mitglied kann in mehreren Abteilungen sein. In jeder Abteilung sind mehrere Mitglieder. Die Beziehung zwischen "tbl_Mitglied" und "tbl_Abteilung" ist also eine n:m-Beziehung. Diese Beziehung wird bei der Tabellenkonstruktion durch eine Tabelle gelöst, in der die Primärschlüssel beider betroffener Tabellen enthalten sind. Hier ist das die "tbl_Mitglied_Abteilung", die lediglich die beiden Fremdschlüssel als gemeinsame Primärschlüssel enthält.

Da Abteilungen häufig mit separaten Beiträgen verbunden sind, ist in "tbl_Abteilung" eine Verknüpfung zu der Beitragstabelle enthalten.

Die Abteilungsverwaltung kann mit Hilfe der Einstellungen des Programms abgewählt werden.

tbl_Mitglied_Abteilung

	mitID	abtID
	3	3
	4	2
	4	3
	5	1
	5	4

Datenziel
Formular: *frm_Mitglied*
Ansicht: *Ansicht_Datum, viw_Beitrag*
Bericht: keine direkt

Feldname	**Feldtyp**	**Beschreibung**
mitID	Integer	Primärschlüssel zusammen mit "abtID", Fremdschlüssel zu "tbl_Mitglied"
abtID	Integer	Primärschlüssel zusammen mit "mitID", Fremdschlüssel zu "tbl_Abteilung"

Die Verknüpfung zwischen "tbl_Mitglied_Abteilung" und "tbl_Mitglied" ist wie in *Abbildung 2* so geregelt, dass bei einer Änderung des Primärschlüssels von "tbl_Mitglied" der Fremdschlüssel angepasst wird. Bei einer Löschung in der "tbl_Mitglied" wird auch der Datensatz in "tbl_Mitglied_Abteilung" gelöscht.

tbl_Abteilung

	ID	Abteilung	beiID
	1	Jugend- und Wildwasserabtlg.	
	2	Frauenabteilung	
	3	Wanderabteilung	
	4	Rennabteilung	6
	7	Drachenbootabteilung	9
▷+	<Auto		

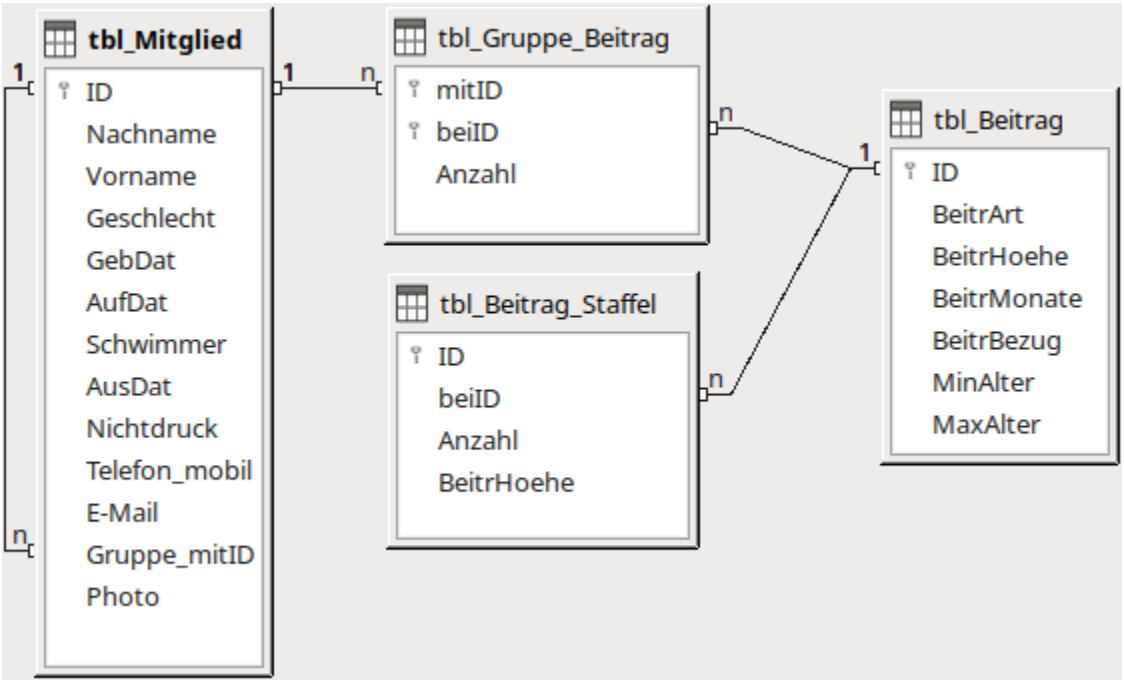
Datenziel
Formular: *frm_Einstellungen, frm_Mitglied*
Ansicht: *Ansicht_Datum, viw_Beitrag*
Bericht: keine direkt

Feldname	**Feldtyp**	**Beschreibung**
ID	Integer	Primärschlüssel der Tabelle. Der Primärschlüssel muss eindeutig sein. Das Feld ist als Auto-Wert-Feld gesetzt.
Abteilung	Text	Beschreibung der Abteilung Eingabe erforderlich: Ja

Feldname	Feldtyp	Beschreibung
beiID	Integer	Fremdschlüssel zur Tabelle "tbl_Beitrag" Hier ist keine Eingabe erforderlich, weil es Abteilungen ohne Beitragsbezug geben kann und außerdem auch die Möglichkeit bestehen soll, die gesamte Datenbank ohne das Beitragsmodul zu betreiben.

Die Verknüpfung zwischen "tbl_Mitglied_Abteilung" und "tbl_Abteilung" ist wie in *Abbildung 2* so geregelt, dass bei einer Änderung des Primärschlüssels von "tbl_Abteilung" der Fremdschlüssel angepasst wird. Bei einer Löschung in der "tbl_Abteilung" wird auch der Datensatz in "tbl_Mitglied_Abteilung" gelöscht.

Beitrag



Der Beitrag wird bei dem Mitglied eingetragen, das über die Gruppierung von Mitgliedern (anhand der Adresse oder des Kontos) als «Gruppenleiter(in)» vorgesehen ist. Ein Mitglied kann für verschiedene Beiträge zahlen müssen. Ein bestimmter Beitrag ist aber wiederum bei verschiedenen Mitgliedern vorgesehen. Die n:m-Beziehung zwischen "tbl_Mitglied" und "tbl_Beitrag" wird über zwei 1:n-Beziehungen zur "tbl_Gruppe_Beitrag" geregelt.

Bei manchen Beitragsarten kann der Beitrag gestaffelt sein. So zahlen in dem Verein, der als Ausgangspunkt für diese Datenbank diente, ein zweites oder drittes Mitglied einer Gruppe einen immer niedrigeren Abteilungsbeitrag für die «Rennabteilung». Das entspricht dann einer Gruppenermäßigung. Bei den Booten hingegen steigt der Preis pro Boot mit der Anzahl der gelagerten Boote. Dies ist entsprechend in "tbl_Beitrag_Staffel" hinterlegt.

Die Beitragsverwaltung kann mit Hilfe der Einstellungen des Programms abgewählt werden.

tbl_Gruppe_Beitrag

	mitID	beiID	Anzahl
	3	3	1
	3	4	1
	8	3	1
	9	3	1
	10	3	1

Datenziel	
Formular:	*Einzelmessung_System*
Makro:	*SubscriptionSave*
Ansicht:	*viw_Beitrag*
Bericht:	keine direkt

Feldname	**Feldtyp**	**Beschreibung**
mitID	Integer	Primärschlüssel zusammen mit "beiID", Fremdschlüssel zu "tbl_Mitglied"
beiID	Integer	Primärschlüssel zusammen mit "mitID", Fremdschlüssel zu "tbl_Beitrag"
Anzahl	Tiny Integer	Wie viele Mitglieder in der Gruppe müssen den entsprechenden Beitrag zahlen?

Die Verknüpfung zwischen "tbl_Gruppe_Beitrag" und "tbl_Mitglied" ist wie in *Abbildung 2* so geregelt, dass bei einer Änderung des Primärschlüssels von "tbl_Mitglied" der Fremdschlüssel angepasst wird. Bei einer Löschung in der "tbl_Mitglied" wird auch der Datensatz in "tbl_Gruppe_Beitrag" gelöscht.

tbl_Beitrag

	ID	BeitrArt	BeitrHoehe	BeitrMonate	BeitrBezug	MinAlter	MaxAlter
	1	Kinder bis einschl. 13 Jahre	5,50 €	1	Mit	0	13
	2	Jugendliche von 14 - 17 Jahre	7,00 €	1	Mit	14	17
	3	Erwachsene ab 18 Jahre	10,00 €	1	Mit	18	127
	4	Eheleute, Paare, Familien max. 2 Pers. ü	15,00 €	1	MitG	0	127
	6	Zusatzbeitr. Rennabtlg. ab 13 J, gestaffe		12	Abt	13	127
	9	Zusatzbeitrag Drachenbootabteilung	60,00 €	12	Abt	0	127
	10	Bootständermiete Kajak (max. 3 Kajaks		1	Boot	0	127
	13	Bootsständermiete Canadier (nur 1 Can	4,00 €	1	Boot	0	127
▷+	<Aut(						

Datenziel	
Formular:	*Einzelmessung_System, frm_Einstellungen*
Ansicht:	*viw_Beitrag, viw_Beitrag_Gruppe_kalkuliert*
Bericht:	keine direkt

Feldname	Feldtyp	Beschreibung
ID	Integer	Primärschlüssel der Tabelle. Der Primärschlüssel muss eindeutig sein. Das Feld ist als Auto-Wert-Feld gesetzt.
BeitrArt	Text	Beschreibung des Beitrags Eingabe erforderlich: Ja
BeitrHoehe	Dezimal	Nicht alle Felder sind mit einem entsprechenden Betrag versehen. Hier gibt es Beiträge, die nach einer entsprechenden Staffel nicht einfach aufsummiert werden. Siehe dazu die folgende Tabelle "tbl_Beitrag_Staffel"
BeitrMonate	Tiny Integer	Anzahl der Monate, für die der angegebenen Beitrag gilt Eingabe erforderlich: Ja
BeitrBezug	Text	Maximal 10 Zeichen, hier: 'Mit' → einfacher Mitgliedsbeitrag 'MitG' → Gruppenbeitrag, z. B. Für Paare, Familien 'Abt' → Abteilungsbeitrag 'Boot' → Bootsständermiete Eingabe erforderlich: Ja
MinAlter	Tiny Integer	Mindestalter, sofern nicht vorgegeben '0' Eingabe erforderlich: Ja
MaxAlter	Tiny Integer	Maximalalter, sofern nicht vorgegeben '127' (Tiny Integer) Eingabe erforderlich: Ja

Die Verknüpfung zwischen "tbl_Gruppe_Beitrag" und "tbl_Beitrag" ist wie in *Abbildung 2* so geregelt, dass bei einer Änderung des Primärschlüssels von "tbl_Beitrag" der Fremdschlüssel angepasst wird. Bei einer Löschung in der "tbl_Beitrag" wird auch der Datensatz in "tbl_Gruppe_Beitrag" gelöscht.

Zur Absicherung der Monatszahl wird unter **Extras → SQL** eingegeben:

```
001 ALTER TABLE "tbl_Beitrag"
002 ADD CONSTRAINT "Monatszahl 1 bis 12"
003 CHECK ("BeitrMonate" BETWEEN 1 AND 12)
```

Hierdurch wird das Feld "BeitrMonate" auf Eingaben von 1 bis 12 begrenzt. Bei Fehlversuchen wird ausgegeben:
«Check constraint violation Monatszahl 1 bis 12 table: tbl_Beitrag…»

tbl_Beitrag_Staffel

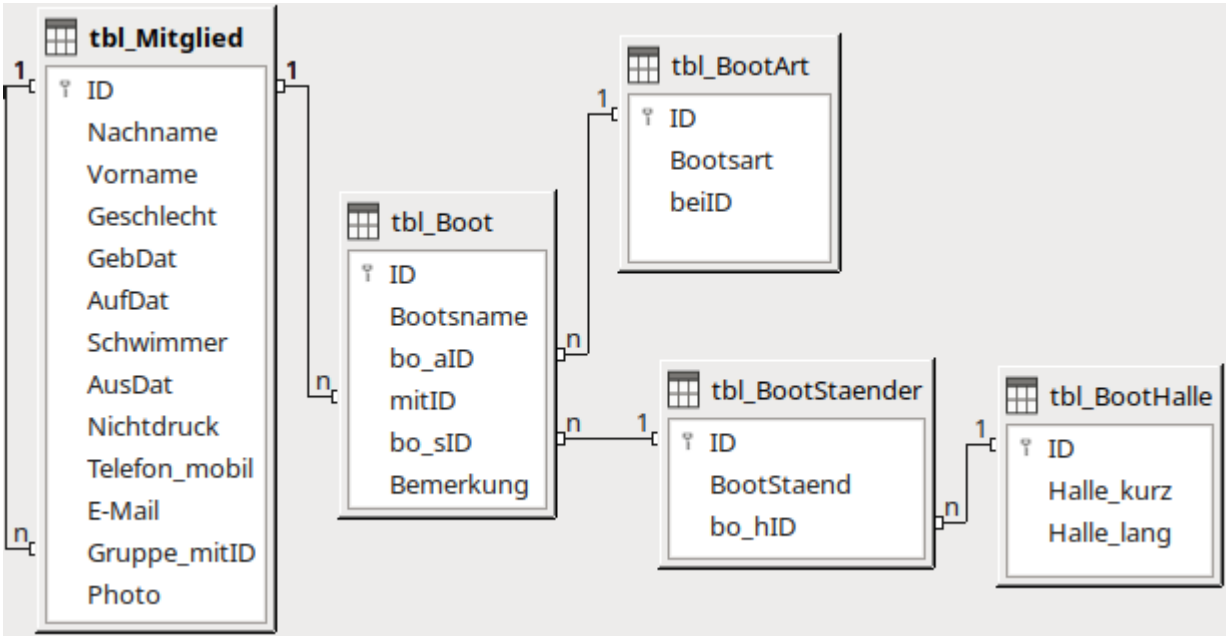
	ID	beiID	Anzahl	BeitrHoehe
	1	6	1	50,00 €
	2	6	2	90,00 €
	3	6	3	120,00 €
	4	6	4	150,00 €
	5	6	5	180,00 €
	6	10	1	2,00 €
	7	10	2	4,50 €
	8	10	3	10,00 €
	9	10	4	20,00 €
	10	10	5	40,00 €
▶+				

Datenziel	
Formular:	*frm_Einstellungen*
Ansicht:	*viw_Beitrag*
Bericht:	keine direkt

Feldname	Feldtyp	Beschreibung
ID	Integer	Primärschlüssel, kein Auto-Wert, da übersichtliche Anzahl an Einrägen
beiID	Integer	Fremdschlüssel zu Tabelle "tbl_Beitrag" Eingabe erforderlich: Ja
Anzahl	TinyInteger	Beitrag für 1, 2, 3 ... Personen/Booten Eingabe erforderlich: Ja
BeitrHoehe	Dezimal	Höhe des Beitrags bei 1, 2, 3 ... Personen/Booten Eingabe erforderlich: Ja

Die Verknüpfung zwischen "tbl_Beitrag_Staffel" und "tbl_Beitrag" ist wie in *Abbildung 2* so geregelt, dass bei einer Änderung des Primärschlüssels von "tbl_Beitrag" der Fremdschlüssel angepasst wird. Bei einer Löschung in der "tbl_Beitrag" wird auch der Datensatz in "tbl_Beitrag_Staffel" gelöscht.

Boot



Jedes Mitglied kann verschiedene Boote in Bootsständern des Vereins lagern. Die Boote sind in "tbl_Boot" verzeichnet.

Boote werden nach verfügbaren Bootsarten ("tbl_BootArt") eingeteilt und haben auch einen entsprechenden Beitragsbezug, weil sie unterschiedlich viel Platz einnehmen. Hier sind dies Canadier (mit einem Stechpaddel zu fahren) und Kajaks. Canadier sind als Wanderboot deutlich breiter und deshalb nur an bestimmten Stellen lagerbar und mit einem besonderen Beitrag versehen.

Die Bootsständer aus "tbl_BootStaender" befinden sich in unterschiedlichen Hallen. Sie können in verschiedenen Hallen gleiche Bezeichnungen haben, sind nur zusammen mit der Hallenbezeichnung aus "tbl_BootHalle" schließlich eindeutig zu erkennen.

Die Bootsständerverwaltung kann mit Hilfe der Einstellungen des Programms abgewählt werden.

tbl_Boot

	ID	Bootsname	bo_aID	mitID	bo_sID	Bemerkung
	0	Moin	2	3	3	Zweier mit Spritzdecke, zum Knien
	1	Ems	1	3	5	
	3	Sturmvögel	1	4	6	
	4	Seeblick	2	10	9	Zweierkajak
	13	Autonom	2	8	10	
	14	Canadier für alle	7	9	11	
▷+	<Auto					

Datenziel	
Formular:	*frm_Boot*
Makro:	*SelectBoatRack, NewBoatAllowed*
Abfrage:	*qry_Bootsstaender*
Ansicht:	*viw_Beitrag, viw_Boote_Mitglied*
Bericht:	keine direkt

Feldname	***Feldtyp***	***Beschreibung***
ID	Integer	Primärschlüssel der Tabelle. Der Primärschlüssel muss eindeutig sein. Das Feld ist als Auto-Wert-Feld gesetzt.
Bootsname	Text	Name des Bootes Eingabe erforderlich: Ja
bo_aID	Integer	Fremdschlüssel zur Tabelle "tbl_BootArt". Ohne die BootArt ist die Ermittlung einer eindeutigen Bootsständermiete nicht möglich. Eingabe erforderlich: Ja
mitID	Integer	Fremdschlüssel zur Tabelle "tbl_Mitglied" Auch Boote, die bei einer Aufräumaktion keinen Mitglied zugeordnet werden können, sollen verzeichnet werden können. Deswegen ist hier die Eingabe auf nicht erforderlich belassen worden.
bo_sID	Integer	Fremdschlüssel zur Tabelle "tbl_BootStaender" Eingabe erforderlich: Ja
Bemerkung	Text	Allgemeine Bemerkungen

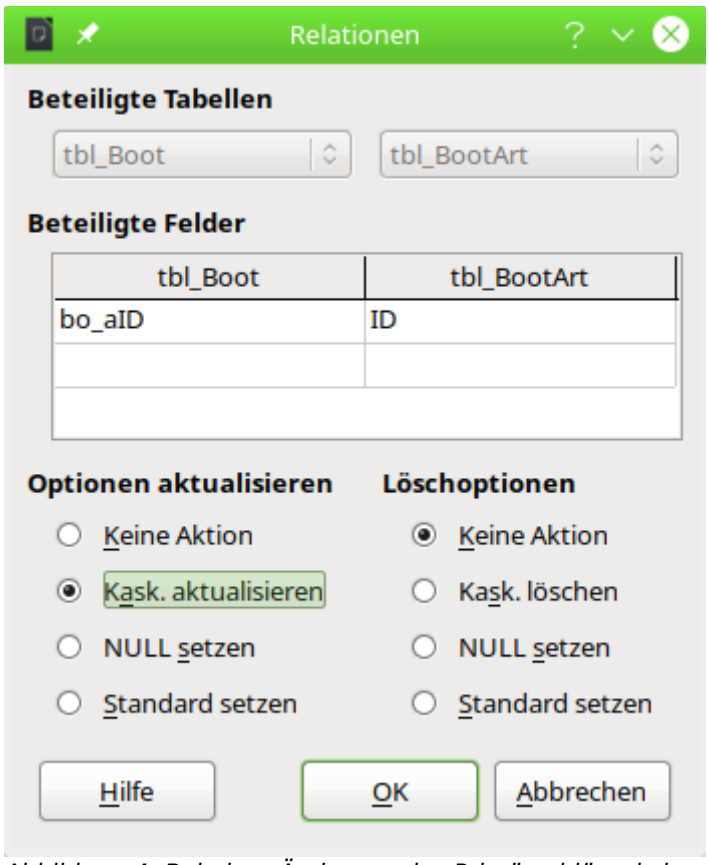


*Abbildung 4: Bei einer Änderung des Primärschlüssels in "tbl_BootArt" soll der Fremdschlüssel angepasst werden. Ein Löschen des Schlüssels ist nicht möglich, da das Feld in "tbl_Boot" als **Eingabe erforderlich → 'Ja'** definiert ist.*

Die Verknüpfung zwischen "tbl_Boot" und "tbl_Mitglied" ist wie in *Abbildung 3* so geregelt, dass bei einer Änderung des Primärschlüssels von "tbl_Mitglied" der Fremdschlüssel angepasst wird. Bei einer Löschung in der "tbl_Mitglied" wird der Fremdschlüssel "mitID" in "tbl_Boot" auf **NULL** gesetzt. So könnte ein Boot z. B. von einem Mitglied zurückgelassen werden und in das Eigentum des Vereins übergehen.

Neben der "bo_aID" ist auch die "bo_sID" zu den Bootsständern so verknüpft, dass ein Löschen des Bootsständers nicht möglich ist, solange ein Boot dort liegt. Siehe: *Abbildung 4*.

tbl_BootArt

	ID	Bootsart	beiID
	1	K1	10
	2	K2	10
	3	K4	10
	4	C1	13
	5	C2	13
	6	C3	13
	7	C4	13
	8	C7	13
	9	C10	13
▸+	<Auto		

Datenziel	
Formular:	*frm_Boot, frm_Einstellungen*
Makro:	*SelectBoat, NewBoatAllowed*
Abfrage:	*qry_Bootsstaender*
Ansicht:	*viw_Beitrag, viw_Boote_Mitglied*
Bericht:	keine direkt

Feldname	***Feldtyp***	***Beschreibung***
ID	Integer	Primärschlüssel der Tabelle. Der Primärschlüssel muss eindeutig sein. Das Feld ist als Auto-Wert-Feld gesetzt.
Bootsart	Text	Kurze Beschreibung der Bootsart, max. 10 Zeichen, hier: 'K1', 'K2', 'C4' ... Eingabe erforderlich: Ja
beiID	Integer	Fremdschlüssel zur Tabelle "tbl_Beitrag". Eingabe nicht erforderlich, da der Beitrag zu den abwählbaren Modulen gehört.

Die Verknüpfung zwischen "tbl_Boot" und "tbl_BootArt" ist wie in *Abbildung 3* so geregelt, dass bei einer Änderung des Primärschlüssels von "tbl_BootArt" der Fremdschlüssel angepasst wird. Bei einer Löschung in der "tbl_BootArt" ist nicht möglich

tbl_BootStaender

	ID	BootStaend	bo_hID
	1	0.1	3
	2	0.2	3
	3	0.3	3
	4	1.1	3
	5	1.2	3
	6	1.3	3
	7	1.4	3
	8	2.1	3

Datenziel	
Formular:	*frm_Boot*
Makro:	*SelectBoatRack*
Abfrage:	*qry_Bootsstaender*
Ansicht:	*viw_Boote_Mitglied*
Bericht:	keine direkt

Feldname	Feldtyp	Beschreibung
ID	Integer	Primärschlüssel der Tabelle. Der Primärschlüssel muss eindeutig sein. Das Feld ist als Auto-Wert-Feld gesetzt.
BootStaend	Text	Kurzer Text, max. 10 Zeichen, hier: '1.1', '1.2' ... Eingabe erforderlich: Ja
bo_hID	Integer	Fremdschlüssel zur Tabelle "tbl_BootHalle" Eingabe erforderlich: Ja

Die Verknüpfung zwischen "tbl_Boot" und "tbl_BootStaender" ist wie in *Abbildung 3* so geregelt, dass bei einer Änderung des Primärschlüssels von "tbl_BootStaender" der Fremdschlüssel angepasst wird. Eine Löschung in der "tbl_BootStaender" ist nicht möglich, sofern in "tbl_Boot" dieser Ständer benutzt wird.

tbl_BootHalle

	ID	Halle_kurz	Halle_lang
	1	B1	Rennboote Bootshaus
	2	B2	Wanderboot Bootshaus
	3	G1	Garage 1
	4	G2	Garage 2
	5	H1	Bootshalle Tor 1
	6	H2	Bootshalle Tor 2
	7	H3	Bootshalle Tor 3
	8	H4	Bootshalle Tor 4

Datenziel	
Formular:	*frm_Boot*
Makro:	*SelectBoatRack*
Abfrage:	*qry_Bootsstaender*
Ansicht:	*viw_Boote_Mitglied*
Bericht:	keine direkt

Feldname	Feldtyp	Beschreibung
ID	Integer	Primärschlüssel der Tabelle. Der Primärschlüssel muss eindeutig sein. Das Feld ist als Auto-Wert-Feld gesetzt.
Halle_kurz	Text	Kurzbeschreibung der Halle, max. 10 Zeichen Eingabe erforderlich: Ja
Halle_lang	Text	Längerer Text zur Verwendung in Berichten

Die Verknüpfung zwischen "tbl_BootStaender" und "tbl_BootHalle" ist wie in *Abbildung 4* so geregelt, dass bei einer Änderung des Primärschlüssels von "tbl_BootHalle" der Fremdschlüssel

angepasst wird. Eine Löschung in der "tbl_BootHalle" ist nicht möglich, wenn der Fremdschlüssel "bo_hID" in "tbl_BootStaender" benutzt wird.

Tabellen ohne Relationen

Einige Tabellen haben keine direkte Beziehung zu anderen Tabellen. Sie werden für Abfragen, Formulare und Berichte genutzt.

tbl_Altergruppen

	ID	MinAlter	MaxAlter
	1	0	6
	2	7	14
	3	15	18
	4	19	26
	5	27	40
	6	41	60
	7	60	120
▶+			

Datenziel	
Abfrage:	*qry_Filter_Austritt*
Ansicht:	*viw_Statistik_Alter_Detail*
Bericht:	keine direkt

Feldname	***Feldtyp***	***Beschreibung***
ID	Integer	Primärschlüssel, wegen der übersichtlichen Tabelle kein Auto-Wert
MinAlter	Tiny Integer	Untere Altersgrenze für die entsprechende Gruppe Eingabe erforderlich: Ja
MaxAlter	Tiny Integer	Obere Altersgrenze für die entsprechende Gruppe Eingabe erforderlich: Ja

tbl_Einstellungen

ID	Vereinsname	Anschrift	Ort	Homepage	EMail	IBAN
☑	LibreOffice Base	Zwischen den Bäum	Pusemuck	https://de.libreoffi	users@de.libre(	DE12 3456 789(

BIC	Bank	Kontover...	Beitragsver...	Schluesselver...	Bootstaenderver...	Abteilungsver...	Logo
BFALO1BA	Banko Phanta	☑	☑	☑	☑	☑	<OBJECT>

Datenziel	
Formular:	*frm_Einstellungen, frm_Mitglied*
Abfrage:	*qry_Forms, qry_Jubilaeum*
Ansicht:	*viw_Reports*
Bericht:	keine direkt
Makro:	*Preferences, SelectGroup*

Feldname	Feldtyp	Beschreibung
ID	Ja/Nein	Primärschlüssel, soll nur einen einzigen Datensatz enthalten, deshalb Ja/Nein
Vereinsname	Text	Name des Vereins
Anschrift	Text	Komplette Anschrift für Absender von Briefen
Ort	Text	Ortsbezeichnung für Datumskombinationen («Rheine, den …»)
Homepage	Text	Homepage des Vereins
EMail	Text	Mailkontakt
IBAN	Text	Wieder 42 Zeichen maximal, auch hier mit 4 Zeichen gefolgt von einem Leerzeichen formatiert
BIC	Text	BIC
Bank	Text	Name der Bank
Kontoverwaltung	Ja/Nein	Wenn nicht gewünscht: Nein
Beitragsverwaltung	Ja/Nein	Wenn nicht gewünscht: Nein
Schluesselverwaltung	Ja/Nein	Wenn nicht gewünscht: Nein
Bootstaenderverwaltung	Ja/Nein	Wenn nicht gewünscht: Nein
Abteilungsverwaltung	Ja/Nein	Wenn nicht gewünscht: Nein
Logo	Bild	Logo wird eingelesen und z. B. Für Berichte genutzt.

Zur Absicherung des Feldes "ID", damit wirklich nur ein Datensatz gespeichert werden kann, wird unter **Extras → SQL** eingegeben:

```
001 ALTER TABLE "tbl_Einstellungen"
002 ADD CONSTRAINT "Nur ein Datensatz"
003 CHECK ("ID" = TRUE)
```

Hierdurch wird das Feld "ID" auf die Eingabe **TRUE** (ausgewählt). Bei Fehlversuchen wird ausgegeben:
«Check constraint violation Nur ein Datensatz table: tbl_Einstellungen…»

tbl_Filter

	ID	mitID	Formular	WahlDat	Bericht	StartDat	EndDat	Austritte
	1	3	frm_Schluessel	31.01.22	rpt_Mitglieder_Au			☐

Unterschrift1ID	Unterschrift2ID	Statistik_Stichtag	Beitrag_Stichtag
1	2	01.06.22	01.11.22

Datenziel	
Abfrage:	*qry_Filter, qry_Filter_Austritt, qry_Filter_Gruppe, qry_Filter_Mitglied, qry_Filter_Vorstand*
Ansicht:	*viw_Filter_Mitglieder_komplett_AusEin, viw_Gruppe, viw_Mitglieder_komplett, viw_Statistik_Alter_Detail*
Bericht:	keine direkt
Makro:	*FilterIDSet, FormStart*

Feldname	***Feldtyp***	***Beschreibung***
ID	Integer	Primärschlüssel, Entspricht der VerbindungsID zur Datenbank. Damit kann bei Serverdatenbanken auch mit dieser Tabelle gefiltert werden. Schlüsselwert wird beim Aufruf der Datei über Makro geschrieben.
mitID	Integer	Schlüsselwert von "tbl_Mitglied"
Formular	Text	Formularname (zur Anzeige des aktuellen Formulars in einem Listenfeld)
WahlDat	Datum	Wahldatum für den Vorstand
Bericht	Text	Berichtsname (zur Anzeige des zu startenden Berichtes in einem Listenfeld)
StartDat	Datum	Startdatum z. B. Für die Berichtausgabe
EndDat	Datum	Enddatum für den Zeitraum beginnend mit "StartDat"
Austritte	Ja/Nein	Bericht: Es existiert ein Bericht, der entweder alle Austritte oder alle Beitritte anzeigt. Sollen die Austritte angezeigt werden, so wird das über 'Ja' hier geregelt.
Unterschrift1ID	Integer	Erste Unterschrift z. B. unter Urkunden. Hier wird der Primärschlüssel aus "tbl_VorstandPosten" gespeichert
Unterschrift2ID	Integer	Zweite Unterschrift, ebenfalls über den Primärschlüssel "tbl_VorstandPosten" zu erreichen
Statistik_Stichtag	Datum	Zu diesem Datum soll die Statistik ausgegeben werden
Beitrag_Stichtag	Datum	Zu diesem Datum soll der Beitrag erhoben werden.

Ansichten

viw_Abteilungen_Mitglied

mitID	Abteilungen
3	Wanderabteilung
4	Frauenabteilung, Wanderabteilung
5	Jugend- und Wildwasserabtlg., Rennabteilung
6	Jugend- und Wildwasserabtlg., Rennabteilung
8	Rennabteilung, Drachenbootabteilung
10	Wanderabteilung, Drachenbootabteilung

Datenquelle

Tabelle: *tbl_Abteilung, tbl_Mitglied_Abteilung*

Datenziel

Ansicht: *viw_Anschrift, viw_Filter_Mitglieder_komplett_AusEin, viw_Mitglieder_komplett*

```
001 SELECT DISTINCT "c"."mitID",
002    ( SELECT "Abteilung"
003    FROM ( SELECT "a"."mitID", "a"."abtID",
004       ( SELECT COUNT( "mitID" ) FROM "tbl_Mitglied_Abteilung"
             WHERE "mitID" = "a"."mitID" AND "abtID" <= "a"."abtID" ) AS "Nr"
005       FROM "tbl_Mitglied_Abteilung" AS "a" ) AS "b", "tbl_Abteilung"
006    WHERE "tbl_Abteilung"."ID" = "b"."abtID" AND "b"."Nr" = 1 AND "mitID" =
          "c"."mitID" ) ||
007    COALESCE ( ', ' || ( SELECT "Abteilung"
008    FROM ( SELECT "a"."mitID", "a"."abtID",
009       ( SELECT COUNT( "mitID" ) FROM "tbl_Mitglied_Abteilung"
             WHERE "mitID" = "a"."mitID" AND "abtID" <= "a"."abtID" ) AS "Nr"
010       FROM "tbl_Mitglied_Abteilung" AS "a" ) AS "b", "tbl_Abteilung"
011    WHERE "tbl_Abteilung"."ID" = "b"."abtID" AND "b"."Nr" = 2 AND "mitID" =
          "c"."mitID" ), '' ) ||

    ...

022    COALESCE ( ', ' || ( SELECT "Abteilung"
023    FROM ( SELECT "a"."mitID", "a"."abtID",
024       ( SELECT COUNT( "mitID" ) FROM "tbl_Mitglied_Abteilung"
             WHERE "mitID" = "a"."mitID" AND "abtID" <= "a"."abtID" ) AS "Nr"
025       FROM "tbl_Mitglied_Abteilung" AS "a" ) AS "b", "tbl_Abteilung"
026    WHERE "tbl_Abteilung"."ID" = "b"."abtID" AND "b"."Nr" = 5 AND "mitID" =
          "c"."mitID" ), '' )
027 AS "Abteilungen"
028 FROM "tbl_Mitglied_Abteilung" AS "c"
```

Mit dieser Ansicht werden dem Mitglied in einem Feld alle Abteilungen zugewiesen, die es ausgewählt hat. Eine derartig aufwendige Abfrage ist nur bei der internen HSQLDB notwendig. Die interne FIREBIRD Datenbank hat dafür die Funktion **`LIST()`**, MYSQL/MARIADB die Funktion **`GROUP_CONCAT()`**, POSTGRESQL die Funktion **`STRING_AGG()`** usw.

Die Unterabfrage von Zeile 2 bis Zeile 6 wiederholt sich hier nahezu identisch fünf mal. Die einzigen Unterschiede sind die im Code mit rot hervorgehobenen Zahlen sowie ab der ersten Wiederholung die Verknüpfung über **`COALESCE( ', '|| … , '' )`**.

In der am weitesten innen liegenden Abfrage von Zeile 4 wird der Abteilung, die einem einzelnen Mitglied zugeordnet wurde, eine laufende "Nr" zugewiesen. Dadurch kann aus der Abfrage anschließend die Abteilung mit der "Nr" = 1 herausgefiltert werden.
Zeile 3 und 5 umschließen diese Abfrage und geben zuerst einmal eine "mitID", eine "abtID" und eine "Nr" aus.
Zeile 2 und 6 dienen dann dazu, diese Unterabfrage mit der "tbl_Abteilung" zu kombinieren und die Abteilung auszulesen, die in der Unterabfrage die "Nr" = 1 zugewiesen bekommen hat.

Über die folgenden gleichlautenden Abfragen werden die Abteilungen mit den Nummern 2 bis 5 zugeordnet. Es können hier also höchstens 5 Abteilungen hintereinander, durch ein Komma getrennt, in das Feld "Abteilungen" geschrieben werden. In der Praxis dürfte das aber völlig genügen.

viw_Adresse_Num_Namen

Geschlecht	Vorname	Nachname	gruID	adrID	GruppenNr	GruppenAdressNr	NachnAdressNr
m	Karl	Müller	3	3	1	1	1
w	Maria	Müller-Mehl	3	3	2	2	2
w	Isabel	Müller	3	3	3	3	1
m	Janis	Müller	3	3	4	4	1
w	Jolantha	Schulze	3	7	5	1	1
m	Gisbert	Querulant	8	8	1	1	1
m	Heinz-Herbert	Schöngeist	9	9	1	1	1
w	Yvonne-Chantal	Power	10	10	1	1	1
m	Helmut-Wilhelm	Oldie	11		1	0	0
w	Annemarie	Oldie	12		1	0	0
w	Irina	Oldie	13		1	0	0

Datenquelle
Tabelle: *tbl_Mitglied, tbl_Adresse*

Datenziel
Ansicht: *viw_Anschrift*

```
001 SELECT "c"."Geschlecht",
002 "c"."Vorname",
003 "c"."Nachname",
004 "c"."gruID",
005 "c"."adrID",
006 ( SELECT COUNT( "ID" ) FROM "tbl_Mitglied" WHERE "ID" <= "c"."ID" AND
    COALESCE ( "Gruppe_mitID", "ID" ) = "c"."gruID" ) AS "GruppenNr",
007 ( SELECT COUNT( "ID" ) FROM "tbl_Mitglied" WHERE "ID" <= "c"."ID" AND
    ( COALESCE ( "Gruppe_mitID", "ID" ) = "c"."adrID" OR "ID" =
    "c"."adrID" ) ) AS "GruppenAdressNr",
008 ( SELECT COUNT( "gruID" )
       FROM ( SELECT DISTINCT "Nachname",
       CASE WHEN "tbl_Adresse"."mitID" = "tbl_Mitglied"."ID" THEN
       "tbl_Mitglied"."ID" ELSE "tbl_Mitglied"."Gruppe_mitID" END AS "adrID"
       FROM "tbl_Mitglied" LEFT JOIN "tbl_Adresse" ON "tbl_Mitglied"."ID" =
       "tbl_Adresse"."mitID" )
       WHERE "Nachname" <= "c"."Nachname" AND "adrID" = "c"."adrID" )
    AS "NachnAdressNr"
```

```
009 FROM ( SELECT "a"."ID",
    "a"."Geschlecht",
    "a"."Vorname",
    "a"."Nachname",
    COALESCE ( "Gruppe_mitID", "ID" ) AS "gruID",
    CASE WHEN "b"."mitID" = "a"."ID" THEN "a"."ID" ELSE "a"."Gruppe_mitID" END
    AS "adrID"
    FROM "tbl_Mitglied" AS "a"
    LEFT JOIN "tbl_Adresse" AS "b" ON "a"."ID" = "b"."mitID" ) AS "c"
```

In Zeile 9 werden die ersten vier Felder direkt aus der Tabelle "tbl_Mitglied" ausgelesen. Das vierte Feld gibt lediglich dann statt des Feldes "Gruppe_mitID" das Feld "ID" wieder, wenn das Feld "Gruppe_mitID" leer ist. Dies ist bei allen Mitgliedern der Fall, die nicht einem vorher eingegebenem Mitglied zugeordnet werden konnten. Neben der so ermittelten Gruppenzugehörigkeit kann es auch in einer Gruppe unterschiedliche Adressen geben. Deswegen wird eine separate "adrID" ermittelt, die dann anders ist als die "gruID", wenn "mitID" in der Tabelle "tbl_Adresse" gleich der "ID" in der Tabelle "tbl_Mitglied" ist, obwohl dieses Mitglied unter "Gruppe_mitID" (über das Konto) einer Gruppe zugeordnet wurde. Im Screenshot ist das beim 5. Datensatz ('Jolantha Schulze') zu sehen.

Damit die Mitglieder einer Adress-Gruppe zusammen im Adressfeld einer Adresse erscheinen können, ist bei der internen HSQLDB ein Umweg über die Durchnummerierung aller Mitglieder der Gruppe notwendig. Die GruppenNr gibt hier die Durchnummerierung nach der Gruppe wieder, die unterschiedliche Adressen nicht berücksichtigt. Die "GruppenAdressNr" ist die Nummerierung, die die gleiche Adresse einer Gruppe betrifft. Damit werden in der Adresse die Vornamen zusammengefasst. Auch bei den Nachnamen kann es vorkommen, dass zu einer Gruppe verschiedene Nachnamen gehören. Der Screenshot zeigt so einen Fall für die Gruppe "gruID" = '3'. Auch die unterschiedlichen Nachnamen müssen für jede Gruppe durchnummeriert werden. Dies geschieht im Feld "NachnAdressNr". Bei der Nutzung anderer Datenbanken wie z. B. der internen FIREBIRD Datenbank ist dieser Klimmzug mit der Durchnummerierung nicht notwendig. Dort erledigt die Gruppierung der Vornamen und Nachnamen die Funktion **`LIST()`** direkt.

viw_Anschrift

adrID	Anrede	Vornamen	Nachnamen	Straße_Nr	Postleitzahl	Ort	Nichtdruck
3		Karl, Maria, Isabel und Janis	Müller und Müller-Mehl	Unter dem Ba	98701	Bergdo	☐
10	Frau	Yvonne-Chantal	Power	Industriegebie	98701	Bergdo	☐
8	Herrn	Gisbert	Querulant	An dem Bach	02431	Plattto	☐
9	Herrn	Heinz-Herbert	Schöngeist	Kirchweg 13	42430	Heidela	☐
7	Frau	Jolantha	Schulze	Konsumrennb	02431	Plattto	☐

Datenquelle	
Tabelle:	*tbl_Adresse*, *tbl_Ort*
Ansicht:	*viw_Adresse_Num_Namen*

Datenziel	
Abfrage:	*qry_Anschriften_2spaltig, qry_Anschriften_3spaltig*
Bericht:	*Ergebnisliste*

```
001 SELECT DISTINCT "a"."adrID",
002 CASE WHEN (SELECT MAX("GruppenAdressNr") FROM "viw_Adresse_Num_Namen"
       WHERE "adrID" = "a"."adrID") = 1 THEN
       ( SELECT CASE WHEN "Geschlecht"='m' THEN 'Herrn' WHEN "Geschlecht"='w'
          THEN 'Frau' ELSE '' END FROM "tbl_Mitglied" WHERE "ID" =
```

```
          "a"."adrID")
        ELSE '' END AS "Anrede",
003 ( SELECT "Vorname" FROM "viw_Adresse_Num_Namen"
        WHERE "adrID" = "a"."adrID" AND "GruppenAdressNr" = 1 ) ||
004 COALESCE( CASE WHEN (SELECT MAX("GruppenAdressNr") FROM
        "viw_Adresse_Num_Namen" WHERE "adrID" = "a"."adrID") = 2 THEN ' und '
        ELSE ', ' END||
005     ( SELECT "Vorname" FROM "viw_Adresse_Num_Namen"
        WHERE "adrID" = "a"."adrID" AND "GruppenAdressNr" = 2 ), '' ) ||

    ...

016 COALESCE( CASE WHEN (SELECT MAX("GruppenAdressNr") FROM
        "viw_Adresse_Num_Namen" WHERE "adrID" = "a"."adrID") = 8 THEN ' und '
        ELSE ', ' END||
017     ( SELECT "Vorname" FROM "viw_Adresse_Num_Namen"
        WHERE "adrID" = "a"."adrID" AND "GruppenAdressNr" = 8 ), '' )
    AS "Vornamen",
018 ( SELECT DISTINCT "Nachname" FROM "viw_Adresse_Num_Namen"
        WHERE "adrID" = "a"."adrID" AND "NachnAdressNr" = 1 ) ||
019 COALESCE( CASE WHEN (SELECT MAX("NachnAdressNr") FROM
        "viw_Adresse_Num_Namen" WHERE "adrID" = "a"."adrID") = 2 THEN ' und '
        ELSE ', ' END||
020     ( SELECT DISTINCT "Nachname" FROM "viw_Adresse_Num_Namen"
        WHERE "adrID" = "a"."adrID" AND "NachnAdressNr" = 2 ), '' ) ||

    ...

025 COALESCE( CASE WHEN (SELECT MAX("NachnAdressNr") FROM
        "viw_Adresse_Num_Namen" WHERE "adrID" = "a"."adrID") = 5 THEN ' und '
        ELSE ', ' END||
026     ( SELECT DISTINCT "Nachname" FROM "viw_Adresse_Num_Namen"
        WHERE "adrID" = "a"."adrID" AND "NachnAdressNr" = 5 ), '' )
027 AS "Nachnamen",
028 "tbl_Adresse"."Straße_Nr",
029 "tbl_Ort"."Postleitzahl",
030 "tbl_Ort"."Ort",
031 "tbl_Adresse"."Nichtdruck"
032 FROM "viw_Adresse_Num_Namen" AS "a"
033 LEFT JOIN "tbl_Adresse" ON "a"."adrID" = "tbl_Adresse"."mitID"
034 LEFT JOIN "tbl_Ort" ON "tbl_Adresse"."ortID" = "tbl_Ort"."ID"
035 WHERE ("Nichtdruck" = False OR "Nichtdruck" IS NULL)
    AND "adrID" IS NOT NULL
036 ORDER BY "Nachnamen" ASC
```

Diese Ansicht baut direkt auf der Ansicht *viw_Adresse_Num_Namen* auf. Sie ist wieder nur für die interne HSQLDB so umfangreich. Bei der internen FIREBIRD würde die Zusammenstellung der Gruppierung wesentlich komfortabler durch **`LIST()`** ablaufen.

Nach dem Auslesen der "adrID" aus "viw_Adresse_Num_Namen" wird anhand der Geschlechtsangaben und der Anzahl der Personen der Gruppe eine Anrede erstellt (Zeile 2). Nur wenn nur eine Person in der Gruppe ist wird nachgesehen, welches Geschlecht der Person zugeordnet wurde. Für männliche Personen wird dann 'Herrn', für weibliche Personen 'Frau' gewählt. Ansonsten bleibt die "Anrede" leer.

Ab Zeile 3 wird zuerst der Vorname für die Person heraus gesucht, die innerhalb der Gruppe die "GruppenAdressNr" = '1' zugewiesen bekommen hat. Anschließend wird für jede weiter "GruppenAdressNr" nachgeschaut, ob zu der "GruppenAdressNr" auch ein Datensatz existiert. Existiert kein Datensatz, so wird einfach leerer Text angehängt. Existiert ein Datensatz, so wird nachgesehen, ob die "GruppenAdressNr" gleichzeitig die höchste GruppenAdressnummer **`Max("GruppenAdressNr")`** ist. Ist dies der Fall, so wird der letzte Vorname mit der Bindung **'**

und ' an den vorherigen Namen angehängt. Existieren noch weitere Einträge, so wird der Vorname mit **', '** angehängt.

Diese Prozedur wird in dem obigen Beispiel bis zum 8. Vornamen durchgeführt. Der Code für die Vornamen 3 bis 7 wurde hier nicht noch aufgeführt.

Das Verfahren für den Nachnamen ab Zeile 18 verläuft identisch zu dem des Vornamens. Hier wird aber die Prozedur lediglich bis zum 5. unterschiedlichen Nachnamen in der Gruppe durchgeführt. In der Regel dürften bei Gruppen ja deutlich weniger unterschiedliche Nachnamen als unterschiedliche Vornamen zu finden sein.

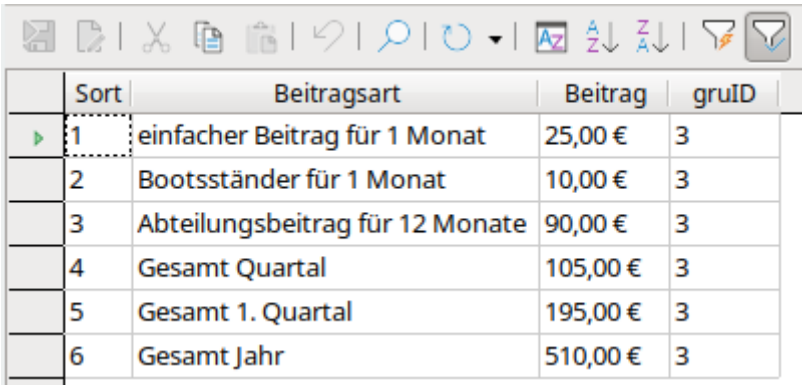
Bei den Nachnamen wird extra mit **DISTINCT** vermieden, dass anschließend gleiche Namen mehrmals auftauchen.

Anschließend werden noch die Straße mit Nummer, die Postleitzahl und der Ort aus den entsprechenden Tabellen zusammengesucht (Zeile 28 bis 30). Beim Nichtdruck wird darauf geachtet, dass er nicht **NULL** ist sondern als nicht angewählt, also **False** erscheint, wenn er nicht tatsächlich ausgewählt wurde. «Nichtdruck» bedeutet hier, dass an die Adresse nichts verschickt wird. Es kommt schließlich immer wieder vor, dass Post nach Umzügen unzustellbar zurück kommt. Da kann dann das Porto auch gleich gespart werden.

Die "tbl_Adresse" und die "tbl_Ort" sind beide über einen **LEFT JOIN** mit der "viw_Adresse_Gruppe" verknüpft. Dadurch werden auf jeden Fall alle Datensätze aus "viw_Adresse_Gruppe" übernommen, auch wenn sie, wie aus dem entsprechenden Screenshot ersichtlich, nicht unbedingt eine Adresse angegeben haben.

viw_Beitrag

Sort	Beitragsart	Beitrag	gruID
1	einfacher Beitrag für 1 Monat	10,00 €	8
1	einfacher Beitrag für 1 Monat	10,00 €	9
1	einfacher Beitrag für 1 Monat	10,00 €	10
1	einfacher Beitrag für 1 Monat	10,00 €	11
1	einfacher Beitrag für 1 Monat	10,00 €	12
1	einfacher Beitrag für 1 Monat	10,00 €	13
1	einfacher Beitrag für 1 Monat	25,00 €	3
2	Bootsständer für 1 Monat	2,00 €	8



Sort	Beitragsart	Beitrag	gruID
1	einfacher Beitrag für 1 Monat	25,00 €	3
2	Bootsständer für 1 Monat	10,00 €	3
3	Abteilungsbeitrag für 12 Monate	90,00 €	3
4	Gesamt Quartal	105,00 €	3
5	Gesamt 1. Quartal	195,00 €	3
6	Gesamt Jahr	510,00 €	3

Abbildung 5: Beiträge gefiltert für die Gruppe "gruID" = '3'. So erscheint das Ergebnis der Ansicht in Formularen und Berichten.

Datenquelle	
Tabelle:	*tbl_Mitglied, tbl_Beitrag, tbl_Gruppe_Beitrag, tbl_Beitrag_Staffel, tbl_Boot, tbl_BootArt,*

tbl_Abteilung, tbl_Mitglied_Abteilung

Datenziel
Abfrage: *qry_Beitraege*
Formular: *Einzelmessung_System*

Hier sind viele Abfragen mit Hilfe von **UNION** untereinander zusammengefasst. Jede dieser Abfragen ergibt maximal einen Datensatz für eine Gruppe ("gruID"). Jede Abfrage enthält neben der "gruID" auch die Spalte "Sort" zur Sortierung der Ergebnisse in der Übersicht. Jede Abfrage enthält außerdem genau 4 Spalten, die vom Typ des Inhaltes (Text, Dezimalzahl) von Abfrage zu Abfrage übereinstimmen müssen.

Zeile 1: Die Abfrage beginnt mit einer festen Spalte "Sort". Der Wert ist '1', da der Inhalt dieser Abfrage ganz oben in der Liste erscheinen soll. In der folgenden Spalte wird aus verschiedenen Feldern der "tbl_Beitrag" die Information für die "Beitragsart" zusammengesetzt. Die Höhe des Beitrags wird aus einer Summe der Multiplikation von "BeitrHoehe" * "Anzahl" für die jeweilige Gruppe ermittelt.

```
001 /* Mitgliedsbeitrag ohne andere Beitraege */
     SELECT '1' AS "Sort",
    'einfacher Beitrag für ' || "tbl_Beitrag"."BeitrMonate" || CASE WHEN
    "tbl_Beitrag"."BeitrMonate" > 1 THEN ' Monate' ELSE ' Monat' END AS
    "Beitragsart",
    SUM( "tbl_Beitrag"."BeitrHoehe" * "tbl_Gruppe_Beitrag"."Anzahl" ) AS
    "Beitrag",
    "tbl_Gruppe_Beitrag"."mitID" AS "gruID"
    FROM "tbl_Beitrag", "tbl_Gruppe_Beitrag"
    WHERE "tbl_Gruppe_Beitrag"."beiID" = "tbl_Beitrag"."ID"
    GROUP BY "gruID", "Beitragsart"
002 UNION
```

In Zeile 3 wird statt des reinen Mitgliedsbeitrages die Bootsständermiete erfasst. Hier muss auf eine ausführlichere Unterabfrage zugegriffen werden, da die Informationen zu den Booten gleich aus mehreren Tabellen geholt werden müssen, so dass alle Boote erfasst werden, die zu einer Gruppe gehören und dann auch noch der gestaffelte Preis berücksichtigt werden kann.

In der innersten Abfrage wird zuerst einmal die Anzahl der Boote und Bootstypen ermittelt, die den jeweiligen Gruppenmitgliedern zugeordnet werden können. Dann wird in der auf diese Abfrage zugreifenden Abfrage in der Tabelle "tbl_Beitrag_Staffel" nachgeschaut, welcher Beitrag für diese Anzahl an Booten fällig ist.

Da nur jeweils ein Datensatz pro Gruppe existiert (entweder hat die Gruppe einen Canadier oder bis zu drei Kajaks im Bootshaus liegen) muss nicht extra noch einmal das Ergebnis der Abfrage bei der Bootsständermiete summiert und gruppiert werden.

```
003 /* Bootsstaendermiete */
    SELECT '2' AS "Sort",
    'Bootsständer für '||"tbl_Beitrag"."BeitrMonate"|| CASE WHEN
    "tbl_Beitrag"."BeitrMonate" > 1 THEN ' Monate' ELSE ' Monat' END AS
    "Beitragsart",
    COALESCE( "tbl_Beitrag"."BeitrHoehe", "tbl_Beitrag_Staffel"."BeitrHoehe")
    AS "Beitrag",
    "a"."gruID"
    FROM (SELECT COUNT( "tbl_Boot"."ID" ) AS "Anzahl",
       "tbl_BootArt"."beiID", COALESCE ( "tbl_Mitglied"."Gruppe_mitID",
       "tbl_Mitglied"."ID" ) AS "gruID"
       FROM "tbl_Boot", "tbl_BootArt", "tbl_Mitglied"
       WHERE "tbl_Boot"."bo_aID" = "tbl_BootArt"."ID" AND
       "tbl_Boot"."mitID" = "tbl_Mitglied"."ID" GROUP BY
       "tbl_BootArt"."beiID", "gruID") AS "a"
    LEFT JOIN "tbl_Beitrag" ON "a"."beiID" = "tbl_Beitrag"."ID"
```

```
     LEFT JOIN "tbl_Beitrag_Staffel" ON "tbl_Beitrag"."ID" =
     "tbl_Beitrag_Staffel"."beiID" WHERE
     "tbl_Beitrag_Staffel"."Anzahl" = "a"."Anzahl" OR
     "tbl_Beitrag_Staffel"."Anzahl" IS NULL
 004 UNION
```

Beim Abteilungsbeitrag können Beiträge für verschiedene Abteilungen anfallen. Hier muss also im Gegensatz zur Bootsständermiete zuerst der Einzelbeitrag pro Abteilung ausgerechnet werden, um dann in der äußersten Abfrage wieder gruppiert und summiert zu werden.

```
 005 /* Abteilungsbeitrag */
     SELECT '3' AS "Sort",
     'Abteilungsbeitrag für '||"tbl_Beitrag"."BeitrMonate"|| CASE WHEN
     "tbl_Beitrag"."BeitrMonate" > 1 THEN ' Monate' ELSE ' Monat' END AS
     "Beitragsart",
     SUM("b"."BeitrHoehe") AS "Beitrag",
     "b"."gruID"
     FROM (SELECT "a"."Anzahl", "a"."beiID", "a"."gruID",
        COALESCE( "tbl_Beitrag"."BeitrHoehe",
        "tbl_Beitrag_Staffel"."BeitrHoehe") AS "BeitrHoehe"
        FROM (SELECT COUNT( "tbl_Mitglied_Abteilung"."mitID" ) AS "Anzahl",
           "tbl_Abteilung"."beiID", COALESCE ( "tbl_Mitglied"."Gruppe_mitID",
           "tbl_Mitglied"."ID" ) AS "gruID" FROM "tbl_Mitglied_Abteilung",
           "tbl_Abteilung", "tbl_Mitglied" WHERE
           "tbl_Mitglied_Abteilung"."abtID" = "tbl_Abteilung"."ID" AND
           "tbl_Mitglied_Abteilung"."mitID" = "tbl_Mitglied"."ID" AND NOT
           "tbl_Abteilung"."beiID" IS NULL GROUP BY "tbl_Abteilung"."beiID",
           "gruID") AS "a"
        LEFT JOIN "tbl_Beitrag" ON "a"."beiID" = "tbl_Beitrag"."ID"
        LEFT JOIN "tbl_Beitrag_Staffel" ON "tbl_Beitrag"."ID" =
        "tbl_Beitrag_Staffel"."beiID" WHERE "tbl_Beitrag_Staffel". "Anzahl" =
        "a"."Anzahl" OR "tbl_Beitrag_Staffel". "Anzahl" IS NULL) AS "b",
     "tbl_Beitrag"
     WHERE "b"."beiID" = "tbl_Beitrag"."ID"
     GROUP BY "gruID", "Beitragsart"
 006 UNION
```

Beim einfachen Quartalsbeitrag werden nur die Zahlungen berücksichtigt, die innerhalb eines Quartals erfolgen. Dabei handelt es sich um die Mitgliedsbeiträge und die Bootsständermiete.

Die Abfragen für den Mitgliedsbeitrag 8 und die Bootsständermiete 10 sind gleich den Abfragen in Zeile 1 und Zeile 3. Der einzige Unterschied besteht darin, dass der Beitrag durch die Beitragsmonate geteilt wird (hier bei beiden Beträgen ist das eine Division durch 1 – könnte ja auch bereits ein Vierteljahresbeitrag sein ...). Anschließend wird dieser Beitrag für einen Monat für das Quartal mit 3 multipliziert.

Diese beiden Beiträge werden über die Abfrage in Zeile 7 und die Verbindungen mit **UNION** zusammengefasst

```
 007 /* Quartal ohne Ganzjahreszahlungen, Quartal 2-3 je nach Abteilung */
     SELECT '4' AS "Sort",
     'Gesamt Quartal' AS "Beitragsart",
     SUM("Beitrag"),
     "gruID" FROM (
 008 -- Mitgliedsbeitrag
        SELECT SUM( "tbl_Beitrag"."BeitrHoehe" *
        "tbl_Gruppe_Beitrag"."Anzahl" / "tbl_Beitrag"."BeitrMonate" ) * 3 AS
        "Beitrag",
        "tbl_Gruppe_Beitrag"."mitID" AS "gruID"
        FROM "tbl_Beitrag", "tbl_Gruppe_Beitrag"
        WHERE "tbl_Gruppe_Beitrag"."beiID" = "tbl_Beitrag"."ID"
        GROUP BY "gruID"
```

```
009    UNION
010 -- Bootstaendermiete
       SELECT ("b"."BeitrHoehe"/"tbl_Beitrag"."BeitrMonate" ) * 3 AS
       "Beitrag",
       "b"."gruID"
       FROM (SELECT "a"."Anzahl", "a"."beiID", "a"."gruID",
          COALESCE( "tbl_Beitrag"."BeitrHoehe",
          "tbl_Beitrag_Staffel"."BeitrHoehe") AS "BeitrHoehe"
          FROM (SELECT COUNT( "tbl_Boot"."ID" ) AS "Anzahl",
             "tbl_BootArt"."beiID", COALESCE ( "tbl_Mitglied"."Gruppe_mitID",
             "tbl_Mitglied"."ID" ) AS "gruID" FROM "tbl_Boot", "tbl_BootArt",
             "tbl_Mitglied" WHERE "tbl_Boot"."bo_aID" = "tbl_BootArt"."ID" AND
             "tbl_Boot"."mitID" = "tbl_Mitglied"."ID" GROUP BY
             "tbl_BootArt"."beiID", "gruID") AS "a"
          LEFT JOIN "tbl_Beitrag" ON "a"."beiID" = "tbl_Beitrag"."ID"
          LEFT JOIN "tbl_Beitrag_Staffel" ON "tbl_Beitrag"."ID" =
          "tbl_Beitrag_Staffel"."beiID" WHERE "tbl_Beitrag_Staffel". "Anzahl"
          = "a"."Anzahl" OR "tbl_Beitrag_Staffel". "Anzahl" IS NULL) AS "b",
         "tbl_Beitrag"
       WHERE "b"."beiID" = "tbl_Beitrag"."ID"
    ) GROUP BY "gruID"
011 UNION
```

Für das Quartal mit den Ganzjahreszahlungen wird die gleiche Abfrage durchgeführt wie für das einfache Quartal. Es wird lediglich die Ganzjahreszahlung für die Abteilungsbeiträge hinzugefügt.

```
012 /* 1. Quartal, mit Ganzjahreszahlungen */
    SELECT '5' AS "Sort",
    'Gesamt 1. Quartal' AS "Beitragsart",
    SUM("Beitrag"),
    "gruID" FROM (
013 -- Mitgliedsbeitrag
       SELECT SUM( "tbl_Beitrag"."BeitrHoehe" *
       "tbl_Gruppe_Beitrag"."Anzahl" / "tbl_Beitrag"."BeitrMonate" ) * 3 AS
       "Beitrag",
       "tbl_Gruppe_Beitrag"."mitID" AS "gruID"
       FROM "tbl_Beitrag", "tbl_Gruppe_Beitrag"
       WHERE "tbl_Gruppe_Beitrag"."beiID" = "tbl_Beitrag"."ID"
       GROUP BY "gruID"
014    UNION
015 -- Bootstaendermiete
       SELECT ("b"."BeitrHoehe"/"tbl_Beitrag"."BeitrMonate" ) * 3 AS
       "Beitrag",
       "b"."gruID"
       FROM (SELECT "a"."Anzahl", "a"."beiID", "a"."gruID",
          COALESCE( "tbl_Beitrag"."BeitrHoehe",
          "tbl_Beitrag_Staffel"."BeitrHoehe") AS "BeitrHoehe"
          FROM (SELECT COUNT( "tbl_Boot"."ID" ) AS "Anzahl",
             "tbl_BootArt"."beiID", COALESCE ( "tbl_Mitglied"."Gruppe_mitID",
             "tbl_Mitglied"."ID" ) AS "gruID" FROM "tbl_Boot", "tbl_BootArt",
             "tbl_Mitglied" WHERE "tbl_Boot"."bo_aID" = "tbl_BootArt"."ID" AND
             "tbl_Boot"."mitID" = "tbl_Mitglied"."ID" GROUP BY
             "tbl_BootArt"."beiID", "gruID") AS "a"
          LEFT JOIN "tbl_Beitrag" ON "a"."beiID" = "tbl_Beitrag"."ID"
          LEFT JOIN "tbl_Beitrag_Staffel" ON "tbl_Beitrag"."ID" =
          "tbl_Beitrag_Staffel"."beiID"
          WHERE "tbl_Beitrag_Staffel". "Anzahl" = "a"."Anzahl" OR
          "tbl_Beitrag_Staffel". "Anzahl" IS NULL) AS "b",
          "tbl_Beitrag"
       WHERE "b"."beiID" = "tbl_Beitrag"."ID"
```

```
016    UNION
017 -- Abteilungsbeitrag
       SELECT SUM("b"."BeitrHoehe") AS "Beitrag",
       "b"."gruID"
       FROM (SELECT "a"."Anzahl", "a"."beiID", "a"."gruID",
          COALESCE( "tbl_Beitrag"."BeitrHoehe",
          "tbl_Beitrag_Staffel"."BeitrHoehe") AS "BeitrHoehe"
          FROM (SELECT COUNT( "tbl_Mitglied_Abteilung"."mitID" ) AS "Anzahl",
             "tbl_Abteilung"."beiID", COALESCE
             ( "tbl_Mitglied"."Gruppe_mitID", "tbl_Mitglied"."ID" ) AS "gruID"
             FROM "tbl_Mitglied_Abteilung", "tbl_Abteilung", "tbl_Mitglied"
             WHERE "tbl_Mitglied_Abteilung"."abtID" = "tbl_Abteilung"."ID"
             AND "tbl_Mitglied_Abteilung"."mitID" = "tbl_Mitglied"."ID"
             AND NOT "tbl_Abteilung"."beiID" IS NULL
             GROUP BY "tbl_Abteilung"."beiID", "gruID") AS "a"
          LEFT JOIN "tbl_Beitrag" ON "a"."beiID" = "tbl_Beitrag"."ID"
          LEFT JOIN "tbl_Beitrag_Staffel" ON "tbl_Beitrag"."ID" =
          "tbl_Beitrag_Staffel"."beiID"
          WHERE "tbl_Beitrag_Staffel". "Anzahl" = "a"."Anzahl" OR
          "tbl_Beitrag_Staffel". "Anzahl" IS NULL) AS "b",
          "tbl_Beitrag"
       WHERE "b"."beiID" = "tbl_Beitrag"."ID" GROUP BY "gruID"
    ) GROUP BY "gruID"
018 UNION
```

Die Ganzjahreszahlung unterscheidet sich von der Zahlung für das erste Quartal nur dadurch, dass jetzt die Mitgliedsbeiträge und die Bootsständermieten, die für einen Monat berechnet wurden, eben vor dem Zusammenfassen mit 12 multipliziert werden.

```
019 /* Jahreszahlungen gesamt */
    SELECT '6' AS "Sort",
    'Gesamt Jahr' AS "Beitragsart",
    SUM("Beitrag"),
    "gruID" FROM (
020 -- Mitgliedsbeitrag
       SELECT SUM( "tbl_Beitrag"."BeitrHoehe" *
       "tbl_Gruppe_Beitrag"."Anzahl" / "tbl_Beitrag"."BeitrMonate" ) * 12 AS
       "Beitrag",
       "tbl_Gruppe_Beitrag"."mitID" AS "gruID"
       FROM "tbl_Beitrag", "tbl_Gruppe_Beitrag"
       WHERE "tbl_Gruppe_Beitrag"."beiID" = "tbl_Beitrag"."ID"
       GROUP BY "gruID"
021  UNION
022 -- Bootstaendermiete
       SELECT ("b"."BeitrHoehe"/"tbl_Beitrag"."BeitrMonate" ) * 12 AS
       "Beitrag",
       "b"."gruID"
       FROM (SELECT "a"."Anzahl", "a"."beiID", "a"."gruID",
          COALESCE( "tbl_Beitrag"."BeitrHoehe",
          "tbl_Beitrag_Staffel"."BeitrHoehe") AS "BeitrHoehe"
          FROM (SELECT COUNT( "tbl_Boot"."ID" ) AS "Anzahl",
             "tbl_BootArt"."beiID", COALESCE ( "tbl_Mitglied"."Gruppe_mitID",
             "tbl_Mitglied"."ID" ) AS "gruID" FROM "tbl_Boot", "tbl_BootArt",
             "tbl_Mitglied" WHERE "tbl_Boot"."bo_aID" = "tbl_BootArt"."ID" AND
             "tbl_Boot"."mitID" = "tbl_Mitglied"."ID" GROUP BY
             "tbl_BootArt"."beiID", "gruID") AS "a"
          LEFT JOIN "tbl_Beitrag" ON "a"."beiID" = "tbl_Beitrag"."ID"
          LEFT JOIN "tbl_Beitrag_Staffel" ON "tbl_Beitrag"."ID" =
          "tbl_Beitrag_Staffel"."beiID"
          WHERE "tbl_Beitrag_Staffel". "Anzahl" = "a"."Anzahl" OR
          "tbl_Beitrag_Staffel". "Anzahl" IS NULL) AS "b",
```

```
          "tbl_Beitrag"
       WHERE "b"."beiID" = "tbl_Beitrag"."ID"
023  UNION
024 -- Abteilungsbeitrag
       SELECT SUM("b"."BeitrHoehe") AS "Beitrag",
       "b"."gruID"
       FROM (SELECT "a"."Anzahl", "a"."beiID", "a"."gruID",
          COALESCE( "tbl_Beitrag"."BeitrHoehe",
          "tbl_Beitrag_Staffel"."BeitrHoehe") AS "BeitrHoehe"
          FROM (SELECT COUNT( "tbl_Mitglied_Abteilung"."mitID" ) AS "Anzahl",
             "tbl_Abteilung"."beiID", COALESCE
             ( "tbl_Mitglied"."Gruppe_mitID", "tbl_Mitglied"."ID" ) AS "gruID"
             FROM "tbl_Mitglied_Abteilung", "tbl_Abteilung", "tbl_Mitglied"
             WHERE "tbl_Mitglied_Abteilung"."abtID" = "tbl_Abteilung"."ID"
             AND "tbl_Mitglied_Abteilung"."mitID" = "tbl_Mitglied"."ID"
             AND NOT "tbl_Abteilung"."beiID" IS NULL
             GROUP BY "tbl_Abteilung"."beiID", "gruID") AS "a"
          LEFT JOIN "tbl_Beitrag" ON "a"."beiID" = "tbl_Beitrag"."ID"
          LEFT JOIN "tbl_Beitrag_Staffel" ON "tbl_Beitrag"."ID" =
          "tbl_Beitrag_Staffel"."beiID"
          WHERE "tbl_Beitrag_Staffel". "Anzahl" = "a"."Anzahl" OR
          "tbl_Beitrag_Staffel". "Anzahl" IS NULL) AS "b",
          "tbl_Beitrag"
       WHERE "b"."beiID" = "tbl_Beitrag"."ID" GROUP BY "gruID"
    ) GROUP BY "gruID"
```

viw_Beitrag_Gruppe_kalkuliert

Gruppe	Anzahl	beiID	Beitrag
3	1	3	10
3	1	4	15
8	1	3	10
9	1	3	10
10	1	3	10
11	1	3	10
12	1	3	10
13	1	3	10

Datenquelle	
Tabelle:	*tbl_Beitrag*
Ansicht:	*viw_Gruppe*

Datenziel	
Makro:	*SubscriptionSave*

```
001 SELECT "Gruppe", "Anzahl", "beiID", "Anzahl"*(SELECT "BeitrHoehe" FROM
    "tbl_Beitrag" WHERE "ID" = "a"."beiID") AS "Beitrag" FROM
002    (SELECT "Gruppe", CASE WHEN "Anzahl_korr" = 0 THEN 1 ELSE "Anzahl_korr"
       END AS "Anzahl", "beiID_korr" AS "beiID" FROM
003       (SELECT  "Gruppe", "beiID", "Anzahl", "MinAlter", "Beitrag",
          "Anzahl_kor1", CASE WHEN "Anzahl" > "Anzahl_kor1" AND "MinAlter" <
          18 THEN 0 WHEN "Anzahl_kor1" < 0 THEN 0 ELSE "Anzahl_kor1" END AS
          "Anzahl_korr", CASE WHEN "Anzahl" > "Anzahl_kor1" AND
```

```
          "Anzahl_kor1" < 1 THEN (SELECT "ID" FROM "tbl_Beitrag" WHERE
          "BeitrBezug" = 'MitG') ELSE "beiID" END AS "beiID_korr" FROM
004          (SELECT  "b"."Gruppe", "b"."beiID", "b"."Anzahl", "b"."MinAlter",
             "b"."Beitrag", CASE WHEN (SELECT SUM(( SELECT "BeitrHoehe" FROM
             "tbl_Beitrag" WHERE "BeitrBezug" = 'Mit' AND "a"."Alter_Beitrag"
             BETWEEN "MinAlter" AND "MaxAlter" )) FROM "viw_Gruppe" AS "a"
             WHERE "Gruppe" = "b"."Gruppe") > (SELECT "BeitrHoehe" FROM
             "tbl_Beitrag" WHERE "BeitrBezug" = 'MitG') THEN "Anzahl" - 2 ELSE
             "Anzahl" END AS "Anzahl_kor1" FROM
005             (SELECT  "a"."Gruppe", "a"."beiID", "a"."Anzahl",
                "a"."MinAlter", "Anzahl" * (SELECT "BeitrHoehe" FROM
                "tbl_Beitrag" WHERE "ID" = "a"."beiID") AS "Beitrag" FROM
006                (SELECT "Gruppe", "beiID", COUNT("beiID") AS "Anzahl",
                   "MinAlter" FROM
007                   (SELECT "Gruppe", ( SELECT "ID" FROM "tbl_Beitrag" WHERE
                      "BeitrBezug" = 'Mit' AND "a"."Alter_Beitrag" BETWEEN
                      "MinAlter" AND "MaxAlter" ) AS "beiID",
                      ( SELECT "MinAlter" FROM "tbl_Beitrag" WHERE
                      "BeitrBezug" = 'Mit' AND "a"."Alter_Beitrag" BETWEEN
                      "MinAlter" AND "MaxAlter" ) AS "MinAlter" FROM
                      "viw_Gruppe" AS "a"
008                   )
                   GROUP BY "Gruppe", "beiID", "MinAlter"
009                ) AS "a"
010             ) AS "b"
011          )
012       )
013    ) AS "a"
014 GROUP BY "Gruppe", "beiID", "Anzahl"
```

Mit dieser Ansicht soll der Beitrag genau kalkuliert werden. Werden nur Einzelbeiträge erhoben, so ist das Ganze nur eine einfache Addition (Beiträge, die zu dem Alter passen, gruppenweise summieren). Die Beitragstabelle sieht hier aber einen Gruppentarif für Paare und Familien vor, der günstiger ist als z. B. Zwei Erwachsenenbeiträge oder auch ein Erwachsenenbeitrag und ein Kinderbeitrag.

Der Code ist so aufgebaut, dass die äußeren Abfragen immer wieder auf innere Abfragen zugreifen müssen. Die Ursprungsabfrage ist die, die Bei Zeile 7 beginnt und bei Zeile 8 endet. Hier wird aus der "viw_Gruppe" die Gruppenbezeichnung ausgelesen. Außerdem wird über zwei korrelierende Unterabfragen zuerst die "beiID" für alle Beiträge, die direkt Mitglieder betreffen ("BeitrBezug = 'Mit'), danach dann das dazugehörige Mindestalter "MinAlter" ermittelt.

Um diese Abfrage herum liegt von Zeile 6 bis Zeile 9 die nächste Abfrage. Sie dient lediglich dazu, die Anzahl der gleichen Beiträge zu ermitteln. Deswegen wird die Abfrage nach der "Gruppe", "beiID" und "MinAlter" gruppiert. Zu der innersten Abfrage kommt also lediglich das Feld mit der Anzahl hinzu. So sind in der Gruppe '3' an dieser Stelle insgesamt 3 Personen mit der "beiID" = '3', also 18 Jahre und älter.

In der Abfrage, die von Zeile 5 bis Zeile 10 läuft, wird jetzt anhand der vorher ermittelten Anzahl der gleichen Beiträge durch eine Multiplikation mit der passenden "BeitrHoehe" der Beitrag ermittelt, der in der Gruppe jeweils für eine Altersgruppe zu entrichten wäre.

Die jetzt weiter außen liegende Abfrage von Zeile 4 bis Zeile 11 greift auf die innenliegende Abfrage mit dem Alias "b" zu. In dieser Abfrage wird über eine **CASE WHEN … THEN … ELSE … END** Bedingung ermittelt, wie hoch der Beitrag für eine komplette Gruppe bisher wäre. Ist der Beitrag höher als der Beitrag mit dem "BeitrBezug" = 'MitG', so wird von jeder vorher ermittelten Anzahl 2 abgezogen. In dem Beitrag für Gruppen sind 2 Erwachsene und beliebig viele Kinder und Jugendliche enthalten.

Von Zeile 3 bis Zeile 12 wird in der Abfrage auf diese neu ermittelte "Anzahl_kor1" Bezug genommen. Alle Mitglieder, bei der sich vorher eine Gruppenmitgliedschaft ergab, stimmt "Anzahl_kor1" nicht mehr mit "Anzahl" überein. Für unter 18-Jährige wird jetzt die Anzahl

grundsätzlich auf '0' gesetzt, da sie bereits im Gruppenbeitrag enthalten sind. Für sie wird jetzt der Gruppenbeitrag als "beiID_korr" statt der vorherigen "beiID" notiert. Bei Mitgliedern ab 18 wird die korrigierte Anzahl auf 0 gesetzt, sofern sie nicht größer als 0 ist. Dadurch werden maximal 2 erwachsene Mitglieder mit in die Gruppe aufgenommen.

In der vorletzten Abfrage von Zeile 2 bis Zeile 13 wird dann lediglich die vorherige 0-Setzung in soweit korrigiert, als daraus eine '1' für den Gruppenbeitrag gemacht wird.

In der äußersten Abfrage von Zeile 1 bis Zeile 14 wird jetzt mit Hilfe dieser korrigierten Werte der Beitrag endgültig berechnet. Würde die Abfrage nicht zum Schluss gruppiert, so würde bei Familien mit 2 Erwachsenen ein Gruppenbeitrag für die unter 18-jährigen und einer für die Erwachsenen ausgegeben. Durch die Gruppierung sind hier die Datensätze aber gleich, so dass einer der Datensätze unterdrückt wird.

viw_Boote_Mitglied

	mitID	Boote
▸	3	K2: Moin → Garage 1: 0.3, K1: Ems → Garage 1: 1.2
	4	K1: Sturmvögel → Garage 1: 1.3
	8	K2: Autonom → Garage 1: 2.3
	9	C4: Canadier für alle → Garage 1: 2.4
	10	K2: Seeblick → Garage 1: 2.2

Datenquelle
Tabelle: *tbl_Boot, tbl_BootArt, tbl_BootStaender, tbl_BootHalle*

Datenziel
Ansicht: *viw_Filter_Mitglieder_komplett_AusEin, viw_Mitglieder_komplett*

```
001 SELECT DISTINCT "c"."mitID",
002    ( SELECT "tbl_BootArt"."Bootsart" || ': ' || "b"."Bootsname" || ' → '
          || "tbl_BootHalle"."Halle_lang" || ': ' ||
          "tbl_BootStaender"."BootStaend" AS "Boote"
003    FROM ( SELECT "a"."ID", "a"."Bootsname", "a"."bo_aID", "a"."bo_sID",
          "a"."mitID",
004          ( SELECT COUNT( "mitID" ) FROM "tbl_Boot" WHERE "mitID" =
             "a"."mitID" AND "ID" <= "a"."ID" ) AS "Nr"
005          FROM "tbl_Boot" AS "a" ) AS "b", "tbl_BootArt", "tbl_BootStaender",
             "tbl_BootHalle"
006    WHERE "b"."bo_aID" = "tbl_BootArt"."ID" AND "b"."bo_sID" =
          "tbl_BootStaender"."ID" AND "tbl_BootStaender"."bo_hID" =
          "tbl_BootHalle"."ID" AND "mitID" = "c"."mitID" AND "b"."Nr" = 1 ) ||
007    COALESCE ( ', ' || ( SELECT "tbl_BootArt"."Bootsart" || ': ' ||
          "b"."Bootsname" || ' → ' || "tbl_BootHalle"."Halle_lang" || ': ' ||
          "tbl_BootStaender"."BootStaend" AS "Boote"
008    FROM ( SELECT "a"."ID", "a"."Bootsname", "a"."bo_aID", "a"."bo_sID",
          "a"."mitID",
009          ( SELECT COUNT( "mitID" ) FROM "tbl_Boot" WHERE "mitID" =
             "a"."mitID" AND "ID" <= "a"."ID" ) AS "Nr"
010          FROM "tbl_Boot" AS "a" ) AS "b", "tbl_BootArt", "tbl_BootStaender",
             "tbl_BootHalle"
011    WHERE "b"."bo_aID" = "tbl_BootArt"."ID" AND "b"."bo_sID" =
          "tbl_BootStaender"."ID" AND "tbl_BootStaender"."bo_hID" =
          "tbl_BootHalle"."ID" AND "mitID" = "c"."mitID" AND "b"."Nr" = 2 ),
          '' ) ||
```

```
012    COALESCE ( ', ' || ( SELECT "tbl_BootArt"."Bootsart" || ': ' ||
          "b"."Bootsname" || ' → ' || "tbl_BootHalle"."Halle_lang" || ': ' ||
          "tbl_BootStaender"."BootStaend" AS "Boote"
013    FROM ( SELECT "a"."ID", "a"."Bootsname", "a"."bo_aID", "a"."bo_sID",
          "a"."mitID",
014       ( SELECT COUNT( "mitID" ) FROM "tbl_Boot" WHERE "mitID" =
             "a"."mitID" AND "ID" <= "a"."ID" ) AS "Nr"
015       FROM "tbl_Boot" AS "a" ) AS "b", "tbl_BootArt", "tbl_BootStaender",
             "tbl_BootHalle"
016    WHERE "b"."bo_aID" = "tbl_BootArt"."ID" AND "b"."bo_sID" =
          "tbl_BootStaender"."ID" AND "tbl_BootStaender"."bo_hID" =
          "tbl_BootHalle"."ID" AND "mitID" = "c"."mitID" AND "b"."Nr" = 3 ),
          '' )
017 AS "Boote"
018 FROM "tbl_Boot" AS "c"
```

Mit dieser Ansicht werden dem Mitglied in einem Feld alle Boote zugewiesen, die es im Bootshaus gelagert hat. Eine derartig aufwendige Abfrage ist nur bei der internen HSQLDB notwendig. Die interne FIREBIRD Datenbank hat dafür die Funktion **`LIST()`**.

Die Unterabfrage von Zeile 2 bis Zeile 6 wiederholt sich hier nahezu identisch drei mal. Die einzigen Unterschiede sind die im Code mit rot hervorgehobenen Zahlen sowie ab der ersten Wiederholung die Verknüpfung über **`COALESCE( ', '|| … , '' )`**.
In der am weitesten innen liegenden Abfrage von Zeile 4 wird dem Boot, das einem einzelnen Mitglied gehört, eine laufende "Nr" zugewiesen. Dadurch kann aus der Abfrage anschließend das Boot mit der "Nr" = 1 herausgefiltert werden.
Zeile 3 und 5 umschließen diese Abfrage und geben zuerst einmal eine "ID" des Bootes, den "Bootsname", eine "bo_aID" (für die Bootsart, hier 'K1', 'K2' usw.), eine "bo_sID" (für den Bootsständer, zusammengesetzt aus der Halle und der Ständernummer), eine "mitID" und eine "Nr" aus.
Zeile 2 und 6 dienen dann dazu, diese Unterabfrage mit den Tabellen "tbl_BootArt", "tbl_BootHalle", "tbl_BootStaneder" und "tbl_Boot" zu kombinieren und das Boot mit Lagerort auszulesen, das in der Unterabfrage die "Nr" = 1 zugewiesen bekommen hat.

Über die folgenden gleichlautenden Abfragen werden die Boote mit den Nummern 2 und 3 zugeordnet. Drei Boote sind die Maximalzahl, die jede Gruppe in den bootshallen lagern darf.

viw_Filter_Mitglieder_komplett_AusEin

gruID	Kontoinhaber	IBAN	BIC	Bank	Straße_Nr	PLZ_Ort	Telefon_fest	Nachname	Vorname
3	Großkopf,	DE47 110	WELAD	Stadtspa	Unter dem	98701 Bergdorf	05971/424	Müller-Mehl	Maria
3	Großkopf,	DE47 110	WELAD	Stadtspa	Unter dem	98701 Bergdorf	05971/424	Müller	Isabel
3	Großkopf,	DE47 110	WELAD	Stadtspa	Unter dem	98701 Bergdorf	05971/424	Müller	Janis
3	Großkopf,	DE47 110	WELAD	Stadtspa	Konsumren	02431 Platttown		Schulze	Jolantha
8	Querulant,	DE23 456	GENOD	Volksban	An dem	02431 Platttown	09090/123	Querulant	Gisbert

GebDat	Schwimmer	Abteilungen	Boote	AusDat	AufDat	FilterDat	StartDat	EndDat	Austritte
13.11.69	✅	Frauenabteilung	K1: Sturm	01.09.22	01.01.22	01.01.22	01.01.22	31.12.22	☐
17.07.07	✅	Jugend- und Wil			01.01.22	01.01.22	01.01.22	31.12.22	☐
23.10.09	✅	Jugend- und Wil			01.01.22	01.01.22	01.01.22	31.12.22	☐
17.09.01	✅				03.02.22	03.02.22	01.01.22	31.12.22	☐
13.11.03	⛔		K2: Auton		07.03.22	07.03.22	01.01.22	31.12.22	☐

Datenquelle	
Tabelle:	*tbl_Mitglied, tbl_Adresse, tbl_Ort, tbl_Konto, tbl_Bank, tbl_Filter*
Ansicht:	*Ansicht_Datum, viw_Boote_Mitglied, viw_ServVar*

Datenziel	
Bericht:	*rpt_Mitglieder_AusEin, rpt_Mitglieder_AusEin_Konto*

Diese Ansicht dient dazu, Austritte und Eintritte in den Verein über einen Bericht dokumentieren zu können. In dem obigen Screenshot sind alle Eintritte aus dem Jahr 2022 dokumentiert. In der Tabelle "tbl_Filter"."StatDat" ist der 1.1.22 als Startdatum eingestellt.[11]

Besondere Probleme können bei so einer Ansicht auftauchen, wenn, wie beim 4. Datensatz (Schulze, Jolantha), eine Person zwar zu einer Gruppe (über das Konto) gehört, aber eine andere Adresse hat (Auszug zur Ausbildung/zum Studium ...). Ist dann auch noch das Festnetztelefon bei der einen Adresse vorhanden und bei der anderen Adresse nicht, dann lassen sich einige einfache Lösungsmöglichkeiten nicht mehr nutzen.

Die Abfrage gestaltet sich sehr umfangreich, weil neben Konto und Adresse auch die Abteilungen und Boote aufgelistet werden. Für eine solche Auflistung steht für die interne HSQLDB keine Funktion zur Verfügung. Mit der internen FIREBIRD Datenbank wäre der Code über die Funktion **LIST()** entsprechend kürzer.

```
001 SELECT COALESCE ( "a"."Gruppe_mitID", "a"."ID" ) AS "gruID",
002 COALESCE ( "tbl_Konto"."Nachname" || COALESCE ( ', ' ||
    "tbl_Konto"."Vorname", '' ), "a"."Nachname" || COALESCE ( ', ' ||
    "a"."Vorname", '' ) ) AS "Kontoinhaber",
```

Immer wieder tauchen Abfragen mit **COALESCE** auf. Wenn der erste Begriff leer ist, dann nimm den zweiten. Ist der auch leer, dann den dritten usw. Ist in der "tbl_Konto" ein "Nachname" angegeben, dann wird dieser genommen. Der Name aus "tbl_Mitglied" (hier mit dem Alias "a") spielt dann keine Rolle mehr. Gibt es einen Vornamen, so wird der mit einem führenden Komma an den Nachnamen angehängt. Diese Konstruktion ist notwendig, da bei einem leeren Feld "Vorname" sonst auch die Verbindung von "Nachname" und "Vorname" über **||** leer wäre.

```
003 "tbl_Konto"."IBAN",
004 "tbl_Bank"."BIC",
005 "tbl_Bank"."Bank",
006 COALESCE("b"."Straße_Nr", (SELECT "Straße_Nr" FROM "tbl_Adresse" WHERE
    "mitID" = COALESCE ( "a"."Gruppe_mitID", "a"."ID" ))) AS "Straße_Nr",
007 COALESCE("tbl_Ort"."Postleitzahl"||' '||"tbl_Ort"."Ort", (SELECT
    "tbl_Ort"."Postleitzahl"||' '||"tbl_Ort"."Ort" FROM "tbl_Ort",
    "tbl_Adresse" WHERE "tbl_Adresse"."ortID" = "tbl_Ort"."ID" AND
    "tbl_Adresse"."mitID" = COALESCE ( "a"."Gruppe_mitID", "a"."ID" ))) AS
    "PLZ_Ort",
008 COALESCE("b"."Telefon_fest", (SELECT "Telefon_fest" FROM "tbl_Adresse"
    WHERE "mitID" = CASE WHEN "b"."mitID" = "a"."ID" THEN "a"."ID" ELSE
    "a"."Gruppe_mitID" END)) AS "Telefon_fest",
```

Für die Zeilen 6 und 8 ist wichtig, ob in der Abfrage mit dem Alias "b" jeweils ein Wert steht. Ist dies der Fall, dann wird dieser Wert auch genommen. Ansonsten wird die Adressbindung der Gruppe benutzt. Die Abfrage mit dem Alias "b" (weiter unten) ermittelt also eine Adresse, die für ein Gruppenmitglied gilt, das nicht (mehr) mit den anderen zusammen wohnt.

```
009 "a"."Nachname",
010 "a"."Vorname",
011 "a"."GebDat",
012 CASE WHEN "a"."Schwimmer" = TRUE THEN '  ' ELSE '  ' END AS "Schwimmer",
013 "viw_Abteilungen_Mitglied"."Abteilungen",
```

11 Im Feld "Schwimmer" wurde statt des Markierfeldes die Darstellung mit UTF8-Symbolen genutzt. Die sind dann auch im Bericht zu sehen.

```
014 "viw_Boote_Mitglied"."Boote",
015 "a"."AusDat",
016 "a"."AufDat",
017 CASE WHEN (SELECT "Austritte" FROM "tbl_Filter" WHERE "ID" = COALESCE
    ( ( SELECT "ID" FROM "viw_ServVar" ), '1' )) = TRUE THEN "a"."AusDat" ELSE
    "a"."AufDat" END AS "FilterDat",
018 (SELECT COALESCE("StartDat", CAST(EXTRACT(YEAR FROM CURRENT_DATE)||'-01-
    01' AS DATE)) FROM "tbl_Filter" WHERE "ID" = COALESCE ( ( SELECT "ID" FROM
    "viw_ServVar" ), '1' )) AS "StartDat",
019 (SELECT COALESCE("EndDat", CAST(EXTRACT(YEAR FROM CURRENT_DATE)||'-12-31'
    AS DATE)) FROM "tbl_Filter" WHERE "ID" = COALESCE ( ( SELECT "ID" FROM
    "viw_ServVar" ), '1' )) AS "EndDat",
020 (SELECT "Austritte" FROM "tbl_Filter" WHERE "ID" = COALESCE ( ( SELECT
    "ID" FROM "viw_ServVar" ), '1' )) AS "Austritte"
```

Die Einbindung der Werte aus "tbl_Filter" in den Zeilen 17 bis 20 ist notwendig, damit der Bericht bei der Ausführung den Zeitraum der Entsprechenden Daten anzeigen kann, auch in der Überschrift erwähnt, ob es sich um Beitritte oder Austritte handelt usw.

```
021 FROM "tbl_Mitglied" AS "a"
022 LEFT JOIN "tbl_Adresse" AS "b" ON "b"."mitID" = "a"."ID"
023 LEFT JOIN "tbl_Ort" ON "b"."ortID" = "tbl_Ort"."ID"
024 LEFT JOIN "tbl_Konto" ON ( "tbl_Konto"."mitID" = "a"."Gruppe_mitID" OR
    "tbl_Konto"."mitID" = "a"."ID" )
025 LEFT JOIN "tbl_Bank" ON "tbl_Konto"."banID" = "tbl_Bank"."ID"
026 LEFT JOIN  "viw_Abteilungen_Mitglied" ON
    "viw_Abteilungen_Mitglied"."mitID" = "a"."ID"
027 LEFT JOIN "viw_Boote_Mitglied" ON "viw_Boote_Mitglied"."mitID" = "a"."ID"
028 WHERE "FilterDat" BETWEEN "StartDat" AND "EndDat"
```

Zum Schluss wird in Zeile 28 auf die Zeilen 17, 18 und 19 Bezug genommen. In "FilterDat" ist je nach Art der Filterung entweder das Aufnahmedatum oder das Austrittsdatum des Mitglieds gefragt. Und dieses Datum soll in dem Bereich liegen, der ebenfalls durch die Filterung, oder bei leeren Feldern durch Bezug auf den Start des aktuellen Jahres ("StartDat") bzw. das Ende des aktuellen Jahres ("EndDat") begrenzt ist.

viw_Gruppe

	ID	Nachname	Vorname	GebDat	Gruppe	Geschlecht	AufDat	Schwimmer
▸	3	Müller	Karl	27.02.64	3	m	04.04.21	☑
	4	Müller-Mehl	Maria	13.11.69	3	w	01.01.22	☑
	5	Müller	Isabel	17.07.07	3	w	01.01.22	☑
	6	Müller	Janis	23.10.09	3	m	01.01.22	☑
	7	Schulze	Jolantha	17.09.01	3	w	03.02.22	☑
	8	Querulant	Gisbert	13.11.03	8	m	07.03.22	☐
	9	Schöngeist	Heinz-Herbert	12.07.87	9	m	05.01.21	☑

Telefon_mobil	Photo	AusDat	Alter_Jahr	Alter_momentan	Alter_Beitrag
0123456789	DB_Bilder/Gr		58	58	58
		01.09.22	53	52	52
			15	15	15
0987654321			13	12	13
			21	20	21
			19	18	18
			35	35	35

Datenquelle	
Tabelle:	*tbl_Mitglied, tbl_Filter*
Ansicht:	*viw_ServVar*

Datenziel	
Ansicht:	*viw_Beitrag_Gruppe_kalkuliert*
Abfrage:	*qry_Filter_Gruppe*
Formular:	*Einzelmessung_System*
Makro:	*Preferences, SubscriptionSave*

```
001 SELECT "ID",
002 "Nachname",
003 "Vorname",
004 "GebDat",
005 COALESCE("Gruppe_mitID","ID") AS "Gruppe",
006 "Geschlecht",
007 "AufDat",
008 "Schwimmer",
009 "Telefon_mobil",
010 "Photo",
011 "AusDat",
012 DATEDIFF('yy',"GebDat",CURRENT_DATE) AS "Alter_Jahr",
013 CASE WHEN (MONTH("GebDat") > MONTH(CURRENT_DATE)) OR ((MONTH("GebDat") =
       MONTH(CURRENT_DATE)) AND (DAY("GebDat") > DAY(CURRENT_DATE)))
    THEN DATEDIFF('yy',"GebDat",CURRENT_DATE)-1
    ELSE DATEDIFF('yy',"GebDat",CURRENT_DATE)
    END
    AS "Alter_momentan",
014 CASE WHEN (MONTH("GebDat") >
       MONTH(COALESCE("Beitrag_Stichtag",CURRENT_DATE))) OR ((MONTH("GebDat")
       = MONTH(COALESCE("Beitrag_Stichtag",CURRENT_DATE))) AND (DAY("GebDat")
       > DAY(COALESCE("Beitrag_Stichtag",CURRENT_DATE))))
    THEN DATEDIFF('yy',"GebDat",COALESCE("Beitrag_Stichtag",CURRENT_DATE))-1
    ELSE DATEDIFF('yy',"GebDat",COALESCE("Beitrag_Stichtag",CURRENT_DATE))
    END
    AS "Alter_Beitrag"
015 FROM "tbl_Mitglied", "tbl_Filter"
016 WHERE "tbl_Filter"."ID" = COALESCE ( ( SELECT "ID" FROM "viw_ServVar" ),
    '1' )
```

Die meisten Felder werden direkt aus der Tabelle "tbl_Mitglied" ausgelesen. In Zeile 5 kommt dann die Gruppenzuweisung hinzu: Steht in "Gruppe_mitID" ein Wert, so wird der als Zuweisung zur Gruppe genommen. Sonst wird die "ID" des Mitgliedes selbst dort angezeigt. Das ist notwendig, da die "Gruppe_mitID" beim ersten Mitglied einer Gruppe nicht automatisch

geschrieben wird. Erst die anderen Mitglieder werden darüber mit dem ersten Mitglied verbunden.

Das Alter wird auf verschiedene Art berechnet. In Zeile 12 ist die einfachste Berechnung aufgeführt, die so z. B. auch im Sport gilt: Es wird einfach die Jahresdifferenz zwischen dem aktuellen Datum und dem Geburtsdatum errechnet. Mitglieder, die später im Jahr geboren wurden, werden also erst einmal mit einem zu hohen Alter versehen. In Zeile 13 wird das Alter schließlich mit Hilfe von Monat und Tag genauer berechnet. Ist der Geburtsmonat größer als der Monat des aktuellen Datums oder der Geburtsmonat gleich und der Tag größer, dann ist das Alter um 1 Jahr kleiner als der Jahresunterschied. In Zeile 14 wird schließlich das Alter zu einem ganz besonderen Datum, z. B. einem Stichtag zur Weitergabe der Statistik, berechnet. Dieses Feld "Beitrag_Stichtag" befindet sich in der Tabelle "tbl_Filter". Da kein gleichlautendes Feld in "tbl_Mitglied" vorhanden ist kann die Angabe des Tabellennamens hier entfallen.

In Zeile 16 wird lediglich deutlich gemacht, aus welcher Zeile der "tbl_Filter" gelesen werden soll. Diese Tabelle wird grundsätzlich für einen Nutzer nur mit einer Zeile beschreiben. Weil das bei der Tabelle so ist muss auch keine Beziehung zur Tabelle "tbl_Mitglied" erstellt werden. Die Zeile in "tbl_Filter" existiert auf jeden Fall. Und eine Kombination von z. B. 13 Datensätzen aus "tbl_Mitglied" mit einem Datensatz aus "tbl_Filter" ergibt wieder 13*1=13 Datensätze.

viw_Mitglieder_komplett

mitID	gruID	adrID	Kontoinhaber	IBAN	BIC	Bank	Straße_Nr	PLZ_Ort	Telefon_fest	Nachname
3	3	3	Großkopf, Robert	DE47 110	WELAD	Stadtspa	Unter dem	98701	05971/4242	Müller
4	3	3	Großkopf, Robert	DE47 110	WELAD	Stadtspa	Unter dem	98701	05971/4242	Müller-Mehl
5	3	3	Großkopf, Robert	DE47 110	WELAD	Stadtspa	Unter dem	98701	05971/4242	Müller
6	3	3	Großkopf, Robert	DE47 110	WELAD	Stadtspa	Unter dem	98701	05971/4242	Müller
7	3	7	Großkopf, Robert	DE47 110	WELAD	Stadtspa	Konsumrenn	02431		Schulze
8	8	8	Querulant,	DE23 456	GENOD	Volksban	An dem Bach	02431	09090/1232	Querulant
9	9	9	Schöngeist,				Kirchweg 13	42430		Schöngeist

Vorname	Geschlecht	GebDat	E-Mail	Telefon_mobil	Schwimmer	Abteilungen	Boote	Schlüssel	AufDat	AusDat
Karl	m	27.02.64		0123456789	✅	Wanderabte	K2: Moin –	GS 1, GS :	04.04.21	
Maria	w	13.11.69			✅	Frauenabtei	K1: Sturm	GS 2	01.01.22	01.09.22
Isabel	w	17.07.07			✅	Jugend- und			01.01.22	
Janis	m	23.10.09	flitzepi	0987654321	✅	Jugend- und			01.01.22	
Jolantha	w	17.09.01		090807060504	✅				03.02.22	
Gisbert	m	13.11.03	mailq@		⛔	Rennabteilu	K2: Auton		07.03.22	
Heinz-Her	m	12.07.87			✅		C4: Canad		05.01.21	

Datenquelle	
Tabelle:	*tbl_Mitglied, tbl_Adresse, tbl_Ort, tbl_Konto, tbl_Bank, tbl_Filter*
Ansicht:	*Ansicht_Datum, viw_Boote_Mitglied, viw_Schluessel_Mitglied, viw_ServVar*

Datenziel	
Abfrage:	*qry_Filter_Austritt, qry_Vorstand_aktuell, qry_Vorstand_aktuell_lt_Satzung*
Bericht:	*rpt_Mitglieder_komplett, rpt_Mitglieder_komplett_Konto*

```
001 SELECT "a"."ID" AS "mitID",
002 COALESCE ( "a"."Gruppe_mitID", "a"."ID" ) AS "gruID",
```

```
003 CASE WHEN "b"."mitID" = "a"."ID" THEN "a"."ID" ELSE "a"."Gruppe_mitID" END
    AS "adrID",
004 COALESCE ( "tbl_Konto"."Nachname" || COALESCE ( ', ' ||
    "tbl_Konto"."Vorname", '' ), "a"."Nachname" || COALESCE ( ', ' ||
    "a"."Vorname", '' ) ) AS "Kontoinhaber",
005 "tbl_Konto"."IBAN",
006 "tbl_Bank"."BIC",
007 "tbl_Bank"."Bank",
008 COALESCE ( "b"."Straße_Nr", ( SELECT "Straße_Nr" FROM "tbl_Adresse" WHERE
    "mitID" = COALESCE ( "a"."Gruppe_mitID", "a"."ID" ) ) ) AS "Straße_Nr",
009 COALESCE ( "tbl_Ort"."Postleitzahl" || ' ' || "tbl_Ort"."Ort", ( SELECT
    "tbl_Ort"."Postleitzahl" || ' ' || "tbl_Ort"."Ort" FROM "tbl_Ort",
    "tbl_Adresse" WHERE "tbl_Adresse"."ortID" = "tbl_Ort"."ID" AND
    "tbl_Adresse"."mitID" = COALESCE ( "a"."Gruppe_mitID", "a"."ID" ) ) ) AS
    "PLZ_Ort",
010 COALESCE ( "b"."Telefon_fest", ( SELECT "Telefon_fest" FROM "tbl_Adresse"
    WHERE "mitID" = CASE WHEN "b"."mitID" = "a"."ID" THEN "a"."ID" ELSE
    "a"."Gruppe_mitID" END ) ) AS "Telefon_fest",
011 "a"."Nachname",
012 "a"."Vorname",
013 "a"."Geschlecht",
014 "a"."GebDat",
015 "a"."E-Mail",
016 "a"."Telefon_mobil",
017 CASE WHEN "a"."Schwimmer" = TRUE THEN '  ' ELSE '  ' END AS "Schwimmer",
018 "viw_Abteilungen_Mitglied"."Abteilungen",
019 "viw_Boote_Mitglied"."Boote",
020 "viw_Schluessel_Mitglied"."Schlüssel",
021 "a"."AufDat",
022 "a"."AusDat"
023 FROM "tbl_Mitglied" AS "a"
024 LEFT JOIN "tbl_Adresse" AS "b" ON "b"."mitID" = "a"."ID"
025 LEFT JOIN "tbl_Ort" ON "b"."ortID" = "tbl_Ort"."ID"
026 LEFT JOIN "tbl_Konto" ON ( "tbl_Konto"."mitID" = "a"."Gruppe_mitID" OR
    "tbl_Konto"."mitID" = "a"."ID" )
027 LEFT JOIN "tbl_Bank" ON "tbl_Konto"."banID" = "tbl_Bank"."ID"
028 LEFT JOIN "viw_Abteilungen_Mitglied" ON "viw_Abteilungen_Mitglied"."mitID"
    = "a"."ID"
029 LEFT JOIN "viw_Boote_Mitglied" ON "viw_Boote_Mitglied"."mitID" = "a"."ID"
030 LEFT JOIN "viw_Schluessel_Mitglied" ON "viw_Schluessel_Mitglied"."mitID" =
    "a"."ID"
031 WHERE ( "a"."AusDat" IS NULL OR "a"."AusDat" > CURRENT_DATE )
032 OR "a"."AusDat" BETWEEN ( SELECT "StartDat" FROM "tbl_Filter" WHERE "ID" =
       COALESCE ( ( SELECT "ID" FROM "viw_ServVar" ), '1' ) ) AND ( SELECT
       "EndDat" FROM "tbl_Filter" WHERE "ID" = COALESCE ( ( SELECT "ID" FROM
       "viw_ServVar" ), '1' ) )
```

Diese Ansicht hat sehr viel Ähnlichkeit mit *viw_Filter_Mitglieder_komplett_AusEin*. Hier werden lediglich noch einige Felder hinzugefügt. In Zeile `31` wird definiert, dass nur die Mitglieder angezeigt werden, bei denen das Austrittsdatum "AusDat" leer ist oder vom momentanen Datum aus gesehen größer ist, also in der Zukunft liegt. Da Berichte auch auf einen bestimmten Zeitraum bezogen werden können, werden bei einer Angabe des "StartDat" und des "EndDat" auch die Mitglieder angezeigt, deren "AusDat" innerhalb dieser Zeit erfolgt ist.

viw_Reports

Title	Report	Sort
☰☰ Adressetiketten (2-spaltig)	rpt_Etiketten_2_Spalten	Adressetiketten (2-s
☰☰☰ Adressetiketten (3-spaltig)	rpt_Etiketten_3_Spalten	Adressetiketten (3-s
Anschriftsliste (Tabellendatei)	rpt_Anschriften_Tabelle	Anschriftsliste (Tabe
Austrittsbestätigung	rpt_Austritt	Austrittsbestätigun
Beitragsübersicht	rpt_Beitraege_Konto	Beitragsübersicht
Bootsständer	rpt_Bootstaender_Belegung	Bootsständer
Dauer Mitgliedschaft	rpt_Mitgliedsdauer	Dauer Mitgliedscha
Mitgliederbewegung (Zeitraum, Austritte?)	rpt_Mitglieder_AusEin_Konto	Mitgliederbewegun
Mitgliederliste (komplett, gruppiert)	rpt_Mitglieder_komplett_Konto	Mitgliederliste (kom
Schlüssel augeliehen	rpt_Schluesseluebersicht	Schlüssel augeliehe
Statistik - Altergruppen am Stichtag	rpt_Statistik	Statistik - Altergrup
Urkunde Jubiläum	rpt_Jubilaeum_Ehrenurkunde	Urkunde Jubiläum
Vorstand aktuell komplett	rpt_Vorstand_aktuell	Vorstand aktuell ko
Vorstand aktuell lt. Satung	rpt_Vorstand_aktuell_lt_Satzung	Vorstand aktuell lt.

Datenquelle

Tabelle: *tbl_Einstellungen*

Datenziel

Formular: *frm_Berichte*

```
001 SELECT '      Austrittsbestätigung' AS "Title", 'rpt_Austritt' AS
    "Report", 'Austrittsbestätigung' AS "Sort" FROM "tbl_Einstellungen" WHERE
    "ID" = TRUE
002 UNION
003 SELECT '      Anschriftsliste (Tabellendatei)' AS "Title",
    'rpt_Anschriften_Tabelle' AS "Report", 'Anschriftsliste (Tabellendatei)'
    AS "Sort" FROM "tbl_Einstellungen" WHERE "ID" = TRUE
004 UNION
005 SELECT '☰☰ Adressetiketten (2-spaltig)' AS "Title",
    'rpt_Etiketten_2_Spalten' AS "Report", 'Adressetiketten (2-spaltig)' AS
    "Sort" FROM "tbl_Einstellungen" WHERE "ID" = TRUE
006 UNION
007 SELECT '☰☰☰ Adressetiketten (3-spaltig)' AS "Title",
    'rpt_Etiketten_3_Spalten' AS "Report", 'Adressetiketten (3-spaltig)' AS
    "Sort" FROM "tbl_Einstellungen" WHERE "ID" = TRUE
008 UNION
009 SELECT '   Urkunde Jubiläum' AS "Title", 'rpt_Jubilaeum_Ehrenurkunde' AS
    "Report", 'Urkunde Jubiläum' AS "Sort" FROM "tbl_Einstellungen" WHERE "ID"
    = TRUE
010 UNION
011 SELECT '   Dauer Mitgliedschaft' AS "Title", 'rpt_Mitgliedsdauer' AS
    "Report", 'Dauer Mitgliedschaft' AS "Sort" FROM "tbl_Einstellungen" WHERE
    "ID" = TRUE
012 UNION
013 SELECT '      Vorstand aktuell komplett' AS "Title",
    'rpt_Vorstand_aktuell' AS "Report", 'Vorstand aktuell komplett' AS "Sort"
    FROM "tbl_Einstellungen" WHERE "ID" = TRUE
```

```
014 UNION
015 SELECT '      Vorstand aktuell lt. Satung' AS "Title",
    'rpt_Vorstand_aktuell_lt_Satzung' AS "Report", 'Vorstand aktuell lt.
    Satzung' AS "Sort" FROM "tbl_Einstellungen" WHERE "ID" = TRUE
016 UNION
017 SELECT '   Statistik - Altergruppen am Stichtag' AS "Title",
    'rpt_Statistik' AS "Report", 'Statistik - Altergruppen am Stichtag' AS
    "Sort" FROM "tbl_Einstellungen" WHERE "ID" = TRUE
018 UNION
019 SELECT '      Bootsständer' AS "Title", 'rpt_Bootstaender_Belegung' AS
    "Report", 'Bootsständer' AS "Sort" FROM "tbl_Einstellungen" WHERE "ID" =
    TRUE AND "Bootstaenderverwaltung" = TRUE
020 UNION
021 SELECT '      Beitragsübersicht' AS "Title", 'rpt_Beitraege' AS "Report",
    'Beitragsübersicht' AS "Sort" FROM "tbl_Einstellungen" WHERE "ID" = TRUE
    AND "Beitragsverwaltung" = TRUE AND "Kontoverwaltung" = FALSE
022 UNION
023 SELECT '      Beitragsübersicht' AS "Title", 'rpt_Beitraege_Konto' AS
    "Report", 'Beitragsübersicht' AS "Sort" FROM "tbl_Einstellungen" WHERE
    "ID" = TRUE AND "Beitragsverwaltung" = TRUE AND "Kontoverwaltung" = TRUE
024 UNION
025 SELECT '      Mitgliederliste (komplett, gruppiert)' AS "Title",
    'rpt_Mitglieder_komplett' AS "Report", 'Mitgliederliste (komplett,
    gruppiert)' AS "Sort" FROM "tbl_Einstellungen" WHERE "ID" = TRUE AND
    "Kontoverwaltung" = FALSE
026 UNION
027 SELECT '      Mitgliederliste (komplett, gruppiert)' AS "Title",
    'rpt_Mitglieder_komplett_Konto' AS "Report", 'Mitgliederliste (komplett,
    gruppiert)' AS "Sort" FROM "tbl_Einstellungen" WHERE "ID" = TRUE AND
    "Kontoverwaltung" = TRUE
028 UNION
029 SELECT '     Mitgliederbewegung (Zeitraum, Austritte?)' AS "Title",
    'rpt_Mitglieder_AusEin' AS "Report", 'Mitgliederbewegung (Zeitraum,
    Austritte?)' AS "Sort" FROM "tbl_Einstellungen" WHERE "ID" = TRUE AND
    "Kontoverwaltung" = FALSE
030 UNION
031 SELECT '     Mitgliederbewegung (Zeitraum, Austritte?)' AS "Title",
    'rpt_Mitglieder_AusEin_Konto' AS "Report", 'Mitgliederbewegung (Zeitraum,
    Austritte?)' AS "Sort" FROM "tbl_Einstellungen" WHERE "ID" = TRUE AND
    "Kontoverwaltung" = TRUE
032 UNION
033 SELECT '   Schlüssel ausgeliehen' AS "Title", 'rpt_Schluesseluebersicht' AS
    "Report", 'Schlüssel ausgeliehen' AS "Sort" FROM "tbl_Einstellungen" WHERE
    "ID" = TRUE AND "Schluesselverwaltung" = TRUE
034 ORDER BY 3
```

Diese Ansicht dient dazu, in dem Formular "frm_Berichte" über ein Listenfeld sämtliche Bericht zugänglich zu machen. Die Anzeige wird durch die "tbl_Einstellungen" beeinflusst, in der ja bestimmte Module der Datenbank abgewählt werden können. Natürlich macht die Auswahl eine Berichtes zu den ausgeliehenen Schlüsseln (Zeile 33) nur dann Sinn, wenn in "tbl_Einstellungen" die "Schluesselverwaltung" auf **TRUE** steht.

Der Ansicht wurden zur besseren Übersicht UTF8-Symbole[12] hinzugefügt, die genauso wie Schrift dann in dem Listenfeld erscheinen. Da diese Symbole allerdings die für uns ersichtliche Sortierung durcheinander bringen ist in der Ansicht noch eine 3. Spalte eingefügt worden, die aus der ersten Spalte ohne die Symbole besteht. Wenn Abfragen über **UNION** zusammengeführt werden, dann kann für die Sortierung die Spalte angegeben werden. Das erfolgt in Zeile 34.

12 Die komplette Übersicht zu UTF-8-Zeichen gibt es auf *https://www.utf8-zeichentabelle.de/* Symbole werden kompakt auch hier präsentiert: *https://www.utf8icons.com/*

viw_Schluessel_Mitglied

	mitID	Schlüssel
▶	3	GS 1, GS 3
	4	GS 2
	12	GS 2

Datenquelle
Tabelle: *tbl_MitgliedSchluessel*

Datenziel
Ansicht: *viw_Mitglieder_komplett*

```
001 SELECT DISTINCT "c"."mitID",
002    ( SELECT "SNr"
003    FROM ( SELECT "a"."mitID", "a"."ID",
004       ( SELECT COUNT( "mitID" ) FROM "tbl_Mitglied_Schluessel"
             WHERE "mitID" = "a"."mitID" AND "ID" <= "a"."ID" AND "RueckDat"
             IS NULL AND NOT "Verlust" = TRUE ) AS "Nr"
005       FROM "tbl_Mitglied_Schluessel" AS "a" ) AS "b",
             "tbl_Mitglied_Schluessel"
006    WHERE "b"."ID" = "tbl_Mitglied_Schluessel"."ID" AND "b"."Nr" = 1 AND
          "mitID" = "c"."mitID" ) ||
007    COALESCE ( ', ' || ( SELECT "SNr"
008    FROM ( SELECT "a"."mitID", "a"."ID",
009       ( SELECT COUNT( "mitID" ) FROM "tbl_Mitglied_Schluessel"
             WHERE "mitID" = "a"."mitID" AND "ID" <= "a"."ID" AND "RueckDat"
             IS NULL AND NOT "Verlust" = TRUE ) AS "Nr"
010       FROM "tbl_Mitglied_Schluessel" AS "a" ) AS "b",
             "tbl_Mitglied_Schluessel"
011    WHERE "b"."ID" = "tbl_Mitglied_Schluessel"."ID" AND "b"."Nr" = 2 AND
          "mitID" = "c"."mitID" ), '' ) ||

    ...

022    COALESCE ( ', ' || ( SELECT "SNr"
023    FROM ( SELECT "a"."mitID", "a"."ID",
024       ( SELECT COUNT( "mitID" ) FROM "tbl_Mitglied_Schluessel"
             WHERE "mitID" = "a"."mitID" AND "ID" <= "a"."ID" AND "RueckDat"
             IS NULL AND NOT "Verlust" = TRUE ) AS "Nr"
025       FROM "tbl_Mitglied_Schluessel" AS "a" ) AS "b",
             "tbl_Mitglied_Schluessel"
026    WHERE "b"."ID" = "tbl_Mitglied_Schluessel"."ID" AND "b"."Nr" = 5 AND
     "mitID" = "c"."mitID" ), '' )
027 AS "Schlüssel"
028 FROM "tbl_Mitglied_Schluessel" AS "c"
```

Mit dieser Ansicht werden dem Mitglied in einem Feld alle Schlüssel zugewiesen, die es ausgeleihen hat. Eine derartig aufwendige Abfrage ist nur bei der internen HSQLDB notwendig. Die interne FIREBIRD Datenbank hat dafür die Funktion **`LIST().`**

Die Unterabfrage von Zeile 2 bis Zeile 6 wiederholt sich hier nahezu identisch fünf mal. Die einzigen Unterschiede sind die im Code mit rot hervorgehobenen Zahlen sowie ab der ersten Wiederholung die Verknüpfung über **`COALESCE( ', '|| … , '' )`**.
In der am weitesten innen liegenden Abfrage von Zeile 4 wird dem Schlüssel, der einem einzelnen Mitglied zugeordnet wurde, eine laufende "Nr" zugewiesen. Dadurch kann aus der Abfrage anschließend der Schlüssel mit der "Nr" = 1 herausgefiltert werden.

Zeile 3 und 5 umschließen diese Abfrage und geben zuerst einmal eine "mitID", eine "ID" (aus "tbl_Mitglied_Schluessel" und eine "Nr" aus.
Zeile 2 und 6 dienen dann dazu, diese Unterabfrage mit der "tbl_Mitglied_Schluessel" zu kombinieren und die Schlüssel auszulesen, die in der Unterabfrage die "Nr" = 1 zugewiesen bekommen hat.

Über die folgenden gleichlautenden Abfragen werden die Schlüssel mit den Nummern 2 bis 5 zugeordnet. Es können hier also höchstens 5 Schlüssel hintereinander, durch ein Komma getrennt, in das Feld "Schlüssel" geschrieben werden. In der Praxis dürfte das aber völlig genügen. Tatsächlich gehen wohl viele Vereine inzwischen zu Transpondern und elektronischer Steuerung über, so dass einem Transponder nur noch ein entsprechender Schließbereich zugeordnet wird.

viw_ServVar

	USER_NAME	SESSION_ID	ID
▶	SA	1	1

Datenquelle	
Tabelle:	Systemtabellen der jeweiligen Datenbank

Datenziel	
Ansicht:	*viw_Filter_Mitglieder_komplett_AusEin, viw_Gruppe, viw_Mitglieder_komplett, viw_Statistik_Alter_Detail*
Abfrage:	*qry_Filter, qry_Filter_Austritt, qry_Filter_Gruppe, qry_Filter_Mitglied, qry_Filter_Vorstand*
Makro:	*FilterIDSet, FormStart*

```
001 SELECT "USER_NAME", "SESSION_ID", "SESSION_ID" AS "ID"
002 FROM "INFORMATION_SCHEMA"."SYSTEM_SESSIONS"
```

Diese Ansicht dient lediglich dazu, die aktuelle "SESSION_ID" zu ermitteln und als "ID" für die "tbl_Filter" zu nutzen. Bei der internen HSQLDB ist diese "SESSION_ID" '1', sofern sie überhaupt zu ermitteln ist. Die Arbeit mit der Session ist bei der Nutzung von Filtertabellen zusammen mit Serverdatenbanken sinnvoll. So können Filterungen in Abhängigkeit von der aktuellen Datenbankverbindung erfolgen und mehrere Nutzer unabhängig voneinander eine Filterung der Daten vornehmen.

viw_Statistik_Alter_Detail

	Alter	Geschlecht	Anzahl	Stichtag	MinAlter
▸	0 bis 6	Frauen	0	01.06.22	0
	7 bis 14	Frauen	1	01.06.22	7
	15 bis 18	Frauen	0	01.06.22	15
	19 bis 26	Frauen	1	01.06.22	19
	27 bis 40	Frauen	2	01.06.22	27
	41 bis 60	Frauen	1	01.06.22	41
	60 bis 12	Frauen	1	01.06.22	60
	0 bis 6	Männer	0	01.06.22	0
	7 bis 14	Männer	1	01.06.22	7
	15 bis 18	Männer	1	01.06.22	15

Datenquelle
Tabelle: *tbl_Mitglied*, *tbl_Altergruppen*, *tbl_Filter*
Ansicht: *viw_ServVar*

Datenziel
Bericht: *rpt_Statistik*

```
001 SELECT "MinAlter" || ' bis ' || "MaxAlter" AS "Alter",
002 'Frauen' AS "Geschlecht",
003 (SELECT COUNT("ID") FROM "tbl_Mitglied" WHERE CASE WHEN
    DAYOFYEAR( "GebDat" ) > DAYOFYEAR( ( SELECT "Statistik_Stichtag" FROM
    "tbl_Filter" WHERE "ID" = COALESCE ( ( SELECT "ID" FROM "viw_ServVar" ),
    '1' ) ) )
    THEN DATEDIFF( 'yy', "GebDat", ( SELECT "Statistik_Stichtag" FROM
    "tbl_Filter" WHERE "ID" = COALESCE ( ( SELECT "ID" FROM "viw_ServVar" ),
    '1' ) ) ) - 1
    ELSE DATEDIFF( 'yy', "GebDat", ( SELECT "Statistik_Stichtag" FROM
    "tbl_Filter" WHERE "ID" = COALESCE ( ( SELECT "ID" FROM "viw_ServVar" ),
    '1' ) ) ) END
    BETWEEN "a"."MinAlter" AND "a"."MaxAlter" AND "Geschlecht" = 'w' AND
    ( "AusDat" > ( SELECT "Statistik_Stichtag" FROM "tbl_Filter" WHERE "ID" =
    COALESCE ( ( SELECT "ID" FROM "viw_ServVar" ), '1' ) ) OR "AusDat" IS NULL
    )) AS "Anzahl",
004 ( SELECT "Statistik_Stichtag" FROM "tbl_Filter" WHERE "ID" = COALESCE
    ( ( SELECT "ID" FROM "viw_ServVar" ), '1' ) ) AS "Stichtag",
005 "MinAlter"
006 FROM "tbl_Altersgruppen" AS "a"
007 UNION
008 SELECT "MinAlter" || ' bis ' || "MaxAlter" AS "Alter",
009 'Männer' AS "Geschlecht",
010 (SELECT COUNT("ID") FROM "tbl_Mitglied" WHERE CASE WHEN
    DAYOFYEAR( "GebDat" ) > DAYOFYEAR( ( SELECT "Statistik_Stichtag" FROM
    "tbl_Filter" WHERE "ID" = COALESCE ( ( SELECT "ID" FROM "viw_ServVar" ),
    '1' ) ) )
    THEN DATEDIFF( 'yy', "GebDat", ( SELECT "Statistik_Stichtag" FROM
    "tbl_Filter" WHERE "ID" = COALESCE ( ( SELECT "ID" FROM "viw_ServVar" ),
    '1' ) ) ) - 1
    ELSE DATEDIFF( 'yy', "GebDat", ( SELECT "Statistik_Stichtag" FROM
    "tbl_Filter" WHERE "ID" = COALESCE ( ( SELECT "ID" FROM "viw_ServVar" ),
```

```
     '1' ) ) ) END
     BETWEEN "a"."MinAlter" AND "a"."MaxAlter" AND "Geschlecht" = 'm' AND
     ( "AusDat" > ( SELECT "Statistik_Stichtag" FROM "tbl_Filter" WHERE "ID" =
     COALESCE ( ( SELECT "ID" FROM "viw_ServVar" ), '1' ) ) OR "AusDat" IS NULL
     )) AS "Anzahl",
 011 ( SELECT "Statistik_Stichtag" FROM "tbl_Filter" WHERE "ID" = COALESCE
     ( ( SELECT "ID" FROM "viw_ServVar" ), '1' ) ) AS "Stichtag",
 012 "MinAlter"
 013 FROM "tbl_Altersgruppen" AS "a"
 014 ORDER BY "Geschlecht", "MinAlter"
```

In der "tbl_Altersgruppen" sind für das Anlegen einer Statistik wichtige Altersgruppen zusammengefasst. Diese Ansicht verknüpft diese Altersgruppen mit der Anzahl der tatsächlich vorhandenen Mitglieder im Verein bezogen auf einen bestimmten Stichtag. Anhand dieser Ansicht kann dann ein entsprechendes Diagramm innerhalb eines Berichtes angezeigt werden.

Die Altersgruppen werden in Zeile `3` bzw. `10` mit dem aktuellen Alter bezogen auf den eingetragenen Stichtag verglichen. Entsprechend erfolgt die Einordnung der Mitglieder in die Altersgruppen. Das Mitglied darf allerdings zum Stichtag nicht aus dem Verein ausgetreten sein.

Die Ansicht umfasst dabei alle Altersgruppen, so dass gegebenenfalls auch 0-Werte erscheinen.

Abfragen

Abfragen können in Formularen zur weiteren Bearbeitung eingesetzt werden, wenn die Eingabe von Daten ermöglicht wird. Dies geschieht dadurch, dass auf jeden Fall alle betreffenden Tabellen mit Primärschlüsseln in der Abfrage vertreten sind.

Bei nicht editierbaren Abfragen gibt es Abfragen, die nur über **Bearbeiten → SQL-Befehl direkt ausführen** überhaupt ablaufen. Das hängt damit zusammen, dass die GUI den Code, der dort steht, nicht richtig deuten kann und einfach so direkt an die Datenbank weiterleitet. Solche Abfragen haben den Vorteil, dass sich der Code auch formatieren lässt. Absätze können eingefügt werden, Kommentare sind auch möglich. Leider haben diese Abfragen aber den Nachteil, dass auch der Report Designer damit nicht so viel anfangen kann. Die gesamte Gruppierung und Sortierung ist damit im Report Designer unmöglich. Für den Report Designer sind in diesem Fall Ansichten (**`Views`**) deutlich besser geeignet. Den Code bekommt der Report Designer nicht zu Gesicht. Für ihn ist das Ganze nur eine Tabelle, die nicht beschreibbar ist.

Nicht editierbare Abfragen, die ohne das Umstellen in die direkte Ausführung funktionieren, sind für den Report Designer auch zur Gruppierung und Sortierung brauchbar.

Hinweis

In der Liste der Abfragen tauchen auch die Abfragen auf, die anschließend zu Ansichten gemacht wurden. Die normalen Abfragen beginnen mit "qry_...", die Abfragen, die zu Ansichten der Datenbank wurden, beginnen mit "v_qry_...".
Diese Maßnahme ist sinnvoll, wenn an der Datenbank noch weiter gearbeitet wird. Ansichten werden in der Datenbank gespeichert und werden dort bei jeder Änderung auch überprüft. So kann z. B. eine Ansicht nicht mehr geändert werden, wenn eine andere Ansicht Daten aus dieser Ansicht bezieht. Auch kann eine Tabelle nicht mehr geändert werden, wenn eine Ansicht Felder dieser Tabelle nutzt. Da hilft es dann nur, die Ansichten entsprechend zu löschen und anschließend aus den als Abfragen gesicherten Ansichten wieder her zu stellen.
Um dieses Vorgehen zu vereinfachen existiert dafür die Prozedur *ViewsErstellen*, die alle vorhandenen Ansichten löscht und aus den existierenden Abfragen "v_qry_..." anschließend in der korrekten Reihenfolge neue Ansichten erstellt. Deshalb sollten die entsprechenden Abfragen nicht gelöscht werden.

qry_Adresse_Gruppe

	mitID	Straße_Nr	ortID	Nichtdruck	Telefon_fest	ID	Gruppe_mitID
	3	Unter dem Baum 42	1	☐	05971/424242	3	
	7	Konsumrennbahn 1a	2	☐		7	3
	8	An dem Bach 162c	2	☐	09090/123234	8	
	9	Kirchweg 13	0	☐		9	
	10	Industriegebiet Nord 4711	1	☐	14567/89123	10	
▷+				☐		<Auto	

Datenquelle

Tabelle: *tbl_Adresse, tbl_Mitglied*

Datenziel

Formular: *frm_Mitglied*

```
001 SELECT "tbl_Adresse".*,
002 "tbl_Mitglied"."ID",
003 "tbl_Mitglied"."Gruppe_mitID"
004 FROM "tbl_Adresse", "tbl_Mitglied"
005 WHERE "tbl_Adresse"."mitID" = "tbl_Mitglied"."ID"
```

Diese Abfrage gibt neben den Feldern der "tbl_Adresse" die Felder "tbl_Mitglied"."ID" und "tbl_Mitglied"."Gruppe_mitID" wieder. Damit zeigt sie die Gruppenzugehörigkeit an, auch wenn eine Person wie die mit der "mitID" = '7' eine eigene Adresse hat.

Die Abfrage ist beschreibbar, da die Primärschlüssel beider betroffenen Tabellen in der Abfrage enthalten sind.

qry_Anschriften_2spaltig

	lfdNr_1	Anschrift_1	lfdNr_2	Anschrift_2
▷	1	Karl, Maria, Isabel und Janis Müller und Müller-M	2	FrauYvonne-Chantal PowerIndustı
	3	HerrnGisbert QuerulantAn dem Bach 162c02431	4	HerrnHeinz-Herbert SchöngeistKir
	5	FrauJolantha SchulzeKonsumrennbahn 1a02431		

Datenquelle

Ansicht: *viw_Anschrift*

Datenziel

Bericht: *Urkunden*

```
001 SELECT "b"."lfdNr" AS "lfdNr_1",
002 "b"."Anschrift" AS "Anschrift_1",
003 "c"."lfdNr" + 1 AS "lfdNr_2",
004 "c"."Anschrift" AS "Anschrift_2"
005 FROM ( SELECT
006    COALESCE ( NULLIF ( "Anrede" || CHAR( 13 ) || CHAR( 10 ), CHAR( 13 ) ||
       CHAR( 10 ) ), '' ) || COALESCE ( "Vornamen" || ' ', '' ) || "Nachnamen"
       || CHAR( 13 ) || CHAR( 10 ) || "Straße_Nr" || CHAR( 13 ) || CHAR( 10 )
       || "Postleitzahl" || ' ' || "Ort" AS "Anschrift",
```

```
007    ( SELECT COUNT( "adrID" ) FROM "viw_Anschrift" WHERE "Nachnamen" ||
       "Straße_Nr" || "Postleitzahl" || "Ort" <= "a"."Nachnamen" ||
       "a"."Straße_Nr" || "a"."Postleitzahl" || "a"."Ort" ) AS "lfdNr"
008 FROM "viw_Anschrift" AS "a" )
009 AS "b"
010 LEFT JOIN ( SELECT
011    COALESCE ( NULLIF ( "Anrede" || CHAR( 13 ) || CHAR( 10 ), CHAR( 13 ) ||
       CHAR( 10 ) ), '' ) || COALESCE ( "Vornamen" || ' ', '' ) || "Nachnamen"
       || CHAR( 13 ) || CHAR( 10 ) || "Straße_Nr" || CHAR( 13 ) || CHAR( 10 )
       || "Postleitzahl" || ' ' || "Ort" AS "Anschrift",
012    ( SELECT COUNT( "adrID" ) FROM "viw_Anschrift" WHERE "Nachnamen" ||
       "Straße_Nr" || "Postleitzahl" || "Ort" <= "a"."Nachnamen" ||
       "a"."Straße_Nr" || "a"."Postleitzahl" || "a"."Ort" ) -1 AS "lfdNr"
013 FROM "viw_Anschrift" AS "a" )
014 AS "c"
015 ON "c"."lfdNr" = "b"."lfdNr"
016 WHERE MOD( "b"."lfdNr", 2 ) = 1
```

Diese Abfrage wird für Etiketten benötigt, die zweispaltig als Bericht gedruckt werden können. Die beiden Spalten werden über die Unterabfragen von Zeile 5 bis 9 und Zeile 10 bis 14 zusammengestellt.

Die Abfrage mit dem Alias "b" (Zeile 9) gibt dabei alle Datensätze wieder, deren laufende Nummer ungerade ist (Zeile 16 – der Rest der Teilung der "lfdNr" durch '2' ist '1').

Die Abfrage mit dem Alias "c" (Zeile 14), erhält eine "lfdNr", die um 1 vermindert wird. Dadurch kann "c"."lfdNr" und "b"."lfdNr" gleichgesetzt werden. Sie zeigen so nicht die gleichen Datensätze an. Der Datensatz in "c" zeigt den Inhalt der "lfdNr" = 2 , während in der gleichen Zeile der Datensatz in "b" den Inhalt der "lfdNr" = 1 anzeigt.

In Zeile 3 wird die Anzeige der "lfdNr_2" wieder auf den ursprünglichen Wert über eine Addition mit 1 zurückgesetzt.

Da auf den Etiketten bei vielen Namen in einer Gruppe wenig Platz ist soll der Platz für die "Anrede" nicht durch einen Zeilenumbruch **`CHAR( 13 ) || CHAR( 10 )`** belegt werden, wenn die Anrede leer ist. Der Zeilenumbruch selbst stellt aber bereits einen Inhalt dar, so dass hier ein einfaches Vorgehen mit **`COALSECE()`** nicht greift. Deshalb muss mit **`NULLIF`** dann **`NULL`** zugewiesen werden, wenn nur noch der Code für den Zeilenumbruch enthalten ist (Zeile 6 und Zeile 11).

Die Abfrage ist nicht beschreibbar, wird aber auch nicht im direkten SQL-Modus ausgeführt, so dass sie zur Sortierung geeignet ist. Dadurch ist sie auch im Bericht für die Sortierung und Gruppierung nutzbar.

qry_Anschriften_3spaltig

	lfdNr_1	Anschrift_1	lfdNr_2	Anschrift_2	lfdNr_3	Anschrift_3
▸	1	Karl, Maria, Isabel un	2	FrauYvonne-Chanta	3	HerrnGisbert Querulant
	4	HerrnHeinz-Herbert	5	FrauJolantha Schulz		

Datenquelle
Tabelle:
Ansicht: *viw_Anschrift*

Datenziel
Bericht: *rpt_Etiketten_3_Spalten*

```
001 SELECT "b"."lfdNr" AS "lfdNr_1",
```

```
002 "b"."Anschrift" AS "Anschrift_1",
003 "c"."lfdNr" + 1 AS "lfdNr_2",
004 "c"."Anschrift" AS "Anschrift_2",
005 "d"."lfdNr" + 2 AS "lfdNr_3",
006 "d"."Anschrift" AS "Anschrift_3"
007 FROM ( SELECT
008    COALESCE ( NULLIF ( "Anrede" || CHAR( 13 ) || CHAR( 10 ), CHAR( 13 ) ||
       CHAR( 10 ) ), '' ) || COALESCE ( "Vornamen" || ' ', '' ) || "Nachnamen"
       || CHAR( 13 ) || CHAR( 10 ) || "Straße_Nr" || CHAR( 13 ) || CHAR( 10 )
       || "Postleitzahl" || ' ' || "Ort" AS "Anschrift",
009    ( SELECT COUNT( "adrID" ) FROM "viw_Anschrift" WHERE "Nachnamen" ||
       "Straße_Nr" || "Postleitzahl" || "Ort" <= "a"."Nachnamen" ||
       "a"."Straße_Nr" || "a"."Postleitzahl" || "a"."Ort" ) AS "lfdNr"
010 FROM "viw_Anschrift" AS "a" )
011 AS "b"
012 LEFT JOIN ( SELECT
013    COALESCE ( NULLIF ( "Anrede" || CHAR( 13 ) || CHAR( 10 ), CHAR( 13 ) ||
       CHAR( 10 ) ), '' ) || COALESCE ( "Vornamen" || ' ', '' ) || "Nachnamen"
       || CHAR( 13 ) || CHAR( 10 ) || "Straße_Nr" || CHAR( 13 ) || CHAR( 10 )
       || "Postleitzahl" || ' ' || "Ort" AS "Anschrift",
014    ( SELECT COUNT( "adrID" ) FROM "viw_Anschrift" WHERE "Nachnamen" ||
       "Straße_Nr" || "Postleitzahl" || "Ort" <= "a"."Nachnamen" ||
       "a"."Straße_Nr" || "a"."Postleitzahl" || "a"."Ort" ) -1 AS "lfdNr"
015 FROM "viw_Anschrift" AS "a" )
016 AS "c"
017 ON "c"."lfdNr" = "b"."lfdNr"
018 LEFT JOIN ( SELECT
019    COALESCE ( NULLIF ( "Anrede" || CHAR( 13 ) || CHAR( 10 ), CHAR( 13 ) ||
       CHAR( 10 ) ), '' ) || COALESCE ( "Vornamen" || ' ', '' ) || "Nachnamen"
       || CHAR( 13 ) || CHAR( 10 ) || "Straße_Nr" || CHAR( 13 ) || CHAR( 10 )
       || "Postleitzahl" || ' ' || "Ort" AS "Anschrift",
020    ( SELECT COUNT( "adrID" ) FROM "viw_Anschrift" WHERE "Nachnamen" ||
       "Straße_Nr" || "Postleitzahl" || "Ort" <= "a"."Nachnamen" ||
       "a"."Straße_Nr" || "a"."Postleitzahl" || "a"."Ort" ) -2 AS "lfdNr"
021 FROM "viw_Anschrift" AS "a" )
022 AS "d"
023 ON "c"."lfdNr" = "d"."lfdNr"
024 WHERE MOD( "b"."lfdNr", 3 ) = 1
```

Diese Abfrage wird für Etiketten benötigt, die dreispaltig als Bericht gedruckt werden können. Die drei Spalten werden über die Unterabfragen von Zeile 7 bis 11, Zeile 12 bis 16 und Zeile 18 bis 22 zusammengestellt.

Die Abfrage mit dem Alias "b" (Zeile 11) gibt dabei alle Datensätze wieder, deren laufende Nummer bei einer Teilung durch 3 einen Rest von '1' ergibt (1, 4, 7 usw., **MOD()** in Zeile 24)

Die Abfrage mit dem Alias "c" (Zeile 16), erhält eine "lfdNr", die um 1 vermindert wird. Dadurch kann "c"."lfdNr" und "b"."lfdNr" gleichgesetzt werden. Sie zeigen so nicht die gleichen Datensätze an. Der Datensatz in "c" zeigt den Inhalt der "lfdNr" = 2 , während in der gleichen Zeile der Datensatz in "b" den Inhalt der "lfdNr" = 1 anzeigt.

In Zeile 3 wird die Anzeige der "lfdNr_2" wieder auf den ursprünglichen Wert über eine Addition mit 1 zurückgesetzt.

Entsprechend erfolgt eine Korrektur für die 3. Spalte , indem zuerst die "lfdNr" um 2 vermindert wird (Zeile 20) und dann in der Anzeige wieder um 2 erhöht wird (Zeile 5).

Da auf den Etiketten bei vielen Namen in einer Gruppe wenig Platz ist soll der Platz für die "Anrede" nicht durch einen Zeilenumbruch **CHAR(13) || CHAR(10)** belegt werden, wenn die Anrede leer ist. Der Zeilenumbruch selbst stellt aber bereits einen Inhalt dar, so dass hier ein einfaches Vorgehen mit **COALSECE()** nicht greift. Deshalb muss mit **NULLIF** dann **NULL** zuge-

wiesen werden, wenn nur noch der Code für den Zeilenumbruch enthalten ist (Zeile 6 und Zeile 11).

Die Abfrage ist nicht beschreibbar, wird aber auch nicht im direkten SQL-Modus ausgeführt, so dass sie zur Sortierung geeignet ist. Dadurch ist sie auch im Bericht für die Sortierung und Gruppierung nutzbar.

qry_Beitraege

	gruID	Kontoinhaber	IBAN	BIC	Bank	Sort	Beitragsart	Beitrag	Quartal	Quartal1	Jahr
	3	Großkopf, Robert	DE47 110	WELAD	Stadtspa	1	einfacher Beitra	25	105	195	510
	3	Großkopf, Robert	DE47 110	WELAD	Stadtspa	2	Bootsständer fi	10	105	195	510
	3	Großkopf, Robert	DE47 110	WELAD	Stadtspa	3	Abteilungsbeitr	90	105	195	510
	8	Querulant,	DE23 456	GENOD	Volksban	1	einfacher Beitra	10	36	146	254
	8	Querulant,	DE23 456	GENOD	Volksban	2	Bootsständer fi	2	36	146	254
	8	Querulant,	DE23 456	GENOD	Volksban	3	Abteilungsbeitr	110	36	146	254
	9	Schöngeist,				1	einfacher Beitra	10	42	42	168
	9	Schöngeist,				2	Bootsständer fi	4	42	42	168

Datenquelle	
Tabelle:	*tbl_Mitglied, tbl_Konto, tbl_Bank*
Ansicht:	*viw_Beitrag*

Datenziel	
Bericht:	*rpt_Beitraege, rpt_Beitraege_Konto*

```
001 SELECT "tbl_Mitglied"."ID" AS "gruID",
002 COALESCE ( "tbl_Konto"."Nachname" || COALESCE ( ', ' ||
    "tbl_Konto"."Vorname", '' ), "tbl_Mitglied"."Nachname" || COALESCE ( ', '
    || "tbl_Mitglied"."Vorname", '' ) ) AS "Kontoinhaber",
003 "tbl_Konto"."IBAN",
004 "tbl_Bank"."BIC",
005 "tbl_Bank"."Bank",
006 "a"."Sort",
007 "a"."Beitragsart",
008 "a"."Beitrag",
009 ( SELECT "Beitrag" FROM "viw_Beitrag" WHERE "gruID" = "a"."gruID" AND
    "Sort" = 4 ) AS "Quartal",
010 ( SELECT "Beitrag" FROM "viw_Beitrag" WHERE "gruID" = "a"."gruID" AND
    "Sort" = 5 ) AS "Quartal1",
011 ( SELECT "Beitrag" FROM "viw_Beitrag" WHERE "gruID" = "a"."gruID" AND
    "Sort" = 6 ) AS "Jahr"
012 FROM "tbl_Mitglied"
013 LEFT JOIN "tbl_Konto" ON "tbl_Konto"."mitID" = "tbl_Mitglied"."ID"
014 LEFT JOIN "tbl_Bank" ON "tbl_Konto"."banID" = "tbl_Bank"."ID"
015 LEFT JOIN "viw_Beitrag" AS "a" ON "tbl_Mitglied"."ID" = "a"."gruID"
016 WHERE "a"."Sort" < 4
```

Hier werden sämtliche Beiträge, die in "viw_Beitrag" ermittelt wurden, zusammen mit dem Kontoinhaber für einen entsprechenden Bericht zusammengeführt. Die Kontoinformationen und die Beträge in "Quartal", "Quartal1" und "Jahr" lauten in allen Zeilen für das entsprechende Konto gleich. Nach diesen Informationen wird der Bericht gruppiert, so dass sie im Gruppenkopf bzw. Gruppenfuß jeweils nur einmal auftauchen.

Die Abfrage ist nicht beschreibbar, wird aber auch nicht im direkten SQL-Modus ausgeführt, so dass sie zur Filterung und Sortierung geeignet ist. Dadurch ist sie auch im Bericht für die Sortierung und Gruppierung nutzbar.

qry_Bootsstaender

	Halle_lang	Halle_kurz	bo_hID	lfdNr	BSt1	Boot1	BSt2	Boot2
	Garage 1	G1	3	1	0.1		0.2	
	Garage 1	G1	3	4	1.1		1.2	K1: Emsge
	Garage 1	G1	3	8	2.1		2.2	K2: Seeblic

BSt3	Boot3	BSt4	Boot4	BSt5	Boot5	BSt6	Boot6	BSt7	Boot7
0.3	K2: Moing								
1.3	K1: Sturm	1.4							
2.3	K2: Auton	2.4	C4: Canad						

Datenquelle
Tabelle: *tbl_Mitglied, tbl_Boot, tbl_BootArt, tbl_BootStaender, tbl_BootHalle*

Datenziel
Bericht: *rpt_Bootstaender_Belegung*

```
001 SELECT "tbl_BootHalle"."Halle_lang",
002 "tbl_BootHalle"."Halle_kurz",
003 "a"."bo_hID",
004 ( SELECT COUNT( "ID" ) FROM "tbl_BootStaender" WHERE "bo_hID" =
    "a"."bo_hID" AND "BootStaend" <= "a"."BootStaend" ) AS "lfdNr",
005 "a"."BootStaend" AS "BSt1",
006 ( SELECT "tbl_BootArt"."Bootsart" || ': ' || "tbl_Boot"."Bootsname" ||
    CHAR( 13 ) || CHAR( 10 ) || 'gehört: ' || COALESCE
    ( "tbl_Mitglied"."Vorname" || ' ', '' ) || "tbl_Mitglied"."Nachname" AS
    "Boot"
    FROM "tbl_Boot", "tbl_BootArt", "tbl_Mitglied", "tbl_BootStaender"
    WHERE "tbl_Boot"."bo_aID" = "tbl_BootArt"."ID"
    AND "tbl_Boot"."mitID" = "tbl_Mitglied"."ID"
    AND "tbl_BootStaender"."ID" = "tbl_Boot"."bo_sID"
    AND "tbl_BootStaender"."bo_hID" = "a"."bo_hID"
    AND "tbl_BootStaender"."BootStaend" = "a"."BootStaend" ) AS "Boot1",
007 ( SELECT "BootStaend" FROM "tbl_BootStaender"
    WHERE RIGHT( "BootStaend", 1 ) = '2' AND "bo_hID" = "a"."bo_hID"
    AND LEFT( "BootStaend", 1 ) = LEFT( "a"."BootStaend", 1 ) ) AS "BSt2",
008 ( SELECT "tbl_BootArt"."Bootsart" || ': ' || "tbl_Boot"."Bootsname" ||
    CHAR( 13 ) || CHAR( 10 ) || 'gehört: ' || COALESCE
    ( "tbl_Mitglied"."Vorname" || ' ', '' ) || "tbl_Mitglied"."Nachname" AS
    "Boot"
    FROM "tbl_Boot", "tbl_BootArt", "tbl_Mitglied", "tbl_BootStaender"
    WHERE "tbl_Boot"."bo_aID" = "tbl_BootArt"."ID"
    AND "tbl_Boot"."mitID" = "tbl_Mitglied"."ID"
    AND "tbl_BootStaender"."ID" = "tbl_Boot"."bo_sID"
    AND "tbl_BootStaender"."bo_hID" = "a"."bo_hID"
    AND RIGHT( "tbl_BootStaender"."BootStaend", 1 ) = '2'
    AND LEFT( "tbl_BootStaender"."BootStaend", 1 ) = LEFT( "a"."BootStaend", 1
    ) ) AS "Boot2",
```

```
...
017 ( SELECT "BootStaend" FROM "tbl_BootStaender"
    WHERE RIGHT( "BootStaend", 1 ) = '7' AND "bo_hID" = "a"."bo_hID"
    AND LEFT( "BootStaend", 1 ) = LEFT( "a"."BootStaend", 1 ) ) AS "BSt7",
018 ( SELECT "tbl_BootArt"."Bootsart" || ': ' || "tbl_Boot"."Bootsname" ||
    CHAR( 13 ) || CHAR( 10 ) || 'gehört: ' || COALESCE
    ( "tbl_Mitglied"."Vorname" || ' ', '' ) || "tbl_Mitglied"."Nachname" AS
    "Boot"
    FROM "tbl_Boot", "tbl_BootArt", "tbl_Mitglied", "tbl_BootStaender"
    WHERE "tbl_Boot"."bo_aID" = "tbl_BootArt"."ID"
    AND "tbl_Boot"."mitID" = "tbl_Mitglied"."ID"
    AND "tbl_BootStaender"."ID" = "tbl_Boot"."bo_sID"
    AND "tbl_BootStaender"."bo_hID" = "a"."bo_hID"
    AND RIGHT( "tbl_BootStaender"."BootStaend", 1 ) = '7'
    AND LEFT( "tbl_BootStaender"."BootStaend", 1 ) = LEFT( "a"."BootStaend", 1
    ) ) AS "Boot7"
019 FROM "tbl_BootStaender" AS "a", "tbl_BootHalle"
020 WHERE "a"."bo_hID" = "tbl_BootHalle"."ID"
021 AND RIGHT( "a"."BootStaend", 1 ) = '1'
```

Bei dem Beispielverein können in einer Halle maximal 7 Boote nebeneinander liegen. Entsprechend werden hier die Informationen für maximal sieben Bootslagerplätze je Halle nebeneinander erfasst, um anschließend in einem Bericht ausdruckbar zu sein.

In Zeile 4 wird eine laufende Nummer erzeugt, die die Einsortierung der Bootsständer nacheinander ermöglicht. Diese laufende Nummer beginnt für jede Bootshalle neu. Der "BSt1" wird durch die Bedingung in Zeile 21 festgelegt. Alle anderen Bootsständer ("BSt2" bis "BSt7") werden durch Unterabfragen (Zeile 7 und Zeile 17) zugeordnet.

Das Boot in dem jeweiligen Bootsständer wird mit allen Informationen, nicht nur dem Bootsnamen, angezeigt. Dazu sind die Informationen aus der "tbl_Mitglied", der "tbl_Boot", der "tbl_BootArt" und der "tbl_BootHalle" erforderlich. Der entsprechende Code ist in Zeile 6, 8 und 18 zu sehen.

Der Code für die Bootsständer 3 bis 6 ist entsprechend dem Code für die Bootsständer 2 und 7 aufgebaut.

Die Abfrage ist nicht beschreibbar, wird aber auch nicht im direkten SQL-Modus ausgeführt, so dass sie zur Filterung und Sortierung geeignet ist. Dadurch ist sie auch im Bericht für die Sortierung und Gruppierung nutzbar.

qry_Filter

	ID	mitID	Formular	WahlDat	Bericht	StartDat	EndDat	Austritte
	1	3	frm_Schluessel	31.01.22	rpt_Mitglieder_Au			☐

Unterschrift1ID	Unterschrift2ID	Statistik_Stichtag	Beitrag_Stichtag
1	2	01.06.22	01.11.22

Datenquelle	
Tabelle:	*tbl_Filter*
Ansicht:	*viw_ServVar*

Datenziel
Formular: *Einzelmessung_System, frm_Berichte, frm_Boot, frm_Einstellungen, frm_Mitglied, frm_Schluessel, frm_Vorstand*

```
001 SELECT *
002 FROM "tbl_Filter"
003 WHERE "ID" = COALESCE ( ( SELECT "ID" FROM "viw_ServVar" ), '1' )
```

Mit dieser Abfrage wird aus der "tbl_Filter" auch im Mehrbenutzerbetrieb nur der Datensatz angezeigt, der für die jeweilige Verbindung gilt. Das Ganze geschieht abhängig von der Variablen "ID" in "viw_ServVar", die allerdings bei der internen HSQLDB '1' ist oder manchmal auch gar nicht belegt ist. Daher die feste Zuschreibung: Kommt die Variable nicht vor, so soll sie '1' sein.

Die Abfrage ist beschreibbar, da die Primärschlüssel beider betroffenen Tabellen in der Abfrage enthalten sind.

qry_Filter_Austritt

	Vereinsname	Anschrift_hintereinander	Anschrift_untereinander	Logo
▸	LibreOffice Base	LibreOffice Base n.e.V., Zwisch	LibreOffice Base n.e.V.Zwisch	<OBJECT>

Bankverbindung	Anschrift_Mitglied	Anrede	Zusatz_Boote_Schluessel
IBAN: DE12 3456 789(	FrauMaria Müller-Meh	Sehr geehrte Frau	Boote müssen zum Ende der Mitglied

Datenquelle	
Tabelle:	*tbl_Einstellungen, qry_Filter*
Ansicht:	*viw_Mitglieder_komplett, viw_ServVar*

Datenziel	
Bericht:	*Ergebnisliste_Zeit*

```
001 SELECT "tbl_Einstellungen"."Vereinsname",
002 "tbl_Einstellungen"."Vereinsname" || ', ' ||
    "tbl_Einstellungen"."Anschrift" AS "Anschrift_hintereinander",
003 "tbl_Einstellungen"."Vereinsname" || CHAR( 13 ) || CHAR( 10 ) ||
    REPLACE( "tbl_Einstellungen"."Anschrift", ', ', CHAR( 13 ) || CHAR( 10 ) )
    AS "Anschrift_untereinander",
004 "tbl_Einstellungen"."Logo",
005 'IBAN: ' || "tbl_Einstellungen"."IBAN" || ' BIC: ' ||
    "tbl_Einstellungen"."BIC" || ' Bank: ' || "tbl_Einstellungen"."Bank" AS
    "Bankverbindung",
006 CASE WHEN "a"."Geschlecht" = 'm' THEN 'Herrn' WHEN "a"."Geschlecht" = 'w'
    THEN 'Frau' ELSE '' END || CHAR( 13 ) || CHAR( 10 ) || "a"."Vorname" || '
    ' || "a"."Nachname" || CHAR( 13 ) || CHAR( 10 ) || "a"."Straße_Nr" ||
    CHAR( 13 ) || CHAR( 10 ) || "a"."PLZ_Ort" AS "Anschrift_Mitglied",
007 CASE WHEN "a"."Geschlecht" = 'm' THEN 'Sehr geehrter Herr' WHEN
    "a"."Geschlecht" = 'w' THEN 'Sehr geehrte Frau ' ELSE 'Hallo ' END ||
    "a"."Vorname" || ' ' || "a"."Nachname" || ',' || CHAR( 13 ) || CHAR( 10 )
    || CHAR( 13 ) || CHAR( 10 ) || CASE WHEN "a"."Geschlecht" = 'm' THEN 'Sie
    haben am ' WHEN "a"."Geschlecht" = 'w' THEN 'Sie haben am ' ELSE 'Du hast
    am ' END || EXTRACT( DAY FROM "a"."AusDat" ) || '.' || EXTRACT( MONTH FROM
    "a"."AusDat" ) || '.' || EXTRACT( YEAR FROM "a"."AusDat" ) || ' den
    Austritt aus dem Verein erklärt.' AS "Anrede",
```

```
 008 COALESCE ( 'Boote müssen zum Ende der Mitgliedschaft aus den Räumen des
     Vereins entfernt werden. Andernfalls muss weiterhin der Mitgliedsbeitrag
     zusammen mit der Bootsständermiete entrichtet werden.' || CHAR( 13 ) ||
     CHAR( 10 ) || '     Verzeichnete Boote: ' || "a"."Boote" || CHAR( 13 ) ||
     CHAR( 10 ), '' ) || COALESCE ( 'Schlüssel müssen zum Ende der
     Mitgliedschaft zurückgegeben werden. Andernfalls werden sie als Verlust
     gekennzeichnet. Der Pfand wird dann einbehalten.' || CHAR( 13 ) ||
     CHAR( 10 ) || '     Verzeichnete Schlüssel: ' || "a"."Schlüssel", '' ) AS
     "Zusatz_Boote_Schluessel",
 009 (SELECT "tbl_Mitglied"."Vorname" || ' ' || "tbl_Mitglied"."Nachname" ||
     CHAR( 13 ) || CHAR( 10 ) || '(' ||
        CASE WHEN "tbl_Mitglied"."Geschlecht" = 'm' THEN "a"."Posten_m"
        WHEN "tbl_Mitglied"."Geschlecht" = 'w' THEN "a"."Posten_w"
        ELSE "a"."Posten" END || ')'
     FROM "tbl_Mitglied_VorstandPosten", "tbl_Mitglied", "tbl_VorstandPosten"
     AS "a"
     WHERE "tbl_Mitglied_VorstandPosten"."mitID" = "tbl_Mitglied"."ID"
     AND "tbl_Mitglied_VorstandPosten"."vorID" = "a"."ID"
     AND "tbl_Mitglied_VorstandPosten"."EndDat" IS NULL
     AND "a"."ID" = COALESCE ( ( SELECT "Unterschrift1ID" FROM "tbl_Filter"
        WHERE "ID" = COALESCE ( ( SELECT "ID" FROM "viw_ServVar" ), '1' ) ),
     1 ) )
     AS "Unterschrift"
 010 FROM "viw_Mitglieder_komplett" AS "a", "tbl_Einstellungen"
 011 WHERE
 012 "a"."mitID" = COALESCE ( ( SELECT "mitID" FROM "tbl_Filter" WHERE "ID" =
     COALESCE ( ( SELECT "ID" FROM "viw_ServVar" ), '1' ) ), "a"."mitID" )
 013 AND NOT "a"."AusDat" IS NULL
```

Mit dieser Abfrage wird ein Bestätigungsbrief für einen Austritt bereit gestellt. Für den Brief wird im Adressfeld die Anschrift in einer Zeile benötigt (Zeile 2), im Briefkopf aber statt mit Kommatrennung mit einem Zeilenumbruch getrennt untereinander (Zeile 3, Funktion **REPLACE**). Danach werden das Logo und die Bankverbindung aus der "tbl_Einstellungen" gelesen.

Für die Anschrift muss die Anrede entsprechend dem Geschlecht ermittelt werden. Auch hier erfolgt anschließend die Zusammenführung zu einem Block über **CHAR(13) || CHAR(10)** mit einem Zeilenumbruch.

Auch die Anrede (Zeile 7) im Brief selbst ist abhängig vom Geschlecht, das aus "viw_Mitglieder_komplett" ausgelesen wird. Hinzu kommt in dieser Anrede, dass das Datum der Austrittserklärung in die deutschsprachige Schreibweise umgewandelt werden muss. Sonst würde dort ein Datum wie '2022-08-20' erscheinen. Es wird also jeweils Tag, Monat und Jahr ausgelesen und dann mit einem Punkt verbunden. Dabei wird Tag und Monat nicht unbedingt zweistellig erfasst.

Gibt es noch Boote in der Bootshalle oder entliehene Schlüssel, so wird darauf in Zeile 8 hingewiesen. Der Text für Boote wird nur angezeigt, wenn "viw_Mitglieder_komplett"."Boote" für den aktuellen Datensatz nicht leer ist. Entsprechend wird auch beim Schlüssel verfahren. Die Texte für die verzeichneten Boote bzw. die verzeichneten Schlüssel werden durch eine Einrückung mit festem Leerzeichen im Bericht übersichtlicher angezeigt.

Die Unterschrift wird anschließend in einem Feld geleistet, das bereits mit Namen und Funktion der unterschreibenden Person versehen ist.

Die Abfrage ist nicht beschreibbar, da sie auf eine Ansicht Bezug nimmt, die grundsätzlich nicht beschreibbar ist.

qry_Filter_Gruppe

	ID	Nachname	Vorname	GebDat	Gruppe	Geschlecht	AufDat	Schwimmer
▸	3	Müller	Karl	27.02.64	3	m	04.04.21	☑
	4	Müller-Mehl	Maria	13.11.69	3	w	01.01.22	☑
	5	Müller	Isabel	17.07.07	3	w	01.01.22	☑
	6	Müller	Janis	23.10.09	3	m	01.01.22	☑
	7	Schulze	Jolantha	17.09.01	3	w	03.02.22	☑
	8	Querulant	Gisbert	13.11.03	8	m	07.03.22	☐
	9	Schöngeist	Heinz-Herbert	12.07.87	9	m	05.01.21	☑

Telefon_mobil	Photo	AusDat	Alter_Jahr	Alter_momentan	Alter_Beitrag
0123456789	DB_Bilder/Gr		58	58	58
		01.09.22	53	52	52
			15	15	15
0987654321			13	12	13
			21	20	21
			19	18	18
			35	35	35

Datenquelle	
Tabelle:	*tbl_Filter*
Ansicht:	*viw_Gruppe, viw_ServVar*

Datenziel
Formular: *Einzelmessung_System*

```
001 SELECT *
002 FROM "viw_Gruppe"
003 WHERE "ID" = COALESCE ( ( SELECT "mitID" FROM "tbl_Filter" WHERE "ID" =
    COALESCE ( ( SELECT "ID" FROM "viw_ServVar" ), '1' ) ), "ID" )
```

Hier wird lediglich die "viw_Gruppe" nach dem Eintrag "mitID" in "tbl_Filter" gefiltert. Ist dort ein Eintag vorhanden, so wird nur der eine Datensatz der Mitglieds aus ""viw_Gruppe" angezeigt. Ist der Eintrag leer, so werden alle Mitglieder angezeigt.

Die Abfrage ist nicht beschreibbar, da eine Ansicht grundsätzlich nicht beschreibbar ist.

qry_Filter_Mitglied

	ID	Nachname	Vorname	Geschlecht	GebDat	AufDat	Schwimmer
	3	Müller	Karl	m	27.02.64	04.04.21	☑
▸+	<Auto						☐

AusDat	Nichtdruck	Telefon_mobil	E-Mail	Gruppe_mitID	Photo
	☐	0123456789			DB_Bilder/
	☐				

Datenquelle	
Tabelle:	*tbl_Mitglied, tbl_Filter*
Ansicht:	*viw_ServVar*

Datenziel	
Formular:	*frm_Boot, frm_Mitglied, frm_Schluessel*

```
001 SELECT *
002 FROM "tbl_Mitglied"
003 WHERE "ID" = COALESCE ( ( SELECT "mitID" FROM "tbl_Filter" WHERE "ID" =
    COALESCE ( ( SELECT "ID" FROM "viw_ServVar" ), '1' ) ), "ID" )
```

Hier wird lediglich die "tbl_Mitglied" nach dem Eintrag "mitID" in "tbl_Filter" gefiltert. Ist dort ein Eintag vorhanden, so wird nur der eine Datensatz angezeigt. Ist der Eintrag leer, so werden alle Mitglieder angezeigt.

Die Abfrage ist beschreibbar, da die Primärschlüssel beider betroffenen Tabellen in der Abfrage enthalten sind.

qry_Filter_Vorstand

	ID	mitID	vorID	WahlDat	EndDat
	3	7	2	31.01.22	
	8	3	4	31.01.22	
▷+	<Auto				

Datenquelle	
Tabelle:	*tbl_Mitglied_VorstandPosten, tbl_Filter*
Ansicht:	*viw_ServVar*

Datenziel	
Formular:	*frm_Vorstand*

```
001 SELECT *
002 FROM "tbl_Mitglied_VorstandPosten"
003 WHERE "WahlDat" = COALESCE ( ( SELECT "WahlDat" FROM "tbl_Filter" WHERE
    "ID" = COALESCE ( ( SELECT "ID" FROM "viw_ServVar" ), '1' ) ), "WahlDat" )
```

Diese Abfrage geht es lediglich darum, aus der entsprechenden Tabelle "tbl_Mitglied_Vorstand-Posten" die an dem Wahldatum (neu) gewählten Vorstandsmitglieder anzuzeigen. Der Screenshot zeigt hier nur die Tabelle mit den Fremdschlüssel. Das deutet an, dass diese Abfrage als Grundlage in einem Formular benötigt wird, bei dem die Fremdschlüssel dann durch Listenfelder mit Inhalten dargestellt werden können.

Die Abfrage ist beschreibbar, da die Primärschlüssel beider betroffenen Tabellen in der Abfrage enthalten sind.

qry_Forms

	Title	Form
▶	Beitrag	frm_Beitrag
	Berichte	frm_Berichte
	Boote	frm_Boot
	Einstellungen	frm_Einstellungen
	Mitglieder	frm_Mitglied
	Schlüssel	frm_Schluessel
	Vorstand	frm_Vorstand

Datenquelle

Tabelle: *tbl_Einstellungen*

Datenziel

Formular: *Einzelmessung_System, frm_Berichte, frm_Boot, frm_Einstellungen, frm_Mitglied, frm_Schluessel, frm_Vorstand*

```
001 SELECT 'Mitglieder' AS "Title", 'frm_Mitglied' AS "Form" FROM
    "tbl_Einstellungen" WHERE "ID" = TRUE
002 UNION
003 SELECT 'Vorstand' AS "Title", 'frm_Vorstand' AS "Form" FROM
    "tbl_Einstellungen" WHERE "ID" = TRUE
004 UNION
005 SELECT 'Einstellungen' AS "Title", 'frm_Einstellungen' AS "Form" FROM
    "tbl_Einstellungen" WHERE "ID" = TRUE
006 UNION
007 SELECT 'Berichte' AS "Title", 'frm_Berichte' AS "Form" FROM
    "tbl_Einstellungen" WHERE "ID" = TRUE
008 UNION
009 SELECT 'Boote' AS "Title", 'frm_Boot' AS "Form" FROM "tbl_Einstellungen"
    WHERE "ID" = TRUE AND "Bootstaenderverwaltung" = TRUE
010 UNION
011 SELECT 'Schlüssel' AS "Title", 'frm_Schluessel' AS "Form" FROM
    "tbl_Einstellungen" WHERE "ID" = TRUE AND "Schluesselverwaltung" = TRUE
012 UNION
013 SELECT 'Beitrag' AS "Title", 'frm_Beitrag' AS "Form" FROM
    "tbl_Einstellungen" WHERE "ID" = TRUE AND "Beitragsverwaltung" = TRUE
```

Mit dieser Abfrage wird von einem Formular zum anderen navigiert. Die anzuzeigenden Formulare werden danach ausgewählt, ob sie in den Einstellungen ("tbl_Einstellungen") auch angewählt wurden.

In der ersten Spalte stehen die anzuzeigenden Namen, in der zweiten Spalte der Name des Formulars, der so an das Makro zum Öffnen des Formular weiter gegeben wird. Die Sortierung erfolgt bei einer **UNION**-Verknüpfung nach der ersten Spalte, sofern nichts anderes festgelegt wurde.

qry_Jubilaeum

	ID	Name	Mitgliedsjahre	Logo	Vereinsname	Ort	Posten_1	Posten_2
▶	11	Helmut-Wilhelm	für 50-jährige Mitg	<OBJECT>	LibreOffice Base	Pusemu	Janis Müller(V	Jolantha Schu
	12	Annemarie Oldie	für 40-jährige Mitg	<OBJECT>	LibreOffice Base	Pusemu	Janis Müller(V	Jolantha Schu
	13	Irina Oldie	für 25-jährige Mitg	<OBJECT>	LibreOffice Base	Pusemu	Janis Müller(V	Jolantha Schu

Datenquelle

Tabelle: *tbl_Mitglied, tbl_Einstellungen, tbl_Mitglied_VorstandPosten, tbl_VorstandPosten*

Datenziel

Bericht: *rpt_Jubilaeum_Ehrenurkunde*

```
001 SELECT "tbl_Mitglied"."ID",
002 "tbl_Mitglied"."Vorname" || ' ' || "tbl_Mitglied"."Nachname" AS "Name",
003 'für ' || DATEDIFF( 'YY', "tbl_Mitglied"."AufDat", CURRENT_DATE ) || '-
    jährige Mitgliedschaft' AS "Mitgliedsjahre",
004 "tbl_Einstellungen"."Logo",
005 "tbl_Einstellungen"."Vereinsname",
006 "tbl_Einstellungen"."Ort" || ', den' AS "Ort",
007 "b"."Posten" AS "Posten_1",
008 "c"."Posten" AS "Posten_2",
009 DATEDIFF( 'YY', "tbl_Mitglied"."AufDat", CURRENT_DATE ) AS "Jahre"
010 FROM "tbl_Mitglied",
011 "tbl_Einstellungen",
012 ( SELECT "tbl_Mitglied"."Vorname" || ' ' || "tbl_Mitglied"."Nachname" ||
    CHAR( 13 ) || CHAR( 10 ) || '(' ||
    CASE WHEN "tbl_Mitglied"."Geschlecht" = 'm' THEN "a"."Posten_m"
    WHEN "tbl_Mitglied"."Geschlecht" = 'w' THEN "a"."Posten_w"
    ELSE "a"."Posten" END || ')' AS "Posten"
    FROM "tbl_Mitglied_VorstandPosten", "tbl_Mitglied", "tbl_VorstandPosten"
    AS "a"
    WHERE "tbl_Mitglied_VorstandPosten"."mitID" = "tbl_Mitglied"."ID"
    AND "tbl_Mitglied_VorstandPosten"."vorID" = "a"."ID"
    AND "tbl_Mitglied_VorstandPosten"."EndDat" IS NULL
    AND "a"."ID" = COALESCE ( ( SELECT "Unterschrift1ID" FROM "tbl_Filter"
       WHERE "ID" = COALESCE ( ( SELECT "ID" FROM "viw_ServVar" ), '1' ) ),
       1 ) )
    AS "b",
013 ( SELECT "tbl_Mitglied"."Vorname" || ' ' || "tbl_Mitglied"."Nachname" ||
    CHAR( 13 ) || CHAR( 10 ) || '(' ||
    CASE WHEN "tbl_Mitglied"."Geschlecht" = 'm' THEN "a"."Posten_m"
    WHEN "tbl_Mitglied"."Geschlecht" = 'w' THEN "a"."Posten_w"
    ELSE "a"."Posten" END || ')' AS "Posten"
    FROM "tbl_Mitglied_VorstandPosten", "tbl_Mitglied", "tbl_VorstandPosten"
    AS "a"
    WHERE "tbl_Mitglied_VorstandPosten"."mitID" = "tbl_Mitglied"."ID"
    AND "tbl_Mitglied_VorstandPosten"."vorID" = "a"."ID"
    AND "tbl_Mitglied_VorstandPosten"."EndDat" IS NULL
    AND "a"."ID" = COALESCE ( ( SELECT "Unterschrift2ID" FROM "tbl_Filter"
       WHERE "ID" = COALESCE ( ( SELECT "ID" FROM "viw_ServVar" ), '1' ) ),
       2 ) )
    AS "c"
014 WHERE "AusDat" IS NULL
015 AND "Jahre" = 70
016 OR "Jahre" = 60
017 OR "Jahre" = 50
018 OR "Jahre" = 40
```

```
019 OR "Jahre" = 25
020 ORDER BY "Mitgliedsjahre" DESC
```

Mit Hilfe dieser Abfrage soll ein Bericht gestartet werden, der zum Ausdruck von Urkunden bei langjähriger Mitgliedschaft dient. Die erste Spalte enthält die "ID" aus der "tbl_Mitglied". Danach folgt der Name, wobei hier fest davon ausgegangen wird, dass Vor- und Nachname bei langjährigen Mitgliedern bekannt ist. In Zeile 3 wird dann der Grundtext festgelegt, in den die Dauer der Mitgliedschaft eingebunden ist. Diese Dauer der Mitgliedschaft wird über Zeile 9 (Definition der Jahre mit dem Alias "Jahre", nicht im Screenshot gezeigt) und die Zeilen 15 bis 19 (Jubiläumsjahre) auf bestimmte Jahre eingeschränkt.

In den Zeilen 4, 5 und 6 werden Informationen aus der Tabelle "tbl_Einstellungen" geholt, die zur grafischen Gestaltung ("Logo"), als Vereinsbezeichnung oder zur Gestaltung des Datumseintrags sinnvoll sind.

Am Aufwendigsten ist es, in den Unterschriftszeilen die beiden Vorstandsmitglieder mit Posten zu ermitteln und als "Posten_1" und "Posten_2" für die Unterschrift nutzen zu können. Deren Ermittlung in Zeile 12 und 13 ist nahezu identisch. In Zeile 12 wird die erste Beschriftung zusammengesetzt, die auf dem Eintag in "tbl_Filter" oder der "ID" = '1' aus den Vorstandsposten beruht. Entsprechend wird in Zeile 13 das gleiche für die zweite Beschriftung vollzogen. Über die **`CASE WHEN … THEN … ELSE`** – Formulierung wird ermittelt, welche Vorstandspostenbezeichnung aus der Liste der Mitglieder genommen werden soll. Ist weder 'w' noch 'm' in der Mitgliedstabelle verzeichnet, so wird einfach die Standardbezeichnung "tbl_VorstandsPosten"."Posten" gewählt.

Eine Verknüpfung der Tabellen ist nicht erforderlich, da die "tbl_Einstellungen" nur einen Datensatz enthält. Dadurch enthält auch die Verbindung mit "tbl_Mitglied" nur genau so viele Datensätze, wie "tbl_Mitglied" liefern kann.

Die Abfrage ist nicht beschreibbar, wird aber auch nicht im direkten SQL-Modus ausgeführt, so dass sie zur Filterung und Sortierung geeignet ist.

qry_Mitglied_Konto

	ID	Konto
	3	IBAN: DE47 1103 5678 1290 2345 12 bei: Stadtsparkasse Rheine, BIC: WELADED1RHN Kontoinhaber: Großkopf, Robert
	4	keine eigene Kontoverbindung vorhanden. Kontoverbindung von: Müller, Karl
	5	keine eigene Kontoverbindung vorhanden. Kontoverbindung von: Müller, Karl
	6	keine eigene Kontoverbindung vorhanden. Kontoverbindung von: Müller, Karl
	7	keine eigene Kontoverbindung vorhanden. Kontoverbindung von: Müller, Karl
	8	IBAN: DE23 4567 8901 2345 6789 02 bei: Volksbank Münsterland Nord eG, BIC: GENODEM1IBB Kontoinhaber: Querulant, Gisbert
	9	

Datenquelle
Tabelle: *tbl_Konto, tbl_Bank, tbl_Mitglied*

Datenziel
Formular: *frm_Mitglied*

```
001 SELECT "a"."ID",
002 COALESCE ( 'IBAN: ' || "tbl_Konto"."IBAN" || CHAR( 13 ) || CHAR( 10 ) ||
        'bei: ' || "tbl_Bank"."Bank" || ', BIC: ' || "tbl_Bank"."BIC" ||
        CHAR( 13 ) || CHAR( 10 ) || 'Kontoinhaber: ' ||
        COALESCE ( "tbl_Konto"."Nachname" ||
           COALESCE ( ', ' || "tbl_Konto"."Vorname", '' ),
        "a"."Nachname" || COALESCE ( ', ' || "a"."Vorname", '' )
003 ), 'keine eigene Kontoverbindung vorhanden.' || CHAR( 13 ) || CHAR( 10 )
        || 'Kontoverbindung von: ' ||
        ( SELECT "Nachname" || COALESCE ( ', ' || "Vorname", '' )
        FROM "tbl_Mitglied" WHERE "ID" = "a"."Gruppe_mitID" ) )
    AS "Konto"
004 FROM "tbl_Mitglied" AS "a"
005 LEFT JOIN "tbl_Konto" ON "a"."ID" = "tbl_Konto"."mitID"
006 LEFT JOIN "tbl_Bank" ON "tbl_Konto"."banID" = "tbl_Bank"."ID"
```

Hier werden sämtliche Kontoverbindungen aufgelistet. Bei Gruppenmitgliedern, deren Beitrag von der Kontoverbindung eines anderen Mitglieds erhoben wird, wird aufgelistet, welche Kontoverbindung genutzt werden kann (Zeile 3). Für den unteren Datensatze im Screenshot ist noch keine Kontoverbindung registriert. Bei Datensätzen, die eine Kontoverbindung enthalten, wird die komplette Verbindung mit Kontoinhaber aufgelistet.

Bei der Zusammenführung der Daten mittels **||** taucht hier immer wieder die Kombination **CHAR(13) || CHAR(10)** auf. Sie dient dazu, einen Zeilenumbruch zu erzeugen.

Die Abfrage ist nicht beschreibbar, wird aber auch nicht im direkten SQL-Modus ausgeführt, so dass sie zur Filterung und Sortierung geeignet ist.

qry_Mitgliedsdauer

	ID	Nachname	Vorname	AufDat	Mitgliedsjahre
	11	Oldie	Helmut-Wilhelm	01.07.72	50
	12	Oldie	Annemarie	12.08.82	40
	13	Oldie	Irina	01.03.97	25
▷+	<Auto				

Datenquelle
Tabelle: *tbl_Mitglied*

Datenziel
Ansicht: *rpt_Mitgliedsdauer*

```
001 SELECT "ID",
002 "Nachname",
003 "Vorname",
004 "AufDat",
005 DATEDIFF( 'YY', "AufDat", CURRENT_DATE ) AS "Mitgliedsjahre"
006 FROM "tbl_Mitglied"
007 WHERE "AufDat" IS NOT NULL
```

```
008 AND ( "AusDat" IS NULL OR "AusDat" > CURRENT_DATE )
009 AND DATEDIFF( 'YY', "AufDat", CURRENT_DATE ) > 9
010 ORDER BY "AufDat" ASC
```

Hier wird eine Übersicht nach der Dauer der Mitgliedschaft erzeugt. Die Mitgliedsjahre werden einfach als Jahresunterschied zum Aufnahmedatum berechnet. Dabei dürfen die Mitglieder nicht ausgetreten sein (Zeile 8). Auch sollen nur Mitglieder aufgelistet werden, die wenigstens 9 Jahre im Verein sind. Die Übersicht wird also auf langjährige Mitglieder eingegrenzt.

Die Abfrage ist beschreibbar, da der Primärschlüssel "ID" der betroffenen Tabelle in der Abfrage enthalten ist.

qry_Schluessel_Uebersicht

	SNr	Name
▸	GS 1	Müller, Karl
	GS 2	Müller-Mehl, Maria
	GS 2	Oldie, Annemarie
	GS 3	Müller, Karl

Datenquelle
Tabelle: *tbl_MitgliedSchluessel, tbl_Mitglied*

Datenziel
Bericht: *rpt_Schluesseluebersicht*

```
001 SELECT "tbl_Mitglied_Schluessel"."SNr",
002 "tbl_Mitglied"."Nachname" || ', ' || "tbl_Mitglied"."Vorname" AS "Name"
003 FROM "tbl_Mitglied_Schluessel", "tbl_Mitglied"
004 WHERE "tbl_Mitglied_Schluessel"."mitID" = "tbl_Mitglied"."ID"
005 AND "tbl_Mitglied_Schluessel"."RueckDat" IS NULL
006 AND NOT "tbl_Mitglied_Schluessel"."Verlust" = TRUE
007 ORDER BY "tbl_Mitglied_Schluessel"."SNr" ASC, "Name" ASC
```

Mit der Abfrage werden alle ausgegebenen Schlüssel und die Nahmen der entsprechenden Entleiher aufgelistet. Die Schlüssel dürfen nach der Bedingung von Zeile 5 nicht mit einem Rückgabedatum "RueckDat" versehen sein. Außerdem dürfen sie nicht als verloren gekennzeichnet sein (Zeile 6). Die Liste wird sortiert nach den Schlüsselnummern "SNr" und dann nach den Namen.

Die Abfrage ist nicht beschreibbar, wird aber auch nicht im direkten SQL-Modus ausgeführt, so dass sie zur Filterung geeignet ist. Eine Änderung der Sortierung ist nur bedingt möglich, da die bereits in der Abfrage enthalten ist. Die Sortierung für die "SNr" kann lediglich durch die GUI umgekehrt werden. Die Sortierung nach dem Feld "Name" greift nur dann, wenn die "SNr" bei Datensätzen gleich ist, im Screenshot also bei den beiden Schlüsseln 'GS 2'.

qry_Vorstand_aktuell

	ID	Posten	Nachname	Vorname	Telefon	WahlDat
▸	1	Vorsitzende(r)	Müller	Janis	05971/424	31.01.21
	2	Geschäftsführer(in)	Schulze	Jolantha		31.01.22
	3	Schatzmeister(in)				
	4	stellv. Vorsitzende(r)	Müller	Karl	05971/424	31.01.22
	5	1. Jugendwart(in)				
	6	2. Jugendwart(in)				
	7	Sportwart(in)				
	8	stellv. Sportwart(in)				
	9	Protokollführer(in)				

Datenquelle	
Tabelle:	*tbl_VorstandPosten, tbl_Mitglied_VorstandPosten*
Ansicht:	*viw_Mitglieder_komplett*

Datenziel	
Bericht:	*rpt_Vorstand_aktuell*

```
001 SELECT "tbl_VorstandPosten"."ID",
002 "tbl_VorstandPosten"."Posten",
003 "a"."Nachname",
004 "a"."Vorname",
005 "a"."Telefon_fest" AS "Telefon",
006 "tbl_Mitglied_VorstandPosten"."WahlDat"
007 FROM "tbl_VorstandPosten"
008 LEFT JOIN "tbl_Mitglied_VorstandPosten" ON
    "tbl_Mitglied_VorstandPosten"."vorID" = "tbl_VorstandPosten"."ID"
009 LEFT JOIN "viw_Mitglieder_komplett" AS "a" ON
    "tbl_Mitglied_VorstandPosten"."mitID" = "a"."mitID"
010 WHERE "tbl_Mitglied_VorstandPosten"."EndDat" IS NULL
011 ORDER BY "tbl_VorstandPosten"."ID" ASC
```

Mit dieser Abfrage werden alle aktuellen Vorstandsmitglieder aufgelistet. Als Zusatzinformationen zu den Namen ist hier die Telefonnummer aus dem Festnetz und das Wahldatum enthalten.

Die Datensätze aus "tbl_VorstandPosten" werden komplett angezeigt, auch wenn nicht alle Posten besetzt sind. Alle anderen Datenquellen sind mit dieser Tabelle über einen **LEFT JOIN** verbunden. Das Auslesen der Mitgliedsdaten erfolgt anhand der ausführlichen Auflistung in "viw_Mitglieder_komplett".

Die Abfrage ist nicht beschreibbar, wird aber auch nicht im direkten SQL-Modus ausgeführt, so dass sie zur Filterung geeignet ist. Eine Änderung der Sortierung ist nicht möglich, da die bereits in der Abfrage enthalten ist. Die Sortierung für die "ID" kann lediglich durch die GUI umgekehrt werden.

qry_Vorstand_aktuell_lt_Satzung

	ID	Posten	Name	Telefon	E-Mail	Anschrift
▸	1	Vorsitzende(r)	Müller, Janis	fest: 05971/4	flitzepiepe@exa	Unter dem Baum 42, 987
	2	Geschäftsführer(in)	Schulze, Jolantha	mobil: 09080		Konsumrennbahn 1a, 02
	3	Schatzmeister(in)				
	4	stellv. Vorsitzende(r)	Müller, Karl	fest: 05971/4		Unter dem Baum 42, 987
	5	1. Jugendwart(in)				
	6	2. Jugendwart(in)				
	7	Sportwart(in)				
	8	stellv. Sportwart(in)				
	9	Protokollführer(in)				

Datenquelle	
Tabelle:	*tbl_VorstandPosten, tbl_Mitglied_VorstandPosten*
Ansicht:	*viw_Mitglieder_komplett*

Datenziel	
Bericht:	*rpt_Vorstand_aktuell_lt_Satzung*

```
001 SELECT "tbl_VorstandPosten"."ID",
002 "tbl_VorstandPosten"."Posten",
003 "a"."Nachname" || COALESCE ( ', ' || "a"."Vorname", '' ) AS "Name",
004 COALESCE ( 'fest: ' || "a"."Telefon_fest" || ' ', '' ) || COALESCE
    ( 'mobil: ' || "a"."Telefon_mobil", '' ) AS "Telefon",
005 "a"."E-Mail",
006 "a"."Straße_Nr" || ', ' || "a"."PLZ_Ort" AS "Anschrift"
007 FROM "tbl_VorstandPosten"
008 LEFT JOIN "tbl_Mitglied_VorstandPosten" ON
    "tbl_Mitglied_VorstandPosten"."vorID" = "tbl_VorstandPosten"."ID"
009 LEFT JOIN "viw_Mitglieder_komplett" AS "a" ON
    "tbl_Mitglied_VorstandPosten"."mitID" = "a"."mitID"
010 WHERE "tbl_Mitglied_VorstandPosten"."EndDat" IS NULL
011 AND "tbl_VorstandPosten"."Posten_laut_Satzung" = TRUE
012 ORDER BY "tbl_VorstandPosten"."ID" ASC
```

Der Vorstand lt. Satzung besteht aus den Mitgliedern, die auch regelmäßig zu Vorstandssitzungen eingeladen werden. Deshalb sind hier nicht nur Namen sondern auch die Kontaktdaten wie Telefon, E-Mail und Anschrift aufgelistet.

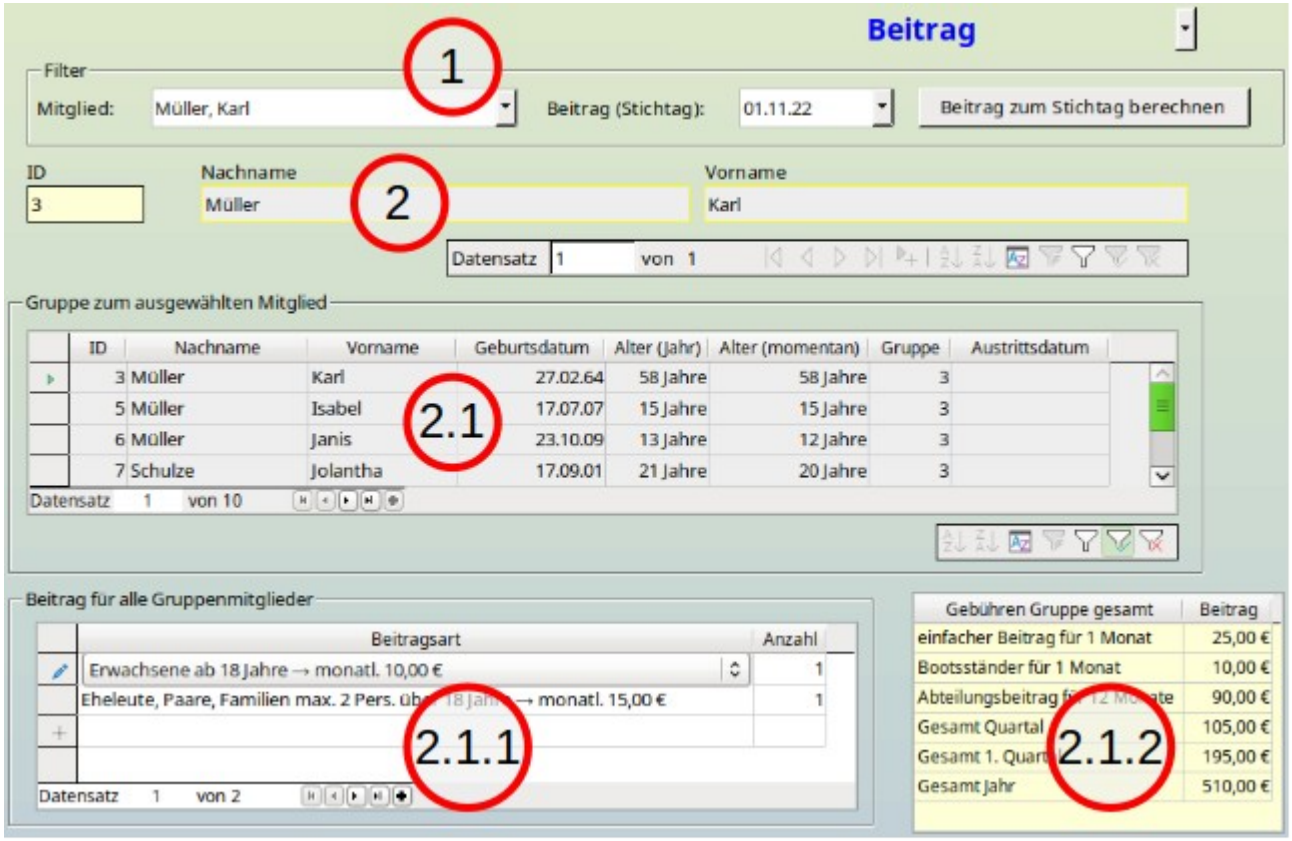
Die Datensätze aus "tbl_VorstandPosten" werden komplett angezeigt, auch wenn nicht alle Posten besetzt sind. Alle anderen Datenquellen sind mit dieser Tabelle über einen **`LEFT JOIN`** verbunden. Das Auslesen der Mitgliedsdaten erfolgt anhand der ausführlichen Auflistung in "viw_Mitglieder_komplett".

Es werden nach den Bedingungen aus Zeile `10` und Zeile `11` nur die Vorstandsmitglieder angezeigt, bei denen in "tbl_Mitglied_VorstandPosten" kein "EndDat" steht und in "tbl_VorstandPosten" der Eintrag für "Posten_laut_Satzung" ausgewählt wurde, also auf **`TRUE`** steht.

Die Abfrage ist nicht beschreibbar, wird aber auch nicht im direkten SQL-Modus ausgeführt, so dass sie zur Filterung geeignet ist. Eine Änderung der Sortierung ist nicht möglich, da die bereits in der Abfrage enthalten ist. Die Sortierung für die "ID" kann lediglich durch die GUI umgekehrt werden.

Formulare

frm_Beitrag



1	frmFilter (Abfrage: *qry_Filter*)				
2	frmMitglied (Abfrage: *qry_Filter_Gruppe*)	**2.1**	frmGruppe (Ansicht: *viw_Gruppe*)	**2.1.1**	FrmGruppeBeitrag (Tabelle: *tbl_Gruppe_Beitrag*)
				2.1.2	FrmBeitragsuebersicht (Ansicht: *viw_Beitrag*)
1	Listenfeld Formularauswahl (Abfrage: *qry_Forms*)				

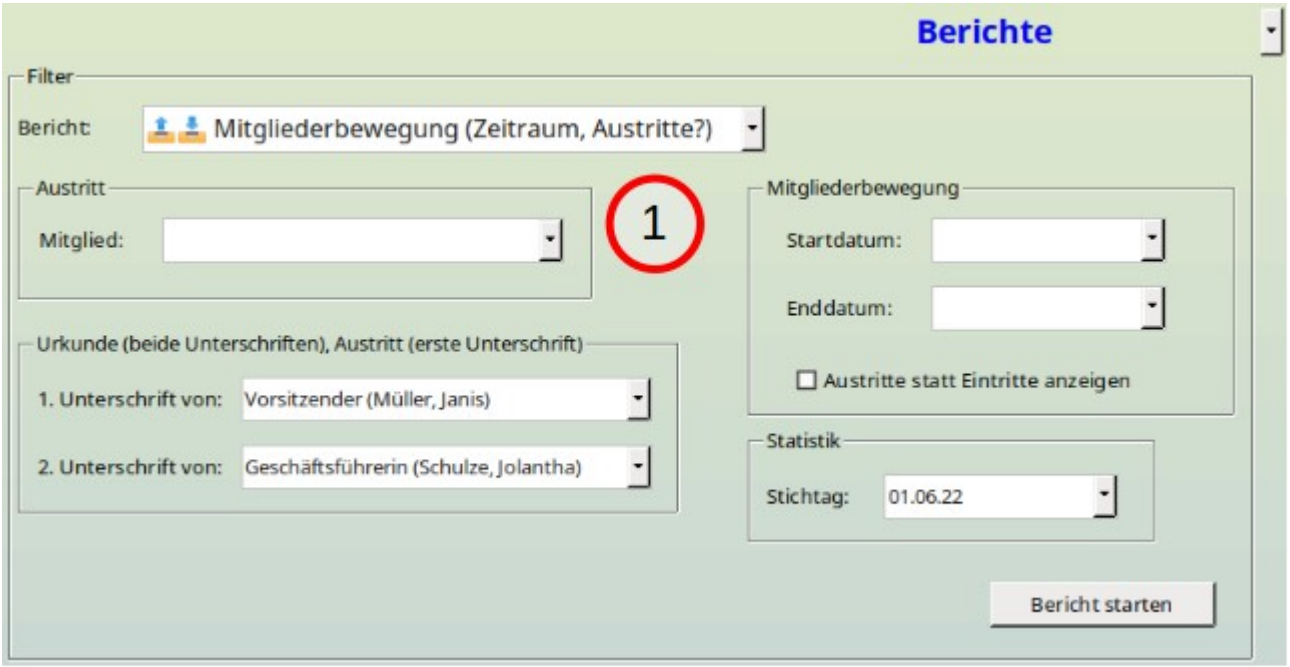
Makros in Formulareigenschaften		
1	Beim Laden	Navigation.*FormStart*
2.1.1	Nach der Datensatzaktion	Speichern.*SubscriptionChanged*
Makros in Feldeigenschaften		
1	Listenfeld Formularauswahl (Status geändert)	Navigation.*Navigation*
1	Listenfeld Mitglied (Status geändert)	Filter.*Filtern*
1	Schaltfläche Beitrag zum Stichtag berechnen (Aktion ausführen)	Speichern.*SubscriptionSave*

SQL-Code in Listenfeldern		
1	Mitglied *tbl_Mitglied*	001 SELECT "Nachname"||COALESCE(', '||"Vorname",'') AS "Name", "ID" 002 FROM "tbl_Mitglied" 003 ORDER BY "Name"
2.1.1	Beitragsart *tbl_Beitrag*	001 SELECT "BeitrArt" || ' → monatl. ' || REPLACE("BeitrHoehe", '.', ',') || ' €' AS "Beitrag", "ID" 002 FROM "tbl_Beitrag" 003 WHERE ("BeitrBezug" = 'Mit' OR "BeitrBezug" = 'MitG')

Formularfilter	
2.1	001 ("AusDat" > EXTRACT(YEAR FROM CURRENT_DATE)||'-12-31' 002 OR "AusDat" IS NULL)

Mit diesem Formular wird die Beitragsermittlung durchgeführt. Der Beitrag kann berechnet werden. Falls dabei aber Fehler auftauchen oder eine Korrektur gewünscht ist (Beitragsermäßigung), dann kann das im Tabellenkontrollfeld links unten (2.1.1) erledigt werden.

frm_Berichte



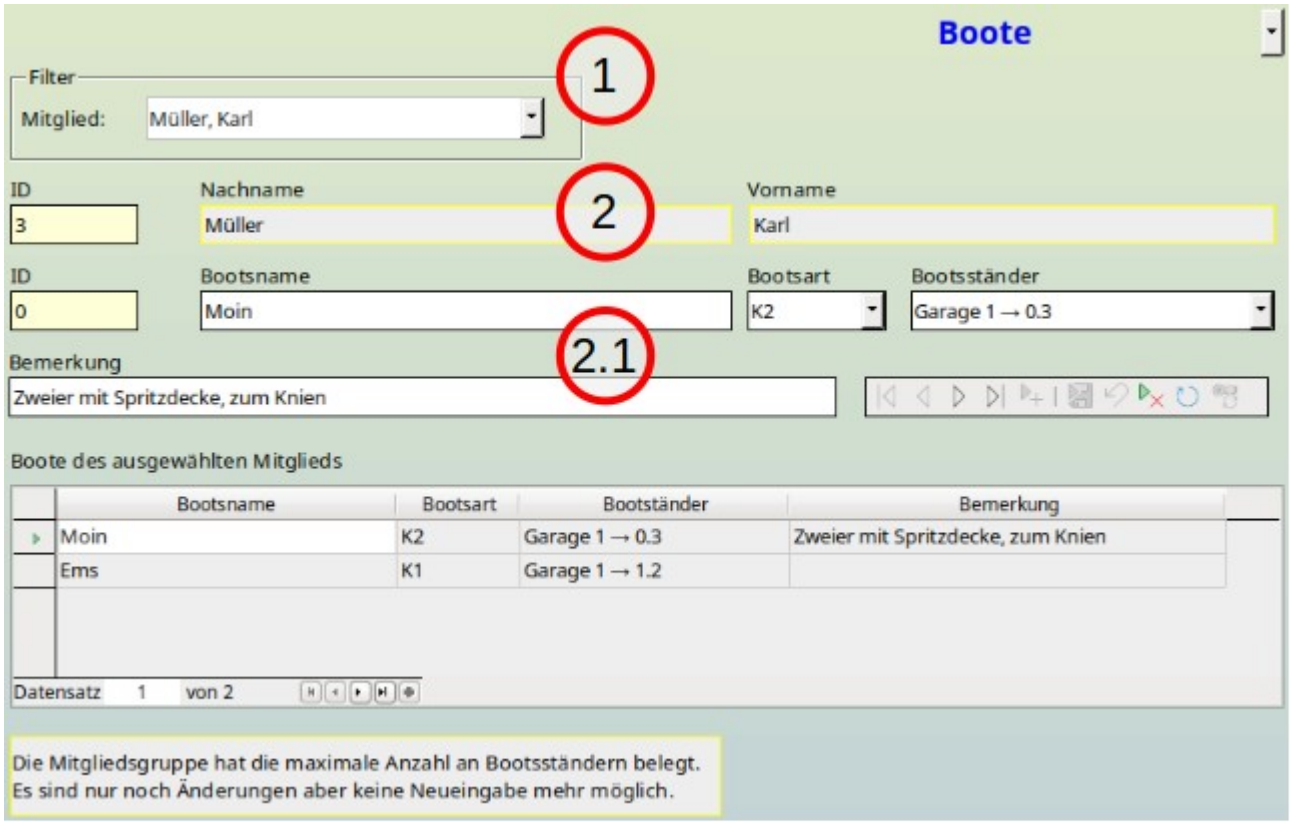
1	frmFilter (Abfrage: *qry_Filter*)
1	Listenfeld Formularauswahl (Abfrage: *qry_Forms*)

Makros in Formulareigenschaften		
1	Beim Laden	Navigation.*FormStart*
Makros in Feldeigenschaften		
1	Listenfeld Formularauswahl (Status geändert)	Navigation.*Navigation*
1	Listenfeld Mitglied (Status geändert)	Filter.*Filtern*
1	Listenfeld Bericht (Status geändert)	Filter.*Filtern*
1	Schaltfläche Bericht starten (Aktion ausführen)	Druck.*ReportStart*

SQL-Code in Listenfeldern		
1	Bericht *viw_Reports*	001 SELECT "Title", "Report" 002 FROM "viw_Reports"
1	Austritt Mitglied *tbl_Mitglied*	001 SELECT "Nachname" || COALESCE (', ' || "Vorname", '') AS "Name", "ID" 002 FROM "tbl_Mitglied" 003 WHERE NOT "AusDat" IS NULL ORDER BY "Name" ASC
1	1. Unterschrift von (2. Unterschrift entsprechend) *tbl_Mitglied_VorstandPosten* *tbl_Vorstand-Posten* *tbl_Mitglied*	001 SELECT CASE WHEN "tbl_Mitglied"."Geschlecht" = 'm' 002 THEN "tbl_VorstandPosten"."Posten_m" 003 WHEN "tbl_Mitglied"."Geschlecht" = 'w' 004 THEN "tbl_VorstandPosten"."Posten_w" 005 ELSE "tbl_VorstandPosten"."Posten" END ||' ('||"tbl_Mitglied"."Nachname"||COALESCE(', '||"tbl_Mitglied"."Vorname",'')||')' AS "Posten", 006 "tbl_VorstandPosten"."ID" 007 FROM "tbl_Mitglied_VorstandPosten", "tbl_Mitglied", "tbl_VorstandPosten" 008 WHERE "tbl_Mitglied_VorstandPosten"."mitID" = "tbl_Mitglied"."ID" 009 AND "tbl_Mitglied_VorstandPosten"."vorID" = "tbl_VorstandPosten"."ID" 010 AND "EndDat" IS NULL 011 ORDER BY "tbl_VorstandPosten"."ID"

Dieses Formular dient zum Start sämtlicher Berichte: Für manche Berichte sind noch Voreinstellungen erforderlich. Soll z. B. die vorausgewählte «Mitgliederbewegung» ausgegeben werden, so sind Einstellungen in der Gruppe «Mitgliederbewegung» notwendig. Dabei handelt es sich um das «Startdatum» und das «Enddatum» für den Bericht. Sollen statt der Beitritte die Austritte angezeigt werden, so kann das durch einen Klick auf das entsprechende Markierfeld erfolgen.

frm_Boot



1	frmFilter (Abfrage: *qry_Filter*)		
2	frmMitglied (Abfrage: *qry_Filter_Mitglied*)	**2.1**	frmGruppe (Tabelle: *tbl_Boot*)
1	Listenfeld Formularauswahl (Abfrage: *qry_Forms*)		

Makros in Formulareigenschaften		
1	Beim Laden	Navigation.*FormStart*
2	Nach dem Datensatzwechsel	Eingabekontrolle.*NewBoatAllowed*
2.1	Nach der Datensatzaktion	Eingabekontrolle.*NewBoatAllowed*
2.1	Nach dem Datensatzwechsel	Eingabekontrolle.*SelectBoat*
Makros in Feldeigenschaften		
1	Listenfeld Formularauswahl (Status geändert)	Navigation.*Navigation*
1	Listenfeld Mitglied (Status geändert)	Filter.*Filtern*

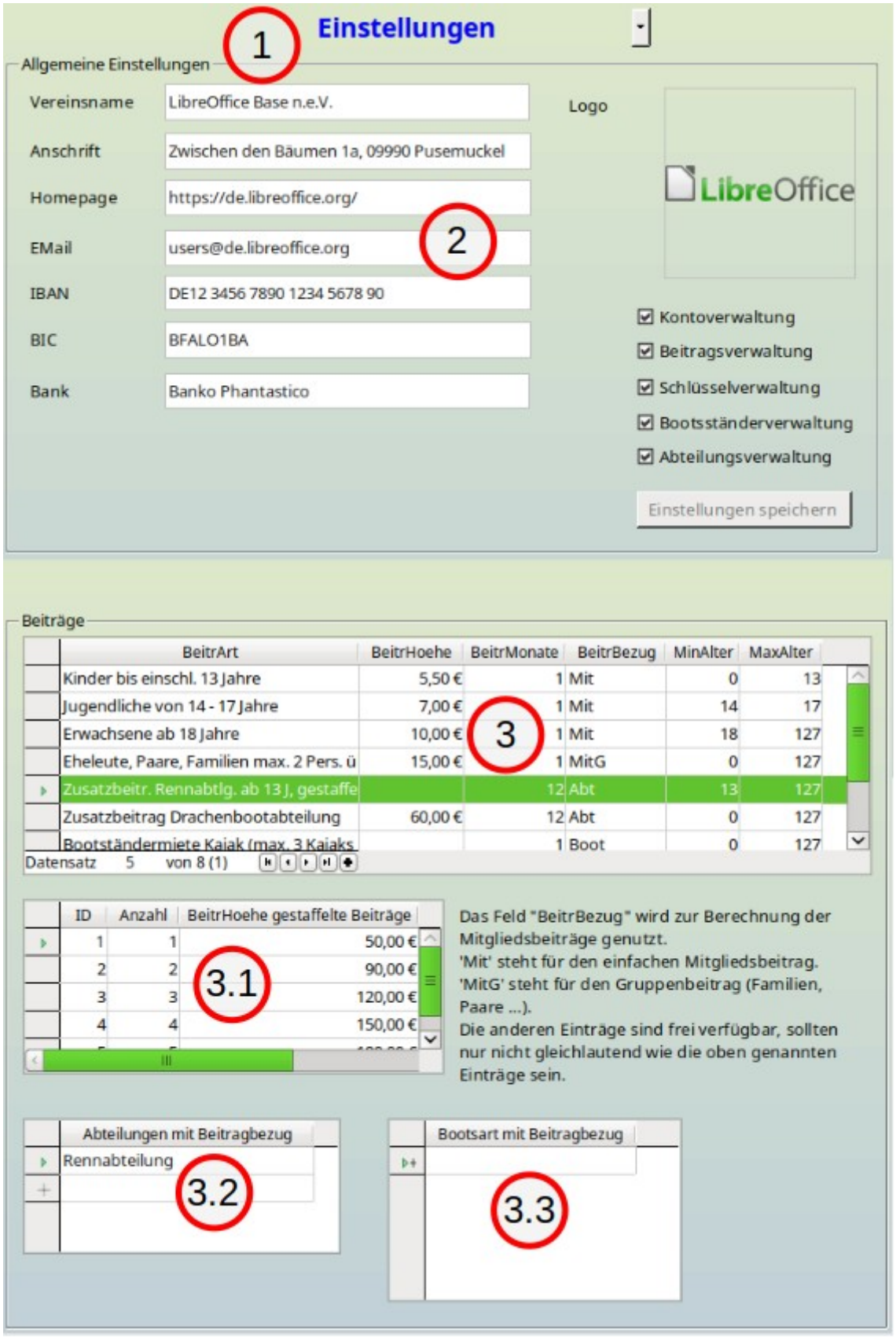
SQL-Code in Listenfeldern		
1	Mitglied *tbl_Mitglied*	001 SELECT "Nachname"||COALESCE(', '||"Vorname",'') AS "Name", "ID" 002 FROM "tbl_Mitglied" 003 ORDER BY "Name"
2.1	Bootsart (2*) *tbl_BootArt*	001 SELECT "Bootsart", "ID" 002 FROM "tbl_BootArt" 003 ORDER BY "Bootsart" ASC

2.1	Bootsständer (2*) *tbl_BootStaender* *tbl_BootHalle*	001 SELECT "tbl_BootHalle"."Halle_lang"||' → '||"tbl_BootStaender"."BootStaend" AS "Bootstaender", "tbl_BootStaender"."ID" 002 FROM "tbl_BootStaender", "tbl_BootHalle" 003 WHERE "tbl_BootStaender"."bo_hID" = "tbl_BootHalle"."ID"

Hier geht es darum, Booten in den Hallen entsprechende Lagerplätze zuzuweisen. Dabei sollen nur die Lagerplätze anwählbar sein, die wirklich frei sind. Deswegen ist das Listenfeld für die Auswahl der Bootsständer entsprechend konfiguriert und wird bei jedem Datensatzwechsel aktualisiert.

Auch eine Beschränkung der Bootsplätze ist möglich. So wird bei der Datenbank davon ausgegangen, dass pro Beitragszahler maximal ein Canadier (größeres Boot mit Stechpaddeln) oder drei Kajaks (kleinere Boote mit Doppelpaddeln) gelagert werden können. In dem obigen Formular erscheint deswegen die Meldung, dass die maximale Anzahl an Bootsständern ausgeschöpft ist. Zwei Boote sind dem ausgewählten Mitglied zugewiesen. Es wird also noch ein drittes Boot bei einem anderen Mitglied der Gruppe registriert sein.

frm_Einstellungen



1	frmFilter (Abfrage: *qry_Filter*)		
2	frmEinstellungen (Tabelle: *tbl_Einstellungen*)		
3	frmBeitrag (Tabelle: *tbl_Beitrag*)	**3.1**	frmBeitragStaffel (Tabelle: *tbl_Beitrag_Staffel*)
		3.2	FrmAbteilung (Tabelle: *tbl_Abteilung*)
		3.3	FrmBootArt (Tabelle: *tbl_BootArt*)
1	Listenfeld Formularauswahl (Abfrage: *qry_Forms*)		

Makros in Formulareigenschaften		
1	Beim Laden	Navigation.*FormStart*
2	Nach der Datensatzaktion	Backup.*Preferences*
3	Beim Laden	Design.*PreferencesHide*
3.1	Beim Laden	Design.*PreferencesHide*
3.2	Beim Laden	Design.*PreferencesHide*
3.3	Beim Laden	Design.*PreferencesHide*
Makros in Feldeigenschaften		
1	Listenfeld Formularauswahl (Status geändert)	Navigation.*Navigation*
2	Maskiertes Feld IBAN (Text modifiziert)	Eingabekontrolle.*IBAN_Length_Change*
2	Maskiertes Feld IBAN (Bei Fokusverlust)	Eingabekontrolle.*IBAN_Test*

Formularfilter	
2	`001` `( "tbl_Einstellungen"."ID" = TRUE )`

Dieses Formular ist vom Screenshot her zu breit, so dass es in 2 Abschnitten unterteilt untereinander dargestellt wurde.

Hier geht es um Standardeinstellungen, die die Datenbank auch so gestalten können, das bestimmte Module ausgeblendet werden und nicht weiter zur Verfügung stehen. Die Einstellungen (siehe oberer Screenshot) werden in einer einzeiligen Tabelle gespeichert.

Daneben ist eine Eingabemöglichkeit für die Beitragsstruktur. Es gibt hier die Möglichkeit, nicht nur Standardbeiträge in eine Tabelle ein zu geben, sondern auch eine Staffelung von Beiträgen in einer separaten Tabelle ab zu speichern.

frm_Mitglied

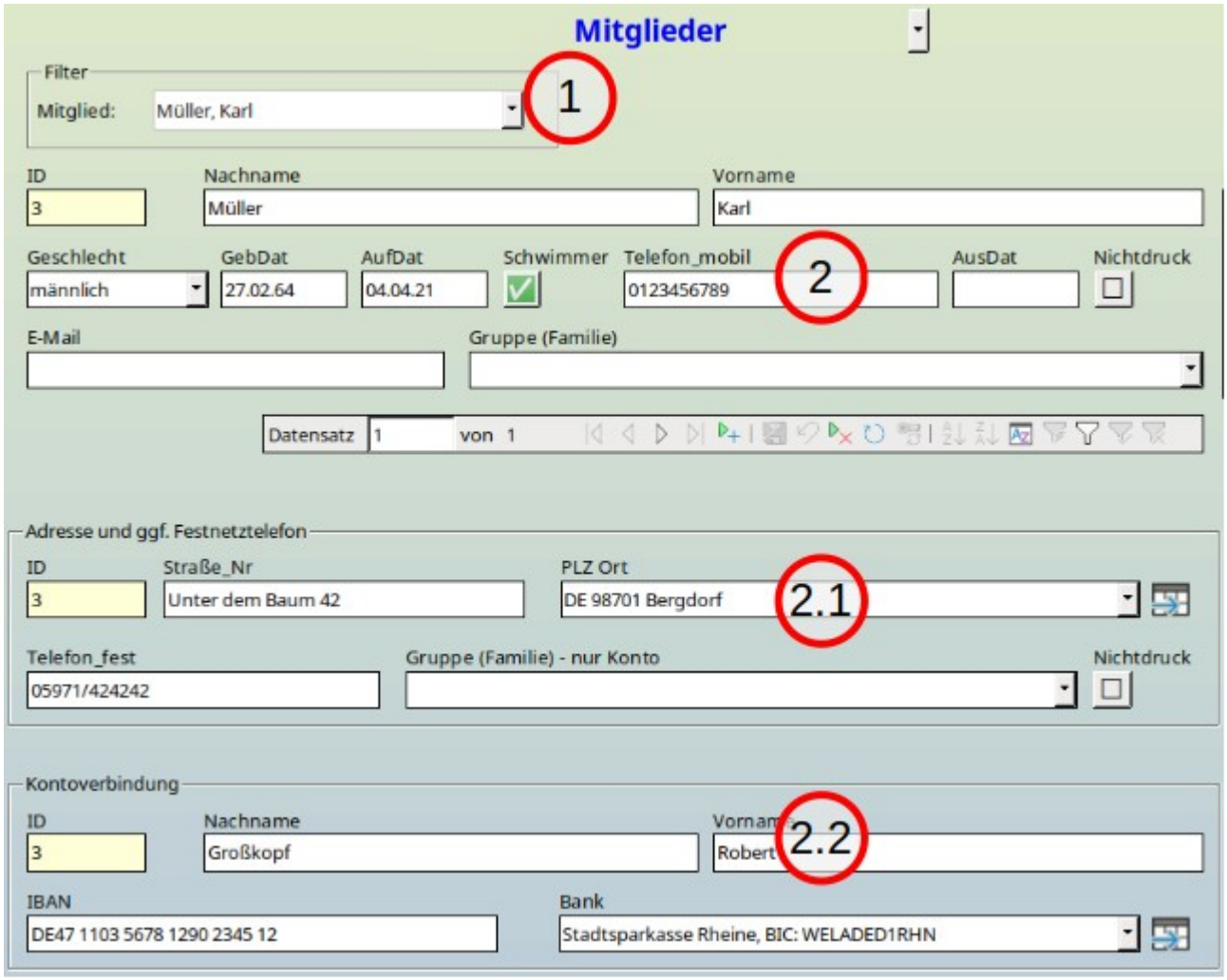


Abbildung 6: Dieses Formular ist von der Breite her im Screenshot nicht komplett darstellbar. Hier die Standardansicht für eine Person, die sowohl eine eigene Adresse als auch ein Konto angegeben hat.

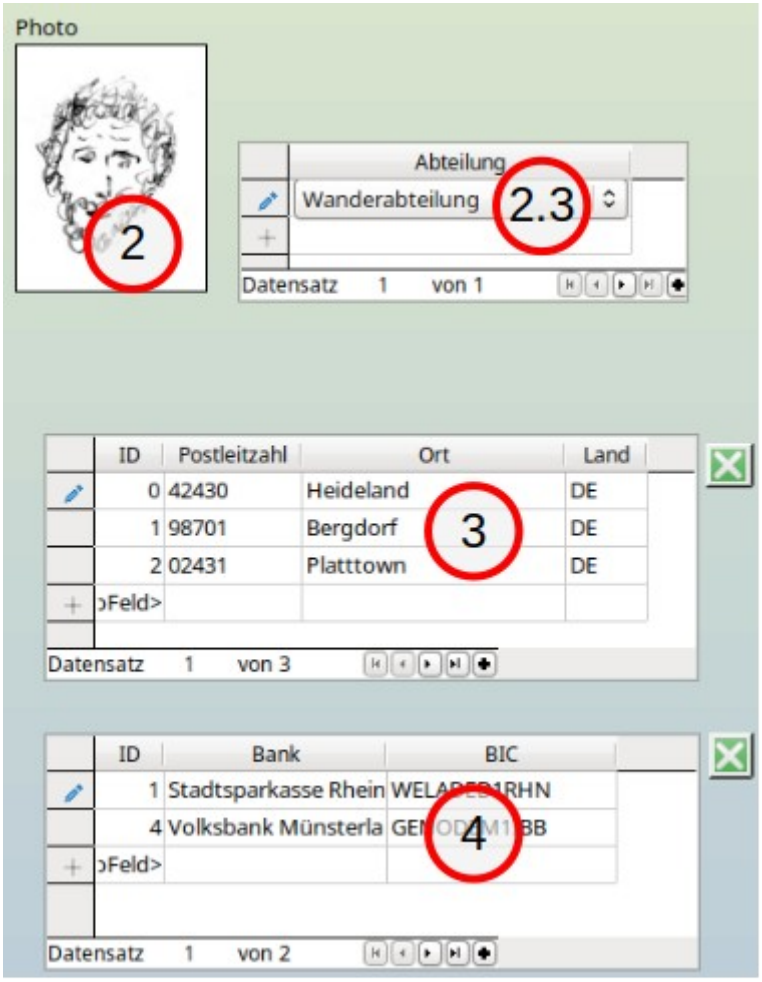


Abbildung 7: Rechts von dem vorhergehenden Screenshot ist noch das Bild des Mitglieds mit eingeblendet. Daneben als Unterformular die Übersicht über die Abteilungen. Die Formulare für die Orte und die Banken werden über Buttons im Formular separat angezeigt und wieder ausgeblendet.

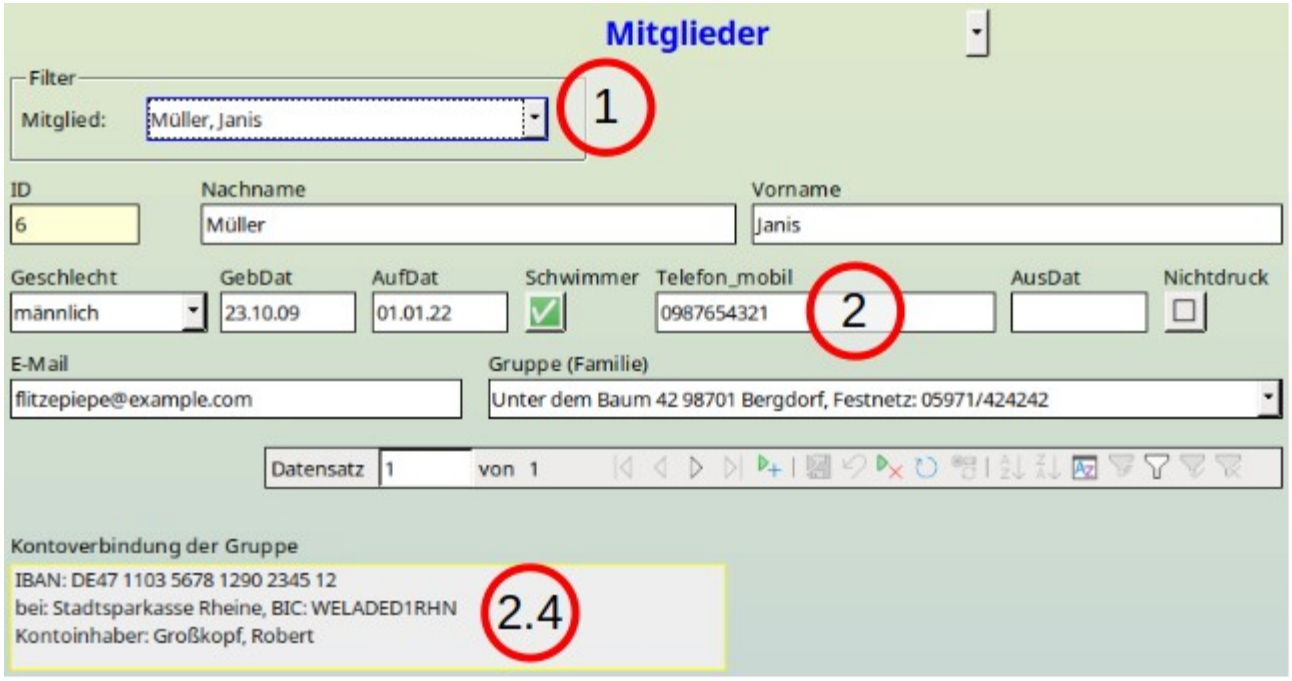


Abbildung 8: Ist unter Gruppe (Familie) eine Adresse einer anderen Person ausgewählt worden, so erscheint statt der Adresseingabe und der Kontoeingabe lediglich ein Verweis, welches Konto dieser Person über die Gruppe zugeordnet wurde.

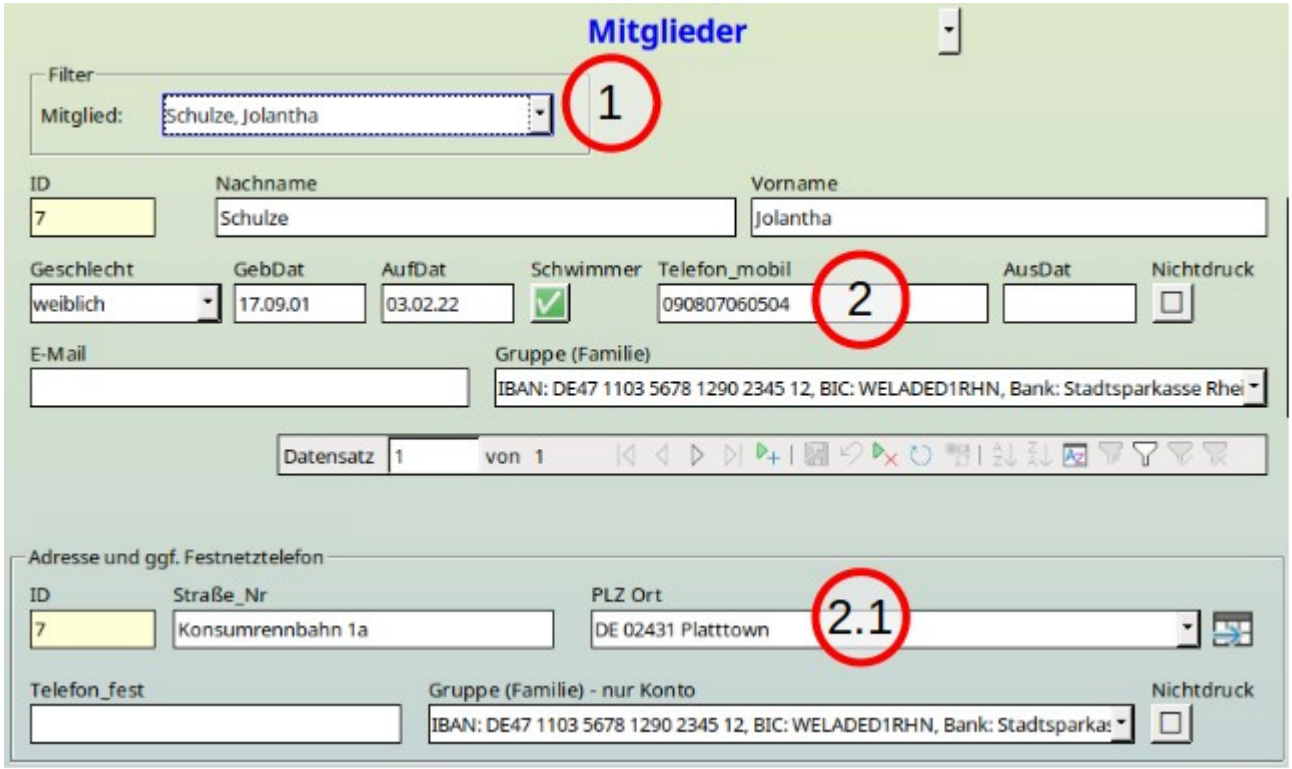


Abbildung 9: Hat die Person eine eigene Adresse, wird aber über das Konto einer Gruppe zugeordnet, so wird zuerst die Adresse angegeben und dort dann die Zuordnung zum Konto und damit zur Gruppe vorgenommen.

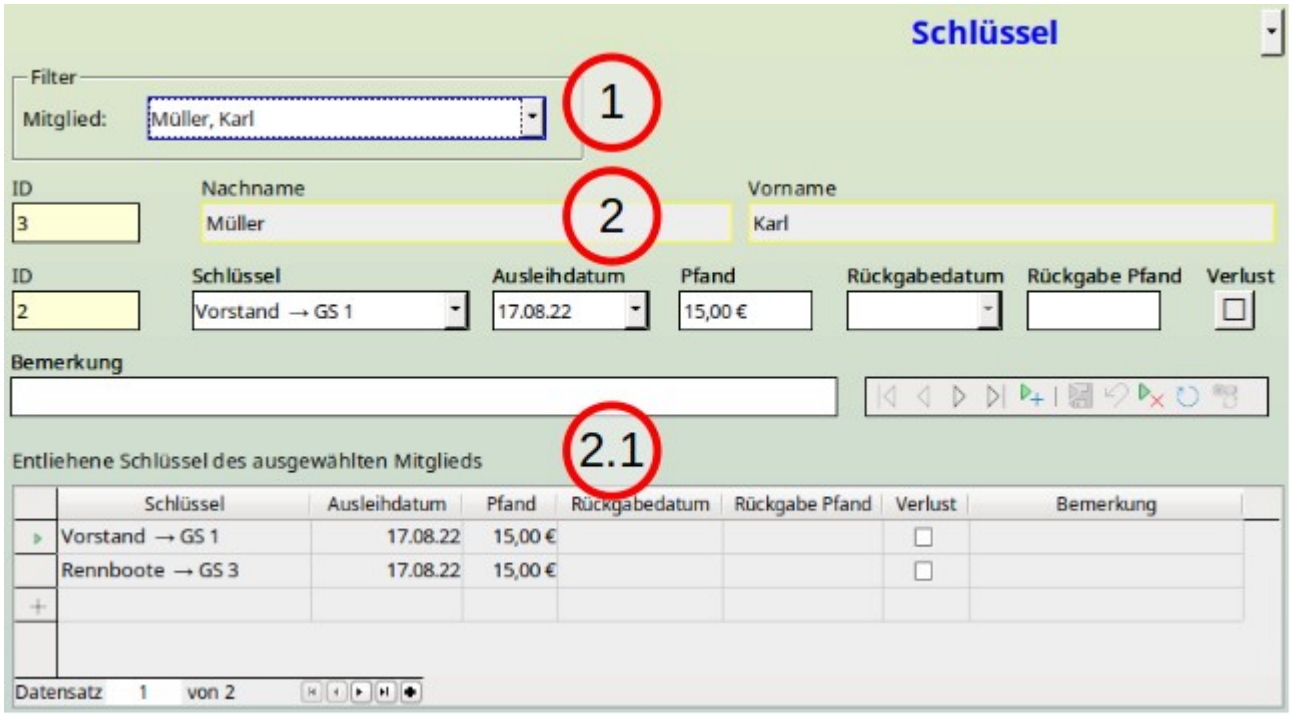
1	frmFilter (Abfrage: *qry_Filter*)		
2	frmMitglied (Abfrage: *qry_Filter_Mitglied*)	**2.1**	frmAdresse (Abfrage: *Abfrage_Start*)
		2.2	frmKonto (Tabelle: *tbl_Konto*)
		2.3	frmAbteilung (Tabelle: *tbl_Mitglied_Abteilung*)
		2.4	frmGruppe (Abfrage: *qry_Mitglied_Konto*)
3	frmOrt (Tabelle: *tbl_Ort*)		
4	frmBank (Tabelle: *tbl_Bank*)		
1	Listenfeld Formularauswahl (Abfrage: *qry_Forms*)		

Makros in Formulareigenschaften		
1	Beim Laden	Navigation.*FormStart*
2	Nach dem Datensatzwechsel	Design.*AfterRecordChange*
2.1	Nach dem Datensatzwechsel	Design.*BoolStart*
3	Nach der Datensatzaktion	Speichern.*ListboxContentChange*
4	Nach der Datensatzaktion	Speichern.*ListboxContentChange*
Makros in Feldeigenschaften		
1	Listenfeld Formularauswahl (Status geändert)	Navigation.*Navigation*
1	Listenfeld Mitglied (Status geändert)	Filter.*Filtern*
2	Schaltfläche Schwimmer (Taste gedrückt)	Design.*BoolChangeKey*
2	Schaltfläche Schwimmer (Maustaste gedrückt)	Design.*BoolChange*
2	Schaltfläche Nichtdruck (Taste gedrückt)	Design.*BoolChangeKey*
2	Schaltfläche Nichtdruck (Maustaste gedrückt)	Design.*BoolChange*
2	Listenfeld Gruppe (Status geändert)	Speichern.*NewGroup*
2.1	Schaltfläche Nichtdruck (Taste gedrückt)	Design.*BoolChangeKey*
2.1	Schaltfläche Nichtdruck (Maustaste gedrückt)	Design.*BoolChange*
2.1	Grafische Schaltfläche (Maustaste gedrückt	Speichern.*ShowTableControl*
2.1	Listenfeld Gruppe (Status geändert)	Speichern.*NewGroup*
2.2	Maskiertes Feld IBAN (Text modifiziert)	Eingabekontrolle.*IBAN_Length_Change*
2.2	Maskiertes Feld IBAN (Bei Fokusverlust)	Eingabekontrolle.*IBAN_Test*
2.2	Grafische Schaltfläche (Maustaste gedrückt	Speichern.*ShowTableControl*
3	Schaltfläche (Aktion ausführen)	Speichern.*ShowTableControl*
4	Schaltfläche (Aktion ausführen)	Speichern.*ShowTableControl*

SQL-Code in Listenfeldern		
1	Mitglied *tbl_Mitglied*	001 SELECT "Nachname"||COALESCE(', '||"Vorname",'') AS "Name", "ID" 002 FROM "tbl_Mitglied" 003 ORDER BY "Name"

2	Gruppe (Familie) *tbl_Einstellungen* *tbl_Adresse* *tbl_Ort* *tbl_Mitglied*	001 SELECT DISTINCT '' AS "Gruppe", NULL AS "ID", -1 AS "Sort" FROM "tbl_Einstellungen" 002 UNION 003 SELECT DISTINCT 'Neu: Personengruppe existiert noch nicht.' AS "Gruppe", NULL AS "ID", 0 AS "Sort" FROM "tbl_Einstellungen" 004 UNION 005 SELECT COALESCE ("tbl_Adresse"."Straße_Nr" || ' ' || "tbl_Ort"."Postleitzahl" || ' ' || "tbl_Ort"."Ort", 'keine Adresse') || COALESCE (', Festnetz: ' || "tbl_Adresse"."Telefon_fest", ', kein Festnetztelefon') AS "Gruppe", "tbl_Mitglied"."ID", 1 AS SORT FROM "tbl_Mitglied", "tbl_Adresse", "tbl_Ort" WHERE "tbl_Mitglied"."ID" = "tbl_Adresse"."mitID" AND "tbl_Adresse"."ortID" = "tbl_Ort"."ID" AND ("tbl_Mitglied"."Gruppe_mitID" IS NULL OR "tbl_Mitglied"."ID" = "tbl_Mitglied"."Gruppe_mitID") 006 ORDER BY "Sort", "Gruppe"
2.1	PLZ Ort *tbl_Ort*	001 SELECT COALESCE("Land"||' ','')||"Postleitzahl" || ' ' || "Ort" AS "Ort", "ID" 002 FROM "tbl_Ort" 003 ORDER BY "Ort" ASC
2.1	Gruppe (Familie) – nur Konto *tbl_Einstellungen* *tbl_Konto* *tbl_Bank* *tbl_Mitglied*	001 SELECT DISTINCT '' AS "Gruppe", NULL AS "ID", -1 AS "Sort" FROM "tbl_Einstellungen" 002 UNION 003 SELECT DISTINCT 'Neu: Konto existiert noch nicht.' AS "Gruppe", NULL AS "ID", 0 AS "Sort" FROM "tbl_Einstellungen" 004 UNION 005 SELECT 'IBAN: '||"tbl_Konto"."IBAN"||', BIC: '||"tbl_Bank"."BIC"||', Bank: '||"tbl_Bank"."Bank" AS "Gruppe", "tbl_Mitglied"."ID", 1 AS SORT FROM "tbl_Mitglied", "tbl_Konto", "tbl_Bank" WHERE "tbl_Mitglied"."ID" = "tbl_Konto"."mitID" AND "tbl_Konto"."banID" = "tbl_Bank"."ID" AND ("tbl_Mitglied"."Gruppe_mitID" IS NULL OR "tbl_Mitglied"."ID" = "tbl_Mitglied"."Gruppe_mitID") 006 ORDER BY "Sort", "Gruppe"
2.2	Bank *tbl_Bank*	001 SELECT "Bank"||', BIC: '||"BIC" AS "Bankverbindung", "ID" 002 FROM "tbl_Bank" 003 ORDER BY "Bankverbindung"
2.3	Abteilung *tbl_Abteilung*	001 SELECT "Abteilung", "ID" 002 FROM "tbl_Abteilung" 003 ORDER BY "Abteilung" ASC

frm_Schluessel



1	frmFilter (Abfrage: *qry_Filter*)		
2	frmMitglied (Abfrage: *qry_Filter_Mitglied*)	**2.1**	frmSchluessel (Tabelle: *tbl_MitgliedSchluessel*)
1	Listenfeld Formularauswahl (Abfrage: *qry_Forms*)		

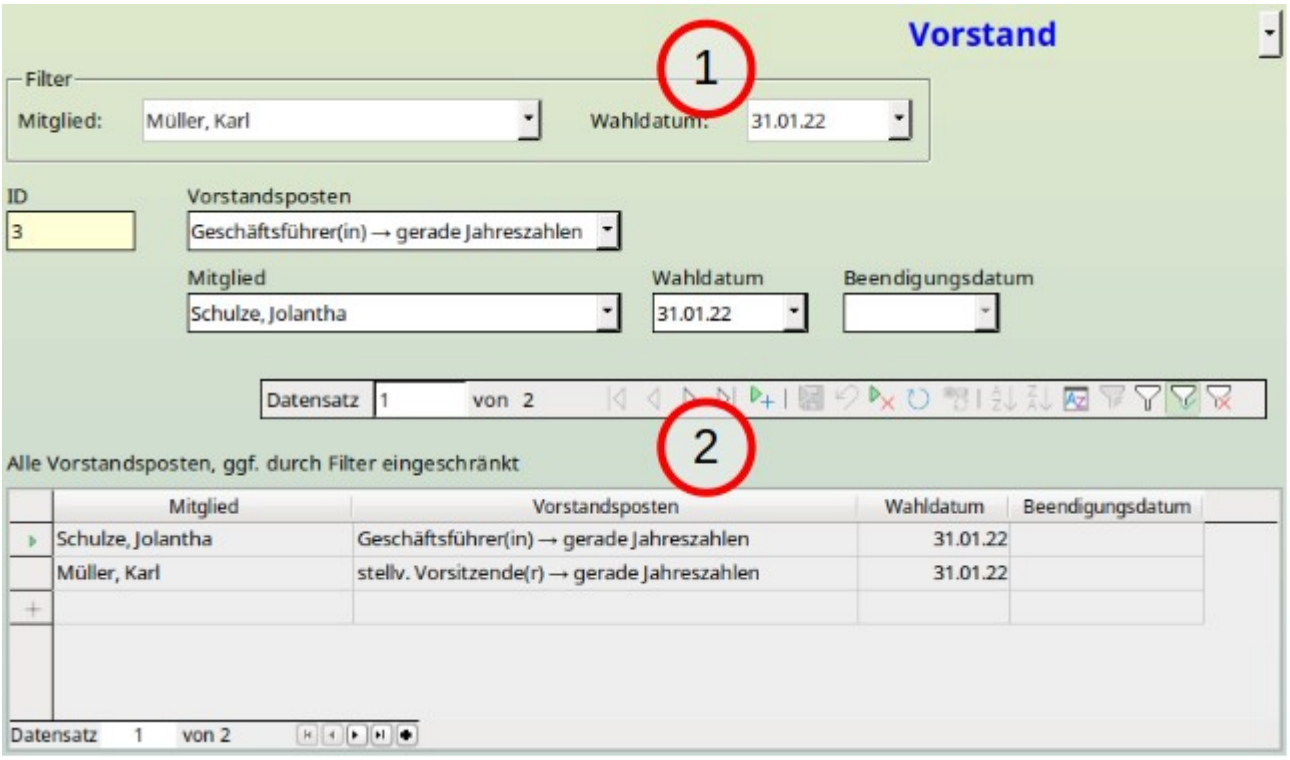
Makros in Formulareigenschaften		
1	Beim Laden	Navigation.*FormStart*
2.1	Nach dem Datensatzwechsel	Eingabekontrolle.*SelectKey*
Makros in Feldeigenschaften		
1	Listenfeld Formularauswahl (Status geändert)	Navigation.*Navigation*
1	Listenfeld Mitglied (Status geändert)	Filter.*Filtern*
2.1	Schaltfläche Verlust (Taste gedrück)	Design.*BoolChangeKey*
2.1	Schaltfläche Verlust (Maustaste gedrück)	Design.*BoolChange*

SQL-Code in Listenfeldern		

1	Mitglied *tbl_Mitglied*	`001 SELECT "Nachname"||COALESCE(', '||"Vorname",'') AS "Name", "ID"`
		`002 FROM "tbl_Mitglied"`
		`003 ORDER BY "Name"`

2.1	Schlüssel (2*) *tbl_Schluessel*	`001 SELECT "Bereich"||'  → '||"SNr" AS "Schluessel", "SNr"`
		`002 FROM "tbl_Schluessel"`
		`003 ORDER BY "Schluessel" ASC`

frm_Vorstand



1	frmFilter (Abfrage: *qry_Filter*)
2	frmVorstand (Abfrage: *qry_Filter_Vorstand*)
1	Listenfeld Formularauswahl (Abfrage: *qry_Forms*)

Makros in Formulareigenschaften		
1	Beim Laden	Navigation.*FormStart*
2	Nach dem Datensatzwechsel	Eingabekontrolle.*SelectBoard*
Makros in Feldeigenschaften		
1	Listenfeld Formularauswahl (Status geändert)	Navigation.*Navigation*
1	Listenfeld Mitglied (Status geändert)	Filter.*Filtern*
2	Listenfeld Vorstandsposten (Status geändert)	Eingabekontrolle.*BoardFilterSelect*

SQL-Code in Listenfeldern		
1, 2	Mitglied (3*) *tbl_Mitglied*	001 SELECT "Nachname"||COALESCE(', '||"Vorname",'') AS "Name", "ID" 002 FROM "tbl_Mitglied" 003 ORDER BY "Name"
2	Vorstands-posten (2*) *tbl_Vor-standPos-ten*	001 SELECT "Posten" || ' → ' || CASE WHEN "Zyklus" = 'g' THEN 'gerade Jahreszahlen' WHEN "Zyklus" = 'u' THEN 'ungerade Jahreszahlen' ELSE 'kein Zyklus' END AS "VPosten", "ID", CASE WHEN "Zyklus" = '-' THEN 'h' ELSE "Zyklus" END AS "Zyklus" 002 FROM "tbl_VorstandPosten" 003 ORDER BY "Zyklus" ASC, "ID" ASC

Formularfilter	
2	001 ("tbl_Mitglied_VorstandPosten"."EndDat" IS NULL)

Berichte

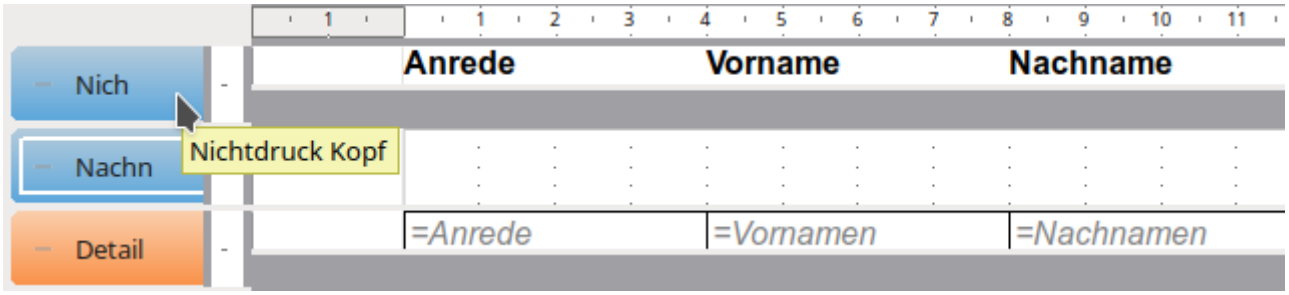
rpt_Anschriften_Tabelle

	A	B	C	D	E	F
1	Anrede	Vorname	Nachnam	Straße un	PLZ	Ort
2		Karl, Maria	Müller und	Unter dem	98701	Bergdorf
3	Frau	Yvonne-Ch	Power	Industriege	98701	Bergdorf
4	Herrn	Gisbert	Querulant	An dem Ba	2431	Platttown
5	Herrn	Heinz-Her	Schöngeis	Kirchweg 1	42430	Heideland
6	Frau	Jolantha	Schulze	Konsumrer	2431	Platttown

	A	B	C	D	E	F
1	Anrede	Vorname	Nachname	Straße und Nr.	PLZ	Ort
2		Karl, Maria, Isabel und Janis	Müller und Müller-Mehl	Unter dem Baum 42	98701	Bergdorf
3	Frau	Yvonne-Chantal	Power	Industriegebiet Nord 4711	98701	Bergdorf
4	Herrn	Gisbert	Querulant	An dem Bach 162c	2431	Platttown
5	Herrn	Heinz-Herbert	Schöngeist	Kirchweg 13	42430	Heideland
6	Frau	Jolantha	Schulze	Konsumrennbahn 1a	2431	Platttown

Datenquelle
Ansicht: *viw_Anschrift*

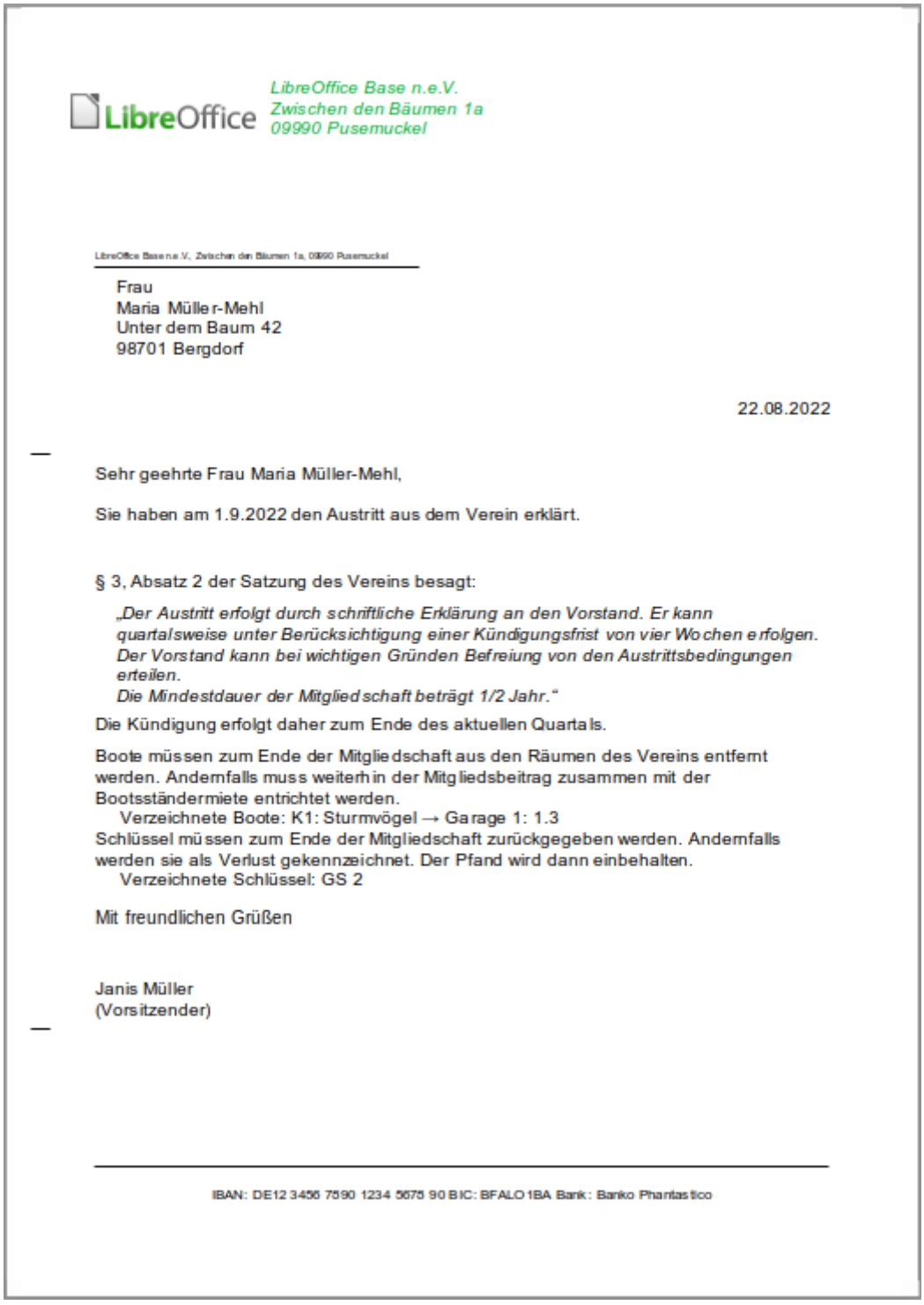
Damit eine derartige Übersicht ohne Probleme in eine Tabelle übertragen werden kann, müssen die Felder für die Titelleiste und für den Inhalt lediglich die gleiche Breite haben. Eine voreingestellte Breite wird nach Calc hin leider nicht übertragen, so dass die Breite der einzelnen Spalten nach der Erstellung des Berichts gegebenenfalls über eine Spaltenmarkierung und die Wahl der optimalen Breite eingestellt werden muss.



Damit die Titelleiste fest über dem gesamten Inhalt erscheint ist über **Sortierung und Gruppierung** die Gruppe «Nichtdruck» erstellt worden. Alle Anschriften dieses Berichtes sind ja in diesem Wert gleich. Deshalb ändert sich die Gruppe nicht und der Inhalt der Gruppe wird entsprechend nicht wiederholt.

Die Sortierung wird nach der Gruppe «Nachname» vorgenommen. Diese Gruppe ist hier für die bessere Übersicht im Entwurf angezeigt, aber über **Eigenschaften Gruppenkopf → Allgemein → Sichtbar: Nein** von der Anzeige im ausgeführten Bericht ausgeschlossen.

rpt_Austritt



LibreOffice Base n.e.V.
Zwischen den Bäumen 1a
09990 Pusemuckel

LibreOffice Base n.e.V., Zwischen den Bäumen 1a, 09990 Pusemuckel

Frau
Maria Müller-Mehl
Unter dem Baum 42
98701 Bergdorf

22.08.2022

Sehr geehrte Frau Maria Müller-Mehl,

Sie haben am 1.9.2022 den Austritt aus dem Verein erklärt.

§ 3, Absatz 2 der Satzung des Vereins besagt:

„Der Austritt erfolgt durch schriftliche Erklärung an den Vorstand. Er kann quartalsweise unter Berücksichtigung einer Kündigungsfrist von vier Wochen erfolgen. Der Vorstand kann bei wichtigen Gründen Befreiung von den Austrittsbedingungen erteilen.
Die Mindestdauer der Mitgliedschaft beträgt 1/2 Jahr."

Die Kündigung erfolgt daher zum Ende des aktuellen Quartals.

Boote müssen zum Ende der Mitgliedschaft aus den Räumen des Vereins entfernt werden. Andernfalls muss weiterhin der Mitgliedsbeitrag zusammen mit der Bootsständermiete entrichtet werden.
Verzeichnete Boote: K1: Sturmvögel → Garage 1: 1.3
Schlüssel müssen zum Ende der Mitgliedschaft zurückgegeben werden. Andernfalls werden sie als Verlust gekennzeichnet. Der Pfand wird dann einbehalten.
Verzeichnete Schlüssel: GS 2

Mit freundlichen Grüßen

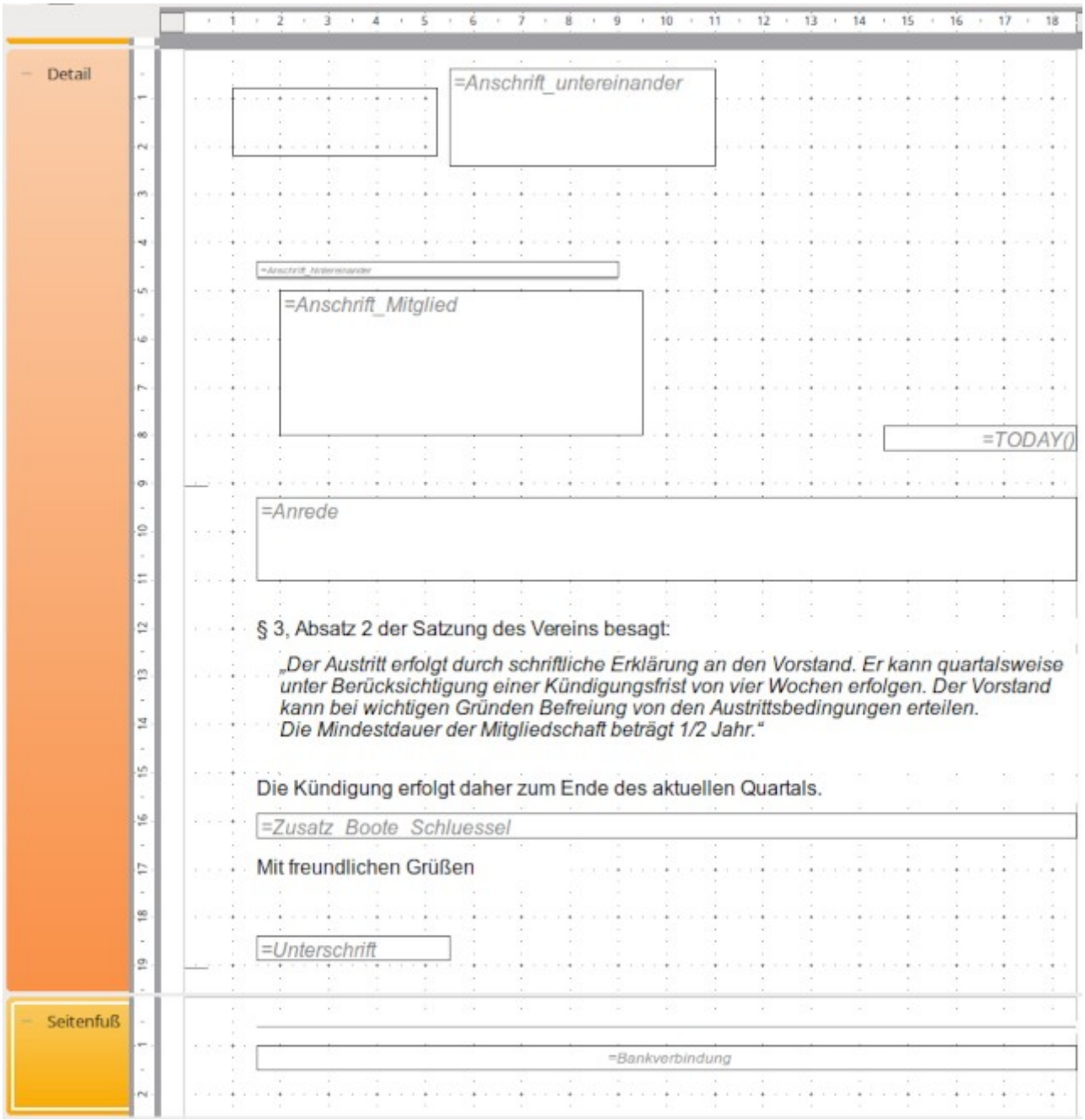
Janis Müller
(Vorsitzender)

IBAN: DE12 3456 7590 1234 5675 90 BIC: BFALO1BA Bank: Banko Phantastico

Datenquelle
Abfrage: *qry_Filter_Austritt*

Datenziel
Makro: *ReportStart*

Hier wird ein Schreiben zum Austritts eines ausgewählten Mitglieds mit Hilfe des Berichtsmoduls erstellt.

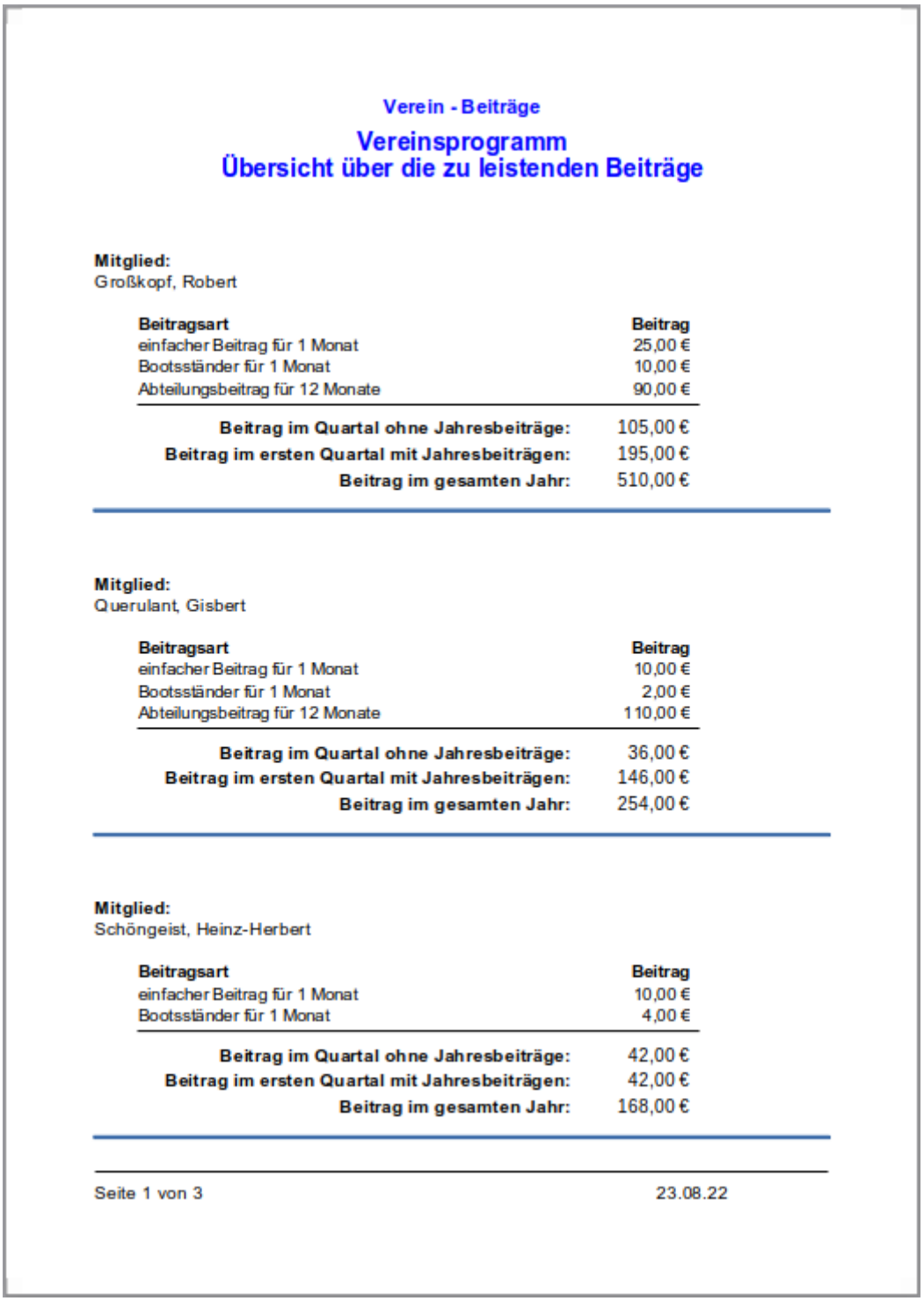


Der Bericht besteht aus einem großen Bereich «Detail». Eine Gruppierung wird nicht benötigt, da nur genau ein Datensatz über den Bericht ausgedruckt wird. Das Feld «Zusatz_Boote_Schluessel» ist als automatisch anwachsendes Feld ausgelegt. Ist kein Inhalt vorgegeben, so belegt es lediglich 0,5 cm des Briefes und fällt nicht weiter auf. Kommt aber, wie in dem angezeigten Beispiel ein entsprechender Text hinzu, so nimmt das Feld den dafür vorgesehenen Platz ein.

Mehrere Felder werden mit komplett zusammengestelltem Text aus der Abfrage beschickt, so dass z. B. «Anschrift_Mitglied» nicht die ganzen einzelnen Felder wie Vorname, Nachname usw. nacheinander anzeigt sondern komplett als Block mit Zeilenumbrüchen in der Abfrage enthalten ist. Das erleichtert die Formatierung besonders bei Text, der aus verschiedenen Feldern innerhalb einer Zeile stehen muss. Würde «Vorname» und «Nachname» in einzelnen Feldern erscheinen, so hätten die Inhalte im Bericht einen fest vorgegebenen Abstand, der nichts mit

einer einfachen Leertaste zu tun hat. Reicht der Platz nicht, so muss später der Bericht noch angepasst werden.

rpt_Beitraege



Verein - Beiträge

Vereinsprogramm
Übersicht über die zu leistenden Beiträge

Mitglied:
Großkopf, Robert

Beitragsart	Beitrag
einfacher Beitrag für 1 Monat	25,00 €
Bootsständer für 1 Monat	10,00 €
Abteilungsbeitrag für 12 Monate	90,00 €
Beitrag im Quartal ohne Jahresbeiträge:	105,00 €
Beitrag im ersten Quartal mit Jahresbeiträgen:	195,00 €
Beitrag im gesamten Jahr:	510,00 €

Mitglied:
Querulant, Gisbert

Beitragsart	Beitrag
einfacher Beitrag für 1 Monat	10,00 €
Bootsständer für 1 Monat	2,00 €
Abteilungsbeitrag für 12 Monate	110,00 €
Beitrag im Quartal ohne Jahresbeiträge:	36,00 €
Beitrag im ersten Quartal mit Jahresbeiträgen:	146,00 €
Beitrag im gesamten Jahr:	254,00 €

Mitglied:
Schöngeist, Heinz-Herbert

Beitragsart	Beitrag
einfacher Beitrag für 1 Monat	10,00 €
Bootsständer für 1 Monat	4,00 €
Beitrag im Quartal ohne Jahresbeiträge:	42,00 €
Beitrag im ersten Quartal mit Jahresbeiträgen:	42,00 €
Beitrag im gesamten Jahr:	168,00 €

Seite 1 von 3 23.08.22

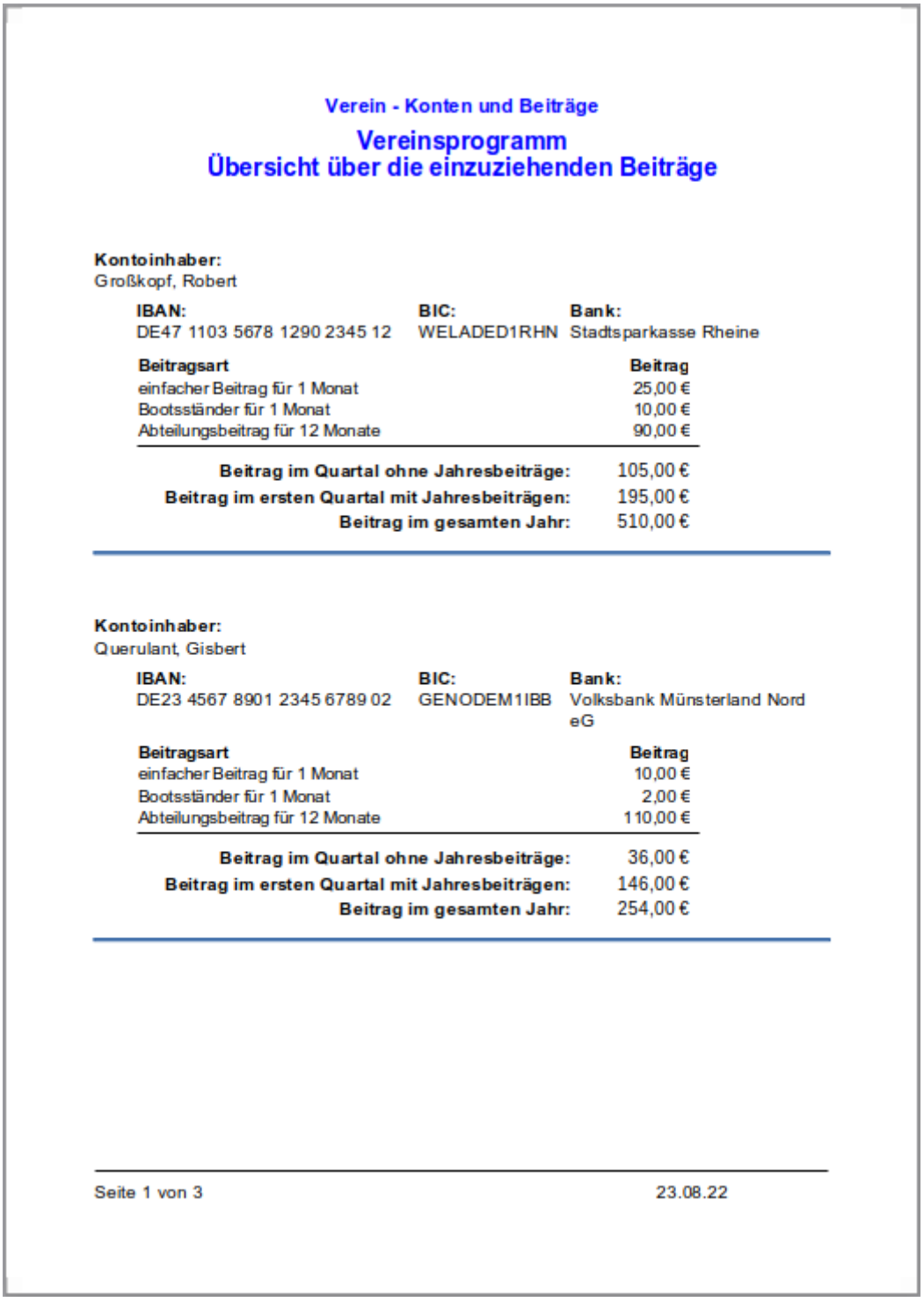
Datenquelle
Abfrage: *qry_Beitraege*

Mit diesem Bericht werden die Beiträge angegeben, die für das entsprechende Mitglied fällig sind. Dabei richtet sich die Beitragshöhe nach der entsprechenden Gruppenzuweisung.

Seitenkopf und Seitenfuß erscheinen auf jeder Seite des Berichtes. Es gibt keine Möglichkeit verschiedene Seitenvorlagen zu erstellen, so dass der Bericht z. B. mit dem Inhalt des Berichts-kopfes statt dem des Seitenkopfes auf der ersten Seite beginnen würde. Der Berichtskopf erscheint in einem Bericht nur einmal zum Beginn. Der Berichtsfuß würde nur einmal zum Schluss erscheinen – vor dem Seitenfuß der letzten Seite. In diesem Falle ist der Berichtsfuß allerdings leer und in den Eigenschaften auf **Sichtbar: → Nein** eingestellt.

Damit die Beitragsarten immer in der gleichen Reihenfolge erscheinen ist eine Gruppe «Sort» erstellt worden. Das Feld "Sort" ist in der Abfrage für den Bericht als Sortierungsfeld enthalten. Der Kopf-Bereich für «Sort» wird nicht angezeigt.

rpt_Beitraege_Konto



Verein - Konten und Beiträge

Vereinsprogramm
Übersicht über die einzuziehenden Beiträge

Kontoinhaber:
Großkopf, Robert

IBAN:	BIC:	Bank:
DE47 1103 5678 1290 2345 12	WELADED1RHN	Stadtsparkasse Rheine

Beitragsart	Beitrag
einfacher Beitrag für 1 Monat	25,00 €
Bootsständer für 1 Monat	10,00 €
Abteilungsbeitrag für 12 Monate	90,00 €

Beitrag im Quartal ohne Jahresbeiträge:	105,00 €
Beitrag im ersten Quartal mit Jahresbeiträgen:	195,00 €
Beitrag im gesamten Jahr:	510,00 €

Kontoinhaber:
Querulant, Gisbert

IBAN:	BIC:	Bank:
DE23 4567 8901 2345 6789 02	GENODEM1IBB	Volksbank Münsterland Nord eG

Beitragsart	Beitrag
einfacher Beitrag für 1 Monat	10,00 €
Bootsständer für 1 Monat	2,00 €
Abteilungsbeitrag für 12 Monate	110,00 €

Beitrag im Quartal ohne Jahresbeiträge:	36,00 €
Beitrag im ersten Quartal mit Jahresbeiträgen:	146,00 €
Beitrag im gesamten Jahr:	254,00 €

Seite 1 von 3 23.08.22

Datenquelle
Abfrage: *qry_Beitraege*

Dieser Bericht entspricht dem vorhergehenden Bericht. Hier wurde lediglich in dem Bereich «Kontoinhaber Kopf» neben dem Kontoinhaber auch das Konto mit aufgenommen. Dieser Bericht ist für die Nutzung der Datenbank mit Kontoverwaltung gedacht.

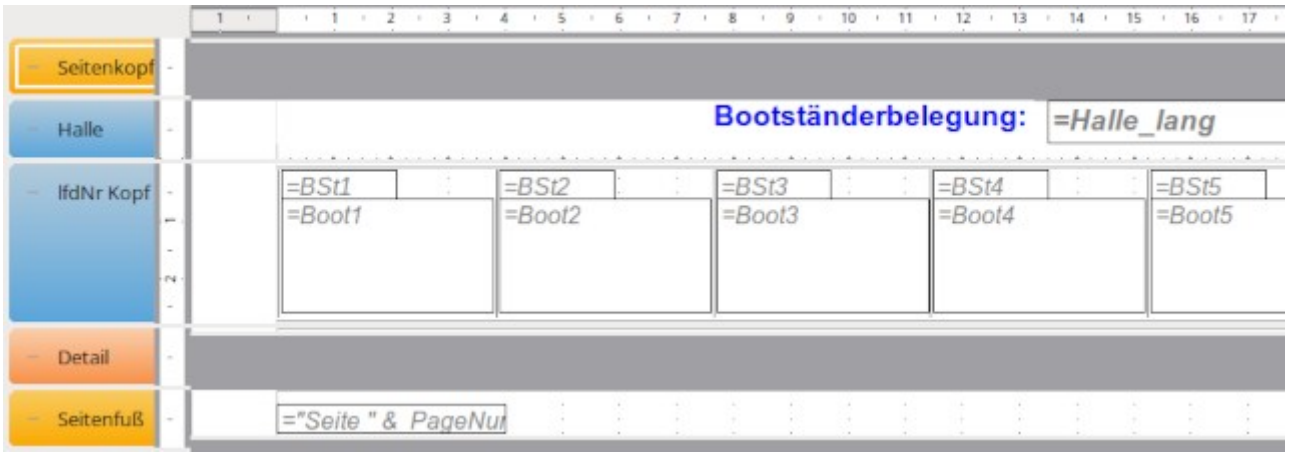
rpt_Bootstaender_Belegung

Bootständerbelegung: Garage 1

4.1	4.2	4.3	4.4			
3.1	3.2	3.3	3.4			
2.1	2.2 K2: Seeblick gehört: Yvonne-Chantal Power	2.3 K2: Autonom gehört: Gisbert Querulant	2.4 C4: Canadier für alle gehört: Heinz-Herbert Schöngeist			
1.1	1.2 K1: Ems gehört: Karl Müller	1.3 K1: Sturmvögel gehört: Maria Müller-Mehl	1.4			
0.1	0.2	0.3 K2: Moin gehört: Karl Müller				

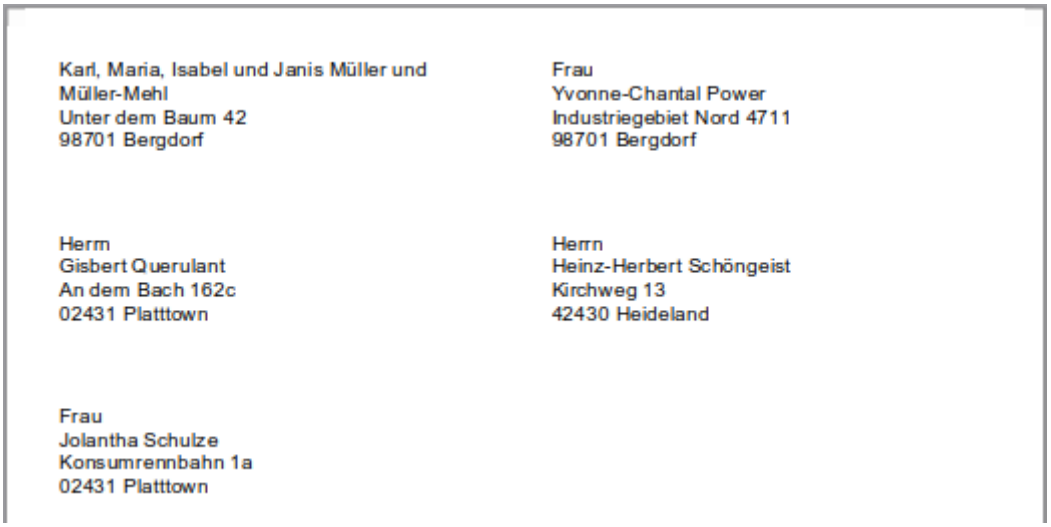
Datenquelle
Abfrage: *qry_Bootsstaender*

Um alle Spalten dieser Abfrage aufnehmen zu können ist dieser Bericht im Querformat gehalten.



Die Bereiche «Seitenkopf» und «Detail» sind unsichtbar geschaltet. Die Gruppierung «lfdNr» ist notwendig, damit die Reihenfolge innerhalb der jeweiligen Hallen gewahrt wird. Der gesamte Inhalt in «lfdNr Kopf» hätte auch in «Detail» stehen können. Leider hat aber der Bereich «Detail» den Fehler, dass dort auch bei **Zusammenhalten → Ja** der Bereich beim Seitenumbruch manchmal in zwei Teile zerfällt. Ein Gruppenkopf oder Gruppenfuß lässt unter den Umständen einen Umbruch mitten im Bereich nicht zu.

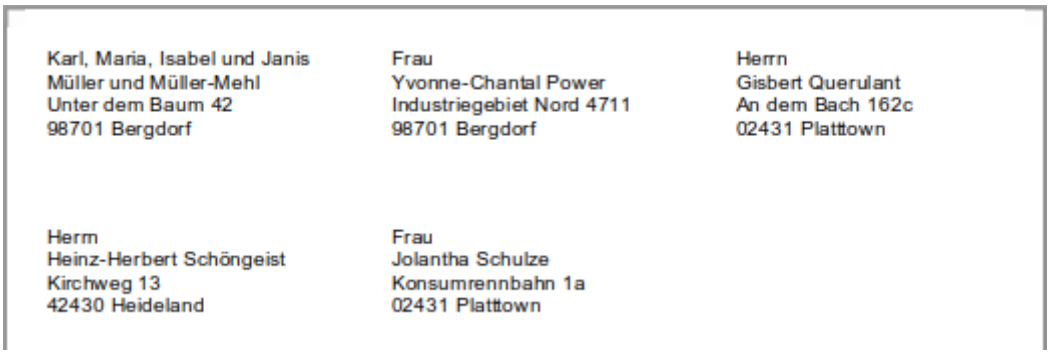
rpt_Etiketten_2_Spalten



Datenquelle
Abfrage: *qry_Anschriften_2spaltig*

Der gesamte Inhalt für den Druck auf ein Etikett ist durch die Abfrage in einem Feld zusammengefasst. Deshalb reicht für den Bericht auch der Bereich «Detail» aus. Die Positionierung der Etiketten ist durch die Seitenrandeinstellungen (**Format → Seite**) und die Einstellungen für die aufgezogenen Rahmen beeinflussbar.

rpt_Etiketten_3_Spalten



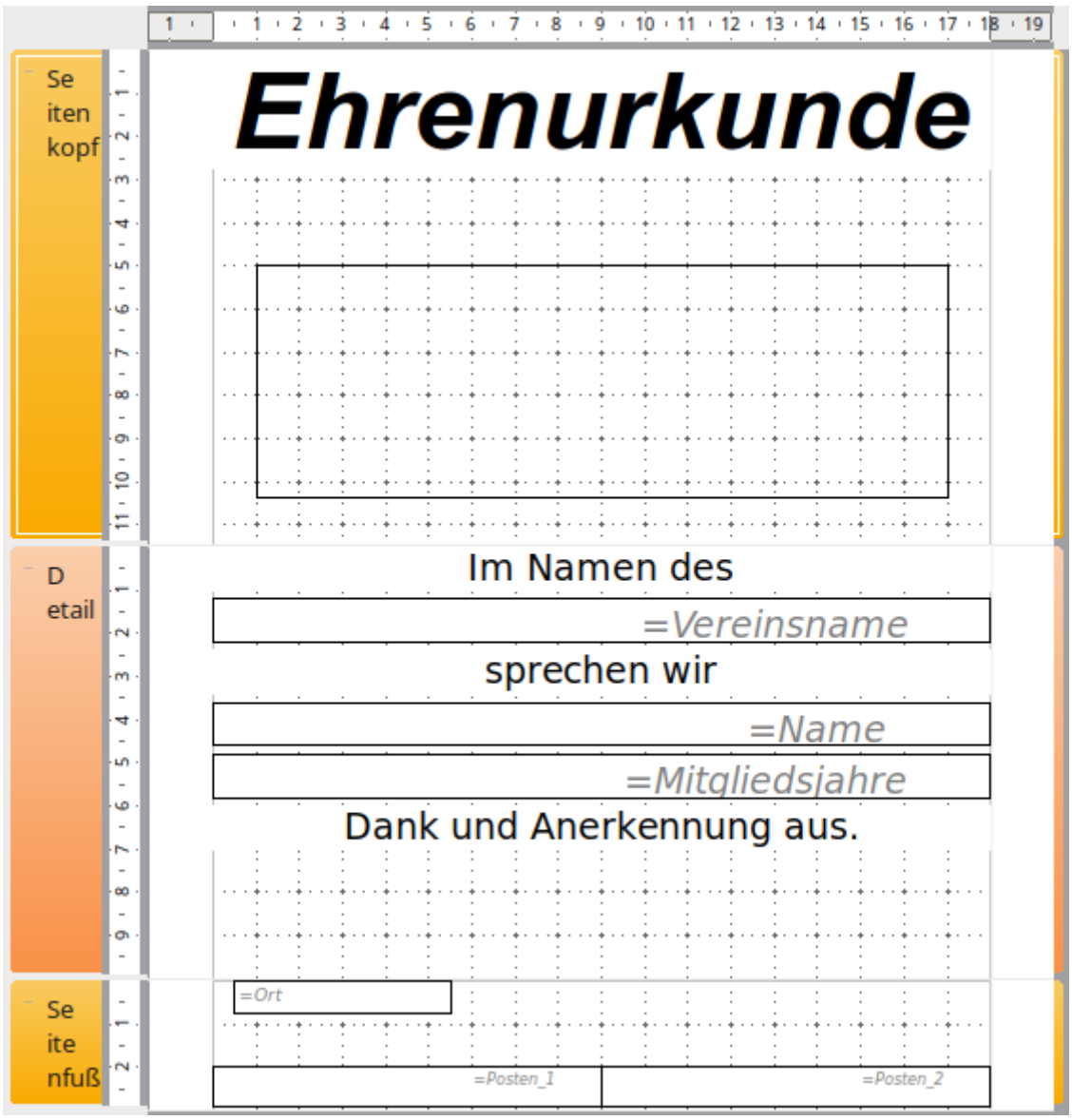
Datenquelle
Abfrage: *qry_Anschriften_3spaltig*

Hier werden lediglich drei statt zwei Etiketten nebeneinander abgebildet. Auch hier muss natürlich die Anordnung abhängig von dem benutzen Etikettenmaterial angeglichen werden.

rpt_Jubilaeum_Ehrenurkunde

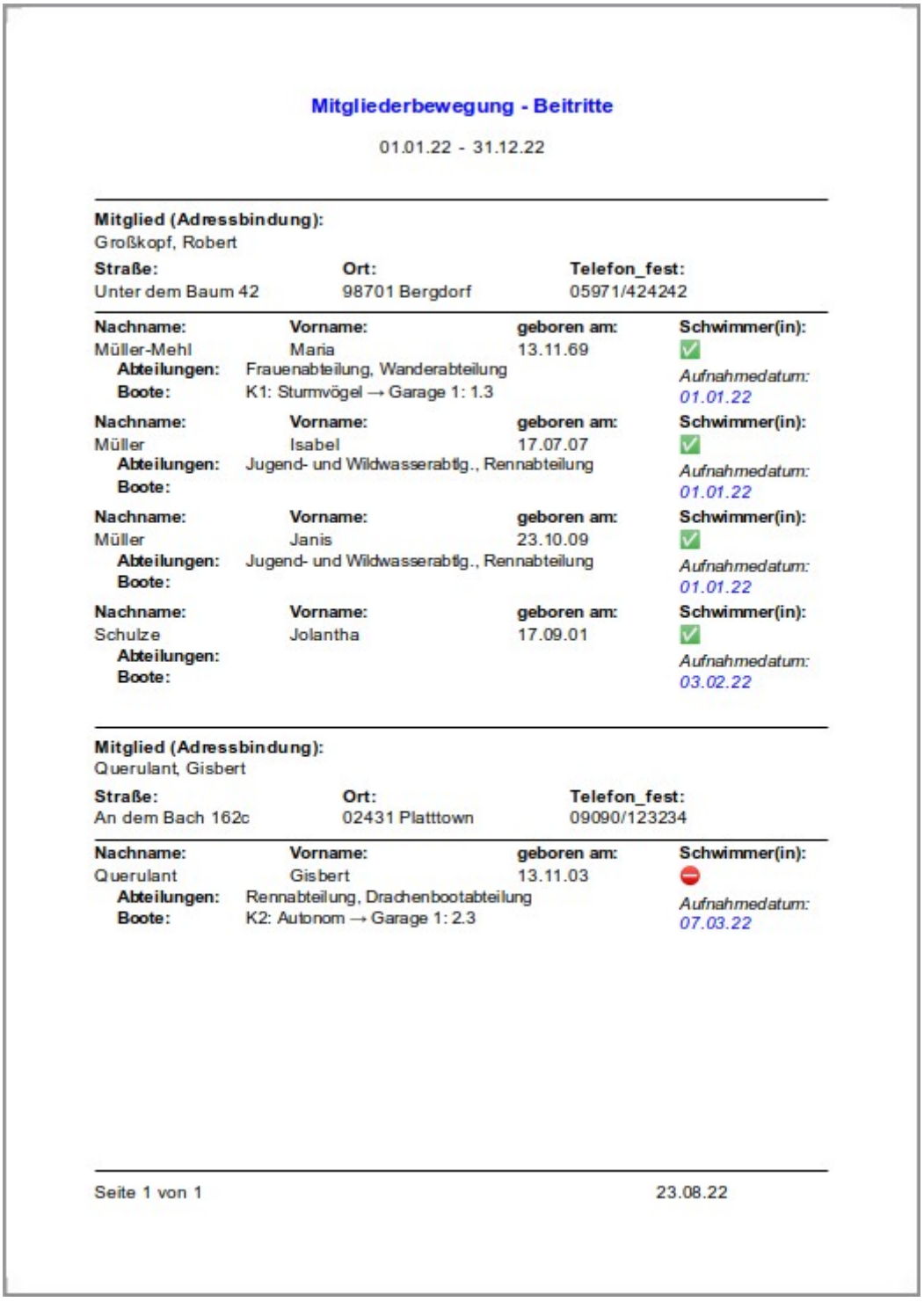
Datenquelle
Abfrage: *qry_Jubilaeum*

Die Größe der Bereiche für den Ausdruck dieser Urkunde ist so gewählt, dass die komplette Seite gefüllt wird. Trotzdem ist im Bereich «Detail» noch einmal eingestellt, dass vor dem Bereich ein Seitenumbruch eingefügt werden muss. So ein Umbruch gilt nicht für den Seitenkopf und den Seitenfuß, so dass der komplette Inhalt für die erste Urkunde auch auf der ersten Seite zu sehen ist.



Im Seitenkopf ist ein Bereich für das Logo vorgesehen, dass in "tbl_Einstellungen" festgelegt wurde. In den dafür vorgesehenen grafischen Kontrollfeldern erscheint leider nicht so ein in Hellgrau gehaltener Verweis wie bei den Textfeldern. Das Feld für das Logo muss hier natürlich bei anderen Dimensionen entsprechend angepasst werden. Hier ist das Design auch zusammen mit der Schriftgröße auf ein Logo im Querformat abgestimmt.

rpt_Mitglieder_AusEin



Mitgliederbewegung - Beitritte

01.01.22 - 31.12.22

Mitglied (Adressbindung):
Großkopf, Robert

Straße:	**Ort:**	**Telefon_fest:**
Unter dem Baum 42	98701 Bergdorf	05971/424242

Nachname:	**Vorname:**	**geboren am:**	**Schwimmer(in):**
Müller-Mehl	Maria	13.11.69	✅
Abteilungen:	Frauenabteilung, Wanderabteilung		*Aufnahmedatum:*
Boote:	K1: Sturmvögel → Garage 1: 1.3		*01.01.22*
Nachname:	**Vorname:**	**geboren am:**	**Schwimmer(in):**
Müller	Isabel	17.07.07	✅
Abteilungen:	Jugend- und Wildwasserabtlg., Rennabteilung		*Aufnahmedatum:*
Boote:			*01.01.22*
Nachname:	**Vorname:**	**geboren am:**	**Schwimmer(in):**
Müller	Janis	23.10.09	✅
Abteilungen:	Jugend- und Wildwasserabtlg., Rennabteilung		*Aufnahmedatum:*
Boote:			*01.01.22*
Nachname:	**Vorname:**	**geboren am:**	**Schwimmer(in):**
Schulze	Jolantha	17.09.01	✅
Abteilungen:			*Aufnahmedatum:*
Boote:			*03.02.22*

Mitglied (Adressbindung):
Querulant, Gisbert

Straße:	**Ort:**	**Telefon_fest:**
An dem Bach 162c	02431 Platttown	09090/123234

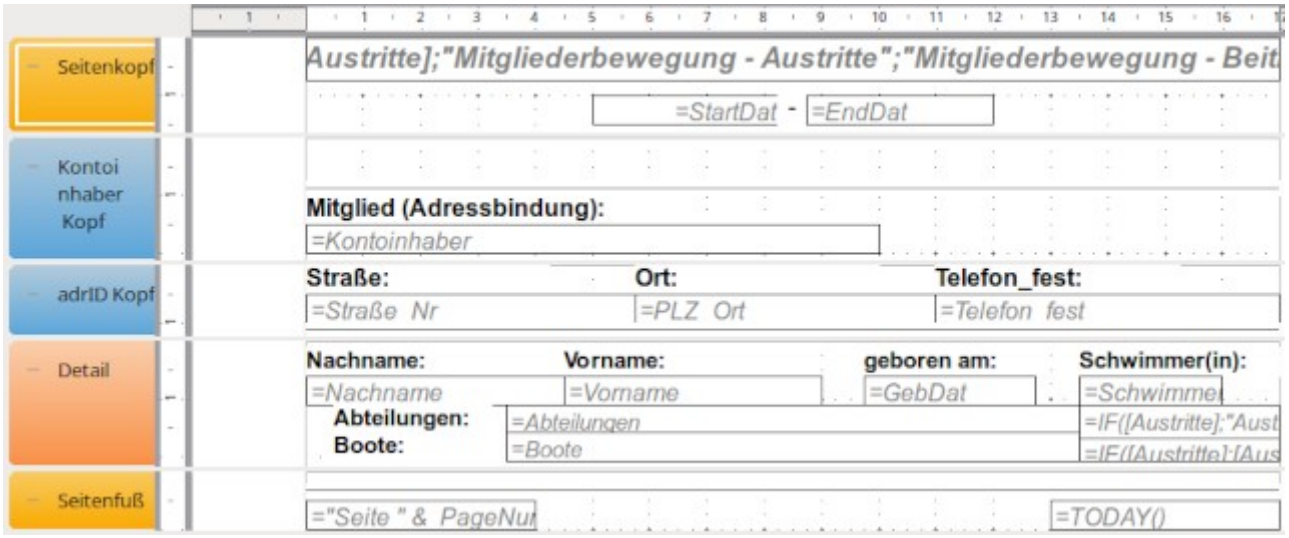
Nachname:	**Vorname:**	**geboren am:**	**Schwimmer(in):**
Querulant	Gisbert	13.11.03	⛔
Abteilungen:	Rennabteilung, Drachenbootabteilung		*Aufnahmedatum:*
Boote:	K2: Autonom → Garage 1: 2.3		*07.03.22*

Seite 1 von 1 23.08.22

Datenquelle
Ansicht: *viw_Filter_Mitglieder_komplett_AusEin*

Abhängig von der Einstellung in "tbl_Filter" werden Austritte oder Beitritte in einem entsprechenden Zeitraum angezeigt. Im Seitenkopf befindet sich deshalb ein Feld für die Überschrift, das mit einer Funktion versehen ist:

```
001 IF([Austritte];"Mitgliederbewegung - Austritte";"Mitgliederbewegung -
    Beitritte")
```



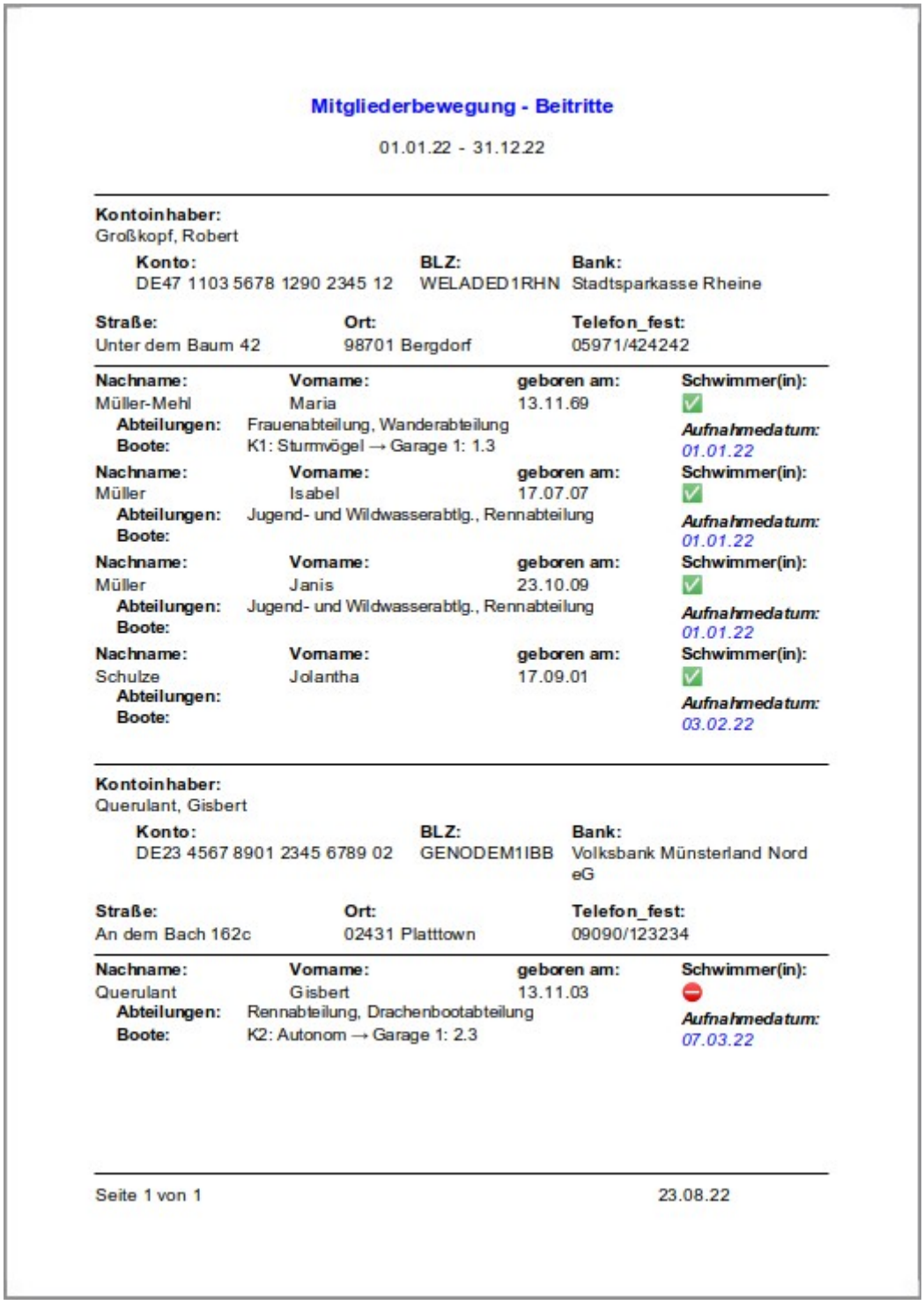
Zu einem Kontoinhaber können Personen mit verschiedenen Adressen zugeordnet werden. Deshalb sind in dem Bericht zwei Gruppierungen enthalten, bei denen zuerst nach dem Kontoinhaber gefragt wird. Allerdings wird diese Gruppierung nur beim folgenden Bericht mit einem Konto beschrieben. Hier steht lediglich die Person, der zuerst die jeweilige Adresse zugewiesen wurde.

Rechts unten im Bereich Detail sind zwei Felder, die abhängig davon, ob Austritte oder Beitritte gezeigt werden, ihren Inhalt entsprechend zeigen:

```
001 IF([Austritte];"Austrittsdatum:";"Aufnahmedatum:")
002 IF([Austritte];[AusDat];[AufDat])
```

In dem oberen Feld wird angezeigt, ob das folgende Feld ein Austrittsdatum oder ein Aufnahmedatum anzeigt. In dem unteren Feld erfolgt dann die Anzeige des "AusDat" oder "AufDat" aus der Ansicht für diesen Bericht.

rpt_Mitglieder_AusEin_Konto



Mitgliederbewegung - Beitritte

01.01.22 - 31.12.22

Kontoinhaber:
Großkopf, Robert

Konto:	BLZ:	Bank:
DE47 1103 5678 1290 2345 12	WELADED1RHN	Stadtsparkasse Rheine

Straße:	Ort:	Telefon_fest:
Unter dem Baum 42	98701 Bergdorf	05971/424242

Nachname:	Vorname:	geboren am:	Schwimmer(in):
Müller-Mehl	Maria	13.11.69	✅
Abteilungen:	Frauenabteilung, Wanderabteilung		*Aufnahmedatum:*
Boote:	K1: Sturmvögel → Garage 1: 1.3		*01.01.22*
Nachname:	**Vorname:**	**geboren am:**	**Schwimmer(in):**
Müller	Isabel	17.07.07	✅
Abteilungen:	Jugend- und Wildwasserabtlg., Rennabteilung		*Aufnahmedatum:*
Boote:			*01.01.22*
Nachname:	**Vorname:**	**geboren am:**	**Schwimmer(in):**
Müller	Janis	23.10.09	✅
Abteilungen:	Jugend- und Wildwasserabtlg., Rennabteilung		*Aufnahmedatum:*
Boote:			*01.01.22*
Nachname:	**Vorname:**	**geboren am:**	**Schwimmer(in):**
Schulze	Jolantha	17.09.01	✅
Abteilungen:			*Aufnahmedatum:*
Boote:			*03.02.22*

Kontoinhaber:
Querulant, Gisbert

Konto:	BLZ:	Bank:
DE23 4567 8901 2345 6789 02	GENODEM1IBB	Volksbank Münsterland Nord eG

Straße:	Ort:	Telefon_fest:
An dem Bach 162c	02431 Platttown	09090/123234

Nachname:	Vorname:	geboren am:	Schwimmer(in):
Querulant	Gisbert	13.11.03	⛔
Abteilungen:	Rennabteilung, Drachenbootabteilung		*Aufnahmedatum:*
Boote:	K2: Autonom → Garage 1: 2.3		*07.03.22*

Seite 1 von 1 23.08.22

Mitgliederbewegung - Austritte

01.01.22 - 31.12.22

Kontoinhaber:
Großkopf, Robert

Konto:	BLZ:	Bank:
DE47 1103 5678 1290 2345 12	WELADED1RHN	Stadtsparkasse Rheine

Straße:	Ort:	Telefon_fest:
Unter dem Baum 42	98701 Bergdorf	05971/424242

Nachname:	Vorname:	geboren am:	Schwimmer(in):
Müller-Mehl	Maria	13.11.69	✅

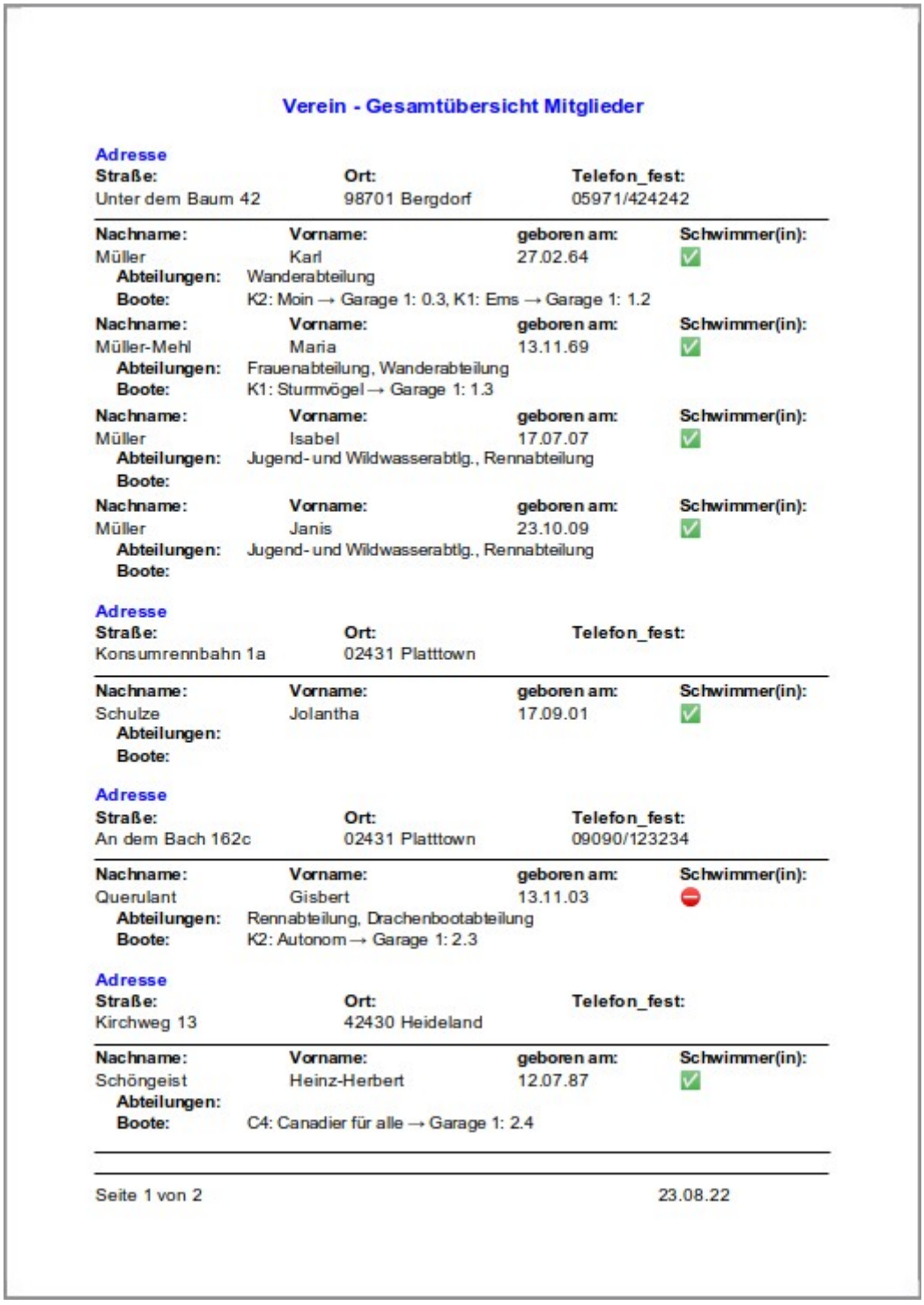
Abteilungen: Frauenabteilung, Wanderabteilung
Boote: K1: Sturmvögel → Garage 1: 1.3

Austrittsdatum:
01.09.22

Datenquelle
Ansicht: *viw_Filter_Mitglieder_komplett_AusEin*

Dieser Bericht unterscheidet sich von dem vorhergehenden nur dadurch, dass jetzt auch eine Gruppierung nach den Kontoeinträgen erfolgt. Der Bericht wird dann als Standard genutzt, wenn in dem Einstellungen die Kontoführung mit angewählt wurde.

rpt_Mitglieder_komplett



Verein - Gesamtübersicht Mitglieder

Adresse

Straße:	Ort:	Telefon_fest:
Unter dem Baum 42	98701 Bergdorf	05971/424242

Nachname:	Vorname:	geboren am:	Schwimmer(in):
Müller	Karl	27.02.64	✅
Abteilungen:	Wanderabteilung		
Boote:	K2: Moin → Garage 1: 0.3, K1: Ems → Garage 1: 1.2		
Nachname:	**Vorname:**	**geboren am:**	**Schwimmer(in):**
Müller-Mehl	Maria	13.11.69	✅
Abteilungen:	Frauenabteilung, Wanderabteilung		
Boote:	K1: Sturmvögel → Garage 1: 1.3		
Nachname:	**Vorname:**	**geboren am:**	**Schwimmer(in):**
Müller	Isabel	17.07.07	✅
Abteilungen:	Jugend- und Wildwasserabtlg., Rennabteilung		
Boote:			
Nachname:	**Vorname:**	**geboren am:**	**Schwimmer(in):**
Müller	Janis	23.10.09	✅
Abteilungen:	Jugend- und Wildwasserabtlg., Rennabteilung		
Boote:			

Adresse

Straße:	Ort:	Telefon_fest:
Konsumrennbahn 1a	02431 Platttown	

Nachname:	Vorname:	geboren am:	Schwimmer(in):
Schulze	Jolantha	17.09.01	✅
Abteilungen:			
Boote:			

Adresse

Straße:	Ort:	Telefon_fest:
An dem Bach 162c	02431 Platttown	09090/123234

Nachname:	Vorname:	geboren am:	Schwimmer(in):
Querulant	Gisbert	13.11.03	⛔
Abteilungen:	Rennabteilung, Drachenbootabteilung		
Boote:	K2: Autonom → Garage 1: 2.3		

Adresse

Straße:	Ort:	Telefon_fest:
Kirchweg 13	42430 Heideland	

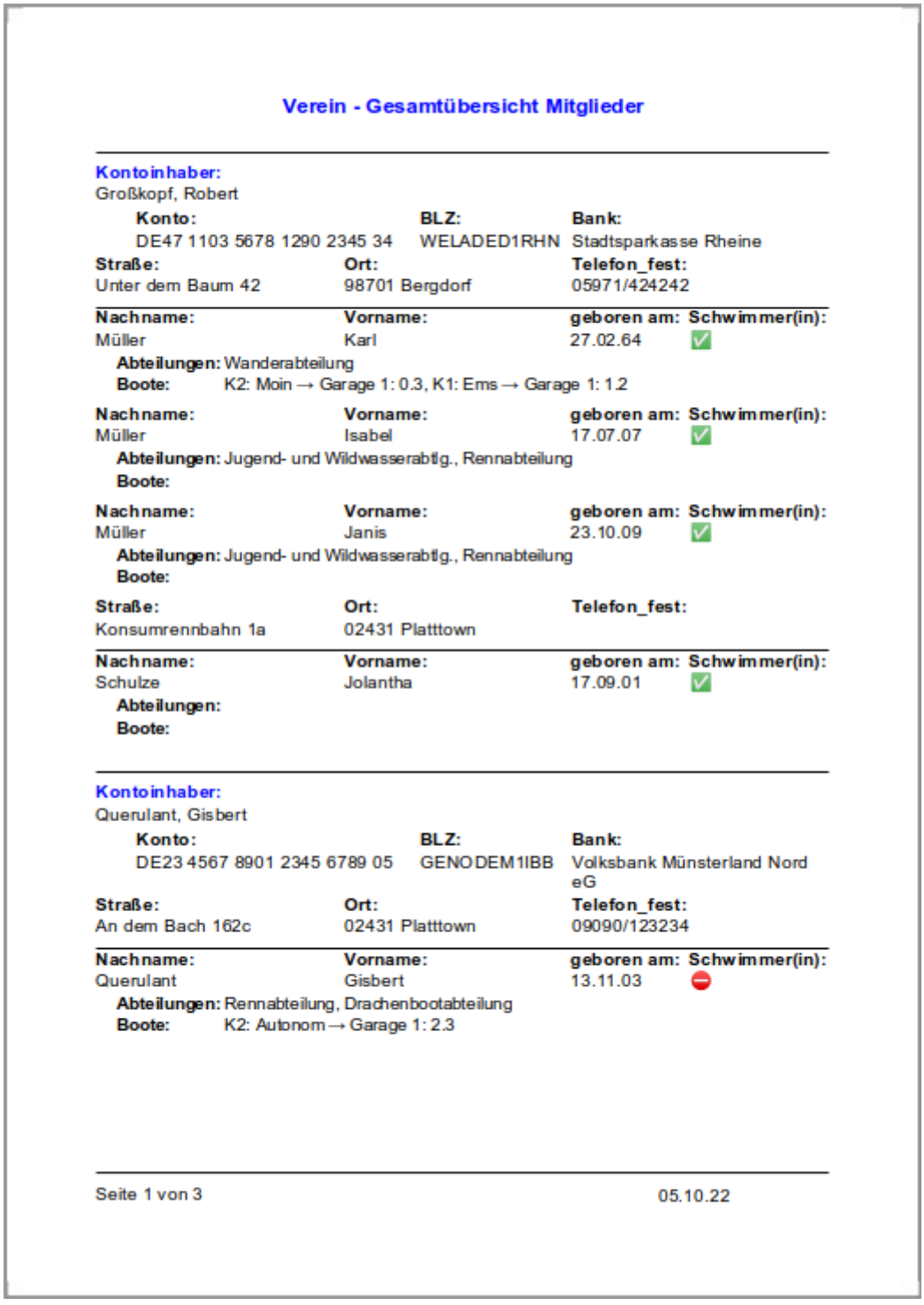
Nachname:	Vorname:	geboren am:	Schwimmer(in):
Schöngeist	Heinz-Herbert	12.07.87	✅
Abteilungen:			
Boote:	C4: Canadier für alle → Garage 1: 2.4		

Seite 1 von 2 23.08.22

Datenquelle
Ansicht: *viw_Mitglieder_komplett*

Hier werden die Mitglieder zu den Adressen gruppiert. Die Reihenfolge der Mitglieder in der Adressengruppe ist egal. Da der Bereich «Detail» über mehrere Zeilen läuft kann es passieren, dass der Inhalt aufgesplittet wird, auch wenn hier **Zusammenhalten → Ja** gesetzt ist.

rpt_Mitglieder_komplett_Konto



Verein - Gesamtübersicht Mitglieder

Kontoinhaber:
Großkopf, Robert

Konto:	BLZ:	Bank:
DE47 1103 5678 1290 2345 34	WELADED1RHN	Stadtsparkasse Rheine

Straße:	Ort:	Telefon_fest:
Unter dem Baum 42	98701 Bergdorf	05971/424242

Nachname:	Vorname:	geboren am:	Schwimmer(in):
Müller	Karl	27.02.64	✅

Abteilungen: Wanderabteilung
Boote: K2: Moin → Garage 1: 0.3, K1: Ems → Garage 1: 1.2

Nachname:	Vorname:	geboren am:	Schwimmer(in):
Müller	Isabel	17.07.07	✅

Abteilungen: Jugend- und Wildwasserabtlg., Rennabteilung
Boote:

Nachname:	Vorname:	geboren am:	Schwimmer(in):
Müller	Janis	23.10.09	✅

Abteilungen: Jugend- und Wildwasserabtlg., Rennabteilung
Boote:

Straße:	Ort:	Telefon_fest:
Konsumrennbahn 1a	02431 Platttown	

Nachname:	Vorname:	geboren am:	Schwimmer(in):
Schulze	Jolantha	17.09.01	✅

Abteilungen:
Boote:

Kontoinhaber:
Querulant, Gisbert

Konto:	BLZ:	Bank:
DE23 4567 8901 2345 6789 05	GENODEM1IBB	Volksbank Münsterland Nord eG

Straße:	Ort:	Telefon_fest:
An dem Bach 162c	02431 Platttown	09090/123234

Nachname:	Vorname:	geboren am:	Schwimmer(in):
Querulant	Gisbert	13.11.03	⛔

Abteilungen: Rennabteilung, Drachenbootabteilung
Boote: K2: Autonom → Garage 1: 2.3

Seite 1 von 3 05.10.22

Datenquelle
Ansicht: *viw_Mitglieder_komplett*

Bei diesem Bericht wird gegenüber dem vorhergehenden das Konto mit angezeigt. Außerdem ist der Bereich «Detail» leer gelassen worden und wird nicht angezeigt. Das hat damit zu tun, dass die Elemente im Bereich «Detail» trotz gegenteiliger Einstellung nicht zusammengehalten werden, wenn ein Seitenumbruch erfolgt. Stattdessen ist gegenüber dem vorherigen Bericht der entsprechende Inhalt in einen Gruppenkopf ausgelagert worden. Der Vorteil dieser Gruppe ist neben dem Zusammenhalten der Informationen auch, dass so eine einheitliche Sortierung der in einem Haus wohnenden Personen nach dem Vornamen erfolgt.

Einem Kontoinhaber können mehrere Adressen zugeordnet werden. Einer Adresse können dann wieder mehrere Personen zugeordnet werden. Das entspricht hier dem Aufbau der Gruppierung.

rpt_Mitgliedsdauer

Mitgliedschaftsdauer
Mitgliedschaft ab 10 Jahren

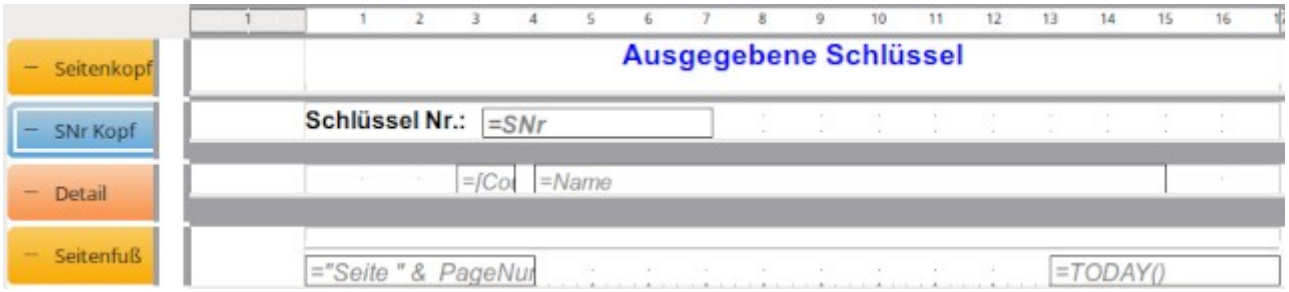
Vorname	Nachname	Aufnahmedatum	Mitgliedsjahre
Helmut-Wilhelm	Oldie	01.07.72	50
Annemarie	Oldie	12.08.82	40
Irina	Oldie	01.03.97	25

Datenquelle
Abfrage: *qry_Mitgliedsdauer*

Dieser Bericht dient lediglich dazu, die Dauer der Mitgliedschaft für Jubiläen zu überprüfen. In Vereinen kommt es immer wieder vor, dass eine Person sich übergangen fühlt und meint "Ich bin schon lange dran ..."

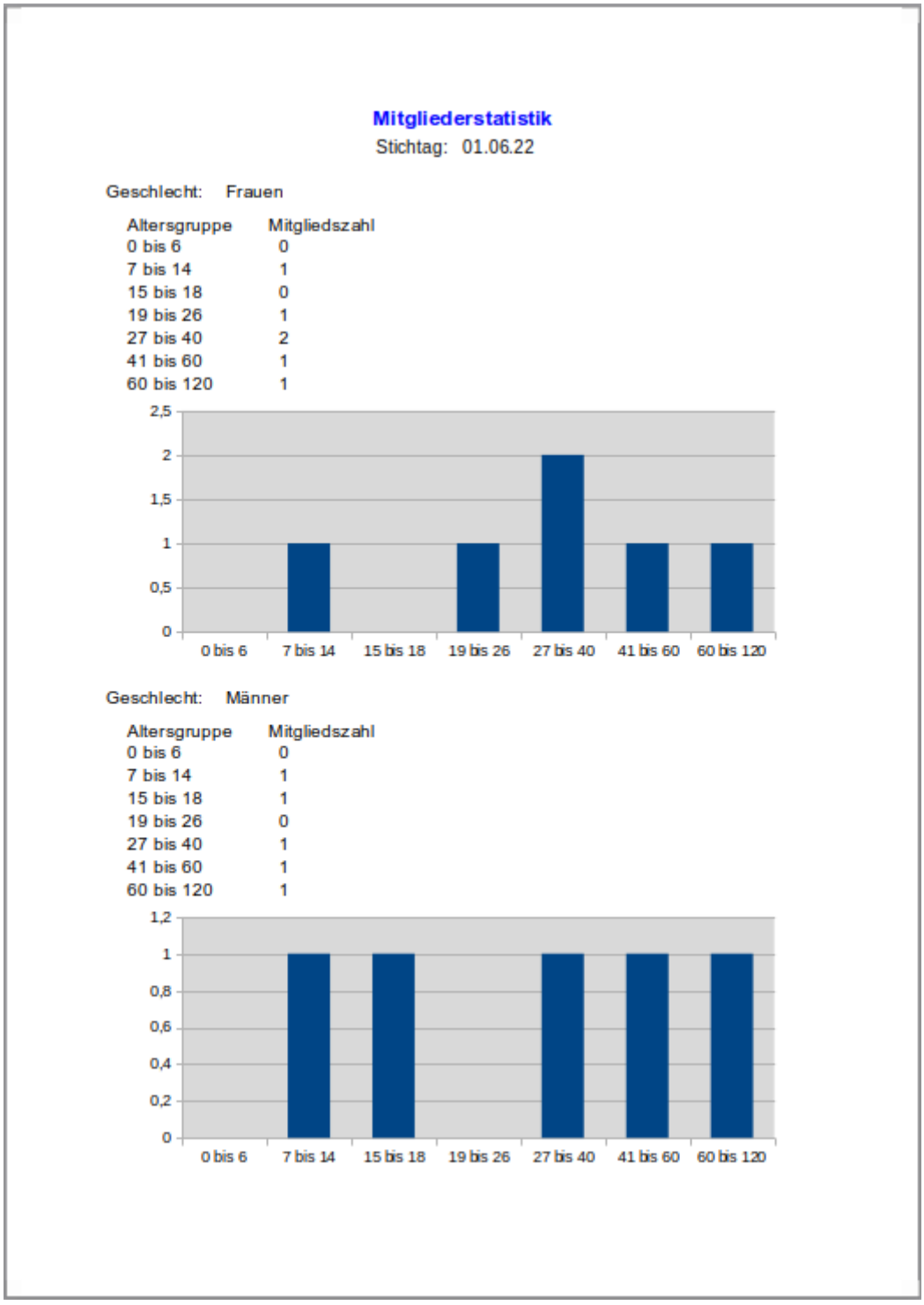
rpt_Schluesseluebersicht

Datenquelle
Abfrage: *qry_Schluessel_Uebersicht*



Hier werden alle Schlüsselausgaben angezeigt. Zu der jeweiligen Schlüsselart werden die Mitglieder benannt, die sich so einen Schlüssel entliehen haben. Vor dem Namen ist durch das Einfügen einer Zählerfunktion (**Eigenschaften → Daten → Datenfeld-Typ → Zähler**) die Nummerierung der Mitglieder dargestellt.

rpt_Statistik



Mitgliederstatistik

Stichtag: 01.06.22

Geschlecht: Frauen

Altersgruppe	Mitgliedszahl
0 bis 6	0
7 bis 14	1
15 bis 18	0
19 bis 26	1
27 bis 40	2
41 bis 60	1
60 bis 120	1

Geschlecht: Männer

Altersgruppe	Mitgliedszahl
0 bis 6	0
7 bis 14	1
15 bis 18	1
19 bis 26	0
27 bis 40	1
41 bis 60	1
60 bis 120	1

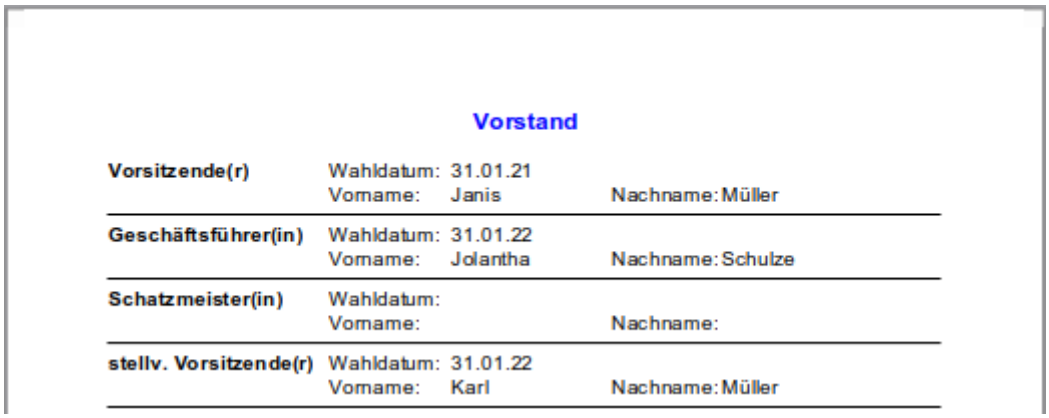
Datenquelle
Ansicht: *viw_Statistik_Alter_Detail*

Für die Erstellung der Diagramme wird über die Steuerelement-Eigenschaften unter **Daten → Inhalt** eine Abfrage innerhalb des Berichtes durchgeführt:

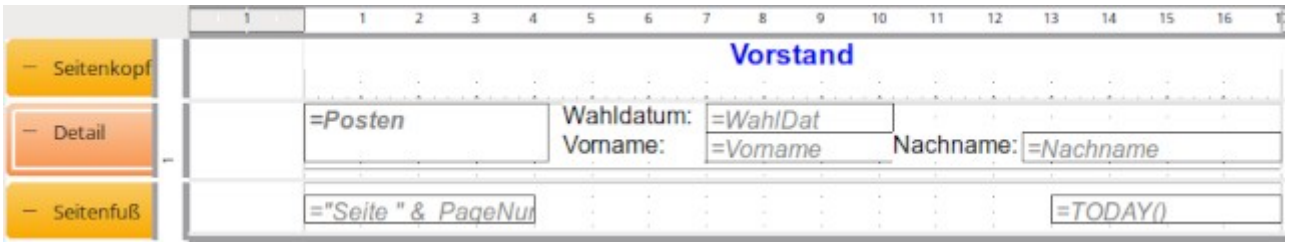
```
001 SELECT "Alter",
002 "Anzahl",
003 "Geschlecht"
004 FROM "viw_Statistik_Alter_Detail"
```

Das ist notwendig, weil sich in der Ansicht auch ein Datumsfeld befindet. Das Diagrammmodul hat leider seit längerem Probleme mit Datumsfeldern, so dass mit Datumsfeldern zusammen Inhalte von Diagrammen grundsätzlich nicht dargestellt werden. Hier deswegen nur die notwendigen Felder "Alter" und "Anzahl" zur Anzeige der Daten und "Geschlecht" zur Verbindung der Diagramme mit der vorhergehenden tabellarischen Übersicht. Bei der Erstellung des Diagramms muss dann auch noch für die Datengrundlage das Feld "Geschlecht" entfernt werden, da der Assistent annimmt, dass über die Abfrage eben zwei Säulen dargestellt werden sollen: Die Säule "Anzahl" und die Säule "Geschlecht". Da im Feld "Geschlecht" kein numerischer Wert liegt fällt nur auf, dass die Säulen für die "Anzahl" nicht mittig über den Bereichen auf der x-Achse liegen.

rpt_Vorstand_aktuell

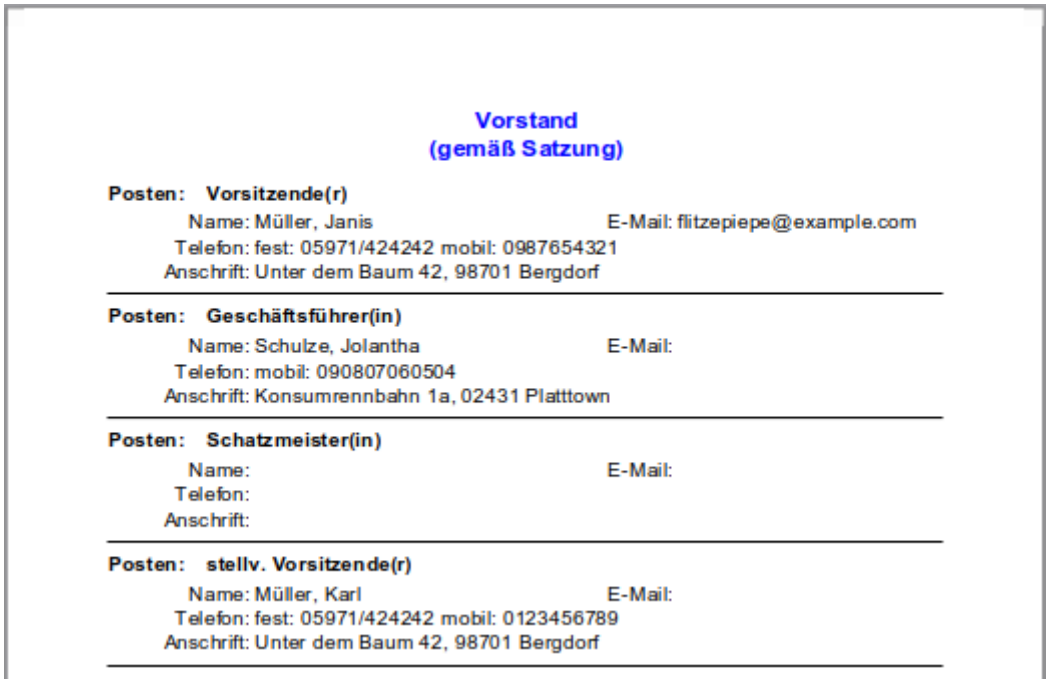


Datenquelle
Abfrage: *qry_Vorstand_aktuell*



Eine einfache Übersicht über alle Vorstandsmitglieder. Jeder Posten wird zusammen mit dem Wahldatum und dem Namen aufgeführt. Der Bereich «Detail» ist zwar auf **Zusammenhalten → Ja** eingestellt, nur funktioniert dies leider nicht unbedingt. Gegebenenfalls müsste also nach einem entsprechenden Kriterium sortiert werden, wobei die Reihenfolge der Posten laut Satzung maßgebend sein sollte.

rpt_Vorstand_aktuell_lt_Satzung



Vorstand
(gemäß Satzung)

Posten: Vorsitzende(r)
Name: Müller, Janis E-Mail: flitzepiepe@example.com
Telefon: fest: 05971/424242 mobil: 0987654321
Anschrift: Unter dem Baum 42, 98701 Bergdorf

Posten: Geschäftsführer(in)
Name: Schulze, Jolantha E-Mail:
Telefon: mobil: 090807060504
Anschrift: Konsumrennbahn 1a, 02431 Platttown

Posten: Schatzmeister(in)
Name: E-Mail:
Telefon:
Anschrift:

Posten: stellv. Vorsitzende(r)
Name: Müller, Karl E-Mail:
Telefon: fest: 05971/424242 mobil: 0123456789
Anschrift: Unter dem Baum 42, 98701 Bergdorf

Datenquelle
Abfrage: *qry_Vorstand_aktuell_lt_Satzung*

Hier eine ausführlichere Übersicht, in der dann nur die Vorstandsmitglieder verzeichnet sind, die lt. Satzung zum Vorstand dazu gehören. Damit der Inhalt pro Posten sicher zusammengehalten wird, wird der Detailbereich nicht genutzt. Die Sortierung der Gruppe erfolgt nach der "ID" aus der "tbl_VorstandPosten". Damit ist die Sortierung immer gleich der Sortierung lt. Satzung.

Makros

Backup

DatabaseBackup

Aufruf aus
Makro: *DatabaseStart*

```
001 SUB DatabaseBackup(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
017    oPath = createUnoService("com.sun.star.util.PathSettings")
018    FOR i = 1 TO inMax + 1
019       IF NOT FileExists(oPath.Backup & "/" & i & "_" & sTitel) THEN
020          IF i > inMax THEN
021             FOR k = inMax - 1 TO 1 STEP -1
022                IF FileDateTime(oPath.Backup & "/" & k & "_" & sTitel) <=
                      FileDateTime(oPath.Backup & "/" & k+1 & "_" & sTitel) THEN
023                   IF k = 1 THEN
024                      i = k
025                      EXIT FOR
026                   END IF
027                ELSE
028                   i = k+1
029                   EXIT FOR
030                END IF
031             NEXT
032          END IF
033          EXIT FOR
034       END IF
035    NEXT
036    sUrl_Ziel = oPath.Backup & "/" & i & "_" & sTitel
037    FileCopy(sUrl_Start,sUrl_Ziel)
038 END SUB
```

Von der Datenbankdatei *.odb wird eine Kopie in das Backup-Verzeichnis erstellt. Die Maximalzahl ist auf den im Aufruf angegebenen Wert beschränkt. Danach wird das älteste Backup überschrieben. Solch ein Backup ist für die internen Datenbanken empfehlenswert. Bei externen Datenbanken und Serverdatenbanken macht es hingegen keinen Sinn, da dort ja die Daten an einem anderen Ort liegen.

In Zeile 10 wird zuerst der Name der Datenbankdatei ausgelesen. Anschließend dann die URL, die zu dieser Datenbankdatei führt. Würde die Prozedur von einem internen Formular und nicht beim Start der Base-Datei ausgeführt, so müsste der Name der Datei und die URL erst über die Schleife von Zeile 12 bis 16 ermittelt werden.

Im Backup-Verzeichnis des Datenbanknutzers von LO soll die Speicherung erfolgen. Das Verzeichnis ist über **Extras → Optionen → LibreOffice → Pfade → Sicherungskopien** einstellbar. Die

Speicherung erfolgt beginnend mit der Nummer, gefolgt durch einen Unterstrich und weiter mit dem ursprünglichen Titel (Zeile 19).

Wenn i größer als die maximale Zahl wird muss nachgesehen werden, welche Kopie die älteste ist, um diese zu ersetzen. Dazu wird ab Zeile 21 rückwärts durch die erstellten Backup-Dateien gesucht. Ist die Datei mit der größeren Nummer älter als die mit der kleineren Nummer, dann wird diese ältere Datei ersetzt. Die Suche beginnt also bei einem Vergleich der Backup-Datei mit der zweithöchsten Nummer mit der Datei mit der höchsten Nummer und wird abwärts fortgesetzt, immer im direkten Vergleich mit der nächst höheren Ziffer. Ist 1 erreicht und immer noch vor 2 erstellt worden, so wird 1 ersetzt.

In Zeile 36 wird der Pfad und Name für die Zieldatei zusammengestellt und schließlich in Zeile 37 eine Kopie der Ausgangsdatei in das Backup-Verzeichnis kopiert.

DatabaseStart

Aufruf aus
Base-Datei über **Extras → Anpassen → Ansicht wurde erzeugt**

Benötigt	
Makro:	*DatabaseBackup, FilterIDSet, Preferences*

```
001  SUB DatabaseStart
002     DatabaseBackup(10)
003     FilterIDSet
004     Preferences
005     ThisDatabaseDocument.FormDocuments.getByName("frm_Mitglied").open
006  END SUB
```

Beim Öffnen der Datenbankdatei wird zuerst ein Backup geschrieben. Hier ist die Anzahl der Backups auf 10 begrenzt worden. Anschließend werden die folgenden beiden Prozeduren ausgeführt. Danach erfolgt dann der Start des Formulars zum Eintragen neuer Mitglieder, da dies das Formular ist, das wohl am häufigsten benötigt wird.

FilterIDSet

Aufruf aus	
Makro:	*DatabaseStart*

Benötigt	
Tabelle:	*tbl_Filter*
Ansicht:	*viw_ServVar*

```
001  SUB FilterIDSet
002     DIM oDatasource AS OBJECT
003     DIM oConnection AS OBJECT
004     DIM oSQL_Statement AS OBJECT
005     DIM stSql AS STRING
006     DIM oResult AS OBJECT
007     DIM stID AS STRING
008     oDatasource = thisDatabaseDocument.CurrentController
009     IF NOT (oDatasource.isConnected()) THEN oDatasource.connect()
010     oConnection = oDatasource.ActiveConnection()
011     oSQL_Statement = oConnection.createStatement()
012     stSql = "SELECT ""ID"" FROM ""viw_ServVar"" "
013     oResult = oSQL_Statement.executeQuery(stSql)
014     WHILE oResult.next
015        stID = oResult.getString(1)
016     WEND
```

```
017     IF stID = "" THEN
018        stID = "1"
019     END IF
020     stSql = "SELECT ""ID"" FROM ""tbl_Filter"" WHERE ""ID"" = '"+stID+"' "
021     oResult = oSQL_Statement.executeQuery(stSql)
022     IF NOT oResult.next THEN
023        stSql = "INSERT INTO ""tbl_Filter"" (""ID"") VALUES ('"+stID+"')"
024        oSQL_Statement.executeUpdate(stSql)
025     END IF
026  END SUB
```

Für die Filtertabelle soll ein Primärschlüssel gesetzt werden, der der Verbindungs-ID der Datenbank entspricht. Dieses Verfahren ermöglicht es, eine Filter-Tabelle auch bei Serverdatenbanken zu nutzen, bei denen mehrere Nutzer unabhängig voneinander Inhalte über die Tabelle filtern können.

Bei der internen HSQLDB funktioniert leider die Abfrage aus Zeile 12 nicht beständig. Selbst im laufenden Betrieb gibt die Ansicht nicht immer einen entsprechenden Wert wieder sondern kann auch leer bleiben. Aus dem Grunde wird bei einer leeren "ID" stattdessen '1' genutzt (Zeile 18).

Ist ein Datensatz mit entsprechender "ID" bereits in der Tabelle "tbl_Filter" vorhanden, dann wird dieser Datensatz genutzt (Zeile 20 bis 22). Ansonsten wird ein neuer Datensatz mit diesem Primärschlüssel erstellt (Zeile 23 und 24).

Ein vorher erstellter Datensatz mit gleicher Verbindungs-ID bleibt so bestehen. Die Filterungen vom letzten Programmstart bleiben erhalten. Sollte dies nicht gewünscht sein, so kann auch einfach zu Beginn aus der "tbl_Filter" der entsprechende Datensatz gelöscht und anschließend neu erstellt werden.

Preferences

Aufruf aus	
Formular:	*frm_Einstellungen*
Makro:	*DatabaseStart*

Benötigt	
Tabelle:	*tbl_Einstellungen*
Ansicht:	*viw_Gruppe*

```
001  GLOBAL boAccount AS BOOLEAN
002  GLOBAL boSubscription AS BOOLEAN
003  GLOBAL boKey AS BOOLEAN
004  GLOBAL boBoatRack AS BOOLEAN
005  GLOBAL boSection AS BOOLEAN
006  GLOBAL arSubscription() AS STRING
```

```
001  SUB Preferences
002     DIM oDatasource AS OBJECT
003     DIM oConnection AS OBJECT
004     DIM oSQL_Statement AS OBJECT
005     DIM stSql AS STRING
006     DIM oResult AS OBJECT
007     DIM i AS INTEGER
008     oDatasource = thisDatabaseDocument.CurrentController
009     IF NOT (oDatasource.isConnected()) THEN oDatasource.connect()
010     oConnection = oDatasource.ActiveConnection()
011     oSQL_Statement = oConnection.createStatement()
012     stSql = "SELECT ""Kontoverwaltung"", ""Beitragsverwaltung"",
           ""Schluesselverwaltung"", ""Bootstaenderverwaltung"",
           ""Abteilungsverwaltung"" FROM ""tbl_Einstellungen"" WHERE ""ID"" = True "
013     oResult = oSQL_Statement.executeQuery(stSql)
```

```
014     WHILE oResult.next
015        boAccount = oResult.getBoolean(1)
016        boSubscription = oResult.getBoolean(2)
017        boKey = oResult.getBoolean(3)
018        boBoatRack = oResult.getBoolean(4)
019        boSection = oResult.getBoolean(5)
020     WEND
021     IF boSubscription THEN
022        stSql = "SELECT ""ID"" FROM ""viw_Gruppe"" WHERE ""Alter_Beitrag"" = 17"
023        oResult = oSQL_Statement.executeQuery(stSql)
024        WHILE oResult.next
025           ReDim Preserve arSubscription(i)
026           arSubscription(i) = oResult.getString(1)
027           i = i + 1
028        WEND
029     END IF
030  END SUB
```

Für die Voreinstellungen der Module muss klar sein, welche Module denn angewählt wurden. Nur so lässt sich regeln, ob bestimmte Elemente bei Formularen nicht von vornherein ausgeblendet werden sollen.

Für die Beitragsverwaltung ist hier gleich ein kleines Makro mit integriert, dass die Gruppen herausfiltert, bei denen im Moment Mitglieder 17 Jahre alt sind. Bei dem Übergang von 17 Jahren zu 18 Jahren fallen diese Mitglieder gegebenenfalls aus dem Gruppenbeitrag heraus.

Zuerst werden die globalen Variablen außerhalb der Prozedur deklariert. So stehen sie die komplette Zeit ab dem Öffnen der Datenbankdatei allen anderen Prozeduren zur Verfügung. In Zeile 12 wird der Abfragecode vorformuliert, in der Schleife von Zeile 14 bis Zeile 20 werden dann die Werte aus der Tabelle "tbl_Einstellungen" in die entsprechenden Variablen geschrieben. Eine Schleife wäre hier eigentlich gar nicht nötig, da ja genau nur ein Datensatz in der "tbl_Einstellungen" existiert.

Ist «boSubcription» (also "Beitragsverwaltung" aus der "tbl_Einstellungen") wahr, dann erfolgt von Zeile 22 bis Zeile 27 die Übertragung von allen Gruppen mit 17-jährigen Mitgliedern in ein Array. Dies wird später benötigt, wenn eventuell Beiträge für die nächste Abbuchung neu bestimmt werden sollen.

Design

PreferencesHide

Aufruf aus
Formular: *frm_Einstellungen*

```
001  SUB PreferencesHide(oEvent AS OBJECT)
002     oForm = oEvent.Source
003     IF boSubscription <> True THEN
004        FOR k = 0 TO oForm.Count - 1
005           oField = oForm.getByIndex(k)
006           IF oField.ServiceName <> "stardiv.one.form.component.Form" AND
                 oField.ServiceName <> "stardiv.one.form.component.Hidden" THEN
007              oField.EnableVisible = False
008           END IF
009        NEXT
010     END IF
011  END SUB
```

Das Formular *frm_Einstellungen* enthält einen großen Bereich zur Beitragsverwaltung. Ist die Variable boSubscripion nicht **True**, dann ist die Beitragsverwaltung abgewählt. Unter diesen Umständen wird im Formular *frm_Einstellungen* alles, was mit der Beitragsverwaltung zu tun hat, unsichtbar geschaltet (Zeile 7). Lediglich Formulare und bereits versteckte Kontrollfelder

lassen sich nicht unsichtbar schalten, da sie das bereits sind. Deswegen sind in Zeile 6 diese beiden Elemente ausgenommen.

BoolChange

Aufruf aus
Formular: *frm_Schluessel, frm_Mitglied*
Makro: *BoolChangeKey*

Benötigt
Makro: *BoolStart*

```
012 SUB BoolChange(oEvent AS OBJECT)
013    DIM oField AS OBJECT
014    DIM oForm AS OBJECT
015    oField = oEvent.Source.Model
016    oForm = oField.Parent
017    IF oField.Label = stChecked THEN
018       oField.Label = stUnChecked
019       oForm.updateBoolean(oForm.findColumn(oField.Tag),false)
020    ELSE
021       oField.Label = stChecked
022       oForm.updateBoolean(oForm.findColumn(oField.Tag),true)
023    END IF
024 END SUB
```

Die Variablen «stUnChecked» und «stChecked» sind globale Variablen, Sie werden durch die Prozedur «BoolStart» mit Inhalt gefüllt. Wenn die Beschriftung eines Feldes dem Text «stChecked» entspricht (Zeile 7), dann wird diese Beschriftung geändert und dann das Feld der Tabelle, das in den Zusatzinformationen der Schaltfläche benannt ist, als nicht ausgewählt (**false**) markiert.

BoolChangeKey

Aufruf aus
Formular: *frm_Schluessel, frm_Mitglied*

Benötigt
Makro: *BoolChange*

```
001 SUB BoolChangeKey(oEvent AS OBJECT)
002    IF oEvent.KeyCode <> 1282 THEN
003       BoolChange(oEvent)
004    END IF
005 END SUB
```

Das Symbol auf dem Button soll nicht ausgetauscht werden, wenn der Button nur mit dem Tabulator angesprungen wird, wohl aber, wenn z. B. die Leertaste gedrückt wird. Hier wird mit dem KeyCode[13] 1282 auf den Tabulator gezielt. Nur wenn nicht der Tabulator auf der Schaltfläche ausgelöst wird wird die Prozedur «BoolChange» aufgerufen.

13 KeyCodes siehe:
http://api.libreoffice.org/docs/idl/ref/namespacecom_1_1sun_1_1star_1_1awt_1_1Key.html

BoolStart

Aufruf aus
Formular: *frm_Mitglied*

```
001  GLOBAL stChecked AS STRING
002  GLOBAL stUnChecked AS STRING
```

```
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
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
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
```

Für die Darstellung von Markierfeldern werden hier Schaltflächen benutzt. Markierfelder können nicht vom Design her angepasst werden. Dies stört vor allem bei der Größe der zu markierenden Fläche. Die Symbole " " und "☐" stammen aus dem UTF-8-Zeichensatz. Das Symbol für die 'Ja' ist dabei eindeutig. Ein vorher ausgewähltes Stop-Symbol für 'Nein' (siehe z. B. *viw_Mitglieder_komplett*) ist für eine allgemeine Verwendung zu wertend. 'Nein' wäre, wie bei den fehlenden Schwimmkenntnissen in der genannten Ansicht, negativ belegt. Bei einem Verlust eines Schlüssels wird aber gerade mit 'Ja' bestätigt, dass der Schlüssel weg ist. Daher also einfach ein leere Kasten für "nicht angekreuzt".

In den Formularen wird ein verstecktes Feld positioniert, das mit dem Namen "hidCheckbox" angesprochen werden kann. In diesem Feld sind alle Felder, durch ein Semikolon getrennt, aufgeführt, die als «grafische Markierfelder» dienen sollen (Zeilen 12 und 13).

Die «grafischen Markierfelder» werden in Zeile 18 auf **false** voreingestellt, wenn der Datensatz noch nicht existiert. Ansonsten wird aus dem zugehörigen Tabellenfeld ermittelt, ob es ausgewählt oder nicht ausgewählt ist. Danach wird dann auf **true** oder **false** eingestellt. Welche Tabellenfeldern mit welchen als «grafische Markierfelder» dienenden Schaltflächen zusammen hängen steht in den Zusatzinformationen (**Tag**) der jeweiligen Schaltfläche.

AfterRecordChange

Aufruf aus
Formular: *frm_Mitglied*

Benötigt
Makro: *BoolStart, Group, SelectGroup*

```
001 SUB AfterRecordChange(oEvent AS OBJECT)
002    BoolStart(oEvent)
003    Group(oEvent)
004    SelectGroup(oEvent)
005 END SUB
```

Diese Prozedur dient lediglich dazu, mit einem Prozedurstart nacheinander drei verschiedene Prozeduren ablaufen zu lassen.

Group

Aufruf aus
Makro: *AfterRecordChange*

Benötigt
Tabelle: *tbl_Adresse, tbl_Mitglied*

```
001 SUB Group(oEvent AS OBJECT)
002    DIM arForms()
003    DIM arFormNames()
004    DIM i AS INTEGER
005    DIM k AS INTEGER
006    DIM oFormTarget AS OBJECT
007    DIM boLeader AS BOOLEAN
008    DIM stID AS STRING
009    DIM stIDAccount AS STRING
010    DIM stIDAddress AS STRING
011    DIM stadrID AS STRING
012    DIM stgruID AS STRING
013    DIM oConnection AS OBJECT
014    DIM oSQL_Statement AS OBJECT
015    oForm = oEvent.Source
016    stID = oForm.getString(oForm.findColumn("ID"))
017    oConnection = oForm.activeConnection()
018    oSQL_Statement = oConnection.createStatement()
019    IF stID <> "" THEN
020       stSql = "SELECT ""mitID"" FROM ""tbl_Adresse"" WHERE ""mitID"" = '"+stID+"'"
021       oResult = oSQL_Statement.executeQuery(stSql)
022       WHILE oResult.next
023          stIDAddress = oResult.getString(1)
024       WEND
025       IF boAccount THEN
026          stSql = "SELECT ""mitID"" FROM ""tbl_Konto"" WHERE ""mitID"" = '"+stID+"'"
027          oResult = oSQL_Statement.executeQuery(stSql)
028          WHILE oResult.next
029             stIDAccount = oResult.getString(1)
030          WEND
031       END IF
032       stSql = "SELECT ""tbl_Adresse"".""mitID"", ""tbl_Mitglied"".""Gruppe_mitID""
             FROM ""tbl_Adresse"", ""tbl_Mitglied"" WHERE "
033       stSql = stSql + """tbl_Adresse"".""mitID"" = ""tbl_Mitglied"".""ID"" AND
             ""tbl_Mitglied"".""ID"" = '"+stID+"' AND NOT
             ""tbl_Mitglied"".""Gruppe_mitID"" IS NULL "
034       oResult = oSQL_Statement.executeQuery(stSql)
035       WHILE oResult.next
036          stadrID = oResult.getString(1)
037          stgruID = oResult.getString(2)
038       WEND
039    END IF
040    IF stIDAddress <> "" THEN
```

```
041         boLeader = True
042      ELSE
043         boLeader = False
044      END IF
045      arForms = array("frmAdresse", "frmKonto")
046      FOR i = 0 TO UBound(arForms)
047         oFormTarget = oForm.getByName(arForms(i))
048         FOR k = 0 TO oFormTarget.Count - 1
049            oField = oFormTarget.getByIndex(k)
050            IF oField.ServiceName <> "stardiv.one.form.component.Form" AND
                  oField.ServiceName <> "stardiv.one.form.component.Hidden" THEN
051               IF boLeader = True THEN
052                  IF (arForms(i) = "frmKonto" AND boAccount AND stadrID = stgruID) OR
                        arForms(i) <> "frmKonto" THEN
053                     oField.EnableVisible = True
054                  ELSE
055                     oField.EnableVisible = False
056                  END IF
057               ELSE
058                  oField.EnableVisible = False
059               END IF
060            END IF
061         NEXT
062      NEXT
063      REDIM arForms()
064      arForms = array("frmGruppe", "frmAbteilung")
065      FOR i = 0 TO UBound(arForms)
066         oFormTarget = oForm.getByName(arForms(i))
067         FOR k = 0 TO oFormTarget.Count - 1
068            oField = oFormTarget.getByIndex(k)
069            IF oField.ServiceName <> "stardiv.one.form.component.Form" AND
                  oField.ServiceName <> "stardiv.one.form.component.Hidden" THEN
070               IF (arForms(i) = "frmGruppe" AND
                     (boLeader = True OR boAccount = False))
                     OR (arForms(i) = "frmAbteilung" AND boSection = False)
                     OR (stID = "") THEN
071                  oField.EnableVisible = False
072               ELSE
073                  oField.EnableVisible = True
074               END IF
075            END IF
076         NEXT
077      NEXT
078   END SUB
```

Mit dieser Prozedur soll die Zuweisung von Mitliedern zu einer Gruppe ermöglicht werden.

In Zeile 14 wird zuerst die ID des angezeigten Mitglieds ausgelesen. Dann wird nachgeschaut, ob dieses Mitglied mit einer Adresse direkt verbunden ist (Zeile 20 – Zeile 24). Falls in den Einstellungen die Kontoverwaltung eingestellt wurde (**boAccount = True**), dann wird auch noch nachgeschaut, ob genau dieses Mitglied mit einem Konto direkt verbunden ist (Zeile 26 – Zeile 30). "mitID" ist sowohl in der Tabelle "tbl_Adresse" als auch in der Tabelle "tbl_Konto" Primärschlüssel.

Jetzt wird ermittelt, welcher Gruppe das entsprechende Mitglied angehört, sofern es nicht selbst als erste Person mit einer Adresse oder einem Konto verbunden ist. Mitglieder, die direkt mit Adresse (und Konto) verbunden sind weisen in "tbl_Mitglied"."Gruppe_mitID" keinen Datensatz auf. Die "tbl_Adresse"."mitID" wird in der Variablen **stadrID** (36), die "tbl_Mitglied"."Gruppe_mitID" in der Variablen **stgruID** (37) gespeichert. An diesen Variablen ist zu erkennen, ob die Person eine eigene Adresse, aber mit einer anderen Person gruppiert ein entsprechendes Konto hat.

Ist der Person eine eigene Adresse zugewiesen, so handelt es sich erst einmal um eine Person, die als Gruppenleitung angenommen wird. **boLeader** wird auf **True** gesetzt (Zeile 41).

Von Zeile 45 bis Zeile 62 werden die Inhalte der Unterformulare «frmAdresse» und «frmKonto» von verschiedenen Bedingungen abhängig auf sichtbar oder unsichtbar geschaltet. Beide werden grundsätzlich nur sichtbar geschaltet, wenn es sich um die erste Person handelt, die einer Adresse oder einem Konto zugeordnet wird (Zeile 51). «frmAdresse» erscheint dann auf jeden Fall, «frmKonto» nur dann, wenn die Kontoverwaltung aktiviert wurde (**boAccount = True**) und die in Zeile 36 und 37 ermittelten Werte gleich sind.

Von Zeile 65 bis Zeile 77 werden die Inhalte der Unterformulare «frmGruppe» und «frmAbteilung» von verschiedenen Bedingungen abhängig auf sichtbar oder unsichtbar geschaltet. Bei «frmGruppe» handelt es sich lediglich um ein Feld, in dem die Information zur Gruppenzugehörigkeit steht. Das Geld kann dann nicht angezeigt werden, wenn es sich um einen Gruppenleiter handelt oder eine Kontoverwaltung nicht angewählt ist. Außerdem wird dieses Feld nicht angezeigt, wenn das Hauptformular noch gar keine Daten für ein Mitglied enthält.

Auch für «frmAbteilung» gilt, dass es erst angezeigt wird, wenn ein Mitglied abgespeichert wurde. Zusätzlich muss natürlich die Abteilungsverwaltung eingeschaltet worden sein (**boSection = True**).

Formulare selbst und versteckte Kontrollfelder können nicht unsichtbar geschaltet werden, da es sich hierbei nicht um sichtbare Elemente handelt. Deswegen werden diese beiden Elemente in den entsprechenden Schleifen jeweils von der Unsichtbarschaltung ausgenommen (Zeilen 50 und 69).

Druck

ReportStart

Aufruf aus
Formular: *frm_Berichte*

Benötigt
Makro: *Ergebnisliste_Zeit*

```
001  SUB ReportStart(oEvent AS OBJECT)
002     DIM stReport AS STRING
003     DIM stmitID AS STRING
004     oForm = oEvent.Source.Model.Parent
005     stReport = oForm.getString(oForm.findColumn("Bericht"))
006     IF stReport <> "" THEN
007        IF stReport = "rpt_Austritt" THEN
008           stmitID = oForm.getString(oForm.findColumn("mitID"))
009           IF stmitID = "" THEN
010              msgbox "Für den Bericht zum Austritt muss eine Person ausgewählt
                    werden."
011              EXIT SUB
012           END IF
013        END IF
014        ThisDatabaseDocument.ReportDocuments.getByName(stReport).open
015     ELSE
016        msgbox "Bitte einen Bericht auswählen."
017     END IF
018  END SUB
```

Hiermit wird die Öffnung eines Berichtes veranlasst. Zuerst wird in der Tabelle "tbl_Einstellungen" im Feld "Bericht" (Zeile 5) nachgesehen, welcher Berichtsname gerade über das Listenfeld abgespeichert wurde. Lediglich für das Anschreiben zum Austritt wird nachgeschaut, ob auch eine entsprechende Person ausgewählt wurde, an die der Bericht gehen soll. Ansonsten wird in Zeile 14 der Bericht geöffnet. Ist kein Bericht ausgewählt, so erscheint eine Information, dass die Auswahl vor dem Öffnen erfolgen muss.

Eingabekontrolle

NewBoatAllowed

Aufruf aus
Formular: *frm_Boot*

Benötigt	
Makro:	*SelectSQL*
Tabelle:	*tbl_Mitglied, tbl_BootArt, tbl_Boot*

```
019 SUB NewBoatAllowed
020    DIM stList AS STRING
021    DIM stID AS STRING
022    DIM oForm AS OBJECT
023    DIM oField AS OBJECT
024    DIM inAnzahl AS INTEGER
025    DIM stGruppeID AS STRING
026    oMainForm = ThisComponent.Drawpage.forms.getByName("frmMitglied")
027    oForm = oMainForm.getByName("frmBoot")
028    oConnection = oForm.activeConnection()
029    oSQL_Statement = oConnection.createStatement()
030    stID = SelectSQL(oMainForm.getString(oMainForm.findColumn("ID")))
031    stList = "SELECT COALESCE(""Gruppe_mitID"", ""ID"") AS ""gruID"" FROM
          ""tbl_Mitglied"" WHERE ""ID"" "+stID+""
032    oResult = oSQL_Statement.executeQuery(stList)
033    WHILE oResult.next
034       stGruppeID = SelectSQL(oResult.getString(1))
035    WEND
036    stList = "SELECT LEFT(""tbl_BootArt"".""Bootsart"", 1) AS ""Art"",
          COUNT(""tbl_BootArt"".""ID"") AS ""Anzahl"", "
037    stList = stList + "COALESCE(""tbl_Mitglied"".""Gruppe_mitID"",
          ""tbl_Mitglied"".""ID"") AS ""gruID"" FROM ""tbl_Boot"", ""tbl_Mitglied"",
          ""tbl_BootArt"" "
038    stList = stList + "WHERE ""tbl_Mitglied"".""ID"" = ""tbl_Boot"".""mitID"" AND
          ""tbl_Boot"".""bo_aID"" = ""tbl_BootArt"".""ID"" AND ""gruID"" "+stGruppeID+"
          GROUP BY ""gruID"", ""Art"" "
039    oResult = oSQL_Statement.executeQuery(stList)
040    WHILE oResult.next
041       stArt = oResult.getString(1)
042       inAnzahl = oResult.getInt(2)
043       IF stArt = "C" OR inAnzahl = 3 THEN
044          oForm.AllowInserts = False
045          oForm.getByName("lblKeineNeueingabe").EnableVisible = True
046       ELSE
047          oForm.AllowInserts = True
048          oForm.getByName("lblKeineNeueingabe").EnableVisible = False
049       END IF
050       IF stArt = "K" AND inAnzahl < 3 THEN
051          boBoatAddableK = True
052       ELSE
053          boBoatAddableK = False
054       END IF
055    WEND
056    DIM loRow AS LONG
057    loRow = oForm.getRow()
058    oForm.reload()
059    oForm.absolute(loRow)
060 END SUB
```

Mit dieser Prozedur wird entschieden, ob für die Gruppe noch weitere Boote eingelagert werden können. Dazu wird zuerst einmal die Gruppenmitgliedschaft für das aktuell ausgewählte Mit-

glied ermittelt (Zeile 31 bis Zeile 35). Anschließend wird die Bootsart und die Anzahl der boote für diese Gruppe bestimmt (Zeile 36 bis Zeile 42).

Enthält die Bootsart ein 'C' als ersten Buchstaben, so handelt es sich um einen Canadier. Ein weiteres Boot kann nicht eingelagert werden. Auch wenn bereits 3 Boote eingelagert wurden, kann kein weiteres Boot hinzugefügt werden. Das Formular wird so eingestellt, dass keine Neueingaben erfolgen können (Zeile 44). Außerdem wird das Feld sichtbar gemacht, dass auf die nicht erlaubte Neueingabe hinweist (Zeile 45). Eine Änderung der vorherigen Eingabe ist allerdings möglich.

Sind noch nicht 3 Boote eingelagert und ist auch kein Canadier eingelagert, so wird in den Formulareinstellungen die Neueingabe erlaubt (47) und der Hinweis auf die nicht erlaubte Eingabe unsichtbar geschaltet (48).

Handelt es sich bei den bisherigen Booten um Kajaks und ist die Anzahl von 3 Booten noch nicht erreicht, dann können weitere Boote hinzugefügt werden. Die globale Variable **boBoatAddableK** wird auf **True** gesetzt (51).

Zum Schluss der Prozedur wird die aktuelle Zeilennummer ausgelesen, das Formular neu geladen und wieder auf die entsprechende Zeilennummer eingestellt, damit die Formulareinstellungen wirksam werden.

SelectBoat

Aufruf aus
Formular: *frm_Boot*

Benötigt
Tabelle: *tbl_BootArt*

```
001 GLOBAL boBoatAddableK AS BOOLEAN
```

```
001 SUB SelectBoat(oEvent AS OBJECT)
002    DIM stSql(0) AS STRING
003    DIM stList AS STRING
004    DIM stID AS STRING
005    DIM oForm AS OBJECT
006    DIM oField AS OBJECT
007    oForm = oEvent.Source
008    IF hasUnoInterfaces(oForm, "com.sun.star.form.XForm" ) THEN
009    IF boBoatAddableK THEN
010       stList = "SELECT ""Bootsart"", ""ID"" FROM ""tbl_BootArt"" WHERE
             LEFT(""Bootsart"", 1) = 'K' ORDER BY ""Bootsart"" ASC "
011    ELSE
012       stList = "SELECT ""Bootsart"", ""ID"" FROM ""tbl_BootArt"" ORDER BY
             ""Bootsart"" ASC "
013    END IF
014    oField = oForm.getByName("lboBootsart")
015    stSql(0) = stList
016    oField.ListSource = stSql
017    oField.refresh
018    SelectBoatRack(oEvent)
019    END IF
020 END SUB
```

Es kann nur ein Canadier oder maximal drei Kajaks im Bootshaus unter gebracht werden. Belegt ein Kajak der Gruppe bereits einen Bootsständer, so werden in dieser Liste nur noch weitere Kajaks zur Auswahl angeboten. Ob überhaupt noch ein Boot unter gebracht werden darf entscheidet dabei die Prozedur *NewBoatAllowed*.

SelectBoatRack

Aufruf aus
Formular: *frm_Boot*

Benötigt
Tabelle: *tbl_BootStaender, tbl_BootHalle, tbl_Boot*

```
001 SUB SelectBoatRack(oEvent AS OBJECT)
002    DIM stSql(0) AS STRING
003    DIM stList AS STRING
004    DIM stID AS STRING
005    DIM oForm AS OBJECT
006    DIM oField AS OBJECT
007    oForm = oEvent.Source
008    IF hasUnoInterfaces(oForm, "com.sun.star.form.XForm" ) THEN
009       IF NOT oForm.MaxRows = 0 THEN
010       stID = SelectSQL(oForm.getString(oForm.findColumn("bo_sID")))
011       oField = oForm.getByName("lboBootsstaender")
012       stList = "SELECT ""tbl_BootHalle"".""Halle_lang""||' →
             '||""tbl_BootStaender"".""BootStaend"" AS ""Bootstaender"",
             ""tbl_BootStaender"".""ID"" "
013       stList = stList + "FROM ""tbl_BootStaender"", ""tbl_BootHalle"" WHERE
             ""tbl_BootStaender"".""bo_hID"" = ""tbl_BootHalle"".""ID"" "
014       stList = stList + "AND (""tbl_BootStaender"".""ID"" NOT IN (SELECT ""bo_sID""
             FROM ""tbl_Boot"") OR ""tbl_BootStaender"".""ID"" "+stID+") "
015       stList = stList + "ORDER BY ""Bootstaender"" ASC"
016       stSql(0) = stList
017       oField.ListSource = stSql
018       oField.refresh
019       END IF
020    END IF
021 END SUB
```

Bootsständer können nur für ein Boot vergeben werden. Bei einer Auswahlliste dürfen also keine Bootsständer angeboten werden, die bereits vergeben wurden. Dies würde aber dazu führen, dass bei bestehenden Datensätzen auch keine Bootsständer angezeigt werden können. Deswegen muss per Makro neben den freien Bootsständern auch der Bootsständer in die Liste aufgenommen werden, der mit dem aktuellen Boot (Zeile 10) zusammenhängt.

SelectSQL

Aufruf aus
Makro: *NewBoatAllowed*

```
001 FUNCTION SelectSQL(stVar AS STRING) AS STRING
002    stVar = Trim(stVar)
003    IF InStr(stVar,"'") THEN
004       stVar = Join(Split(stVar,"'"),"''")
005    END IF
006    IF stVar = "" THEN
007       stVar = "IS NULL"
008    ELSE
009       stVar = "='"+stVar+"'"
010    END IF
011    SelectSQL = stVar
012 END FUNCTION
```

Hier geht es darum, dass Variablen so vorbereitet werden, dass sie problemlos in die Datenbank per SQL übertragen werden.

In Zeile 4 werden einfache Anführungszeichen mit einfachen Anführungszeichen maskiert. Sonst würde SQL das einfache Anführungszeichen als Ende eines Textes nehmen und könnte die anschließenden Daten nicht mehr einlesen.

Ist die Variable leer, so soll **IS NULL** zurückgegeben werden (Zeile 6 bis ^). Ansonsten wird ein Gleichheitszeichen vor die Variable gesetzt und außerdem der anschließende Inhalt in einfache Anführungszeichen geschrieben.

Im SQL-Code selbst entfällt so das Gleichheitszeichen und die einfachen Anführungszeichen. Dadurch ist es auch möglich, ein Feld daraufhin abzufragen, ob es leer ist.

SelectGroup

Aufruf aus
Makro: *AfterRecordChange*

Benötigt
Tabelle: *tbl_Adresse, tbl_Ort, tbl_Einstellungen, tbl_Mitglied, tbl_Konto, tbl_Bank*

```
001 SUB SelectGroup(oEvent AS OBJECT)
002    DIM stSql(0) AS STRING
003    DIM stList AS STRING
004    DIM stID AS STRING
005    DIM oForm AS OBJECT
006    DIM oField AS OBJECT
007    DIM stadrID AS STRING
008    DIM stkonID AS STRING
009    oForm = oEvent.Source
010    stID = SelectSQL(oForm.getString(oForm.findColumn("ID")))
011    oField = oForm.getByName("lboGruppe_mitID")
012    oConnection = oForm.activeConnection()
013    oSQL_Statement = oConnection.createStatement()
014    stList = "SELECT ""mitID"" FROM ""tbl_Adresse"" WHERE ""mitID"" "+stID+""
015    oResult = oSQL_Statement.executeQuery(stList)
016    WHILE oResult.next
017       stadrID = oResult.getString(1)
018    WEND
019    IF stadrID <> "" AND boAccount THEN ' Konto anzeigen zur Auswahl
020       stList = "SELECT DISTINCT '' AS ""Gruppe"", NULL AS ""ID"", -1 AS ""Sort""
             FROM ""tbl_Einstellungen"" UNION "
021       stList = stList + "SELECT DISTINCT 'Neu: Konto existiert noch nicht.'
             AS ""Gruppe"", NULL AS ""ID"", 0 AS ""Sort"" FROM ""tbl_Einstellungen""
             UNION "
022       stList = stList + "SELECT 'IBAN: '||""tbl_Konto"".""IBAN""||', BIC:
             '||""tbl_Bank"".""BIC""||', Bank: '||""tbl_Bank"".""Bank"" AS ""Gruppe"",
             ""tbl_Mitglied"".""ID"", 1 AS SORT "
023       stList = stList + "FROM ""tbl_Mitglied"", ""tbl_Konto"", ""tbl_Bank"" WHERE
             ""tbl_Mitglied"".""ID"" = ""tbl_Konto"".""mitID"" AND
             ""tbl_Konto"".""banID"" = ""tbl_Bank"".""ID"" "
024       stList = stList + "AND (""tbl_Mitglied"".""Gruppe_mitID"" IS NULL OR
             ""tbl_Mitglied"".""ID"" = ""tbl_Mitglied"".""Gruppe_mitID"") ORDER BY
             ""Sort"", ""Gruppe"" "
025    ELSE ' Adresse anzeigen zur Auswahl
026       stList = "SELECT DISTINCT '' AS ""Gruppe"", NULL AS ""ID"", -1 AS ""Sort""
             FROM ""tbl_Einstellungen"" UNION "
027       stList = stList + "SELECT DISTINCT 'Neu: Personengruppe existiert noch
             nicht.' AS ""Gruppe"", NULL AS ""ID"", 0 AS ""Sort"" FROM
             ""tbl_Einstellungen"" UNION "
028       stList = stList + "SELECT COALESCE ( ""tbl_Adresse"".""Straße_Nr"" || ' ' ||
             ""tbl_Ort"".""Postleitzahl"" || ' ' || ""tbl_Ort"".""Ort"", 'keine
             Adresse' ) || "
```

```
029         stList = stList + "COALESCE ( ', Festnetz: ' ||
               ""tbl_Adresse"".""Telefon_fest"", ', kein Festnetztelefon' ) AS
               ""Gruppe"", ""tbl_Mitglied"".""ID"", 1 AS SORT "
030         stList = stList + "FROM ""tbl_Mitglied"", ""tbl_Adresse"", ""tbl_Ort"" WHERE
               ""tbl_Mitglied"".""ID"" = ""tbl_Adresse"".""mitID"" AND
               ""tbl_Adresse"".""ortID"" = ""tbl_Ort"".""ID"" "
031         stList = stList + "AND (""tbl_Mitglied"".""Gruppe_mitID"" IS NULL OR
               ""tbl_Mitglied"".""ID"" = ""tbl_Mitglied"".""Gruppe_mitID"") ORDER BY
               ""Sort"", ""Gruppe"" "
032      END IF
033      stSql(0) = stList
034      oField.ListSource = stSql
035      oField.refresh
036   END SUB
```

Hiermit wird ein Listenfeld abhängig von den Voreinstellungen der Datenbank entweder mit Kontodaten oder mit Adressdaten gefüllt.

Existiert bereits eine Adresse und gibt es eine Kontoverwaltung, so wird das Konto angezeigt (Zeile 20). Soll die Datenbank keine Kontoverwaltung enthalten, so wird grundsätzlich die Adresse zur Auswahl für eine Gruppenzuordnung angezeigt (Zeile 25).

Jedes der Listenfelder enthält dabei eine Zeile zum Zurücksetzen des vorherigen Wertes und eine Zeile, mit der ggf. neue Datensätze eingefügt werden können, die eine Kontozuordnung oder eine Adresszuordnung ermöglichen. Diese Zeilen (21 und 27) ermöglichen, dass im Formular entsprechend die Formularfelder für die Neueingabe eingeblendet werden.

BoardFilterSelect

Aufruf aus
Formular: *frm_Vorstand*

```
001   SUB BoardFilterSelect(oEvent AS OBJECT)
002      DIM stmitID AS STRING
003      DIM stDate AS STRING
004      oField = oEvent.Source.Model
005      oForm = oField.Parent
006      oListField = oForm.getByName("lboMitglied")
007      oDateField = oForm.getByName("datWahlDat")
008      oFilterForm = oForm.Parent.GetByName("frmFilter")
009      stmitID = oFilterForm.getString(oFilterForm.findColumn("mitID"))
010      IF stmitID <> "" THEN
011         IF oForm.IsNew THEN
012            oListField.BoundField.updateString(stmitID)
013         END IF
014      END IF
015      stDate = oFilterForm.getString(oFilterForm.findColumn("WahlDat"))
016      IF stDate <> "" THEN
017         IF oForm.IsNew THEN
018            oDateField.BoundField.updateString(stDate)
019         END IF
020      END IF
021      oField.commit
022   END SUB
```

Bei einer Auswahl des Vorstandspostens soll eine Anzeige in dem unteren Tabellenkontrollfeld übertragen werden. Hierzu werden die Einzelfelder für die Anzeige des Mitglieds (Zeile 6) und das Wahldatum (Zeile 7) unter den Umständen mit einem Update versehen, wenn der Datensatz komplett neu ist. Vorherige Einträge sollen nicht überschrieben werden.

Zum Schluss wird der Inhalt des Listenfeldes für den Vorstandsposten übertragen (Zeile 21).

SelectBoard

Aufruf aus
Formular: *frm_Vorstand*

Benötigt	
Tabelle:	*tbl_VorstandPosten, tbl_Mitglied_VorstandPosten*

```
001  SUB SelectBoard(oEvent AS OBJECT)
002     DIM stSql(0) AS STRING
003     DIM stList AS STRING
004     DIM stID AS STRING
005     DIM oForm AS OBJECT
006     DIM oField AS OBJECT
007     DIM i AS INTEGER
008     oForm = oEvent.Source
009     stID = SelectSQL(oForm.getString(oForm.findColumn("vorID")))
010     oFilterForm = oForm.Parent.GetByName("frmFilter")
011     stDate = oFilterForm.getString(oFilterForm.findColumn("WahlDat"))
012     IF stDate <> "" THEN
013        inYear = CInt(Left(stDate,4))
014        i = inYear/2
015        IF i * 2 = inYear THEN
016           stOrder = "ASC"
017        ELSE
018           stOrder = "DESC"
019        END IF
020     ELSE
021        stOrder = "ASC"
022     END IF
023     oField = oForm.getByName("lboVorstandPosten")
024     stList = "SELECT ""Posten"" || ' → ' || CASE WHEN ""Zyklus"" = 'g' THEN
           'gerade Jahreszahlen' WHEN ""Zyklus"" = 'u' THEN 'ungerade Jahreszahlen' ELSE
           'kein Zyklus' END AS ""VPosten"", "
025     stList = stList + """ID"", CASE WHEN ""Zyklus"" = '-' THEN 'h' ELSE ""Zyklus""
           END AS ""Zyklus"" FROM ""tbl_VorstandPosten"" "
026     stList = stList + "WHERE ""ID"" NOT IN (SELECT
           ""tbl_Mitglied_VorstandPosten"".""vorID"" FROM ""tbl_VorstandPosten"",
           ""tbl_Mitglied_VorstandPosten"" "
027     stList = stList + "WHERE ""tbl_VorstandPosten"".""ID"" =
           ""tbl_Mitglied_VorstandPosten"".""vorID"" AND
           ""tbl_VorstandPosten"".""Zyklus"" <> '-' AND
           ""tbl_Mitglied_VorstandPosten"".""EndDat"" IS NULL) "
028     stList = stList + "OR ""ID"" "+stID+" ORDER BY ""Zyklus"" "+stOrder+", ""ID""
           ASC "
029     stSql(0) = stList
030     oField.ListSource = stSql
031     oField.refresh
032  END SUB
```

Der Inhalt für das Listenfeld, mit dem die Vorstandsposten zugewiesen werden, wird abhängig von dem Jahr erstellt. Da die Posten nach geraden und ungeraden Jahren vergeben werden wird von Zeile 14 bis Zeile 19 die Sortierung nach dem Wahldatum vorgenommen. Entweder erscheinen die Posten mit den ungeraden oder mit den geraden Jahreszahlen in der Liste oben.

Es werden nur Posten angezeigt, die zur Zeit frei verfügbar sind. Ausnahme ist der Posten, der mit dem aktuellen Datensatz bereits abgespeichert ist. Wäre das nicht der Fall, dann würden alle Zuweisungen über ein Formular nicht mehr sichtbar sein.

Dieses Makro ist bei dem Formular mit dem Listenfeld verbunden, das als einzelnes Kontrollfeld eingebaut ist (Zeile 23). Das Listenfeld im Tabellenkontrollfeld ist davon nicht berührt.

SelectKey

Aufruf aus
Formular: *frm_Schluessel*

Benötigt
Tabelle: *tbl_Schluessel, tbl_MitgliedSchluessel, tbl_SchluesselAnzahl*

```
001 SUB SelectKey(oEvent AS OBJECT)
002    BoolStart(oEvent)
003    DIM stSql(0) AS STRING
004    DIM stList AS STRING
005    DIM stID AS STRING
006    DIM oForm AS OBJECT
007    DIM oField AS OBJECT
008    oForm = oEvent.Source
009    stID = SelectSQL(oForm.getString(oForm.findColumn("SNr")))
010    oField = oForm.getByName("lboSchluessel")
011    stList = "SELECT ""tbl_Schluessel"".""Bereich"" || '  → ' ||
          ""tbl_Schluessel"".""SNr"" AS ""Schluessel"", ""tbl_Schluessel"".""SNr"" "
012    stList = stList + "FROM ""tbl_Schluessel"" LEFT JOIN ( SELECT COUNT( ""ID"" )
          AS ""Anzahl"", ""SNr"" FROM ""tbl_Mitglied_Schluessel"" "
013    stList = stList + "WHERE ""RueckDat"" IS NULL GROUP BY ""SNr"" ) AS ""a"" ON
          ""tbl_Schluessel"".""SNr"" = ""a"".""SNr"" "
014    stList = stList + "LEFT JOIN ( SELECT SUM( ""Anzahl"" ) AS ""Anzahl"", ""SNr""
          FROM ""tbl_SchluesselAnzahl"" GROUP BY ""SNr"" ) AS ""b"" "
015    stList = stList + "ON ""tbl_Schluessel"".""SNr"" = ""b"".""SNr"" WHERE COALESCE
          ( ""b"".""Anzahl"", 0 ) - COALESCE ( ""a"".""Anzahl"", 0 ) > 0 "
016    stList = stList + "OR ""SNr"" "+stID+" ORDER BY ""Schluessel"" ASC"
017    stSql(0) = stList
018    oField.ListSource = stSql
019    oField.refresh
020 END SUB
```

Mit dieser Prozedur wird ein Listenfeld für die Schlüsselauswahl mit neuem Inhalt gefüllt. Dabei werden nur die Schlüssel angezeigt, bei denen die Anzahl größer als 0 ist. Einzige Ausnahme bildet der Schlüssel, der im aktuellen Datensatz bereits vorher ausgewählt wurde. Wäre dies nicht der Fall, so würden auch bereits vergebene Schlüssel nicht mehr zuverlässig angezeigt, sofern eben kein Schlüssel mehr zur Ausgabe vorhanden ist.

Solch ein Makro funktioniert für Listenfelder nur zuverlässig in Formularen mit einzelnen Kontrollfeldern. Innerhalb eines Tabellenkontrollfeldes wird die Aktualisierung des Listenfeldes nicht zuverlässig durchgeführt, so dass eben manchmal der anzuzeigende Inhalt fehlt.

IBAN_Test

Aufruf aus
Formular: *frm_Einstellungen, frm_Mitglied*

```
001 SUB IBAN_Test(oEvent AS OBJECT)
002    DIM arChar()
003    DIM arCountry()
004    DIM stIBAN AS STRING
005    DIM stStart AS STRING
006    arChar = array("A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M",
          "N", "O", "P", "Q", "R", "S", "T", "U", "V", "W", "X", "Y", "Z")
007    oFieldIBAN = oEvent.Source.Model
008    stIBAN = Join(Split(oFieldIBAN.Text),"")
009    stIBAN = Join(Split(stIBAN,"_")
010    stChar = Left(stIBAN,1)
011    FOR k = LBound(arChar) TO UBound(arChar)
012       IF arChar(k) = stChar THEN
```

```
013            stCharNum1 = k + 10
014            EXIT FOR
015         END IF
016      NEXT
017      stChar = Right(Left(stIBAN,2),1)
018      FOR k = LBound(arChar) TO UBound(arChar)
019         IF arChar(k) = stChar THEN
020            stCharNum2 = k + 10
021            EXIT FOR
022         END IF
023      NEXT
024      stIBAN_PZ = Mid(stIBAN,3,2)
025      FOR x = 5 TO Len(stIBAN) STEP 7
026         stStart = Trim(Str(CDbl(stStart & Mid(stIBAN,x,7)) Mod 97))
027      NEXT
028      IF CDbl(stStart & stCharNum1 & stCharNum2 & stIBAN_PZ) Mod 97 <> 1 THEN
029         msgbox "Die IBAN ist falsch."
030      END IF
031   END SUB
```

Die IBAN soll hier überprüft werden. Dazu wird in Zeile 8 von dem eingegebenen Text die Leerzeichen entfernt. Sind noch irgendwelche Unterstriche der Eingabemaske vorhanden, so stören die bei der Berechnung. Das sollte natürlich nicht vorkommen, da die Länge der IBAN direkt vorgegeben ist, aber es könnte ja passieren, dass jemand einfach das Feld nach den ersten Eingaben verlässt. Das Entfernen der Unterstriche geschieht in Zeile 9.

Für die Berechnung der IBAN müssen die Landeskennungen in Zahlen umgewandelt werden. Dafür ist in Zeile 6 ein Array mit allen Buchstaben nach der Reihenfolge im Alphabet zusammengestellt worden. Der erste Buchstabe wird in Zeile 10 ausgelesen. Anschließend wird untersucht, an welcher Position im Array dieser Buchstabe liegt (Zeilen 11 bis 16). Zu der ermittelten Position (0 für "A", 1 für "B" usw.) wird nach dem Standard für die Prüfberechnung '10' addiert.

Von Zeile 17 bis Zeile 23 erfolgt das gleiche Verfahren für den zweiten Buchstaben. Dadurch wird aus 'DE' die Ziffernkombination '1314'.

In Zeile 24 wird die Prüfziffer von der IBAN abgeschnitten. Von dem Rest der IBAN wird danach ermittelt, wie groß der Rest ist, wenn die gesamte Zahl durch 97 dividiert wird (**MOD 97**). Dies geht wegen der Größe der Zahl nicht in einem Schritt. Stattdessen werden die ersten 7 Ziffern genommen und der Rest hieraus ermittelt, anschließend die folgenden 7 Ziffern an den Rest angehängt usw. Dies Verfahren ist ähnlich der schriftlichen Division, wie es in der Schule praktiziert wird.

Im letzten Schritt (Zeile 28)wird der Rest dieser Division mit der Zahl für den ersten Landesbuchstaben, der Zahl für den zweiten Landesbuchstaben und der IBAN-Prüfziffer nach einmal durch 97 dividiert. Der Rest dieser Division muss '1' sein – nur dann ist die eingegebene IBAN korrekt.

IBAN_Length_Change

Aufruf aus
Formular: *frm_Einstellungen, frm_Mitglied*

```
001 SUB IBAN_Length_Change(oEvent AS OBJECT)
002    DIM i AS INTEGER
003    DIM oSel AS OBJECT
004    oSel = oEvent.Source.Selection
005    oFieldIBAN = oEvent.Source.Model
006    stIBAN = Left(oFieldIBAN.Text, oSel.Min)
007    stLiteralMask = Left(oFieldIBAN.LiteralMask, 2)
008    stEMask = "AANNLNNNNLNNNNLNNNNLNNNNLNNNNLNNNNLNNNNLNN"
009    stLMask = stIBAN & "__ ____ ____ ____ ____ ____ ____ ____ __"
010    IF stLiteralMask <> stIBAN THEN
011       SELECT CASE stIBAN
```

```
012             CASE "DE"
013                i = 27
014             CASE "NL"
015                i = 22
016             CASE "BE"
017                i = 19
018             CASE "LU"
019                i = 24
020             CASE "FR"
021                i = 33
022             CASE "CH"
023                i = 26
024             CASE "AT"
025                i = 24
026             CASE "CZ"
027                i = 29
028             CASE "PL"
029                i = 34
030             CASE "DK"
031                i = 22
032          END SELECT
033          IF i >= 18 THEN
034             oFieldIBAN.EditMask = Left(stEMask,i)
035             oFieldIBAN.LiteralMask = Left(stLMask,i)
036          END IF
037          oEvent.Source.setSelection(oSel)
038       END IF
039    END SUB
```

Mit dieser Prozedur wird das maskierte Feld für die IBAN auf eine neue Eingabemaske eingestellt. Die Grundsätzliche Maske ist so aufgebaut, dass in den ersten beiden Positionen Großbuchstaben erlaubt sind, in den folgenden Positionen nur Zahlen. Nach jeweils 4 Zeichen ist keine Eingabe erlaubt. An diesen Stellen wird ein Leerzeichen eingefügt (Zeile 5). Die maximale IBAN-Länge beträgt 34 Zeichen. Werden die Leerzeichen nach jeweils 4 Zeichen hinzugefügt, dann sind das mit den 8 Leerzeichen maximal 42 Zeichen für das Feld.

Die IBAN ist abhängig von der Landeskennung unterschiedlich lang[14]. So hat die IBAN mit der Kennung 'DE' eine Gesamtlänge von 22 Zeichen. Da über das maskierte Feld eine IBAN mit Leerzeichen nach jeweils 4 Zeichen versehen wird, ergibt sich daraus eine Länge von 22 Zeichen und 22/4 = 5 Rest 2, also 5 Leerzeichen. Deshalb in Zeile 13 die Zuweisung von 27 Zeichen.

In Zeile 34 und 35 wird die Eingabemaske in das Formularfeld übertragen. Die Länge i muss dafür mindestens 18 betragen, da dies die kürzeste IBAN ist, die möglich ist.

Am Anfang der Prozedur wurde in Zeile 4 die aktuelle Cursorposition über die SELECTION ermittelt. Diese Position wird zum Schluss genutzt um den Cursor direkt hinter der Landeskennung zu positionieren. So kann bei der Eingabe ohne Probleme eine Änderung der Landeskennung erfolgen und der Rest des maskierten Feldes wird während der Eingabe angepasst.

Filter

Filtern

Aufruf aus
Formular: *Einzelmessung_System, frm_Berichte, frm_Boot*

```
001 SUB Filtern(oEvent AS OBJECT)
002    DIM oForm AS OBJECT
003    DIM oField AS OBJECT
004    DIM stFormTarget AS STRING
```

14 *https://de.wikipedia.org/wiki/Internationale_Bankkontonummer*

```
005     DIM n AS INTEGER
006     DIM arForm()
007     oField = oEvent.Source.Model
008     oField.commit()
009     stFormTarget = oField.Tag
010     oForm = oField.Parent
011     oForm.updateRow    ' Das Formular wird gespeichert
012     arForm = Split(stFormTarget,",")
013     FOR n = LBound(arForm()) TO UBound(arForm())
014        oForm.Parent.getByName(arForm(n)).reload()
015     NEXT
016 END SUB
```

Von einem Listenfeld aus wird die Filterung von Formularen veranlasst. In Zeile 8 wird der Inhalt des Feldes übertragen. In den Zusatzinformationen (Zeile 9) des Feldes stehen die Formulare, die auf den Filter eingestellt werden sollen. Die Namen der Formulare sind durch Kommas voneinander getrennt. Nachdem das Formular abgespeichert ist werden diese Namen in ein Array eingelesen (Zeile 12). Das Array wird in einer Schleife ausgelesen und die entsprechenden Formulare neu geladen.

Navigation

FormStart

Aufruf aus	
Formular:	*Einzelmessung_System, frm_Berichte, frm_Boot, frm_Einstellungen, frm_Mitglied, frm_Schluessel, frm_Vorstand*

Benötigt	
Tabelle:	*tbl_Filter*
Ansicht:	*viw_ServVar*

```
001 SUB FormStart(oEvent AS OBJECT)
002     DIM arTitle()
003     oForm = oEvent.Source
004     arTitle = split(ThisComponent.Title, ThisComponent.UntitledPrefix)
005     stTitle = arTitle(1)
006     oConnection = oForm.activeConnection()
007     oSQL_Statement = oConnection.createStatement()
008     stSql = "UPDATE ""tbl_Filter"" SET ""Formular"" = '"+stTitle+"' WHERE ""ID"" =
           COALESCE ( ( SELECT ""ID"" FROM ""viw_ServVar"" ), '1' )"
009     oSQL_Statement.executeUpdate(stSql)
010     oForm.reload()
011 END SUB
```

Wird irgendein Formular über Mausklick als erstes Formular gestartet, so muss das Listenfeld für die Formularanzeige das korrekte Formular auch anzeigen. Hier wird aus dem Titel des Formularfensters der Name des Formulars geschlossen (Zeile 5). Anschließend wird dieser Text in die Tabelle "tbl_Filter" übertragen und das Formular, das ja beständig nur den einen Datensatz aus "tbl_Filter" anzeigt, neu geladen. Danach zeigt das Listenfeld den Namen des aktuellen Formulars an.

Navigation

Aufruf aus	
Formular:	*Einzelmessung_System, frm_Berichte, frm_Boot, frm_Einstellungen, frm_Mitglied, frm_Schluessel, frm_Vorstand*

```
001 SUB Navigation(oEvent AS OBJECT)
```

```
002     DIM oListField AS OBJECT
003     DIM stZiel AS STRING
004     DIM stStart AS STRING
005     DIM arStart()
006     oListField = oEvent.Source.Model
007     oListField.commit()
008     oForm = oListField.Parent
009     IF oForm.Parent.hasByName("frmMitglied") THEN
010        oForm1 = oForm.Parent.getByName("frmMitglied")
011        stID = oForm1.getString(oForm1.findColumn("ID"))
012        oForm.updateString(oForm.findColumn("mitID"), stID)
013     ELSEIF oForm.Parent.hasByName("frmVorstand") THEN
014        oForm1 = oForm.Parent.getByName("frmVorstand")
015        stID = oForm1.getString(oForm1.findColumn("mitID"))
016        oForm.updateString(oForm.findColumn("mitID"), stID)
017     ELSE
018     END IF
019     oForm.updateRow
020     arStart = split(ThisComponent.Title, ThisComponent.UntitledPrefix)
021     stStart = arStart(1)
022     stZiel = oListField.CurrentValue
023     oFormStart = ThisDatabaseDocument.FormDocuments.getByName(stStart)
024     oFormStart.Component.CurrentController.Frame.ContainerWindow.Visible = False
025     oFormZiel = ThisDatabaseDocument.FormDocuments.getByName(stZiel)
026     IF NOT IsNull(oFormZiel.Component) THEN
027        IF NOT IsNull(oFormZiel.Component.CurrentController) THEN
028           oFormZiel.Component.CurrentController.Frame.ContainerWindow.Visible = True
029        ELSE
030           oFormZiel.open
031        END IF
032     ELSE
033        oFormZiel.open
034     END IF
035     oFormZiel.Component.Drawpage.Forms.getByName("frmFilter").reload()
036 END SUB
```

Mit dieser Prozedur wird von einem Formular zum anderen über ein Listenfeld gewechselt. Dabei wird ein einmal geöffnetes Formular nicht wieder geschlossen sondern nur unsichtbar geschaltet, um einen Wechsel von Formular zu Formular reibungsloser ablaufen zu lassen.

In Zeile 7 wird der Inhalt des Listenfeldes zur Formularauswahl auf die Tabelle "tbl_Filter" übertragen. Das dazugehörige Formular hat die Bezeichnung "frmFilter". Von diesem Formular aus wird jetzt zuerst gesucht, ob ein Parallelformular "frmMitglied" existiert (Zeile 9). Unter diesen Umständen wird die Mitgliedsnummer aus dem Formular in die "tbl_Filter" übertragen. Bei einem Formular mit der Bezeichnung "frmVorstand" wird entsprechend verfahren. So wird die Mitgliedsnummer von einem Formular zum nächsten mitgenommen.

Danach wird das Formular, das auf der "tblFilter" als Datenbasis beruht, abgespeichert (Zeile 19).

Der Name des Formulars, aus dem die Prozedur aufgerufen wird, wird aus dem Titel des Fensters ausgelesen und in der globalen Variablen **oFormStart** gespeichert. Der Name des Zielformulars wird aus dem Listenfeld ausgelesen, das die Prozedur ausgelöst hat. Wenn dieses Formular bereits geöffnet ist, dann ist **oFormZiel.Component.CurrentController** nicht leer (Zeile 27). In dem Fall muss dieses Formular nur sichtbar gemacht werden. Andernfalls wird das Formular geöffnet (Zeile 30, 33). Zum Schluss wird das Formular noch einmal neu geladen, damit eventuell in anderen Formularen gemachte Änderungen auch übernommen werden.

Speichern

ListboxContentChange

Aufruf aus
Formular: *frm_Mitglied*

Benötigt	
Makro:	*ShowTableControl*

```
001  SUB ListboxContentChange(oEvent AS OBJECT)
002     oForm = oEvent.Source
003     stListbox = oForm.getByName("hidListbox").HiddenValue
004     oListbox = oFormStart.getByName(stListbox)
005     oListbox.refresh()
006  END SUB
```

Wurde in den einzublendenden Tabellenkontrollfeldern ein Eintrag geändert, so sollte der Eintrag auch in den entsprechenden Listenfeldern anschließend zur Verfügung stehen. Innerhalb des Formulars, in dem die Tabellenkontrollfelder liegen, liegt deswegen auch ein verstecktes Feld «hidListbox», aus dem die Bezeichnung der Listbox gelesen wird (Zeile 3). Das entsprechende Formular, in dem diese Listbox liegt, wurde durch die Prozedur *ShowTableControl* in einer globalen Variablen «oFormStart» hinterlegt.

NewGroup

Aufruf aus
Formular: *frm_Mitglied*

```
001  SUB NewGroup(oEvent AS OBJECT)
002     DIM loRow AS LONG
003     oField = oEvent.Source.Model
004     oField.Commit()
005     oForm = oField.Parent
006     IF oForm.IsNew THEN
007        oForm.insertRow()
008     ELSE
009        oForm.updateRow()
010     END IF
011     IF oField.Tag = "parent" THEN
012        oForm = oForm.parent
013     END IF
014     loRow = oForm.getRow()
015     oForm.reload()
016     oForm.absolute(loRow)
017  END SUB
```

In dem «frm_Mitglied» sind zwei Listenfelder enthalten, mit denen eine Gruppenzuweisung verbunden ist. In dem einen wird eine Adresse ausgewählt und davon abhängig soll dann ggf. eine Kontoverbindung erscheinen. Im zweiten Feld wird bei unterschiedlichen Adressen ggf. eine Kontoverbindung gewählt.

Beide Listenfelder sollen den entsprechenden Datensatz zuerst sichern und dann das Formular neu laden. Da ein Listenfeld in einem Unterformular liegt ist in diesem in den Zusatzinformationen "parent" angegeben (Zeile 11). Beim Auslösen über dieses Feld wird das Laden im darüber liegenden Formular veranlasst, so dass das bis dahin noch sichtbare weitere Formular für die Kontoverbindung unsichtbar geschaltet wird.

ShowTableControl

Aufruf aus
Formular: *frm_Mitglied*
Makros: *Navigation, ListboxContentChange* (beide: globale Variable oFormStart)

```
001  GLOBAL oFormStart AS OBJECT
```

```
001  SUB ShowTableControl(oEvent AS OBJECT)
002     DIM stForm AS STRING
003     oField = oEvent.Source.Model
004     stForm = oField.Tag
005     IF stForm <> "" THEN
006        oForm = ThisComponent.Drawpage.Forms.getByName(stForm)
007        oFormStart = oField.Parent
008     ELSE
009        oForm = oField.Parent
010     END IF
011     FOR k = 0 TO oForm.Count - 1
012        oField = oForm.getByIndex(k)
013        IF oField.ServiceName <> "stardiv.one.form.component.Form" AND
              oField.ServiceName <> "stardiv.one.form.component.Hidden" THEN
014           IF stForm <> "" THEN
015              oField.EnableVisible = True
016           ELSE
017              oField.EnableVisible = False
018           END IF
019        END IF
020     NEXT
021  END SUB
```

Im Formulardokument «frm_Mitglied» befinden sich zwei Tabellenkontrollfelder (für Banken und für Orte), die beim Öffnen des Formulars nicht sichtbar sind. Sie sollen nur bei Bedarf eingeblendet werden und eine Datenänderung ermöglichen. Die Prozedur schaltet dabei abhängig von dem auslösenden Element entweder das jeweiligen Tabellenkontrollfeld mit der Schaltfläche «Schließen» sichtbar (**`stForm <> ""`**) oder unsichtbar. Die Schaltfläche zum Schließen hat dabei keinen Eintrag in den Zusatzinformationen **`oField.Tag`** (Zeile 4), so dass sie eben lediglich das Tabellenkontrollfeld und sich selbst unsichtbar macht.

Um immer zu wissen, welches Formular denn nun sichtbar gemacht wurde wird der entsprechende Eintrag in der globalen Variable oFormStart zwischengespeichert. Diese Variable wird auch in den Prozeduren *Navigation* und *ListboxContentChange* genutzt.

SubscriptionChanged

Aufruf aus
Formular: *Einzelmessung_System*

```
001  SUB SubscriptionChanged(oEvent AS OBJECT)
002     oForm = oEvent.Source.Parent.getByName("frmBeitragsuebersicht")
003     oForm.reload()
004  END SUB
```

Diese Prozedur dient nur dazu, nach einer Datenänderung im Formulardokument «frm_Beitrag» das Formular «frmBeitragsuebersicht» neu zu laden.

SubscriptionSave

Aufruf aus
Formular: *Einzelmessung_System*

Benötigt
Tabelle: *tbl_Gruppe_Beitrag*
Ansicht: *viw_Gruppe, viw_Beitrag_Gruppe_kalkuliert*

```
001 SUB SubscriptionSave(oEvent AS OBJECT)
002    oForm = oEvent.Source.Model.Parent
003    oForm.updateRow()
004    oConnection = oForm.ActiveConnection()
005    oSQL_Statement = oConnection.createStatement()
006    oSQL_Statement1 = oConnection.createStatement()
007    stSql = "SELECT ""ID"", ""Gruppe"" FROM ""viw_Gruppe"" WHERE
          ""Alter_Beitrag"" = 18"
008    oResult = oSQL_Statement.executeQuery(stSql)
009    WHILE oResult.next
010       stID = oResult.getString(1)
011       stgruID = SelectSQL(oResult.getString(2))
012       FOR i = LBound(arSubscription) TO UBound(arSubscription)
013          IF arSubscription(i) = stID THEN
014             stSql = "DELETE FROM ""tbl_Gruppe_Beitrag"" WHERE
                   ""gruID"" "+stgruID+" "
015             oSQL_Statement1.executeUpdate(stSql)
016             stSql = "INSERT INTO ""tbl_Gruppe_Beitrag"" (""mitID"", ""beiID"",
                   ""Anzahl"") (SELECT ""Gruppe"", ""beiID"", ""Anzahl"" FROM
                   ""viw_Beitrag_Gruppe_kalkuliert"" WHERE ""Gruppe"" "+stgruID+")"
017             oSQL_Statement1.executeUpdate(stSql)
018             EXIT FOR
019          END IF
020       NEXT
021    WEND
022    oForm.Parent.getByName("frmMitglied").getByName("frmGruppe").reload()
023 END SUB
```

Vor dem Update eines Datensatzes für die Beitragsberechnung wird zuerst nachgeschaut, ob eine vorher 17-jährige Person jetzt für den nächsten Beitrag 18 Jahre alt geworden ist. Diese Information ist durch die Prozedur *Preferences* beim Start der Datenbankdatei in das Array **arSubscription** geschrieben worden.

In Zeile 7 wird die Abfrage formuliert, mit der alle Gruppen ausgelesen werden, bei denen eine Person aktuell 18 Jahre alt ist. In der Schleife beginnend ab Zeile 13 wird dann nachgesehen, ob diese Gruppen-ID gleich einem Wert aus dem Array **arSubscription** ist, also vorher eine Person hatte, die eben gerade noch 17 Jahre alt war. Nur bei diesen Gruppen wird die Neuberechnung des Beitrages angestoßen. Dies erfasst dann auf jeden Fall alle Gruppen, bei denen eventuell ein vorher 17-jähriges Mitglied jetzt 18 Jahre alt geworden ist.

Wartung

Im Modul «Wartung» befinden sich nur Prozeduren, die keine weitere Verbindung zu Formularen haben. Sie werden direkt über **Extras → Makros → Makro ausführen** gestartet.

Tabellenindex_runter

```
001 Sub Tabellenindex_runter(tablename AS STRING)
002    DIM inAnzahl AS INTEGER
003    DIM inSequence_Value AS INTEGER
004    oDatenquelle = ThisDatabaseDocument.CurrentController
005    IF NOT (oDatenquelle.isConnected()) THEN oDatenquelle.connect()
006    oVerbindung = oDatenquelle.ActiveConnection()
007    oSQL_Anweisung = oVerbindung.createStatement()
008    stSql = "SELECT MAX(""ID"") FROM """+tablename+""""
009    oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
010    IF NOT ISNULL(oAbfrageergebnis) THEN
011       WHILE oAbfrageergebnis.next
012          inAnzahl = oAbfrageergebnis.getInt(1)
```

```
013         WEND
014         IF inAnzahl = "" THEN
015            inAnzahl = -1
016         END IF
017         inSequence_Value = inAnzahl + 1
018         oSQL_Anweisung1 = oVerbindung.createStatement()
019         oSQL_Anweisung1.executeQuery("ALTER TABLE """+tablename+""" ALTER COLUMN
               ""ID"" RESTART WITH "+inSequence_Value+"")
020      END IF
021   End Sub
```

Mit dieser Prozedur wird der automatisch hoch geschriebene Primärschlüssel mit der vorgegebenen Bezeichnung "ID" auf den niedrigst möglichen Wert eingestellt.

Der Code für den höchsten in "ID" eingetragene Wert wird in Zeile 8 erstellt. Die Abfrage wird anschließend gestartet und der Rückgabewert in einer Variablen oAbfrageergebnis gespeichert.

In Zeile 12 wird das Ergebnis der Abfrage ausgelesen. Ein weiterer Datensatz existiert nicht, so dass die WHILE-Schleife auch entfallen könnte.

Falls der höchste Wert gar kein Wert ist, also die Tabelle leer ist wird der höchste Wert als -1 angenommen. Hier könnte in Zeile 15 auch gegebenenfalls ein anderer Wert eingetragen werden. Bei der internen HSQLDB beginnt die Zählung aber standardmäßig bei 0. Das wurde hier so übernommen.

Der höchste Wert wird um 1 erhöht und anschließend direkt in die Tabellendefinition übertragen.

Tabellenstart

Benötigt
Tabelle: *tbl_Mitglied, tbl_Ort, tbl_Bank*

```
001   Sub Tabellenstart
002      Tabellenindex_runter("tbl_Mitglied")
003      Tabellenindex_runter("tbl_Ort")
004      Tabellenindex_runter("tbl_Bank")
005   End Sub
```

In diese Prozedur werden nacheinander alle Tabellen eingetragen, bei denen der Wert für den automatisch erstellten Primärschlüssel so weit wie möglich herunter gesetzt werden soll. Diese Prozedur kann beliebig um entsprechende Tabellen wie z. B. *tbl_SchluesselAnzahl* erweitert werden.

Tabellenbereinigung

```
001   SUB Tabellenbereinigung(NameBezugsTabelle AS STRING, NameBezugsID AS STRING,
         NameLoeschTabelle AS STRING)
002      oDatenquelle = ThisDatabaseDocument.CurrentController
003      If NOT (oDatenquelle.isConnected()) Then oDatenquelle.connect()
004      oVerbindung = oDatenquelle.ActiveConnection()
005      oSQL_Anweisung = oVerbindung.createStatement()
006      oSQL_Anweisung.executeQuery("DELETE FROM """+NameLoeschTabelle+""" WHERE
            (""ID"" NOT IN (SELECT """+NameBezugsID+""" FROM
            """+NameBezugsTabelle+"""))")
007   END SUB
```

Hiermit werden alle Datensätze entfernt, die in der Bezugstabelle keine Verwendung mehr finden. Dies geht nicht mittels **`ON DELETE CASCADE`**, was in dem Relationenentwurf eingestellt werden kann, wenn sich die Bezüge durch eine n:1-Beziehung ergeben. Es ließe sich so z.B. beim Löschen eines Ortes die Adresse löschen, nicht aber der Ort, falls keine entsprechende Adresse existiert.

In Zeile 6 werden die Datensätze der Zieltabelle gelöscht, für die in der Bezugstabelle keine Einträge mehr existieren.

Tabellenbereinigung_Ort

Benötigt
Tabelle: *tbl_Adresse, tbl_Ort*

```
001  SUB Tabellenbereinigung_Ort
002     Tabellenbereinigung("tbl_Adresse","ortID","tbl_Ort")
003     Tabellenindex_runter("tbl_Ort")
004  END SUB
```

Zuerst wird in der Tabelle *tbl_Adresse* überprüft, ob noch ein Eintag im Feld "ort_ID" existiert. Existiert kein Eintag mehr, sehr wohl aber ein entsprechender "Ort" in der Tabelle *tbl_Ort*, so wird dieser Ort gelöscht.

Anschließend wird der automatisch hoch zählende Primärschlüssel für die Tabelle *tbl_Ort* so weit wie möglich zurück gesetzt.

Tabellenbereinigung_Bank

Benötigt
Tabelle: *tbl_Konto, tbl_Bank*

```
001  SUB Tabellenbereinigung_Bank
002     Tabellenbereinigung("tbl_Konto","banID","tbl_Bank")
003     Tabellenindex_runter("tbl_Bank")
004  END SUB
```

Zuerst wird in der Tabelle *tbl_Konto* überprüft, ob noch ein Eintag im Feld "ban_ID" existiert. Existiert kein Eintag mehr, sehr wohl aber eine entsprechende "Bank" in der Tabelle *tbl_Bank*, so wird diese Bank gelöscht.

Anschließend wird der automatisch hoch zählende Primärschlüssel für die Tabelle *tbl_Bank* so weit wie möglich zurück gesetzt.

ViewsErstellen

Benötigt
Abfrage: *tbl_Gruppe_Beitrag*

```
001  SUB ViewsErstellen
002     DIM oDatasource AS OBJECT
003     DIM oConnection AS OBJECT
004     DIM oSQL_Statement AS OBJECT
005     DIM oQuery AS OBJECT
006     DIM stSql AS STRING
007     DIM stView AS STRING
008     DIM stQuery AS STRING
009     DIM arQueries()
010     DIM arViews()
011     oDatasource = thisDatabaseDocument.CurrentController
012     IF NOT (oDatasource.isConnected()) THEN oDatasource.connect()
013     oConnection = oDatasource.ActiveConnection()
014     arViews = array("viw_Reports", "viw_Beitrag_Gruppe_kalkuliert", "viw_Gruppe",
           "viw_Beitrag", "viw_Mitglieder_komplett",
           "viw_Filter_Mitglieder_komplett_AusEin", "viw_Statistik_Alter_Detail",
           "viw_Anschrift", "viw_Adresse_Num_Namen", "viw_Schluessel_Mitglied",
           "viw_Boote_Mitglied", "viw_Abteilungen_Mitglied", "viw_ServVar" )
015     FOR i = LBound(arViews()) TO UBound(arViews())
016        stSql = "DROP VIEW IF EXISTS """+arViews(i)+""""
017        oSQL_Statement = oConnection.createStatement()
018        oSQL_Statement.executeUpdate(stSql)
019     NEXT
```

```
020     oQuery = oDatasource.DataSource.getQueryDefinitions()
021     arQueries = array("v_qry_ServVar", "v_qry_Abteilungen_Mitglied",
           "v_qry_Boote_Mitglied", "v_qry_Schluessel_Mitglied",
           "v_qry_Adresse_Num_Namen", "v_qry_Anschrift",
           "v_qry_Statistik_Alter_Detail", "v_qry_Filter_Mitglieder_komplett_AusEin",
           "v_qry_Mitglieder_komplett", "v_qry_Beitrag", "v_qry_Gruppe",
           "v_qry_Beitrag_Gruppe_kalkuliert", "v_qry_Reports")
022     FOR i = LBound(arQueries()) TO UBound(arQueries())
023        stQuery = oQuery.getByName(arQueries(i)).Command
024        stView = arViews(UBound(arViews()) - i)
025        stSql = "CREATE VIEW """+stView+""" AS "+stQuery+""
026        oSQL_Statement = oConnection.createStatement()
027        oSQL_Statement.executeUpdate(stSql)
028     NEXT
029  END SUB
```

Von Zeile 11 bis Zeile 13 wird die Verbindung zur Datenbank eingelesen, gegebenenfalls neu erstellt.

Anschließend werden alle Ansichten in einem Array zusammengefasst. Die Reihenfolge der Ansichten ist so gewählt, dass sie nacheinander gelöscht werden können, ohne dass sich eine nachfolgende Ansicht noch auf die vorhergehende Ansicht bezieht. Die Löschung der Ansichten erfolgt in einer Schleife (Zeile 15 bis Zeile 19).

Zeile 20 ermöglicht den Zugriff auf den SQL-Code für die Abfragen der Datenbank. Die Abfragen für die Ansichten werden wie die Ansichten selbst in einem Array zusammengefasst. Nur ist die Reihenfolge genau anders herum. Zuerst sollen die Ansichten erstellt werden, auf die die anderen Ansichten aufbauen.

Auch die Erstellung der neuen Ansichten erfolgt in einer Schleife. Dabei wird das Array für die Ansichten in Zeile 24 in umgekehrter Reihenfolge (von hinten nach vorne) eingelesen. Die Ansichten werden durch den Code der entsprechenden Abfragen (Zeile 25) erstellt.