



LibreOffice
The Document Foundation

Base-Beispiele

Datenbanklösungen für
einzelne Problemstellungen



7.0

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

Copyright

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

Warennamen werden ohne Gewährleistung der freien Verwendbarkeit benutzt.

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

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

Mitwirkende/Autoren

Robert Großkopf

Titelblatt: Robert Großkopf

basierend auf dem Design von Bayu Rizaldhan Rayes und des 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 29.07.2020. Basierend auf der LibreOffice Version 7.0.

Inhaltsverzeichnis

Vorwort.....	11
Zeitmessung.....	12
Einführung.....	12
Tabellen.....	12
Start.....	12
Zeitspeicher.....	13
Abfragen.....	14
Abfrage_Start.....	14
Abfrage_Unterwegs.....	15
Abfrage_Ziel.....	15
Formulare.....	16
Einzelmessung_System.....	16
Einzelmessung_Zeit.....	17
Meldung_Liste_Teilnehmer.....	18
Meldung_Zeit_Teilnehmer.....	19
Nur_Start.....	20
Nur_Ziel.....	21
Start_Ziel_Platz.....	23
Start_Ziel_Gruppe.....	24
Berichte.....	25
Ergebnisliste.....	25
Ergebnisliste_Zeit.....	26
Urkunden.....	27
Makros.....	28
Starten und Stoppen der Zeit.....	29
Zeitmessung_Start_System.....	29
Zeitmessung_Ende_System.....	29
Zeitmessung_Start.....	30
Zeitmessung_Ende.....	31
Sekunden_2_SQLTime.....	32
Formulare mit Zeitmessung verknüpfen.....	33
Zeitmessung_Start_Einzel.....	33
Zeitmessung_Ende_Einzel.....	34
Zeitmessung_Start_Gruppe.....	34
Zeitmessung_Ende_Gruppe.....	34
Zeitmessung_Start_Ziel.....	35
Zeitmessung_Ende_Listenfeld.....	35
Aktuelle Standardwerte für Datum und Zeit.....	37
Einführung.....	37
Tabellen.....	37
Datum_Aenderdatum.....	37
Zeitstempel_Aenderstempel.....	38
Ansichten.....	39
Abfragen.....	40
DatumZeitvorgabe.....	40
Zeitdifferenzen.....	40
Zeitdifferenzen_Formularvorlage.....	41
Formulare.....	43
Defaultdatum_Subform_neue_Datensaetze.....	43
Defaultdatum_Zeit_Subform_neue_Datensaetze.....	44

Defaultdatum_Aenderung.....	45
Defaultdatum_Makro_SQL.....	45
Defaultdatum_Zeit_Makro_SQL.....	46
Defaultdatum_Makro.....	47
Defaultdatum_Zeit_Makro.....	47
Defaultdatum_Makro_Standarddatum.....	48
Defaultdatum_Makro_Standarddatum_verlegt.....	48
Defaultdatum_Makro_Standardzeitstempel.....	49
Defaultdatum_Zeitdifferenz_Makro.....	49
Berichte.....	50
Bericht_Zeitdifferenzen.....	50
Bericht_Zeitdifferenzen_Formularvorlage.....	52
Makros.....	53
Update.....	53
Datum_aktuell.....	53
Datum_Zeit_aktuell.....	54
Standarddatum.....	55
Standarddatum_verlegt.....	55
StandardZeitstempel.....	56
UpdateTimestamp.....	57
Dateien einbinden.....	58
Einführung.....	58
Tabellen.....	59
Dateien.....	59
Dateien_intern.....	59
Ansichten.....	60
Abfragen.....	61
Formulare.....	61
Pfadeingabe.....	61
Pfadeingabe_Tabellenkontrollfeld.....	62
Dateiaufnahme.....	63
Dateiaufnahme_Name.....	64
Berichte.....	64
Makros.....	66
Betrachten.....	66
DateiAuslesen.....	67
DateiAuslesen_mitName.....	68
DateiName.....	68
Serienbrief direkt.....	70
Einführung.....	70
Tabellen.....	70
Anschrift.....	70
Rechnung.....	71
Rechnungsinhalt.....	72
Verkauf.....	72
Waren.....	73
Abfragen.....	73
Adresseingabe.....	73
Etiketten.....	74
Rechnung_einzeilig.....	74
Rechnung_nicht_gedruckt.....	75
Rechnung_Textfelder.....	75
Formulare.....	77

Anschrift.....	77
Anschrift_Textfelder.....	79
Etiketten_Serienbrief_im_Formular.....	81
Rechnung.....	84
Rechnung_Textfelder.....	87
Rechnung_Textfelder_Uebertrag.....	89
Rechnung_Textfelder_Uebertrag_Seriendruck.....	91
Waren.....	92
Berichte.....	92
Makros.....	92
Rechnung.....	92
Rechnungsinhalt_zusammenstellen.....	93
Serienbrief.....	94
Output_MailMerge.....	94
Textfelder_Fuellen.....	95
Textfelder_Fuellen_ID.....	96
Tabellen_fuellen.....	97
Rechnungsinhalt_zusammenstellen_Tabelle.....	98
Tabellen_fuellen_Uebertrag.....	99
Rechnungsinhalt_zusammenstellen_Tabelle_Uebertrag.....	101
Rechnungsinhalt_zusammenstellen_Tabelle_Uebertrag_Direktdruck.....	101
PDF_Export.....	102
Direktdruck_Start.....	103
Terminübersicht Ferienhäuser.....	105
Einführung.....	105
Tabellen.....	105
Mieter.....	105
Haus.....	106
Reservierung.....	106
Filter.....	107
NrBis31.....	107
Ansichten.....	108
Ansicht_Datum.....	108
Ansicht_Datum_lfdNr.....	109
Abfragen.....	109
Belegung.....	110
Bericht.....	111
Formulare.....	112
Berichte.....	115
Makros.....	115
Startdatum.....	115
Enddatum.....	117
BuchungsAenderung.....	118
NeueBuchung.....	118
DatumsAenderung.....	119
Mailaufruf.....	121
Einführung.....	121
Tabellen.....	121
Kontakte.....	121
Mails.....	122
Mail_CC.....	122
Anhaenge.....	123
Ansichten.....	124

Abfragen.....	124
Formulare.....	124
Kontakte_Mail_Web.....	124
Mail_mit_Anhang.....	126
Berichte.....	127
Makros.....	127
Mauszeiger.....	127
FormSave.....	127
Website_Aufruf.....	128
Mail_Aufruf.....	129
Attachment.....	131
Mail_Attachment.....	132
Suchen und Filtern.....	135
Einführung.....	135
Tabellen.....	135
Filter.....	136
Kategorien.....	137
Medien.....	138
Medienart.....	138
Ort.....	139
rel_Medien_Verfasser.....	139
Verfasser.....	140
Ansichten.....	140
Ansicht_Suche.....	140
Ansicht_Filter_Start.....	141
Ansicht_Standardfilter.....	141
Standardfilter_Firebird.....	142
Abfragen.....	142
Filter.....	142
Filter_Form.....	142
Filter_Form_Subform.....	143
Filter_Form_Subform_3Filter.....	143
Filter_Ansicht.....	144
Filter_Form_Nullwerte.....	145
Filter_Leerfeld.....	145
Suche.....	145
Suche_Form.....	145
Suche_Form_Subform.....	146
Parametersuche_Medien.....	147
Filter und Suche kombiniert.....	148
Suche_Filter_Form_Subform_3Filter.....	148
Filter_Such_Ansicht.....	149
Listenfeldabfragen.....	150
Listenfeld_Kategorie_ohne_Eintrag.....	150
Listenfeld_Kategorie_bedingt.....	151
Listenfeld_Kategorie_bedingt_Datensaetze.....	151
Listenfeld_Kategorie_bedingt_Suche_Filter.....	152
Listenfeld_Kategorie_bedingt_Suche_Filter_Datensaetze.....	152
Listenfeld_Medienart_bedingt.....	153
Listenfeld_Medienart_bedingt_Datensaetze.....	153
Listenfeld_Medienart_bedingt_Suche_Filter.....	154
Listenfeld_Medienart_bedingt_Suche_Filter_Datensaetze.....	154
Listenfeld_Verfasser_bedingt.....	154
Listenfeld_Verfasser_bedingt_Datensaetze_langsam.....	155
Listenfeld_Verfasser_bedingt_Datensaetze_optimiert.....	155

Listenfeld_Verfasser_bedingt_Suche_Filter.....	156
Listenfeld_Verfasser_bedingt_Suche_Filter_Datensaetze.....	156
Formulare.....	157
Filter_ohne_Makros.....	157
Filter_Formular.....	157
Filter_Formular_Nullwerte.....	159
Filter_Formular_Subformular.....	159
Filter_Formular_Subformular_3Filter.....	160
Standardfilter_Formular_Subformular.....	161
Standardfilter_Formular_Subformular_Datensatzkorrektur.....	162
Filter_mit_Makros.....	162
Filter_Formular.....	163
Filter_Formular_direkt_Nullwerte.....	164
Filter_Formular_Listfeldbegrenzung.....	165
Filter_Formular_Subformular.....	166
Filter_Formular_Subformular_bedingende_Filter.....	167
Filter_Formular_Subformular_bedingende_Filter_allround.....	168
Filter_Formular_Subformular_bedingende_Filter_Datensaetze.....	168
Suche_ohne_Makros.....	169
Suche_Formular.....	169
Suche_Formular_Subformular.....	170
Parameter_Suche_Formular_Subformular.....	171
Suche_mit_Makros.....	171
Suche_Formular.....	172
Suche_Formular_Subformular.....	173
Suche_Filter_Formular_Subformular_ohne_Makros.....	174
Suche_Filter_hierarchisch_mit_Makros.....	175
Suche_Filter_hierarchisch_allround_Datensaetze_mit_Makros.....	176
Berichte.....	177
Makros.....	177
Filter.....	177
Filter_Formular_direkt.....	178
Filter_bedingt.....	179
Filter_bedingt_allround.....	179
Filter_Suche.....	181
Filter_Suche_hierarchisch.....	181
Filter_Suche_hierarchisch_Start.....	182
Filter_Suche_hierarchisch_allround.....	183
ListFilter.....	184
Haushaltsbuch.....	187
Einführung.....	187
Tabellen.....	187
Kasse.....	187
Adressat.....	188
Konto.....	189
Kategorie.....	189
rel_Kasse_Kategorie.....	190
Filter.....	191
Ansichten.....	192
Ansicht_Kasse_Kategorie.....	192
Ansicht_Kasse_mit_Umbuchungen.....	193
Ansicht_Filter.....	194
Ansicht_Kategorie_Diagramm.....	195
Ansicht_Konto_Diagramm.....	196
Ansicht_Bericht_Kategorie.....	196

Abfragen.....	197
Kategorieverlauf.....	197
Kasse_Saldo_Konto_Kategorie.....	198
Kasse_Saldo_Konto_Kategorie_Umbuch.....	198
Kontoverlauf.....	199
Saldo_Kategorie.....	200
Saldo_Kategorie_ungesplittet.....	201
Saldo_Konto.....	202
Formulare.....	203
Formulare ohne Makros.....	203
Kasse_oM.....	203
Konto_oM.....	203
Konto_Salden_oM.....	204
Konto_Salden_separat_oM.....	205
Formulare mit einfachem Aktualisierungsmakro.....	206
Konto_Salden_komplett_eM.....	206
Konto_Salden_komplett_Umbuchung_eM.....	207
Formulare mit mehrfachem Makroeinsatz.....	208
Kasse.....	208
Kasse_Umbuchung.....	209
Konto.....	210
Konto_Salden.....	211
Konto_Salden_komplett.....	212
Konto_Salden_komplett_Umbuchung.....	213
Buchung_Umbuchung_Salden.....	214
Kategorie_Konto_Auswertung.....	217
Berichte.....	218
Makros.....	221
Aktualisieren_einfach.....	221
Aktualisieren.....	221
ZeileZwischenspeichern.....	222
SplitRest.....	223
periodische_Buchung.....	225
Adressat_neu.....	227
Monat_aktuell.....	229
Formular_starten.....	229
Aktuelles_Buchungsdatum.....	229
Kontofilter_Formstart.....	230
Kontofilter_Feldstart.....	231
Kontoliste_filtern.....	231
ChangeData.....	232
Termine.....	235
Einführung.....	235
Tabellen.....	236
Tabellen zur Dateneingabe über Formulare oder direkt.....	236
Termine.....	236
Terminart.....	237
Zielgruppe.....	238
Ferien.....	238
Filtertabelle.....	239
Durch Makros gefüllt Tabellen.....	240
tmp_Halbjahreskalender.....	240
tmp_Gruppe_multi.....	241
tmp_Kalender.....	241
tmp_Monat.....	242
tmp_Terminuebersicht.....	242

tmp_Wochenkalender.....	242
tmp_Wochenkalender_half.....	243
Ansichten.....	244
Ansicht_Gruppe_multi.....	244
Ansicht_Kalender_Export.....	245
Ansicht_Kalender_ohne_Ferien_Export.....	247
Ansicht_Termine_Monat_Jahre.....	248
Ansicht_Wochen_einzeln.....	248
Ansicht_Wochentage.....	249
Abfragen.....	250
Liste_Gruppe_Multi.....	250
Termine_Gruppe_multi.....	251
Terminuebersicht.....	251
Wochenkalender_Mo_Fr.....	252
Formulare.....	252
Terminart_Termin.....	252
Termine_Kalenderübersicht.....	254
Termine_Monat_Terminart.....	255
Termine_Tabelleneingabe.....	256
Berichte.....	257
Halbjahreskalender_A4_quer.....	258
Liste_Gruppe_multi.....	258
Liste_Gruppe_multi_1HJ.....	259
Liste_Gruppe_multi_2HJ.....	259
Monatskalender_Mo_Fr.....	260
Monatskalender_Mo_So.....	261
Wochenkalender_half.....	261
Wochentage_Mensaabo.....	262
Zeitplan.....	263
Makros.....	263
AA_Variablen.....	263
Datenbankstart.....	263
OfficeVersion.....	264
Datumswert.....	264
Listenfeld_Text.....	265
ID_Ermittlung.....	266
Abfragen.....	266
Kalender.....	266
Monatskalender.....	269
Wochenkalender_half.....	272
Halbjahreskalender.....	273
Zeitplan.....	276
GruppeMultiUebersicht.....	279
Monat.....	282
Monatsansicht.....	283
Date_2_SQLDate.....	284
Date_2_Datum.....	284
SQLDate_2_Datum.....	285
Date_2_Monat.....	285
Datum_Ostersonntag.....	286
FeiertageArray.....	286
BerichtZeilenhoeheAuto.....	287
Bericht_Start.....	288
Backup.....	291
Datenbankbackup.....	291
Backup_sofort.....	292

Eingabe.....	292
DatenSpeichern.....	292
DatenAktualisieren_Termine.....	293
DatenAktualisieren_Terminort_Termine.....	293
LetzterDatensatz.....	294
LetzterDatensatz_Termine_Kalender.....	294
Anfangswert_Datum.....	295
Monatsansicht_aktualisieren.....	295
Monat_aktualisieren.....	297
Terminaten_uebertragen.....	297
GruppeMultiSpeichern.....	299
DialogStart.....	299
DialogEnde.....	300
Export.....	300
Ical_Export.....	300
Export_Start.....	302
Import.....	303
Ical_Import.....	303
Ical_Timestamp.....	306
Import_Start.....	306
LetzterSonntag.....	307
Filter.....	308
Wartung.....	308
Tabellenindex_runter.....	308
Tabellenstart.....	309
Dialog Datumseingabe.....	310

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: <i>Einzelmessung_System, Einzelmessung_Zeit, Meldung_Liste_Teilnehmer, Meldung_Zeit_Teilnehmer, Nur_Start, Nur_Ziel, Start_Ziel_Platz</i>
Makro: <i>Zeitmessung_Ende_System, Zeitmessung_Ende</i>
Abfrage: <i>Abfrage_Unterwegs, Abfrage_Ziel</i>
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

1 Beispieldatenbank Beispiel_Zeitmessung.odt

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

Zeitspeicher

Datenziel	
Formular:	<i>Einzelmessung_System, Einzelmessung_Zeit, Meldung_Liste_Teilnehmer, Meldung_Zeit_Teilnehmer, Nur_Ziel, Start_Ziel_Platz</i>
Makro:	<i>Zeitmessung_Start_System, Zeitmessung_Ende_System, Zeitmessung_Start, Zeitmessung_Ende</i>
Abfrage:	<i>Abfrage_Start, Abfrage_Unterwegs, Abfrage_Ziel</i>
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

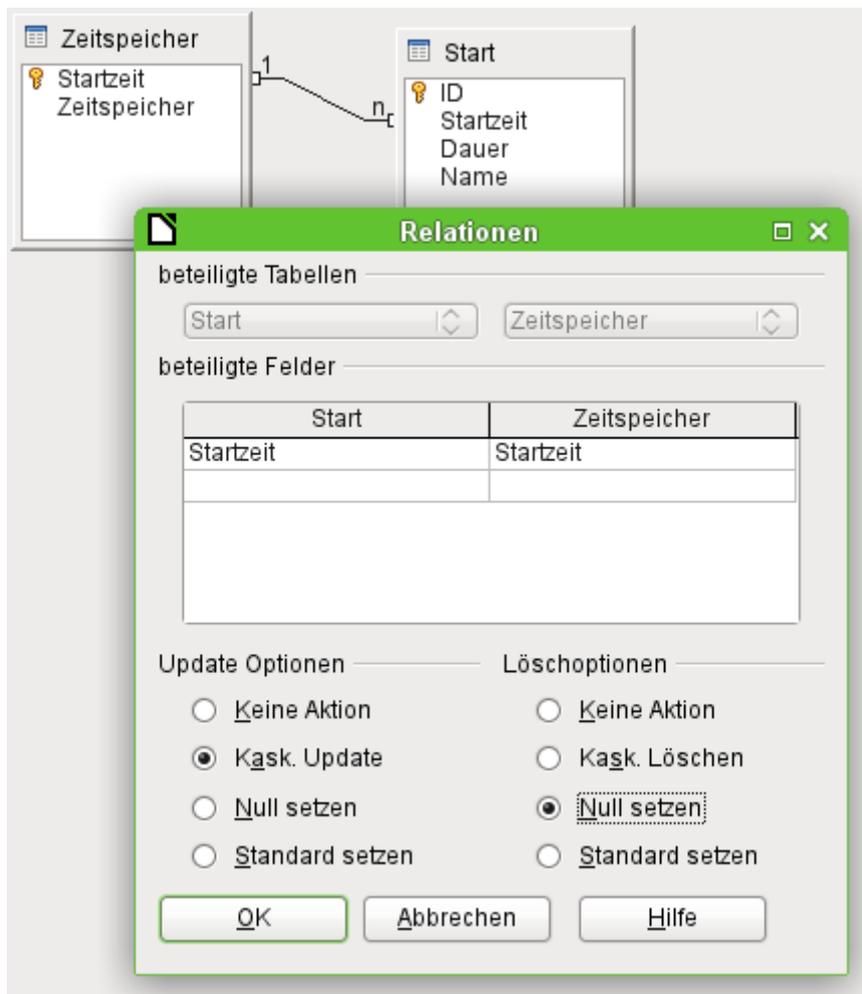
Die Datum/Zeit-Felder in der HSQLDB können, sofern sie durch die GUI erstellt wurden, keine Bruchteile einer Sekunde darstellen. Sie müssen erst mit dem Befehl

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

```
1 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

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

FIREBIRD

```
1 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*

```
1 SELECT "Start".*
2 FROM "Start", "Zeitspeicher"
3 WHERE "Start"."Startzeit" = "Zeitspeicher"."Startzeit"
4 AND "Start"."Zeit" IS NULL
5 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:

```
1 SELECT "a".*,
2 ( SELECT "Timestamp" FROM "Zeitspeicher" WHERE "Startzeit" = "a"."Startzeit" ) AS
   "Zeitspeicher",
3 ( SELECT COUNT( "ID" ) FROM "Start" WHERE "Zeit" <= "a"."Zeit" AND "Startzeit" =
   "a"."Startzeit" ) AS "Platz"
4 FROM "Start" AS "a"
5 WHERE ( SELECT "Timestamp" FROM "Zeitspeicher" WHERE "Startzeit" = "a"."Startzeit" )
   IS NOT NULL
```

Änderung für **FIREBIRD**:

```
1 SELECT "a".*,
2 ( SELECT "Zeitspeicher" FROM "Zeitspeicher" WHERE "Startzeit" = "a"."Startzeit" ) AS
   "Zeitspeicher",
```

```

3 ( SELECT COUNT( "ID" ) FROM "Start" WHERE "Zeit" <= "a"."Zeit" AND "Startzeit" =
  "a"."Startzeit" ) AS "Platz",
4 CAST( "a"."Zeit" AS TIMESTAMP ) AS "RBZeit
5 FROM "Start" AS "a"
6 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

Vorgaben aus der Tabelle

ID		<input type="text" value="1"/>
Startzeit	1	<input type="text" value="01.05.20 10:00"/>
Name		<input type="text" value="Karl Müller"/>

Dauer

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, Hundertstel Sekunden**.

Vorgaben aus der Tabelle

ID	1
Startzeit	01.05.20 10:00
Name	Karl Müller

Zeit

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:

```
1 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, das 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

Teilnehmer

	StartNr	Name	Startzeit
▶	1	Karl Müller	01.05.20 10:00
	2	Uwe Maier	01.05.20 10:00
	3	Nils Weglauf	01.05.20 10:00
	4	Karl Cash	01.05.20 11:00
	5	Egon Werkel	01.05.20 11:00
	6	Xavers Franz	01.05.20 10:00
	7	Kai Knieperich	01.05.20 11:00
☼	<AutoFeld>		

Datensatz 1 von 7

- 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

Startzeit

01.05.20 10:00	1
----------------	----------

Teilnehmer zur ausgewählten Startzeit

	StartNr	Name
▶	1	Karl Müller
	2	Uwe Maier
	3	Nils Weglauf
	6	Xavers Franz
☀	<AutoFeld>	

Datensatz 1 von 4

1 Formular (Tabelle: <i>Zeitspeicher</i>)	1.1 Teilnehmer (Tabelle: <i>Start</i>)
---	--

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.

Startzeit (Datum)

Startzeit (Zeit) **1**

Datensatz von 1

ID	Name
4	Karl Cash
5	Egon Werkel
7	Kai Knieperich
↳Feld>	

Datensatz von 3

1.1

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.

Zieleinlauf

Startnummer 1 Name
 Zeit

Stop: Startnummer anklicken

ID	Startzeit	Zeit	Name	Platz
3	01.05.20 10:00	00:00:19,52	Nils Weglauf	1
2	01.05.20 10:00	00:00:31,86	Uwe Maier	2

1.1

Datensatz 1 von 2

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.

```

1 SELECT "Start"."ID"
2   FROM "Start", "Zeitspeicher"
3   WHERE "Start"."Startzeit" = "Zeitspeicher"."Startzeit"
4     AND "Zeitspeicher"."Zeitspeicher" IS NOT NULL
5     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 darunterliegende 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:

```
6 ("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:

```
7 ( "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:

```
8 "Platz" ASC
```

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

Start_Ziel_Platz

Startzeit (Datum) **1**

Startzeit (Zeit) **1**

Datensatz von 1

ID	Name
4	Karl Cash
5	Egon Werkel
7	Kai Knieperich
»Feld>	

1.1

Datensatz von 3

Zieleinlauf

Startnummer **2** Name

Zeit

ID	Startzeit	Zeit	Name	Platz
3	01.05.20 10:00	00:00:19,52	Nils Weglauf	1
2	01.05.20 10:00	00:14:11,55	Uwe Maier	2

2.1

Datensatz von 2

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.Fehler: Verweis nicht gefunden
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

Startzeit (Datum) 1

Startzeit (Zeit) Start

Datensatz von 1

Datensatz markieren, dann auf «Stop» drücken

ID	Name
1	Karl Müller
6	Xavers Franz

Stop

2

Datensatz von 2 (1)

ID	Startzeit	Name	Zeit	Platz
2	10:00	Uwe Maier	00:00:31,86	2
3	10:00	Nils Weglauf	00:00:19,52	1

3

Datensatz von 2

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

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

Formulareigenschaften → Daten → Sortierung

```
10 "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:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	
^ Seite...	Ergebnisliste																	
^ Startz...	Startzeit	=Startzeit																
	Platz	Name					Dauer											
^ Daue...																		
^ Detail	=Platz	=Name					=Dauer											
^ Seite...	="Seite " & PageNu																	

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 TIMESTAMP-Feld in der Abfrage erfolgen. Das Feld für FIREBIRD heißt deswegen gesondert "RBZeit".

Urkunden

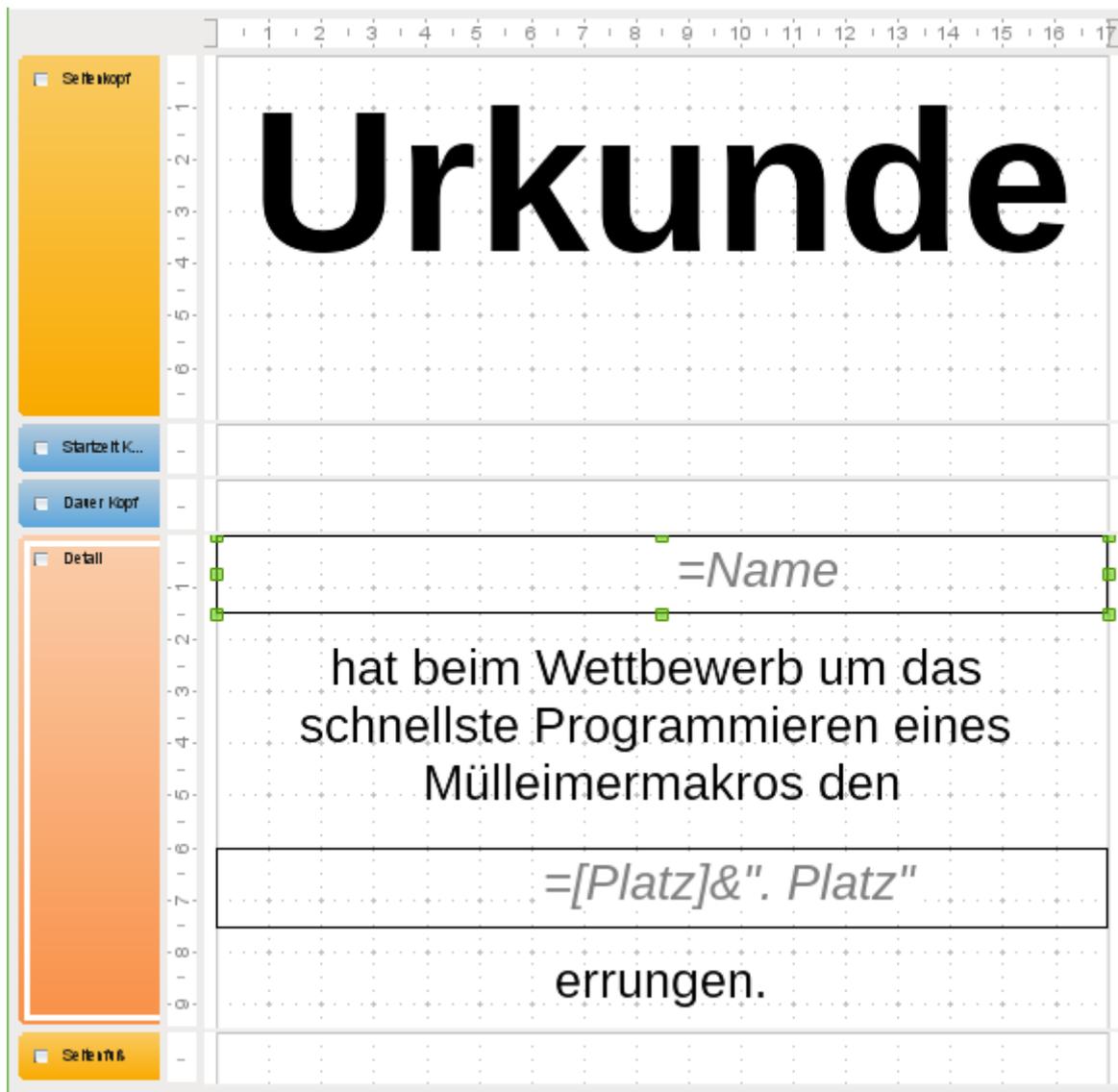
<h1>Urkunde</h1> <p>Karl Müller</p> <p>hat beim Wettbewerb um das schnellste Programmieren eines Mülleimermakros den</p> <p>1. Platz</p> <p>errungen.</p>	<h1>Urkunde</h1> <p>Xavers Franz</p> <p>hat beim Wettbewerb um das schnellste Programmieren eines Mülleimermakros den</p> <p>2. Platz</p> <p>errungen.</p>
---	--

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: <i>Einzelmessung_System</i>	
Benötigt	
Tabelle: <i>Zeitspeicher.Zeitspeicher</i>	nur HSQLDB
<pre>1 SUB Zeitmessung_Start_System(oEvent AS OBJECT) 2 DIM oForm AS OBJECT 3 DIM oFeld AS OBJECT 4 DIM oConnection AS OBJECT 5 DIM oSQL_Statement AS OBJECT 6 DIM stStartzeit AS STRING 7 DIM stSql AS STRING 8 DIM loZeit AS LONG 9 oForm = oEvent.Source.Model.Parent 10 stStartzeit = Left(oForm.getString(oForm.findColumn("Startzeit")),19) 11 loZeit = getSystemTicks() ' Gibt die Systemzeit in Millisekunden an 12 oConnection = oForm.activeConnection() 13 oSQL_Statement = oConnection.createStatement() 14 stSql = "UPDATE ""Zeitspeicher"" SET ""Zeitspeicher"" = '"+loZeit+"' WHERE ""Startzeit"" = '"+stStartzeit+"' " 15 oSQL_Statement.executeUpdate(stSql) 16 END SUB</pre>	

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_End_System

Aufruf aus	
Formular: <i>Einzelmessung_System</i>	
Benötigt	
Tabelle: <i>Zeitspeicher.Zeitspeicher, Start.Dauer</i>	nur HSQLDB
<pre>1 SUB Zeitmessung_End_System(oEvent AS OBJECT) 2 DIM oForm AS OBJECT 3 DIM oConnection AS OBJECT 4 DIM oSQL_Statement AS OBJECT 5 DIM inRow AS INTEGER 6 DIM stID AS STRING</pre>	

```

7   DIM stSql AS STRING
8   DIM loZeit AS LONG
9   oForm = oEvent.Source.Model.Parent
10  stStartzeit = Left(oForm.getString(oForm.findColumn("Startzeit")),19)
11  stID = oForm.getString(oForm.findColumn("ID"))
12  loZeit = getSystemTicks() ' Gibt die Systemzeit in Millisekunden an
13  oConnection = oForm.activeConnection()
14  oSQL_Statement = oConnection.createStatement()
15  stSql = "SELECT "+loZeit+" - ""Zeitspeicher""FROM ""Zeitspeicher"" WHERE
        ""Startzeit"" = '"+stStartzeit+'"'
16  oResult = oSQL_Statement.executeQuery(stSql)
17  WHILE oResult.next
18      loZeit = oResult.getLong(1)
19  WEND
20  stSql = "UPDATE ""Start"" SET ""Dauer"" = '"+loZeit+' WHERE ""ID"" = '"+stID+'"'
21  oSQL_Statement.executeUpdate(stSql)
22  inRow = oForm.getRow()
23  oForm.reload()
24  oForm.absolute(inRow)
25  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: <i>Nur_Start</i>
Makro: <i>Zeitmessung_Start_Einzel, Zeitmessung_Start_Gruppe, Fehler: Verweis nicht gefunden</i>

Benötigt
Tabelle: <i>Zeitspeicher.Timestamp</i>

```

1  SUB Zeitmessung_Start(oEvent AS OBJECT)
2  DIM oForm AS OBJECT
3  DIM oConnection AS OBJECT
4  DIM oSQL_Statement AS OBJECT
5  DIM stStartzeit AS STRING
6  DIM stSql AS STRING
7  oForm = oEvent.Source.Model.Parent
8  IF oForm.RowCount > 0 THEN
9      stStartzeit = Left(oForm.getString(oForm.findColumn("Startzeit")),19)
10     oConnection = oForm.activeConnection()
11     oSQL_Statement = oConnection.createStatement()
12     stSql = "UPDATE ""Zeitspeicher"" SET ""Timestamp"" = NOW() WHERE ""Startzeit""
            = '"+stStartzeit+'"'
13     oSQL_Statement.executeUpdate(stSql)

```

```

14         oForm.reload()
15     END IF
16 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*

```

1 SUB Zeitmessung_Ende(oEvent AS OBJECT)
2     DIM oForm AS OBJECT
3     DIM oConnection AS OBJECT
4     DIM oSQL_Statement AS OBJECT
5     DIM oResult AS OBJECT
6     DIM stID AS STRING
7     DIM stStartzeit AS STRING
8     DIM stSql AS STRING
9     DIM stZeit AS STRING
10    DIM loZeit AS LONG
11    oForm = oEvent.Source.Model.Parent
12    IF oForm.RowCount > 0 THEN
13        stID = oForm.getString(oForm.findColumn("ID"))
14        stStartzeit = Left(oForm.getString(oForm.findColumn("Startzeit")),19)
15        oConnection = oForm.activeConnection()
16        oSQL_Statement = oConnection.createStatement()
17        stSql = "SELECT DATEDIFF('ms', ""Timestamp"", NOW()) FROM ""Zeitspeicher""
18                WHERE ""Startzeit"" = '"+stStartzeit+"'
19        oResult = oSQL_Statement.executeQuery(stSql)
20        WHILE oResult.next
21            loZeit = oResult.getLong(1) ' Zeitangabe in Millisekunden
22        WEND
23        stZeit = Sekunden_2_SQLTime(loZeit)
24        stSql = "UPDATE ""Start"" SET ""Zeit"" = '"+stZeit+"' WHERE ""ID"" =
25                '"+stID+"'
26        oSQL_Statement.executeUpdate(stSql)
27        oForm.reload()
28    END IF
29 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:

```

17     stSql = "SELECT TIME 'NOW' - ""Zeitspeicher"" FROM ""Zeitspeicher"" WHERE
        ""Startzeit"" = '"+stStartzeit+'"'
18     oResult = oSQL_Statement.executeQuery(stSql)
19     WHILE oResult.next
20         doZeit = oResult.getDouble(1)
21     WEND
22     stZeit = Sekunden_2_SQLTime(doZeit)
23     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*

```

1  FUNCTION Sekunden_2_SQLTime(loHSek AS LONG) AS STRING
2      DIM stSek AS STRING
3      DIM doHSek AS DOUBLE
4      DIM inSek AS INTEGER
5      DIM inMin AS INTEGER
6      DIM inStd AS INTEGER
7      DIM stHSek AS STRING
8      doSek = loHSek/1000
9      stSek = CStr(doSek)
10     stHSek = Mid(stSek, InStr(stSek, ",")+1)
11     doHSek = doSek - Fix(doSek)
12     inSek = (doSek - doHSek) Mod 60
13     inMin = ( (doSek - inSek - doHSek) / 60 ) Mod 60
14     inStd = ( doSek - inSek - doHSek - 60 * inMin) / 3600
15     IF inStd < 0 THEN
16         inStd = inStd + 24
17     END IF
18     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
19 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:

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

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 '0' 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*

```
1 SUB Zeitmessung_Start_Einzel(oEvent AS OBJECT)
2   DIM oForm AS OBJECT
3   DIM inRow AS INTEGER
4   oForm = oEvent.Source.Model.Parent
5   inRow = oForm.getRow()
6   Zeitmessung_Start(oEvent)
```

```

7   oForm.absolute(inRow)
8   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: <i>Einzelmessung_Zeit</i>

Benötigt
Makro: <i>Zeitmessung_Ende</i>

```

1 SUB Zeitmessung_Ende_Einzel(oEvent AS OBJECT)
2   DIM oForm AS OBJECT
3   DIM inRow AS INTEGER
4   oForm = oEvent.Source.Model.Parent
5   inRow = oForm.getRow()
6   Zeitmessung_Ende(oEvent)
7   oForm.absolute(inRow)
8 END SUB

```

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

Zeitmessung_Start_Gruppe

Aufruf aus
Formular: <i>Start_Ziel_Gruppe</i>

Benötigt
Makro: <i>Zeitmessung_Start</i>

```

1 SUB Zeitmessung_Start_Gruppe(oEvent AS OBJECT)
2   DIM oForm AS OBJECT
3   Zeitmessung_Start(oEvent)
4   oForm = oEvent.Source.Model.Parent
5   oFormZiel = oForm.Parent.getByname("Ziel")
6   oFormZiel.reload()
7 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: <i>Start_Ziel_Gruppe</i>

Benötigt
Makro: <i>Zeitmessung_Ende</i>

```

1 SUB Zeitmessung_Ende_Gruppe(oEvent AS OBJECT)
2   DIM oForm AS OBJECT
3   Zeitmessung_Ende(oEvent)

```

```

4   oForm = oEvent.Source.Model.Parent
5   oFormPlatz = oForm.Parent.GetByName("Platz")
6   oFormPlatz.reload()
7   END SUB

```

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

Zeitmessung_Start_Ziel

Aufruf aus

Formular: *Start_Ziel_Platz*

Benötigt

Makro: *Zeitmessung_Start*

```

1   SUB Zeitmessung_Start_Ziel(oEvent AS OBJECT)
2   DIM oForm AS OBJECT
3   DIM oFeld AS OBJECT
4   Zeitmessung_Start(oEvent)
5   oForm = oEvent.Source.Model.Parent
6   oFormZiel = oForm.Parent.GetByName("Ziel")
7   oFeld = oFormZiel.GetByName("Startnummern")
8   oFeld.refresh()
9   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_End_Listenfeld

Aufruf aus

Formular: *Nur_Ziel, Start_Ziel_Platz*

Benötigt

Makro: *Zeitmessung_End*

```

1   SUB Zeitmessung_End_Listenfeld(oEvent AS OBJECT)
2   DIM oForm AS OBJECT
3   DIM oFeld AS OBJECT
4   DIM stID AS STRING
5   oFeld = oEvent.Source.Model
6   stID = oFeld.CurrentValue
7   IF stID <> "" THEN
8       oForm = oFeld.Parent
9       oForm.Filter = ""ID"" = '"+stID+'""
10      oForm.ApplyFilter = TRUE
11      oForm.reload()
12      Zeitmessung_End(oEvent)
13      oFeld.refresh()
14   END IF
15   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_End» ab. Anschließend wird das Listenfeld

neu eingelesen, da jetzt ja eine "ID" weniger zur Verfügung steht. Die Person ist durchs Ziel gekommen.

Aktuelle Standardwerte 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 Datenbank² 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 Beispiele 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: <i>Defaultdatum_Aenderung, Defaultdatum_Makro, Defaultdatum_Makro_Standarddatum, Defaultdatum_Makro_Standarddatum_verlegt, Defaultdatum_Subform_neue_Datensaetze</i>
Abfrage: <i>DatumZeitvorgabe</i>
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

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:

```
16 ALTER TABLE "Datum_SQLDefault_Aenderdatum"
17 ALTER COLUMN "Datum"
18 SET DEFAULT CURRENT_DATE;
```

```
1 ALTER TABLE "Datum_SQLDefault_Aenderdatum"
2 ALTER COLUMN "AenderDatum"
3 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.

```
1 ALTER TABLE "Datum_SQLDefault_Aenderdatum"
2 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_A enderung	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 Tabellename 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

```

1 ALTER TABLE "Zeitstempel_SQLDefault_Aenderstempel"
2 ALTER COLUMN "Zeitstempel"
3 SET DEFAULT CURRENT_TIMESTAMP;

1 ALTER TABLE "Zeitstempel_SQLDefault_Aenderstempel"
2 ALTER COLUMN "Zeitstempel_Aenderung"
3 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.

```

1 ALTER TABLE "Zeitstempel_SQLDefault_Aenderstempel"
2 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 direkter SQL-Eingabe oder Makros das aktuelle Datum und (etwas begrenzt) auch der Zeitstempel in ein Formular übertragen werden.

```
1 SELECT DISTINCT CURRENT_TIMESTAMP AS "Jetzt",
2 CURRENT_DATE AS "Heute"
3 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».

```
1 SELECT "ID",
2 "Name",
3 "Zeitstempel",
4 "Zeitstempel_Aenderung",
5 CAST ( EXTRACT ( YEAR FROM "Zeitstempel" ) || '-' ||
RIGHT( '0' || EXTRACT ( MONTH FROM "Zeitstempel" ), 2 ) || '-' ||
RIGHT( '0' || EXTRACT ( DAY FROM "Zeitstempel" ), 2 ) AS DATE )
AS "StartDatum",
```

```

6 DATEDIFF( 'mi', "Zeitstempel", "Zeitstempel_Aenderung" )
  AS "Differenz_Minuten"
7 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 übrigen Zeichen ausgelesen. Das Datum wird dann in einem für SQL lesbaren 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 sondern eine Ansicht mit der Bezeichnung «Ansicht_Zeitdifferenzen_Form».

```

1 SELECT "ID",
2        "Name",
3        "Zeitstempel",
4        "Zeitstempel_Aenderung",
5        CONVERT ( EXTRACT ( YEAR FROM "Zeitstempel" ) || '-' ||
                  RIGHT( '0' || EXTRACT ( MONTH FROM "Zeitstempel" ), 2 ) || '-' ||
                  RIGHT( '0' || EXTRACT ( DAY FROM "Zeitstempel" ), 2 ) , DATE )
              AS "StartDatum",

```

```

6 ( 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",
7 ( ( 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"
8 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 Nachkommastelle 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:

```

6 ( ( 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",
7 ( ( ( 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 Rechnungen 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

The screenshot shows a form with the following fields and values:

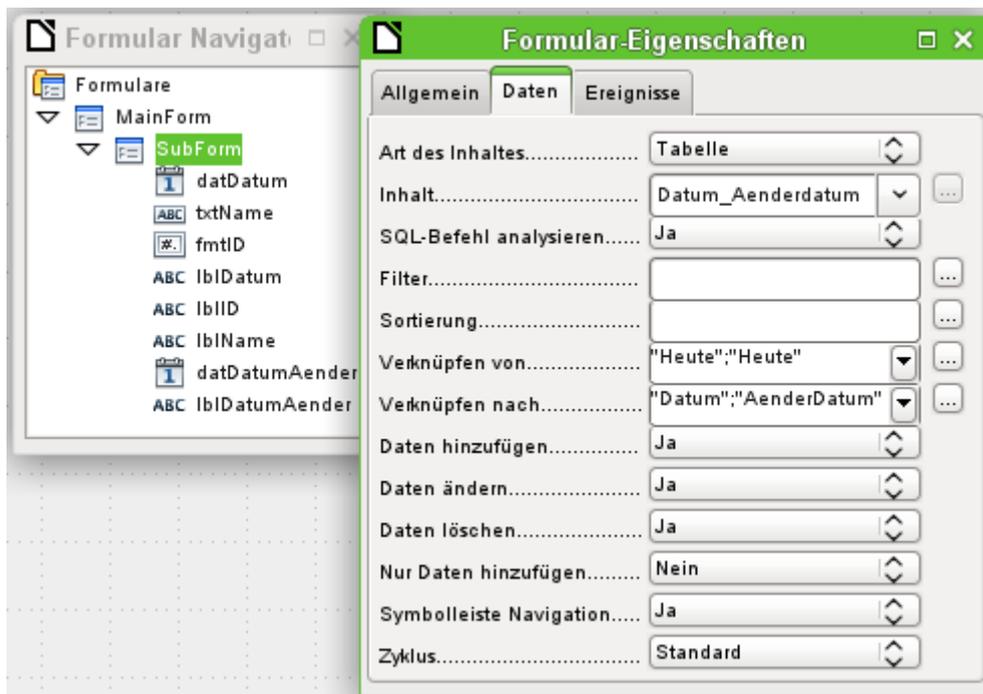
- ID:
- Name:
- Datum:
- Änderungsdatum:

Red circles highlight the 'ID' field (with a '1' next to it) and the 'Name' field (with '1.1' next to it).

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 Formelnavigator 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: <i>DatumZeitvorgabe</i>)	1.1	SubForm (Tabelle: <i>Zeitstempel_Aenderstempel</i>)
---	--	-----	--

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. <i>Update</i>
---	---	------------------------

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

The screenshot shows a form with the following fields and values:

- ID: 1
- Name: Duda
- Datum und Zeit: 13.10.2019 09:49:00
- Änderungsdatum und -zeit: 10.11.2019 19:39:00
- Button: Neuer Zeitstempel

A red circle highlights the number '1' next to the 'Datum und Zeit' field.

1 MainForm (Tabelle: [Zeitstempel_SQLDefault_Aenderstempel](#))

Makros in Feldeigenschaften

1	Button Neuer Zeitstempel (Aktion ausführen)	Module1. Update
---	---	---------------------------------

In den Zusatzinformationen des 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:

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

Eintrag für den Datumsstempel-Button:

```
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 Tabellename noch angepasst werden, da die Bezeichnung über 30 Zeichen lang ist.

Defaultdatum_Makro

ID	1
Name	Angie
Datum	02.05.20
Änderungsdatum	19.06.20

1

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

ID	3
Name	Eva
Datum und Zeit	12.10.2019 08:56:00
Änderungsdatum und -zeit	12.10.2019 15:55:00

1

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. Direkt 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 «Änderungsdatum und -zeit» mit einem neuen Wert versehen.

Defaultdatum_Makro_Standarddatum

The screenshot shows a form with three fields: 'ID' (empty), 'Name' (containing '1' and circled in red), and 'Datum' (containing '19.06.20').

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

The screenshot shows a form with three fields: 'ID' (empty), 'Name' (containing '1' and circled in red), and 'Datum' (containing '19.04.20').

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

ID	3
Name	Eva
Zeitstempel	12.10.2019 08:56:00

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

ID	1
Name	Adam
Datum und Zeit	11.10.2019 09:08:00
Änderungsdatum und -zeit	11.10.2019 15:27:00
Zeitdifferenz	6:19
Zeitdifferenzen der Person insgesamt	31:22

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

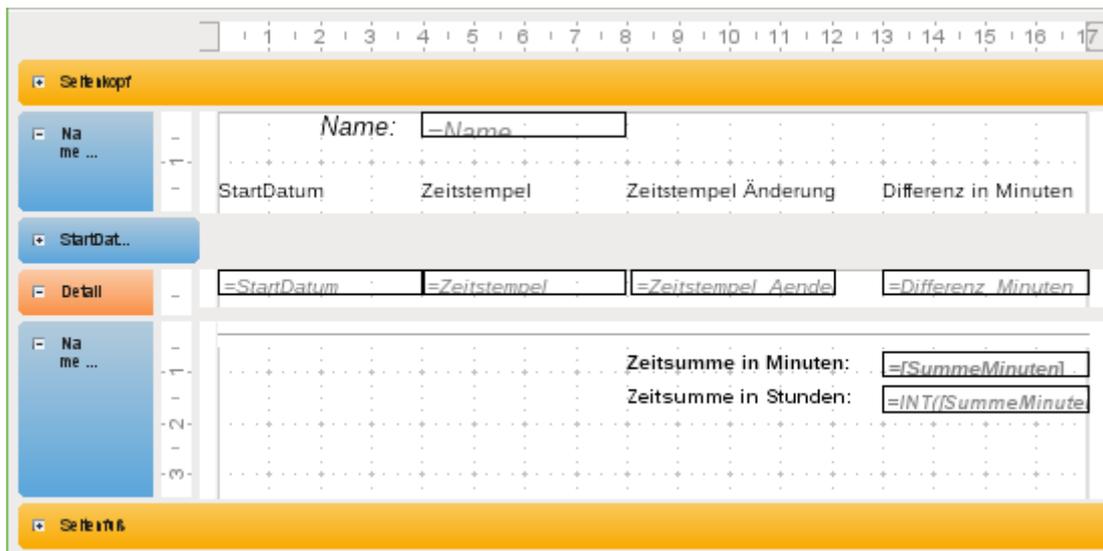
Zeitsumme in Minuten: 2337

Zeitsumme in Stunden: 38:57

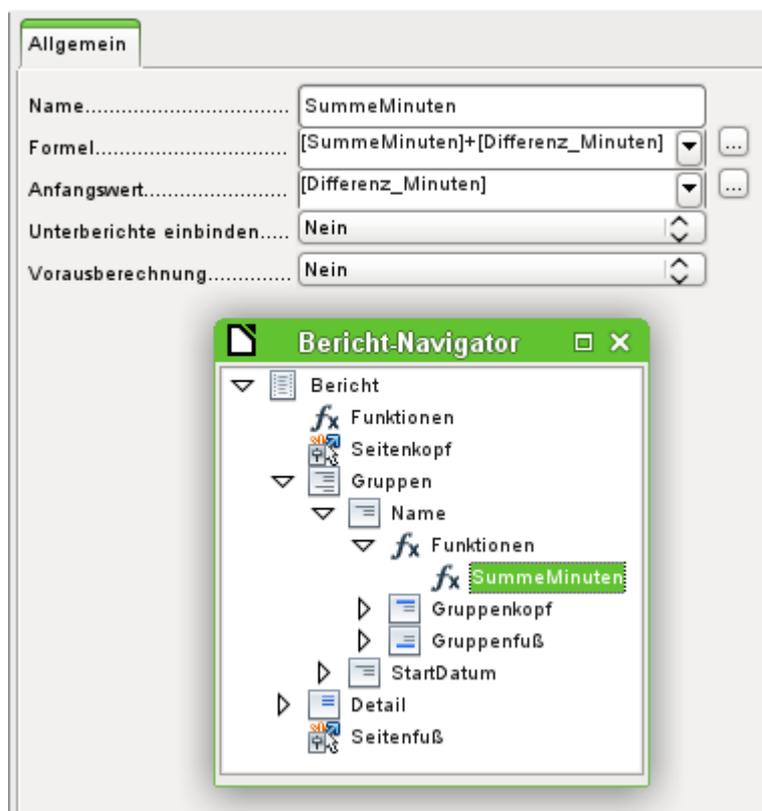
Datenquelle

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]**.

Allgemein	Daten
Datenfeld-Typ.....	Feld oder Formel
Datenfeld.....	INT([SummeMinuten]/60)&":"&MOD([SummeMinuten];60)
Funktion.....	
Geltungsbereich.....	

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_Formularvorlage» zu Grunde. In dieser Abfrage werden bereits die Zeiten berechnet, auch die Gesamtzeiten 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*

```
1 SUB Update(oEvent AS OBJECT)
2   DIM oForm AS OBJECT
3   DIM oFeld AS OBJECT
4   DIM oFeldID AS OBJECT
5   DIM stTag AS STRING
6   DIM arList()
7   DIM loRow AS LONG
8   oFeld = oEvent.Source.Model
9   oForm = oFeld.Parent
10  loRow = oForm.getRow
11  stTag = oFeld.Tag
12  arList = Split(stTag, ";")
13  oFeldID = oForm.getByName(Trim(arList(1)))
14  IF NOT IsEmpty(oFeldID.currentValue) THEN
15    oDatenquelle = ThisComponent.Parent.CurrentController
16    If NOT (oDatenquelle.isConnected()) THEN
17      oDatenquelle.connect()
18    END IF
19    oVerbindung = oDatenquelle.ActiveConnection()
20    oSQL_Anweisung = oVerbindung.createStatement()
21    stSql = arList(0) + " WHERE ""ID"" = " + oFeldID.CurrentValue
22    oSQL_Anweisung.executeUpdate(stSql)
23    oForm.reload
24    oForm.absolute(loRow)
25  END IF
26 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*

```
1 SUB Datum_aktuell
```

```

2   DIM oDoc AS OBJECT
3   DIM oDrawpage AS OBJECT
4   DIM oForm AS OBJECT
5   DIM oFeld AS OBJECT
6   DIM oFeldAender AS OBJECT
7   DIM unoDate
8   oDoc = thisComponent
9   oDrawpage = oDoc.drawpage
10  oForm = oDrawpage.forms.getByname("MainForm")
11  oFeld = oForm.getByname("datDatum")
12  oFeldAender = oForm.getByname("datDatumAender")
13  unoDate = createUnoStruct("com.sun.star.util.Date")
14  unoDate.Year = Year(Date)
15  unoDate.Month = Month(Date)
16  unoDate.Day = Day(Date)
17  IF isEmpty(oFeld.Date) THEN
18      oFeld.BoundField.updateDate(unoDate)
19  END IF
20  oFeldAender.BoundField.updateDate(unoDate)
21 END SUB

```

Der Zugriff auf die Felder im Formular wird über das Formulare Dokument erstellt (Zeile 8 bis 12). Anschließend wird das Datum als **Struct** zusammgebaut. 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: <i>Defaultdatum_Zeit_Makro, Defaultdatum_Zeitdifferenz_Makro</i>
Makro: <i>UpdateTimestamp</i>

```

1 SUB Datum_Zeit_aktuell
2   DIM oDoc AS OBJECT
3   DIM oDrawpage AS OBJECT
4   DIM oForm AS OBJECT
5   DIM oFeld AS OBJECT
6   DIM oFeldAender AS OBJECT
7   DIM unoStmp
8   oDoc = thisComponent
9   oDrawpage = oDoc.drawpage
10  oForm = oDrawpage.forms.getByname("MainForm")
11  oFeld = oForm.getByname("fmtDatumZeit")
12  oFeldAender = oForm.getByname("fmtDatumZeitAender")
13  unoStmp = createUnoStruct("com.sun.star.util.DateTime")
14  WITH unoStmp
15      .Year = Year(Date)
16      .Month = Month(Date)
17      .Day = Day(Date)
18      .Hours = Hour(Time)
19      .Minutes = Minute(Time)
20      .Seconds = Second(Time)
21      .NanoSeconds = 000000
22  END WITH
23  IF isEmpty(oFeld.CurrentValue) THEN
24      oFeld.BoundField.updateTimestamp(unoStmp)
25  END IF

```

```

26     oFeldAender.BoundField.updateTimestamp(unoStmp)
27 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 erstellte 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*

```

1 SUB Standarddatum
2     DIM oDoc AS OBJECT
3     DIM oDrawpage AS OBJECT
4     DIM oForm AS OBJECT
5     DIM oFeld AS OBJECT
6     DIM unoDate
7     oDoc = thisComponent
8     oDrawpage = oDoc.drawpage
9     oForm = oDrawpage.forms.getByName("MainForm")
10    oFeld = oForm.getByName("datDatum")
11    unoDate = createUnoStruct("com.sun.star.util.Date")
12    unoDate.Year = Year(Date)
13    unoDate.Month = Month(Date)
14    unoDate.Day = Day(Date)
15    oFeld.setPropertyValue("DefaultDate", unoDate)
16 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*

```

1 SUB Standarddatum_verlegt
2     DIM oDoc AS OBJECT
3     DIM oDrawpage AS OBJECT
4     DIM oForm AS OBJECT
5     DIM oFeld AS OBJECT
6     DIM unoDate
7     DIM daNew AS DATE
8     DIM i AS INTEGER
9     oDoc = thisComponent

```

```

10  oDrawpage = oDoc.drawpage
11  oForm = oDrawpage.forms.getByName("MainForm")
12  oFeld = oForm.getByName("datDatum")
13  unoDate = createUnoStruct("com.sun.star.util.Date")
14  i = -2
15  daNew = DateAdd("m", i, Date)
16  unoDate.Year = Year(daNew)
17  unoDate.Month = Month(daNew)
18  unoDate.Day = Day(daNew)
19  oFeld.setPropertyValue("DefaultDate", unoDate)
20  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 Vergangenheit (bei negativen Werten für **i**).

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

Standardzeitstempel

Aufruf aus

Formular: *Defaultdatum_Makro_Standardzeitstempel*

```

1  SUB StandardZeitstempel
2  DIM oDoc AS OBJECT
3  DIM oDrawpage AS OBJECT
4  DIM oForm AS OBJECT
5  DIM oFeld AS OBJECT
6  DIM stStmp AS STRING
7  oDoc = thisComponent
8  oDrawpage = oDoc.drawpage
9  oForm = oDrawpage.forms.getByName("MainForm")
10 oFeld = oForm.getByName("fmtZeitstempel")
11 stStmp = Year(Date) & "-" & Month(Date) & "-" & Day(Date) & " " & Hour(Time) &
    ":" & Minute(Time) & ":" & Second(Time)
12 oFeld.setPropertyValue("EffectiveDefault", stStmp)
13 END SUB

```

Beim Zeitstempel muss der Wert anders festgelegt werden. Die Formularassistenten bilden häufig den Zeitstempel in zwei getrennten Feldern ab: zum einen ein Datumfeld, 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*

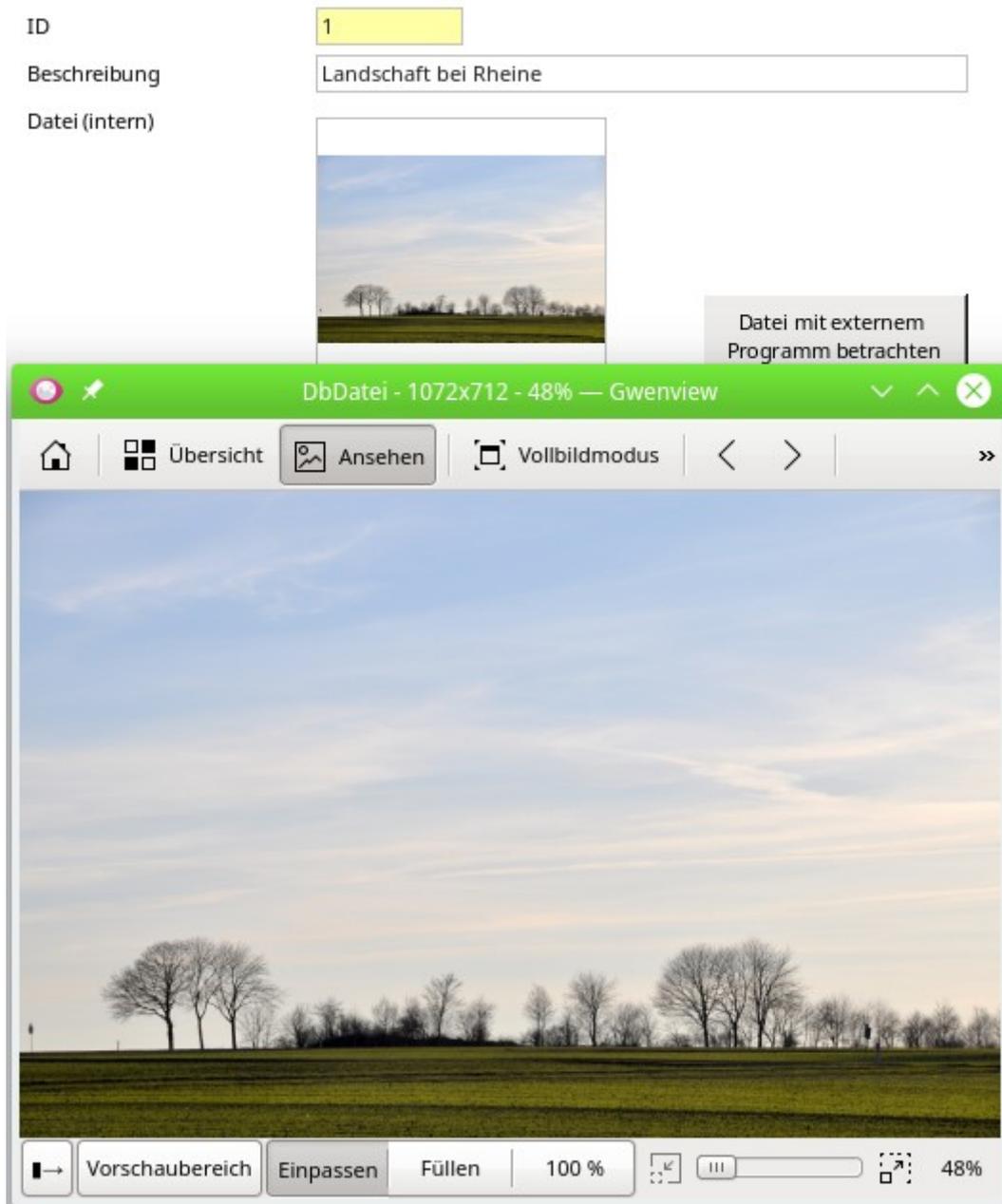
```
1 SUB UpdateTimestamp(oEvent AS OBJECT)
2   DIM oForm AS OBJECT
3   DIM oFeld AS OBJECT
4   DIM loRow AS LONG
5   Datum_Zeit_aktuell
6   oFeld = oEvent.Source.Model
7   oForm = oFeld.Parent
8   loRow = oForm.getRow
9   IF oForm.IsNew THEN
10    oForm.insertRow
11  ELSE
12    oForm.updateRow
13  END IF
14  oForm.reload
15  oForm.absolute(loRow)
16 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

Einführung



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 Betriebssystems 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.

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: <i>Pfadeingabe, Pfadeingabe_Tabellenkontrollfeld</i>
Ansicht: <i>Ansicht_Dateiendung</i>
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: <i>Dateiaufnahme, Dateiaufnahme_Name</i>
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.

3 Beispieldatenbank Beispiel_Dateien_einbinden.odt

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 Dateieindung 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_Dateieindung

Datenquelle
Tabelle: <i>Dateien</i>

Datenziel
Abfrage: <i>Bilder</i>

```

17 SELECT "Dateien"."ID","Dateien"."Beschreibung","Dateien"."Pfad" ,
18 LOWER ( SUBSTRING ( RIGHT( "Pfad", 5 )
FROM POSITION( '.' IN RIGHT( "Pfad", 5 ) ) + 1 ) ) AS "D_Endung"
19 FROM "Dateien"

```

Die Ansicht dient lediglich dazu, aus den Pfadbezeichnungen für die Dateien die Dateieindungen 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 Dateieindung für Bilder stecken. Die Dateieindung ist durch einen '.' abgetrennt. Die Position dieses Punktes dient dazu, die Dateieindung 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 Dateieindungen 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

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

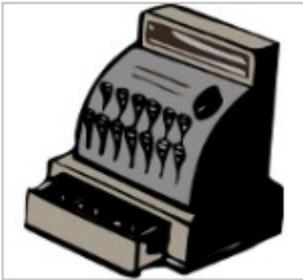
Die Abfrage filtert aus den Dateien diejenigen heraus, deren Endung den in Zeile 2 aufgeführten Endungen entspricht. Dies ist für den Bericht notwendig, der sonst bei anderen Dateien als Bilddateien und PDF-Dateien nur eine Fehlermeldung anzeigt. Diese Liste in Zeile 2 kann natürlich wesentlich alle 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.

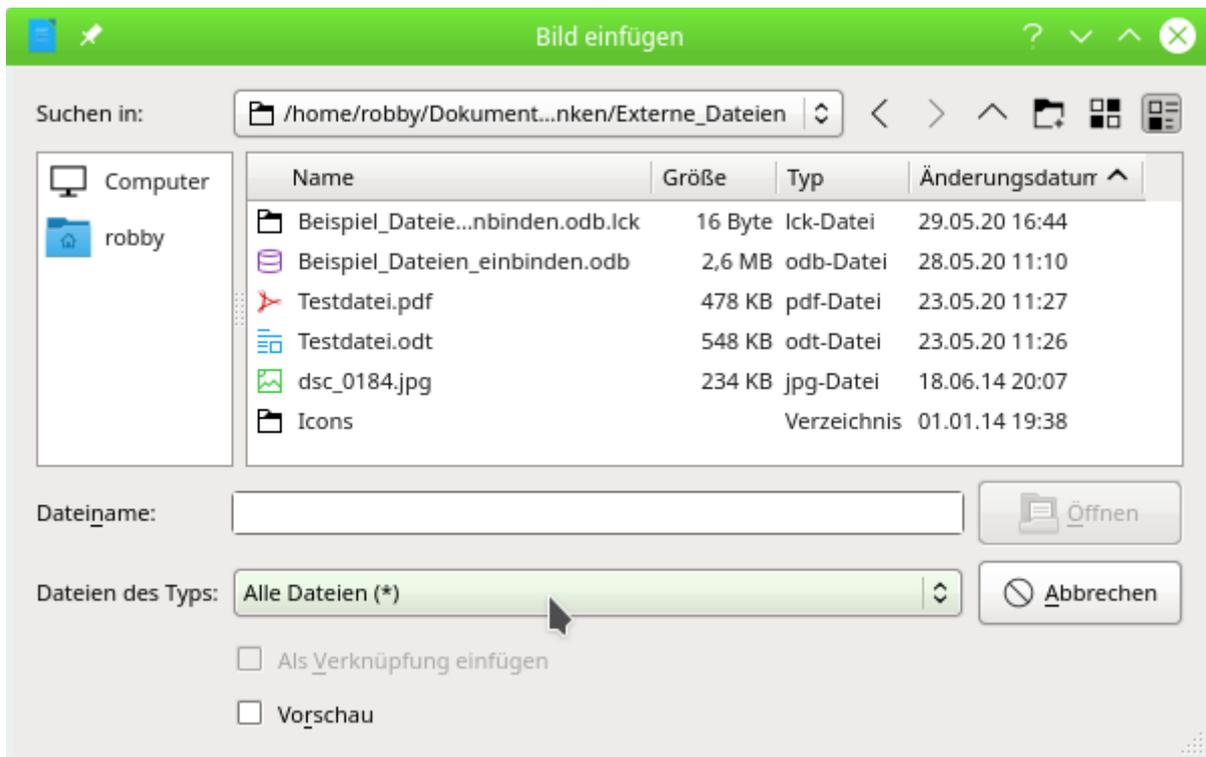
Pfadeingabe

ID	2
Beschreibung	Kasse
Datei (Pfad)	 

Datei mit externem Programm betrachten

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 Datei-betrachter 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

ID	Beschreibung	Datei (Pfad)
1	Werkzeug	Icons/Andy_Tools_Hammer_Spanner.png
2	Kasse	Icons/Anonymous_Architetto_-_Cassa_d_epo
3	Spätherbst in Rheine	dsc_0184.jpg
4	PDF-Datei mit Bild	Testdatei.pdf
5	ODT-Datei mit Bild	Testdatei.odt
>Feld<		



1

Datei mit externem Programm betrachten

1 Formular (Tabelle: *Dateien*)

Makros in Feldeigenschaften		
1	Schaltfläche Datei mit externem Programm betrachten (Aktion ausführen)	Module1. <i>Betrachten</i>

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

ID	1
Beschreibung	Landschaft bei Rheine
Datei (intern)	 <div style="text-align: right;"> 1 <input type="button" value="Datei mit externem Programm betrachten"/> </div>

1	Formular (Tabelle: <i>Dateien_intern</i>)
---	--

Makros in Feldeigenschaften		
1	Schaltfläche Datei mit externem Programm betrachten (Aktion ausführen)	Module1. <i>DateiAuslesen</i>

Äußerlich unterscheidet sich dieses Formular nicht vom Formular «Pfadeingabe». Allerdings wird hier die Datei in die Tabelle selbst eingelesen. Eine Verknüpfung nach 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 werden. Das Abspeichern von normalen Textdateien dagegen ist kein Problem.

Dateiaufnahme_Name

ID	4
Beschreibung	PDF-Datei
Datei (intern)	 <div style="text-align: right;"> 1 </div> <div style="text-align: right; margin-top: 10px;"> <div style="border: 1px solid gray; padding: 5px; background-color: #f0f0f0;">Datei mit externem Programm betrachten</div> </div>
DateiName	Testdatei.pdf

1 Formular (Tabelle: *Dateien_intern*)

Makros in Formulareigenschaften		
1	Vor der Datensatzaktion	Module1. <i>DateiName</i>
Makros in Feldeigenschaften		
1	Schaltfläche Datei mit externem Programm betrachten (Aktion ausführen)	Module1. <i>DateiAuslesen_mitName</i>
1	Graphisches Feld «Datei (intern)» (Bei Fokusverlust)	Module1. <i>DateiName</i>

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 Dateierdung 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	
<hr/>		
ID	2	
Beschreibung	Kasse	
Pfad	Icons/Anonymous_Architetto_-- _Cassa_d_epoca.png	
<hr/>		
ID	3	
Beschreibung	Spätherbst in Rheine	
Pfad	dsc_0184.jpg	
<hr/>		
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:

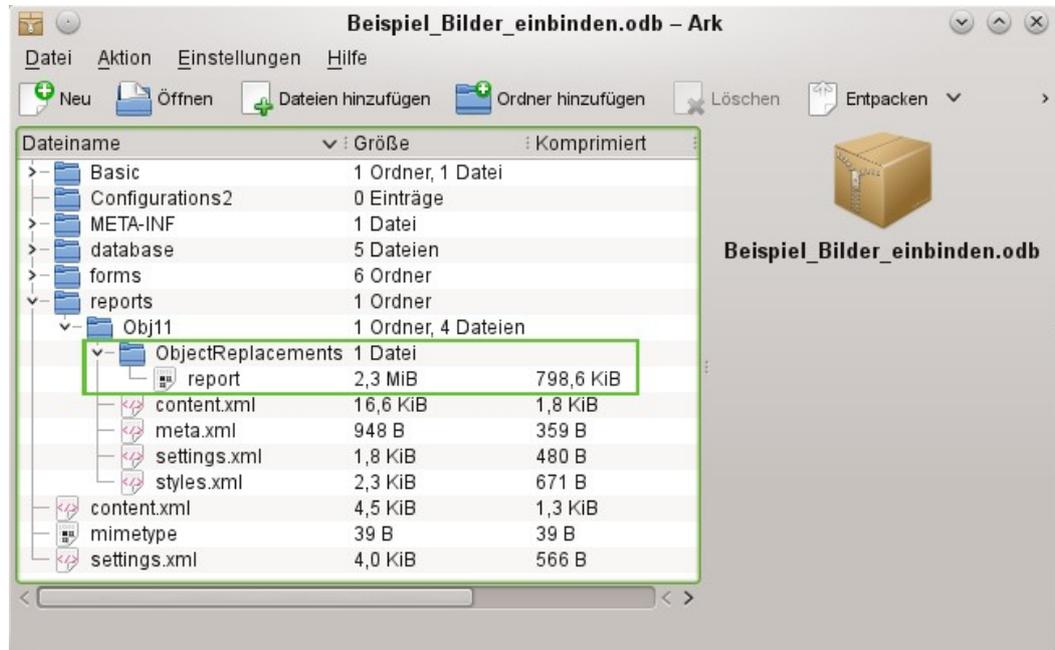


Der Bericht wird zum Editieren geöffnet. Das grafische Steuerelement wird markiert. In **Eigenschaften** → **Allgemein** → **Als Verknüpfung vorbereiten** wird «Nein» ausgewählt. Alle Bilder erscheinen dann problemlos. Bei anderen Dateien wie z.B. einer Writer-Datei erscheint dann ein kleines Rechteck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

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.

Hinweis



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*

```
3 SUB Betrachten
4   DIM oDoc AS OBJECT
5   DIM oDrawpage AS OBJECT
6   DIM oForm AS OBJECT
7   DIM oFeld AS OBJECT
8   DIM oShell AS OBJECT
9   DIM stUrl AS STRING
10  DIM stFeld AS STRING
11  DIM arUrl_Start()
12  oDoc = thisComponent
13  oDrawpage = oDoc.Drawpage
14  oForm = oDrawpage.Forms.getByName("Formular")
```

```

15  oFeld = oForm.getByName("GraphischesFeld")
16  stUrl = oFeld.BoundField.getString
17  arUrl_Start = split(oDoc.Parent.Url, oDoc.Parent.Title)
18  oShell = createUnoService("com.sun.star.system.SystemShellExecute")
19  stFeld = convertToUrl(arUrl_Start(0) + stUrl)
20  oShell.execute(stFeld,,0)
21  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*

```

1  SUB Dateiauslesen
2  DIM oDoc AS OBJECT
3  DIM oDrawpage AS OBJECT
4  DIM oForm AS OBJECT
5  DIM oFeld AS OBJECT
6  DIM oStream AS OBJECT
7  DIM oShell AS OBJECT
8  DIM oPath AS OBJECT
9  DIM oSimpleFileAccess AS OBJECT
10 DIM st AS STRING
11 DIM stPfad AS STRING
12 DIM stFeld AS STRING
13 oDoc = thisComponent
14 oDrawpage = oDoc.Drawpage
15 oForm = oDrawpage.Forms.getByName("Formular")
16 oFeld = oForm.getByName("GraphischesFeld")
17 oStream = oFeld.BoundField.getBinaryStream
18 oPath = createUnoService("com.sun.star.util.PathSettings")
19 st = ""
20 stPfad = oPath.Temp & "/DbDatei" & st
21 oSimpleFileAccess = createUnoService("com.sun.star.ucb.SimpleFileAccess")
22 oSimpleFileAccess.writeFile(stPfad, oStream)
23 oShell = createUnoService("com.sun.star.system.SystemShellExecute")
24 stFeld = convertToUrl(stPfad)
25 oShell.execute(stFeld,,0)
26 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»

```
1 SUB DateiAuslesen_mitName
2   ...
3   oFeld2 = oForm.getByNamed("DateiName")
4   stName = oFeld2.Text
5   IF stName = "" THEN
6     stName = "DbDatei"
7   END IF
8   ...
9   stPfad = oPath.Temp & "/" & stName
10  ...
11 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 in das entsprechende Wunschverzeichnis übertragen werden.

DateiName

Aufruf aus

Formular: *Dateiaufnahme_Name*

```
1 SUB DateiName
2   DIM oDoc AS OBJECT
3   DIM oDrawpage AS OBJECT
4   DIM oForm AS OBJECT
5   DIM oField AS OBJECT
6   DIM stImageUrl AS STRING
7   DIM stFile AS STRING
8   GlobalScope.BasicLibraries.LoadLibrary("Tools")
```

```

9      oDoc = thisComponent
10     oDrawpage = oDoc.Drawpage
11     oForm = oDrawpage.Forms.getByname("Formular")
12     oField = oForm.getByname("GraphischesFeld")
13     stImageUrl = oField.imageUrl
14     IF stImageUrl <> "" THEN
15         stFile = FileNameoutofPath(stImageUrl)
16         oForm.UpdateString(oForm.FindColumn("Dateiname"), stFile)
17     END IF
18 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 Serienbriefeffunktionalität 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: <i>Anschrift, Rechnung, Rechnung_Textfelder, Rechnung_Textfelder_Uebertrag, Rechnung_Textfelder_Uebertrag_Seriendruck</i>
Abfrage: <i>Adresseingabe, Rechnung_einzeilig, Fehler: Verweis nicht gefunden, Rechnung_Textfelder</i>

Feldname	Feldtyp	Beschreibung
ID	Integer	Die ID ist Primärschlüssel dieser Tabelle. Sie ist als Autowert festgelegt.

⁴ Beispieldatenbank Beispiel_Datenbank_Serienbrief_direkt.odt

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
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: <i>Rechnung_einzeilig</i>
Makro: <i>Textfelder_Fuellen, Rechnungsinhalt_zusammenstellen, Tabellen_fuellen_Uebertrag</i>
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: <i>Rechnung, Rechnung_Textfelder, Rechnung_Textfelder_Uebertrag, Rechnung_Textfelder_Uebertrag_Seriendruck</i>
Abfrage: <i>Rechnung_Textfelder</i>
Makro: <i>Rechnungsinhalt_zusammenstellen, Rechnungsinhalt_zusammenstellen_Tabelle, Rechnungsinhalt_zusammenstellen_Tabelle_Uebertrag, Rechnungsinhalt_zusammenstellen_Tabelle_Uebertrag_Direktdruck</i>

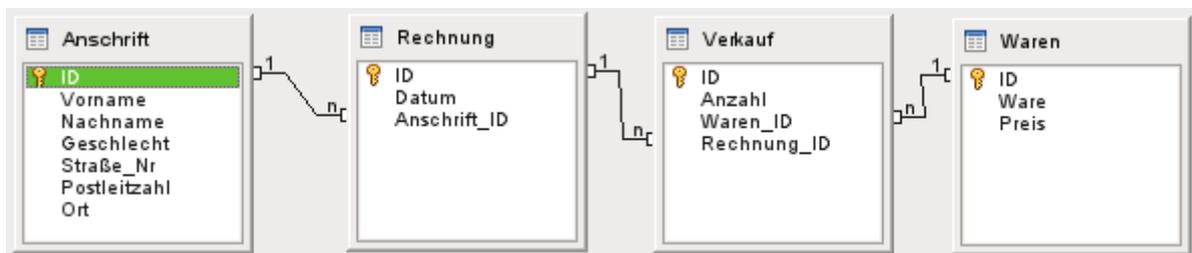
Feldname	Feldtyp	Beschreibung
ID	Integer	Die ID ist Primärschlüssel dieser Tabelle. Sie ist als Autowert festgelegt.
Anzahl	Tiny Integer	Je größer die Anzahl der Waren sein sollte, desto größer ist natürlich der Umfang dieses Feldes festzulegen. Mit Tiny Integer geht die Anzahl von -128 bis +127 – für 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: <i>Rechnung, Rechnung_Textfelder, Rechnung_Textfelder_Uebertrag, Rechnung_Textfelder_Uebertrag_Seriendruck, Waren</i>
Abfrage: <i>Etiketten, Rechnung_einzeilig, Rechnung_Textfelder</i>
Makro: <i>Rechnungsinhalt_zusammenstellen, Rechnungsinhalt_zusammenstellen_Tabelle, Rechnungsinhalt_zusammenstellen_Tabelle_Uebertrag, Rechnungsinhalt_zusammenstellen_Tabelle_Uebertrag_Direktdruck</i>

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: <i>Anschrift</i>
Datenziel
Formular: <i>Anschrift_Textfelder</i>

```

1 SELECT "ID", "Vorname", "Nachname",
2 "Geschlecht" AS "Geschl_kurz",

```

```

3 "Straße_Nr", "Postleitzahl", "Ort",
4 CASE WHEN "Geschlecht" = 'm' THEN 'Herrn' ELSE 'Frau' END AS "Geschlecht"
5 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*

```

1 SELECT RIGHT( '0000000' || "ID", 8 ) "WarenNr",
2 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",
3 REPLACE( "Preis" || ' €', '.', ',' ) "Preis"
4 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 höchstens zwei Zeilen.

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

```

2 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 Etiketle 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*

```
1 SELECT "Anschrift"."Vorname", "Anschrift"."Nachname", "Anschrift"."Straße_Nr",
2 "Anschrift"."Postleitzahl", "Anschrift"."Ort",
3 "a"."Datum" AS "Rechnungsdatum", "a"."ID",
4 "Rechnungsinhalt"."Rechnungsinhalt",
5 REPLACE( ( SELECT SUM( "Verkauf"."Anzahl" * "Waren"."Preis" ) FROM "Verkauf", "Waren"
6 WHERE "Verkauf"."Waren_ID" = "Waren"."ID" AND "Verkauf"."Rechnung_ID" = "a"."ID" ) ||
' €', '.', ',' ) AS "Summe"
FROM "Rechnung" AS "a", "Anschrift", "Rechnungsinhalt"
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 «Ambiguous 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*

```
1 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*

```
1 SELECT "Anschrift"."Vorname", "Anschrift"."Nachname", "Anschrift"."Straße_Nr",
2 "Anschrift"."Postleitzahl", "Anschrift"."Ort",
3 RIGHT( '0' || EXTRACT ( DAY FROM "a"."Datum" ), 2 ) || '.' || RIGHT( '0' || EXTRACT (
4 MONTH FROM "a"."Datum" ), 2 ) || '.' || EXTRACT ( YEAR FROM "a"."Datum" )
5 "Rechnungsdatum",
```

```

3 'RE_' || RIGHT( '0000000' || "a"."ID", 8 ) AS "Rechnungsnummer,
4 "a"."ID",
5 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",
6 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",
7 REPLACE( ( SELECT SUM( "Verkauf"."Anzahl" * "Waren"."Preis" ) FROM "Verkauf", "Waren"
WHERE "Verkauf"."Waren_ID" = "Waren"."ID" AND "Verkauf"."Rechnung_ID" = "a"."ID" ) ||
' €', '.', ',' ) "Summe"
8 FROM "Rechnung" AS "a", "Anschrift"
9 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 Währungsbeitrag 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

The form contains the following fields and values:

- ID: 0
- Vorname: Robert
- Nachname: Großkopf
- Geschlecht: männlich
- Straße_Nr: Alleestraße mit vielen Bäumen rechts und links
- Postleitzahl: 12390
- Ort: Flachland

A red circle with the number '1' is drawn around the 'Geschlecht' dropdown menu.

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.

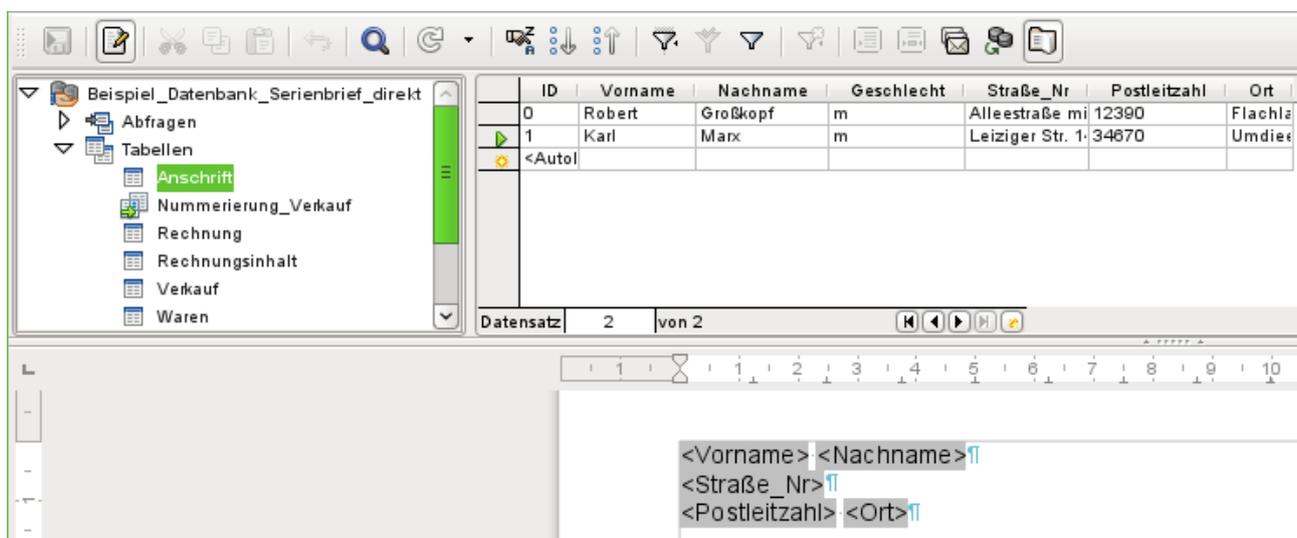


Das Listenfeld wird nicht über eine zusätzliche Tabelle per SQL geladen, sondern ist hier direkt über eine Werteliste definiert. Unter **Eigenschaften** → **Daten** → **Listeninhalt** sind die Werte «m» und «w». Um die Werte in dieser Form einzugeben muss lediglich das vorgesehene Feld mit dem Cursor 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

ID
0

Vorname: Robert Nachname: Großkopf

Geschlecht: männlich

Straße_Nr: Alleestraße mit vielen Bäumen rechts und links

Postleitzahl: 12390 Ort: Flachland

Druck über Writer

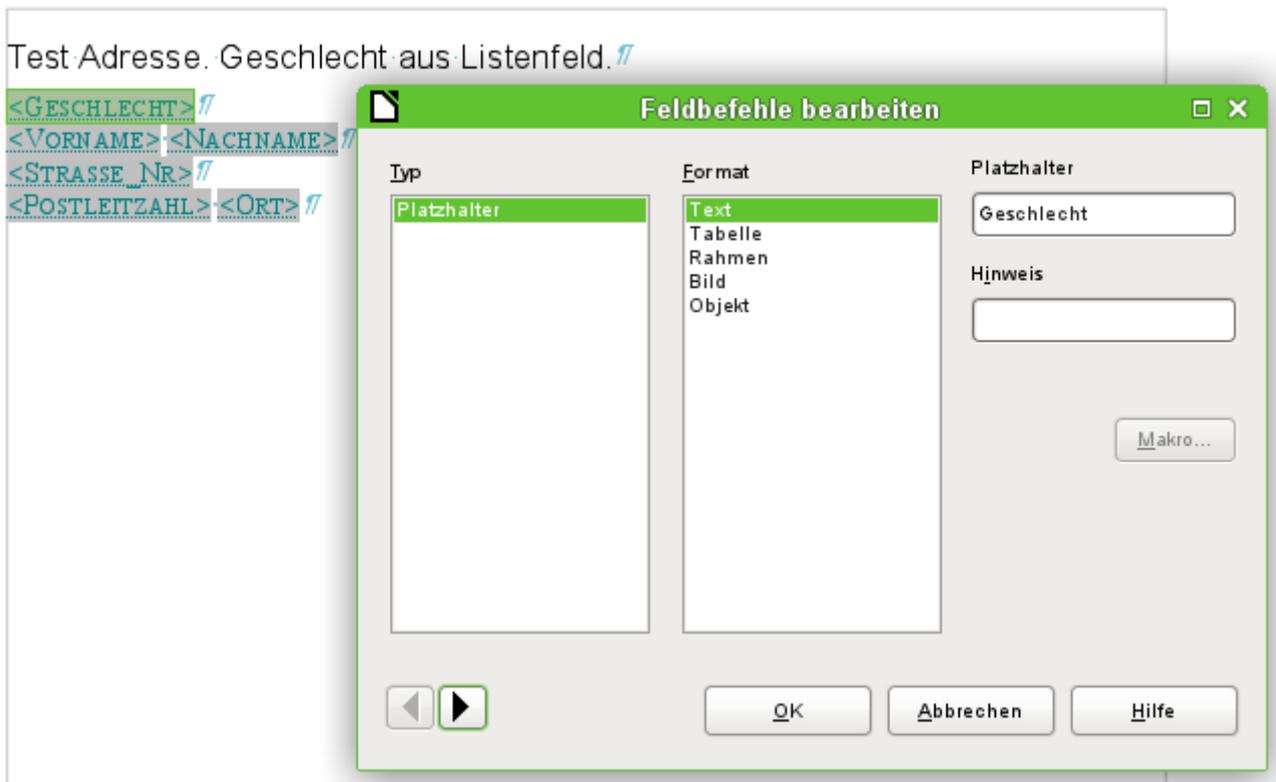
1

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.



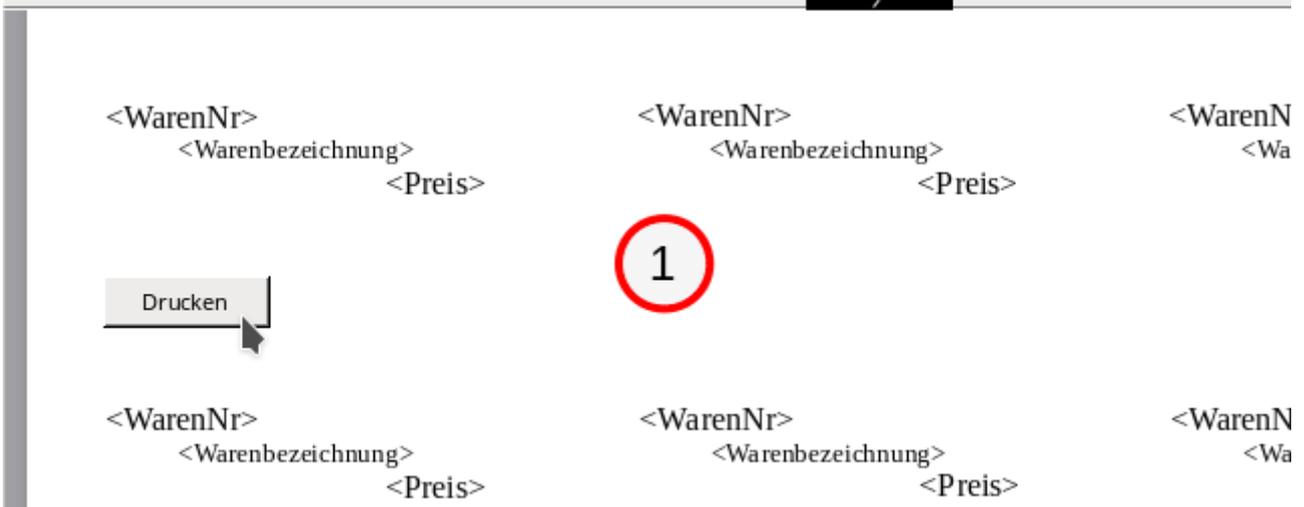
Für einfache Zwecke reicht hier der Typ «Text», mit den anderen Varianten, z.B. «Grafik», können dann weitere Inhalte umgesetzt werden.

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

WarenNr	Warenbezeich...	Preis
00000000	Papier, 500	5,65 €
00000001	Bleistift HB	0,25 €
00000002	Schnellhefter,	0,46 €
00000003	Hefter,	11,25 €
00000004	Locher,	15,48 €
00000005	Ultrabook	689,00 €
00000006	MiniPC Nano	398,00 €
00000007	Desktop-PC	499,00 €

Datensatz 1 von 15



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

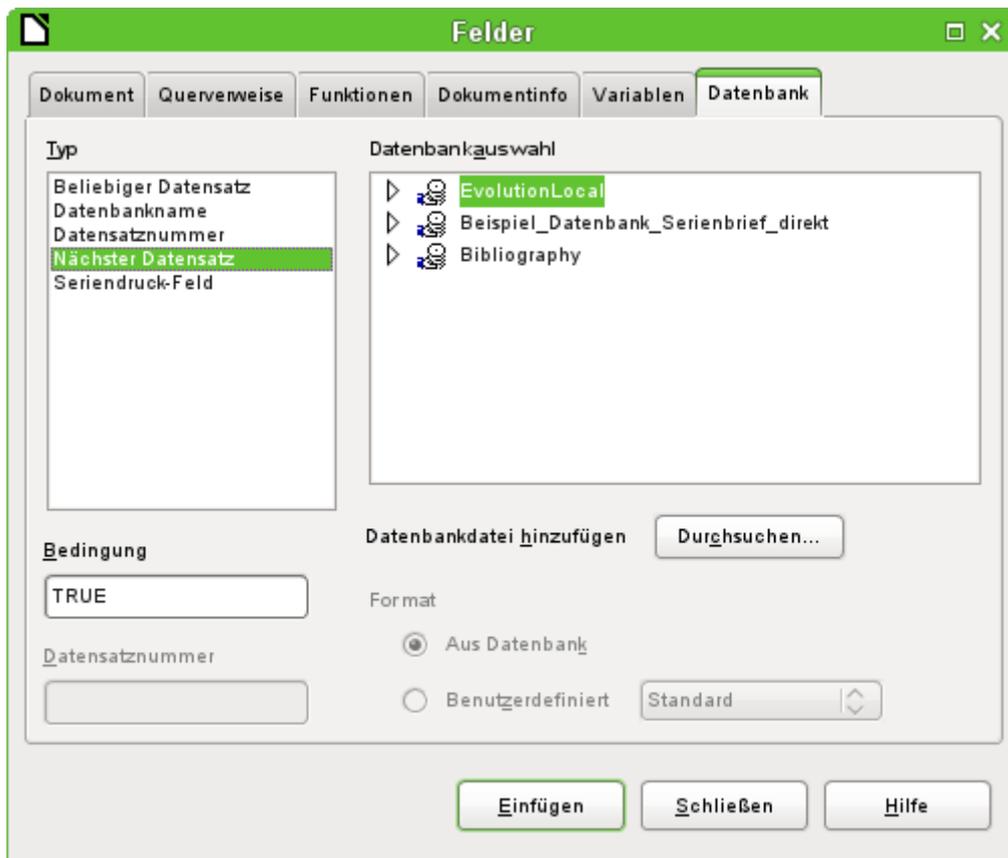
Benötigt

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 Formulare Dokument 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.

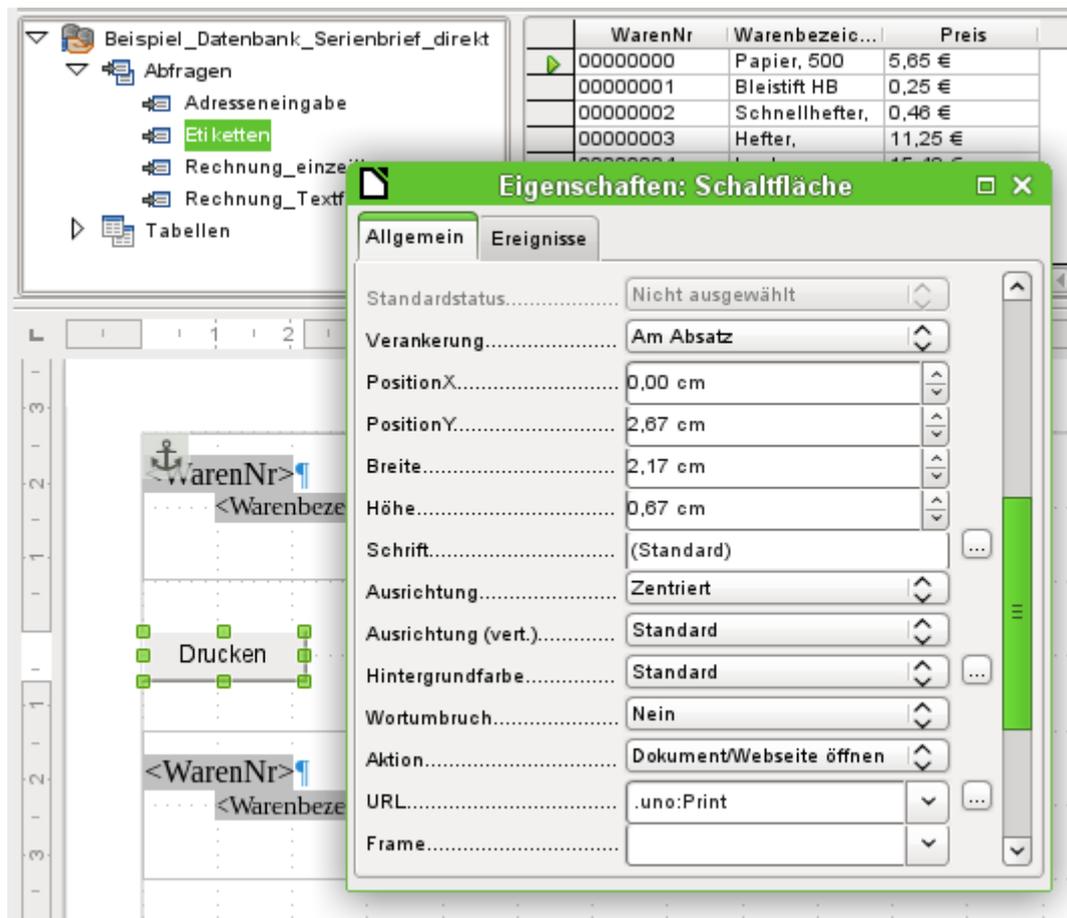


Über **Einfügen** → **Feldbefehl** → **Andere** → **Datenbank** wird als Abschlussfeld eines jeden Etiketts **Nächster Datensatz** → **TRUE** gewählt. Lediglich das letzte Etikett der Seite ist hiervon ausgenommen, da mit einem Seitenumbruch automatisch zum nächsten Datensatz gewechselt wird.

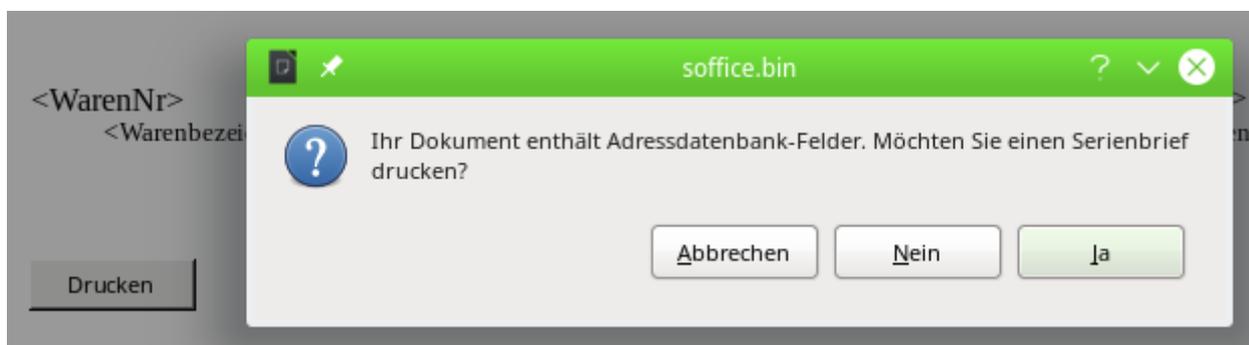
Die Rahmen werden in der gewünschten Anzahl kopiert, neu eingefügt und an entsprechender Stelle auf dem Blatt positioniert. In der Beispieldatei wurde dabei jeder Rahmen an die Seite gebunden – dies kann natürlich nach Belieben anders gehandhabt werden.

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

Anzahl	Ware		
2	Papier, 500 Blatt	-	5,65 €
10	Bleistift HB	-	0,25 €
5	Schnellhefter, Pappe	-	0,46 €
1	Hefter, Tischgerät	-	11,25 €
1	Locher, Registratur	-	15,48 €
	Bleistift HB	-	0,25 €
	Hefter, Tischgerät	-	11,25 €
	Locher, Registratur	-	15,48 €
	Papier, 500 Blatt	-	5,65 €
	Schnellhefter, Pappe	-	0,46 €

1	MainForm (Tabelle: <i>Rechnung</i>)
1	Listenfeld Kunde (Tabelle: <i>Anschrift</i>)
1.1	SubForm (Tabelle: <i>Verkauf</i>)
1.1	Listenfeld Ware (Tabelle: <i>Waren</i>)

Makros in Feldeigenschaften

1	Schaltfläche Rechnung drucken (Aktion ausführen)	Module1. <i>Rechnung</i>
---	--	--------------------------

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:

```

1 SELECT "Nachname"||', '||"Vorname", "ID"
2 FROM "Anschrift"
3 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 werden. Das

Tabellenkontrollfeld wird markiert. Über das **Kontextmenü** → **Kontrollfeld** → **Eigenschaften: Tabellen-Steuer-element** → **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:

```
1 SELECT
2     LEFT("Ware" || SPACE(25), 25) || ' - ' ||
3     REPLACE( RIGHT( SPACE(8) || "Preis", 8), '.', ',') || ' €',
4     "ID"
5     FROM "Waren"
6     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:

```
1 SELECT
2     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

```
2     RPAD(REPLACE("Ware", ASCII_CHAR(10), ' '), 25) || ' - ' ||
3     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:

Base – Writer Serienbrieftest

Panikallee 13
12345 Woheraachimmer

Serienbrief – Panikallee 13 – 12345 Woheraachimmer

Robert Großkopf
Alleestraße mit vielen Bäumen rechts und links
12390 Flachland

Rechnungsdatum: 11.05.13
Rechnungsnummer: 0

2 Papier, 500 Blatt	5,65 € → 11,30 €
10 Bleistift HB	0,25 € → 2,50 €
5 Schnellhefter, Papp	0,46 € → 2,30 €
1 Hefter, Tischgerät	11,25 € → 11,25 €
1 Locher, Registratur	15,48 € → 15,48 €

Insgesamt zu zahlen: 42,83 €

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.

Base – Writer Serienbrieftest

Panikallee 13
12345 Woheraachimmer

Serienbrief – Panikallee 13 – 12345 Woheraachimmer

<Vorname> <Nachname>
<Straße_Nr>
<Postleitzahl> <Ort>

Rechnungsdatum: <Rechnungsdatum>
Rechnungsnummer: <ID>

<Rechnungsinhalt>

Insgesamt zu zahlen: <Summe>

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

ID
0

Datum
11.04.20

Kunde
Großkopf, Robert

Rechnung d... 1.2

Anzahl	Ware	
2	Papier, 500 Blatt	- 5,65 €
10	Bleistift HB	- 0,25 €
5	Schnellhefter, Pappe	- 0,46 €
1	Hefter, Tischgerät	- 11,25 €
1	Locher, Registratur	- 15,48 €
2	Bleistift HB	- 0,25 €
1	Locher, Registratur	- 15,48 €
3	Papier, 500 Blatt	- 5,65 €
4	Hefter, Tischgerät	- 11,25 €
5	Bleistift HB	- 0,25 €
2	Bleistift HB	- 0,25 €
1	Locher, Registratur	- 15,48 €

Datensatz 1 von 30

1	MainForm (Tabelle: <i>Rechnung</i>)
1	Listefeld Kunde (Tabelle: <i>Anschrift</i>)
1.1	SubForm (Tabelle: <i>Verkauf</i>)
1.1	Listefeld Ware (Tabelle: <i>Waren</i>)
1.2	Druck (Abfrage: <i>Rechnung_Textfelder</i>)

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 Woheraachimmer ¶

Rechnungstest – Panikallee 13 – 12345 Woheraachimmer ¶

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 Woheraachimmer ¶

Rechnungstest – Panikallee 13 – 12345 Woheraachimmer ¶				
<VORNAME> <NACHNAME> ¶				
<STRASSE NR> ¶				
<POSTLEITZAHL> <ORT> ¶				
	→	Rechnungsdatum:	→	<RECHNUNGSDATUM> ¶
	→	Rechnungsnummer:	→	<RECHNUNGSNUMMER> ¶
→	→		→	Nettobetrag: → <NETTO> ¶
→	→		→	Umsatzsteuer 19%: → <UMS ATZS TEUER> ¶
→	→		→	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

The screenshot shows a software interface for creating an invoice. At the top, there are several input fields: 'ID' with the value '0', 'Datum' with '11.04.20', and 'Kunde' with 'Großkopf, Robert'. To the right of these fields is a button labeled 'Rechnung drucken'. Below these fields is a table with two columns: 'Anzahl' and 'Ware'. The table contains several rows of items, including 'Papier, 500 Blatt', 'Bleistift HB', 'Schnellhefter, Pappe', 'Hefter, Tischgerät', and 'Locher, Registratur'. At the bottom of the table, there are navigation buttons and a status bar showing 'Datensatz 1 von 30'. Red circles with numbers are overlaid on the interface: a '1' is circled around the date field, a '1.1' is circled around the quantity '1' in the table row '1 Locher, Registratur', and a '1.2' is circled around the 'Rechnung drucken' button.

1	MainForm (Tabelle: <i>Rechnung</i>)
1	Listefeld Kunde (Tabelle: <i>Anschrift</i>)
1.1	SubForm (Tabelle: <i>Verkauf</i>)
1.1	Listefeld Ware (Tabelle: <i>Waren</i>)
1.2	Druck (Abfrage: <i>Rechnung_Textfelder</i>)

Makros in Feldeigenschaften		
1.2	Schaltfläche Rechnung drucken (Aktion ausführen)	Module1. <i>Rechnungsinhalt_zusammenstellen_Tabelle_Uebertrag</i>

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 Bildschirm: 15", entspiegelt, matt Prozessor: Intel Core i5 Arbeitsspeicher: 8 GB	689,00 €	1.378,00 €
1	Spiegelreflex-Kamera mit Zoomoptik 25-70mm incl. Speicherkarte 32GB, Li-Ionen-Akkus und Akkuladegerät	834,56 €	834,56 €
		Übertrag:	8.320,85 €

→ Seite 2 von 3

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

Rechnungstest

			Panikallee 13 12345 Woherauchimmer
		Übertrag:	8.320,85 €
3	Ultrabook Bildschirm: 15", entspiegelt, matt Prozessor: Intel Core i5 Arbeitsspeicher: 8 GB	689,00 €	2.067,00 €
→	→	→ 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: «Seriendruck»	Module1. Direktdruck_Start
1.2	Schaltfläche einzelne Rechnung drucken (Aktion ausführen) Eigenschaften → Allgemein → Zusatzinformation: «Einzeldruck»	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

Ware	MiniPC Nano Prozessor: i5 Arbeitsspeicher: 8 GB Anschlüsse: 2*USB3.0, 4*USB2.0, 1*eSata, HDMI, 1Gbit LAN Größe: 220mm*197mm*63mm	Preis	398,00 €
	<div style="border: 2px solid red; border-radius: 50%; width: 30px; height: 30px; display: flex; align-items: center; justify-content: center; margin: 0 auto;">1</div>		

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 Ausdrücke 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](#)

```

1 SUB Rechnung
2   DIM oForm AS OBJECT
3   DIM loFeldID AS LONG
4   DIM loID AS LONG
5   oForm = thisComponent.Drawpage.Forms.MainForm
6   loFeldID = oForm.findColumn("ID")
7   loID = oForm.getLong(loFeldID)
8   Rechnungsinhalt_zusammenstellen(loID)
9   Output_MailMerge(loID, "Beispiel_Rechnung.odt", 1, "Rechnung_einzeilig")

```

10 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 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:

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

Rechnungsinhalt_zusammenstellen

Aufruf aus

Makro: *Rechnung*

Benötigt

Tabelle: *Verkauf, Waren, Rechnungsinhalt*

```
1 SUB Rechnungsinhalt_zusammenstellen(loID AS LONG)
2   DIM oDatenquelle AS OBJECT
3   DIM oVerbindung AS OBJECT
4   DIM oSQL_Anweisung AS OBJECT
5   DIM oAbfrageergebnis AS OBJECT
6   DIM stSql AS STRING
7   DIM stText AS STRING
8   oDatenquelle = ThisComponent.Parent.CurrentController
9   If NOT (oDatenquelle.isConnected()) THEN
10      oDatenquelle.connect()
11  END IF
12  oVerbindung = oDatenquelle.ActiveConnection()
13  oSQL_Anweisung = oVerbindung.createStatement()
14  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+"
15  oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
16  WHILE oAbfrageergebnis.next
17      stText = stText + oAbfrageergebnis.getString(1)
18  WEND
19  stSql = "DELETE FROM ""Rechnungsinhalt"" WHERE ""ID"" = "+loID+"
20  oSQL_Anweisung.executeUpdate(stSql)
21  stSql = "INSERT INTO ""Rechnungsinhalt"" (""ID"", ""Rechnungsinhalt"") VALUES
          (""+loID+", '"+stText+"')
22  oSQL_Anweisung.executeUpdate(stSql)
23 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 Daten-

bank 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: <i>Anschrift</i>

Benötigt
Makro: <i>Output_MailMerge</i>
Vorlage: <i>Beispiel_Serienbrief.odt</i>

```
1 SUB Serienbrief
2   DIM oForm AS OBJECT
3   DIM loFeldID AS LONG
4   DIM loID AS LONG
5   oForm = thisComponent.Drawpage.Forms.MainForm
6   loFeldID = oForm.findColumn("ID")
7   loID = oForm.getLong(loFeldID)
8   Output_MailMerge(loID, "Beispiel_Serienbrief.odt", 0, "Anschrift")
9 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: <i>Rechnung, Serienbrief</i>

```
1 SUB Output_MailMerge(loID AS LONG, stFile AS STRING, inCommandType AS INTEGER,
2   stCommand AS STRING)
3   DIM oDB AS OBJECT
4   DIM stDir AS STRING
5   DIM arProps()
6   MailMerge = CreateUnoService("com.sun.star.text.MailMerge")
7   oDB = ThisComponent.Parent
8   stDir = Left(oDB.Location, Len(oDB.Location)-Len(oDB.Title))
9   WITH MailMerge
10    .DataSourceName = "Beispiel_Datenbank_Serienbrief_direkt"
11    .DocumentURL = stDir + stFile
12    .CommandType = inCommandType
13    .Command = stCommand
14    .Filter = "ID =" + loID
15    .SaveAsSingleFile = true
16    .FileNameFromColumn = true
17    .FileNamePrefix = "Nachname"
```

```

17         .OutputType = 2
18         .OutputUrl = stDir
19     END WITH
20     MailMerge.execute(arProps())
21 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 Ausgabebetyp 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](#)

```

1 SUB Textfelder_Fuellen(oEvent AS OBJECT)
2     DIM oForm AS OBJECT

```

```

3   DIM oDB AS OBJECT
4   DIM oNewDoc AS OBJECT
5   DIM stDir AS STRING
6   oForm = oEvent.Source.Model.Parent
7   oDB = ThisComponent.Parent
8   stDir = Left(oDB.Location, Len(oDB.Location) - Len(oDB.Title))
9   stDir = stDir & "Beispiel_Textfelder.ott"
10  DIM args(0) AS NEW com.sun.star.beans.PropertyValue
11  args(0).Name = "AsTemplate"
12  args(0).Value = True
13  oNewDoc = StarDesktop.loadComponentFromURL(stDir, "_blank", 0, args)
14  Textfelder_fuellen_ID(oNewDoc, oForm)
15 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 Vorlagendatei «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 «Anschritt_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*

```

1  FUNCTION Textfelder_Fuellen_ID(oNewDoc AS OBJECT, oForm AS OBJECT) AS LONG
2  DIM oColumns AS OBJECT
3  DIM oTextfields AS OBJECT
4  DIM oTextfield AS OBJECT
5  DIM stColumnName AS STRING
6  DIM stDir AS STRING
7  DIM inIndex AS INTEGER
8  DIM loID AS LONG
9  oColumns = oForm.Columns
10 oTextfields = oNewDoc.Textfields.createEnumeration
11 DO WHILE oTextfields.hasMoreElements
12     oTextfield = oTextfields.nextElement
13     IF oTextfield.supportsService("com.sun.star.text.TextField.JumpEdit") THEN
14         stColumnName = oTextfield.PlaceHolder
15         IF oColumns.hasByName(stColumnName) THEN
16             inIndex = oForm.findColumn(stColumnName)
17             oTextfield.Anchor.String = oForm.getString(inIndex)
18         END IF
19     END IF
20 LOOP
21 loID = oForm.getLong(oForm.findColumn("ID"))
22 Textfelder_fuellen_ID = loID
23 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*

```

1 SUB Tabellen_fuellen(oNewDoc AS OBJECT, stSql AS STRING)
2   DIM oDatenquelle AS OBJECT
3   DIM oVerbindung AS OBJECT
4   DIM oSQL_Anweisung AS OBJECT
5   DIM oAbfrageergebnis AS OBJECT
6   DIM oTabelle AS OBJECT
7   DIM oTabellen AS OBJECT
8   DIM oRows AS OBJECT
9   DIM i AS INTEGER
10  oTabellen = oNewDoc.getTextTables
11  oTabelle = oTabellen.getByname("Rechnungsinhalt")
12  oDatenquelle = ThisComponent.Parent.CurrentController
13  If NOT (oDatenquelle.isConnected()) THEN
14    oDatenquelle.connect()
15  END IF
16  oVerbindung = oDatenquelle.ActiveConnection()
17  oSQL_Anweisung = oVerbindung.createStatement()
18  oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
19  WHILE oAbfrageergebnis.next
20    oTabelle.getCellByPosition(0,i).setValue(oAbfrageergebnis.getInt(1))
21    oTabelle.getCellByPosition(1,i).setString(oAbfrageergebnis.getString(2))
22    oTabelle.getCellByPosition(2,i).setValue(oAbfrageergebnis.getDouble(3))
23    oTabelle.getCellByPosition(3,i).setValue(oAbfrageergebnis.getDouble(4))
24    oRows = oTabelle.getrows()
25    oRows.insertByIndex(oRows.getCount(),1)
26    i = i + 1
27  WEND
28  oRows.removeByIndex(oRows.getCount()-1,1)
29 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: <i>Rechnung_Textfelder</i>

Benötigt
Tabelle: <i>Verkauf, Waren</i>
Makro: <i>Textfelder_Fuellen_ID, Tabellen_fuellen</i>
Vorlage: <i>Beispiel_Rechnung_Textfelder.ott</i>

```
1 SUB Rechnungsinhalt_zusammenstellen_Tabelle(oEvent AS OBJECT)
2   DIM oForm AS OBJECT
3   DIM oDB AS OBJECT
4   DIM oNewDoc AS OBJECT
5   DIM stDir AS STRING
6   DIM stSql AS STRING
7   oForm = oEvent.Source.Model.Parent
8   oForm.reload()
9   oForm.last()
10  oDB = ThisComponent.Parent
11  stDir = Left(oDB.Location, Len(oDB.Location)-Len(oDB.Title))
12  stDir = stDir & "Beispiel_Rechnung_Textfelder.ott"
13  DIM args(0) AS NEW com.sun.star.beans.PropertyValue
14  args(0).Name = "AsTemplate"
15  args(0).Value = True
16  oNewDoc = StarDesktop.loadComponentFromURL(stDir, "_blank", 0, args)
17  loID = Textfelder_fuellen_ID(oNewDoc, oForm)
18  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+"
19  Tabellen_fuellen(oNewDoc, stSql)
20 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*

```
1 SUB Tabellen_fuellen_Uebertrag(oNewDoc AS OBJECT, stSql AS STRING)
2   DIM oTabellen AS OBJECT
3   DIM oTabelle AS OBJECT
4   DIM oDatenquelle AS OBJECT
5   DIM oVerbindung AS OBJECT
6   DIM oSQL_Anweisung AS OBJECT
7   DIM oAbfrageergebnis AS OBJECT
8   DIM oCursor AS OBJECT
9   DIM oTxtCursor AS OBJECT
10  DIM oRows AS OBJECT
11  DIM i AS INTEGER
12  DIM ink AS INTEGER
13  DIM inStartSeite AS INTEGER
14  DIM inFolgezeilen AS INTEGER
15  DIM doUebertrag AS DOUBLE
16  oTabellen = oNewDoc.getTextTables
17  oTabelle = oTabellen.getByNome("Rechnungsinhalt")
18  oDatenquelle = ThisComponent.Parent.CurrentController
19  If NOT (oDatenquelle.isConnected()) THEN
20    oDatenquelle.connect()
21  END IF
22  oVerbindung = oDatenquelle.ActiveConnection()
23  oSQL_Anweisung = oVerbindung.createStatement()
24  oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
25  inStartSeite = 1
26  inFolgezeilen = 3
27  oCursor = oNewDoc.CurrentController.ViewCursor
28  WHILE oAbfrageergebnis.next
29    doAnzahlPreis = oAbfrageergebnis.getDouble(4)
30    oTabelle.getCellByPosition(0,i).setValue(oAbfrageergebnis.getInt(1))
31    oTabelle.getCellByPosition(1,i).setString(oAbfrageergebnis.getString(2))
32    oTabelle.getCellByPosition(2,i).setValue(oAbfrageergebnis.getDouble(3))
33    oTabelle.getCellByPosition(3,i).setValue(doAnzahlPreis)
34    doUebertrag = doUebertrag + doAnzahlPreis
35    oRows = oTabelle.getrows()
36    oRows.insertByIndex(oRows.getCount(),1)
37    oCursor.JumpToLastPage()
38    oCursor.JumpToEndOfPage()
```

```

39     oCursor.goUp(inFolgezeilen, False)
40     IF oCursor.Page > inStartSeite THEN
41         oCursor.goUp(1, False)
42         FOR ink = 1 TO 2
43             i = i + 1
44             oRows = oTabelle.getRows()
45             oRows.insertByIndex(oRows.getCount()-2, 1)
46             oTabelle.getCellByPosition(2, i-1).setString("Übertrag:")
47             oTxtCursor = oTabelle.getCellByPosition(2, i-1)
48                 .createTextCursorByRange(oTabelle.getCellByPosition(2, i-1).text)
49             oTxtCursor.CharPosture = com.sun.star.awt.FontSlant.ITALIC
50             oTabelle.getCellByPosition(3, i-1).setValue(doUebertrag - doAnzahlPreis)
51             oTxtCursor = oTabelle.getCellByPosition(3, i-1)
52                 .createTextCursorByRange(oTabelle.getCellByPosition(3, i-1).text)
53             oTxtCursor.CharPosture = com.sun.star.awt.FontSlant.ITALIC
54         NEXT ink
55     oCursor.gotoRange(oTxtCursor, false)
56     Wait(100)
57     DO WHILE oCursor.Page = inStartSeite
58         oRows = oTabelle.getRows()
59         oRows.insertByIndex(oRows.getCount()-3, 1)
60         oCursor.gotoRange(oTxtCursor, false)
61         i = i + 1
62     LOOP
63     inStartSeite = inStartSeite + 1
64 END IF
65 i = i + 1
66 WEND
67 oRows.removeByIndex(oRows.getCount()-1, 1)
68 oCursor.JumpToLastPage()
69 oCursor.JumpToEndOfPage()
70 END SUB

```

Der grundlegende Aufbau dieser Prozedur ist wie der Aufbau der Prozedur «Tabellen_fuellen». Auch hier 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 Übertrags 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 Übertrags 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: <i>Rechnung_Textfelder_Uebertrag</i>

Benötigt
Tabelle: <i>Verkauf, Waren</i>
Makro: <i>Textfelder_Fuellen_ID, Tabellen_fuellen_Uebertrag</i>
Vorlage: <i>Beispiel_Rechnung_Textfelder.ott</i>

```

1 SUB Rechnungsinhalt_zusammenstellen_Tabelle_Uebertrag(oEvent AS OBJECT)
2   DIM oForm AS OBJECT
3   DIM oDB AS OBJECT
4   DIM oNewDoc AS OBJECT
5   DIM stDir AS STRING
6   DIM stSql AS STRING
7   DIM loID AS LONG
8   oForm = oEvent.Source.Model.Parent
9   oForm.reload()
10  oForm.last()
11  oDB = ThisComponent.Parent
12  stDir = Left(oDB.Location, Len(oDB.Location)-Len(oDB.Title))
13  stDir = stDir & "Beispiel_Rechnung_Textfelder.ott"
14  DIM args(0) AS NEW com.sun.star.beans.PropertyValue
15  args(0).Name = "AsTemplate"
16  args(0).Value = True
17  oNewDoc = StarDesktop.loadComponentFromURL(stDir, "_blank", 0, args)
18  loID = Textfelder_fuellen_ID(oNewDoc, oForm)
19  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+"
20  Tabellen_fuellen_Uebertrag(oNewDoc, stSql)
21 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: <i>Direktdruck_Start</i>

Benötigt
Tabelle: <i>Verkauf, Waren</i>
Makro: <i>Textfelder_Fuellen_ID, Tabellen_fuellen_Uebertrag, PDF_Export</i>
Vorlage: <i>Beispiel_Rechnung_Textfelder.ott</i>

```

1 SUB Rechnungsinhalt_zusammenstellen_Tabelle_Uebertrag_Direktdruck(oEvent AS OBJECT)
2   DIM oForm AS OBJECT
3   DIM oDB AS OBJECT
4   DIM oNewDoc AS OBJECT
5   DIM stDir AS STRING
6   DIM stPrintDir AS STRING
7   DIM stSql AS STRING
8   DIM loID AS LONG
9   oForm = oEvent.Source.Model.Parent
10  oForm.reload()
11  oForm.last()
12  oDB = ThisComponent.Parent
13  stDir = Left(oDB.Location, Len(oDB.Location)-Len(oDB.Title))
14  stPrintDir = stDir
15  stDir = stDir & "Beispiel_Rechnung_Textfelder.ott"
16  DIM args(1) AS NEW com.sun.star.beans.PropertyValue
17  args(0).Name = "AsTemplate"
18  args(0).Value = True
19  args(1).Name = "Hidden"
20  args(1).Value = True
21  oNewDoc = StarDesktop.loadComponentFromURL(stDir, "_blank", 0, args)
22  loID = Textfelder_fuellen_ID(oNewDoc, oForm)
23  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"""
24  Tabellen_fuellen_Uebertrag(oNewDoc, stSql)
25  PDF_Export(oNewDoc, loID, stPrintDir)
26 END SUB

```

Diese Prozedur unterscheidet 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: <i>Rechnungsinhalt_zusammenstellen_Tabelle_Uebertrag_Direktdruck</i>
Benötigt
Tabelle: <i>Rechnung</i>

```

1 SUB PDF_Export(oNewDoc AS OBJECT, loID AS LONG, stPrintDir AS STRING)
2   DIM oDatenquelle AS OBJECT
3   DIM oVerbindung AS OBJECT

```

```

4 DIM oSQL_Anweisung AS OBJECT
5 DIM stDatei AS STRING
6 DIM stSql AS STRING
7 stDatei = "RE_" & Right("0000000" & loID, 8) & ".pdf"
8 stPrintDir = stPrintdir & stDatei
9 DIM arg(0) AS NEW com.sun.star.beans.PropertyValue
10 arg(0).name = "FilterName"
11 arg(0).value = "writer_pdf_Export"
12 oNewDoc.storeToURL(stPrintDir, arg())
13 oNewDoc.close(true)
14 oDatenquelle = thisDatabaseDocument.CurrentController
15 IF NOT (oDatenquelle.isConnected()) THEN oDatenquelle.connect()
16 oVerbindung = oDatenquelle.ActiveConnection()
17 oSQL_Anweisung = oVerbindung.createStatement()
18 stSql = "UPDATE ""Rechnung"" SET ""Druckdatum"" = CURRENT_DATE WHERE ""ID"" =
      "+loID
19 oSQL_Anweisung.executeUpdate(stSql)
20 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*

```

1 SUB Direktdruck_Start(oEvent AS OBJECT)
2 DIM oMainForm AS OBJECT
3 DIM stSerie AS STRING
4 DIM stIDField AS STRING
5 DIM stID AS STRING
6 stSerie = oEvent.Source.Model.Tag
7 oMainForm = oEvent.Source.Model.Parent.Parent
8 stIDField = oMainForm.findColumn("ID")
9 stID = oMainForm.getString(stIDField)
10 IF stSerie = "Seriendruck" THEN
11     DO WHILE stID <> ""
12         Rechnungsinhalt_zusammenstellen_Tabelle_Uebertrag_Direktdruck
13         oMainForm.reload()
14         stID = oMainForm.getString(stIDField)
15     LOOP
16 ELSE
17     IF stID <> "" THEN
18         Rechnungsinhalt_zusammenstellen_Tabelle_Uebertrag_Direktdruck
19         oMainForm.reload()
20     END IF
21 END IF
22 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

Neuer Buchungstermin

Haus:

Mieter:

Startdatum:

Enddatum:

Löschen eines Buchungstermins

	Haus	Mieter	Startdatum	Enddatum
▶	Haus 1	Maier	27.06.20	04.07.20
	Haus 1	Dompapst	11.06.20	25.06.20
	Haus 2	Appelkorn	11.06.20	25.06.20
	Haus 3	Bunner	14.06.20	26.06.20

Datensatz 1 von 4

BeginnDatum: EndeDatum:

	Tag	Datum	Haus1	Haus2	Haus3	Haus4	Haus5	Haus6	Haus7	H.
▶	Do.	11.06.20	Dompapst	Appelkorn						
	Fr.	12.06.20	Dompapst	Appelkorn						
	Sa.	13.06.20	Dompapst	Appelkorn						
	So.	14.06.20	Dompapst	Appelkorn	Bunner					
	Mo.	15.06.20	Dompapst	Appelkorn	Bunner					
	Di.	16.06.20	Dompapst	Appelkorn	Bunner					
	Mi.	17.06.20	Dompapst	Appelkorn	Bunner					
	Do.	18.06.20	Dompapst	Appelkorn	Bunner					

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? 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 Lösungsvorschlag⁵.

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: <i>Belegung, Bericht</i>
Formular, Ansicht, Bericht, Makro: keine direkt

⁵ Beispieldatenbank Beispiel_Terminuebersicht_Ferienhausvermietung.odt

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

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: <i>Bericht</i>
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 Autp-Wert-Feld für den Primärschlüssel vorgesehen.

Reservierung

Datenziel
Formular: <i>Haus</i>
Abfrage: <i>Belegung, Bericht</i>
Makro: <i>Startdatum, Enddatum</i>
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.

Feldname	Feldtyp	Beschreibung
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:	<i>Haus</i>
Ansicht:	<i>Ansicht_Datum</i>
Abfrage:	<i>Belegung</i>
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:	<i>Ansicht_Datum</i>
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

Feldname	Feldtyp	Beschreibung
		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: <i>NrBis31, Filter</i>

Datenziel
Ansicht: <i>Ansicht_Datum_lfdNr</i>
Abfrage: <i>Belegung</i>
Makro: <i>Enddatum</i>

```

23 SELECT DISTINCT
24 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"
25 FROM "NrBis31" AS "D", "NrBis31" AS "M", "NrBis31" AS "Y"
26 WHERE "Y"."Nr" <= 3
27 AND "M"."Nr" <= 12
28 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: <i>Ansicht_Datum</i>
Abfrage: <i>Bericht</i>
Makro: <i>Startdatum</i>

```

1 SELECT "a"."Datum",
2     (SELECT COUNT(*) FROM "Ansicht_Datum" WHERE "Datum" <= "a"."Datum") AS "lfdNr"
3 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:

```

1 SELECT CAST( ( ( SELECT EXTRACT( YEAR FROM COALESCE ( "BeginnDatum", CURRENT_DATE ) )
2     FROM "Filter" WHERE "ID" = TRUE ) - 2 ) || '-12-31' AS DATE ) + "Nr" "Datum",
3 "Nr" AS "lfdNr"
4 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: <i>Mieter, Reservierung, Filter</i>
Ansicht: <i>Ansicht_Datum</i>

Datenziel
Formular: <i>Haus</i>

```
1 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",
2 "a"."Datum",
3 ( 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",
4 ( 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",
5 ...
6 ... AS "Haus8"
7 FROM "Ansicht_Datum" AS "a"
8 WHERE "a"."Datum" BETWEEN
9 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.

```
1 SELECT CASE
2 WHEN EXTRACT( WEEKDAY FROM "a"."Datum" ) = 0 THEN '■' || 'So.'
3 WHEN EXTRACT( WEEKDAY FROM "a"."Datum" ) = 6 THEN '■' || 'Sa.'
4 WHEN EXTRACT( WEEKDAY FROM "a"."Datum" ) = 1 THEN '■' || 'Mo.'
5 ...
6 END AS "Wochentag", "a"."Datum",
7 ( 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",
8 ( 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",
9 ...
10 ... "Haus8"
11 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 Makroereigenschaften 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: <i>Mieter, Reservierung, Haus</i>
Ansicht: <i>Ansicht_Datum_lfdNr</i>

Datenziel
Bericht: <i>Hausbelegung</i>

```

1 SELECT "Haus"."Haus", "Mieter"."Name",
2       "a"."Startdatum", "a"."Enddatum",
3       DATEDIFF( 'dd', "a"."Startdatum", "a"."Enddatum" ) AS "Tage",
4       ( SELECT "Datum" FROM "Ansicht_Datum_lfdNr" WHERE "lfdNr" =
          ( SELECT "lfdNr" FROM "Ansicht_Datum_lfdNr" WHERE "Datum" =
            "a"."Startdatum" ) - 30 )
          AS "AnzahlungBis"
5 FROM "Reservierung" AS "a", "Haus", "Mieter"
6 WHERE "a"."Haus_ID" = "Haus"."ID"
7       AND "a"."Mieter_ID" = "Mieter"."ID"
8 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 Datensatzes 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:

```
1 SELECT "Haus"."Haus", "Mieter"."Name",
2       "a"."Startdatum", "a"."Enddatum",
3       "a"."Enddatum" - "a"."Startdatum" "Tage",
4       "a"."Startdatum" - 30 "AnzahlungBis"
5 FROM "Reservierung" AS "a", "Haus", "Mieter"
6 WHERE "a"."Haus_ID" = "Haus"."ID"
7       AND "a"."Mieter_ID" = "Mieter"."ID"
8 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.

Neuer Buchungstermin

Haus:

Mieter: **1**

Startdatum: **1**

Enddatum:

Löschen eines Buchungstermins

Haus	Mieter	Startdatum	Enddatum
Haus1	Maier	27.06.20	04.07.20
Haus1	Dompapst	11.06.20	25.06.20
Haus2	Appelkorn	11.06.20	25.06.20
Haus3	Bunner	14.06.20	26.06.20

Datensatz 1 von 4

BeginnDatum: 01.01.20 **2** EndeDatum: 31.12.20

Tag	Datum	Haus1	Haus2	Haus3	Haus4	Haus5	Haus6	Haus7	Haus8
Do.	11.06.20	Dompapst	Appelkorn						
Fr.	12.06.20	Dompapst	Appelkorn						
Sa.	13.06.20	Dompapst	Appelkorn						
So.	14.06.20	Dompapst	Appelkorn	Bunner					
Mb.	15.06.20	Dompapst	Appelkorn	Bunner					
Di.	16.06.20	Dompapst	Appelkorn	Bunner					
Mi.	17.06.20	Dompapst	Appelkorn	Bunner					
Do.	18.06.20	Dompapst	Appelkorn	Bunner					
Fr.	19.06.20	Dompapst	Appelkorn	Bunner					
Sa.	20.06.20	Dompapst	Appelkorn	Bunner					
So.	21.06.20	Dompapst	Appelkorn	Bunner					
Mb.	22.06.20	Dompapst	Appelkorn	Bunner					
Di.	23.06.20	Dompapst	Appelkorn	Bunner					
Mi.	24.06.20	Dompapst	Appelkorn	Bunner					
Do.	25.06.20	Dompapst	Appelkorn	Bunner					
Fr.	26.06.20			Bunner					
Sa.	27.06.20	Maier							
So.	28.06.20	Maier							
Mb.	29.06.20	Maier							
Di.	30.06.20	Maier							
Mi.	01.07.20	Maier							
Do.	02.07.20	Maier							
Fr.	03.07.20	Maier							
Sa.	04.07.20	Maier							
So.	05.07.20								
Mb.	06.07.20								
Di.	07.07.20								

Datensatz 163 von 366 (1)

- 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. <i>DatumsAenderung</i>
---	---------------------------------	---------------------------------

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*

1 SUB Startdatum(oEvent AS OBJECT)

```

2   DIM oFeldStart AS OBJECT
3   DIM oForm AS OBJECT
4   DIM oFeld AS OBJECT
5   DIM inID AS INTEGER
6   DIM stSql(0) AS STRING
7   DIM stTeilsql AS STRING
8   oFeldStart = oEvent.Source.Model
9   oForm = oFeldStart.Parent
10  oFeld = oForm.getByname("Startdatum")
11  inID = oFeldStart.CurrentValue
12  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 "
13  stSql(0) = "SELECT ""Datum"" FROM "+stTeilsql+" AND ""lfdNr""+1 IN
      (SELECT ""lfdNr"" FROM "+stTeilsql+)"
14  oFeld.ListSource = stSql
15  oFeld.refresh
16  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.

Für FIREBIRD muss hier der Begriff BETWEEN umgangen werden, da dadurch der Kontakt zur Datenbank unterbrochen wird:

```

12 ""a"". ""Datum"" >= ""Startdatum"" AND ""a"". ""Datum"" <= ""Enddatum""

```

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 Eddatum 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: <i>Haus</i>
Benötigt
Tabelle: <i>Reservierung</i>
Ansicht: <i>Ansicht_Datum</i>
Makro: <i>DatumsAenderung</i>

```
1 SUB Enddatum(oEvent AS OBJECT)
2   DIM oDatField AS OBJECT
3   DIM oForm AS OBJECT
4   DIM oFeld AS OBJECT
5   DIM inID AS INTEGER
6   DIM stSql(0) AS STRING
7   DIM stTeilsql AS STRING
8   DIM stDatum AS STRING
9   oDatField = oEvent.Source.Model
10  stDatum = oDatField.CurrentValue.Year & "-" &
      Right("0" & oDatField.CurrentValue.Month, 2) & "-" &
      Right("0" & oDatField.CurrentValue.Day, 2)
11  oForm = oDatField.Parent
12  oFeld = oForm.getByName("Enddatum")
13  inID = oForm.getByName("Haus_ID").CurrentValue
14  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+"')"
```

```
15  stSql(0) = "SELECT ""Datum"" FROM "+stTeilsql+" WHERE ""Datum"" <
      COALESCE((SELECT MIN("""Datum"")) FROM "+stTeilsql+"
      WHERE ""Haus"" = 'belegt'),(SELECT MAX (""Datum"")) FROM ""Ansicht_Datum""))"
```

```
16  oFeld.ListSource = stSql
17  oFeld.refresh
18  DatumsAenderung(oEvent)
19 END SUB
```

Das Enddatum wird in Abhängigkeit von dem Startdatum und dem Haus in dem Listefeld 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 Listefeld weitergegeben (Zeile 16), das Listefeld 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 Listefeld 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.

Auch hier: Für FIREBIRD muss hier der Begriff BETWEEN umgangen werden, da dadurch der Kontakt zur Datenbank unterbrochen wird:

```
14 ""a"."Datum" >= "Startdatum" AND ""a"."Datum" <= "Enddatum"
```

Außerdem existiert dort «Ansicht_Datum» nicht, so dass stattdessen «Ansicht_Datum_lfdNr» eingesetzt werden muss.

BuchungsAenderung

Aufruf aus
Formular: <i>Haus</i>
Makro: <i>NeueBuchung</i>

```
1 SUB BuchungsAenderung
2   DIM oDoc AS OBJECT
3   DIM oDrawpage AS OBJECT
4   DIM oForm AS OBJECT
5   DIM iRow AS INTEGER
6   oDoc = thisComponent
7   oDrawpage = oDoc.drawpage
8   oForm = oDrawpage.forms.getByName("Uebersicht")
9   iRow = oForm.getRow
10  oForm.reload
11  oForm.last
12  oForm.absolute(iRow)
13 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 Tabellekontrollfeldes genau stimmig gehalten wird.

NeueBuchung

Aufruf aus
Formular: <i>Haus</i>

Benötigt
Makro: <i>BuchungsAenderung</i>

```
1 SUB NeueBuchung
```

```

2   BuchungsAenderung
3   DIM oDoc AS OBJECT
4   DIM oDrawpage AS OBJECT
5   DIM oForm AS OBJECT
6   oDoc = thisComponent
7   oDrawpage = oDoc.drawpage
8   oForm = oDrawpage.forms.getByName("BuchungLoeschen")
9   oForm.reload
10  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: <i>Haus</i>
Makro: <i>Enddatum</i>

```

1  SUB DatumsAenderung(oEvent AS OBJECT)
2  DIM oDoc AS OBJECT
3  DIM oDrawpage AS OBJECT
4  DIM oForm AS OBJECT
5  DIM oDatField AS OBJECT
6  DIM oConnection AS OBJECT
7  DIM oSQL_Statement AS OBJECT
8  DIM oResult AS OBJECT
9  DIM iRow AS INTEGER
10 DIM stDatum AS STRING
11 DIM stSql AS STRING
12 oDatField = oEvent.Source.Model
13 stDatum = oDatField.CurrentValue.Year & "-" &
    Right("0" & oDatField.CurrentValue.Month , 2) & "-" &
    Right("0" & oDatField.CurrentValue.Day , 2)
14 oDoc = thisComponent
15 oDrawpage = oDoc.drawpage
16 oForm = oDrawpage.forms.getByName("Uebersicht")
17 oConnection = oForm.activeConnection()
18 oSQL_Statement = oConnection.createStatement()
19 stSql = oForm.SingleSelectQueryComposer.Query
20 oResult = oSQL_Statement.executeQuery(stSql)
21 Do
22     oResult.next
23     iRow = iRow + 1
24     IF oResult.isLast THEN Exit Do
25 LOOP UNTIL stDatum = oResult.getString(2)
26 oForm.last
27 oForm.absolute(iRow)
28 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

fehl 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:

```
20     oResult = oSQL_Statement.executeQuery(stSql)
21     oForm.last
22     iLastRow = oForm.getRow
23     Do
24         oResult.next
25         iRow = iRow + 1
26         IF iRow = iLastRow THEN Exit Do
27     LOOP UNTIL stDatum = oResult.getString(2)
28     oForm.absolute(iRow)
29 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.

ID	<input type="text" value="1"/>		
Name	<input type="text" value="Robert Großkopf"/>	Web	<input type="text" value="robert.familiegrosskopf.de"/>
E-Mail	<input type="text" value="robert@familiegrosskopf.de"/>		
Datensatz	<input type="text" value="1"/> von 4		
Betreff	<input type="text" value="Mailtest aus Base heraus, mit CC und BCC"/>		
Inhalt	<input type="text" value="Hallo *,
hier ein kleiner Mailtest. Damit z.B. das Komma nicht den Inhalt abtrennt
muss im Makro eine Zeichenumwandlung stattfinden."/>		<input type="button" value="E-Mail an
Mailprogramm
übergeben"/>

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

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 Standard-einstellung 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

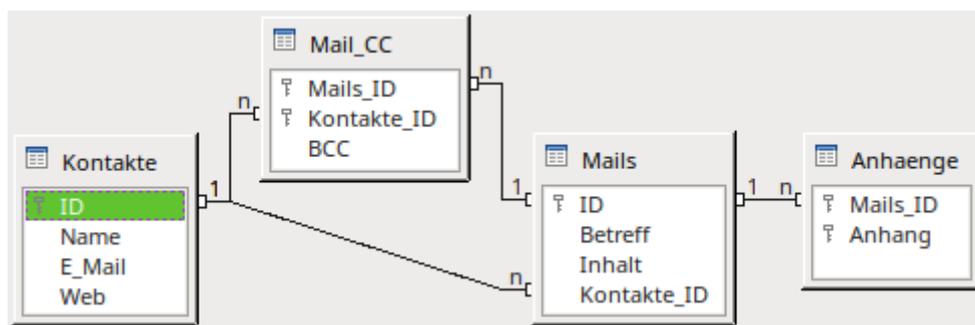
Feldname	Feldtyp	Beschreibung
Mails_ID	Integer	Die Mails_ID wird als Fremdschlüssel aus der Tabelle "Mails" bezogen. Dadurch können mehrere Personen eine Mailkopie erhalten.
Kontakte_ID	Integer	Die Kontakte_ID dient dazu, die Mailadresse von anderen Personen aus der Tabelle "Kontakte" zu ermitteln. Mails_ID 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 Primärschlüssel 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

ID

Name Web

E-Mail **1**

Datensatz von 4

Betreff

Inhalt **1.1**

Hallo *,

hier ein kleiner Mailtest. Damit z.B. das Komma nicht den Inhalt abtrennt muss im Makro eine Zeichenumwandlung stattfinden.

Gruß

Robert

E-Mail an Mailprogramm übergeben

Kopie an:		Blindkopie?
<input type="checkbox"/>	elvira@localhost	<input type="checkbox"/>
<input type="checkbox"/>	kurt@baggerloch.com	<input type="checkbox"/>
<input checked="" type="checkbox"/>	pippi@vimmerby.se	<input checked="" type="checkbox"/>

Datensatz von 3

1	Formular (Tabelle: <i>Kontakte</i>)	1.1	Unterformular (Tabelle: <i>Mails</i>)	1.1.1	FormMailsCC (Tabelle: <i>Mail_CC</i>)
----------	--------------------------------------	------------	--	--------------	--

Makros in Feldeigenschaften		
1	Feld «Web» (Maus innerhalb)	Module1. <i>Mauszeiger</i>
1	Feld «Web» (Maustaste gedrückt)	Module1. <i>Website_Aufruf</i>
1	Feld «E-Mail» (Maus innerhalb)	Module1. <i>Mauszeiger</i>
1	Feld «E-Mail» (Maustaste gedrückt)	Module1. <i>Website_Aufruf</i>
1.1	Button <i>E-Mail an Mailprogramm übergeben</i> (Aktion ausführen)	Module1. <i>Mail_Aufruf</i>

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

ID

Name Web

E-Mail **1**

Datensatz von 4

Betreff

Inhalt **1.1**

Hallo *,

hier ein kleiner Mailtest. Damit z.B. das Komma nicht den Inhalt abtrennt muss im Makro eine Zeichenumwandlung stattfinden.

Gruß

Robert

E-Mail an Mailprogramm übergeben

Anhänge

neuer Anhang: 

	Kopie an:	Blindkopie?
	<input type="text" value="elvira@localhost"/>	<input type="checkbox"/>
	<input type="text" value="kurt@baggerloch.com"/>	<input type="checkbox"/>
	<input type="text" value="pippi@vimmerby.se"/>	<input checked="" type="checkbox"/>

1.1.1

	Anhänge
	../Bilder/document...as.
	../Bilder/Zeitmessung/Zeit...
	../Bilder/Zeitmessung/Zeit...

1.1.2

1	Formular (Tabelle: <i>Kontakte</i>)	1.1	Unterformular (Tabelle: <i>Mails</i>)	1.1.1	FormMailsCC (Tabelle: <i>Mail_CC</i>)
				1.1.2	FormAnhaenge (Tabelle: <i>Anhaenge</i>)

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 <i>E-Mail an Mailprogramm übergeben</i> (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 Dateiauswahlfeld 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*

```
30 SUB Mauszeiger(oEvent AS OBJECT)
31   DIM oPointer AS OBJECT
32   oPointer = createUnoService("com.sun.star.awt.Pointer")
33   oPointer.setType(28)
34   oEvent.Source.Peer.SetPointer(oPointer)
35 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 Mauszeiger neu eingestellt (Zeile 5).

FormSave

Aufruf aus

Makro: *Mail_Aufruf, Attachment, Mail_Attachment*

```
1 SUB FormSave(oForm AS OBJECT)
2   IF oForm.IsModified THEN
3     IF oForm.isNew THEN
4       oForm.InsertRow
5     ELSE
6       oForm.UpdateRow
```

```

7     END IF
8     END IF
9 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 wiederholt hätte geschrieben werden müssen.

Website_Aufruf

Aufruf aus

Formular: *Kontakte_Mail_Web, Mail_mit_Anhang*

```

1 SUB Website_Aufruf(oEvent AS OBJECT)
2   DIM oFeld AS OBJECT
3   DIM oShell AS OBJECT
4   DIM stFeld AS STRING
5   oFeld = oEvent.Source.Model
6   stFeld = oFeld.Text
7   IF stFeld = "" THEN
8     EXIT SUB
9   END IF
10  IF InStr(stFeld, "@") THEN
11    stFeld = "mailto:"+stFeld
12  ELSEIF InStr(stFeld, "http://") OR InStr(stFeld, "https://") THEN
13    stFeld = convertToUrl(stFeld)
14  ELSE
15    stFeld = "http://" + stFeld
16    stFeld = convertToUrl(stFeld)
17  END IF
18  oShell = createUnoService("com.sun.star.system.SystemShellExecute")
19  oShell.execute(stFeld, , 0)
20 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: <i>Kontakte_Mail_Web, Mail_mit_Anhang</i>

Benötigt
Tabelle: <i>Kontakte, Mail_CC</i>
Makro: <i>FormSave</i>

```

1 SUB Mail_Aufruf
2   DIM oDoc AS OBJECT
3   DIM oDrawpage AS OBJECT
4   DIM oForm AS OBJECT
5   DIM oFeld1 AS OBJECT
6   DIM oFeld2 AS OBJECT
7   DIM oFeld3 AS OBJECT
8   DIM oFeld4 AS OBJECT
9   DIM oShell AS OBJECT
10  DIM oColumns AS OBJECT
11  DIM oDatenquelle AS OBJECT
12  DIM oVerbindung AS OBJECT
13  DIM oSQL_Anweisung AS OBJECT
14  DIM oAbfrageergebnis AS OBJECT
15  DIM inIndex AS INTEGER
16  DIM i AS INTEGER
17  DIM loMails_ID AS LONG
18  DIM stFeld1 AS STRING
19  DIM stFeld2 AS STRING
20  DIM stFeld3 AS STRING
21  DIM stFeld4 AS STRING
22  DIM stSql AS STRING
23  DIM stMail_CC AS STRING
24  DIM stMail_BCC AS STRING
25  DIM st_CC AS STRING
26  DIM st_BCC AS STRING
27  DIM arSo_Zeichen()
28  DIM arSo_Ersatz()
29  oDoc = thisComponent
30  oDrawpage = oDoc.Drawpage
31  oForm = oDrawpage.Forms.getByName("Formular")
32  oFeld1 = oForm.getByName("E-Mail")
33  oSubForm = oForm.getByName("UnterFormular")
34  oFeld2 = oSubForm.getByName("Betreff")
35  oFeld3 = oSubForm.getByName("Inhalt")
36  stFeld1 = oFeld1.Text
37  IF stFeld1 = "" THEN
38      MsgBox "Keine Mailadresse vorhanden." & CHR(13) &
          "Das Mailprogramm wird nicht aufgerufen" , 48, "Mail senden"
39      EXIT SUB
40  END IF
41  FormSave(oSubForm)
42  oColumns = oSubForm.Columns
43  inIndex = oSubForm.findColumn("ID")
44  loMails_ID = oSubForm.getLong(inIndex)
45  stFeld2 = Mid(ConvertToUrl(oFeld2.Text),9)
46  stFeld3 = Mid(ConvertToUrl(oFeld3.Text),9)
47  arSo_Zeichen = Array("\", "^", "~", "[", "]", " ", "&", "+", " ", ":", "=", "?", "@")

```

```

48 arSo_Ersatz = Array("5C", "5E", "7E", "5B", "5D", "20", "26", "2B", "2C", "3B",
    "3A", "3D", "3F", "40")
49 FOR i = LBound(arSo_Zeichen()) TO UBound(arSo_Zeichen())
50     stFeld3 = Join(Split(stFeld3, arSo_Zeichen(i)), "%" & arSo_Ersatz(i))
51 NEXT
52 stMail_CC = ""
53 stMail_BCC = ""
54 st_CC = ""
55 st_BCC = ""
56 oDatenquelle = ThisComponent.Parent.CurrentController
57 IF NOT (oDatenquelle.isConnected()) THEN
58     oDatenquelle.connect()
59 END IF
60 oVerbindung = oDatenquelle.ActiveConnection()
61 oSQL_Anweisung = oVerbindung.createStatement()
62 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 +'""
63 oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
64 WHILE oAbfrageergebnis.next
65     IF oAbfrageergebnis.getBoolean(2) = TRUE THEN
66         stMail_BCC = stMail_BCC + oAbfrageergebnis.getString(1) + ", "
67     ELSE
68         stMail_CC = stMail_CC + oAbfrageergebnis.getString(1) + ", "
69     END IF
70 WEND
71 IF stMail_BCC <> "" THEN
72     st_BCC = Left(stMail_BCC, Len(stMail_BCC)-1)
73     stMail_BCC = "&bcc="+st_BCC
74     st_BCC = ", bcc='"+st_BCC+"'"
75 END IF
76 IF stMail_CC <> "" THEN
77     st_CC = Left(stMail_CC, Len(stMail_CC)-1)
78     stMail_CC = "&cc="+st_CC
79     st_CC = ", cc='"+st_CC+"'"
80 END IF
81 oShell = createUnoService("com.sun.star.system.SystemShellExecute")
82 oShell.execute("mailto:"+stFeld1+"?subject="+stFeld2+"&body="+stFeld3
    +stMail_CC+stMail_BCC,,0)
83 ' Shell("thunderbird -compose ""to="+stFeld1+st_CC+st_BCC+", subject="+stFeld2+
    ", body="+stFeld3+", attachment='file:///home/user/Testdatei.pdf' """)
84 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 **ConvertToUrl** 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 weitergeleitet 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](#)

```
1 SUB Attachment(oEvent AS OBJECT)
2   DIM oForm AS OBJECT
```

```

3     oForm = oEvent.Source.Model.Parent.Parent
4     FormSave(oForm)
5 END SUB

```

Die Anhänge werden über ein grafisches Kontrollfeld eingefügt. Eine Änderung in dem zugehörigen 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 Elternelement 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*

```

1 SUB Mail_Attachment
2   DIM arAttach(0)
3   DIM arCC(0)
4   DIM arBCC(0)
5   DIM arURL_Start()
6   DIM oDoc AS OBJECT
7   DIM oDrawpage AS OBJECT
8   DIM oForm AS OBJECT
9   DIM oSubForm AS OBJECT
10  DIM oFeld1 AS OBJECT
11  DIM oFeld2 AS OBJECT
12  DIM oFeld3 AS OBJECT
13  DIM oColumns AS OBJECT
14  DIM oDatenquelle AS OBJECT
15  DIM oVerbindung AS OBJECT
16  DIM oSQL_Anweisung AS OBJECT
17  DIM oAbfrageergebnis AS OBJECT
18  DIM oMailProgram AS OBJECT
19  DIM oNewMessage AS OBJECT
20  DIM inIndex AS INTEGER
21  DIM i AS INTEGER
22  DIM k AS INTEGER
23  DIM loMails_ID AS LONG
24  DIM stFeld1 AS STRING
25  DIM stFeld2 AS STRING
26  DIM stFeld3 AS STRING
27  DIM stSql AS STRING
28  oDoc = thisComponent
29  oDrawpage = oDoc.Drawpage
30  oForm = oDrawpage.Forms.getByName("Formular")
31  oFeld1 = oForm.getByName("E_Mail")
32  oSubForm = oForm.getByName("UnterFormular")
33  oFeld2 = oSubForm.getByName("Betreff")
34  oFeld3 = oSubForm.getByName("Inhalt")
35  stFeld1 = oFeld1.Text
36  IF stFeld1 = "" THEN
37     msgbox "Keine Mailadresse vorhanden." & CHR(13) & "Das Mailprogramm wird nicht
        aufgerufen" , 48, "Mail senden"
38  EXIT SUB
39  END IF

```

```

40 FormSave(oForm)
41 stFeld2 = oFeld2.Text
42 stFeld3 = oFeld3.Text
43 oColumns = oSubForm.Columns
44 inIndex = oSubForm.findColumn("ID")
45 loMails_ID = oSubForm.getLong(inIndex)
46 oDatenquelle = ThisComponent.Parent.CurrentController
47 IF NOT (oDatenquelle.isConnected()) THEN
48     oDatenquelle.connect()
49 END IF
50 oVerbindung = oDatenquelle.ActiveConnection()
51 oSQL_Anweisung = oVerbindung.createStatement()
52 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 +'""
53 oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
54 i = 0
55 k = 0
56 WHILE oAbfrageergebnis.next
57     IF oAbfrageergebnis.getBoolean(2) = TRUE THEN
58         IF i > 0 THEN
59             ReDim Preserve arBCC(i)
60         END IF
61         arBCC(i) = oAbfrageergebnis.getString(1)
62         i = i + 1
63     ELSE
64         IF k > 0 THEN
65             ReDim Preserve arCC(k)
66         END IF
67         arCC(k) = oAbfrageergebnis.getString(1)
68         k = k + 1
69     END IF
70 WEND
71 stSql = "SELECT ""Anhang"" FROM ""Anhaenge"" WHERE ""Mails_ID"" = '"+ loMails_ID
        +""""
72 oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
73 arUrl_Start = split(oDoc.Parent.Url,oDoc.Parent.Title)
74 i = 0
75 WHILE oAbfrageergebnis.next
76     IF i > 0 THEN
77         ReDim Preserve arAttach(i)
78     END IF
79     arAttach(i) = ConvertToUrl(arUrl_Start(0) & oAbfrageergebnis.getString(1))
80     i = i + 1
81 WEND
82 IF GetGuiType() = 1 THEN
83     oMailer = createUnoService("com.sun.star.system.SimpleSystemMail")
        ' Sonst Linux/Mac
84 ELSE
85     oMailer = createUnoService("com.sun.star.system.SimpleCommandMail")
86 END IF
87 oMailProgram = oMailer.querySimpleMailClient()
88 oNewMessage = oMailProgram.createSimpleMailMessage()
89 oNewMessage.Recipient = stFeld1
90 oNewMessage.CcRecipient = arCC()
91 oNewMessage.BccRecipient = arBCC()
92 oNewMessage.Subject = stFeld2
93 oNewMessage.Body = stFeld3
94 oNewMessage.setAttachment(arAttach())
95 oMailProgram.sendSimpleMailMessage(oNewMessage, 0 )
96 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 Dimensionierung 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_sun_1_star_1_system_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_sun_1_star_1_system_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.

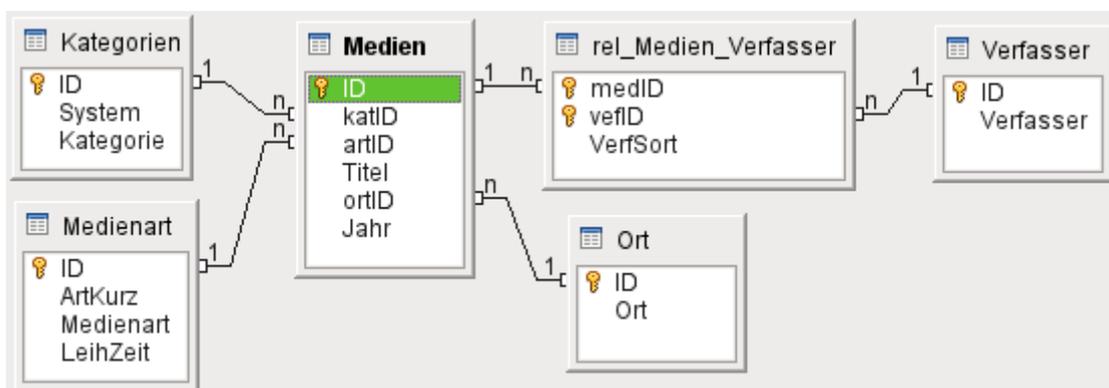
Die eingebaute Suchfunktion ist allerdings recht langsam, die Datenfilterung erfordert jeweils Einstellungen unterschiedlicher Werte und Zuordnungen. Die Datenfilterung ist sehr gut universell nutzbar, sofern sich sämtliche zu durchsuchenden Daten in einer Tabelle oder Abfrage befinden. Bei immer wiederkehrenden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

Als praktische Beispiel⁷ 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

⁷ Beispieldatenbank Beispiel_Suchen_Filtern.odb

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.

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: <i>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</i>
Abfrage: <i>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</i>
Makro: <i>Filter_Suche_hierarchisch_Start</i>
Ansicht, Bericht: keine

Feldname	Feldtyp	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_Fil-

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
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

```
97 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",
98 COALESCE("rel_Medien_Verfasser"."vefID", 0 ) AS "VerfasserID",
99 COALESCE("Medien"."katID", 0 ) AS "KategorieID",
100 COALESCE("Medien"."artID", 0 ) AS "MedienartID"
101 FROM "Medien"
102 LEFT JOIN "Kategorien" ON "Medien"."katID" = "Kategorien"."ID"
103 LEFT JOIN "Medienart" ON "Medien"."artID" = "Medienart"."ID"
104 LEFT JOIN "Ort" ON "Medien"."ortID" = "Ort"."ID"
105 LEFT JOIN "rel_Medien_Verfasser" ON "rel_Medien_Verfasser"."medID" = "Medien"."ID"
106 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

```

1 SELECT "ID", "System", "Kategorie", "Titel", "Jahr", "ArtKurz", "Medienart",
"LeihZeit", "Ort",
2 ( SELECT COUNT( "ID" ) FROM "Ansicht_Suche" WHERE "ID" = "a"."ID" AND "vefID" <=
"a"."vefID" ) AS "GruppenNr",
3 "Verfasser"
4 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

```

1 SELECT DISTINCT "ID", "System", "Kategorie", "Titel", "Jahr", "ArtKurz", "Medienart",
"LeihZeit", "Ort",
2 ( SELECT "Verfasser" FROM "Ansicht_Filter_Start" WHERE "ID" = "a"."ID" AND
"GruppenNr" = 1 ) ||

```

```

3 COALESCE( ';' || ( SELECT "Verfasser" FROM "Ansicht_Filter_Start" WHERE "ID" =
  "a"."ID" AND "GruppenNr" = 2 ), '' ) ||
4 COALESCE( ';' || ( SELECT "Verfasser" FROM "Ansicht_Filter_Start" WHERE "ID" =
  "a"."ID" AND "GruppenNr" = 3 ), '' ) ||
5 COALESCE( ';' || ( SELECT "Verfasser" FROM "Ansicht_Filter_Start" WHERE "ID" =
  "a"."ID" AND "GruppenNr" = 4 ), '' ) ||
6 COALESCE( ';' || ( SELECT "Verfasser" FROM "Ansicht_Filter_Start" WHERE "ID" =
  "a"."ID" AND "GruppenNr" = 5 ), '' ) ||
7 COALESCE( ';' || ( SELECT "Verfasser" FROM "Ansicht_Filter_Start" WHERE "ID" =
  "a"."ID" AND "GruppenNr" = 6 ), '' ) AS "Verfasser"
8 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

```

1 SELECT "ID", "Titel", "Jahr", "System", "Kategorie", "ArtKurz", "Medienart",
  "LeihZeit", "Ort",
2 LIST("Verfasser",';') AS "Verfasser"
3 FROM "Ansicht_Suche"
4 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: <i>Medien, Filter</i>
Datenziel
Formular: <i>Filter_Formular, Filter_Formular_Nullwerte, Makro_Filter_Formular, Filter_Formular_Listfeldbegrenzung</i>

```

1 SELECT "Medien".*,
2 COALESCE( "katID", 0 ) AS "kat"
3 FROM "Medien"
4 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:

```

1 SELECT *
2 FROM "Medien"
3 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

```

1 SELECT * FROM "Medien"
2 WHERE "ID" IN (
3 SELECT DISTINCT "ID" FROM "Ansicht_Suche" WHERE "VerfasserID" = COALESCE( ( SELECT
"Filter_Verfasser" FROM "Filter" WHERE "ID" = TRUE ), "VerfasserID" )
4 )

```

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

```

1 SELECT *
2 FROM "Medien"
3 WHERE "ID" IN (
4 SELECT DISTINCT "ID" FROM "Ansicht_Suche" WHERE
5 "VerfasserID" = COALESCE( ( SELECT "Filter_Verfasser" FROM "Filter" WHERE "ID" = TRUE
6 ), "VerfasserID" )
7 AND "MedienartID" = COALESCE( ( SELECT "Filter_Medienart" FROM "Filter" WHERE "ID" =
8 TRUE ), "MedienartID" )
9 AND "KategorieID" = COALESCE( ( SELECT "Filter_Kategorie" FROM "Filter" WHERE "ID" =
10 TRUE ), "KategorieID" )
11 )

```

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

```

1 SELECT *
2 FROM "Ansicht_Suche"
3 WHERE "ID" IN (
4 SELECT DISTINCT "ID" FROM "Ansicht_Suche" WHERE
5 "VerfasserID" = COALESCE( ( SELECT "Filter_Verfasser" FROM "Filter" WHERE "ID" = TRUE
6 ), "VerfasserID" )
7 AND "MedienartID" = COALESCE( ( SELECT "Filter_Medienart" FROM "Filter" WHERE "ID" =
8 TRUE ), "MedienartID" )
9 AND "KategorieID" = COALESCE( ( SELECT "Filter_Kategorie" FROM "Filter" WHERE "ID" =
10 TRUE ), "KategorieID" )
11 )

```

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 Listenfeldabfrage 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

```
1 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 ,

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

Der Tabelle "Filter" wird ein leeres Feld hinzugefügt. Dem Feld wird der Alias "Leer" zugewiesen. Dies ist notwendig 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

```
1 SELECT *  
2 FROM "Medien"  
3 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

```

1 SELECT *
2 FROM "Medien"
3 WHERE "ID" IN (
4 SELECT DISTINCT "ID" FROM "Ansicht_Suche"
5 WHERE LOWER ( "Titel" ) LIKE COALESCE( '%' || LOWER ( ( SELECT "Suche" FROM "Filter"
6 WHERE "ID" = TRUE ) ) || '%', LOWER ( "Titel" ) )
7 OR LOWER ( "Kategorie" ) LIKE '%' || LOWER ( ( SELECT "Suche" FROM "Filter" WHERE
8 "ID" = TRUE ) ) || '%'
9 OR LOWER ( "Medienart" ) LIKE '%' || LOWER ( ( SELECT "Suche" FROM "Filter" WHERE
10 "ID" = TRUE ) ) || '%'
11 OR LOWER ( "Ort" ) LIKE '%' || LOWER ( ( SELECT "Suche" FROM "Filter" WHERE "ID" =
12 TRUE ) ) || '%'
13 OR LOWER ( "Verfasser" ) LIKE '%' || LOWER ( ( SELECT "Suche" FROM "Filter" WHERE
14 "ID" = TRUE ) ) || '%'
15 )

```

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*

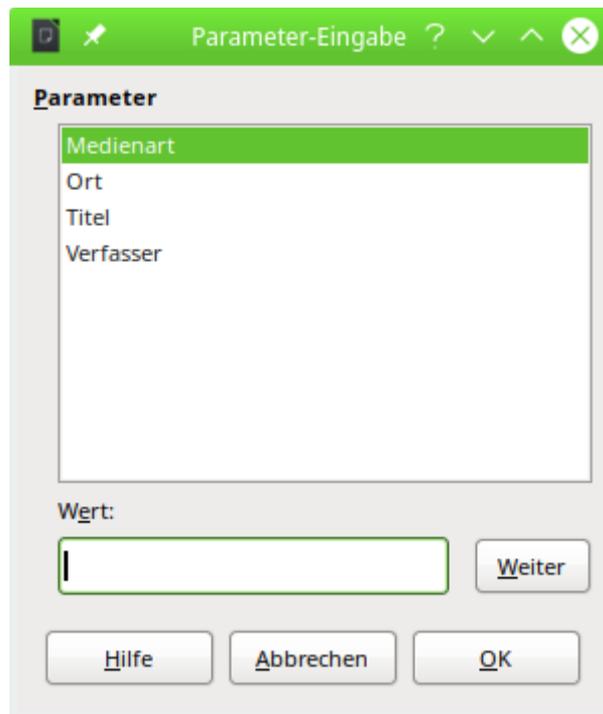
```
1 SELECT *
2 FROM "Medien"
3 WHERE "ID" IN (
4     SELECT "ID" FROM "Ansicht_Suche" WHERE
5         COALESCE( LOWER ( "Titel" ), '' ) LIKE '%' || COALESCE( LOWER
6             ( :Titel ), '' ) || '%' AND
7         COALESCE( LOWER ( "Medienart" ), '' ) LIKE '%' || COALESCE( LOWER
8             ( :Medienart ), '' ) || '%' AND
9         COALESCE( LOWER ( "Ort" ), '' ) LIKE '%' || COALESCE( LOWER
10            ( :Ort ), '' ) || '%' AND
11        COALESCE( LOWER ( "Verfasser" ), '' ) LIKE '%' || COALESCE( LOWER
12            ( :Verfasser ), '' ) || '%'
13 )
```

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 Textelemente 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 wird stattdessen ein leerer Text als Inhalt angenommen. So wird aus dem leeren Feld "Titel" gegebenenfalls ein leerer Text. Auch ein leerer 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 Elemente aus "Medien", die dem Suchergebnis entsprechen, werden angezeigt. Durch die Anzeige aller Elemente, 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: <i>Medien, Filter</i>
Ansicht: <i>Ansicht_Suche</i>

Datenziel
Formular: <i>Suche_Filter_Formular_Subformular_ohne_Makros, Suche_Filter_hierarchisch_mit_Makros, Suche_Filter_hierarchisch_allround_Datensaetze_mit_Makros</i>
Abfrage: <i>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</i>

```

1 SELECT *
2 FROM "Medien"
3 WHERE "ID" IN (
4 SELECT DISTINCT "ID" FROM "Ansicht_Suche"
5 WHERE "VerfasserID" = COALESCE( ( SELECT "Filter_Verfasser" FROM "Filter" WHERE "ID"
6 = TRUE ), "VerfasserID" )
7 AND "MedienartID" = COALESCE( ( SELECT "Filter_Medienart" FROM "Filter" WHERE "ID" =
8 = TRUE ), "MedienartID" )
9 AND "KategorieID" = COALESCE( ( SELECT "Filter_Kategorie" FROM "Filter" WHERE "ID" =
10 = TRUE ), "KategorieID" )
11 AND (

```

```

9 LOWER ( "Titel" ) LIKE COALESCE( '%' || LOWER ( ( SELECT "Suche" FROM "Filter" WHERE
  "ID" = TRUE ) ) || '%', LOWER ( "Titel" ) )
10 OR LOWER ( "Kategorie" ) LIKE '%' || LOWER ( ( SELECT "Suche" FROM "Filter" WHERE
  "ID" = TRUE ) ) || '%'
11 OR LOWER ( "Medienart" ) LIKE '%' || LOWER ( ( SELECT "Suche" FROM "Filter" WHERE
  "ID" = TRUE ) ) || '%'
12 OR LOWER ( "Ort" ) LIKE '%' || LOWER ( ( SELECT "Suche" FROM "Filter" WHERE "ID" =
  TRUE ) ) || '%'
13 OR LOWER ( "Verfasser" ) LIKE '%' || LOWER ( ( SELECT "Suche" FROM "Filter" WHERE
  "ID" = TRUE ) ) || '%'
14 )
15 )

```

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

```

1 SELECT *
2 FROM "Ansicht_Suche"
3 WHERE "ID" IN (
4 SELECT DISTINCT "ID" FROM "Ansicht_Suche"
5 WHERE "VerfasserID" = COALESCE( ( SELECT "Filter_Verfasser" FROM "Filter" WHERE "ID"
  = TRUE ), "VerfasserID" )
6 AND "MedienartID" = COALESCE( ( SELECT "Filter_Medienart" FROM "Filter" WHERE "ID" =
  TRUE ), "MedienartID" )
7 AND "KategorieID" = COALESCE( ( SELECT "Filter_Kategorie" FROM "Filter" WHERE "ID" =
  TRUE ), "KategorieID" )
8 AND (
9 LOWER ( "Titel" ) LIKE COALESCE( '%' || LOWER ( ( SELECT "Suche" FROM "Filter" WHERE
  "ID" = TRUE ) ) || '%', LOWER ( "Titel" ) )
10 OR LOWER ( "Kategorie" ) LIKE '%' || LOWER ( ( SELECT "Suche" FROM "Filter" WHERE
  "ID" = TRUE ) ) || '%'

```

```

11 OR LOWER ( "Medienart" ) LIKE '%' || LOWER ( ( SELECT "Suche" FROM "Filter" WHERE
    "ID" = TRUE ) ) || '%'
12 OR LOWER ( "Ort" ) LIKE '%' || LOWER ( ( SELECT "Suche" FROM "Filter" WHERE "ID" =
    TRUE ) ) || '%'
13 OR LOWER ( "Verfasser" ) LIKE '%' || LOWER ( ( SELECT "Suche" FROM "Filter" WHERE
    "ID" = TRUE ) ) || '%'
14 )
15 )

```

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

```

1 SELECT "Kat", "ID"
2 FROM (
3 SELECT '1' AS "C", 'ohne Eintrag' AS "Kat", 0 AS "ID" FROM "Kategorien"
4 UNION
5 SELECT "Kategorien"."System",
6     LEFT( "Kategorien"."System" || '          ', 7 )
7     || ' - ' ||
8     LEFT("Kategorien"."Kategorie",50) ||
9     CASE WHEN CHAR_LENGTH("Kategorien"."Kategorie") > 50 THEN '...' ELSE '' END
10    AS "Kat",
11    "Kategorien"."ID"
12    FROM "Kategorien", "Medien" WHERE "Kategorien"."ID" = "Medien"."katID"
13 UNION
14 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 Primärschlüsseintrag 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 Eintrag 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).

Listefeld_Kategorie_bedingt

Datenquelle
Tabelle: Kategorien
Abfrage: Filter_Form_Subform_3Filter

Datenziel
Formular: Filter_Formular_Subformular_bedingende_Filter , Filter_Formular_Subformular_bedingende_Filter_allround

```

1 SELECT "System" || ' - ' || "Kategorie" AS "Kat", "ID"
2 FROM "Kategorien"
3 WHERE "ID" IN
4 ( SELECT "katID" FROM "Filter_Form_Subform_3Filter" )
5 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.

Listefeld_Kategorie_bedingt_Datensaetze

Datenquelle
Tabelle: Kategorien

Abfrage: [Filter_Form_Subform_3Filter](#)

Datenziel

Formular: [Filter_Formular_Subformular_bedingende_Filter_Datensaetze](#)

```
1 SELECT "System" || ' - ' || LEFT("Kategorie",40) ||  
   CASE WHEN CHAR_LENGTH("Kategorie") > 40 THEN '...' ELSE '' END || ' → ' || (  
2 SELECT COUNT( "katID" ) FROM "Filter_Form_Subform_3Filter" WHERE "katID" = "a"."ID"  
3 ) || ' Treffer' AS "Kat", "ID"  
4 FROM "Kategorien" AS "a"  
5 WHERE "ID" IN  
6 ( SELECT "katID" FROM "Filter_Form_Subform_3Filter" )  
7 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](#)

```
1 SELECT "System" || ' - ' || "Kategorie" AS "Kat", "ID"  
2 FROM "Kategorien"  
3 WHERE "ID" IN  
4 ( SELECT "katID" FROM "Suche_Filter_Form_Subform_3Filter" )  
5 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](#)

```
1 SELECT "System" || ' - ' || "LEFT("Kategorie",40) ||  
   CASE WHEN CHAR_LENGTH("Kategorie") > 40 THEN '...' ELSE '' END || ' → ' || (  
2 SELECT COUNT( "katID" ) FROM "Suche_Filter_Form_Subform_3Filter" WHERE "katID" =  
   "a"."ID"  
3 ) || ' Treffer' AS "Kat", "ID"  
4 FROM "Kategorien" AS "a"  
5 WHERE "ID" IN  
6 ( SELECT "katID" FROM "Suche_Filter_Form_Subform_3Filter" )  
7 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](#)

```
1 SELECT "Medienart", "ID"  
2 FROM "Medienart"  
3 WHERE "ID" IN  
4 ( SELECT "artID" FROM "Filter_Form_Subform_3Filter" )  
5 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](#)

```
1 SELECT "Medienart" || ' → ' || (  
2 SELECT COUNT( "artID" ) FROM "Filter_Form_Subform_3Filter" WHERE "artID" = "a"."ID"  
3 ) || ' Treffer' AS "Med", "ID"  
4 FROM "Medienart" AS "a"  
5 WHERE "ID" IN ( SELECT "artID" FROM "Filter_Form_Subform_3Filter" )  
6 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

```

1 SELECT "Medienart", "ID"
2 FROM "Medienart"
3 WHERE "ID" IN
4 ( SELECT "artID" FROM "Suche_Filter_Form_Subform_3Filter" )
5 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

```

1 SELECT "Medienart" || ' → ' || (
2 SELECT COUNT( "artID" ) FROM "Suche_Filter_Form_Subform_3Filter" WHERE "artID" =
   "a"."ID"
3 ) || ' Treffer' AS "Med", "ID"
4 FROM "Medienart" AS "a" WHERE "ID" IN
5 ( SELECT "artID" FROM "Suche_Filter_Form_Subform_3Filter" )
6 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](#)

```
1 SELECT DISTINCT "Verfasser"."Verfasser", "Verfasser"."ID"
2 FROM "rel_Medien_Verfasser", "Verfasser"
3 WHERE "rel_Medien_Verfasser"."vefID" = "Verfasser"."ID"
4 AND "rel_Medien_Verfasser"."VerfSort" < 99
5 AND "rel_Medien_Verfasser"."medID" IN
6 ( SELECT "ID" FROM "Filter_Form_Subform_3Filter" )
7 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 Formulaierungen 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 Demonstariation 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.

Listefeld_Verfasser_bedingt_Datensaetze_langsam

Datenquelle

Ansicht: [Ansicht_Suche](#)

Abfrage: [Filter_Form_Subform_3Filter](#)

Datenziel

keins, da die Abfrage zu langsam läuft

```
1 SELECT DISTINCT "a"."Verfasser" || ' → ' || (
2 SELECT COUNT( "vefID" ) FROM "Ansicht_Suche" WHERE "vefID" = "a"."vefID" AND "ID" IN
3 ( SELECT "ID" FROM "Filter_Form_Subform_3Filter" )
4 ) || ' Treffer' AS "Ver", "a"."vefID"
5 FROM "Ansicht_Suche" AS "a"
6 WHERE "a"."VerfSort" < 99 AND "a"."ID" IN ( SELECT "ID" FROM
7 "Filter_Form_Subform_3Filter" )
8 ORDER BY "Ver" ASC
```

Diese Abfrage liefert die gleichen Ergebnisse wie [Listefeld_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.

Listefeld_Verfasser_bedingt_Datensaetze_optimiert

Datenquelle

Abfrage: [Filter_Ansicht](#)

Datenziel

Formular: [Filter_Formular_Subformular_bedingende_Filter_Datensaetze](#)

```
1 SELECT DISTINCT "Verfasser" || ' → ' || (  
2 SELECT COUNT( "vefID" ) FROM "Filter_Ansicht" WHERE "vefID" = "a"."vefID"  
3 ) || ' Treffer' AS "Ver", "vefID"  
4 FROM "Filter_Ansicht" AS "a"  
5 WHERE "VerfSort" < 99  
6 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](#)

```
1 SELECT DISTINCT "Verfasser"."Verfasser", "Verfasser"."ID"  
2 FROM "rel_Medien_Verfasser", "Verfasser"  
3 WHERE "rel_Medien_Verfasser"."vefID" = "Verfasser"."ID"  
4 AND "rel_Medien_Verfasser"."VerfSort" < 99  
5 AND "rel_Medien_Verfasser"."medID" IN  
6 ( SELECT "ID" FROM "Suche_Filter_Form_Subform_3Filter" )  
7 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](#)

```
1 SELECT DISTINCT "Verfasser" || ' → ' ||
```

```

2 ( SELECT COUNT( "vefID" ) FROM "Filter_Such_Ansicht" WHERE "vefID" = "a"."vefID" )
3 || ' Treffer' AS "Ver", "vefID"
4 FROM "Filter_Such_Ansicht" AS "a"
5 WHERE "VerfSort" < 99
6 ORDER BY "Ver" ASC

```

Im Unterschied zu *Listefeld_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: <i>Filter</i>)	
2	MainForm (Abfrage: <i>Filter_Form</i>)	2.1 SubForm (Tabelle: <i>rel_Medien_Verfasser</i>)

Listefeld «Filter Kategorie» (Tabelle: *Kategorien, Medien*)

```

1 SELECT DISTINCT LEFT( "Kategorien"."System" || ' ', 7 ) || ' - ' ||
   LEFT("Kategorien"."Kategorie", 50)|| CASE WHEN
   CHAR_LENGTH("Kategorien"."Kategorie") > 50 THEN '...' ELSE '' END AS "Kat",
2 "Kategorien"."ID"
3 FROM "Kategorien", "Medien"
4 WHERE "Kategorien"."ID" = "Medien"."katID"
5 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:

Property	Value
Art des Inhaltes.....	Tabelle
Inhalt.....	Filter
SQL-Befehl analysieren.....	Ja
Filter.....	("Filter"."ID" = TRUE)
Sortierung.....	
Daten hinzufügen.....	Nein
Daten ändern.....	Ja
Daten löschen.....	Nein
Nur Daten hinzufügen.....	Nein
Symbolleiste Navigation.....	Nein
Zyklus.....	Aktueller Datensatz

Formulareigenschaften des Formulars «Filter»

Filter_Formular_Nullwerte

Verfasser	VerfSort
Mevers Lexikonred.	1
Anger, Eberhard	2
(Hrsg., Red.)	99

1	Filter (Tabelle: <i>Filter</i>)	
2	MainForm (Abfrage: <i>Filter_Form</i>)	2.1 SubForm (Tabelle: <i>rel_Medien_Verfasser</i>)

Listefeld «Filter Kategorie» (Abfrage: *Listenfeld_Kategorie_ohne_Eintrag*)

Dieses Formular unterscheidet sich nur in Bezug auf das Listefeld «Filter Kategorie» vom *Filter_Formular*.

Filter_Formular_Subformular

Verfasser	VerfSort
Adriaensen-Busch, M.	1
Adriaensen, P.	2
Beulens, B.	3

1	Filter (Tabelle: <i>Filter</i>)		
2	MainForm (Abfrage: <i>Filter_Form_Subform</i>)	2.1	SubForm (Tabelle: <i>rel_Medien_Verfasser</i>)

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

The screenshot shows a filter form with the following fields and elements:

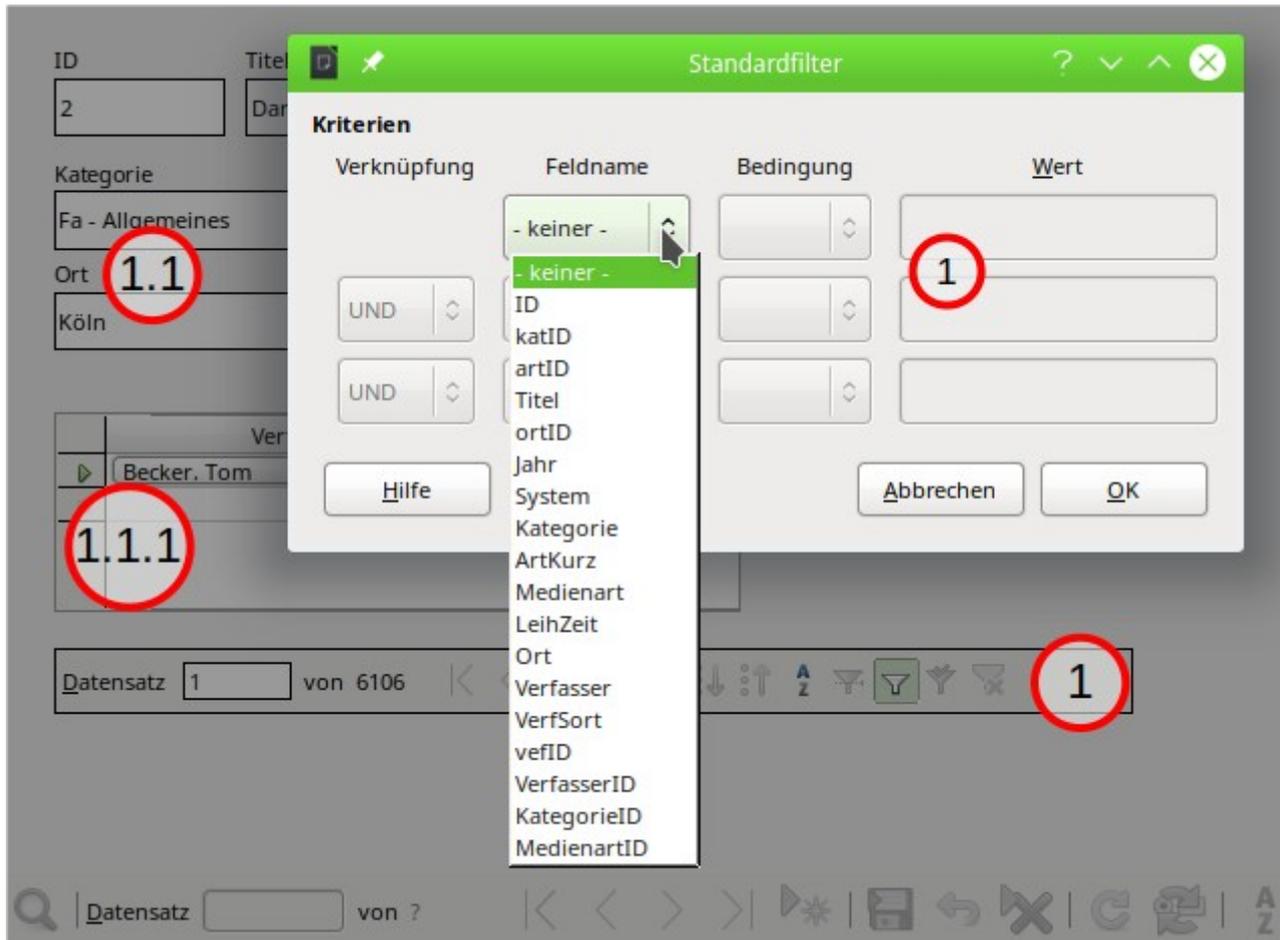
- Kategorie:** Empty dropdown.
- Verfasser:** "Lindgren, Astrid" (circled 1).
- Medienart:** "Video-DVD" (circled 2).
- ID:** "78".
- Tonband:** "Video".
- Verfasser (SubForm):** "Lindgren, Astrid" (circled 2.1).
- VerfSort (SubForm):** "1" (circled 2.1).
- Ort:** "München".
- Erscheinungsjahr:** "1972" (circled 2).
- Buttons:** "Filter" (circled 2), "An machen".

1	Filter (Tabelle: <i>Filter</i>)		
2	MainForm (Abfrage: <i>Filter_Form_Subform_3Filter</i>)	2.1	SubForm (Tabelle: <i>rel_Medien_Verfasser</i>)

Listefeld «Filter Kategorie» (Abfrage: *Listefeld_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: <i>Ansicht_Standardfilter</i>)	1.1	SubForm (Tabelle: <i>Medien</i>)	1.1.1	SubSubForm (Tabelle: <i>rel_Medien_Verfasser</i>)
---	---	-----	-----------------------------------	-------	---

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

The screenshot shows a 'Filter Kategorie' form. At the top, a dropdown menu labeled 'Filter Kategorie' is open, showing a list of categories. The category 'Aax - Weitere Formen allgemeiner Nachschlagewerke' is highlighted in green. Below this, there are two main dropdown menus: 'Kategorie' (set to 'Fa - Allgemeines') and 'Medienart' (set to 'BU - Buch'). Underneath, there are two text boxes: 'Ort' (set to 'Köln') and 'Erscheinungsjahr' (set to '2008'). At the bottom, there is a sub-form with two columns: 'Verfasser' and 'VerfSort'. The 'Verfasser' dropdown is set to 'Becker, Tom' and 'VerfSort' is set to '1'. Red circles with numbers 1, 2, and 2.1 are overlaid on the image to indicate specific elements.

1	Filter (Tabelle: <i>Filter</i>)	
2	MainForm (Abfrage: <i>Filter_Form</i>)	2.1 SubForm (Tabelle: <i>rel_Medien_Verfasser</i>)

Makros in Feldeigenschaften

1	Feld «Filter Kategorie» (Modifiziert)	Module1. <i>Mauszeiger</i>
---	---------------------------------------	----------------------------

Listenfeld «Filter Kategorie» (Tabelle: *Kategorien, Medien*)

```

1 SELECT DISTINCT LEFT( "Kategorien"."System" || '
      ', 7 ) || ' - ' ||
      LEFT("Kategorien"."Kategorie", 50)|| CASE WHEN
      CHAR_LENGTH("Kategorien"."Kategorie") > 50 THEN '...' ELSE '' END AS "Kat",
2 "Kategorien"."ID"
3 FROM "Kategorien", "Medien"
4 WHERE "Kategorien"."ID" = "Medien"."katID"
5 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

The screenshot shows a database filter form with the following elements:

- Filter Kategorie:** A dropdown menu with 'Aal - Deutschsprachige allgemeine Lexika' selected. A red circle '1' is around the dropdown arrow.
- Kategorie:** A dropdown menu with 'Aal - Deutschsprachige allgemeine Lexika' selected.
- Medienart:** A dropdown menu with 'BU - Buch' selected. A red circle '2' is around the dropdown arrow.
- Ort:** A dropdown menu with 'Mannheim / Wien / Zü' selected.
- Erscheinungsjahr:** A text input field with '1991' entered.
- Verfasser Table:** A table with columns 'Verfasser' and 'VerfSort'. The first row is 'Mevers Lexikonred.' with '1'. The second row is 'Anger, Eberhard (Hrsg., Red.)' with '2' and '99'. A red circle '2.1' is around the '2' in the second row.
- Navigation Bar:** A bar with 'Datensatz 1 von 37' and various icons. A red circle '2' is around the filter icon.

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

Listefeld «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

Filter Kategorie

China. Taiwa **1**

Anfangsbuchstaben beschränken das Listenfeld

China. Taiwan
Christliche Ethik
Computertechnik und Programmierung (EDV)

Kategorie: Cgm 3 - China. Taiwan
Medienart: BU - Buch

Ort: Frankfurt/M. [u.a.]
Erscheinungsjahr: **2** 1983

	Verfasser	VerfSort
▶	Scholl-Latour. Peter	1
	Kaufmann, Josef	2
☀		

2.1

1	Filter (Tabelle: <i>Filter</i>)	
2	MainForm (Abfrage: <i>Filter_Form</i>)	2.1 SubForm (Tabelle: <i>rel_Medien_Verfasser</i>)

Makros in Feldeigenschaften		
1	Feld «Filter Kategorie» (Modifiziert)	Module1. <i>Mauszeiger</i>
1	Feld «Filter Kategorie» (Taste losgelassen)	Module2. <i>ListFilter</i>

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.

Filter_Formular_Subformular

Filter Verfasser

Abendroth, Wolfgang

Abendroth, Wolfgang

Abrahams, Peter

Achilles, Olaf

Aczona, Rafael

ADAC

Gkk1 - Gesamtdarstellungen - Darstellungen

BU - Buch

Ort

Frankfurt am Main

Erscheinungsjahr

1981

	Verfasser	VerfSort
	Abe. Kobo	1
	Abendroth, Wolfgang	2
	(u.a.)	99

1	Filter (Tabelle: <i>Filter</i>)	
2	MainForm (Abfrage: <i>Filter_Form_Subform</i>)	2.1 SubForm (Tabelle: <i>rel_Medien_Verfasser</i>)

Makros in Feldeigenschaften

1	Feld «Filter Verfasser» (Modifiziert)	Module1. <i>Mauszeiger</i>
---	---------------------------------------	----------------------------

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

The screenshot shows a multi-level form. At the top is a 'Filter' section with three dropdown menus: 'Kategorie', 'Verfasser' (containing 'Lindgren, Astrid'), and 'Medienart'. A red circle labeled '1' is around the 'Filter' dropdown. Below this is a table with two columns: 'ID' and 'Medienart'. The first row has '10' and 'Buch' (highlighted in green), the second has '10' and 'Video-DVD'. A red circle labeled '2' is around the 'Medienart' dropdown in the section below, which is set to 'Buch'. Below that are 'Ort' (Hamburg) and 'Erscheinungsjahr' (1994). At the bottom is a sub-form with columns 'Verfasser' and 'VerfSort'. The 'Verfasser' dropdown is set to 'Lindaren, Astrid' and 'VerfSort' is '1'. A red circle labeled '2.1' is around the 'VerfSort' field.

1	Filter (Tabelle: <i>Filter</i>)		
2	MainForm (Abfrage: <i>Filter_Form_Subform_3Filter</i>)	2.1	SubForm (Tabelle: <i>rel_Medien_Verfasser</i>)

Makros in Feldeigenschaften		
1	Feld «Filter Kategorie» (Modifiziert)	Module1. <i>Filter_bedingt</i>
1	Feld «Filter Verfasser» (Modifiziert)	Module1. <i>Filter_bedingt</i>
1	Feld «Filter Medienart» (Modifiziert)	Module1. <i>Filter_bedingt</i>

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_Datensatze

1	Filter (Tabelle: <i>Filter</i>)	
2	MainForm (Abfrage: <i>Filter_Form_Subform_3Filter</i>)	2.1 SubForm (Tabelle: <i>rel_Medien_Verfasser</i>)

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

Listefeld «Filter Kategorie» (Abfrage: [Listefeld_Kategorie_bedingt_Datensaetze](#))

Listefeld «Filter Verfasser» (Abfrage: [Listefeld_Verfasser_bedingt_Datensaetze_optimiert](#))

Listefeld «Filter Medienart» (Abfrage: [Listefeld_Medienart_bedingt_Datensaetze](#))

Zusätzlich zu den Funktionen von [Filter_Formular_Subformular_bedingende_Filter](#) bietet dieses Formular eine Übersicht über die jeweilige Anzahl an vorhandenen Datensätzen 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

The screenshot shows a search interface with the following elements:

- Suche:** Input field with 'lind' (circled 1), 'Suchen' button (circled 2).
- ID:** Input field with '1645'.
- Titel:** Input field with 'Lindner Biologie' (highlighted by a green arrow from the search button).
- Kategorie:** Dropdown menu with 'Ufl - Allgemeine Biologie'.
- Medienart:** Dropdown menu with 'Buch' (circled 2).
- Ort:** Dropdown menu with 'Hannover'.
- Erscheinungsjahr:** Input field with '1995'.
- Sub-Form Table:** Table with columns 'Verfasser' and 'VerfSort'. The first row has '(u.a.)' and '99' (circled 2.1).
- Footer:** 'Datensatz 1 von 4' and navigation icons.

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: <i>Filter</i>)	
2	MainForm (Abfrage: <i>Suche_Form_Subform</i>)	2.1 SubForm (Tabelle: <i>rel_Medien_Verfasser</i>)

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: <i>Filter</i>)	1.1	SubForm (Abfrage: <i>Parametersuche_Medien</i>)	1.1.1	SubSubForm (Tabelle: <i>rel_Medien_Verfasser</i>)
---	---------------------------------	-----	--	-------	--

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 Listfelder 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: <i>Filter</i>)	
2	MainForm (Abfrage: <i>Suche_Form</i>)	2.1 SubForm (Tabelle: <i>rel_Medien_Verfasser</i>)

Makros in Feldeigenschaften		
1	Feld «Suche» (Text modifiziert)	Module1. <i>Filter_Suche</i>

Der einzige Unterschied zu *Suche_Formular* ohne Makros ist der, dass hier der Einsatz einer 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: <i>Filter</i>)	
2	MainForm (Abfrage: <i>Suche_Form_Subform</i>)	2.1 SubForm (Tabelle: <i>rel_Medien_Verfasser</i>)

Makros in Feldeigenschaften

1	Feld «Suche» (Text modifiziert)	Module1. <i>Filter_Suche</i>
---	---------------------------------	------------------------------

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

Suche und Filter

Suchbegriff:

Kategorie **1**

Verfasser

Medienart

Suchen und Filtern **2**

ID Titel **Suchen und Filtern**

Kategorie **2** Medienart

Ort Erscheinungsjahr

	Verfasser	VerfSort
<input type="checkbox"/>	Kästner, Erich	1
<input type="checkbox"/>		

2.1

1	Filter (Tabelle: <i>Filter</i>)	
2	MainForm (Abfrage: <i>Suche_Filter_Form_Subform_3Filter</i>)	2.1 SubForm (Tabelle: <i>rel_Medien_Verfasser</i>)

Listefeld «Filter Kategorie» (Abfrage: *Listefeld_Kategorie_ohne_Eintrag*)

Hier wird die freie Inhaltssuche mit der Filterung durch feste Vorgaben aus den Listefeldern kombiniert. Ohne Makrozusatz zeigen allerdings die Listfelder 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: <i>Filter</i>)	
2	MainForm (Abfrage: <i>Suche_Filter_Form_Subform_3Filter</i>)	2.1 SubForm (Tabelle: <i>rel_Medien_Verfasser</i>)

Makros im Formularelement		
Extras → Anpassen → Ereignisse → Ansicht wurde erzeugt		Module1. <i>Filter_Suche_hierarchisch_Start</i>
Makros in Feldeigenschaften		
1	Feld «Suchbegriff» (Bei Fokusverlust)	Module1. <i>Filter_Suche_hierarchisch</i>
1	Feld «Filter Medienart» (Modifiziert)	Module1. <i>Filter_Suche_hierarchisch</i>
1	Feld «Filter Kategorie» (Modifiziert)	Module1. <i>Filter_Suche_hierarchisch</i>
1	Feld «Filter Verfasser» (Modifiziert)	Module1. <i>Filter_Suche_hierarchisch</i>

Listefeld «Filter Medienart» (Abfrage: [Listefeld_Medienart_bedingt_Suche_Filter](#))

Listefeld «Filter Kategorie» (Abfrage: [Listefeld_Kategorie_bedingt_Suche_Filter](#))

Listefeld «Filter Verfasser» (Abfrage: [Listefeld_Verfasser_bedingt_Suche_Filter](#))

Um eine Aktivierung der Filterfelder zu erledigen wird in den Listefeldern 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_Datensaeetze_mit_Makros

The screenshot shows a search and filter interface with the following elements:

- Suchbegriff:** Text input field containing 'lind'.
- Medienart:** Dropdown menu with a red circle '1' around it. The menu is open, showing 'Buch -> 41 Treffer' (highlighted) and 'Video-DVD -> 1 Treffer'.
- Kategorie:** Dropdown menu with a red circle '2' around it. The menu is open, showing 'A - Allgemeines. Wissenschaft, Kultur, Inform' and 'BU - Buch'.
- Verfasser:** Dropdown menu with a red circle '2.1' around it. The menu is open, showing 'Lindaren. Astrid'.
- ID:** Text input field containing '104'.
- Titel:** Text input field containing 'Die Brüder Löwenherz'.
- Ort:** Dropdown menu containing 'Hamburg'.
- Erscheinungsjahr:** Text input field containing '1994'.
- Table:** A table with columns 'Verfasser' and 'VerfSort'. The first row contains 'Lindaren. Astrid' and '1'. A red circle '2.1' is around the 'VerfSort' value '1'.

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. <i>Filter_Suche_hierarchisch_Start</i>
Makros in Feldeigenschaften		
1	Feld «Suchbegriff» (Bei Fokusverlust)	Module1. <i>Filter_Suche_hierarchisch_allround</i>
1	Feld «Filter Medienart» (Modifiziert)	Module1. <i>Filter_Suche_hierarchisch_allround</i>
1	Feld «Filter Kategorie» (Modifiziert)	Module1. <i>Filter_Suche_hierarchisch_allround</i>
1	Feld «Filter Verfasser» (Modifiziert)	Module1. <i>Filter_Suche_hierarchisch_allround</i>

Listefeld «Filter Medienart» (Abfrage: [Listenfeld_Medienart_bedingt_Suche_Filter_Datensaetze](#)).

Die folgenden Listensfelder beziehen sich zwar auf die entsprechenden Abfragen. Zum Formularstart ist die Abfrage aber leer. Dadurch wird das Feld erst geladen, wenn es aktiv wird:

Listefeld «Filter Kategorie» (Abfrage: [Listenfeld_Kategorie_bedingt_Suche_Filter_Datensaetze](#))

Listefeld «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 Listensfeldern in den Zusatzinformationen ein Text hinterlegt:

Textfeld «Suchbegriff» **Allgemein → Zusatzinformation:**

lboFilter1>Listenfeld_Medienart_bedingt_Suche_Filter_Datensaetze,lboFilter2

Listefeld «Filter_Medienart» **Allgemein → Zusatzinformation:**

lboFilter2>Listenfeld_Kategorie_bedingt_Suche_Filter_Datensaetze,lboFilter3

Listefeld «Filter_Kategorie» **Allgemein → Zusatzinformation:**

lboFilter3>Listenfeld_Verfasser_bedingt_Suche_Filter_Datensaetze

Listefeld «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 Listensfeldern werden auch die möglichen Treffer angezeigt.
- Die inaktiven Listensfelder 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 Listensfeld die Anzahl der anzuzeigenden Datensätze bei der Eingabe begrenzt.

Filter

Aufruf aus
Formular: Filter_Formular , Filter_Formular_Listfeldbegrenzung , Filter_Formular_Subformular

1 **SUB Filter**

```

2   DIM oDoc AS OBJECT
3   DIM oDrawpage AS OBJECT
4   DIM oForm1 AS OBJECT
5   DIM oForm2 AS OBJECT
6   DIM oFeldList AS OBJECT
7   oDoc = thisComponent
8   oDrawpage = oDoc.drawpage
9   oForm1 = oDrawpage.forms.getByname("Filter")
10  oForm2 = oDrawpage.forms.getByname("MainForm")
11  oFeldList = oForm1.getByname("lboFilter")
12  oFeldList.commit()
13  oForm1.updateRow()
14  oForm2.reload()
15 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*

```

1 SUB Filter_Formular_direkt
2   DIM oDoc AS OBJECT
3   DIM oDrawpage AS OBJECT
4   DIM oForm1 AS OBJECT
5   DIM oForm2 AS OBJECT
6   DIM oFeldList AS OBJECT
7   DIM stListValue AS STRING
8   oDoc = thisComponent
9   oDrawpage = oDoc.drawpage
10  oForm1 = oDrawpage.forms.getByname("Filter")
11  oForm2 = oDrawpage.forms.getByname("MainForm")
12  oFeldList = oForm1.getByname("lboFilter")
13  stListValue = oFeldList.getCurrentValue()
14  IF stListValue = "" THEN
15      oForm2.Filter = ""
16  ELSE
17      oForm2.Filter = "kat = " + stListValue + ""
18  END IF
19  oForm2.reload()
20 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 die-

sen 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*

```
1 SUB Filter_bedingt
2   DIM oDoc AS OBJECT
3   DIM oDrawpage AS OBJECT
4   DIM oForm1 AS OBJECT
5   DIM oForm2 AS OBJECT
6   DIM oFeldList1 AS OBJECT
7   DIM oFeldList2 AS OBJECT
8   DIM oFeldList3 AS OBJECT
9   oDoc = thisComponent
10  oDrawpage = oDoc.drawpage
11  oForm1 = oDrawpage.forms.getByname("Filter")
12  oForm2 = oDrawpage.forms.getByname("MainForm")
13  oFeldList1 = oForm1.getByname("lboFilter1")
14  oFeldList2 = oForm1.getByname("lboFilter2")
15  oFeldList3 = oForm1.getByname("lboFilter3")
16  oFeldList1.commit()
17  oFeldList2.commit()
18  oFeldList3.commit()
19  oForm1.updateRow()
20  oFeldList1.refresh()
21  oFeldList2.refresh()
22  oFeldList3.refresh()
23  oForm2.reload()
24 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*

```
1 SUB Filter_bedingt_allround(oEvent AS OBJECT)
2   DIM oDoc AS OBJECT
3   DIM oDrawpage AS OBJECT
4   DIM oForm1 AS OBJECT
```

```

5   DIM oForm2 AS OBJECT
6   DIM oFeld AS OBJECT
7   DIM stTag AS String
8   DIM arList()
9   DIM i AS INTEGER
10  oFeld = oEvent.Source.Model
11  oForm1 = oFeld.Parent
12  stTag = oFeld.Tag
13  arList = Split(stTag, ",")
14  oDoc = thisComponent
15  oDrawpage = oDoc.drawpage
16  oForm2 = oDrawpage.forms.getByname("MainForm")
17  oFeld.commit()
18  oForm1.updateRow()
19  FOR i = LBound(arList()) TO UBound(arList())
20      oForm1.getByname(arList(i)).refresh()
21  NEXT
22  IF IsEmpty(oFeld.currentValue) THEN
23      oFeld.refresh()
24  END IF
25  oForm2.reload()
26 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 **IboFilter2**, **arList(1)** ist gleich **IboFilter3**. 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*

```
1 SUB Filter_Suche(oEvent AS OBJECT)
2   DIM oDoc AS OBJECT
3   DIM oDrawpage AS OBJECT
4   DIM oForm1 AS OBJECT
5   DIM oForm2 AS OBJECT
6   DIM stFeld AS STRING
7   oDoc = thisComponent
8   oDrawpage = oDoc.drawpage
9   oForm2 = oDrawpage.forms.getByName("MainForm")
10  oFeld = oEvent.Source.Model
11  stFeld = oFeld.CurrentValue
12  IF Len(stFeld) > 2 OR Len(stFeld) > 0
      OR oFeld.ImplementationName = "com.sun.star.form.OListBoxModel" THEN
13      oForm1 = oFeld.Parent
14      oFeld.commit()
15      oForm1.updateRow()
16      oForm2.reload()
17  END IF
18 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*

```
1 SUB Filter_Suche_hierarchisch(oEvent AS OBJECT)
2   DIM oDoc AS OBJECT
3   DIM oDrawpage AS OBJECT
4   DIM oForm1 AS OBJECT
5   DIM oForm2 AS OBJECT
6   DIM oFeld AS OBJECT
7   DIM stTag AS String
8   DIM arList()
9   DIM i AS INTEGER
10  oDoc = thisComponent
11  oDrawpage = oDoc.drawpage
12  oForm2 = oDrawpage.forms.getByName("MainForm")
13  oFeld = oEvent.Source.Model
14  oFeld.commit()
15  oForm1 = oFeld.Parent
16  oForm1.updateRow()
```

```

17     stTag = oFeld.Tag
18     arList = Split(stTag, ",")
19     FOR i = LBound(arList()) TO UBound(arList())
20         IF IsEmpty(oFeld.currentValue) THEN
21             oForm1.getByName(arList(i)).Enabled = FALSE
22         ELSE
23             oForm1.getByName(arList(0)).Enabled = TRUE
24         END IF
25         oForm1.getByName(arList(i)).BoundField.UpdateNULL()
26         oForm1.updateRow()
27         oForm1.getByName(arList(i)).refresh()
28     NEXT
29     oForm2.reload()
30 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*

```

1 SUB Filter_Suche_hierarchisch_Start
2     DIM oDatenquelle AS OBJECT
3     DIM oVerbindung AS OBJECT
4     DIM oSQL_Anweisung AS OBJECT
5     DIM stSql AS STRING
6     oDatenquelle = ThisComponent.Parent.CurrentController
7     If NOT (oDatenquelle.isConnected()) THEN
8         oDatenquelle.connect()
9     END IF
10    oVerbindung = oDatenquelle.ActiveConnection()
11    oSQL_Anweisung = oVerbindung.createStatement()
12    stSql = "DELETE FROM ""Filter""
13    oSQL_Anweisung.executeUpdate(stSql)
14    stSql = "INSERT INTO ""Filter"" (""ID"" ) VALUES (True)"
15    oSQL_Anweisung.executeUpdate(stSql)
16 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*

```
1 SUB Filter_Suche_hierarchisch_allround(oEvent AS OBJECT)
2   DIM oDoc AS OBJECT
3   DIM oDrawpage AS OBJECT
4   DIM oForm1 AS OBJECT
5   DIM oForm2 AS OBJECT
6   DIM oFeld AS OBJECT
7   DIM stTag AS String
8   DIM arList()
9   DIM arTmp()
10  DIM stSql(0) AS STRING
11  DIM i AS INTEGER
12  oDoc = thisComponent
13  oDrawpage = oDoc.drawpage
14  oForm2 = oDrawpage.forms.getByName("MainForm")
15  oFeld = oEvent.Source.Model
16  oFeld.commit()
17  oForm1 = oFeld.Parent
18  oForm1.updateRow()
19  stTag = oFeld.Tag
20  arList = Split(stTag, ",")
21  FOR i = LBound(arList()) TO UBound(arList())
22    IF i = 0 THEN
23      arTmp = Split(arList(0), ">")
24      arList(0) = arTmp(0)
25      stSql(0) = arTmp(1)
26    END IF
27    IF IsEmpty(oFeld.currentValue) THEN
28      stSql(0) = ""
29      oForm1.getByName(arList(i)).Enabled = FALSE
30      oForm1.getByName(arList(i)).ListSource = stSql
31    ELSE
32      oForm1.getByName(arList(0)).Enabled = TRUE
33      oForm1.getByName(arList(0)).ListSource = stSql
34    END IF
35    oForm1.getByName(arList(i)).BoundField.UpdateNULL()
36    oForm1.updateRow()
37    oForm1.getByName(arList(i)).refresh()
38  NEXT
39  oForm2.reload()
```

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*

```
1 GLOBAL stListStart AS STRING
2 GLOBAL lZeit AS LONG
```

```
1 SUB ListFilter(oEvent AS OBJECT)
2   oFeld = oEvent.Source.Model
3   IF oEvent.KeyCode < 538 OR oEvent.KeyCode = 1283 OR oEvent.KeyCode = 1284
4     OR oEvent.KeyCode = 1281 THEN
5     DIM stSql(0) AS STRING
6     DIM stText AS STRING
7     DIM stFeld AS STRING
8     DIM stQuery AS STRING
9     DIM ar0()
10    DIM ar1()
11    ar0() = Split(oFeld.ListSource(0), ",", 2)
12    ar1() = Split(ar0(0), ":", 2)
13    stFeld = "" & Rtrim(ar1(1))
14    stQuery = Left(oFeld.ListSource(0), InStr(1, oFeld.ListSource(0), "ORDER", 1) - 1)
15    IF InStr(stQuery, "LOWER") > 0 THEN
16      stQuery = Left(stQuery, InStr(stQuery, "LOWER") - 1)
17    ELSEIF InStr(1, stQuery, "WHERE", 1) > 0 AND Right(stQuery, 5) <> " AND " THEN
18      stQuery = stQuery & " AND "
```

```

19         stQuery = stQuery & " WHERE "
20     END IF
21     IF oEvent.KeyCode = 1281 THEN 'Taste ESC
22         stListStart = "" 'Bei ESC wieder den gesamten Inhalt anzeigen
23     ELSEIF lZeit > 0 AND Timer() - lZeit < 5 THEN
24         stListStart = stListStart & oEvent.KeyChar
25     ELSE
26         stListStart = oEvent.KeyChar
27     END IF
28     lZeit = Timer()
29     stText = LCase( stListStart & "%")
30     stSql(0) = stQuery + "LOWER("+stFeld+") LIKE '"+stText+"' ORDER BY "+stFeld+"
31     oFeld.ListSource = stSql
32     oFeld.refresh
33 END IF
34 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.

Entsprechend muss auch beim Auslesen des SQL-Codes aus dem Listenfeld darauf geachtet werden, dass der SQL-Code nicht direkt als Text erreichbar ist. Stattdessen ist der Code in einem Array als einziger Eintrag vorhanden: **oFeld.ListSource(0)** (Zeile 10).

Der SQL-Code wird nach der Deklaration der Variablen für die weitere Verwendung aufgesplittet. Für das Feld, das gefiltert werden soll, wird nach dem ersten Komma der Code abgetrennt (Zeile 10 bis 12). Das Feld muss also an der ersten Position stehen. Anschließend wird der verbleibende Teil an dem ersten erscheinenden Anführungsstrich «"» aufgetrennt. Damit beginnt die Feldbezeichnung. Diese Aufteilungen erfolgen hier mit einfachen Arrays. Der Variablen **stFeld** wird schließlich wieder das doppelte Anführungszeichen am Beginn hinzugefügt. Außerdem wird über **Rtrim** vermieden, dass eine eventuell noch vorhandene Leertaste am Schluss des Ausdrucks bestehen bleibt.

In dem SQL-Code wird eine Sortieranweisung erwartet. Allerdings kann die Anweisung in SQL in Großbuchstaben, Kleinbuchstaben oder beliebig gemischt erfolgen. Deshalb wird hier nicht mit **Split**, sondern mit Hilfe der Funktion **inStr** nach der Zeichenkette «ORDER» gesucht. Der abschließende Parameter in dieser Funktion sagt mit der «1» aus, dass nicht nach Groß- und

Kleinschreibung unterschieden werden soll. Alles, was links von der Zeichenkette «ORDER» steht, soll für die Konstruktion des neuen SQL-Codes weiter genutzt werden (Zeile 13). Damit ist gewährleistet, dass auch Listenfelder bestückt werden können, die aus unterschiedlichen Tabellen oder über weitere Bedingungen im SQL-Code definiert worden sind.

Enthält die ermittelte Abfrage den Begriff «LOWER», so wird davon ausgegangen, dass die Abfrage bereits über die Prozedur **ListFilter** erstellt wurde. Deswegen wird die neu zu konstruierende Abfrage nur bis zur Position dieses Begriffes übernommen (Zeile 14 und 15).

Ist dies nicht der Fall und es existiert in der Abfrage bereits der Begriff «WHERE» in beliebiger Schreibweise, so müssen weitere Bedingungen an die Abfrage mit **AND** angehängt werden, sofern nicht bereits beim letzten Zugriff ein **AND** angehängt wurde (Zeile 16 und 17).

Sind beide Bedingungen nicht erfüllt, so wird ein **WHERE** an den bestehenden Code angehängt (Zeile 18 und 19).

In der folgenden Verzweigung wird zuerst die Bedingung abgefragt, ob **ESC** gedrückt wurde (Zeile 21). 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 23 und 24). Anderenfalls wird der eingegebene Buchstabe als einzige (neue) Eingabe verstanden (Zeile 26). 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 29 und 30). 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 Datenbank⁸ lässt sich mit unterschiedlichen Formularen, gegebenenfalls auch ganz ohne Makrozusatz, bedienen.

Tabellen

Kasse

Datenziel
Ansicht: <i>Ansicht_Kasse_Kategorie, Ansicht_Kasse_mit_Umbuchungen, Ansicht_Bericht_Kategorie</i>
Abfrage: <i>Kasse_Saldo_Konto_Kategorie, Kasse_Saldo_Konto_Kategorie_Umbuch, Fehler: Verweis nicht gefunden, Saldo_Kategorie, Saldo_Kategorie_ungesplittet</i>
Formular: <i>Konto_Salden_komplett_Umbuchung_eM, Konto_Salden_komplett_Umbuchung</i>
Makro: <i>periodische_Buchung, SplitRest</i>
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

8 Beispieldatenbank Beispiel_Haushaltsbuch.odt

Feldname	Feldtyp	Beschreibung
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
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:

```
35 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.

```
1 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: <i>Ansicht_Kasse_Kategorie, Ansicht_Kasse_mit_Umbuchungen</i>
Makro: <i>Adressat_neu</i>
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 Listefeldern 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

```
1 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

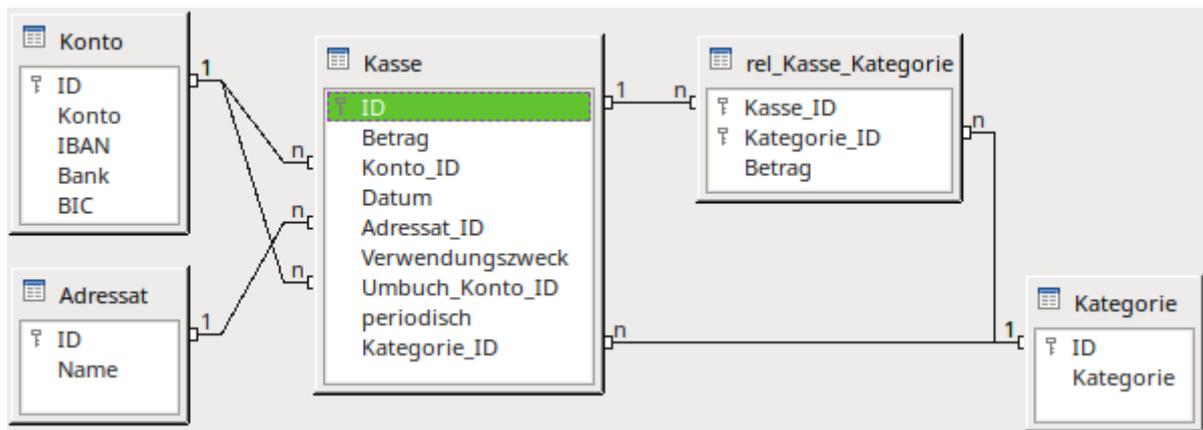
Feldname	Feldtyp	Beschreibung
Kasse_ID	Integer	Zusammen mit "Kategorie_ID" Primärschlüssel der Tabelle. Gleichzeitig Fremdschlüssel für die Verbindung zur Tabelle "Kasse".
Kategorie_ID	Tiny Integer	Zusammen mit "Kasse_ID" Primärschlüssel der Tabelle. Gleichzeitig Fremdschlüssel für die Verbindung zur 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.

```
2 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 "Kategorie_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 Tabellen 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: <i>Kasse, Adressat, Konto, Kategorie, rel_Kasse_Kategorie</i>

Datenziel
Ansicht: <i>Ansicht_Kategorie_Diagramm</i>
Abfrage: <i>Kategorieverlauf, Fehler: Verweis nicht gefunden, Saldo_Kategorie</i>

```

1 SELECT "Kasse"."ID", "Kasse"."Betrag" AS "Gesamtbetrag",
2 COALESCE( "rel_Kasse_Kategorie"."Betrag", "Kasse"."Betrag" ) AS "Splitbetrag",
3 "Konto"."Konto", "Kasse"."Datum", "Adressat"."Name", "Kasse"."Verwendungszweck",
4 "rel_Kasse_Kategorie"."Kategorie_ID",
5 COALESCE( "Kategorie"."Kategorie",
6 (SELECT "Kategorie" FROM "Kategorie" WHERE "ID" = 0)) AS "Kategorie",
7 EXTRACT( YEAR FROM "Datum" ) || '-' || RIGHT( '0' || EXTRACT( MONTH FROM "Datum" ), 2
8 ) AS "JahrMonat"
9 FROM "Kasse"
10 LEFT JOIN "Konto" ON "Kasse"."Konto_ID" = "Konto"."ID"
11 LEFT JOIN "Adressat" ON "Kasse"."Adressat_ID" = "Adressat"."ID"
12 LEFT JOIN "rel_Kasse_Kategorie" ON "Kasse"."ID" = "rel_Kasse_Kategorie"."Kasse_ID"
13 LEFT JOIN "Kategorie" ON "rel_Kasse_Kategorie"."Kategorie_ID" = "Kategorie"."ID"
14 WHERE "Kasse"."Umbuch_Konto_ID" IS NULL
15 UNION
16 SELECT "a"."ID", "a"."Betrag" AS "Gesamtbetrag",
17 "a"."Betrag" - COALESCE((SELECT SUM( "Betrag" ) FROM "rel_Kasse_Kategorie" WHERE
18 "Kasse_ID" = "a"."ID" ),0) AS "Splitbetrag",
19 "Konto"."Konto", "a"."Datum", "Adressat"."Name", "a"."Verwendungszweck",
20 0 AS "Kategorie_ID",
21 (SELECT "Kategorie" FROM "Kategorie" WHERE "ID" = 0) AS "Kategorie",
22 EXTRACT( YEAR FROM "a"."Datum" ) || '-' || RIGHT( '0' || EXTRACT( MONTH FROM
23 "a"."Datum" ), 2 ) AS "JahrMonat"
24 FROM "Kasse" AS "a"
25 LEFT JOIN "Konto" ON "a"."Konto_ID" = "Konto"."ID"
26 LEFT JOIN "Adressat" ON "a"."Adressat_ID" = "Adressat"."ID"
27 LEFT JOIN "rel_Kasse_Kategorie" ON "a"."ID" = "rel_Kasse_Kategorie"."Kasse_ID"
28 LEFT JOIN "Kategorie" ON "rel_Kasse_Kategorie"."Kategorie_ID" = "Kategorie"."ID"
29 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:

```
25 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 vorgefiltert, 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 vorgehen mit der Umbuchung gerechnet, so würde der Verlust zwar z.B. auf dem Girokonto berücksichtigt, das Geld aber nicht vom Kassensautomaten 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: <i>Kasse, Adressat, Konto</i>
Datenziel
Ansicht: <i>Ansicht_Konto_Diagramm</i>
Abfrage: <i>Kontoverlauf, Fehler: Verweis nicht gefunden, Saldo_Konto</i>

```

1 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"
2 FROM "Kasse"
3 LEFT JOIN "Konto" ON "Kasse"."Konto_ID" = "Konto"."ID"
4 LEFT JOIN "Adressat" ON "Kasse"."Adressat_ID" = "Adressat"."ID"
5 UNION
6 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"
7 FROM "Kasse"
8 LEFT JOIN "Konto" ON "Kasse"."Umbuch_Konto_ID" = "Konto"."ID"
9 LEFT JOIN "Adressat" ON "Kasse"."Adressat_ID" = "Adressat"."ID"
10 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

```

1 SELECT "Filter"."ID", "Filter"."Kasse_ID",
2 COALESCE("Filter"."StartDatum",CAST( EXTRACT( YEAR FROM CURRENT_DATE ) || '-01-01' AS
   DATE )) AS "StartDatum",
3 COALESCE("Filter"."EndDatum",CAST( EXTRACT( YEAR FROM CURRENT_DATE ) || '-12-31' AS
   DATE )) AS "EndDatum",
4 "Filter"."DatensatzDatum",
5 CASE WHEN (SELECT MAX( "Datum" ) FROM "Kasse") > CURRENT_DATE THEN (SELECT
   MAX( "Datum" ) FROM "Kasse") ELSE CURRENT_DATE END AS "Datum_Max"
6 FROM "Filter"
7 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

```

1 SELECT "R"."RowDescription", "C"."ColumnDescription",
2 (SELECT SUM( "Splitbetrag" ) FROM "Ansicht_Kasse_Kategorie" WHERE "Kategorie" =
   "R"."RowDescription" AND "JahrMonat" = "C"."ColumnDescription") AS "Data",
3 'BC' AS "Type"
4 FROM
5 (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",
6 (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"
7 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

```

1 SELECT "Konto" AS "RowDescription", '' AS "ColumnDescription",
2 SUM( "Betrag" ) AS "Data",
3 'CC' AS "Type"
4 FROM "Ansicht_Kasse_mit_Umbuchungen"
5 WHERE "Datum" BETWEEN ( SELECT "StartDatum" FROM "Ansicht_Filter" ) AND ( SELECT
  "EndDatum" FROM "Ansicht_Filter" )
6 GROUP BY "Konto"
7 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

```

1 SELECT "Kasse"."ID", "Kasse"."Betrag", "Kasse"."Datum" AS "Datum",
  "Kasse"."Verwendungszweck",
2 COALESCE("Kategorie"."Kategorie", (SELECT "Kategorie" FROM "Kategorie" WHERE "ID" =
  0)) AS "Kategorie",
3 ( SELECT "StartDatum" FROM "Ansicht_Filter" ) AS "StartDatum",
4 ( SELECT "EndDatum" FROM "Ansicht_Filter" ) AS "EndDatum"
5 FROM "Kasse"

```

```

6 LEFT JOIN "Kategorie" ON "Kasse"."Kategorie_ID" = "Kategorie"."ID"
7 WHERE "Kasse"."Umbuch_Konto_ID" IS NULL
8 AND "Kasse"."Datum" BETWEEN ( SELECT "StartDatum" FROM "Ansicht_Filter" ) AND
  ( SELECT "EndDatum" FROM "Ansicht_Filter" )
9 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](#)

```

1 SELECT "a"."Kategorie", "a"."Datum", "a"."Splitbetrag" "Betrag",
2 ( 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",
3 ( SELECT "StartDatum" FROM "Ansicht_Filter" ) AS "StartDatum",
4 ( SELECT "EndDatum" FROM "Ansicht_Filter" ) AS "EndDatum"
5 FROM "Ansicht_Kasse_Kategorie" AS "a"
6 WHERE "a"."Datum" BETWEEN "StartDatum" AND "EndDatum"
7 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:

```

6 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: <i>Kasse, Konto</i>

Datenziel
Formular: <i>Kasse_oM, Konto_oM, Konto_Salden_oM, Konto_Salden_separat_oM, Konto_Salden_komplett_eM, Kasse, Konto, Konto_Salden, Konto_Salden_komplett</i>

```
1 SELECT "a".*,
2 CASE WHEN "a"."Betrag" > 0 THEN "a"."Betrag" ELSE 0 END AS "Einnahme",
3 CASE WHEN "a"."Betrag" < 0 THEN "a"."Betrag" ELSE 0 END AS "Ausgabe",
4 ( SELECT "Konto" FROM "Konto" WHERE "ID" = "a"."Konto_ID" ) AS "Kontoname",
5 ( 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",
6 ( 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",
7 ( 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"
8 FROM "Kasse" AS "a"
9 WHERE "a"."Umbuch_Konto_ID" IS NULL
10 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: <i>Kasse, Konto</i>

Datenziel

Formular: [Kasse_Umbuchung](#)

```
1 SELECT "a".*,
2 CASE WHEN "a"."Betrag" > 0 AND "Umbuch_Konto_ID" IS NULL THEN "a"."Betrag" ELSE 0 END
  AS "Einnahme",
3 CASE WHEN "a"."Betrag" < 0 AND "Umbuch_Konto_ID" IS NULL THEN "a"."Betrag" ELSE 0 END
  AS "Ausgabe",
4 ( SELECT "Konto" FROM "Konto" WHERE "ID" = "a"."Konto_ID" ) AS "Kontoname",
5 ( 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",
6 ( 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",
7 ( SELECT "Konto" FROM "Konto" WHERE "ID" = "a"."Umbuch_Konto_ID" ) AS
  "Umbuch_Kontoname",
8 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",
9 ( 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"
10 FROM "Kasse" AS "a"
11 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" 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](#)

```
1 SELECT "a"."Konto", "a"."Datum", "a"."Betrag",
```

```

2 ( 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",
3 ( SELECT "StartDatum" FROM "Ansicht_Filter" ) AS "StartDatum",
4 ( SELECT "EndDatum" FROM "Ansicht_Filter" ) AS "EndDatum"
5 FROM "Ansicht_Kasse_mit_Umbuchungen" AS "a"
6 WHERE "a"."Datum" BETWEEN "StartDatum" AND "EndDatum"
7 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:

```

6 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: <i>Kasse, Kategorie</i>
Ansicht: <i>Ansicht_Kasse_Kategorie</i>

Datenziel
Formular: <i>Konto_Salden, Konto_Salden_komplett, Konto_Salden_komplett_Umbuchung, Buchung_Umbuchung_Salden</i>

```

1 SELECT "a"."ID", "a"."Kategorie",
2 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",
3 ( SELECT SUM( "Betrag" ) FROM "Kasse" WHERE "Umbuch_Konto_ID" IS NULL AND "Datum"
  <= :qDatum ) AS "Saldo_gesamt"
4 FROM "Kategorie" AS "a"
5 WHERE "a"."ID" IN ( SELECT DISTINCT "Kategorie_ID" FROM "Kasse" WHERE "Datum"
  <= :qDatum )
6 OR "a"."ID" = 0
7 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*

```

1 SELECT "a"."ID", "a"."Kategorie",
2 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",
3 ( SELECT SUM( "Betrag" ) FROM "Kasse" WHERE "Umbuch_Konto_ID" IS NULL AND "Datum"
  <= :qDatum ) AS "Saldo_gesamt"
4 FROM "Kategorie" AS "a"
5 WHERE "a"."ID" IN ( SELECT DISTINCT "Kategorie_ID" FROM "Kasse" WHERE "Datum"
  <= :qDatum )
6 OR "a"."ID" = 0
7 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](#)

```
1 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"  
2 FROM "Ansicht_Kasse_mit_Umbuchungen"  
3 WHERE "Datum" <= :qDatum  
4 GROUP BY "Konto_ID", "Konto"  
5 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 LittleBox	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 I	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 LittleBox	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 €

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 Listefeldern 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

Konto	Giro	Bank	Spara Phantastica
IBAN	DE13267834000001234	BIC	WOAUCH2DD

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 LittleBox	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 LittleBox	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 LittleBox	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 LittleBox	0,00 €	-545,00 €	4.156,27 €	-2.180,00 €	3.805,27 €

1	MainForm (Tabelle: <i>Konto</i>)	1.1	SubForm (Abfrage: <i>Kasse_Saldo_Konto_Kategorie</i>)
---	-----------------------------------	-----	--

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

The screenshot shows a software interface for account management. At the top, there are input fields for 'Konto' (Giro) and 'Bank' (Spara Phantastica). Below this is a table of transactions with columns for ID, Datum, Betrag, Verwendungszweck, Kategorie, and Adressat. A red circle highlights the number '1' in the account ID field. Another red circle highlights the number '1.1' in the sub-form ID field. A green arrow points from the '1.1' circle to the 'Saldo bis zum' field, which is set to '10.01.19'. Below the main table are two summary tables: 'Konto' and 'Kategorie', both with red circles around their respective IDs '1.1.1' and '1.1.2'.

1	MainForm (Tabelle: <i>Konto</i>)	1.1	SubForm (Abfrage: <i>Kasse_Saldo_Konto_Kategorie</i>)	1.1.1	SaldoKonto (Abfrage: <i>Saldo_Konto</i>)
				1.1.2	SaldoKategorie (Abfrage: <i>Saldo_Kategorie_ungesplittet</i>)

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 Konto-stand wieder, sondern den Kassenstand zum Ende des Buchungstages.

Konto_Salden_separat_oM

Konto		Bank	
Bargeld			
IBAN		BIC	

ID	Datum	Betrag	Verwendungszweck	Kategorie	Adressat
3	02.01.19	-12,69 €	Medikamente	Gesundheit	
10	09.01.19	-45,37 €	Markt, Lebensmittel	Ernährung	Bio leben und leben la
11	12.01.19	-45,57 €	Markt, Lebensmittel	Ernährung	Arzt für Auszahlung
46	05.02.19	-45,37 €	Markt, Lebensmittel	Ernährung	Bio leben und leben la
58	23.01.20	-65,49 €	Markt, Lebensmittel	Ernährung	Bio leben und leben la
59	06.02.20	-45,13 €	Markt, Lebensmittel	Ernährung	Bio leben und leben la
pFeld>					

Datensatz	4	von 6 (1)	⏪ ⏩ ⏴ ⏵ ⚙
-----------	---	-----------	-----------

Saldo Konto	Saldo Kategorie	Saldo gesamt
-149,00 €	-136,31 €	1.533,09 €

1 MainForm (Tabelle: <i>Konto</i>)	1.1 SubForm (Abfrage: <i>Kasse_Saldo_Konto_Kategorie</i>)
--	---

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

The screenshot shows a financial application interface. At the top, there are input fields for 'Konto' (Giro) and 'Bank' (Spara Phantastica). Below that, 'IBAN' (DE13267834000001234) and 'BIC' (WOAUCH2DD) are displayed. A table of transactions follows, with columns for ID, Datum, Betrag, Verwendungszweck, Kategorie, and Adressat. A red circle '1' highlights the Giro account number. Another red circle '1.1' highlights a transaction entry. Below the table, there are navigation controls and a 'Datensatz' section showing '1 von 31'. A 'Saldo bis zum' date of '23.07.20' is shown with a red circle '2'. At the bottom, there are two summary tables: 'Konto' (with columns Saldo_Konto and Saldo_gesamt) and 'Kategorie' (with columns Kategorie and Saldo_Kategorie). Red circles '2.1' and '2.2' highlight these summary tables.

1	MainForm (Tabelle: <i>Konto</i>)	1.1	SubForm (Abfrage: <i>Kasse_Saldo_Konto_Kategorie</i>)
2	MaxDatum (Ansicht: <i>Ansicht_Filter</i>)	2.1	SaldoKonto (Abfrage: <i>Saldo_Konto</i>)
		2.2	SaldoKategorie (Abfrage: <i>Saldo_Kategorie_ungesplittet</i>)

Makros in Formulareigenschaften

1.1	Nach der Datensatzaktion	Module1. <i>Aktualisieren_einfach</i>
-----	--------------------------	---------------------------------------

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 Formulare 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

The screenshot displays a financial application interface. At the top, there are input fields for 'Konto' (Giro) and 'Bank' (Spa Phantastica). Below this is a table of transactions with columns for ID, Datum, Betrag, Verwendungszweck, Kategorie, Adressat, and Umbuchung auf. A red circle labeled '1' is around the 'Giro' account name. Another red circle labeled '1.1' is around a transaction row with ID 5. Below the transaction table, there is a 'Datensatz' section showing '1 von 41 *'. Below that, there are two summary tables: 'Saldo bis zum 26.07.20' (with a red circle '2') and two smaller tables for 'Konto' (with a red circle '2.1') and 'Kategorie' (with a red circle '2.2').

1	MainForm (Tabelle: <i>Konto</i>)	1.1	SubForm (Tabelle: <i>Kasse</i>)
2	MaxDatum (Ansicht: <i>Ansicht_Filter</i>)	2.1	SaldoKonto (Abfrage: <i>Saldo_Konto</i>)
		2.2	SaldoKategorie (Abfrage: <i>Saldo_Kategorie_ungesplittet</i>)

Makros in Formulareigenschaften		
1.1	Nach der Datensatzaktion	Module1. <i>Aktualisieren_einfach</i>

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

Konto	Datum	Betrag	Verwendungszweck	Adressat	Einnahme	Ausgabe	Saldo Konto	Saldo gesamt
Giro	01.01.19	1.685,34 €	Gehalt	Amt für Auszahlung	1.685,34 €	0,00 €	1.685,34 €	1.685,34 €
Bargeld	02.01.19	-12,69 €	Medikamente		0,00 €	-12,69 €	-12,69 €	1.672,65 €
Giro	02.01.19	-545,00 €	Miete	Wohnungen LittleBoxes	0,00 €	-545,00 €	1.140,34 €	1.127,65 €
Bargeld	09.01.19	-45,37 €	Markt, Lebensmittel	Bio leben und leben lass	0,00 €	-45,37 €	-58,06 €	1.082,28 €
Sparbuch	10.01.19	-215,00 €			0,00 €	-215,00 €	-215,00 €	867,28 €
Giro	10.01.19	-250,00 €	Versicherungen etc.	Desaster-Versicherung	0,00 €	-250,00 €	890,34 €	617,28 €
Bargeld	12.01.19	-45,57 €	Markt, Lebensmittel	Amt für Auszahlung	0,00 €	-45,57 €	-103,63 €	571,71 €
Giro	15.01.19	-16,64 €	Fahrt nach Pusemuckel	Deutsche Bahn	0,00 €	-16,64 €	873,70 €	555,07 €
Giro	01.02.19	-545,00 €	Miete	Wohnungen LittleBoxes	0,00 €	-545,00 €	328,70 €	10,07 €
Giro	03.02.19	1.685,34 €	Gehalt	Amt für Auszahlung	1.685,34 €	0,00 €	2.014,04 €	1.695,41 €
Sparbuch	04.02.19	13,00 €			13,00 €	0,00 €	-202,00 €	1.708,41 €
Giro	05.02.19	-129,95 €	Joggingschuhe	Fit for Sweat	0,00 €	-129,95 €	1.884,09 €	1.578,46 €
Bargeld	05.02.19	-45,37 €	Markt, Lebensmittel	Bio leben und leben lass	0,00 €	-45,37 €	-149,00 €	1.533,09 €
Giro	17.02.19	-8,50 €	Laufsocken	Fit for Sweat	0,00 €	-8,50 €	1.875,59 €	1.524,59 €
Giro	01.03.19	-545,00 €	Miete	Wohnungen LittleBoxes	0,00 €	-545,00 €	1.330,59 €	979,59 €
Giro	01.03.19	1.685,34 €	Gehalt	Amt für Auszahlung	1.685,34 €	0,00 €	3.015,93 €	2.664,93 €
Giro	01.04.19	1.685,34 €	Gehalt	Amt für Auszahlung	1.685,34 €	0,00 €	4.701,27 €	4.350,27 €
Giro	01.04.19	-545,00 €	Miete	Wohnungen LittleBoxes	0,00 €	-545,00 €	4.156,27 €	3.805,27 €
Giro	01.05.19	1.685,34 €	Gehalt	Amt für Auszahlung	1.685,34 €	0,00 €	5.841,61 €	5.490,61 €

Datensatz 6 von 39 (1)

Betrag aus Buchung: -250,00 € Buchungsdatum: 10.01.2019

Kategorie	Betrag
Wohnen	-88,00 €
Mobilität	-75,60 €
Gesundheit	-86,40 €

Datensatz 1 von 3

1	MainForm (Abfrage: <i>Kasse_Saldo_Konto_Kategorie</i>)	1.1	SplitKategorie (Tabelle: <i>rel_Kasse_Kategorie</i>)
---	---	-----	---

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

Konto	Datum	Betrag	Verwendungszweck	Adressat	Umbuchung auf	Einnahme	Ausgabe	Saldo Konto	Saldo gesamt
Giro	01.01.19	1.685,34 €	Gehalt	Arzt für Auszahlung		1.685,34 €	0,00 €	1.685,34 €	1.685,34 €
Giro	01.01.19	-250,00 €	Umbuchung	Sparkasse	Bargeld	0,00 €	0,00 €	1.435,34 €	1.685,34 €
Bargeld	02.01.19	-12,69 €	Medikamente			0,00 €	-12,69 €	237,31 €	1.672,65 €
Giro	02.01.19	-545,00 €	Miete	Wohnungen LittleBoxes		0,00 €	-545,00 €	890,34 €	1.127,65 €
Giro	03.01.19	-300,00 €	Umbuchung	Sparkasse	Sparbuch	0,00 €	0,00 €	590,34 €	1.127,65 €
Bargeld	09.01.19	-45,37 €	Markt, Lebensmittel	Bio leben und leben lass		0,00 €	-45,37 €	191,94 €	1.082,28 €
Sparbuch	10.01.19	-215,00 €				0,00 €	-215,00 €	85,00 €	867,28 €
Giro	10.01.19	-250,00 €	Versicherungen etc.	Desaster-Versicherung		0,00 €	-250,00 €	340,34 €	617,28 €
Bargeld	12.01.19	-45,57 €	Markt, Lebensmittel	Arzt für Auszahlung		0,00 €	-45,57 €	146,37 €	571,71 €
Giro	15.01.19	-16,64 €	Fahrt nach Pusemuckel	Deutsche Bahn		0,00 €	-16,64 €	323,70 €	555,07 €
Giro	01.02.19	-545,00 €	Miete	Wohnungen LittleBoxes		0,00 €	-545,00 €	-221,30 €	10,07 €
Giro	03.02.19	1.685,34 €	Gehalt	Arzt für Auszahlung		1.685,34 €	0,00 €	1.464,04 €	1.695,41 €
Sparbuch	04.02.19	13,00 €				13,00 €	0,00 €	98,00 €	1.708,41 €
Giro	05.02.19	-129,95 €	Joggingschuhe	Fit for Sweat		0,00 €	-129,95 €	1.334,09 €	1.578,46 €
Bargeld	05.02.19	-45,37 €	Markt, Lebensmittel	Bio leben und leben lass		0,00 €	-45,37 €	101,00 €	1.533,09 €
Giro	17.02.19	-8,50 €	Laufsocken	Fit for Sweat		0,00 €	-8,50 €	1.325,59 €	1.524,59 €
Giro	01.03.19	-545,00 €	Miete	Wohnungen LittleBoxes		0,00 €	-545,00 €	780,59 €	979,59 €
Giro	01.03.19	1.685,34 €	Gehalt	Arzt für Auszahlung		1.685,34 €	0,00 €	2.465,93 €	2.664,93 €
Giro	07.03.19	-250,00 €	Umbuchung Giro - Bargeld	Sparkasse	Bargeld	0,00 €	0,00 €	2.215,93 €	2.664,93 €

Datensatz 1 von 41 *

Betrag aus Buchung: 1.685,34 € Buchungsdatum: 01.01.2019

Kategorie	Betrag
Einkünfte	1.685,34 €

Datensatz 1 von 1

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

Konto: Giro, Bank: Spira Phantastica

IBAN: DE13267834000001234, BIC: WOAUCH2DD

ID	Datum	Betrag	Verwendungszweck	Adressat	Einnahme	Ausgabe	Saldo Konto	Saldo gesa...
1	01.01.19	1.685,34 €	Gehalt	Amt für Auszahlung	1.685,34 €	0,00 €	1.685,34 €	1.685,34 €
19	02.01.19	-545,00 €	Miete	Wohnungen LittleBox	0,00 €	-545,00 €	1.140,34 €	1.127,65 €
12	10.01.19	-250,00 €	Versicherungen etc.	Desaster-Versicherun	0,00 €	-250,00 €	890,34 €	617,28 €
5	15.01.19	-16,64 €	Fahrt nach Pusemuckel	Deutsche Bahn	0,00 €	-16,64 €	873,70 €	555,07 €
32	01.02.19	-545,00 €	Miete	Wohnungen LittleBox	0,00 €	-545,00 €	328,70 €	10,07 €
6	03.02.19	1.685,34 €	Gehalt	Amt für Auszahlung	1.685,34 €	0,00 €	2.014,04 €	1.695,41 €
8	05.02.19	-129,95 €	Joggingschuhe	Fit for Sweat	0,00 €	-129,95 €	1.884,09 €	1.578,46 €
9	17.02.19	-8,50 €	Laufsocken	Fit for Sweat	0,00 €	-8,50 €	1.875,59 €	1.524,59 €
33	01.03.19	-545,00 €	Miete	Wohnungen LittleBox	0,00 €	-545,00 €	1.330,59 €	979,59 €
44	01.03.19	1.685,34 €	Gehalt	Amt für Auszahlung	1.685,34 €	0,00 €	3.015,93 €	2.664,93 €
24	01.04.19	1.685,34 €	Gehalt	Amt für Auszahlung	1.685,34 €	0,00 €	4.701,27 €	4.350,27 €
34	01.04.19	-545,00 €	Miete	Wohnungen LittleBox	0,00 €	-545,00 €	4.156,27 €	3.805,27 €

Datensatz: 4 von 31 (1)

Betrag aus Buchung: -16,64 €, Buchungsdatum: 15.01.2019

Kategorie	Betrag
Mobilität	-16,64 €

Datensatz: 1 von 1

1	MainForm (Tabelle: <i>Konto</i>)	1.1	SubForm (Abfrage: <i>Kasse_Saldo_Konto_Kategorie</i>)	1.1.1	SplitKategorie (Tabelle: <i>rel_Kasse_Kategorie</i>)
---	-----------------------------------	-----	--	-------	---

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

Konto: Giro Bank: Spara Phantastica

IBAN: DE13267834000001234 BIC: WOAUCH2DD

ID	Datum	Betrag	Verwendungszweck	Adressat
1	01.01.19	1.685,34 €	Gehalt	Amt für Auszahlung
19	02.01.19	-545,00 €	Miete	Wohnungen LittleBoxe:
12	10.01.19	-250,00 €	Versicherungen etc.	Desaster-Versicherung
5	15.01.19	-16,64 €	Fahrt nach Pusemuckel	Deutsche Bahn
32	01.02.19	-545,00 €	Miete	Wohnungen LittleBoxe:
6	03.02.19	1.685,34 €	Gehalt	Amt für Auszahlung
8	05.02.19	-129,95 €	Loggingschuhe	Fit for Sweat
9	17.02.19	-8,50 €	Laufsöckchen	Fit for Sweat
33	01.03.19	-545,00 €	Miete	Wohnungen LittleBoxe:
44	01.03.19	1.685,34 €	Gehalt	Amt für Auszahlung
24	01.04.19	1.685,34 €	Gehalt	Amt für Auszahlung

Datensatz 7 von 31 (1)

Betrag aus Buchung: -129,95 € Buchungsdatum: 05.02.2019

Kategorie	Betrag
Hobby	-129,95 €

Datensatz 1 von 1

Saldo bis zum 05.02.19

Kontoname	Saldo_Konto	Saldo_gesamt
Bargeld	101,00 €	1.533,09 €
Giro	1.334,09 €	1.533,09 €
Sparbuch	98,00 €	1.533,09 €

Kategorie	Saldo
Ernährung	-122,81 €
Gesundheit	-99,09 €
Hobby	67,45 €
Mobilität	3,24 €
ohne Zuweisung	0,00 €
Wohnen	178,00 €

1	MainForm (Tabelle: <i>Konto</i>)	1.1	SubForm (Abfrage: <i>Kasse_Saldo_Konto_Kategorie</i>)	1.1.1	SplitKategorie (Tabelle: <i>rel_Kasse_Kategorie</i>)
				1.1.2	SaldoKonto (Abfrage: <i>Saldo_Konto</i>)
				1.1.3	SaldoKategorie (Abfrage: <i>Saldo_Kategorie</i>)

Makros in Formulareigenschaften

1.1	Vor der Datensatzaktion	Module1. <i>ZeileZwischenspeichern</i>
1.1	Nach der Datensatzaktion	Module1. <i>SplitRest</i>
1.1.1	Nach der Datensatzaktion	Module1. <i>SplitRest</i>

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

The screenshot displays a financial application interface. At the top, account information is shown: 'Konto' (Giro) and 'Bank' (Spara Phantastica). Below this is a table of transactions with columns for ID, Datum, Betrag, Verwendungszweck, and Adressat. A specific transaction on 05.02.19 for 'Joggingschuhe' is highlighted. Below the table, a detail view for this transaction shows the category 'Hobbv' and the amount '-129,95 €'. At the bottom, two summary tables are visible: 'Saldo bis zum 26.07.20' showing account balances (Bargeld, Giro, Sparsbuch) and a category balance table (Einkünfte, Ernährung, Gesundheit, Hobby, Mobilität, ohne Zuwe...

1	MainForm (Tabelle: <i>Konto</i>)	1.1	SubForm (Abfrage: <i>Kasse_Saldo_Konto_Kategorie</i>)	1.1.1	SplitKategorie (Tabelle: <i>rel_Kasse_Kategorie</i>)
2	MaxDatum (Ansicht: <i>Ansicht_Filter</i>)	2.1	SaldoKonto (Abfrage: <i>Saldo_Konto</i>)		
		2.2	SaldoKategorie (Abfrage: <i>Saldo_Kategorie</i>)		

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

The screenshot displays the 'Konto_Salden' interface. At the top, account details for 'Giro' (ID 1) at 'Spara Phantastica' bank are shown. Below is a list of transactions, with the first transaction (ID 1, 01.01.19, 1.685,34 €) highlighted. A sub-form for splitting the category 'Einkünfte' (1.1.1) is visible, showing a balance of 1.685,34 €. At the bottom, two summary tables are shown: 'Saldo bis zum 20.07.20' for accounts (2.1) and a category split table (2.2).

1	MainForm (Tabelle: <i>Konto</i>)	1.1	SubForm (Tabelle: <i>Kasse</i>)	1.1.1	SplitKategorie (Tabelle: <i>rel_Kasse_Kategorie</i>)
2	MaxDatum (Ansicht: <i>Ansicht_Filter</i>)	2.1	SaldoKonto (Abfrage: <i>Saldo_Konto</i>)		
		2.2	SaldoKategorie (Abfrage: <i>Saldo_Kategorie</i>)		

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

Haushaltskasse

Buchung aus Vorlage: 2 aktueller Monat 1 Adressat hinzufügen

Konto	Datum	Betrag	Verwendungszweck	Adressat	Umbuchung auf	periodisch
Giro	01.01.19	1.685,34 €	Gehalt	Amt für Auszahlung		<input checked="" type="checkbox"/>
Giro	01.01.19	-250,00 €	Umbuchung	Sparkasse	Bargeld	<input type="checkbox"/>
Bargeld	02.01.19	-12,69 €	Medikamente			<input type="checkbox"/>
Giro	02.01.19	-545,00 €	Miete	Wohnungen LittleBoxes		<input checked="" type="checkbox"/>
Giro	03.01.19	-300,00 €	Umbuchung	Sparkasse	Sparbuch	<input type="checkbox"/>
Bargeld	09.01.19	-45,37 €	Markt, Lebensmittel	Bio leben und leben lasse		<input checked="" type="checkbox"/>
Sparbuch	10.01.19	-215,00 €				<input type="checkbox"/>
Giro	10.01.19	-250,00 €	Versicherungen etc.	Desaster-Versicherung		<input type="checkbox"/>
Bargeld	12.01.19	-45,57 €	Markt, Lebensmittel	Amt für Auszahlung		<input type="checkbox"/>
Giro	15.01.19	-16,64 €	Fahrt nach Pusemuckel	Deutsche Bahn		<input type="checkbox"/>
Giro	01.02.19	-545,00 €	Miete	Wohnungen LittleBoxes		<input type="checkbox"/>
Giro	03.02.19	1.685,34 €	Gehalt	Amt für Auszahlung		<input type="checkbox"/>
Sparbuch	04.02.19	13,00 €				<input type="checkbox"/>
Giro	05.02.19	-129,95 €	Joggingschuhe	Fit for Sweat		<input type="checkbox"/>
Bargeld	05.02.19	-45,37 €	Markt, Lebensmittel	Bio leben und leben lasse		<input type="checkbox"/>

Datensatz 1 von 41 *

Betrag aus Buchung: 1.685,34 € Buchungsdatum: 01.01.2019 Saldo bis zum 01.01.19 2

Kategorie	Betrag
Einkünfte	1.685,34 €

Datensatz 1 von 1

Kontoname	Saldo_Konto	Saldo_gesamt
Bargeld	250,00 €	1.685,34 €
Giro	1.435,34 €	1.685,34 €

1.1 2.1 2.2

Saldo bis zum 21.07.20 3

Diagrammübersicht starten

vom 2 bis zum

01.01.19 31.12.19

Formular aufrufe 3

Kontoname	Saldo_Konto	Saldo_gesamt
Bargeld	4.090,38 €	12.611,79 €
Giro	7.673,41 €	12.611,79 €
Sparbuch	848,00 €	12.611,79 €

Kategorie	Saldo
Einkünfte	20.237,08 €
Ernährung	-233,43 €
Gesundheit	-99,09 €
Hobby	-399,95 €
Mobilität	-200,84 €
Wohnen	-6.691,98 €

3.1 3.2

1	MainForm (Tabelle: <i>Kasse</i>)	1.1	SplitKategorie (Tabelle: <i>rel_Kasse_Kategorie</i>)
2	Filter (Tabelle: <i>Filter</i>)	2.1	SaldoKonto (Abfrage: <i>Saldo_Konto</i>)
		2.2	SaldoKategorie (Abfrage: <i>Saldo_Kategorie</i>)
3	MaxDatum (Ansicht: <i>Ansicht_Filter</i>)	3.1	SaldoKonto (Abfrage: <i>Saldo_Konto</i>)
		3.2	SaldoKategorie (Abfrage: <i>Saldo_Kategorie</i>)

Makros in Formulareigenschaften		
1	Vor der Datensatzaktion	Module1. <i>ZeileZwischenspeichern</i>
1	Nach der Datensatzaktion	Module1. <i>Aktualisieren</i>
1	Nach dem Datensatzwechsel	Module1. <i>Aktuelles_Buchungsdatum</i>
1.1	Nach der Datensatzaktion	Module1. <i>Aktualisieren</i>
Makros in Feldeigenschaften		
1	Schaltfläche Aktueller Monat (Aktion Ausführen)	Module1. <i>Monat_aktuell</i>
1	Schaltfläche Adressat hinzufügen (Aktion Ausführen)	Module1. <i>Adressat_neu</i>
1	Listefeld «Konto» (Status geändert)	Module1. <i>Kontofilter_Feldstart</i>
2	Listefeld «Buchung aus Vorlage» (Modifiziert)	Module1. <i>periodische_Buchung</i>
3	Schaltfläche Formularaufrufen (Aktion Ausführen)	Module1. <i>Formular_starten</i>

In diesem Formular werden schließlich alle Register gezogen. Schon allein die Zuordnung von einzelnen Schaltflächen und des Listefeldes für die Buchung aus einer Vorlage zu verschiedenen Formularen muss intensiver erklärt werden.

Doch zuerst einmal die Funktionalität:

Wird über das Listefeld «Buchung aus Vorlage» ein vorheriger Datensatz ausgesucht, so wird dieser Datensatz mit aktuellem Datum am Schluss der Tabelle von Formular 1 eingefügt. Das Listefeld 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 Listefeld 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 Listefeldes «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 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:

```
1 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",
2 "ID"
3 FROM "Kasse" WHERE "periodisch" = TRUE
4 ORDER BY "Periode" ASC
```

FIREBIRD:

```
1 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",
2 "ID"
3 FROM "Kasse" WHERE "periodisch" = TRUE
4 ORDER BY "Periode" ASC
```

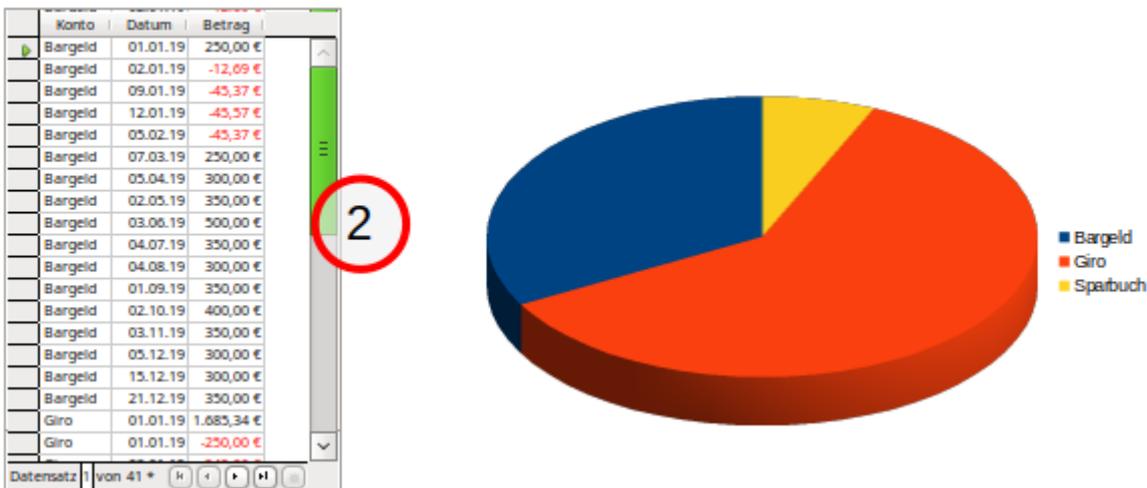
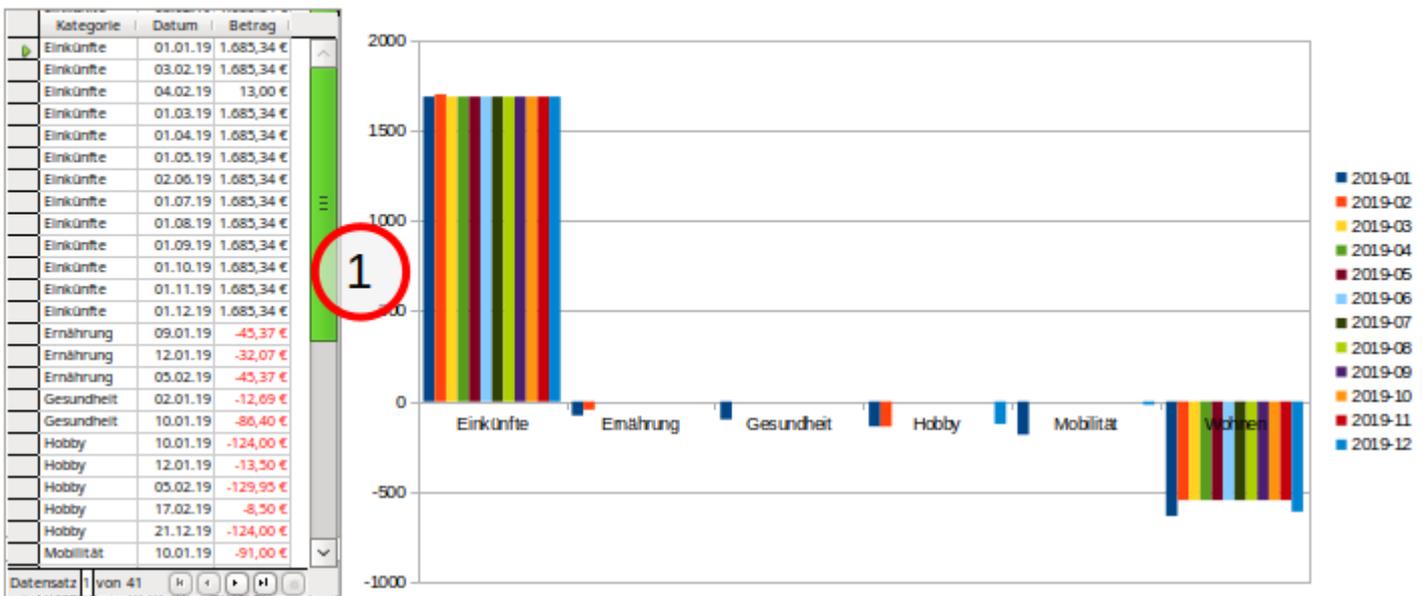
Der Code für die **HSQLDB** wurde vor dem Export so weit wie möglich angepasst. Die Länge des Feldes "Verwendungszweck" wird zuerst für alle betroffenen Felder, bei denen "periodisch" = TRUE ist, ermittelt. Damit kann dann ein entsprechend breiter Platz für diesen Eintrag bei fester Schriftart erzeugt 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

Auswertung vom 01.01.2019 bis 31.12.2019



- | | |
|---|--|
| 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			
<i>Datum</i>	<i>Verwendungszweck</i>	<i>Kategorie</i>	<i>Betrag</i>
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 €
			<u>555,07 €</u>
Februar 2019			
<i>Datum</i>	<i>Verwendungszweck</i>	<i>Kategorie</i>	<i>Betrag</i>
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 €
			<u>969,52 €</u>

Benötigt

Ansicht: *Ansicht_Bericht_Kategorie*

Der Bericht gibt eine Übersicht über die monatlichen Ausgaben. Die entsprechenden Summen dazu werden im Bericht erstellt.

The screenshot shows a software interface for a financial report. At the top, there is a navigation bar with page numbers 1 through 18. Below this, the report title 'Finanzübersicht nach Monaten' is displayed in blue. The interface includes several input fields for filtering: '=StartDat' bis '=EndDat' for the date range, and '=Datum' for a specific date. Below these, a table structure is visible with columns for 'Datum', 'Verwendungszweck', 'Kategorie', and 'Betrag'. There are also summary rows: one for '=SummeBetrag' and a final row for 'Gesamtbetrag: '=SummeBetrag'. The interface uses a color-coded sidebar with green, blue, orange, and green buttons.

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 Datumfeld 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.

Februar 2020			
Datum	Verwendungszweck	Kategorie	Betrag
06.02.20	Markt, Lebensmittel	Ernährung	-45,13 €
			<u>-45,13 €</u>
		Gesamtbetrag:	<u>12.611,79 €</u>

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: <i>Konto_Salden_komplett_eM, Konto_Salden_komplett_Umbuchung_eM</i>
Makro: <i>Aktualisieren</i>

```

1 SUB Aktualisieren_einfach
2 DIM oDoc AS OBJECT
3 DIM oDrawpage AS OBJECT
4 DIM oForm AS OBJECT
5 oDoc = thisComponent
6 oDrawpage = oDoc.drawpage
7 oForm = oDrawpage.forms.getByName("MaxDatum")
8 oForm.reload
9 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: <i>Konto_Salden_komplett, Konto_Salden_komplett_Umbuchung, Buchung_Umbuchung_Salden</i>

Benötigt

Makro: *SplitRest, Aktualisieren_einfach*

```
1 SUB Aktualisieren(oEvent AS OBJECT)
2     SplitRest(oEvent)
3     Aktualisieren_einfach
4 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*

```
1 SUB ZeileZwischenspeichern(oEvent AS OBJECT)
2     DIM oForm AS OBJECT
3     DIM oHiddenFeld AS OBJECT
4     DIM inUmbuchID AS INTEGER
5     DIM stUmbuch AS STRING
6     oForm = oEvent.Source
7     IF InStr(oForm.ImplementationName, "ODatabaseForm") THEN
8         oHiddenFeld = oForm.getByname("Hidden")
9         inUmbuchID = oForm.findColumn("Umbuch_Konto_ID")
10        stUmbuch = oForm.getString(inUmbuchID)
11        IF oEvent.Action = 3 OR stUmbuch <>"" THEN
12            oHiddenFeld.HiddenValue = "no"
13        ELSEIF oEvent.Action = 1 THEN
14            oHiddenFeld.HiddenValue = oForm.RowCount + 1
15        ELSE
16            oHiddenFeld.HiddenValue = oForm.Bookmark()
17        END IF
18    END IF
19 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: <i>Kasse, Kasse_Umbuchung, Konto, Konto_Salden</i>
Makro: <i>Aktualisieren</i>

Benötigt
Tabelle: <i>Kasse, rel_Kasse_Kategorie</i>

```

1 SUB Splitrest(oEvent AS OBJECT)
2   DIM oDoc AS OBJECT
3   DIM oForm AS OBJECT
4   DIM oSubForm AS OBJECT
5   DIM oFeld AS OBJECT
6   DIM oHiddenFeld AS OBJECT
7   DIM oController AS OBJECT
8   DIM oView AS OBJECT
9   DIM oDatenquelle AS OBJECT
10  DIM oVerbindung AS OBJECT
11  DIM oSQL_Anweisung AS OBJECT
12  DIM stSql AS STRING
13  DIM oSQL_Anweisung2 AS OBJECT
14  DIM stSql2 AS STRING
15  DIM oAbfrageergebnis AS OBJECT
16  DIM inID AS INTEGER
17  DIM loID AS LONG
18  DIM inBetrag AS INTEGER
19  DIM doBetrag AS DOUBLE
20  DIM doRest AS DOUBLE
21  oDoc = thisComponent
22  IF oEvent.Source.HasByName("Hidden") THEN
23    oForm = oEvent.Source
24  ELSE
25    oForm = oEvent.Source.Parent
26  END IF
27  oHiddenFeld = oForm.GetByName("Hidden")
28  IF oHiddenFeld.HiddenValue = "no" THEN
29    EXIT SUB
30  END IF
31  IF oHiddenFeld.HiddenValue <> "" THEN

```

```

32     oForm.absolute(oHiddenFeld.HiddenValue)
33     oHiddenFeld.HiddenValue = ""
34 END IF
35 oDatenquelle = ThisComponent.Parent.CurrentController
36 If NOT (oDatenquelle.isConnected()) THEN
37     oDatenquelle.connect()
38 END IF
39 oVerbindung = oDatenquelle.ActiveConnection()
40 oSQL_Anweisung = oVerbindung.createStatement()
41 oSQL_Anweisung2 = oVerbindung.createStatement()
42 inID = oForm.findColumn("ID")
43 loID = oForm.getLong(inID)
44 IF oHiddenFeld.Tag <> "" THEN
45     stSql = "SELECT ""Kategorie_ID"", ""Betrag"" FROM ""rel_Kasse_Kategorie""
46         WHERE ""Kasse_ID"" = '"+oHiddenFeld.Tag+'''"
47     oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
48     oHiddenFeld.Tag = ""
49     WHILE oAbfrageergebnis.next
50         doBetrag = oAbfrageergebnis.getDouble(2)
51         stSql2 = "INSERT INTO ""rel_Kasse_Kategorie"" (""Kasse_ID"",
52             ""Kategorie_ID"", ""Betrag"") VALUES
53             ('"+loID+''', '"+oAbfrageergebnis.getShort(1)+''', '"+str(doBetrag)+''')'"
54         oSQL_Anweisung2.executeUpdate(stSql2)
55     WEND
56 END IF
57 stSql = "SELECT ""a"". ""Betrag"" - COALESCE((SELECT SUM(""Betrag"")
58     FROM ""rel_Kasse_Kategorie"" WHERE ""Kasse_ID"" = ""a"". ""ID""), 0.00)
59     AS ""Wert"" FROM ""Kasse"" AS ""a"" WHERE ""a"". ""ID"" = '"+loID+'''"
60 oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
61 WHILE oAbfrageergebnis.next
62     doRest = oAbfrageergebnis.getDouble(1)
63 WEND
64 oSubForm = oForm.getByname("SplitKategorie")
65 oFeld = oSubForm.getByname("tblKategorieBetrag")
66 oController = oDoc.getCurrentController()
67 oView = oController.getControl(oFeld)
68 oView.setFocus
69 wait 500
70 IF doRest <> 0 THEN
71     inBetrag = oSubForm.findColumn("Betrag")
72     oSubForm.MoveToInsertRow
73     oSubForm.UpdateDouble(inBetrag, doRest)
74 END IF
75 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 Unter-

formular 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*

```
1 SUB periodische_Buchung(oEvent AS OBJECT)
2   DIM oDoc AS OBJECT
3   DIM oDrawpage AS OBJECT
4   DIM oForm AS OBJECT
5   DIM oFeld AS OBJECT
6   DIM oHiddenFeld AS OBJECT
7   DIM oDatenquelle AS OBJECT
8   DIM oVerbindung AS OBJECT
9   DIM oSQL_Anweisung AS OBJECT
```

```

10 DIM stSql AS STRING
11 DIM oAbfrageergebnis AS OBJECT
12 DIM loID AS LONG
13 DIM loIDNew AS LONG
14 DIM inSave AS INTEGER
15 DIM unoDate
16 oFeld = oEvent.Source.Model
17 IF oFeld.CurrentValue = "" THEN
18     EXIT SUB
19 END IF
20 loID = oFeld.CurrentValue
21 oDoc = thisComponent
22 oDrawpage = oDoc.drawpage
23 oForm = oDrawpage.forms.getByName("MainForm")
24 oForm.moveToInsertRow
25 oHiddenFeld = oForm.getByName("Hidden")
26 oHiddenFeld.Tag = loID
27 oDatenquelle = ThisComponent.Parent.CurrentController
28 IF NOT (oDatenquelle.isConnected()) THEN
29     oDatenquelle.connect()
30 END IF
31 oVerbindung = oDatenquelle.ActiveConnection()
32 oSQL_Anweisung = oVerbindung.createStatement()
33 stSql = "SELECT ""Betrag"", ""Konto_ID"", ""Adressat_ID"", ""Verwendungszweck"",
        ""Umbuch_Konto_ID"" FROM ""Kasse"" WHERE ""ID"" ='" + loID + "'"
34 oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
35 WHILE oAbfrageergebnis.next
36     oForm.UpdateDouble(oForm.findColumn("Betrag"), oAbfrageergebnis.getDouble(1))
37     oForm.UpdateLong(oForm.findColumn("Konto_ID"), oAbfrageergebnis.getLong(2))
38     oForm.UpdateLong(oForm.findColumn("Adressat_ID"), oAbfrageergebnis.getLong(3))
39     IF oAbfrageergebnis.isNull() THEN
40         oForm.UpdateNULL(oForm.findColumn("Adressat_ID"), NULL)
41     END IF
42     oForm.UpdateString(oForm.findColumn("Verwendungszweck"),
43         oAbfrageergebnis.getString(4))
44     oForm.UpdateLong(oForm.findColumn("Umbuch_Konto_ID"),
45         oAbfrageergebnis.getLong(5))
46     IF oAbfrageergebnis.isNull() THEN
47         oForm.UpdateNULL(oForm.findColumn("Umbuch_Konto_ID"), NULL)
48     END IF
49 WEND
50 inSave = MsgBox("Soll der Datensatz mit aktuellem Datum abgespeichert und
        die Kategorien hinzugefügt werden?", 36, "Buchungsübernahme")
51 IF inSave = 6 THEN
52     unoDate = createUnoStruct("com.sun.star.util.Date")
53     unoDate.Year = Year(Now())
54     unoDate.Month = Month(Now())
55     unoDate.Day = Day(Now())
56     oForm.UpdateDate(oForm.findColumn("Datum"), unoDate)
57     IF oForm.isNew THEN
58         oForm.insertRow()
59     ELSE
60         oForm.updateRow()
61     END IF
62 END IF
63 oSQL_Anweisung.executeUpdate("UPDATE ""Filter"" SET ""Kasse_ID"" = NULL WHERE
        ""ID"" = TRUE")
64 oFeld.Parent.reload
65 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 Listefeld 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 werden, 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. **oAbfrageergebnis.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*

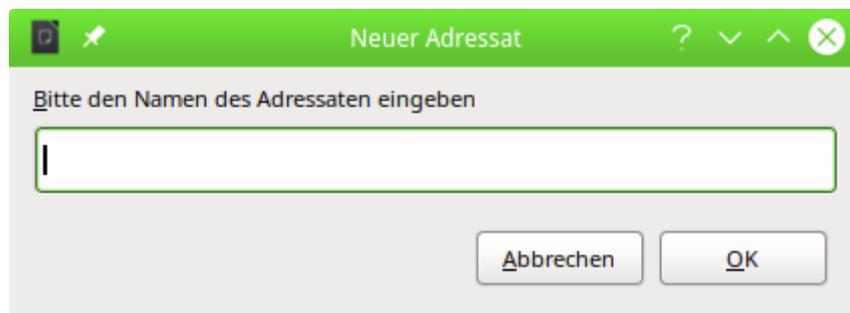
```
1 SUB Adressat_neu(oEvent AS OBJECT)
2   DIM oForm AS OBJECT
```

```

3  DIM oListbox AS OBJECT
4  DIM stInput AS STRING
5  DIM oDatenquelle AS OBJECT
6  DIM oVerbindung AS OBJECT
7  DIM oSQL_Anweisung AS OBJECT
8  DIM oSQL_Anweisung2 AS OBJECT
9  DIM stSql AS STRING
10 DIM oAbfrageergebnis AS OBJECT
11 stInput = InputBox("Bitte den Namen des Adressaten eingeben","Neuer Adressat")
12 stInput = Trim(stInput)
13 IF stInput = "" THEN
14     MsgBox "Der Eintrag für den Adressaten ist leer."
15     EXIT SUB
16 END IF
17 oDatenquelle = ThisComponent.Parent.CurrentController
18 IF NOT (oDatenquelle.isConnected()) THEN
19     oDatenquelle.connect()
20 END IF
21 oVerbindung = oDatenquelle.ActiveConnection()
22 oSQL_Anweisung = oVerbindung.createStatement()
23 oSQL_Anweisung2 = oVerbindung.createStatement()
24 stSql = "SELECT ""Name"" FROM ""Adressat"" WHERE ""Name"" = '"+stInput+"'"
25 oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
26 IF oAbfrageergebnis.Next THEN
27     MsgBox "Der Adressat ""'+stInput+'"" ist schon vorhanden."
28 ELSE
29     oSQL_Anweisung2.executeUpdate("INSERT INTO ""Adressat"" (""Name"" ) VALUES
        ('" + stInput + "')")
30     oForm = oEvent.Source.Model.Parent
31     oListbox = oForm.getByName("MainForm_Grid").getByName("lboAdressat")
32     oListbox.refresh
33 END IF
34 END SUB

```

Am Anfang fehlen noch viele Adressaten. Hier soll auf einfache Weise wenigstens ein Name für den Adressaten abgefragt werden, damit eine Eingabe zügig erfolgen kann.



Solch ein Eingabefeld wird in der Zeile 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: <i>Buchung_Umbuchung_Salden</i>	
1	SUB Monat_aktuell(oEvent AS OBJECT)
2	DIM oForm AS OBJECT
3	oForm = oEvent.Source.Model.Parent
4	oForm.Filter = "EXTRACT(YEAR FROM ""Datum"") = EXTRACT(YEAR FROM CURRENT_DATE) AND EXTRACT(MONTH FROM ""Datum"") = EXTRACT(MONTH FROM CURRENT_DATE)"
5	oForm.ApplyFilter = True
6	oForm.reload
7	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: <i>Buchung_Umbuchung_Salden</i>	
1	SUB Formular_starten(oEvent AS OBJECT)
2	DIM stFormular AS STRING
3	stFormular = oEvent.Source.Model.Tag
4	ThisDatabaseDocument.FormDocuments.getByname(stFormular).open
5	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: <i>Buchung_Umbuchung_Salden</i>	
Benötigt	
Tabelle:	<i>Filter</i>
Makro:	<i>Kontoliste_filtern</i>
1	SUB Aktuelles_Buchungsdatum(oEvent AS OBJECT)
2	DIM oForm AS OBJECT

```

3   DIM oDatenquelle AS OBJECT
4   DIM oVerbindung AS OBJECT
5   DIM oSQL_Anweisung AS OBJECT
6   DIM stSql AS STRING
7   DIM oDoc AS OBJECT
8   DIM oDrawpage AS OBJECT
9   DIM oForm2 AS OBJECT
10  DIM oFeld AS OBJECT
11  DIM inID AS INTEGER
12  oForm = oEvent.Source
13  IF oForm.IsNew THEN
14      stSql = "UPDATE ""Filter"" SET ""DatensatzDatum"" = CURRENT_DATE
              WHERE ""ID"" = TRUE"
15  ELSE
16      stSql = "UPDATE ""Filter"" SET ""DatensatzDatum"" = ' " +
              oForm.getString(oForm.findColumn("Datum")) + "' WHERE ""ID"" = TRUE"
17  END IF
18  oDatenquelle = ThisComponent.Parent.CurrentController
19  If NOT (oDatenquelle.isConnected()) THEN
20      oDatenquelle.connect()
21  END IF
22  oVerbindung = oDatenquelle.ActiveConnection()
23  oSQL_Anweisung = oVerbindung.createStatement()
24  oSQL_Anweisung.executeUpdate(stSql)
25  oDoc = thisComponent
26  oDrawpage = oDoc.drawpage
27  oForm2 = oDrawpage.forms.getByName("Filter")
28  oForm2.reload
29  inID = oForm.getInt(oForm.findColumn("Konto_ID"))
30  oFeld = oForm.getByName("Kasse_Grid").getByName("lboUmbuchID")
31  Kontoliste_filtrern(inID,oFeld)
32  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_filtrern*

```

1  SUB Kontofilter_Formstart(oEvent AS OBJECT)
2      DIM oForm AS OBJECT

```

```

3   DIM oFeld AS OBJECT
4   DIM inID AS INTEGER
5   oForm = oEvent.Source
6   inID = oForm.getInt(oForm.findColumn("Konto_ID"))
7   oFeld = oForm.getByName("Kasse_Grid").getByName("lboUmbuchID")
8   Kontoliste_filtern(inID,oFeld)
9   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*

```

1   SUB Kontofilter_Feldstart(oEvent AS OBJECT)
2   DIM oFeldStart AS OBJECT
3   DIM oFeld AS OBJECT
4   DIM inID AS INTEGER
5   oFeldStart = oEvent.Source.Model
6   inID = oFeldStart.ValueItemList(oEvent.Selected)
7   oFeld = oFeldStart.Parent.getByName("lboUmbuchID")
8   Kontoliste_filtern(inID,oFeld)
9   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*

```

1   SUB Kontoliste_filtern(inID,oFeld)
2   DIM stSql(0) AS STRING

```

```

3   stSql(0) = "SELECT ""Konto"", ""ID"" FROM ""Konto"" WHERE ""ID"" <> "+inID+"
      ORDER BY ""Konto"""
4   oFeld.ListSource = stSql
5   oFeld.refresh
6   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*

```

1   SUB ChangeData(oEvent AS OBJECT)
2   DIM oDiag AS OBJECT
3   DIM oDatasource AS OBJECT
4   DIM oConnection AS OBJECT
5   DIM oSQL_Command AS OBJECT
6   DIM oResult AS OBJECT
7   DIM oForm AS OBJECT
8   DIM oHiddenControl AS OBJECT
9   DIM stSql AS STRING
10  DIM stRow AS STRING
11  DIM stType AS STRING
12  DIM i AS INTEGER
13  DIM k AS INTEGER
14  DIM x AS INTEGER
15  DIM n AS INTEGER
16  DIM aNewData(0)
17  DIM aNewRowDescription(0)
18  DIM aTmp() AS DOUBLE
19  DIM aNewColumnDescription()
20  DIM aType()
21  DIM arView()
22  DIM arDiag()
23  oForm = oEvent.Source
24  oHiddenControl = oForm.getByName("Chart")
25  arView = Split(oHiddenControl.Tag, ",")
26  arDiag = Split(oHiddenControl.HiddenValue, ",")
27  FOR n = LBound(arView()) TO UBound(arView())
28      stSql = "SELECT * FROM """+arView(n)+"""
29      oDatasource = ThisComponent.Parent.CurrentController
30      IF NOT (oDatasource.isConnected()) THEN
31          oDatasource.connect()
32      END IF
33      oConnection = oDatasource.ActiveConnection()
34      oSQL_Command = oConnection.createStatement()
35      oResult = oSQL_Command.executeQuery(stSql)
36      i = 0
37      k = 0
38      x = 0
39      WHILE oResult.next
40          stRow = oResult.getString(1)
41          stType = oResult.getString(4)
42          IF aNewRowDescription(i) = stRow THEN
43              ReDim Preserve aNewColumnDescription(k)
44              ReDim Preserve aTmp(k)
45          ELSE
46              IF x > 0 THEN

```

```

47         i = i + 1
48     ELSE
49         x = 1
50     END IF
51     ReDim Preserve aNewRowDescription(i)
52     ReDim Preserve aNewData(i)
53     k = 0
54     ReDim Preserve aNewColumnDescription(k)
55     ReDim aTmp(k)
56 END IF
57 aNewRowDescription(i) = stRow
58 aNewColumnDescription(k) = oResult.getString(2)
59 aTmp(k) = oResult.getDouble(3)
60 aNewData(i) = aTmp()
61 k = k + 1
62 WEND
63 oDiag = thisComponent.EmbeddedObjects.getByName(arDiag(n))
64 oXCOEO = oDiag.ExtendedControlOverEmbeddedObject
65 oXCOEO.changeState(4)
66 IF stType <> "XY" THEN
67     oDiag.model.Data.setData(aNewData)
68 END IF
69 oDiag.model.DataProvider.setData(aNewData)
70 oDiag.model.DataProvider.setRowDescriptions(aNewRowDescription)
71 oDiag.model.DataProvider.setColumnDescriptions(aNewColumnDescription)
72 oDiag.Component.setmodified(true)
73 oXCOEO.changeState(1)
74 NEXT
75 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.

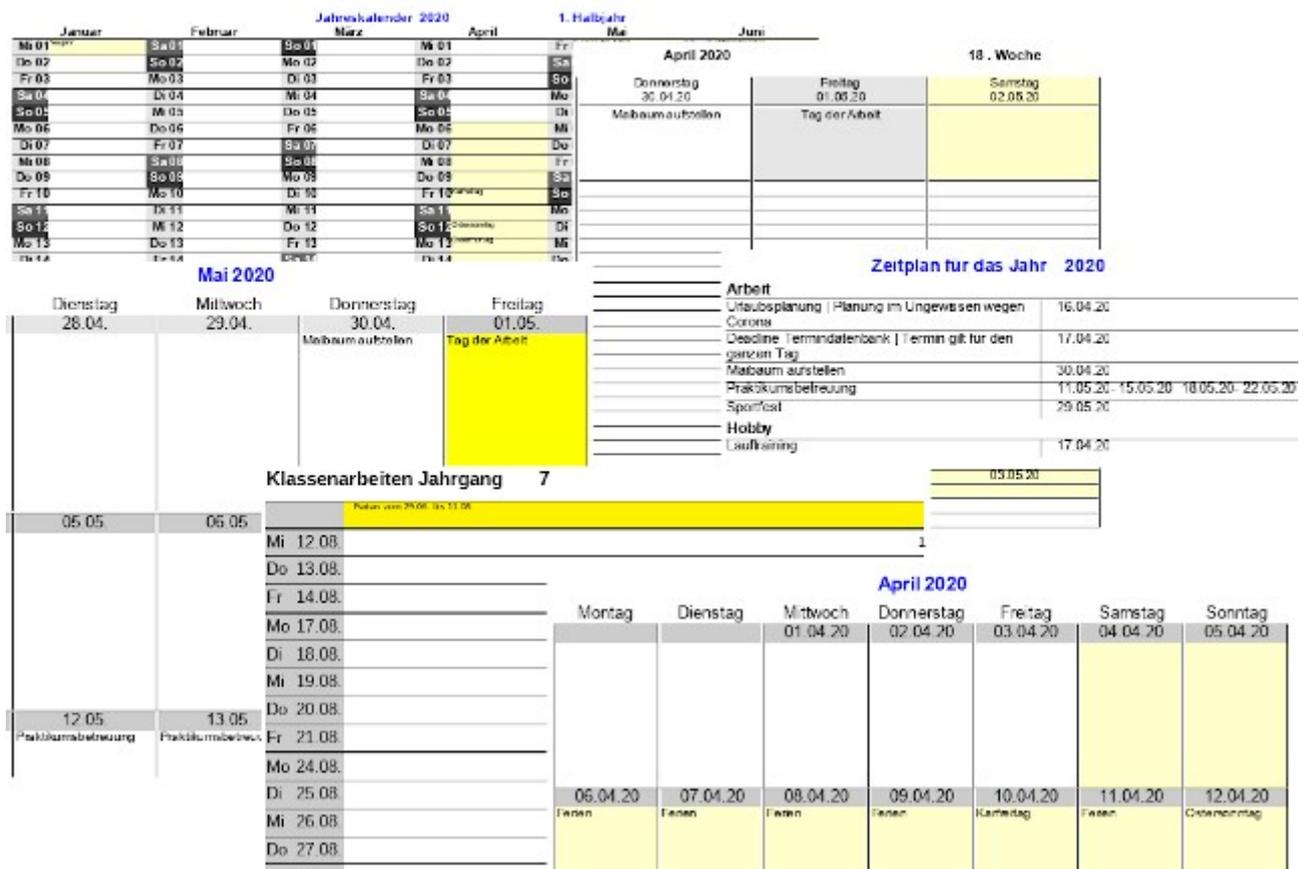
Nach der Änderung wird der Status des Diagramms auf '1' gesetzt, da ansonsten das Diagramm die ganze Zeit im Bearbeitungsmodus erscheint.

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.

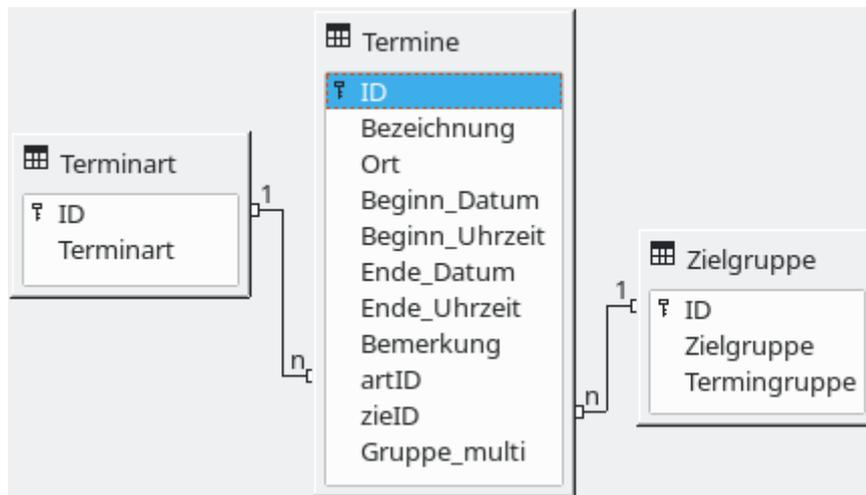
9 Beispieldatenbank Beispiel_Termine.odb

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.

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:	<i>Ansicht_Kalender_Export, Ansicht_Kalender_ohne_Ferien_Export, Ansicht_Termine_Monat_Jahre</i>
Abfrage:	<i>Termine_Gruppe_multi, Terminuebersicht</i>
Makro:	<i>Kalender, Zeitplan, Monat, GruppeMultiUebersicht, Termindaten_uebertragen, Tabellenstart, Ical_Import</i>
Formular, Bericht::	keine direkt

Feldname	Feldtyp	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
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 DESCRIPTION
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
zielID	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: $2^0 = 1$ $2^1 = 2$ $2^2 = 4$ $2^3 = 8$ 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

--

Feldname	Feldtyp	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	Feldtyp	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
Termingruppe	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	Feldtyp	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 Beginn-datum. Eingabe erforderlich: Nein

Diese Tabelle wird direkt editiert. Sie wird über Formulare nicht beeinflusst und nur bei den Berichten berücksichtigt.

Filtertabelle

Datenziel	
Ansicht:	<i>Ansicht_Gruppe_multi, Ansicht_Kalender_Export, Ansicht_Kalender_ohne_Ferien_Export, Ansicht_Termine_Monat_Jahre</i>
Formular:	<i>Terminart_Termin</i> (Berichte, Export), <i>Termine_Kalenderübersicht</i> (Monat, Berichte, Export), <i>Termine_Monat_Terminart</i> (Berichte, Export), <i>Termine_Tabelleneingabe</i> (Monat, Berichte, Export)
Makro:	<i>Bericht_Start, Monatsansicht, Monatsansicht_aktualisieren, DialogEnde, Ical_Import</i>
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 Ausdrücke auch halbjahresweise erfolgen können. Eingabe erforderlich: Nein
Ferien	Ja/Nein	Sollen Ferien beim Export mit exportiert werden? Eingabe erforderlich: Nein

BerichtZielgruppelID	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
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: <i>Halbjahreskalender_A4_quer</i>
Makro: <i>Halbjahreskalender</i>
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	Feldtyp	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	Feldtyp	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	Feldtyp	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
F1 ... F7	Text	Formatierung (Ferien, Feiertage) , 1 Zeichen
Monat_Sort	Tiny Integer	Sortierung bei Jahresübergängen sicherstellen

tmp_Wochenkalender_half

Datenziel
Bericht: Wochenkalender_half
Makro: Wochenkalender_half
Ansicht, Abfrage, 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 ... 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

Feldname	Feldtyp	Beschreibung
F1 ... F4	Text	Formatierung (Ferien, Feiertage) , 1 Zeichen

Ansichten

Ansicht_Gruppe_multi

Datenquelle
Tabelle: <i>tmp_Gruppe_multi, Filter, Ferien</i>

Datenziel
Abfrage: <i>Liste_Gruppe_Multi</i>
Ansicht, Formular, Bericht: keine direkt

```

76 SELECT "Datum" AS "Sort",
77 CASE WHEN DAYOFWEEK("Datum")=2 THEN 'Mo'
78 WHEN DAYOFWEEK("Datum")=3 THEN 'Di'
79 WHEN DAYOFWEEK("Datum")=4 THEN 'Mi'
80 WHEN DAYOFWEEK("Datum")=5 THEN 'Do'
81 WHEN DAYOFWEEK("Datum")=6 THEN 'Fr'
82 END AS "Tag",
83 RIGHT('0' || EXTRACT(DAY FROM "Datum"),2) || '.' || RIGHT('0' || EXTRACT (MONTH FROM
"Datum"),2) || '.' AS "Dat",
84 "Bezeichnung", "Format", EXTRACT (YEAR FROM "Datum") || RIGHT(0 || WEEK("Datum"),2) AS
"W1", "Gruppe_multi"
85 FROM "tmp_Gruppe_multi"
86 WHERE "Datum" BETWEEN ( SELECT "Startjahr" FROM "Filter" WHERE "ID" = TRUE ) || '-08-
01' AND (SELECT "Startjahr" FROM "Filter" WHERE "ID" = TRUE ) + 1 || '-08-01'
87 AND DAYOFWEEK("Datum") BETWEEN 2 AND 6
88 AND ( NOT "Bezeichnung" = 'Ferien' OR "Bezeichnung" IS NULL )
89 UNION
90 SELECT "Beginn_Datum",
91 NULL AS "Tag",
92 NULL AS "Dat",
93 '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) || '.' ,
94 '1',
95 NULL,
96 (SELECT DISTINCT "Gruppe_multi" FROM "tmp_Gruppe_multi")
97 FROM "Ferien"
98 WHERE ("Beginn_Datum" BETWEEN ( SELECT "Startjahr" FROM "Filter" WHERE "ID" = TRUE )
|| '-08-01' AND ( SELECT "Startjahr" FROM "Filter" WHERE "ID" = TRUE ) + 1 || '-08-
01' )
99 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

```
1 SELECT RIGHT('0000000' || "ID", 8) AS "UID",
2 "Ort" AS "Location", "Bezeichnung" AS "Summary", "Bemerkung" AS "Description",
3 COALESCE(
4 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 ),
5 EXTRACT( YEAR FROM "Beginn_Datum" ) || RIGHT( '0' || EXTRACT( MONTH FROM
  "Beginn_Datum" ), 2 ) || RIGHT( '0' || EXTRACT( DAY FROM "Beginn_Datum" ), 2 )) AS
  "DtStart",
6 COALESCE (
7 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 ),
8 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 ),
9 EXTRACT( YEAR FROM "Ende_Datum" ) || RIGHT( '0' || EXTRACT( MONTH FROM
  "Ende_Datum" ), 2 ) || RIGHT( '0' || EXTRACT( DAY FROM "Ende_Datum" )+1, 2 ),
```

```

10 EXTRACT( YEAR FROM "Beginn_Datum" ) || RIGHT( '0' || EXTRACT( MONTH FROM
"Beginn_Datum" ), 2 ) || RIGHT( '0' || EXTRACT( DAY FROM "Beginn_Datum" )+1, 2 ) AS
"DtEnd",
11 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"
12 FROM "Termine"
13 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")
14 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")
15 UNION
16 SELECT RIGHT('000000' || "ID" || 'F',8) AS "UID", '', 'Ferien', '',
17 EXTRACT( YEAR FROM "Beginn_Datum" ) || RIGHT( '0' || EXTRACT( MONTH FROM
"Beginn_Datum" ), 2 ) || RIGHT( '0' || EXTRACT( DAY FROM "Beginn_Datum" ), 2 ) AS
"DtStart",
18 COALESCE(
19 EXTRACT( YEAR FROM "Ende_Datum" ) || RIGHT( '0' || EXTRACT( MONTH FROM
"Ende_Datum" ), 2 ) || RIGHT( '0' || EXTRACT( DAY FROM "Ende_Datum" )+1, 2 ) ,
20 EXTRACT( YEAR FROM "Beginn_Datum" ) || RIGHT( '0' || EXTRACT( MONTH FROM
"Beginn_Datum" ), 2 ) || RIGHT( '0' || EXTRACT( DAY FROM "Beginn_Datum" )+1, 2 ) ) AS
"DtEnd",
21 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"
22 FROM "Ferien"
23 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")
24 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")
25 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 zusammgebaut 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

```
1 SELECT RIGHT('0000000' || "ID", 8) AS "UID",
2 "Ort" AS "Location", "Bezeichnung" AS "Summary", "Bemerkung" AS "Description",
3 COALESCE(
4 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 ),
5 EXTRACT( YEAR FROM "Beginn_Datum" ) || RIGHT( '0' || EXTRACT( MONTH FROM
  "Beginn_Datum" ), 2 ) || RIGHT( '0' || EXTRACT( DAY FROM "Beginn_Datum" ), 2 )) AS
  "DtStart",
6 COALESCE (
7 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 ),
8 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 ),
```

```

9  EXTRACT( YEAR FROM "Beginn_Datum" ) || RIGHT( '0' || EXTRACT( MONTH FROM
"Beginn_Datum" ), 2 ) || RIGHT( '0' || EXTRACT( DAY FROM "Beginn_Datum" )+1, 2 ) AS
"DtEnd",
10 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"
11 FROM "Termine"
12 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")
13 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")
14 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

```

1  SELECT DISTINCT "Jahr", "Jahr"||', '||RIGHT('0'||"Monat_Z",2) AS "Monat_Zahl",
2  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",
3  "Monat_Z" FROM
4  (SELECT EXTRACT (YEAR FROM "Beginn_Datum" ) AS "Jahr", EXTRACT (MONTH FROM
"Beginn_Datum" ) AS "Monat_Z" FROM "Termine"
5  UNION ALL
6  SELECT EXTRACT (YEAR FROM "Ende_Datum" ) AS "Jahr", EXTRACT (MONTH FROM
"Ende_Datum" ) AS "Monat_Z" FROM "Termine" WHERE NOT "Ende_Datum" IS NULL)
7  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_half
Ansicht, Abfrage, Formular, Bericht: keine direkt

```

1 SELECT "tmp_Wochenkalender".*
2 FROM ( SELECT "Woche", MIN( "ID" ) AS "ID_einzeln" FROM "tmp_Wochenkalender" AS
   "tmp_Wochenkalender" GROUP BY "Woche", "Jahr" ) AS "Woche_ID", "tmp_Wochenkalender"
3 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_half" 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

```

1 SELECT COUNT( "ID" ) AS "MoDiMiDoFr",
2 ( 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",
3 ( 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",
4 ( 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 "MoMiDoFr",
5 EXTRACT (YEAR FROM "Datum" ) AS "Jahr", EXTRACT (MONTH FROM "Datum" ) AS "Monat",
6 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"
7 FROM "tmp_Kalender" AS "a"
8 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](#)

```
1 SELECT "a".*,
2 ( SELECT COUNT( DISTINCT ( "w1" ) ) FROM "Ansicht_Gruppe_multi" WHERE "w1" <=
  "a"."w1" ) AS "Woche"
3 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)

```
1 SELECT "Termine".*,
2 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",
3 EXTRACT( YEAR FROM "Beginn_Datum" ) AS "Jahr", EXTRACT( MONTH FROM "Beginn_Datum" )
  AS "Monat"
4 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 2^0 , 2^1 , 2^2 , 2^3 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 2^2 . 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](#) (SubTerminMonat, SubTerminart)

```
1 SELECT "Termine".*,
2 EXTRACT( MONTH FROM "Beginn_Datum" ) AS "Monat",
3 EXTRACT( YEAR FROM "Beginn_Datum" ) AS "Jahr",
4 FROM "Termine"
5 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: <i>tmp_Wochenkalender</i>

Datenziel
Bericht: <i>Monatskalender_Mo_Fr</i>

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.

```

1 SELECT "tmp_Wochenkalender".*
2 FROM "tmp_Wochenkalender"
3 WHERE EXTRACT( MONTH FROM "D_1" ) = "Monat_Zahl"
4 OR EXTRACT( MONTH FROM "D_2" ) = "Monat_Zahl"
5 OR EXTRACT( MONTH FROM "D_3" ) = "Monat_Zahl"
6 OR EXTRACT( MONTH FROM "D_4" ) = "Monat_Zahl"
7 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

Terminartenbank - Eingabeformular

Terminart: 1

Datensatz: von 3

	Bezeichnung	Ort	Beginn Dat.	Beginn Zeit	Ende Dat.	Ende Zeit	Bemerkung	Zielgruppe
	Urlaubsplanung	Raum 2	16.04.20	12:00			Planung im Ungewissen we	Alle
	Deadline Terminatenbank	Arbeitszimmer	17.04.20				Termin gilt für den ganzen	Alle
	Maibaum aufstellen	Draußen	20.04.20					Alle
	Klassenarbeit Mathe 7	Schule	05.05.20	07:55	05.05.20	08:40		Robert
	Tabellenkalkulation in Physik 8	Schule	07.05.20	09:50	07.05.20	11:25		Robert
	Praktikumsbetreuung	Schule	11.05.20		15.05.20			Alle
	Praktikumsbetreuung	Schule	18.05.20		22.05.20			Alle
	Sportfest	Schule	29.05.20					Alle

Datensatz: 4 von 8 (1)

Terminübersicht zum ausgewählten Monat

Beginn:	Uhrzeit	Ende:	Uhrzeit	Bezeichnung	Terminart
05.05.20	07:55	05.05.20	08:40	Klassenarbeit Mathe 7	
07.05.20	09:50	07.05.20	11:25	Tabellenkalkulation in Physik 8	
11.05.20		15.05.20		Praktikumsbetreuung	
18.05.20		22.05.20		Praktikumsbetreuung	
29.05.20				Sportfest	

Datensatz: 1 von 6

Übersicht zum ausgewählten Ort

Datum	Bezeichnung
05.05.20	Klassenarbeit Mathe 7
07.05.20	Tabellenkalkulation in Physik 8
11.05.20	Praktikum
18.05.20	Praktikumsbetreuung
29.05.20	Sportfest

Datensatz: 1 von 5

Kalenderansicht

von bis Kalender 2 Monate Montag - Freitag Zielgruppe: Jahrgang:

Ical-Export

von bis Ferien 3 exportieren

1	MainForm (Tabelle: <i>Terminart</i>)	1.1	SubTermine (Abfrage: <i>Termine_Gruppe_multi</i>)	1.1.1	SubTerminMonat (Abfrage: <i>Terminuebersicht</i>)
				1.1.2	SubTerminOrt (Abfrage: <i>Terminuebersicht</i>)
2	Berichte (Tabelle: <i>Filter</i>)				
3	Export (Tabelle: <i>Filter</i>)				

Makros in Formulareigenschaften		
keine		
Makros in Feldeigenschaften		
1.1	Beginn Dat. (Text modifiziert)	Eingabe. <i>DatenAktualisieren_Terminort_Termine</i>
1.1	Checkboxen "Jahrgänge" (Status geändert)	Eingabe. <i>GruppeMultiSpeichern</i>
2	Schaltfläche OK (Aktion ausführen)	Abfragen. <i>Bericht_Start</i>
3	Schaltfläche Exportieren (Aktion ausführen)	Export. <i>Export_Start</i>

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

Terminatenbank - Eingabeformular

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. Terminaten_uebertragen
1	Beginn Datum (Fokuserhalt)	Eingabe. Anfangswert_Datum
1	Beginn Datum (Fokusverlust)	Eingabe. Monatsansicht_aktualisieren
1	Checkboxen Jahrgänge (Status geändert)	Eingabe. GruppeMultiSpeichern
2	Listefeld 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

Termin Datenbank - Eingabeformular

Bezeichnung: Sportfest Ort: Schule

Beginn: Datum * 29.05.20 Uhrzeit [] **1**

Ende: Datum [] Uhrzeit []

Terminart: Arbeit Zielgruppe: Alle

Bemerkung: []

Jahrgänge: Jg. 5 Jg. 6 Jg. 7 Jg. 8 Jg. 9 Jg. 10 Jg. 11 Jg. 12 Jg. 13

Datensatz 10 von 11

Terminübersicht zum ausgewählten Monat

Beginn	Uhrzeit	Ende	Uhrzeit	Bezeichnung	Terminart
05.05.20	07:55	05.05.20	08:40	Klassenarbeit Mathe 7	Arbeit
07.05.20	09:50	07.05.20	11:25	Tabellenkalkulation in Physi	Arbeit
11.05.20		15.05.20		Praktikumsbetreuung	Arbeit
18.05.20		22.05.20		Praktikumsbetreuung	Arbeit
29.05.20				Sportfest	Arbeit

Datensatz 1 von 6

Übersicht zur ausgewählten Terminart

Datum	Bezeichnung
16.04.20	Urlaubsplanung
17.04.20	Deadline Termin Datenbank
30.04.20	Maibaum
05.05.20	Klassenarbeit Mathe 7
07.05.20	Tabellenkalkulation in Phy

Datensatz 1 von 8

Kalenderansicht

von 01.01.20 bis 30.06.20 Kalenderart Monate Montag - Freitag **2** Gruppe Robert Jahrgang [] OK

ICal-Export

von 13.04.20 bis 07.06.20 Ferien inkl. **3** exportieren

1	MainForm (Abfrage: <i>Termine_Gruppe_multi</i>)	1.1	SubTermineMonat (Abfrage: <i>Terminuebersicht</i>)
		1.2	SubTerminart (Abfrage: <i>Terminuebersicht</i>)
2	Berichte (Tabelle: <i>Filter</i>)		
3	Export (Tabelle: <i>Filter</i>)		

Makros in Formulareigenschaften		
1	Beim Laden	Eingabe. <i>LetzterDatensatz</i>
Makros in Feldeigenschaften		
1	Beginn Datum (Text modifiziert)	Eingabe. <i>DatenAktualisieren_Termine</i>
1	Listenfeld "Terminart" (Modifiziert)	Eingabe. <i>DatenAktualisieren_Termine</i>
1	Checkboxen Jahrgänge (Status geändert)	Eingabe. <i>GruppeMultiSpeichern</i>

2	Schaltfläche OK (Aktion ausführen)	Abfragen. <i>Bericht_Start</i>
3	Schaltfläche Exportieren (Aktion ausführen)	Export. <i>Export_Start</i>

Das Hauptformular "MainForm" dient hier wieder zur Dateneingabe in einfache Formularfelder. Die Übersicht zu den aktuell eingegebenen Daten vermitteln die Unterformulare "SubTermineMonat" 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

Terminatenbank - Schnelleingabe

1	Monat (Tabelle: <i>Filter</i>)
2	MainForm (Abfrage: <i>Termine_Gruppe_multi</i>)
3	Berichte (Tabelle: <i>Filter</i>)
4	Export (Tabelle: <i>Filter</i>)

Makros in Formulareigenschaften		
keine		
Makros in Feldeigenschaften		
1	Listenfeld Monat (Modifiziert)	Filter. <i>Filter_Abgleich</i>
2	Checkboxen Jahrgänge (Status geändert)	Eingabe. <i>GruppeMultiSpeichern</i>

3	Schaltfläche OK (Aktion ausführen)	Abfragen. <i>Bericht_Start</i>
4	Schaltfläche Exportieren (Aktion ausführen)	Export. <i>Export_Start</i>
5	Schaltfläche Importieren (Aktion ausführen)	Import. <i>Import_Start</i>

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:

```
(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						
Januar	Februar	März	April	1. Halbjahr	Mai	Juni
Mi 01 ^{Asper}	Sa 01	So 01	Mi 01	Fr 01 ^{Tag der Arbeit}	Mo 01 ^{Freitag}	Di 02
Do 02	So 02	Mo 02	Do 02	Sa 02	Di 03	Mi 03
Fr 03	Mo 03	Di 03	Fr 03	So 03	Do 04	Fr 05
Sa 04	Di 04	Mi 04	Sa 04	Mo 04	So 06	Mo 08
So 05	Mi 05	Do 05	So 05	Di 05	Do 07	Di 09
Mo 06	Do 06	Fr 06	Mo 06	Mi 06	So 10	Mi 10
Di 07	Fr 07	Sa 07	Di 07	Do 07	Mo 11 ^{Freitag}	Do 11 ^{Freitag}
Mi 08	Sa 08	So 08	Mi 08	Fr 08	Di 12 ^{Freitag}	Fr 12
Do 09	So 09	Mo 09	Do 09	Sa 09	Mi 13 ^{Freitag}	Sa 13
Fr 10	Mo 10	Di 10	Fr 10 ^{Freitag}	So 10	Do 14 ^{Freitag}	So 14
Sa 11	Di 11	Mi 11	Sa 11 ^{Freitag}	Mo 11 ^{Freitag}	Mo 15	Mo 15
So 12	Mi 12	Do 12	So 12 ^{Freitag}	Di 12 ^{Freitag}	Di 16	Di 16
Mo 13	Do 13	Fr 13	Mo 13 ^{Freitag}	Mi 13 ^{Freitag}	Do 17	Do 17
Di 14	Fr 14	Sa 14	Di 14	Do 14 ^{Freitag}	Mo 18	Mo 18
Mi 15	Sa 15	So 15	Mi 15	Fr 15 ^{Freitag}	Di 19	Di 19
Do 16	So 16	Mo 16	Do 16 ^{12:00 Uhr Sitzplanung Planung im Ungeheissen wegen Corona}	Sa 16	Do 18	Do 18
Fr 17	Mo 17	Di 17	Fr 17 ^{17:30 Lauftraining Quadrate Tennisballenbank}	So 17	Mo 20	Mo 20
Sa 18	Di 18	Mi 18	Sa 18	Mo 18 ^{Freitag}	Di 21	Di 21
So 19	Mi 19	Do 19	So 19	Di 19 ^{Freitag}	Fr 22	Fr 22
Mo 20	Do 20	Fr 20	Mo 20	Mi 20 ^{Freitag}	So 21	So 21
Di 21	Fr 21	Sa 21	Di 21	Do 21 ^{Freitag}	Mo 22	Mo 22
Mi 22	Sa 22	So 22	Mi 22	Fr 22 ^{Freitag}	Di 23	Di 23
Do 23	So 23	Mo 23	Do 23	Sa 23	Mi 24	Mi 24
Fr 24	Mo 24 ^{Freitag}	Di 24	Fr 24	So 24	Do 25	Do 25
Sa 25	Di 25	Mi 25	Sa 25	Mo 25	Fr 26	Fr 26
So 26	Mi 26	Do 26	So 26	Di 26	So 27	So 27
Mo 27	Do 27	Fr 27	Mo 27	Mi 27	Mo 29	Mo 29
Di 28	Fr 28	Sa 28	Di 28	Do 28	Di 30	Di 30
Mi 29	Sa 29	So 29	Mi 29	Fr 29 ^{Freitag}	Mo 31	Mo 31
Do 30	So 30	Mo 30	Do 30 ^{Freitag}	Sa 30	Di 31	Di 31
Fr 31		Di 31		So 31 ^{Freitag}		

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 verkleinert 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
Di 01.09.	.

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 Ausdrücke 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

Mai 2020

Montag 27.04.	Dienstag 28.04.	Mittwoch 29.04.	Donnerstag 30.04.	Freitag 01.05.
			Maibaum aufstellen	Tag der Arbeit
04.05.	05.05.	06.05.	07.05.	08.05.
11.05. Praktikumsbetreuung	12.05. Praktikumsbetreuung	13.05. Praktikumsbetreuung	14.05. Praktikumsbetreuung	15.05. Praktikumsbetreuung

Datenquelle

Abfrage: [Wochenkalender_Mo_Fr](#)

Dieser Kalender war der ursprüngliche Kalenderausdruck des Projekts. Die doch recht dunkle 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 Builder noch die Möglichkeit, die Einträge entsprechend anzupassen. Vielleicht wird etwas an den Formulierungen 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 Terminatenbank 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_half

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.

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 Backup-Ordner geschrieben.

Außerdem wird noch global das Startdatum mit der ersten Formularöffnung festgelegt.

```
1 GLOBAL oVerbindung AS OBJECT
2 GLOBAL daStartDat AS DATE

1 SUB Datenbankstart
2   DIM oDatenquelle AS OBJECT
3   Datenbankbackup(10)
4   oDatenquelle = thisDatabaseDocument.CurrentController
5   IF NOT (oDatenquelle.isConnected()) THEN oDatenquelle.connect()
6   oVerbindung = oDatenquelle.ActiveConnection()
7 END SUB
```

OfficeVersion

Aufruf aus
Makro: <i>Datumswert, Listenfeld_Text, ID_Ermittlung</i>

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 Datumfelder anders ausgelesen. Um die Datenbank kompatibel zu älteren Versionen zu halten muss also die Office-Version gegebenenfalls bestimmt werden können.

```
1 FUNCTION OfficeVersion()
2   DIM aSettings, aConfigProvider
3   DIM aParams2(0) AS NEW com.sun.star.beans.PropertyValue
4   DIM sProvider$, sAccess$
5   sProvider = "com.sun.star.configuration.ConfigurationProvider"
6   sAccess = "com.sun.star.configuration.ConfigurationAccess"
7   aConfigProvider = createUnoService(sProvider)
8   aParams2(0).Name = "nodepath"
9   aParams2(0).Value = "/org.openoffice.Setup/Product"
10  aSettings = aConfigProvider.CreateInstanceWithArguments(sAccess, aParams2())
11  OfficeVersion() = array(aSettings.ooName, aSettings.ooSetupVersionAboutBox)
12 END FUNCTION
```

Der **nodepath** in Zeile 8 stellt den Einstieg in **Extras → Optionen → LibreOffice → Erweitert → Experimenteneinstellungen** 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: <i>Bericht_Start, Anfangswert_Datum, Monatsansicht_aktualisieren</i>

Benötigt
Makro: <i>OfficeVersion</i>

Die Behandlung des Datumswertes hat sich in Abhängigkeit von der LO-Version geändert.

```
1 FUNCTION Datumswert(oFeld AS OBJECT) AS DATE
2   DIM b()
```

```

3  a() = OfficeVersion()
4  b = Split(a(1), ".")
5  IF a(0) = "LibreOffice" AND (b(0) = 4 AND b(1) > 0) OR b(0) > 4 THEN
6    DIM stMonat AS STRING
7    DIM stTag AS STRING
8    stMonat = Right("0" & Trim(Str(oFeld.CurrentValue.Month)), 2)
9    stTag = Right("0" & Trim(Str(oFeld.CurrentValue.Day)), 2)
10   Datumswert = CDateFromIso(oFeld.CurrentValue.Year & stMonat & stTag)
11 ELSE
12   Datumswert = CDateFromIso(oFeld.GetCurrentValue)
13 END IF
14 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 Eintrag 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 Eintrag **Year**, **Month** und **Day**.

Listenfeld_Text

Aufruf aus

Makro: *Bericht_Start*

Benötigt

Makro: *OfficeVersion*

```

1  FUNCTION Listenfeld_Text(oFeld AS OBJECT) AS STRING
2    DIM b()
3    a() = OfficeVersion()
4    b = Split(a(1), ".")
5    IF a(0) = "LibreOffice" AND (b(0) = 4 AND b(1) > 0) OR b(0) > 4 THEN
6      Listenfeld_Text = oFeld.StringItemList(oFeld.SelectedItems(0))
7    ELSE
8      Listenfeld_Text = oFeld.CurrentValue()
9    END IF
10 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*

```
1 FUNCTION ID_Ermittlung(oFeld AS OBJECT) AS INTEGER
2     DIM stInhalt AS STRING
3     DIM b()
4     a() = OfficeVersion()
5     b = Split(a(1), ".")
6     IF a(0) = "LibreOffice" AND (b(0) = 4 AND b(1) > 0) OR b(0) > 4 THEN
7         stInhalt = oFeld.CurrentValue
8     ELSE
9         stInhalt = oFeld.ValueItemList(oFeld.SelectedItems(0))
10    END IF
11    IF IsEmpty(stInhalt) THEN
12        ID_Ermittlung = -1
13    ELSE
14        ID_Ermittlung = Cint(stInhalt)
15    END IF
16 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*

```
1 SUB Kalender(inJahr AS INTEGER, inFerien AS INTEGER, stTermin AS STRING)
2     DIM stDatum AS STRING
3     DIM stDat_Beginn AS STRING
4     DIM stDat_Ende AS STRING
5     DIM stUpdate AS STRING
6     DIM stZeit AS STRING
7     DIM i AS INTEGER
8     DIM d AS DATE
9     DIM d_neu AS DATE
10    DIM d_end AS DATE
11    DIM stBezeichnung AS STRING
12    DIM stBemerkung AS STRING
13    inJahresschleife = inJahr
14    oSQL_Anweisung = oVerbindung.createStatement()
```

```

15 oSQL_Anweisung1 = oVerbindung.createStatement()
16 oSQL_Anweisung2 = oVerbindung.createStatement()
17 stSql = "DELETE FROM ""tmp_Kalender""
18 oSQL_Anweisung.executeUpdate(stSql)
19 oSQL_Anweisung.executeQuery("ALTER TABLE ""tmp_Kalender"" ALTER COLUMN ""ID""
    RESTART WITH 1")
20 FOR inJahr=inJahresschleife TO inJahresschleife+1
21   Feiertage = FeiertageArray(inJahr)
22   d = "01.01."+inJahr
23   d_end = "01.01."+(inJahr+1)
24   stDatum = Date_2_SQLDate(d)
25   d_neu=d
26   WHILE d_neu < d_end
27     stSql = "INSERT INTO ""tmp_Kalender"" (""Datum"") VALUES ('"+stDatum+"') "
28     oSQL_Anweisung.executeUpdate(stSql)
29     d_neu=DateAdd("d",1,d_neu)
30     stDatum = Date_2_SQLDate(d_neu)
31   WEND
32   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"
33   oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
34   WHILE oAbfrageergebnis.next
35     stDat_Beginn = oAbfrageergebnis.getString(1)
36     stDat_Ende = oAbfrageergebnis.getString(2)
37     IF inFerien = 1 THEN
38       stUpdate = """"Bezeichnung"" ='Ferien', ""Format"" = '1'"
39     ELSE
40       stUpdate = """"Format"" = '1'"
41     END IF
42     stSql1 = "UPDATE ""tmp_Kalender"" SET "+stUpdate+" WHERE ""Datum"" =
        '"+stDat_Beginn+"'"
43     oSQL_Anweisung1.executeUpdate(stSql1)
44     IF stDat_Ende <>"" AND stDat_Ende <> stDat_Beginn THEN
45       d_neu=DateAdd("d",1,CDate(stDat_Beginn))
46       WHILE d_neu <= CDate(stDat_Ende)
47         stDatum = Date_2_SQLDate(d_neu)
48         stSql1 = "UPDATE ""tmp_Kalender"" SET "+stUpdate+" WHERE ""Datum"" =
            '"+stDatum+"'"
49         oSQL_Anweisung1.executeUpdate(stSql1)
50         d_neu=DateAdd("d",1,d_neu)
51       WEND
52     END IF
53   WEND
54   FOR i=0 TO 14
55     stDatum = Feiertage(i,1)
56     stBezeichnung = Feiertage(i,0)
57     stSql = "UPDATE ""tmp_Kalender"" SET ""Bezeichnung"" ='"+stBezeichnung+"',
        ""Format"" = '1' WHERE ""Datum"" = '"+stDatum+"'"
58     oSQL_Anweisung.executeUpdate(stSql)
59   NEXT
60   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"
61   oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
62   WHILE oAbfrageergebnis.next
63     stBezeichnung = oAbfrageergebnis.getString(1)
64     stBezeichnung_Dat_Ende = oAbfrageergebnis.getString(1)
65     stDat_Beginn = oAbfrageergebnis.getString(2)
66     stDat_Ende = oAbfrageergebnis.getString(3)
67     stBemerkung = oAbfrageergebnis.getString(4)
68     stZeit = oAbfrageergebnis.getString(5)
69     stZeit = left(stZeit,5)
70     IF stBemerkung <> "" THEN
71       stBezeichnung = stBezeichnung + " " + stBemerkung
72       stBezeichnung_Dat_Ende = stBezeichnung_Dat_Ende + " " + stBemerkung

```

```

73     END IF
74     IF stZeit <> "" THEN
75         stBezeichnung = stZeit + " " + stBezeichnung
76         stBezeichnung_Dat_End = stZeit + " " + stBezeichnung_Dat_End
77     END IF
78     stSql1 = "SELECT ""Bezeichnung"" FROM ""tmp_Kalender"" WHERE ""Datum"" =
           '"+stDat_Beginn+"'
79     oAbfrageergebnis1 = oSQL_Anweisung1.executeQuery(stSql1)
80     WHILE oAbfrageergebnis1.next
81         IF oAbfrageergebnis1.getString(1) <> "" THEN
82             stBezeichnung = oAbfrageergebnis1.getString(1) + CHR(13) +
           stBezeichnung
83         END IF
84     WEND
85     stSql1 = "UPDATE ""tmp_Kalender"" SET ""Bezeichnung"" = '"+stBezeichnung+'
           WHERE ""Datum"" = '"+stDat_Beginn+"'
86     oSQL_Anweisung1.executeUpdate(stSql1)
87     IF stDat_End <> "" AND stDat_End <> stDat_Beginn THEN
88         d_neu = DateAdd("d",1,CDate(stDat_Beginn))
89         WHILE d_neu <= CDate(stDat_End)
90             stDatum = Date_2_SQLDate(d_neu)
91             stSql2 = "SELECT ""Bezeichnung"" FROM ""tmp_Kalender"" WHERE
           ""Datum"" = '"+stDatum+"'
92             oAbfrageergebnis2 = oSQL_Anweisung2.executeQuery(stSql2)
93             WHILE oAbfrageergebnis2.next
94                 IF oAbfrageergebnis2.getString(1) <> "" THEN
95                     stBezeichnung = oAbfrageergebnis2.getString(1) + CHR(13) +
           stBezeichnung_Dat_End
96                 ELSE
97                     stBezeichnung = stBezeichnung_Dat_End
98                 END IF
99             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 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_End** 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*

```
1 SUB Monatskalender(daStart AS DATE, daEnde AS DATE)
2   DIM i AS INTEGER
3   DIM inMonat AS INTEGER
4   DIM inWoche AS INTEGER
5   DIM inMonat_Sort AS INTEGER
6   DIM inTag AS INTEGER
7   DIM inDd AS INTEGER
8   DIM inMd AS INTEGER
```

```

9     DIM inYd AS INTEGER
10    DIM inW AS INTEGER
11    DIM inID AS INTEGER
12    DIM inWcount AS INTEGER
13    DIM inArrayCount AS INTEGER
14    DIM inMonat_2 AS INTEGER
15    DIM inJahr AS INTEGER
16    DIM d AS DATE
17    DIM d_neu AS DATE
18    DIM d_end AS DATE
19    DIM stBezeichnung AS STRING
20    DIM stBezeichnung_2 AS STRING
21    DIM stDatum AS STRING
22    DIM stDat AS STRING
23    DIM stDat_2 AS STRING
24    DIM stMonat AS STRING
25    DIM stForm AS STRING
26    d = YEAR(daStart) & "-" & MONTH(daStart) & "-01"
27    inJahr = Year(d)
28    d_end = daEnde
29    d_neu = DateAdd("d",1,d)
30    stDatum = Date_2_SQLDate(d)
31    oSQL_Anweisung = oVerbindung.createStatement()
32    oSQL_Anweisung1 = oVerbindung.createStatement()
33    oSQL_Anweisung2 = oVerbindung.createStatement()
34    stSql = "DELETE FROM ""tmp_Wochenkalender""
35    oSQL_Anweisung.executeUpdate(stSql)
36    oSQL_Anweisung.executeQuery("ALTER TABLE ""tmp_Wochenkalender"" ALTER COLUMN
    ""ID"" RESTART WITH 1")
37    stMonat = Date_2_Monat(d)
38    inTag = DatePart("w",d,2)
39    inWoche = DatePart("ww",d,2,2)
40    inMonat = Trim(Month(d))
41    inMonatSort = inMonat
42    WHILE d_neu < d_end
43        stSql1 = "INSERT INTO ""tmp_Wochenkalender"" (""Jahr"",""Woche"",
    ""D_""+inTag+""", ""Monat_Text"", ""Monat_Zahl"", ""Monat_Sort"" ) VALUES
    (''+inJahr+'', ''+inWoche+'', ''+stDatum+'', ''+stMonat+'', ''+inMonat+'',
    ''+inMonatSort+'')"
44        oSQL_Anweisung1.executeUpdate(stSql1)
45        stSql = "CALL IDENTITY()"
46        oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
47        WHILE oAbfrageergebnis.next
48            inID = oAbfrageergebnis.getInt(1)
49        WEND
50        FOR i = inTag+1 TO 7
51            stDatum = Date_2_SQLDate(d_neu)
52            stSql2 = "UPDATE ""tmp_Wochenkalender"" SET ""D_""+i+"" =''+stDatum+''
    WHERE ""Woche"" = ''+inWoche+'' AND ""Jahr"" = ''+inJahr+''"
53            oSQL_Anweisung2.executeUpdate(stSql2)
54            d_neu=DateAdd("d",1,d_neu)
55        NEXT
56        stDatum = Date_2_SQLDate(d_neu)
57        stMonat = Date_2_Monat(d_neu)
58        inTag = 1
59        inMonat_2 = Trim(Month(d_neu))
60        inMonatSort = inMonat_2
61        inJahr = Year(d_neu)
62        IF Year(d) < inJahr THEN
63            inMonatSort = inMonat_2+12
64        END IF
65        IF inMonat_2 <> inMonat AND d_neu < d_end THEN
66            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+'', "
67        FOR i = 1 TO 7

```

```

68         stSql = stSql + "(SELECT ""D_""+i+"""" FROM ""tmp_Wochenkalender"" WHERE
           ID = ''"+inID+'''),"
69     NEXT
70     stSql = Left(stSql , Len(stSql) -1 ) &""
71     oSQL_Anweisung.executeUpdate(stSql)
72 END IF
73 inWoche = DatePart("ww",d_neu,2,2)
74 inMonat = Trim(Month(d_neu))
75 inMonatSort = inMonat
76 IF Year(d) < inJahr THEN
77     inMonatSort = inMonat + 12
78 END IF
79 d_neu=DateAdd("d",1,d_neu)
80 WEND
81 stSql = "SELECT ""Bezeichnung"", DAYOFWEEK( ""Datum""), ""Datum"", ""Format""
        FROM ""tmp_Kalender"""
82 oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
83 WHILE oAbfrageergebnis.next
84     stBezeichnung = oAbfrageergebnis.getString(1)
85     inTag = oAbfrageergebnis.getInt(2)
86     inID = inTag - 1
87     IF inID = 0 THEN
88         inID = 7
89     END IF
90     stDat = oAbfrageergebnis.getString(3)
91     stForm = oAbfrageergebnis.getString(4)
92     stSql1 = "UPDATE ""tmp_Wochenkalender"" SET ""T_""+inID+""""
              =''"+stBezeichnung+''', ""F_""+inID+"""" =''"+stForm+''' WHERE ""D_""+inID+"""" =
              ''"+stDat+'''
93     oSQL_Anweisung1.executeUpdate(stSql1)
94 WEND
95 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_half

Aufruf aus
Makro: <i>Bericht_Start</i>

Benötigt
Makro: keins
Tabelle: <i>tmp_Wochenkalender_half</i>
Ansicht: <i>Ansicht_Wochen_einzeln</i>

```

1 SUB Wochenkalender_half(daStart AS DATE, daEnde AS DATE)
2   DIM inWoche AS INTEGER
3   DIM inJahr AS INTEGER
4   DIM i AS INTEGER
5   DIM d AS DATE
6   DIM d_end AS DATE
7   DIM arRow1()
8   DIM arRow2()
9   d = daStart
10  inJahr = Year(d)
11  d_end = daEnde
12  oSQL_Anweisung = oVerbindung.createStatement()
13  stSql = "DELETE FROM ""tmp_wochenkalender_half""
14  oSQL_Anweisung.executeUpdate(stSql)
15  oSQL_Anweisung.executeQuery("ALTER TABLE ""tmp_wochenkalender_half"" ALTER COLUMN
    ""ID"" RESTART WITH 1")
16  inWoche = DatePart("ww", d, 2, 2)
17  arRow1 = array("Monat_Text", "Monat_Zahl", "D_1", "D_2", "D_3", "T_1", "T_2", "T_3",
    "F_1", "F_2", "F_3")
18  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")
19  WHILE d < d_end
20    stSql = "INSERT INTO ""tmp_wochenkalender_half"" ("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+"', "
21  FOR i = LBound(arRow1()) TO UBound(arRow1())
22    stSql = stSql + "(SELECT """+arRow1(i)"" FROM ""Ansicht_Wochen_einzeln""
    WHERE ""Woche"" = '"+inWoche+"' AND ""Jahr"" = '"+inJahr+"'), "
23  NEXT
24  stSql = Left(stSql, Len(stSql) - 1) & ")"
25  oSQL_Anweisung.executeUpdate(stSql)

```

```

26      stSql = "INSERT INTO ""tmp_Wochenkalender_half"" (""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+'', "
27      FOR i = LBound(arRow2()) TO UBound(arRow2())
28      stSql = stSql + "(SELECT """+arRow2(i)"" FROM ""Ansicht_Wochen_einzeln""
          WHERE ""Woche"" = '"+inWoche+' AND ""Jahr"" = '"+inJahr+''),"
29      NEXT
30      stSql = Left(stSql, Len(stSql) - 1) & ")"
31      oSQL_Anweisung.executeUpdate(stSql)
32      d = DateAdd("d", 7, d)
33      inJahr = Year(d)
34      inWoche = DatePart("ww", d, 2, 2)
35      WEND
36      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_half" 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: <i>Bericht_Start</i>
Benötigt
Makro: <i>Date_2_SQLDate</i>
Tabelle: <i>tmp_Halbjahreskalender, tmp_Kalender</i>

```

1 SUB Halbjahreskalender(inJahr AS INTEGER)
2   DIM i AS INTEGER

```

```

3     DIM inMonat AS INTEGER
4     DIM inWoche AS INTEGER
5     DIM inTag AS INTEGER
6     DIM inDd AS INTEGER
7     DIM inMd AS INTEGER
8     DIM inYd AS INTEGER
9     DIM inW AS INTEGER
10    DIM inID AS INTEGER
11    DIM inWcount AS INTEGER
12    DIM inArrayCount AS INTEGER
13    DIM inMonat_2 AS INTEGER
14    DIM k AS INTEGER
15    DIM d AS DATE
16    DIM d_neu AS DATE
17    DIM d_end AS DATE
18    DIM stBezeichnung AS STRING
19    DIM stBezeichnung_2 AS STRING
20    DIM stDatum AS STRING
21    DIM stDat AS STRING
22    DIM stDat_2 AS STRING
23    DIM stMonat AS STRING
24    DIM stTag AS STRING
25    DIM stJahr AS STRING
26    DIM arMon()
27    d = "01.01."+inJahr
28    d_end = "01.01."+inJahr+1
29    d_neu=DateAdd("d",1,d)
30    stDatum = Date_2_SQLDate(d)
31    stJahr = Trim(Str(inJahr))
32    oSQL_Anweisung = oVerbindung.createStatement()
33    oSQL_Anweisung1 = oVerbindung.createStatement()
34    oSQL_Anweisung2 = oVerbindung.createStatement()
35    stSql = "DELETE FROM ""tmp_Halbjahreskalender""
36    oSQL_Anweisung.executeUpdate(stSql)
37    oSQL_Anweisung.executeQuery("ALTER TABLE ""tmp_Halbjahreskalender"" ALTER COLUMN
    ""ID"" RESTART WITH 1")
38    arMon = array("-01-", "-02-", "-03-", "-04-", "-05-", "-06-", "-07-", "-08-", "-09-",
    "-10-", "-11-", "-12-")
39    inID = 28
40    IF isDate("29.02."+stJahr) THEN
41        inID = 29
42    END IF
43    FOR i=1 TO inID
44        IF i < 10 THEN
45            stTag = "0"+Trim(Str(i))
46        ELSE
47            stTag = Trim(Str(i))
48        END IF
49        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', "
50    FOR k = 0 TO 5
51        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+', "
52    NEXT
53    stSql = Left(stSql , Len(stSql) -1 ) &")"
54    oSQL_Anweisung.executeUpdate(stSql)
55    NEXT
56    inID = inID + 1
57    FOR i = inID TO 30
58        stTag = Trim(Str(i))
59        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', "
60    FOR k = 0 TO 5

```

```

61     IF k = 1 THEN
62     ELSE
63         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+''), "
64     END IF
65     NEXT
66     stSql = Left(stSql , Len(stSql) -1 ) &")"
67     oSQL_Anweisung.executeUpdate(stSql)
68     NEXT
69     stTag = "31"
70     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', "
71     FOR k = 0 TO 4 STEP 2
72         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+''), "
73     NEXT
74     stSql = Left(stSql , Len(stSql) -1 ) &")"
75     oSQL_Anweisung.executeUpdate(stSql)
76     FOR i=1 TO 30
77         IF i < 10 THEN
78             stTag = "0"+Trim(Str(i))
79         ELSE
80             stTag = Trim(Str(i))
81         END IF
82         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', "
83     FOR k = 6 TO 11
84         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+''), "
85     NEXT
86     stSql = Left(stSql , Len(stSql) -1 ) &")"
87     oSQL_Anweisung.executeUpdate(stSql)
88     NEXT
89     stTag = "31"
90     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', "
91     FOR k = 6 TO 11
92         IF k = 8 OR k = 10 THEN
93         ELSE
94             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+''), "
95     END IF
96     NEXT
97     stSql = Left(stSql , Len(stSql) -1 ) &")"
98     oSQL_Anweisung.executeUpdate(stSql)
99     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: <i>Bericht_Start</i>

Benötigt
Makro: <i>SQLDate_2_Datum, Date_2_SQLDate</i>
Tabelle: <i>tmp_Terminuebersicht, Termine</i>

```

1 SUB Zeitplan(daStart AS DATE, daEnde AS DATE, stTermin AS STRING)
2   DIM stStart AS STRING
3   DIM stEnde AS STRING
4   DIM stDat AS STRING
5   DIM stDatum AS STRING
6   DIM stBemerkung AS STRING
7   DIM stBezeichnung AS STRING
8   DIM stBezeichnung_alt AS STRING
9   DIM stBezeichNeu AS STRING
10  DIM stSpaltengrenze AS STRING
11  DIM inID AS INTEGER
12  DIM inID_alt AS INTEGER

```

```

13 DIM i AS INTEGER
14 DIM k AS INTEGER
15 stStart = Date_2_SQLDate(daStart)
16 stEnde = Date_2_SQLDate(daEnde)
17 oSQL_Anweisung = oVerbindung.createStatement()
18 oSQL_Anweisung1 = oVerbindung.createStatement()
19 oSQL_Anweisung2 = oVerbindung.createStatement()
20 stSql = "DELETE FROM ""tmp_Terminuebersicht""
21 oSQL_Anweisung.executeUpdate(stSql)
22 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 ""Terminart"". ""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)"
23 oSql_Anweisung.executeUpdate(stSql)
24 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) "
25 oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
26 WHILE oAbfrageergebnis.next
27     inID = oAbfrageergebnis.getInt(1)
28     stDat = oAbfrageergebnis.getString(2)
29     stBemerkung = oAbfrageergebnis.getString(3)
30     stBezeichnung = oAbfrageergebnis.getString(4)
31     IF stDat <> "" THEN
32         stDatum = "- "+SQLDate_2_Datum(stDat)
33         stSql1 = "UPDATE ""tmp_Terminuebersicht"" SET ""D_2"" = '"+stDatum+"' WHERE
            ""ID"" = '"+inID+""
34         oSQL_Anweisung1.executeUpdate(stSql1)
35     END IF
36     IF stBemerkung <> "" THEN
37         stBezeichnung = stBezeichnung + " | " + stBemerkung
38         stSql1 = "UPDATE ""tmp_Terminuebersicht"" SET ""Bezeichnung""
            = '"+stBezeichnung+"' WHERE ""ID"" = '"+inID+""
39         oSQL_Anweisung1.executeUpdate(stSql1)
40     END IF
41 WEND
42 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"""
43 oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
44 inID_alt = -1
45 stBezeichnung_alt = ""
46 k = 1
47 stSpaltengrenze = "Zu viele Termine in einer Zeile. Nicht alle Termine wurden
    übertragen."
48 WHILE oAbfrageergebnis.next
49     inID = oAbfrageergebnis.getInt(1)
50     stDatum = SQLDate_2_Datum(oAbfrageergebnis.getString(2))
51     stEnde = oAbfrageergebnis.getString(3)
52     stBemerkung = oAbfrageergebnis.getString(4)
53     stBezeichnung = oAbfrageergebnis.getString(5)
54     IF stBezeichnung = stBezeichnung_alt THEN
55         IF i>7 THEN
56             msgbox stSpaltengrenze
57         ELSE
58             stSql1 = "UPDATE ""tmp_Terminuebersicht"" SET ""D_"+i+"" = '"+stDatum+"'
                WHERE ""ID"" = '"+inID_alt+""
59             oSQL_Anweisung1.executeUpdate(stSql1)

```

```

60     i = i + 1
61     IF stEnde <> "" THEN
62         IF i>7 THEN
63             msgbox stSpaltengrenze
64         ELSE
65             stEnde = "- "+SQLDate_2_Datum(stEnde)
66             stSql1 = "UPDATE ""tmp_Terminuebersicht"" SET ""D_"+i+""
                    ='+stEnde+' WHERE ""ID"" = '"+inID_alt+'""
67             oSQL_Anweisung1.executeUpdate(stSql1)
68             i = i + 1
69         END IF
70     END IF
71     IF stBemerkung <> "" THEN
72         stSql2 = "SELECT ""Bezeichnung"" FROM ""tmp_Terminuebersicht"" WHERE
                    ""ID"" = '"+inID_alt+'""
73         oAbfrageergebnis2 = oSQL_Anweisung2.executeQuery(stSql2)
74         WHILE oAbfrageergebnis2.next
75             stBezeichNeu = oAbfrageergebnis2.getString(1)
76         WEND
77         stBezeichNeu = stBezeichNeu + " | " + stBemerkung
78         stSql1 = "UPDATE ""tmp_Terminuebersicht"" SET ""Bezeichnung""
                    ='+stBezeichNeu+' WHERE ""ID"" = '"+inID_alt+'""
79         oSQL_Anweisung1.executeUpdate(stSql1)
80     END IF
81     stSql1 = "DELETE FROM ""tmp_Terminuebersicht"" WHERE ""ID"" =
                '"+inID+'""
82     oSQL_Anweisung1.executeUpdate(stSql1)
83     END IF
84 ELSE
85     i = 2
86     stSql2 = "SELECT ""D_2"" FROM ""tmp_Terminuebersicht"" WHERE ""ID"" =
                '"+inID+' AND NOT ""D_2"" IS NULL"
87     oAbfrageergebnis2 = oSQL_Anweisung2.executeQuery(stSql2)
88     WHILE oAbfrageergebnis2.next
89         i = 3
90     WEND
91     stBezeichnung_alt = stBezeichnung
92     inID_alt = inID
93     END IF
94     WEND
95 END SUB

```

Mit dieser Prozedur wird der Zeitplan über die Tabelle "tmp_Terminuebersicht" mit Daten versorgt. Nachdem die Tabelle "tmp_Terminuebersicht" 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_Terminuebersicht" 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 "zielID" 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 unterscheidet 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*

```
1 SUB GruppeMultiUebersicht(inJahr AS INTEGER, inFerien AS INTEGER, stTermin AS STRING,
2   inJahrgang AS INTEGER, stJahrgang AS STRING)
3   DIM stDatum AS STRING
4   DIM stDat_Beginn AS STRING
5   DIM stDat_Ende AS STRING
6   DIM stUpdate AS STRING
7   DIM stZeit AS STRING
8   DIM i AS INTEGER
9   DIM d AS DATE
10  DIM d_neu AS DATE
11  DIM d_end AS DATE
12  DIM stBezeichnung AS STRING
13  DIM stBemerkung AS STRING
14  inJahresschleife = inJahr
15  oSQL_Anweisung = oVerbindung.createStatement()
16  oSQL_Anweisung1 = oVerbindung.createStatement()
17  oSQL_Anweisung2 = oVerbindung.createStatement()
18  stSql = "DELETE FROM ""tmp_Gruppe_multi""
19  oSQL_Anweisung.executeUpdate(stSql)
20  oSQL_Anweisung.executeQuery("ALTER TABLE ""tmp_Gruppe_multi"" ALTER COLUMN ""ID""
21    RESTART WITH 1")
22  FOR inJahr=inJahresschleife TO inJahresschleife+1
23    Feiertage = FeiertageArray(inJahr)
24    d = "01.01."+inJahr
```

```

23     d_end = "01.01."+inJahr+1
24     stDatum = Date_2_SQLDate(d)
25     d_neu=d
26     WHILE d_neu < d_end
27         stSql = "INSERT INTO ""tmp_Gruppe_multi"" (""Datum"") VALUES
                ('"+stDatum+"') "
28         oSQL_Anweisung.executeUpdate(stSql)
29         d_neu=DateAdd("d",1,d_neu)
30         stDatum = Date_2_SQLDate(d_neu)
31     WEND
32     FOR i=0 TO 14
33         stDatum = Feiertage(i,1)
34         stBezeichnung = Feiertage(i,0)
35         stSql = "UPDATE ""tmp_Gruppe_multi"" SET ""Bezeichnung""
                ='" +stBezeichnung+"', ""Format"" = '1' WHERE ""Datum"" = '" +stDatum+"'"
36         oSQL_Anweisung.executeUpdate(stSql)
37     NEXT
38     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"
39     oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
40     WHILE oAbfrageergebnis.next
41         stDat_Beginn = oAbfrageergebnis.getString(1)
42         stDat_End = oAbfrageergebnis.getString(2)
43         IF inFerien = 1 THEN
44             stUpdate = """"Bezeichnung"" ='Ferien', ""Format"" = '1'"
45         ELSE
46             stUpdate = """"Format"" = '1'"
47         END IF
48         stSql1 = "UPDATE ""tmp_Gruppe_multi"" SET "+stUpdate+" WHERE ""Datum"" =
                '"+stDat_Beginn+"'"
49         oSQL_Anweisung1.executeUpdate(stSql1)
50         IF stDat_End <> "" AND stDat_End <> stDat_Beginn THEN
51             d_neu=DateAdd("d",1,CDate(stDat_Beginn))
52             WHILE d_neu <= CDate(stDat_End)
53                 stDatum = Date_2_SQLDate(d_neu)
54                 stSql1 = "UPDATE ""tmp_Gruppe_multi"" SET "+stUpdate+" WHERE
                        ""Datum"" = '" +stDatum+"'"
55                 oSQL_Anweisung1.executeUpdate(stSql1)
56                 d_neu=DateAdd("d",1,d_neu)
57             WEND
58         END IF
59     WEND
60     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"
61     oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
62     WHILE oAbfrageergebnis.next
63         stBezeichnung = oAbfrageergebnis.getString(1)
64         stBezeichnung_Dat_End = oAbfrageergebnis.getString(1)
65         stDat_Beginn = oAbfrageergebnis.getString(2)
66         stDat_End = oAbfrageergebnis.getString(3)
67         stBemerkung = oAbfrageergebnis.getString(4)
68         stZeit = oAbfrageergebnis.getString(5)
69         stZeit = left(stZeit,5)
70         IF stBemerkung <> "" THEN
71             stBezeichnung = stBezeichnung + " " + stBemerkung
72             stBezeichnung_Dat_End = stBezeichnung_Dat_End + " " + stBemerkung
73         END IF
74         IF stZeit <> "" THEN
75             stBezeichnung = stZeit + " " + stBezeichnung
76             stBezeichnung_Dat_End = stZeit + " " + stBezeichnung_Dat_End
77         END IF
78         stSql1 = "SELECT ""Bezeichnung"" FROM ""tmp_Gruppe_multi"" WHERE ""Datum""
                = '" +stDat_Beginn+"'"

```

```

79      oAbfrageergebnis1 = oSQL_Anweisung1.executeQuery(stSql1)
80      WHILE oAbfrageergebnis1.next
81          IF oAbfrageergebnis1.getString(1) <> "" THEN
82              stBezeichnung = oAbfrageergebnis1.getString(1) + CHR(13) +
                  stBezeichnung
83          END IF
84      WEND
85      stSql1 = "UPDATE ""tmp_Gruppe_multi"" SET ""Bezeichnung""
              = '"+stBezeichnung+"' WHERE ""Datum"" = '"+stDat_Beginn+''''
86      oSQL_Anweisung1.executeUpdate(stSql1)
87      IF stDat_Ende <> "" AND stDat_Ende <> stDat_Beginn THEN
88          d_neu=DateAdd("d",1,CDate(stDat_Beginn))
89          WHILE d_neu <= CDate(stDat_Ende)
90              stDatum = Date_2_SQLDate(d_neu)
91              stSql2 = "SELECT ""Bezeichnung"" FROM ""tmp_Gruppe_multi"" WHERE
                  ""Datum"" = '"+stDatum+''''
92              oAbfrageergebnis2 = oSQL_Anweisung2.executeQuery(stSql2)
93              WHILE oAbfrageergebnis2.next
94                  IF oAbfrageergebnis2.getString(1) <> "" THEN
95                      stBezeichnung = oAbfrageergebnis2.getString(1) + CHR(13) +
                          stBezeichnung_Dat_Ende
96              ELSE
97                  stBezeichnung = stBezeichnung_Dat_Ende
98              END IF
99          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: <i>Monatsansicht</i>

Benötigt
Makro: <i>Date_2_Datum, Date_2_SQLDate</i>
Tabelle: <i>tmp_Monat, Termine, Terminart</i>

```

1 SUB Monat(inMonat AS INTEGER, inJahr AS INTEGER)
2   DIM d AS DATE
3   DIM d_neu AS DATE
4   DIM d_end AS DATE
5   DIM stDatum AS STRING
6   DIM stDatumSql AS STRING
7   DIM inID AS INTEGER
8   DIM i AS INTEGER
9   oSQL_Anweisung = oVerbindung.createStatement()
10  oSQL_Anweisung1 = oVerbindung.createStatement()
11  stSql = "DELETE FROM ""tmp_Monat""
12  oSQL_Anweisung.executeUpdate(stSql)
13  d = CDate("01."+inMonat+"."+inJahr)
14  d_end = DateAdd("m",1,d)
15  inID = 1
16  FOR d_neu = d TO d_end-1
17    stDatumSql = Date_2_SQLDate(d_neu)
18    stDatum = Date_2_Datum(d_neu)
19    inWochentag = DatePart("w",d_neu,2,2)
20    stSql = "INSERT INTO ""tmp_Monat"" (""ID"", ""T_"+inWochentag+""") VALUES
21      (""+inID+""',""+stDatum+""')
22    oSQL_Anweisung.executeUpdate(stSql)
23    stBezeichnung = stDatum
24    stSql = "SELECT ""Terminart"". ""Terminart""||': '|""Termine"". ""Bezeichnung""
25      FROM ""Termine"" LEFT JOIN ""Terminart"" ON ""Terminart"". ""ID"" =
26      ""Termine"". ""artID"" WHERE ""Termine"". ""Beginn_Datum"" = '"+stDatumSql+"'
27      OR (""Termine"". ""Beginn_Datum"" < '"+stDatumSql+"' AND
28      ""Termine"". ""Ende_Datum"" >= '"+stDatumSql+"' )"
29    oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
30    WHILE oAbfrageergebnis.next
31      stBezeichnung = stBezeichnung + CHR(13) + oAbfrageergebnis.getString(1)
32      stSql1 = "UPDATE ""tmp_Monat"" SET ""T_"+inWochentag+""
33      = '"+stBezeichnung+"' WHERE ""ID""='"+inID+""'
34      oSQL_Anweisung1.executeUpdate(stSql1)
35    WEND
36    FOR i = inWochentag+1 TO 7
37      d_neu = DateAdd("d",1,d_neu)
38      IF d_neu < d_end THEN
39        stDatumSql = Date_2_SQLDate(d_neu)
40        stDatum = Date_2_Datum(d_neu)
41        stBezeichnung = stDatum
42        stSql1 = "UPDATE ""tmp_Monat"" SET ""T_"+i+"" = '"+stBezeichnung+"'
43        WHERE ""ID""='"+inID+""'
44        oSQL_Anweisung1.executeUpdate(stSql1)
45      stSql = "SELECT ""Terminart"". ""Terminart""||': '|
46        '||""Termine"". ""Bezeichnung"" FROM ""Termine"" LEFT JOIN
47        ""Terminart"" ON ""Terminart"". ""ID"" = ""Termine"". ""artID"" WHERE
48        ""Termine"". ""Beginn_Datum"" = '"+stDatumSql+"' OR
49        (""Termine"". ""Beginn_Datum"" < '"+stDatumSql+"' AND
50        ""Termine"". ""Ende_Datum"" >= '"+stDatumSql+"' )"

```

```

39         oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
40     WHILE oAbfrageergebnis.next
41         stBezeichnung = stBezeichnung + CHR(13) +
            oAbfrageergebnis.getString(1)
42     stSql1 = "UPDATE ""tmp_Monat"" SET ""T_"+i+"" = '"+stBezeichnung+'
            WHERE ""ID""=''+inID+''"
            oSQL_Anweisung1.executeUpdate(stSql1)
43     WEND
44     END IF
45     NEXT
46     inID = inID + 1
47     NEXT
48 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 Eintrag 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: <i>Monatsansicht_aktualisieren, Monat_aktualisieren</i>

Benötigt
Makro: <i>Monat</i>
Tabelle: <i>Filter</i>

```

1 SUB Monatsansicht
2     DIM oForm AS OBJECT
3     DIM oSubForm AS OBJECT
4     DIM stBezeichnung AS STRING
5     DIM a()
6     oDoc = thisComponent
7     oDrawpage = oDoc.Drawpage
8     oForm = oDrawpage.Forms.getByname("Monat")
9     oSQL_Anweisung = oVerbindung.createStatement()
10    stSql = "SELECT ""Jahr_Monat"" FROM ""Filter"" WHERE ""ID"" = TRUE "
11    oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
12    WHILE oAbfrageergebnis.next
13        stBezeichnung = oAbfrageergebnis.getString(1)
14    WEND
15    a = Split(stBezeichnung, ",")

```

```

16  Monat(CInt(a(1)),CInt(a(0)))
17  oSubForm = oForm.getByName("Ansicht")
18  oSubForm.reload()
19  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:	<i>Kalender, Monatskalender, Halbjahreskalender, Zeitplan, Monat, Bericht_Start, Gruppe-MultiUebersicht, FeiertageArray</i>

Benötigt	
Makro:	keins

```

1  FUNCTION Date_2_SQLDate(d AS DATE) AS STRING
2  DIM stMonth AS STRING
3  DIM stDay AS STRING
4  stMonth = Right("0" + Trim(Str(Month(d))),2)
5  stDay = Right("0" + Trim(Str(Day(d))),2)
6  Date_2_SQLDate = Trim(Str(Year(d))) + "-" + stMonth + "-" + stDay
7  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:	<i>Monat</i>

Benötigt	
Makro:	keins

```

1  FUNCTION Date_2_Datum(d AS DATE) AS STRING
2  DIM stMonth AS STRING
3  DIM stDay AS STRING
4  stMonth = Right("0" + Trim(Str(Month(d))),2)
5  stDay = Right("0" + Trim(Str(Day(d))),2)
6  Date_2_Datum = stDay + "." + stMonth + "." + Trim(Str(Year(d)))
7  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

```
1 FUNCTION SQLDate_2_Datum(d AS STRING) AS STRING
2     DIM a()
3     DIM stDatum AS STRING
4     a = split(d, "-")
5     stDatum = a(2)+"."+a(1)+"."+Right(a(0),2)
6     SQLDate_2_Datum = stDatum
7 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

```
1 FUNCTION Date_2_Monat(d AS DATE) AS STRING
2     IF Month(d) = 1 THEN
3         Date_2_Monat = "Januar"
4     ELSEIF Month(d) = 2 THEN
5         Date_2_Monat = "Februar"
6     ELSEIF Month(d) = 3 THEN
7         Date_2_Monat = "März"
8     ELSEIF Month(d) = 4 THEN
9         Date_2_Monat = "April"
10    ELSEIF Month(d) = 5 THEN
11        Date_2_Monat = "Mai"
12    ELSEIF Month(d) = 6 THEN
13        Date_2_Monat = "Juni"
14    ELSEIF Month(d) = 7 THEN
15        Date_2_Monat = "Juli"
16    ELSEIF Month(d) = 8 THEN
17        Date_2_Monat = "August"
18    ELSEIF Month(d) = 9 THEN
19        Date_2_Monat = "September"
20    ELSEIF Month(d) = 10 THEN
21        Date_2_Monat = "Oktober"
22    ELSEIF Month(d) = 11 THEN
23        Date_2_Monat = "November"
24    ELSE
25        Date_2_Monat = "Dezember"
26    END IF
27 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

```
1 FUNCTION Datum_Ostersonntag(inJahr AS INTEGER) AS LONG
2 REM Spencers Osterformel mit Ganzzahldivision \
3 DIM a AS INTEGER
4 DIM b AS INTEGER
5 DIM c AS INTEGER
6 DIM d AS INTEGER
7 DIM e AS INTEGER
8 DIM f AS INTEGER
9 DIM g AS INTEGER
10 DIM h AS INTEGER
11 DIM i AS INTEGER
12 DIM k AS INTEGER
13 DIM l AS INTEGER
14 DIM m AS INTEGER
15 DIM n AS INTEGER
16 DIM p AS INTEGER
17 DIM Os AS INTEGER
18 DIM loDatum AS LONG
19 a = inJahr MOD 19
20 b = inJahr \ 100
21 c = inJahr MOD 100
22 d = b \ 4
23 e = b MOD 4
24 f = (b + 8) \ 25
25 g = (b - f + 1) \ 3
26 h = (19*a + b - d - g + 15) MOD 30
27 i = c \ 4
28 k = c MOD 4
29 l = ( 32 + 2*e + 2*i - h - k ) MOD 7
30 m = ( a + 11*h + 22*l ) \ 451
31 n = ( h + l - 7*m + 114 ) \ 31 'Monat
32 p = ( h + l - 7*m + 114 ) MOD 31
33 Os = p + 1 'Tag
34 loDatum = DateSerial(inJahr, n, Os)
35 Datum_Ostersonntag() = loDatum
36 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*

```
1 FUNCTION FeiertageArray(inJahr AS INTEGER) AS OBJECT
```

```

2   DIM d_Ostersonntag AS DATE
3   DIM Feiertage(14,1)
4   d_Ostersonntag = CDate(Datum_Ostersonntag(inJahr))
5   Feiertage(0,0) = "Neujahr"
6   Feiertage(0,1) = Trim(Str(inJahr)) + "-01-01"
7   Feiertage(1,0) = "Rosenmontag"
8   Feiertage(1,1) = Date_2_SQLDate(DateAdd("d",-48,d_Ostersonntag))
9   Feiertage(2,0) = "Karfreitag"
10  Feiertage(2,1) = Date_2_SQLDate(DateAdd("d",-2,d_Ostersonntag))
11  Feiertage(3,0) = "Ostersonntag"
12  Feiertage(3,1) = Date_2_SQLDate(d_Ostersonntag)
13  Feiertage(4,0) = "Ostermontag"
14  Feiertage(4,1) = Date_2_SQLDate(DateAdd("d",1,d_Ostersonntag))
15  Feiertage(5,0) = "Chr.Himmelfahrt"
16  Feiertage(5,1) = Date_2_SQLDate(DateAdd("d",39,d_Ostersonntag))
17  Feiertage(6,0) = "Pfingstsonntag"
18  Feiertage(6,1) = Date_2_SQLDate(DateAdd("d",49,d_Ostersonntag))
19  Feiertage(7,0) = "Pfingstmontag"
20  Feiertage(7,1) = Date_2_SQLDate(DateAdd("d",50,d_Ostersonntag))
21  Feiertage(8,0) = "Fronleichnam"
22  Feiertage(8,1) = Date_2_SQLDate(DateAdd("d",60,d_Ostersonntag))
23  Feiertage(9,0) = "Tag der dt. Einheit"
24  Feiertage(9,1) = Trim(Str(inJahr)) + "-10-03"
25  Feiertage(10,0) = "Allerheiligen"
26  Feiertage(10,1) = Trim(Str(inJahr)) + "-11-01"
27  Feiertage(11,0) = "Heiligabend"
28  Feiertage(11,1) = Trim(Str(inJahr)) + "-12-24"
29  Feiertage(12,0) = "1. Weihnachtstag"
30  Feiertage(12,1) = Trim(Str(inJahr)) + "-12-25"
31  Feiertage(13,0) = "2. Weihnachtstag"
32  Feiertage(13,1) = Trim(Str(inJahr)) + "-12-26"
33  Feiertage(14,0) = "Tag der Arbeit"
34  Feiertage(14,1) = Trim(Str(inJahr)) + "-05-01"
35  FeiertageArray = Feiertage
36  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

```

1  SUB BerichtZeilenhoeheAuto(oReport AS OBJECT)
2  DIM oTables AS OBJECT
3  DIM oTable AS OBJECT
4  DIM inT AS INTEGER
5  DIM inI AS INTEGER
6  DIM oRows AS OBJECT
7  DIM oRow AS OBJECT
8  oTables = oReport.getTextTables()
9  FOR inT = 0 TO oTables.count() - 1
10     oTable = oTables.getByIndex(inT)
11     IF Left$(oTable.name, 6) = "Detail" THEN
12         oRows = oTable.Rows
13         FOR inI = 0 TO oRows.count - 1
14             oRow = oRows.getByIndex(inI)
15             oRow.IsAutoHeight = True

```

```

16         NEXT inI
17     ENDIF
18     NEXT inT
19 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: <i>Terminart_Termin, Termine_Kalenderübersicht, Termine_Monat_Terminart, Termine_Tabelleneingabe</i>

Benötigt
Makro: <i>Kalender, Monatskalender, Wochenkalender_half, Halbjahreskalender, Zeitplan, GruppeMultiUebersicht, BerichtZeilenhoeheAuto, DialogStart, Date_2_SQLDate</i>
Tabelle: <i>Zielgruppe, Filter</i>

```

1 SUB Bericht_Start(oEvent AS OBJECT)
2   DIM oForm AS OBJECT
3   DIM oFeld_1 AS OBJECT
4   DIM oFeld_2 AS OBJECT
5   DIM oFeld_3 AS OBJECT
6   DIM oFeld_4 AS OBJECT
7   DIM oReport AS OBJECT
8   DIM daStart AS DATE
9   DIM daEnde AS DATE
10  DIM stKalender AS STRING
11  DIM inZielgruppe AS INTEGER
12  DIM stTermin AS STRING
13  DIM stGruppeMulti AS STRING
14  DIM stHalbjahr AS STRING
15  DIM inGruppeMulti AS INTEGER
16  oForm = oEvent.Source.Model.Parent
17  oForm.updateRow
18  Wait 1000
19  oFeld_1 = oForm.getByName("datStart")
20  oFeld_2 = oForm.getByName("datEnde")
21  oFeld_3 = oForm.getByName("lboKalenderart")
22  oFeld_4 = oForm.getByName("lboZielgruppe")
23  oFeld_5 = oForm.getByName("lboGruppeMulti")
24  stKalender = Listenfeld_Text(oFeld_3)
25  inZielgruppe = ID_Ermittlung(oFeld_4)
26  oSQL_Anweisung = oVerbindung.createStatement()
27  stSql = "SELECT ""Termingruppe"" FROM ""Zielgruppe"" WHERE ""ID""
          =""+inZielgruppe+"" "
28  oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
29  WHILE oAbfrageergebnis.next
30      stTermin = oAbfrageergebnis.getString(1)
31  WEND
32  IF stKalender <>"" THEN
33      IF NOT isEmpty(oFeld_1.currentValue) THEN
34          daStart = Datumswert(oFeld_1)
35          IF NOT isEmpty(oFeld_2.currentValue) THEN
36              daEnde = Datumswert(oFeld_2)
37          ELSE
38              daEnde = "1900-01-01"
39          END IF

```

```

40     IF daEnde < daStart THEN
41         daEnde = DateAdd("yyyy",1,daStart)
42     END IF
43 ELSE
44     daStart = now
45     daEnde = DateAdd("yyyy",1,daStart)
46 END IF
47 IF stKalender = "Terminübersicht" THEN
48     Zeitplan(daStart,daEnde,stTermin)
49     oReport = ThisDatabaseDocument.ReportDocuments.getByName("Zeitplan").open
50     BerichtZeilenhoeheAuto(oReport)
51 ELSEIF stKalender = "Halbjahreskalender" THEN
52     Kalender(YEAR(daStart),0,stTermin)
53     Halbjahreskalender(YEAR(daStart))
54     ThisDatabaseDocument.ReportDocuments.getByName(
55         "Halbjahreskalender_A4_quer").open
56 ELSEIF stKalender = "Monate Montag - Sonntag" THEN
57     Kalender(YEAR(daStart),1,stTermin)
58     Monatskalender(daStart,daEnde)
59     ThisDatabaseDocument.ReportDocuments.getByName("Monatskalender_Mo_So").open
60 ELSEIF stKalender = "Monate Montag - Freitag" THEN
61     Kalender(YEAR(daStart),1,stTermin)
62     Monatskalender(daStart,daEnde)
63     ThisDatabaseDocument.ReportDocuments.getByName("Monatskalender_Mo_Fr").open
64 ELSEIF stKalender = "Halbwochenkalender als Notizbuch" THEN
65     Kalender(YEAR(daStart),1,stTermin)
66     Monatskalender(daStart,daEnde)
67     Wochenkalender_half(daStart,daEnde)
68     ThisDatabaseDocument.ReportDocuments.getByName("Wochenkalender_half").open
69 ELSEIF stKalender = "Wochentage Mensaabo" THEN
70     Kalender(YEAR(daStart),1,stTermin)
71     ThisDatabaseDocument.ReportDocuments.getByName("Wochentage_Mensaabo").open
72 ELSEIF stKalender = "Klassenarbeitenübersicht" THEN
73     stSql = "UPDATE ""Filter"" SET ""Startjahr"" = '"+Year(daStart)+"' WHERE
74         ""ID"" = TRUE "
75     oSQL_Anweisung.executeUpdate(stSql)
76     IF NOT isEmpty(oFeld_5.currentValue) THEN
77         stGruppeMulti = Listenfeld_Text(oFeld_5)
78         inGruppeMulti = ID_Ermittlung(oFeld_5)
79         GruppeMultiUebersicht(YEAR(daStart),1,stTermin,inGruppeMulti,
80             stGruppeMulti)
81     ThisDatabaseDocument.ReportDocuments.getByName(
82         "Liste_Gruppe_multi").open
83 ELSE
84     MsgBox "Ein Jahrgang muss ausgewählt werden."
85     END IF
86 ELSEIF stKalender = "Klassenarbeitenübersicht 1. HJ" THEN
87     stSql = "UPDATE ""Filter"" SET ""Startjahr"" = '"+Year(daStart)+"' WHERE
88         ""ID"" = TRUE "
89     oSQL_Anweisung.executeUpdate(stSql)
90     stSql = "SELECT ""Halbjahrende"" FROM ""Filter"" WHERE ""ID"" = TRUE AND
91         ""Halbjahrende"" BETWEEN '"+Date_2_SQLDate(daStart)+"' AND
92         '"+Date_2_SQLDate(daEnde)+"'"
93     oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
94     WHILE oAbfrageergebnis.next
95         stHalbjahr = oAbfrageergebnis.getString(1)
96     WEND
97     IF stHalbjahr = "" THEN
98         DialogStart
99     END IF
100    IF NOT isEmpty(oFeld_5.currentValue) THEN
101        stGruppeMulti = Listenfeld_Text(oFeld_5)
102        inGruppeMulti = ID_Ermittlung(oFeld_5)
103        GruppeMultiUebersicht(YEAR(daStart),1,stTermin,inGruppeMulti,
104            stGruppeMulti)
105        ThisDatabaseDocument.ReportDocuments.getByName(
106            "Liste_Gruppe_multi_1HJ").open
107    ELSE

```

```

99         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 Ein-

trag 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

```

1 SUB Datenbankbackup(inMax AS INTEGER)
2   DIM oPath AS OBJECT
3   DIM oDoc AS OBJECT
4   DIM sTitel AS STRING
5   DIM sUrl_Ziel AS STRING
6   DIM sUrl_Start AS STRING
7   DIM i AS INTEGER
8   DIM k AS INTEGER
9   oDoc = ThisComponent
10  sTitel = oDoc.Title
11  sUrl_Start = oDoc.URL
12  DO WHILE sUrl_Start = ""
13    oDoc = oDoc.Parent
14    sTitel = oDoc.Title
15    sUrl_Start = oDoc.URL
16  LOOP
17  oPath = createUnoService("com.sun.star.util.PathSettings")
18  FOR i = 1 TO inMax + 1
19    IF NOT FileExists(oPath.Backup & "/" & i & "_" & sTitel) THEN
20      IF i > inMax THEN
21        FOR k = inMax - 1 TO 1 STEP -1
22          IF FileDateTime(oPath.Backup & "/" & k & "_" & sTitel) <=
23             FileDateTime(oPath.Backup & "/" & k+1 & "_" & sTitel) THEN
24            IF k = 1 THEN
25              i = k
26              EXIT FOR
27            END IF
28          ELSE
29            i = k+1
30            EXIT FOR
31          END IF
32        NEXT
33      END IF
34    END IF
35  NEXT
36  sUrl_Ziel = oPath.Backup & "/" & i & "_" & sTitel
37  FileCopy(sUrl_Start,sUrl_Ziel)
38 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*

```
1 SUB Backup_sofort
2   oVerbindung.flush
3   Datenbankbackup(10)
4 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

```
1 SUB Datenspeichern
2   DIM oDocument AS OBJECT
3   DIM oDispatcher AS OBJECT
4   oDocument = ThisComponent.CurrentController.Frame
5   oDispatcher = createUnoService("com.sun.star.frame.DispatchHelper")
6   oDispatcher.executeDispatch(oDocument, ".uno:RecSave", "", 0, Array())
7 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*

```
1 SUB DatenAktualisieren_Termine
2   DIM oForm AS OBJECT
3   DIM oSubForm_1 AS OBJECT
4   DIM oSubForm_2 AS OBJECT
5   Datenspeichern
6   oDoc=thisComponent
7   oDrawpage=oDoc.drawpage
8   oForm = oDrawpage.Forms.getByName("MainForm")
9   oSubForm_1 = oForm.getByName("SubForm")
10  oSubForm_2 = oForm.getByName("SubTerminart")
11  oSubForm_1.reload()
12  oSubForm_2.reload()
13 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*

```

1 SUB DatenAktualisieren_Terminart_Termine
2 DIM oForm AS OBJECT
3 DIM oSubForm AS OBJECT
4 DIM oSubSubForm_1 AS OBJECT
5 DIM oSubSubForm_2 AS OBJECT
6   DatenSpeichern
7   oDoc=thisComponent
8   oDrawpage=oDoc.drawpage
9   oForm = oDrawpage.Forms.getByName("MainForm")
10  oSubForm = oForm.getByName("SubTermine")
11  oSubSubForm_1 = oSubForm.getByName("SubTerminMonat")
12  oSubSubForm_2 = oSubForm.getByName("SubTerminOrt")
13  oSubSubForm_1.reload()
14  oSubSubForm_2.reload()
15 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: <i>Termine_Monat_Terminart</i>
Makro: <i>LetzterDatensatz_Termine_Kalender</i>

Benötigt
Makro: keins

```

1 SUB LetzterDatensatz(oEvent AS OBJECT)
2   DIM oForm AS OBJECT
3   oForm = oEvent.Source
4   oForm.last()
5 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: <i>Termine_Kalenderübersicht</i>

Benötigt
Makro: <i>LetzterDatensatz, Monatsansicht_aktualisieren</i>

```

1 SUB LetzterDatensatz_Termine_Kalender(oEvent AS OBJECT)
2   LetzterDatensatz(oEvent)

```

```

3 Monatsansicht_aktualisieren
4 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: <i>Termine_Kalenderübersicht</i>

Benötigt
Makro: <i>Datumswert</i>

```

1 SUB Anfangswert_Datum(oEvent AS OBJECT)
2   DIM oFeld AS OBJECT
3   oFeld = oEvent.Source.Model
4   IF NOT ISEMPTY(oFeld.GetCurrentValue()) THEN
5     daStartDat = Datumswert(oFeld)
6   END IF
7 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: <i>Termine_Kalenderübersicht</i>
Makro: <i>LetzterDatensatz_Termine_Kalender</i>

Benötigt
Makro: <i>Datumswert, Monatsansicht</i>
Tabelle: <i>Filter</i>

```

1 SUB Monatsansicht_aktualisieren
2   DIM oForm AS OBJECT
3   DIM oForm_2 AS OBJECT
4   DIM oFeld_0 AS OBJECT
5   DIM oFeld_1 AS OBJECT
6   DIM oFeld_2 AS OBJECT
7   DIM daStart AS DATE
8   DIM stMonat AS STRING
9   DIM stBezeichnung AS STRING

```

```

10  oDoc = thisComponent
11  oDrawpage = oDoc.Drawpage
12  oForm = oDrawpage.Forms.getByname("MainForm")
13  oFeld_1 = oForm.getByname("datBeginn_Datum")
14  IF NOT IEMPTY(oFeld_1.getCurrentValue()) THEN
15      IF oForm.getString(oForm.findColumn("ID")) = "" THEN
16          oForm.insertRow()
17      ELSE
18          oForm.updateRow()
19      END IF
20      oSQL_Anweisung = oVerbindung.createStatement()
21      stSql = "SELECT ""Jahr_Monat"" FROM ""Filter"" WHERE ""ID"" = TRUE "
22      oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
23      WHILE oAbfrageergebnis.next
24          stBezeichnung = oAbfrageergebnis.getString(1)
25      WEND
26      daStart = Datumswert(oFeld_1)
27      IF daStartDat <> daStart OR stBezeichnung = "" THEN
28          IF isDate(daStart) THEN
29              stMonat = Trim(str(Year(daStart)))+", "+
                    Right("0"+Trim(str(Month(daStart))),2)
30              stSql = "UPDATE ""Filter"" SET ""Jahr_Monat"" = '"+stMonat+"' WHERE
                    ""ID"" = TRUE "
31              oSQL_Anweisung.executeUpdate(stSql)
32              oForm_2 = oDrawpage.Forms.getByname("Monat")
33              oFeld_2=oForm_2.getByname("lboMonat_Jahr")
34              oFeld_2.refresh()
35              oForm_2.reload()
36              Monatsansicht
37              daStartDat = daStart
38          END IF
39      END IF
40  END IF
41  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 Datumfeld 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** überschrieben, damit beim nächsten Eintrag wieder davon ausgegangen werden kann, dass die Datumsübersicht auf **daStartDat** eingestellt ist.

Monat_aktualisieren

Aufruf aus
Formular: <i>Termine_Kalenderübersicht</i>

Benötigt
Makro: <i>Monatsansicht</i>

```

1 SUB Monat_aktualisieren(oEvent AS OBJECT)
2   DIM oForm AS OBJECT
3   DIM oFeld AS OBJECT
4   oFeld = oEvent.Source.Model
5   oForm = oFeld.Parent
6   oFeld.commit()
7   oForm.updateRow()
8   Monatsansicht
9 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.

Terminaten_uebertragen

Aufruf aus
Formular: <i>Termine_Kalenderübersicht</i>

Benötigt
Makro: keins
Tabelle: <i>Termine</i>

```

1 SUB Terminaten_uebertragen
2   DIM oForm AS OBJECT
3   DIM oFeld_0 AS OBJECT
4   DIM oFeld_1 AS OBJECT
5   DIM oFeld_2 AS OBJECT
6   DIM oFeld_3 AS OBJECT

```

```

7   DIM oFeld_4 AS OBJECT
8   DIM oFeld_5 AS OBJECT
9   DIM oFeld_6 AS OBJECT
10  DIM stBezeichnung AS STRING
11  DIM stBeginn AS STRING
12  DIM stEnde AS STRING
13  DIM stOrt AS STRING
14  DIM stBemerkung AS STRING
15  DIM startID AS STRING
16  oDoc = thisComponent
17  oDrawpage = oDoc.Drawpage
18  oForm = oDrawpage.Forms.getByName("MainForm")
19  IF oForm.getString(oForm.findColumn("ID")) = "" THEN
20      oFeld_1=oForm.getByName("comBezeichnung")
21      stBezeichnung = oFeld_1.getCurrentValue()
22      oSQL_Anweisung = oVerbindung.createStatement()
23      stSql = "SELECT ""Beginn_Uhrzeit"", ""Ende_Uhrzeit"", ""Ort"", ""Bemerkung"",
              ""artID"" FROM ""Termine"" WHERE ""ID"" = ( SELCET MAX(""ID"") FROM
              ""Termine"" WHERE ""Bezeichnung"" = '"+stBezeichnung+' )"
24      oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
25      WHILE oAbfrageergebnis.next
26          stBeginn = oAbfrageergebnis.getString(1)
27          stEnde = oAbfrageergebnis.getString(2)
28          stOrt = oAbfrageergebnis.getString(3)
29          stBemerkung = oAbfrageergebnis.getString(4)
30          startID = oAbfrageergebnis.getString(5)
31      WEND
32      oFeld_2=oForm.getByName("timBeginn_Uhrzeit")
33      oFeld_3=oForm.getByName("timEnde_Uhrzeit")
34      oFeld_4=oForm.getByName("comOrt")
35      oFeld_5=oForm.getByName("txtBemerkung")
36      oFeld_6=oForm.getByName("lboartID")
37      IF stBeginn <> "" THEN
38          oFeld_2.BoundField.updateString(stBeginn)
39      END IF
40      IF stEnde <> "" THEN
41          oFeld_3.BoundField.updateString(stEnde)
42      END IF
43      IF stOrt <> "" THEN
44          oFeld_4.BoundField.updateString(stOrt)
45      END IF
46      IF stBemerkung <> "" THEN
47          oFeld_5.BoundField.updateString(stBemerkung)
48      END IF
49      IF startID <> "" THEN
50          oFeld_6.BoundField.updateString(startID)
51      END IF
52  END IF
53 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

```
1 SUB GruppeMultiSpeichern(oEvent AS OBJECT)
2   DIM aFields()
3   DIM oForm AS OBJECT
4   DIM inValue AS INTEGER
5   DIM i AS INTEGER
6   oForm = oEvent.Source.Model.Parent
7   aFields = Array("Check1", "Check2", "Check3", "Check4", "Check5", "Check6", "Check7",
8     "Check8", "Check9")
9   inValue = 0
10  FOR i = LBound(aFields()) TO UBound(aFields())
11    IF oForm.getByName(aFields(i)).State = 1 THEN
12      inValue = inValue + Cint(oForm.getByName(aFields(i)).refValue)
13    END IF
14  NEXT
15  oForm.UpdateInt(oForm.findColumn("Gruppe_multi"), inValue)
16 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*

```
1 DIM oDialogDat AS OBJECT
2
3 SUB DialogStart
4   DialogLibraries.LoadLibrary("Standard")
5   oDialogDat = createUnoDialog(DialogLibraries.Standard.Datumseingabe)
6   oDialogDat.Execute()
7 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: <i>Dialog Datumseingabe</i>

Benötigt
Makro: <i>DialogStart</i>
Tabelle: <i>Filter</i>

```
1 SUB DialogEnde
2   DIM stDatum AS STRING
3   oField = oDialogDat.getControl("datDatum")
4   stDatum = oField.Date.Year & "-" & Right("0" & Trim(Str(oField.Date.Month)),2) &
      "-" & Right("0" & Trim(Str(oField.Date.Day)),2)
5   oSQL_Anweisung = oVerbindung.createStatement()
6   stSql = "UPDATE ""Filter"" SET ""Halbjahrende"" = '"+stDatum+"' WHERE ""ID"" =
      TRUE "
7   oSQL_Anweisung.executeUpdate(stSql)
8   oDialogDat.EndExecute()
9 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: <i>Export_Start</i>

Benötigt
Makro: keins

```
1 SUB Ical_Export(stView AS STRING)
2   DIM oUcb AS OBJECT
3   DIM oOutputStream AS OBJECT
4   DIM oFile AS OBJECT
5   DIM oSQL_Anweisung AS OBJECT
6   DIM oAbfrageergebnis AS OBJECT
7   DIM stBaseUrl AS STRING
8   DIM stFileUrl AS STRING
9   DIM stString AS STRING
10  DIM stSql AS STRING
11  DIM stUID AS STRING
12  DIM stLocation AS STRING
13  DIM stSummary AS STRING
14  DIM stDescription AS STRING
15  DIM stDtStart AS STRING
16  DIM stDtEnd AS STRING
17  DIM stDtStamp AS STRING
18  DIM i AS INTEGER
19  DIM ar()
```

```

20  oUcb = createUnoService("com.sun.star.ucb.SimpleFileAccess")
21  oOutputStream = createUnoService("com.sun.star.io.TextOutputStream")
22  ar = split(ThisDatabaseDocument.URL, "/")
23  stBaseUrl = ""
24  FOR i = LBound(ar()) TO UBound(ar()) - 1
25      stBaseUrl = stBaseUrl & ar(i) & "/"
26  NEXT
27  stBaseFile = ar(i)
28  stFileUrl = stBaseUrl & "BaseTermine.ics"
29  IF oUcb.exists(stFileUrl) THEN
30      oUcb.kill(stFileUrl)
31  END IF
32  oFile = oUcb.OpenFileReadWrite(stFileUrl)
33  oOutputStream.SetOutputStream(oFile.getOutputStream)
34  stString = "BEGIN:VCALENDAR" & CHR(13) & CHR(10)
35  stString = stString & "VERSION:2.0" & CHR(13) & CHR(10)
36  stString = stString & "https://www.familiegrosskopf.de/robert/" & CHR(13) &
    CHR(10)
37  stString = stString & "METHOD:PUBLISH" & CHR(13) & CHR(10)
38  oOutputStream.writeString(stString)
39  oSQL_Anweisung = oVerbindung.createStatement()
40  stSql = "SELECT * FROM ""+stView+""""
41  oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
42  WHILE oAbfrageergebnis.next
43      stUID = oAbfrageergebnis.getString(1)
44      stLocation = oAbfrageergebnis.getString(2)
45      stSummary = oAbfrageergebnis.getString(3)
46      stDescription = oAbfrageergebnis.getString(4)
47      stDtStart = oAbfrageergebnis.getString(5)
48      IF Len(stDtStart) = 8 THEN
49          stDtStart = ";VALUE=DATE:"&stDtStart
50      ELSE
51          stDtStart = ":"&stDtStart
52      END IF
53      stDtEnd = oAbfrageergebnis.getString(6)
54      IF Len(stDtEnd) = 8 THEN
55          stDtEnd = ";VALUE=DATE:"&stDtEnd
56      ELSE
57          stDtEnd = ":"&stDtEnd
58      END IF
59      stDtStamp = oAbfrageergebnis.getString(7)
60      stString = "BEGIN:VEVENT" & CHR(13) & CHR(10)
61      stString = stString & "UID:" & stUID & stBaseFile & CHR(13) & CHR(10)
62      stString = stString & "LOCATION:" & stLocation & CHR(13) & CHR(10)
63      stString = stString & "SUMMARY:" & stSummary & CHR(13) & CHR(10)
64      stString = stString & "DESCRIPTION:" & stDescription & CHR(13) & CHR(10)
65      stString = stString & "CLASS:PUBLIC" & CHR(13) & CHR(10)
66      stString = stString & "DTSTART" & stDtStart & CHR(13) & CHR(10)
67      stString = stString & "DTEND" & stDtEnd & CHR(13) & CHR(10)
68      stString = stString & "DTSTAMP:" & stDtStamp & CHR(13) & CHR(10)
69      stString = stString & "END:VEVENT" & CHR(13) & CHR(10)
70      oOutputStream.writeString(stString)
71  WEND
72  stString = "END:VCALENDAR"
73  oOutputStream.writeString(stString)
74  oOutputStream.closeOutput()
75  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 erstellte 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 werden, 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*

```
1 SUB Export_Start(oEvent AS OBJECT)
2   DIM stView AS STRING
3   oForm = oEvent.Source.Model.Parent
4   oForm.updateRow
5   IF oForm.getByName("chkFerien").State = 1 THEN
6     stView = "Ansicht_Kalender_Export"
7   ELSE
8     stView = "Ansicht_Kalender_ohne_Ferien_Export"
9   END IF
10  Ical_Export(stView)
11 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_Stamp, LetzterSonntag*

Tabelle: *Termine, Filter*

```
1 FUNCTION Ical_Import(stFileUrl AS STRING) AS INTEGER
2     DIM oSQLAnweisung AS OBJECT
3     DIM oAbfrageergebnis AS OBJECT
4     DIM stSql AS STRING
5     DIM stRow AS STRING
6     DIM stBez AS STRING
7     DIM stBezeichnung AS STRING
8     DIM stBemerkung AS STRING
9     DIM stOrt AS STRING
10    DIM stDat AS STRING
11    DIM stStartDat AS STRING
12    DIM stStartZeit AS STRING
13    DIM stEndDat AS STRING
14    DIM stEndZeit AS STRING
15    DIM inBegin AS INTEGER
16    DIM inCount AS INTEGER
17    DIM daStart AS DATE
18    DIM daEnd AS DATE
19    DIM fn AS INTEGER
20    DIM ink AS INTEGER
21    DIM loID AS LONG
22    fn = freeFile
23    OPEN stFileUrl FOR INPUT AS #fn
24    DO WHILE NOT eof(#fn)
25        LINE INPUT #fn, stRow
26        IF stRow = "BEGIN:VEVENT" THEN
27            stBemerkung = "NULL,"
28            stOrt = "NULL,"
29            stEndDat = "NULL,"
30            stEndZeit = "NULL"
31            inBegin = 1
32        END IF
33        IF stRow = "END:VEVENT" THEN
34            oSQL_Anweisung = oVerbindung.createStatement()
35            stSql = "SELECT COUNT( ""ID"" ) FROM ""Termine"" WHERE ""Bezeichnung"" =
36                ""+stBez+"" AND ""Beginn_Datum"" = '"+stDat+'""
37            oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
38            WHILE oAbfrageergebnis.next
39                inCount = oAbfrageergebnis.getInt(1)
40            WEND
41            IF inCount = 0 THEN
```

```

41      stSql = "INSERT INTO ""Termine"" (""zieID"", ""Bezeichnung"",
42          ""Bemerkung"", ""Ort"", ""Beginn_Datum"", ""Beginn_Uhrzeit"",
43          ""Ende_Datum"", ""Ende_Uhrzeit"") VALUES ('0', ""+stBezeichnung+
44          stBemerkung+stOrt+stStartDat+stStartZeit+stEndDat+stEndZeit+"") "
45      oSQL_Anweisung.executeUpdate(stSql)
46      stSql = "CALL IDENTITY()"
47      oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
48      WHILE oAbfrageergebnis.next
49          loID = oAbfrageergebnis.getLong(1)
50      WEND
51      IF ink = 0 THEN
52          stSql = "UPDATE ""Filter"" SET ""StartIDImport"" = '"+loID+"',
53              ""EndIDImport"" = '"+loID+"' WHERE ""ID"" = TRUE"
54      ELSE
55          stSql = "UPDATE ""Filter"" SET ""EndIDImport"" = '"+loID+"' WHERE
56              ""ID"" = TRUE"
57      END IF
58      oSQL_Anweisung.executeUpdate(stSql)
59      ink = ink + 1
60  END IF
61  inBegin = 0
62  END IF
63  IF inBegin = 1 THEN
64      IF InStr(stRow, "SUMMARY:") THEN
65          stBez = Mid(stRow, inStr(stRow, ":")+1)
66          stBezeichnung = "" & stBez & ""
67      ELSEIF InStr(stRow, "DESCRIPTION:") THEN
68          stBemerkung = "" & Mid(stRow, inStr(stRow, ":")+1) & ""
69      ELSEIF InStr(stRow, "LOCATION:") THEN
70          stOrt = "" & Mid(stRow, inStr(stRow, ":")+1) & ""
71      ELSEIF InStr(stRow, "DTSTART") THEN
72          daStart = Ical_Stamp(Mid(stRow, inStr(stRow, ":")+1))
73          IF InStr(stRow, "UTC") OR Right(stRow, 1) = "Z" THEN
74              daStart = DateAdd("h", 1, daStart)
75              IF daStart > LetzterSonntag(DateValue(Year(daStart) & "-04-01")) AND
76                  daStart < LetzterSonntag(DateValue(Year(daStart) & "-11-01"))
77                  THEN
78                  daStart = DateAdd("h", 1, daStart)
79              END IF
80          END IF
81      ELSEIF InStr(stRow, "DTEND") THEN
82          daEnd = Ical_Stamp(Mid(stRow, inStr(stRow, ":")+1))
83          IF InStr(stRow, "UTC") OR Right(stRow, 1) = "Z" THEN
84              daEnd = DateAdd("h", 1, daEnd)
85              IF daEnd > LetzterSonntag(DateValue(Year(daEnd) & "-04-01")) AND
86                  daEnd < LetzterSonntag(DateValue(Year(daEnd) & "-11-01")) THEN
87                  daEnd = DateAdd("h", 1, daEnd)
88              END IF
89          END IF
90      stEndDat = "" & Year(daEnd) & "-" & Right("0" & Month(daEnd), 2)
91          & "-" & Right("0" & Day(daEnd), 2) & ""
92      IF InStr(Mid(stRow, inStr(stRow, ":")+1), "T") THEN
93          stEndZeit = "" & Right("0" & Hour(daEnd), 2) & ":" &
94              Right("0" & Minute(daEnd), 2) & ":" &
95              Right("0" & Second(daEnd), 2) & ""
96      ELSE
97          stEndZeit = "NULL"
98      END IF
99  END IF

```

```

95         ELSE
96         END IF
97     END IF
98     LOOP
99     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 beschriftet werden kann.

Ab Zeile 54 wird nach den Schlüsselbegriffen SUMMARY, DESCRIPTION, 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.

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 glei-

chem 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

```
1 FUNCTION IcalTimestamp(d AS STRING) AS DATE
2     DIM daT AS DATE
3     IF InStr(d,"T") THEN
4         daT = DateSerial(Left(d,4), Mid(d,5,2), Mid(d,7,2)) + TimeSerial(Mid(d,10,2),
5             Mid(d,12,2), Mid(d,14,2))
6     ELSE
7         daT = DateSerial(Left(d,4), Mid(d,5,2), Mid(d,7,2))
8     END IF
9     IcalTimestamp = daT
10 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*

BenötigtMakro: *Ical_Import*

```
1 SUB Import_Start(oEvent AS OBJECT)
2   DIM oFeld AS OBJECT
3   DIM oFeld1 AS OBJECT
4   DIM stFileUrl AS STRING
5   DIM inImport AS INTEGER
6   oFeld = oEvent.Source.Model
7   oFeld1 = oFeld.parent.getByName("Dateiauswahl")
8   IF oFeld1.Text <> "" THEN
9     stFileUrl = ConvertToUrl(oFeld1.Text)
10    inImport = Ical_Import(stFileUrl)
11  END IF
12  IF inImport > 0 THEN
13    oForm = oFeld.Parent.Parent.getByName("MainForm")
14    oForm.filter = ""ID"" BETWEEN (SELECT ""StartIDImport"" FROM ""Filter"" WHERE
      ""ID"" = True) AND (SELECT ""EndIDImport"" FROM ""Filter"" WHERE ""ID"" =
      True) "
15    oForm.ApplyFilter = TRUE
16    oForm.reload()
17    msgbox "Die importierten Datensätze werden im Formular angezeigt."
18  ELSE
19    msgbox "Es wurden keine Datensätze importiert."
20  END IF
21 END SUB
```

Es wird nicht direkt nach der Auswahl der Datei importiert, da dies bei einer 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). Werden keine importiert, so wird auch dies vermeldet (Zeile 19).

LetzterSonntag**Aufruf aus**Makro: *Ical_Import***Benötigt**

Makro: keins

```
1 FUNCTION LetzterSonntagVormonat(d AS DATE)
2   d = DateAdd("d", -1, d)
3   DO WHILE DatePart("w", d) <> 1
4     d = DateAdd("d", -1, d)
5   LOOP
6   d = DateValue(d) + TimeValue("02:00:00")
7   LetzterSonntag = d
8 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: <i>Termine_Tabelleneingabe</i>

Benötigt
Makro: keins

```

1 SUB Filter_Abgleich(oEvent AS OBJECT)
2   DIM oFeld AS OBJECT
3   DIM oForm1 AS OBJECT
4   DIM oForm2 AS OBJECT
5   DIM stTag AS String
6   DIM atTag()
7   oFeld = oEvent.Source.Model
8   stTag = oFeld.Tag
9   oForm1 = oFeld.parent
10  oFeld.commit()
11  oForm1.updateRow()
12  oForm2 = oForm1.parent.getByname(atTag(0))
13  IF UBound(atTag) > 0 THEN
14    oForm2.filter = atTag(1)
15    oForm2.ApplyFilter = TRUE
16  END IF
17  oForm2.reload()
18 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 ermittelt.

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: <i>Tabellenstart</i>

Benötigt

Makro: keins

```
1 Sub Tabellenindex_runter(stTabelle AS STRING)
2   DIM stAnzahl AS INTEGER
3   DIM inAnzahl AS INTEGER
4   DIM inSequence_Value AS INTEGER
5   oSQL_Anweisung = oVerbindung.createStatement()
6   stSql = "SELECT MAX("ID") FROM ""+stTabelle+""
7   oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
8   WHILE oAbfrageergebnis.next
9     stAnzahl = oAbfrageergebnis.getString(1)
10  WEND
11  IF stAnzahl = "" THEN
12    inAnzahl = -1
13  ELSE
14    inAnzahl = cInt(stAnzahl)
15  END IF
16  inSequence_Value = inAnzahl+1
17  oSQL_Anweisung1 = oVerbindung.createStatement()
18  oSQL_Anweisung1.executeQuery("ALTER TABLE ""+tablename+"" ALTER COLUMN "ID"
    RESTART WITH "+inSequence_Value+")
19 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*

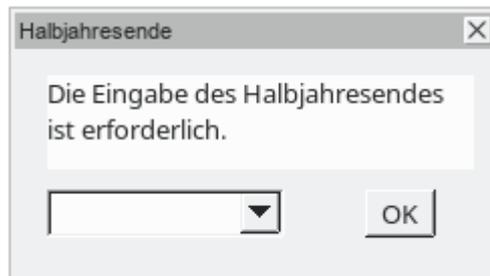
Tabelle: *Termine, Terminart, Zielgruppe, Ferien*

```
1 Sub Tabellenstart
2   Tabellenindex_runter("Termine")
3   Tabellenindex_runter("Terminart")
4   Tabellenindex_runter("Zielgruppe")
5   Tabellenindex_runter("Ferien")
6 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: <i>DialogStart</i>
Benötigt
Makro: <i>DialogEnde</i>



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.