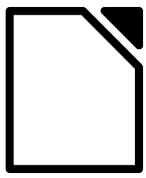


LibreOffice
The Document Foundation

Base

Praktische Beispiele für besondere Problemstellungen in Base

Dokumentationen zu LibreOffice unter de.libreoffice.org

Copyright

Mitwirkende/Autoren

Robert Großkopf

Rückmeldung (Feedback)

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

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

Datum der Veröffentlichung und Softwareversion

Veröffentlicht am 15.8.2014. Basierend auf der LibreOffice Version 4.3.

Anmerkung für Macintosh Nutzer

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

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

Inhalt

Wozu Datenbankbeispiele?

Für viele Nutzer ist die Hürde beim Erstellen von Datenbanken recht hoch. Schon die Erstellung von Tabellen, ihre Verknüpfung sowie die Beschickung in einfachen Formularen stellt häufig ein Problem dar.

Diese Probleme sollte allerdings das LO-Handbuch «Erste Schritte» sowie das Base-Handbuch bewältigen helfen. Diese Beispiele ergänzen vor allem das Base-Handbuch. Die Reihenfolge der Beispiele ist nicht vom Einfachen zum Komplizierten gewählt. Jedes Kapitel steht für sich. Es kann aber sein, dass in nachfolgenden Kapiteln auf bestimmte Konstruktionen des vorangegangenen Kapitel Bezug genommen wird. In diesem Falle also bitte einfach die entsprechende Stelle aufsuchen.

Diese Beschreibung deckt bisher nicht alle Beispieldatenbanken ab, die mit dem Base-Handbuch weiter gegeben werden. Die Beschreibung soll daher bis zur nächsten LO-Version weiter vervollständigt werden.

Zeitmessung in einer Datenbank

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 einer Datenbank auch Zeitmessungen durchführen zu können werden in geringem Umfang Makrokenntnisse gebraucht.

Tabellen

Die Datenbank besteht im sparsamsten Aufbau aus zwei Tabellen.

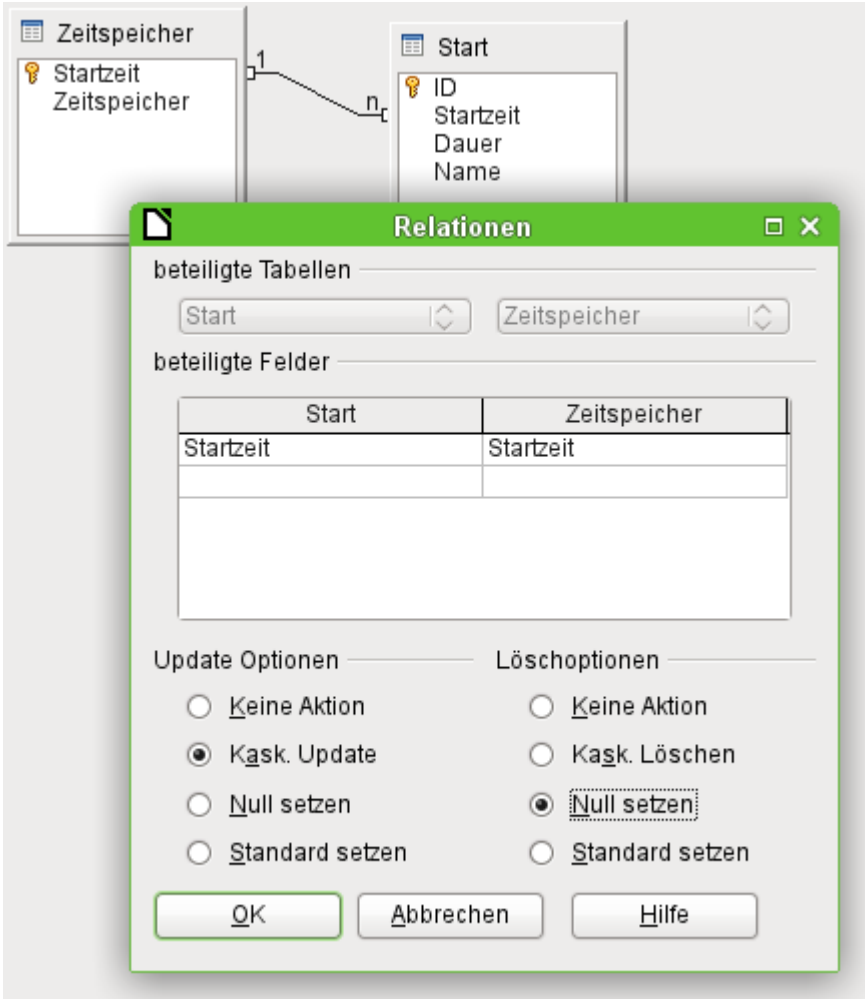
In der Tabelle "Start" werden folgende Felder definiert:

Feldname	***Feldtyp***	***Beschreibung***
ID	Integer	Primärschlüssel der Tabelle. Der Primärschlüssel muss eindeutig sein. Das Feld kann auch als Auto-Wert-Feld gesetzt werden.
Startzeit	Datum/Zeit	Hier wird Tag und Uhrzeit des vorgesehenen Startes festgehalten. Personen, die zusammen starten, haben die gleiche Startzeit.
Name	Text	Der Name der Person, die startet. Bei Schulwettkämpfen wären hier Zusätze wie Klasse, Schule oder gegebenenfalls auch Geburtsdaten notwendig.
Dauer	Integer	Die Dauer wird in Sekunden oder Millisekunden gemessen. Es handelt sich dabei um einen Zahlenwert ohne Nachkommastellen, keinen Wert wie bei einer Uhrzeit.
Zeit (nachträglich eingefügt)	Datum/Zeit	Die Zeitdauer wird hier als Zeit im Format Minuten:Sekunden,Hundertstelsekunden aus dem Formular «Einzelmessung_Zeitfeld» übernommen. Der Datumsteil wird durch ein Makro hinzugefügt.

In der Tabelle "Zeitspeicher" werden die folgenden Felder definiert:

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	Integer	Hier wird ein interner Basic-Wert für die tatsächliche Startzeit abgespeichert. Der Wert wird erst dann eingefügt, wenn der Button «Start» betätigt wurde.

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 werden und trotzdem ein genaues Messergebnis erwartet werden.

Zusätzlich gibt es eine Tabelle "Filter". Diese Tabelle dienst nur dazu, einen Filterwert aus einem Hauptformular an ein Unterformular weiter zu geben.

Feldname	*Feldtyp*	*Beschreibung*
ID	Ja/Nein	Die Filtertabelle soll nur einen Datensatz abspeichern. So lässt sie sich auch problemlos in Abfragen als Datenquelle für eine Unterabfrage nutzen.
Start_ID	Integer	Hier wird der Primärschlüssel der Tabelle "Start" gespeichert. Das ist im Formular der Wert für den Zieleinlauf. Die Eingabe einer Startnummer sollte beim Abspeichern die Zeit mit speichern können. Dies soll über die Filtertabelle erreicht werden.

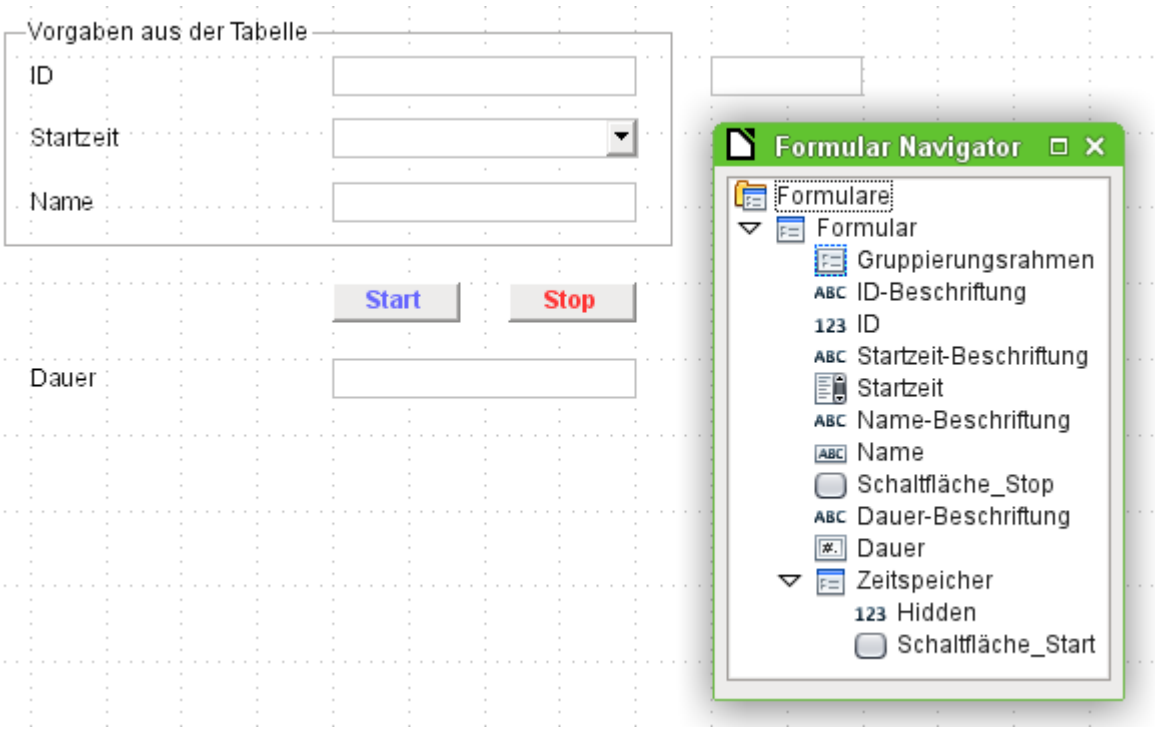
Formulare

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.

Das Formular «Einzelmessung»

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 Entwurfsansicht des Formulars gibt etwas genauer Aufschluss darüber, wie die Zeit ermittelt wird.

In dem Formular befindet sich ein Unterformular mit dem Namen «Zeitspeicher». Dieses Unterformular ist verbunden mit der Tabelle "Zeitspeicher". Das als unsichtbar definierte Element «Hidden» ist mit dem Datenfeld "Zeitspeicher" verbunden. Formular und Unterformular sind über die "Startzeit" verbunden.

Der Button Start liegt in dem Unterformular. Durch den Formularwechsel vom Hauptformular zum Unterformular wird beim Betätigen des Start-Buttons also der Inhalt des Hauptformulars abgespeichert. Wird Start betätigt, so schreibt ein Makro die interne Startzeit in das versteckte Feld.

Der Button Stop könnte auch im Unterformular liegen. Die Abspeicherung des gemessenen Wertes wird schließlich per Makro erledigt. Wird Stop betätigt, so wird über das Makro die Differenz zwischen dem Wert in dem versteckten Feld und dem momentanen Wert ermittelt und in das Feld "Dauer" übertragen. Die Messung erfolgt in Millisekunden.

Makros für das Formular «Einzelmessung»

Der Button Start startet über das Ereignis «Aktion ausführen» das Makro «Zeitmessung_Start».

```
SUB Zeitmessung_Start
    DIM oDoc AS OBJECT
    DIM oDrawpage AS OBJECT
    DIM oForm AS OBJECT
    DIM oFeld AS OBJECT
    DIM lZeit AS LONG
```

Zuerst werden die Variablen benannt und mit den ihnen entsprechenden Typen versehen. Bis auf eine Variable sind alle vom Typ «Object». Die Zeitangabe wird hingegen als Zahl gespeichert. Der Typ «Long» entspricht dem größten Zahlentyp ohne Nachkommastellen in StarBasic. Er entspricht vom Umfang her einem Integer-Feld der Datenbank.

```
    oDoc = thisComponent
    oDrawpage = oDoc.drawpage
    oForm = oDrawpage.forms.getByName("Formular")
    oSubForm = oForm.getByName("Zeitspeicher")
    oFeld = oSubForm.getByName("Hidden")
```

Das Feld «Hidden» wird aufgesucht. Es soll den internen Wert für die Startzeit speichern. Das Feld wird über das Formular und das darin liegende Unterformular erreicht. Die angegebenen Namen entsprechen den Namen, die im Formularnavigator sichtbar sind.

```
    'lZeit = Timer() ' Gibt die Systemzeit in Sekunden an
    lZeit = getSystemTicks() ' Gibt die Systemzeit in Millisekunden an
```

Die Zeitspeicherung kann entweder in Sekunden oder in Millisekunden vorgenommen werden. Hier ist die Variante für die Millisekunden ausgewählt. Die Systemzeit wird ausgelesen.

```
    oFeld.BoundField.UpdateInt(lZeit)
    oSubForm.UpdateRow()
END SUB
```

Anschließend wird die Systemzeit in das Feld «Hidden» übertragen. **`BoundField`** bedeutet hier das an dieses Feld gebundene Feld der Tabelle, in der der Wert abgespeichert werden soll. **`UpdateInt`** heißt, dass dieses Feld einen Integer-Wert abspeichern wird. Es ist in der Tabelle jetzt das Feld auf einen neuen Integer-Wert gesetzt worden. Dieser Wert ist jetzt in dem Feld der Tabelle enthalten und muss mit **`UpdateRow`** über das entsprechende Formular noch abgespeichert werden.

Der Button Stop startet über das Ereignis «Aktion ausführen» das Makro «Zeitmessung_Ende».

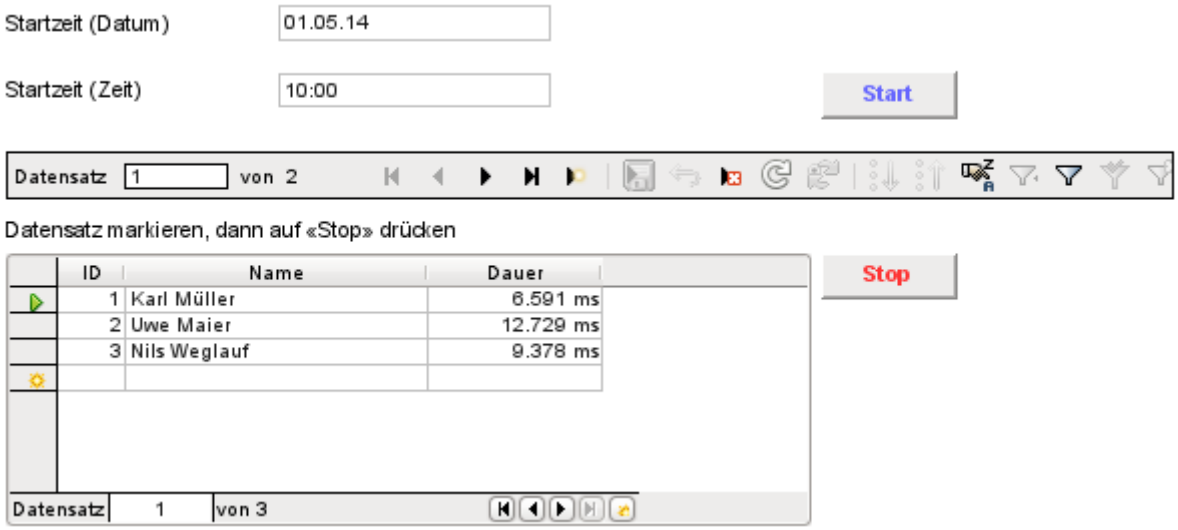
```
SUB Zeitmessung_Ende
    DIM oDoc AS OBJECT
    DIM oDrawpage AS OBJECT
    DIM oForm AS OBJECT
    DIM oFeld AS OBJECT
    DIM oFeld1 AS OBJECT
    DIM lZeit AS LONG
    DIM lZeit1 AS LONG
    oDoc = thisComponent
    oDrawpage = oDoc.drawpage
    oForm = oDrawpage.forms.getByName("Formular")
    oSubForm = oForm.getByName("Zeitspeicher")
    oFeld = oSubForm.getByName("Hidden")
    oFeld1 = oForm.getByName("Dauer")
```

Die Ansprache der Felder im Formular ist gleich. Jetzt wird allerdings noch ein zusätzliches das Feld «Dauer» angesteuert, in dem schließlich die gemessene Zeit gespeichert werden soll.

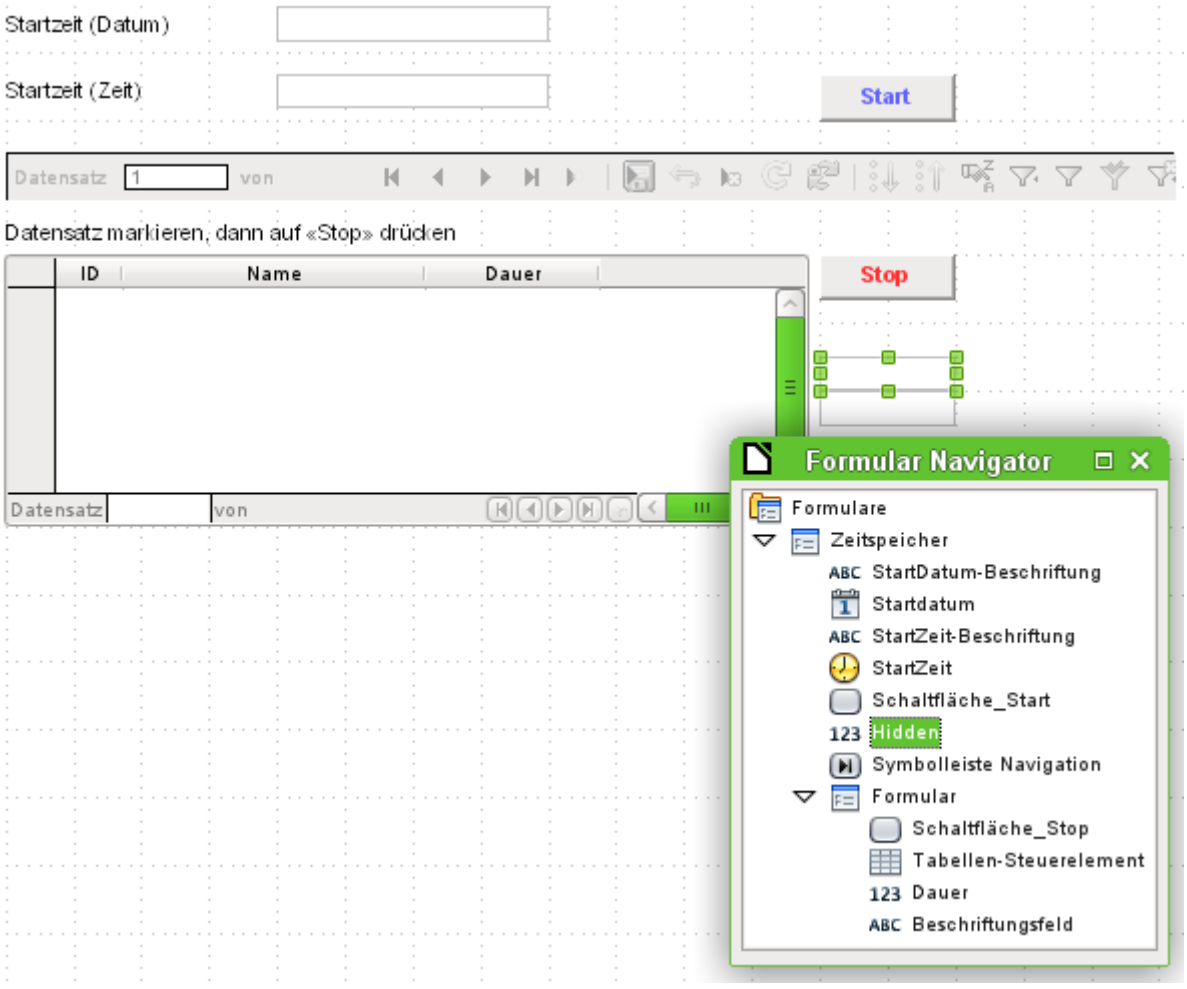
```
    'lZeit1 = Timer()   ' Gibt die Systemzeit in Sekunden an
    lZeit1 = getSystemTicks() ' Gibt die Systemzeit in Millisekunden an
    lZeit = oFeld.getCurrentValue
    oFeld1.BoundField.UpdateInt(lZeit1-lZeit)
    oForm.UpdateRow()
END SUB
```

Es wird die aktuellen Zeit, wie im vorhergehenden Makro, ausgelesen. Dann wird der gespeicherte Startwert aus dem Feld «Hidden» ausgelesen. Der Startwert wird von dem aktuellen Wert subtrahiert. Das Ergebnis der Subtraktion wird in das Feld «Dauer» übertragen und anschließend über das entsprechende Formular abgespeichert.

Das Formular «Startzeit_Gruppe»



Dieses Formular ermöglicht es, neue Startzeiten einzugeben. Den Startzeiten werden im Unterformular die Personen zugeordnet, die zu der jeweiligen Zeit an den Start gehen wollen. Hier ist z.B. eine Gruppe von 3 Personen am 1.5.14 um 10:00 Uhr gemeinsam gestartet. Die Zeit, die die einzelnen Personen für die Strecke benötigt haben, wird unter «Dauer» erfasst.



Im Hauptformular wird auf die Tabelle "Zeitspeicher" zugegriffen. In ihr wird die "Startzeit" und auch die systeminterne Startzeit (in dem Feld "Zeitspeicher") abgespeichert. Die systeminterne Startzeit

braucht nicht für den Nutzer sichtbar zu erscheinen. Das Feld mit der Bezeichnung «Hidden» ist deshalb einfach irgendwo auf dem Formular untergebracht.

Über Start wird dieses Feld mit der systeminternen Startzeit versorgt.

Die Tabelle "Start" liegt jetzt im Unterformular. In ihr wird auch der Zieleinlauf des Rennens gespeichert. Verbindungsglied zwischen Hauptformular und Unterformular ist das Feld "Startzeit". Die Zeit, die die jeweilige Person für die Strecke benötigt hat, wird in dem außerhalb des Tabellenkontrollfeldes versteckten Feld «Dauer» abgespeichert. Dieses Feld ist über Makros erst einmal einsichtiger zu erreichen als ein Feld innerhalb des Tabellenkontrollfeldes. Gleichzeitig zeigt es auch, dass Werte des aktuellen Datensatzes des Tabellenkontrollfeldes außerhalb des Tabellenkontrollfeldes angezeigt oder eingegeben werden können. So etwas kann z.B. auch für Textfelder mit viel Inhalt oder auch für abgespeicherte Bilder sinnvoll sein.

Makros für das Formular «Startzeit_Gruppe»

```
SUB Zeitmessung_Massenstart
    DIM oDoc AS OBJECT
    DIM oDrawpage AS OBJECT
    DIM oForm AS OBJECT
    DIM oFeld AS OBJECT
    DIM lZeit AS LONG
    oDoc = thisComponent
    oDrawpage = oDoc.drawpage
    oForm = oDrawpage.forms.getByName("Zeitspeicher")
    oFeld = oForm.getByName("Hidden")
    'lZeit = Timer() ' Gibt die Systemzeit in Sekunden an
    lZeit = getSystemTicks() ' Gibt die Systemzeit in Millisekunden an
    oFeld.BoundField.UpdateInt(lZeit)
    IF oForm.RowInserted() THEN
        oForm.InsertRow()
    ELSE
        oForm.UpdateRow()
    END IF
END SUB
```

Da der Formularaufbau (Hauptformular zu Unterformular) umgedreht wurde müssen die Felder entsprechend anders angesprochen werden. Im Hauptformular wird jetzt der Start eingetragen. Es wäre zwar sinnlos, einen Start ohne entsprechend zugeordnete Personen ablaufen zu lassen. Trotzdem soll hier einfach über Start auch abgesichert werden, dass es bei einer Neueingabe von Daten nicht zu einer Fehlermeldung kommt. Aus diesem Grunde wird abgefragt, ob es sich um eine eingefügten Datensatz handelt (**`oForm.RowInserted`**). Ist der Datensatz neu einzufügen, dann müssen die Daten an die Tabelle mit **`oForm.InsertRow`** weitergegeben werden. Ansonsten handelt es sich um eine Datenänderung, also **`oForm.UpdateRow`**.

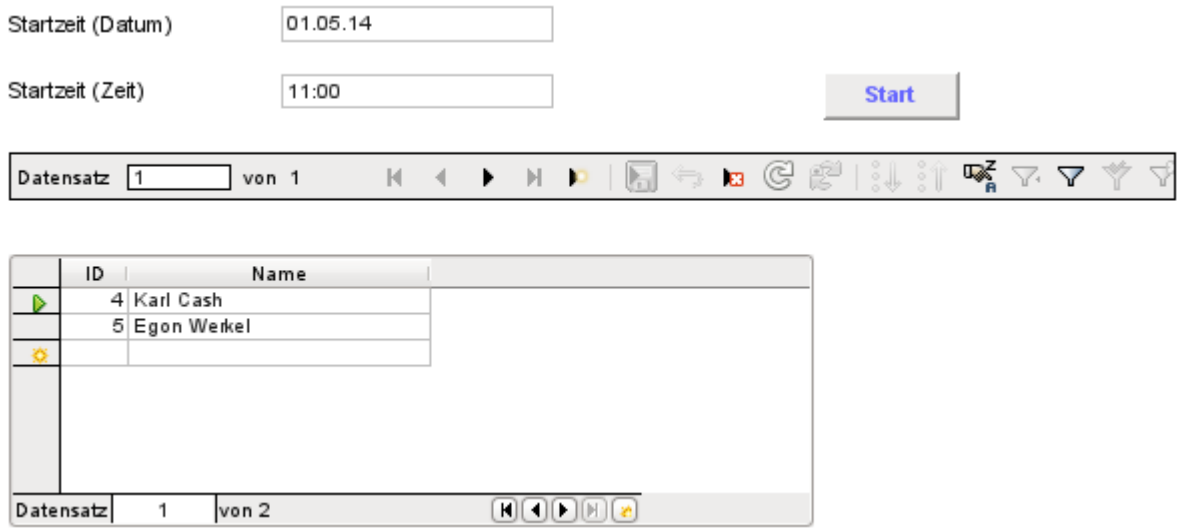
```
SUB Zeitmessung_Ende_Massenstart
    DIM oDoc AS OBJECT
    DIM oDrawpage AS OBJECT
    DIM oForm AS OBJECT
    DIM oFeld AS OBJECT
    DIM oFeld1 AS OBJECT
    DIM lZeit AS LONG
    DIM lZeit1 AS LONG
    oDoc = thisComponent
    oDrawpage = oDoc.drawpage
    oForm = oDrawpage.forms.getByName("Zeitspeicher")
    oSubForm = oForm.getByName("Formular")
    oFeld = oForm.getByName("Hidden")
    oFeld1 = oSubForm.getByName("Dauer")
    'lZeit1 = Timer()    ' Gibt die Systemzeit in Sekunden an
    lZeit1 = getSystemTicks() ' Gibt die Systemzeit in Millisekunden an
    lZeit = oFeld.getCurrentValue
    oFeld1.BoundField.UpdateInt(lZeit1-lZeit)
    oSubForm.UpdateRow()
END SUB
```

Die Zeitmessung wird genau so wie im ersten Formular gestoppt. Unterschied zwischen den Makros ist hier lediglich die entsprechende Abfrage der Felder. Das Feld «Dauer» liegt dabei im Unterformular. Um das Feld innerhalb des Tabellenkontrollfeldes anzusprechen wäre noch eine zusätzliche Zeile notwendig gewesen. Der Zugriff über Namen kann dabei nur funktionieren, wenn unterschiedliche Namen vergeben wurden. Ansonsten müssen Felder über **`getByIndex`** angesprochen werden.

Das Formular «Nur_Start»

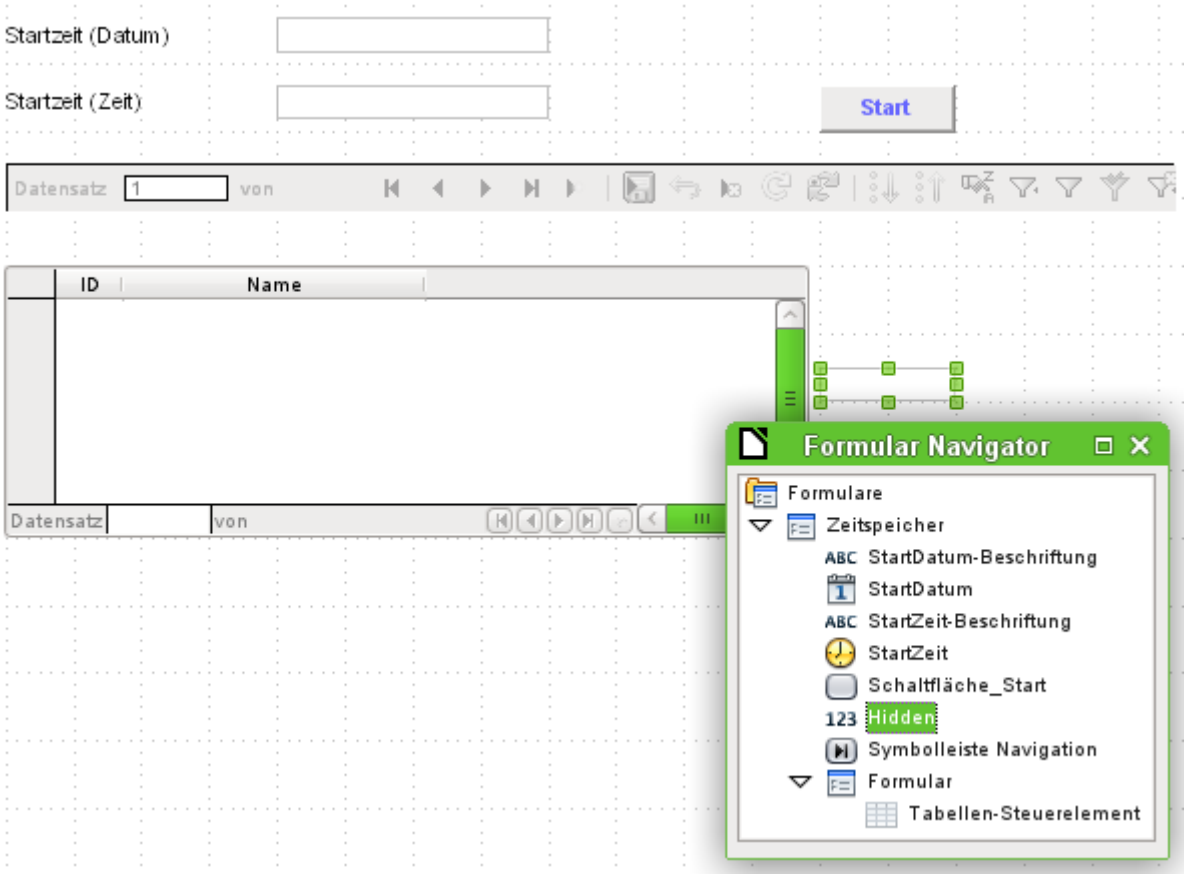
Bei den bisherigen Konstruktionen kann es ohne weiteres passieren, dass ein Start mehrmals hintereinander eingegeben, also die Zeit nicht korrekt gemessen wird. Dies lässt sich vermeiden, wenn nach dem erfolgten Start ein erneutes Beschreiben des Start-Wertes unmöglich gemacht wird.

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

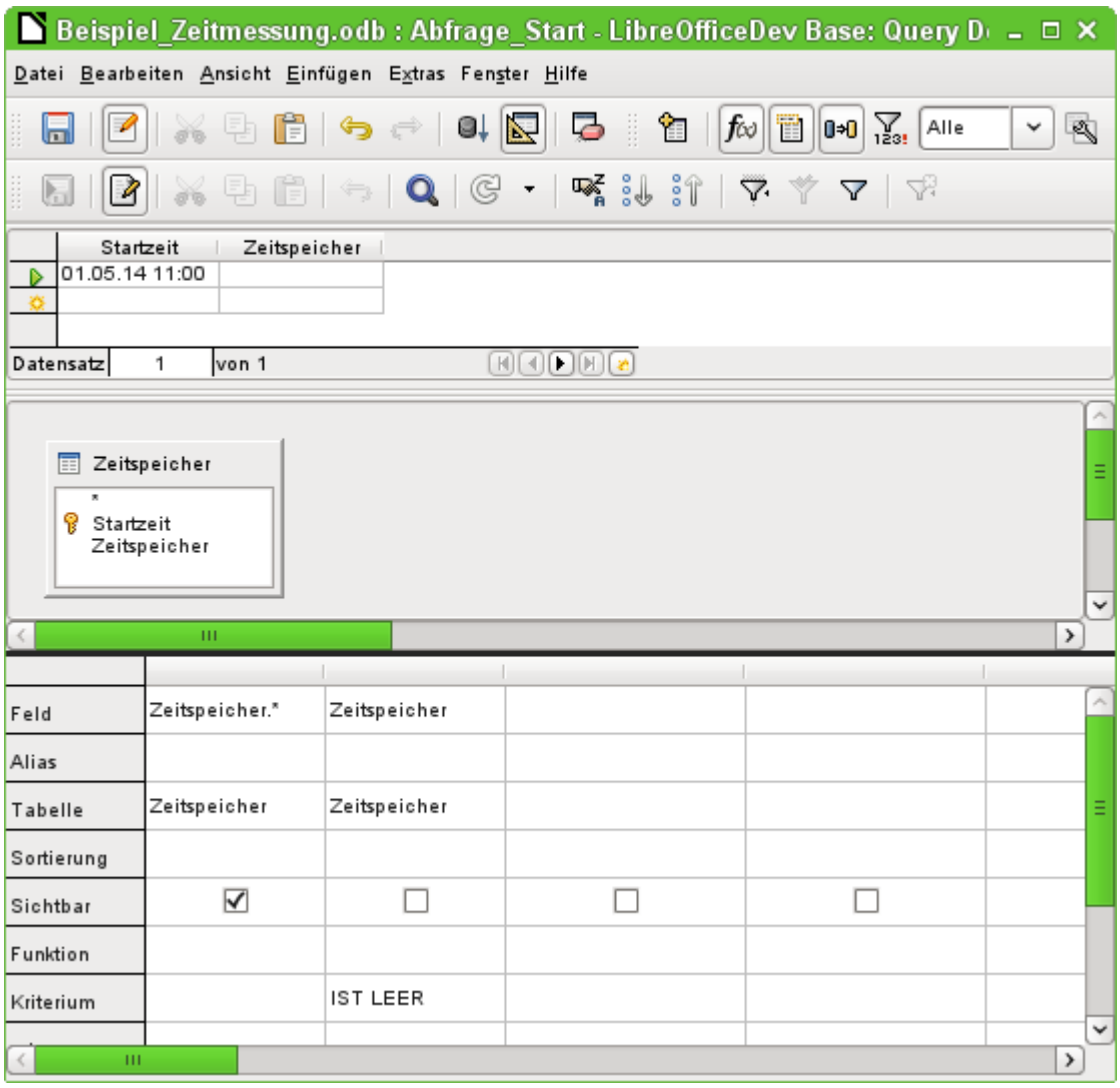


Die Formularansicht ist gegenüber dem Formular «Startzeit_Gruppe» etwas reduziert. Natürlich gibt es in einem reinen Star-Formular keinen Stop-Button. Außerdem werden nur die Namen der Personen aufgelistet, die zu der entsprechenden Zeit am Start stehen müssen. Die «Dauer», die sie unterwegs sind, wird in diesem Formular nicht erfasst.

Auch der Formularnavigator zeigt nur eine reduzierte Ansicht des Formulars «Startzeit_Gruppe». Elemente für den Zieleinlauf fehlen.



Als Grundlage für das Formular «Zeitspeicher» fungiert hier allerdings nicht die Tabelle "Zeitspeicher", sondern eine Abfrage:



Die Abfrage bezieht sich auf die Tabelle "Zeitspeicher". Sie zeigt alle Felder dieser Tabelle an (im ersten Feld des grafischen Editors steht: Zeitspeicher.*). Da sich die Abfrage nur auf diese eine Tabelle bezieht ist sie so auf jeden Fall editierbar.

Im zweiten Feld wird definiert, dass "Zeitspeicher" auf jeden Fall leer sein soll, wenn der Datensatz angezeigt wird. Alle Felder mit ausgefülltem "Zeitspeicher" erscheinen also nicht. Die so formulierte Abfrage lautet schließlich:

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

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

rechte Maustaste über dem Formularnamen «Zeitspeicher» → Eigenschaften → Daten → Filter

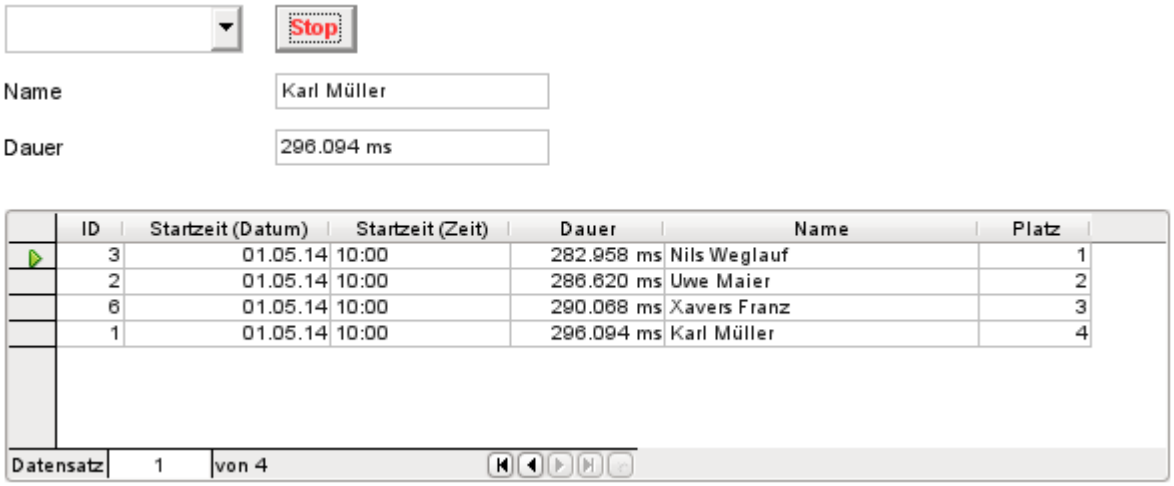
In das freie Feld kann eingetragen werden: **`("Zeitspeicher" IS NULL)`**

Makros für das Formular «Nur_Start»

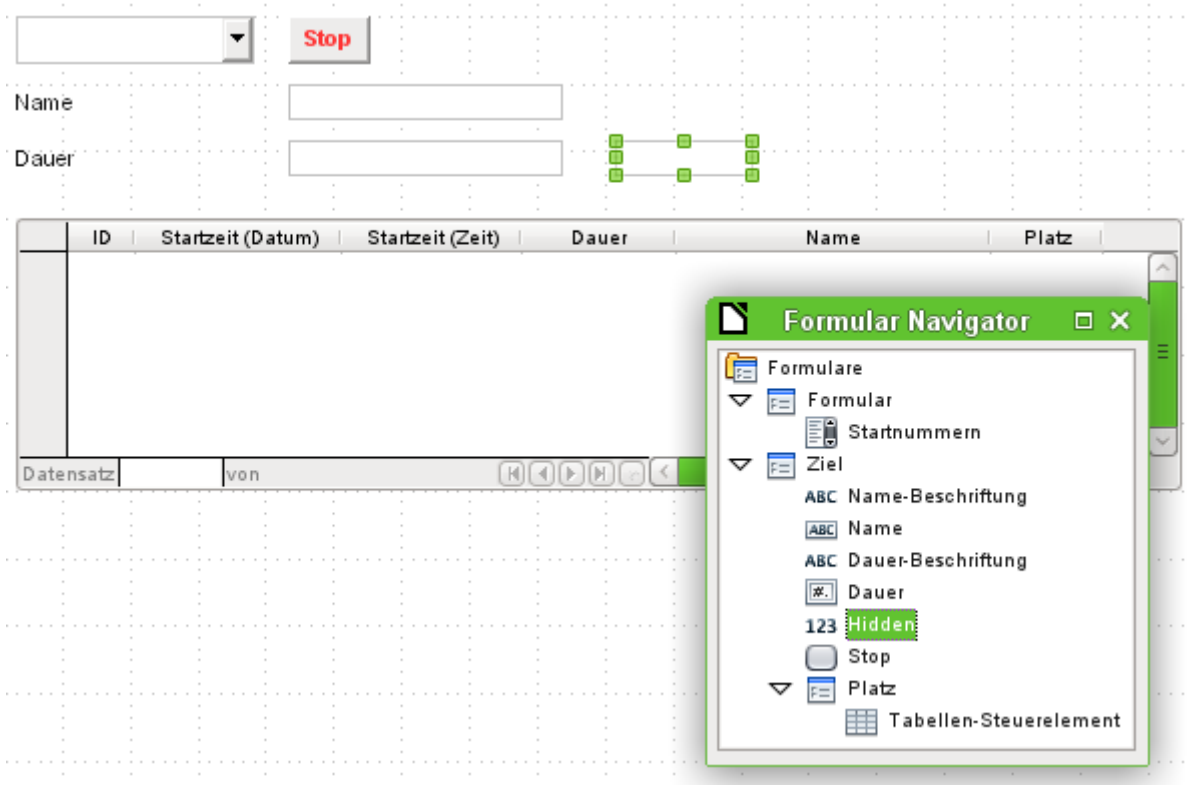
Das Formular «Nur_Start» nutzt das Makro «Zeitmessung_Massenstart», das bereits im vorangegangenen Formular über den Button Start ausgelöst wurde.

Das Formular «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.



Hier ist gerade Karl Müller als 4. Person ins Ziel gekommen. Die Startnummer ("ID") von Karl Müller wurde aus dem Listenfeld direkt neben dem Button Stop ausgewählt, als sich Karl Müller dem Ziel näherte. Stop wurde gedrückt, 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 die Startnummer «1» von Karl Müller jetzt nicht mehr an, da Karl Müller durchs Ziel gekommen ist.



Der Formularnavigator zeigt gleich drei Formulare. Das erste Formular «Formular» enthält nur ein Listenfeld für die «Startnummern». Es ist von den anderen Formularen getrennt, liegt also neben dieses Formularen auf der Benutzeroberfläche. Die Anzeige von «Formular» soll keine Auswirkung

auf die anderen Formulare haben. Schließlich ist ein Wert, der im Listenfeld «Startnummern» noch verzeichnet ist, nicht in den anderen Formularen zu finden.

Das Formular «Formular» hat als Datengrundlage die Tabelle "Filter" angegeben. Das Formular ist nur zur Änderung von Daten vorgesehen. Es sollen keine Daten gelöscht werden und keine neuen Datensätze angelegt werden. Die Tabelle "Filter" besteht so nur aus einem Datensatz.

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.

```
SELECT "Start"."ID", "Start"."ID"
    FROM "Start", "Zeitspeicher"
    WHERE "Start"."Startzeit" = "Zeitspeicher"."Startzeit"
        AND "Zeitspeicher"."Zeitspeicher" IS NOT NULL
        AND "Start"."Dauer" IS NULL
```

Diese Abfrage stellt zweimal das gleiche Feld "Start"."ID". Das ist notwendig, wenn in den **Eigenschaften des Listenfeldes → Daten** Gebundenes Feld = 1 steht. In neueren Versionen von LO kann mit der Einstellung Gebundenes Feld = 0 auch nur ein Feld angegeben werden.

Die Abfrage stellt zuerst einmal die Beziehung zwischen den Tabellen "Start" und "Zeitspeicher" her. Dann muss die Bedingung erfüllt sein, dass "Zeitspeicher"."Zeitspeicher" nicht leer ist. Das bedeutet, dass eben ein Start bereits erfolgt ist. Und schließlich sollen nur die Datensätze erscheinen, bei denen in "Start"."Dauer" bisher kein Eintrag existiert. 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:

```
("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:

```
( "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:

```
"Platz" ASC
```

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

Die Abfrage, auf die sich die Formulare «Ziel» und «Platz» beziehen, sieht im SQL-Code so aus:

```
SELECT "a".*,
    ( SELECT "Zeitspeicher" FROM "Zeitspeicher" WHERE "Startzeit" = "a"."Startzeit" ) AS
        "Zeitspeicher",
    ( SELECT COUNT( "ID" ) FROM "Start" WHERE "Dauer" <= "a"."Dauer" AND "Startzeit" =
        "a"."Startzeit" ) AS "Platz"
    FROM "Start" AS "a"
    WHERE "Zeitspeicher" IS NOT NULL
```

Bei dieser Abfrage handelt es sich um eine korrelierte Unterabfrage. Innerhalb der Abfrage wird auf Werte der aktuellen Abfrage Bezug genommen. Um so eine korrelierte Abfrage zu erstellen muss der Tabelle, auf der die äußere Abfrage beruht, ein Alias zugewiesen werden: **`"Start" AS "a"`**.

Mit **`"a".*`** werden alle Felder der Tabelle "Start" angezeigt. Die Abfrage erfüllt also eine wichtige Voraussetzung dafür, dass sie editierbar ist, indem der Primärschlüssel aus der Tabelle "Start" auch in der Abfrage auftaucht.

Der **"Zeitspeicher"** wird hier durch eine Unterabfrage abgerufen, da sonst die Abfrage nicht mehr editierbar wäre. Die Unterabfrage sucht in der Tabelle **"Zeitspeicher"** nach dem Wert für das Feld **"Zeitspeicher"**. Der Wert muss mit der "Startzeit" des aktuellen Datensatzes von **"Start"**, also **"a"."Startzeit"**, übereinstimmen. Das trifft jeweils auf höchstens einen Datensatz zu und kann also in einer Unterabfrage verwandt werden. Unterabfragen, die mehrere Datensätze ergeben, können in Feldern einer Abfrage nicht genutzt werden.

Die zweite korrelierte Unterabfrage ermittelt die aktuelle Platzierung der im jeweiligen Datensatz repräsentierten Person. Es werden alle Einträge im Feld **"ID"** gezählt. Die **"Dauer"** dieser Einträge muss allerdings kleiner oder gleich der Dauer des aktuellen Datensatzes, also **"a"."Dauer"** sein. Außerdem muss die **"Startzeit"** gleich der Startzeit des aktuellen Datensatzes also **"a"."Startzeit"** sein.

Mit dieser Abfrage "Abfrage_Ziel" lässt sich der gesamte Zieldurchlauf mit Platzierungen darstellen. Wird für die Dauer des Rennens kein Wert im Millisekunden-Bereich benötigt, sondern z.B. nur im Zehntelsekunden-Bereich, so kann in so einer Abfrage entsprechend nachgebessert werden.

Makro für das Formular «Nur_Ziel»

```
SUB Zeitmessung_nur_Ziel
    DIM oDoc AS OBJECT
    DIM oDrawpage AS OBJECT
    DIM oForm AS OBJECT
    DIM oForm2 AS OBJECT
    DIM oSubForm2 AS OBJECT
    DIM oFeld AS OBJECT
    DIM oFeld1 AS OBJECT
    DIM lZeit AS LONG
    DIM lZeit1 AS LONG
    oDoc = thisComponent
    oDrawpage = oDoc.drawpage
    oForm = oDrawpage.forms.getByName("Formular")
    oForm2 = oDrawpage.forms.getByName("Ziel")
    oSubForm2 = oForm2.getByName("Platz")
    oFeld = oForm2.getByName("Hidden")
    oFeld1 = oForm2.getByName("Dauer")
    oForm.UpdateRow()
    oForm2.Reload()
    oForm2.Last()
    'lZeit1 = Timer()   ' Gibt die Systemzeit in Sekunden an
    lZeit1 = getSystemTicks() ' Gibt die Systemzeit in Millisekunden an
    lZeit = oFeld.getCurrentValue
    oFeld1.BoundField.UpdateInt(lZeit1-lZeit)
    oForm2.UpdateRow()
    oForm.getByName("Startnummern").Refresh()
END SUB
```

Zuerst werden die verschiedenen Formularfelder, wie vorher bereits, zugänglich gemacht. Dann wird bei Betätigung des Buttons Stop das Formular "Formular" mit **oForm.UpdateRow** abgespeichert. Anschließend wird das Formular "Ziel" mit **oForm2.Reload** neu geladen. Da hier der Datensatzzeiger in LibreOffice 4.3.0.0 Beta1 (aus nicht nachvollziehbaren Gründen) nicht auf den letzten Datensatz springt, sondern einen neuen Datensatz erstellen will, wird zusätzlich der Datensatzzeiger des Formulars auf den letzten (einzigen) Datensatz eingestellt: **oForm2.Last**.

Das Auslesen und Eintragen des Zeitmesswertes erfolgt wie bereits bei den anderen makros erklärt. Gleiches gilt auch für die entsprechende Abspeicherung.

Schließlich wird noch das Listenfeld für die Startnummern neu eingelesen, da ja jetzt genau der Datensatz fehlt, den es vorher angezeigt hatte:
oForm.getByName("Startnummern").Refresh().

Das Formular «Einzelmessung_Zeitfeld»

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

Um diese Zeitangabe entsprechend speichern zu können muss zuerst einmal der Tabelle «Start» ein zusätzliches Feld «Zeit» hinzugefügt werden. Dieses Feld muss als «Datum/Zeit [TIMESTAMP]» - Feld definiert werden. Die reinen Zeitfelder sind 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:

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

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

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

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

Makro für das Formular «Einzelmessung_Zeitfeld»

```
SUB Zeitmessung_Timestamp
    DIM unoStmp AS NEW com.sun.star.util.DateTime
    DIM daStart AS NEW com.sun.star.util.DateTime
    DIM oDoc AS OBJECT
    DIM oDrawpage AS OBJECT
    DIM oForm AS OBJECT
    DIM oFeld AS OBJECT
    DIM oSubForm AS OBJECT
    DIM oSubFeld AS OBJECT
    DIM stZeit AS STRING
    DIM ar()
    DIM arMandS()
    DIM loNano AS LONG
    DIM inSecond AS INTEGER
    DIM inMinute AS INTEGER
    DIM inIndex AS INTEGER
    DIM lZeit AS LONG
```

```
    DIM lZeit1 AS LONG
```

Sämtliche Variablen werden zuerst deklariert. Hier ist darauf zu achten, dass die entsprechenden Zuweisungen für die Timestamp-Felder und für die Sekundenbruchteile, die Sekunden und Minuten stimmen. Die Zeitdefinition für die Sekundenbruchteile wird über das Struct von LibreOffice auf Nanosekunden festgelegt und benötigt daher eine Ganzzahl, die entsprechend groß genug ist, also vom Typ Long.

```
    oDoc = thisComponent
    oDrawpage = oDoc.Drawpage
    oForm = oDrawpage.forms.getByName("Formular")
    oColumns = oForm.Columns
    inIndex = oForm.findColumn("Startzeit")
    daStart = oForm.getTimestamp(inIndex)
```

Das Formular wird angesteuert. Der Timestamp für den Start wird aus der dem Formular zugrundeliegenden Tabelle ausgelesen, damit aus diesem Feld der Datumsteil für das Feld «Zeit» ausgelesen werden kann, in dem schließlich ein kompletter Timestamp gespeichert werden muss.

Alternativ kann auch einfach das Standardstartdatum 1899-12-30 eingesetzt werden. Schließlich wird der Datumswert für die Zeitauswertung nicht benötigt.

```
    oFeld = oForm.getByName("Zeit")
    oSubForm = oForm.getByName("Zeitspeicher")
    oSubFeld = oSubForm.getByName("Hidden")
    lZeit1 = getSystemTicks()
    lZeit = oSubFeld.getCurrentValue
    doZeit = (Cdbl(lZeit1-lZeit))/1000
```

Die Zeit wird in Sekunden umgerechnet. Damit auch Nachkommastellen angezeigt werden, muss die Differenz der Felder **`lZeit1-lZeit`** mit **`Cdbl`** in eine Double-Variable umgewandelt werden. Die Tausendstel erscheinen jetzt hinter dem Dezimalpunkt. Die Zeitangabe wird in einen Text umgewandelt und am Dezimalpunkt aufgetrennt.

```
    ar() = Split(Str(doZeit),".")
    loNano = Clng(ar(1)&"000000")
```

Die Tausendstel-Sekunden werden aus dem 2. Teil des Arrays ausgelesen. Es müssen noch 6 Nullen angehängt werden, da Nanosekunden gespeichert werden.

```
    doZeit = Fix(doZeit)
    inMinute = Fix(doZeit/60)
    inSecond = doZeit - inMinute*60
```

Mit der Funktion Fix werden die Nachkommastellen der Zeitvariablen abgeschnitten. Für die Minutenangabe werden die verbleibenden Sekunden durch 60 dividiert. Die Sekunden werden ermittelt, indem der Rest aus der vorhergehenden Division bestimmt wird.

```
    WITH unoStmp
        .NanoSeconds = loNano
        .Seconds = inSecond
        .Minutes = inMinute
        .Hours = 0
        .Day = daStart.Day
        .Month = daStart.Month
        .Year = daStart.Year
    END WITH
    oFeld.BoundField.updateTimestamp(unoStmp)
    oForm.UpdateRow()
END SUB
```

Die einzelnen Variablen der Timestamps werden gesetzt. In manchen Anleitungen steht hierzu noch statt des Anteils **`NanoSeconds`** der Anteil **`HundrethSeconds`**. Für LibreOffice in der aktuellen Fassung ist diese Angabe nicht mehr korrekt.

Zum Schluss wird der Zeitstempel in das Anzeigefeld für die Zeit geschrieben und der Datensatz mit dem neuen Wert überschrieben.

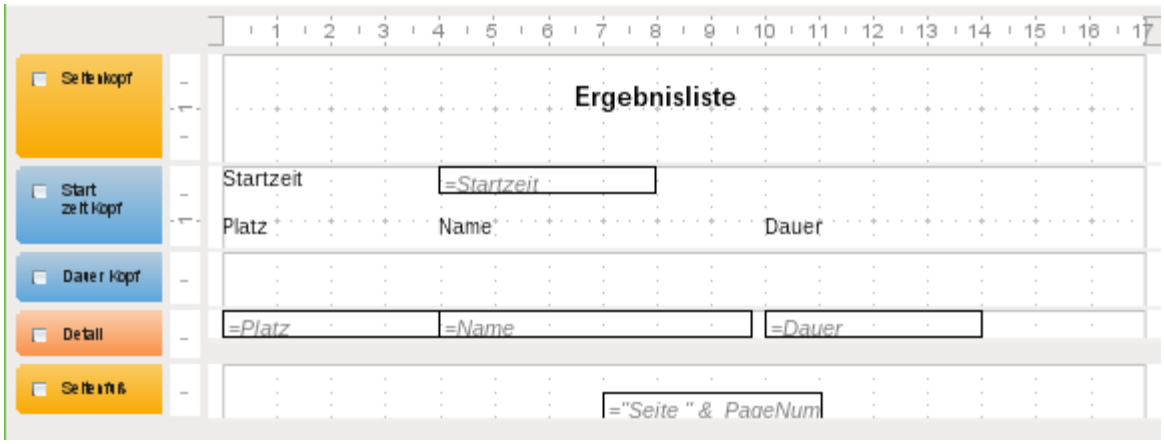
Berichte

Ergebnisliste

Startzeit	01.05.14 10:00	
Platz	Name	Dauer
1	Nils Weglauf	282958 ms
2	Uwe Maier	286620 ms
3	Xavers Franz	290068 ms
4	Karl Müller	296094 ms

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.

Im Entwurf sieht dieser Bericht so aus:



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

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

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

Urkunde

Nils Weglauf

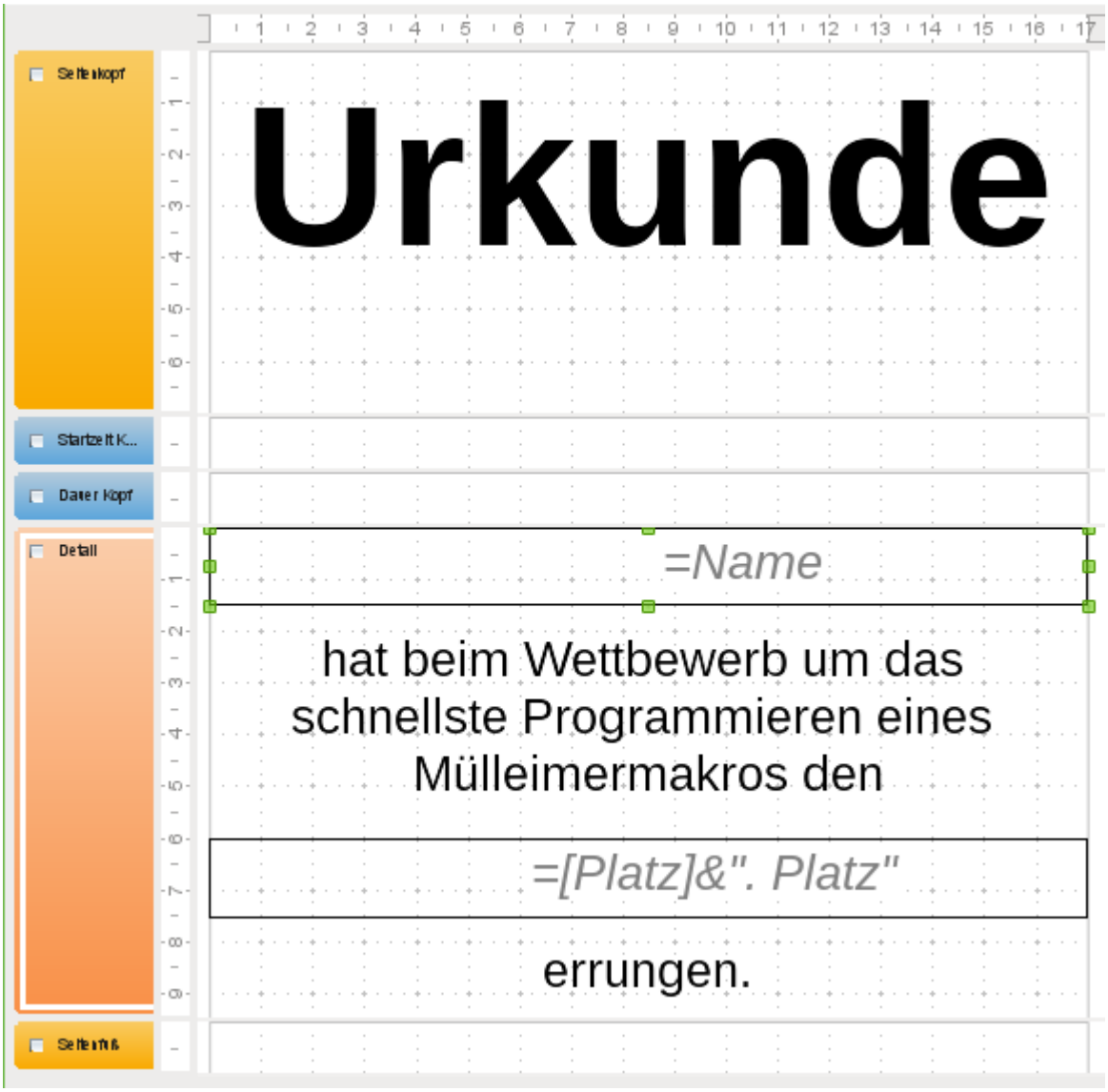
hat beim Wettbewerb um das
schnellste Programmieren eines
Mülleimermakros den

1. Platz

errungen.

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.

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.

Serienbriefe direkt aus Base heraus

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.

Tabellen

Die Tabellen haben in dieser Datenbank nur die Funktion, für eine Bestückung der Serienbriefe eine Grundlage zu bieten. Sie werden deshalb nur wenig zusätzlich erklärt.

In der Tabelle "Anschrift" werden folgende Felder definiert:

Feldname	***Feldtyp***	***Beschreibung***
ID	Integer	Die ID ist Primärschlüssel dieser Tabelle. Sie ist als Autowert festgelegt.
Vorname	Text	
Nachname	Text	
Geschlecht	Text	Länge des Textes ist 1 Zeichen. Hier wird nur 'm' oder 'w' eingetragen, damit später daraus eine Anrede ermittelt werden kann.
Straße_Nr	Text	
Postleitzahl	Text	Länge des Textes ist 5 Zeichen. Postleitzahlen sollten nicht als Zahlen abgespeichert werden. Führende Nullen sind sonst nicht möglich.
Ort	Text	

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.

In der Tabelle "Rechnung" werden die folgenden Felder definiert:

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.
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.

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

In der Tabelle "Rechnungsinhalt" werden die folgenden Felder definiert:

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.
Rechnungsin-	Text	Länge des Textes sollte groß gewählt werden, da hier für eine

halt		der Serienbrieffunktionen alle Rechnungsinhalte in einem Datensatz gespeichert werden. Im Beispiel ist der Länge auf 10000 Zeichen festgelegt.

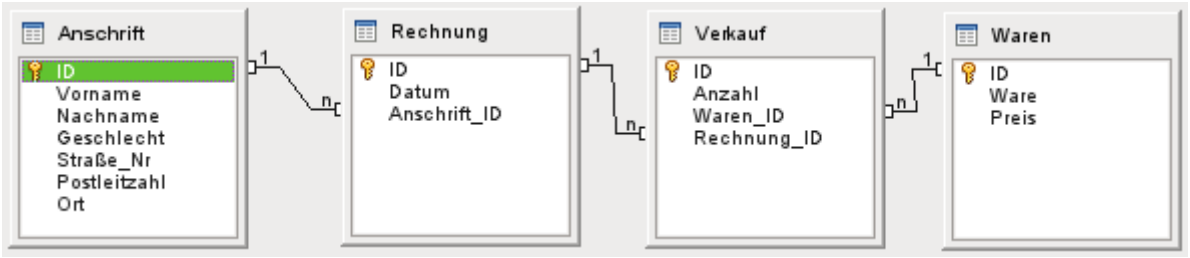
In der Tabelle "Verkauf" werden die folgenden Felder definiert:

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.
Waren_ID	Integer	Hier wird die "ID" der Tabelle "Waren" als Fremdschlüssel gespeichert.
Rechnung_ID	Integer	Hier wird die "ID" der Tabelle "Rechnung" als Fremdschlüssel gespeichert.

In der Tabelle "Waren" werden die folgenden Felder definiert:

Feldname	***Feldtyp***	***Beschreibung***
ID	Integer	Die ID ist Primärschlüssel dieser Tabelle. Sie ist als Autowert festgelegt.
Ware	Text	
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. Für die letztlich zu erfolgende Darstellung im Druck hat dies aber keine Bedeutung.

Die Tabellen sind für die Rechnungserstellung folgendermaßen verbunden:



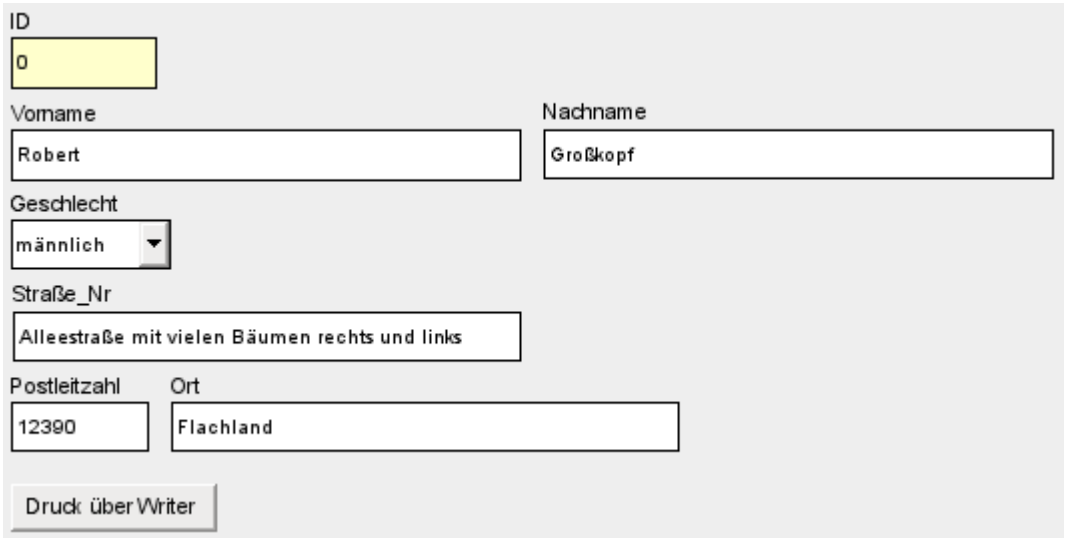
Die Tabelle Anschrift wird separat bestückt und dient als erstes Beispiel für den Seriendruck. Die Kombination mit der Rechnungserstellung zeigt dann ein erweitertes Anwendungsbeispiel.

Formulare

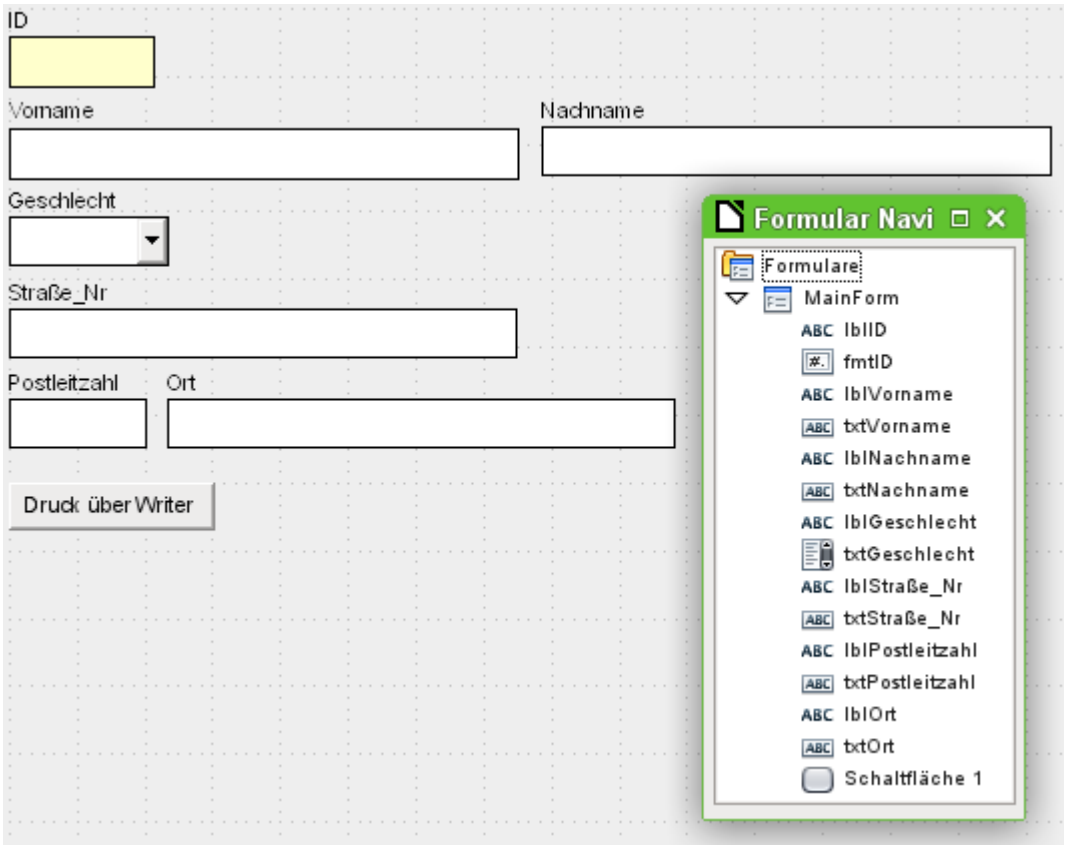
Die Formulare greifen mit unterschiedlichem Schwierigkeitsgrad auf die Serienbrieffunktion bzw. auf Textfelder eine Vorlage zu. Der einfachste Zugriff erfolgt für den Inhalt, der in der Tabelle "Anschrift" gespeichert wird.

Das Formulare «Anschrift»

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 Feld "Geschlecht" wird über ein Listenfeld beschrieben. Der Druck des aktuellen Datensatzes kann direkt über den Writer erfolgen. Über den Button wird hierzu ein Makro ausgeführt. Mehr dazu etwas weiter unten.



Der Formularnavigator zeigt keine Besonderheiten auf. Bei einem Formular, das lediglich zum Bestücken einer Tabelle genutzt wird, ist dies auch nicht weiter verwunderlich.

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.

Ausdruck aus dem Formular «Anschrift»

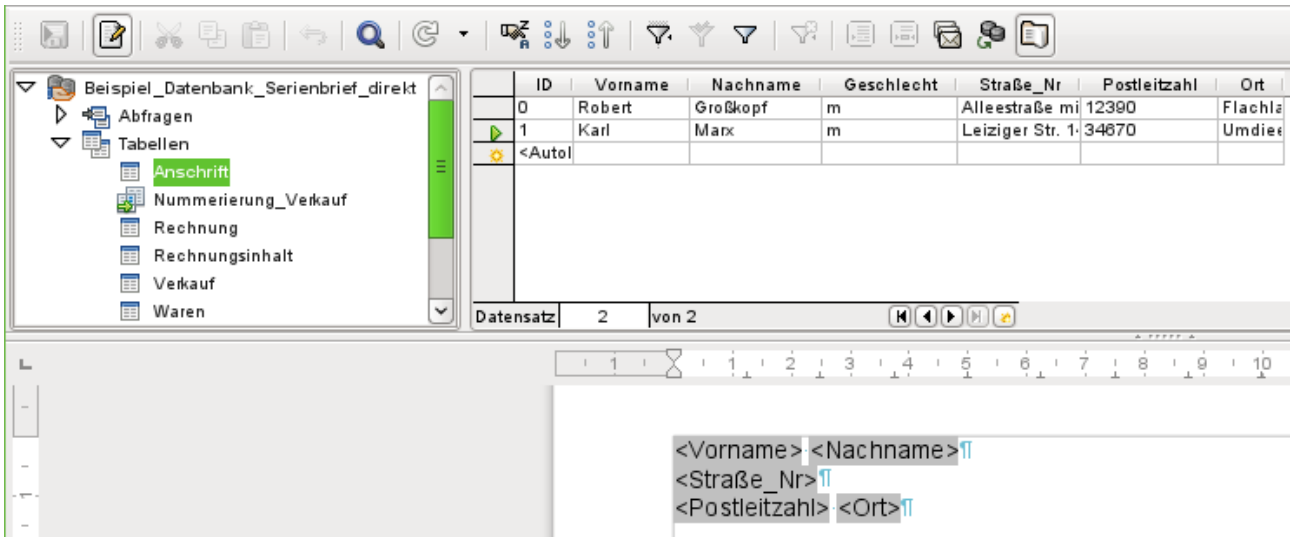
Bevor ein Ausdruck eines Serienbriefes über die Datei «Beispiel_Serienbrief.odt» erfolgen kann, muss die Datenbank in LibreOffice angemeldet werden:

Extras → Optionen → LibreOffice Base → Datenbanken → Neu

Damit das Makro richtig funktioniert, muss für die Datenbank der im Makro vermerkte Name «Beispiel_Datenbank_Serienbrief_direkt» gewählt werden. Datenbank und Textdokument für den Serienbrief müssen außerdem in dem gleichen Verzeichnis liegen.

Wird anschließend 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 Vorlagebrief 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.

Makrosteuerung zum Ausdruck im Serienbrief

```
SUB Serienbrief
    DIM oForm AS OBJECT
    DIM oDB AS OBJECT
    DIM stDir AS STRING
    DIM loFeldID AS LONG
    DIM loID AS LONG
```

Der Datenbankwertebereich für "Integer" entspricht dem StarBasic-Wertebereich für "Long". Deshalb müssen Integer-Felder aus der Datenberg mit einer Long-Variablen ausgelesen werden, wenn wirklich alle Werte erfasst werden sollen.

```
    DIM arProps()
```

Die nötigen Variablen für den Serienbrief werden in dem Array **`arProps()`** gespeichert.

Nach der Deklaration aller Variablen wird aus dem aktuell im Formular angezeigten Datensatz der Primärschlüsselwert ausgelesen. Dadurch wird erreicht, dass tatsächlich nur ein Datensatz anschließend für den Druck weitergegeben wird. Es wird davon ausgegangen, dass das Feld in der Tabelle die Bezeichnung "ID" hat und die Tabelle aktuell im Formular «MainForm» als Datengrundlage dient. Das Feld "ID" muss für dieses Vorgehen nicht im Formular als Formularfeld angezeigt werden. Der Zugriff geht hier direkt über die in der Datengrundlage des Formular vorhandenen Felder.

```
    oForm = thisComponent.Drawpage.Forms.MainForm
    loFeldID = oForm.findColumn("ID")
    loID = oForm.getLong(loFeldID)
```

Die Variable «loID» enthält jetzt den aktuellen Primärschlüsselwert.

```
    MailMerge = CreateUnoService("com.sun.star.text.MailMerge")
    oDB = ThisComponent.Parent
```

Der Zugriff auf die URL ist nicht vom Formular aus direkt möglich. Es muss auf den darüber liegenden Frame der Datenbank Bezug genommen werden, damit der Pfad ermittelt werden kann, in dem die Datenbank zur Zeit liegt.

```
    stDir = Left(oDB.Location,Len(oDB.Location)-Len(oDB.Title))
```

Der Titel der Datenbank wird von der URL abgetrennt. Anschließend werden die verschiedenen Details für den UnoService festgelegt. Dies kann entweder über eine jeweils einzelne Definition wie **`MailMerge.DataSoureName`**, **`MailMerge.DocumentURL`** usw. **oder** über die Anweisung **`WITH MailMerge`** und der Aufzählung der Detailanweisungen erfolgen.

```
    WITH MailMerge
        .DataSourceName = "Beispiel_Datenbank_Serienbrief_direkt"
```

Die Datenbank muss unter diesem Namen eingebunden sein.

```
        .DocumentURL = stDir+"Beispiel_Serienbrief.odt"
```

Die Datei «Beispiel_Serienbrief.odt» liegt im gleichem Pfad wie die Datenbank.

```
        .SaveAsSingleFile = true
```

Aus dem Inhalt sollen Einzeldokumente erstellt werden, nicht mehrere Briefe hintereinander in einem Dokument. **`false`** oder **`0`** bedeutet «nein», **`true`** oder **`1`** bedeutet «ja».

```
        .CommandType = 0
```

Die folgenden Datenquellentypen gibt es: 0 = Tabelle, 1 = Abfrage, 2 = SQL-Code

```
        .Command = "Anschrift"
```

Abhängig von dem Typen wird hier entweder die Bezeichnung der Tabelle/Abfrage oder der komplette SQL-Code angegeben.

```
        .Filter = "ID ="+loID
```

Der Wert des Primärschlüssels wird hier als Filter für den SQL-Code an den Serienbrief übergeben.

```
        .FileNameFromColumn = true
```

Hier wird angegeben, dass der Bezeichner für den Brief aus dem Inhalt eines Tabellenfeldes erstellt werden soll

```
        .FileNamePrefix = "Nachname"
```

Aus dem Inhalt dieses Tabellenfeldes wird der Bezeichner für die Datei übernommen. Dabei wird hinter den Nachnamen eine **`0`** gesetzt, um gegebenenfalls mehrere Serienbriefe an eine Person mit gleichem Namen zu senden. Der folgende Brief erhält dann eine **`1`** usw.

```
        .OutputType = 2
```

Der Serienbrief kann direkt an den Drucker gehen (Drucker = 1), er kann als Datei abgespeichert (Datei = 2) oder als E-Mail versandt werden (Mail = 3).

```
        .OutputUrl = stDir
```

Die zu erstellende Datei wird hier im selben Pfad abgelegt, in dem sich auch die Datenbank befindet. Hier könnte natürlich jeder beliebige Pfad eingegeben werden.

```
    END WITH
    MailMerge.execute(arProps())
END SUB
```

Das Formular «Anschrift_Textfelder»

Vom Aufbau her unterscheidet sich das Formular «Anschrift_Textfelder» nicht vom Formular «Anschrift». Allerdings ist in den Formular-Eigenschaften unter Daten → Art des Dateninhaltes «Abfrage» ausgewählt. Dies ist nur notwendig, da in den Adressen eine Anrede dargestellt werden soll.

Die Abfrage, auf der das Formular beruht, zeigt die gesamte Tabelle "Anschrift" auf:

```
SELECT
    "ID", "Vorname", "Nachname", "Geschlecht" AS "Geschl_kurz",
    "Straße_Nr", "Postleitzahl", "Ort",
    CASEWHEN( "Geschlecht" = 'm', 'Herrn', 'Frau' ) AS "Geschlecht"
    FROM "Anschrift"
```

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

Ausdruck aus dem Formular «Anschrift_Textfelder»

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.

Makrosteuerung zum Ausdruck mit Textfeldern

```
SUB Textfelder_Fuellen
    DIM oForm AS OBJECT
    DIM oColumns AS OBJECT
    DIM oDB AS OBJECT
    DIM oNewDoc AS OBJECT
    DIM oTextfields AS OBJECT
    DIM oTextfield AS OBJECT
    DIM stColumnname AS STRING
    DIM stDir AS STRING
    DIM inIndex AS INTEGER
    oForm = thisComponent.Drawpage.Forms.MainForm
```

Nach der Definition der Variablen wird das Hauptformular angesteuert. Hier könnte auch die Lage des auslösenden Buttons das Formular selbst ermitteln.

```
    oColumns = oForm.Columns
    oDB = ThisComponent.Parent
    stDir = Left(oDB.Location,Len(oDB.Location)-Len(oDB.Title))
```

Der Zugriff auf die URL wird wie im Makro «Serienbrief» geregelt. Die Vorlage «Beispiel_Textfelder.ott» liegt im gleichen Pfad wie die Datenbank.

```
    stDir = stDir & "Beispiel_Textfelder.ott"
```

Die Vorlage wird geladen.

```
DIM args(0) AS NEW com.sun.star.beans.PropertyValue
args(0).Name = "AsTemplate"
args(0).Value = True
oNewDoc = StarDesktop.loadComponentFromURL(stDir,"_blank",0,args)
```

Die Textfelder aus dem Dokument werden ausgelesen. Zu beachten ist, dass hier für die Gesamtzahl der Felder die Variable **`oTextfields`** (mit einem «s»), für das einzelne Textfeld **`oTextfield`** gewählt wurde.

```
oTextfields = oNewDoc.Textfields.createEnumeration
DO WHILE oTextfields.hasMoreElements
   oTextfield = oTextfields.nextElement
   IF oTextfield.supportsService("com.sun.star.text.TextField.JumpEdit") THEN
      stColumnname = oTextfield.PlaceHolder
```

`Placeholder` ist die Benennung für das Textfeld in dem Writer-Dokument.

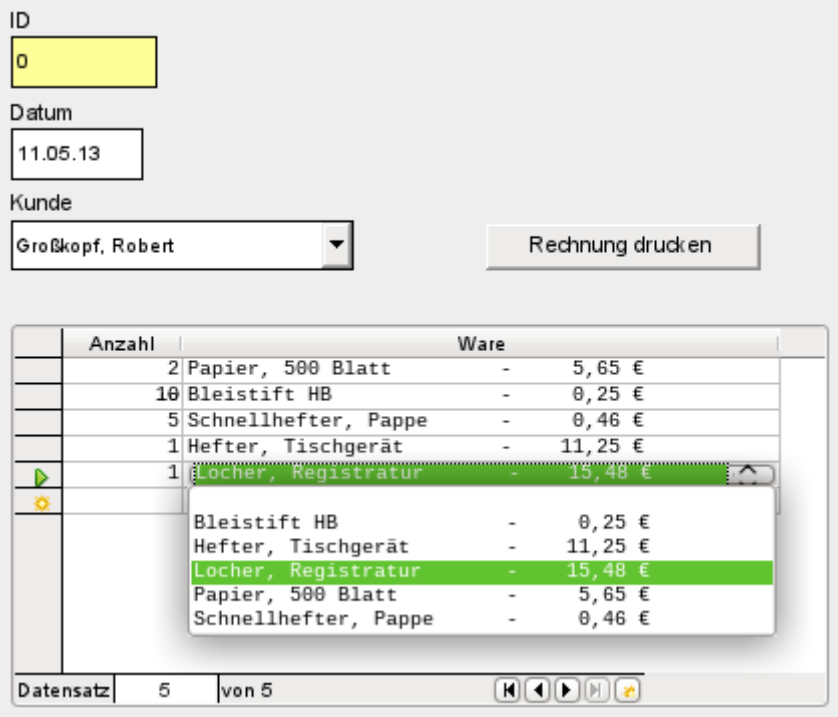
```
         IF oColumns.hasByName(stColumnname) THEN
```

Wenn der Name des Textfeldes gleich dem Spaltennamen der Daten ist, die dem Formular zugrunde liegen, dann wird dem Textfeld der Inhalt dieses Feldes zugewiesen. Da für das Formular eine Abfrage als Datenbasis gewählt wurde, wird jetzt aus dieser z.B. der Wert für das Feld "Geschlecht" ermittelt. Das ist dank der Abfrage aber mit den Inhalten der zum Geschlecht passenden Anrede gefüllt.

```
            inIndex = oForm.findColumn(stColumnname)
            oTextfield.Anchor.String = oForm.getString(inIndex)
         END IF
      END IF
   LOOP
END SUB
```

Das Formular «Rechnung»

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.



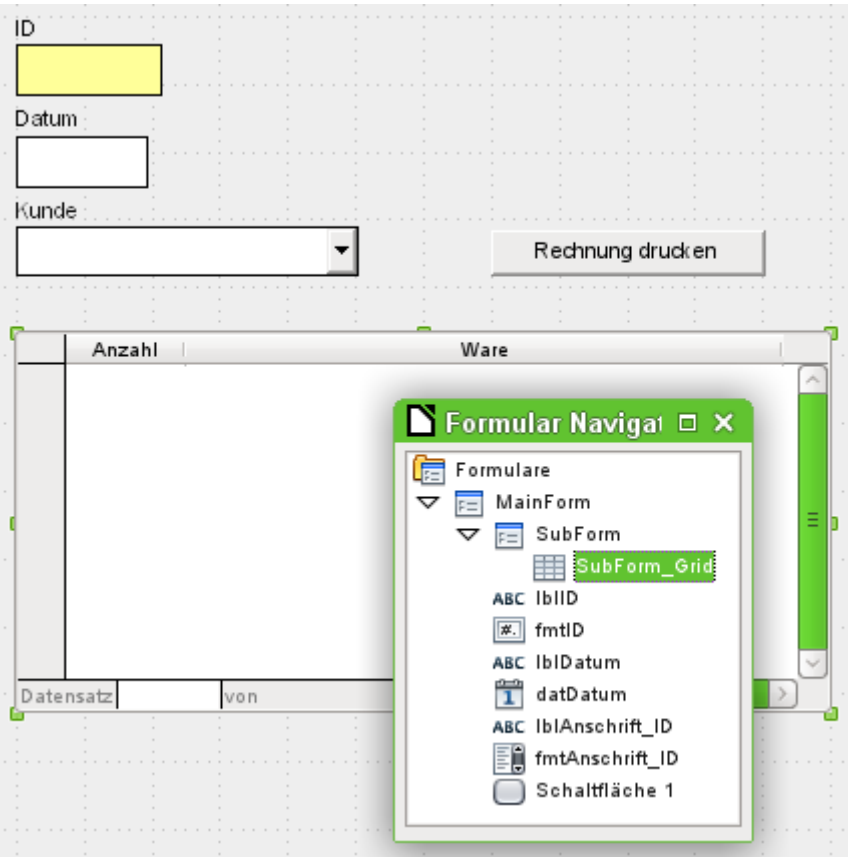
Um ein **Listenfeld** so wie oben erscheinen zu lassen muss zuerst einmal die Schriftart von einer proportionalen Schriftart zu einer Schriftart mit fester Zeichenbreite umgewandelt werden. Das Tabellenkontrollfeld wird markiert. Über das **Kontextmenü → Kontrollfeld → Eigenschaften: Tabellen-Steuerelement → Allgemein → Schrift** wird die Schriftart «Liberation Mono, Standard» ausgesucht.

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

```
SELECT
   LEFT("Ware"|| SPACE(25), 25) || ' - ' ||
      REPLACE( RIGHT( SPACE(8) ||"Preis", 8),'.',',') || ' €',
      "ID"
   FROM "Waren"
   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.

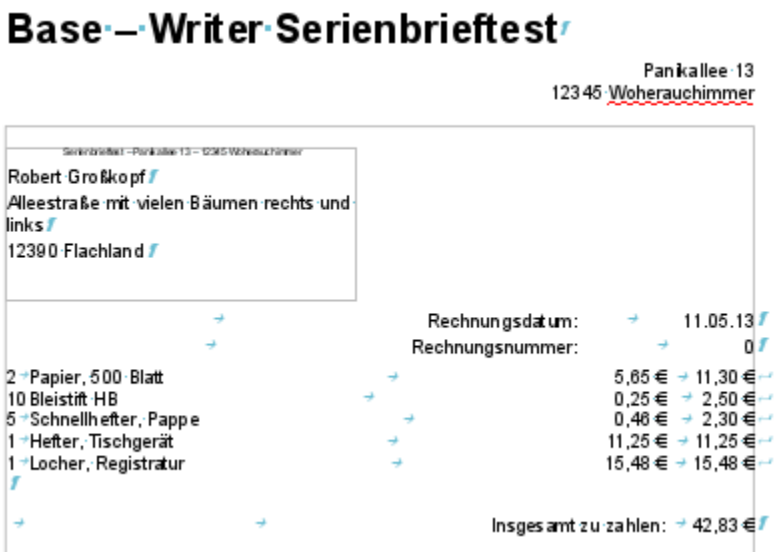
Wie die Anzahl der erforderlichen Leerzeichen auch automatisch ermittelt werden kann ist im Base-Handbuch beschrieben.



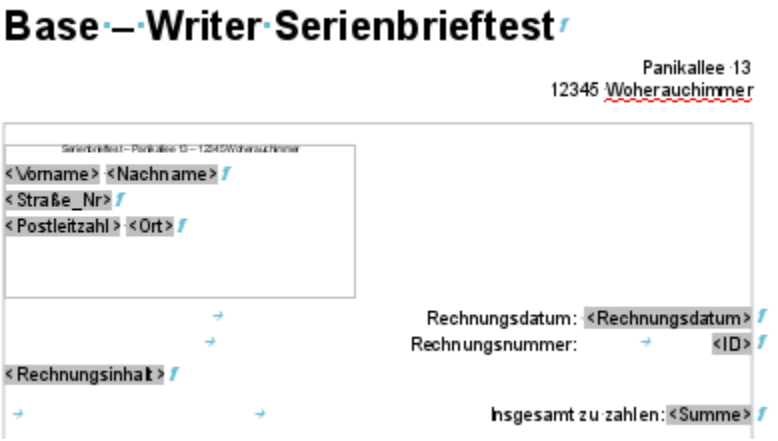
Der Blick auf den Formularnavigator zeigt, dass in diesem Formular neben dem Hauptformular ein Unterformular existiert. Die Datenquelle des Hauptformulars ist die Tabelle "Rechnung", das Unterformular hat als Datenquelle die Tabelle "Verkauf". Die Verbindung erfolgt von "Rechnung"."ID" nach "Verkauf"."Rechnung_ID". So ein Formular mit Unterformular wird auch vom Formularassistenten direkt erstellt, wenn vorher die Tabellenbeziehung über **Extras → Beziehungen** definiert wurde.

Ausdruck aus dem Formular «Rechnung»

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



Durch die seit LO 4.3 farbige Anzeige der Steuerzeichen wird im Ausdruck deutlich, wie eine Formatierung in Tabellenform erreicht wurde. Der Rechnungsinhalt wurde durch Tabulatoren und weiche Umbrüche in das Dokument eingefügt. Die Lage der Tabulatoren im Absatz ist über das Absatzformat der Vorlage definiert. Sie kann natürlich noch angepasst werden, falls die Preise oder auch die Anzahl mehr Platz benötigen. Die Länge des Gesamtinhaltes ist allerdings durch die Zeile begrenzt.



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

Makrosteuerung zum Ausdruck der Rechnung im Serienbrief

Das Makro, das die Inhalte aus der Datenbank an den Serienbrief für die Rechnung weiterleiten soll, unterscheidet sich nur wenig von dem, das genau das gleiche Vorgehen für die Anschrift vorsieht. Im folgenden sind deshalb nur die Unterschiede der entsprechenden Prozedur aufgezeigt.

```
SUB Rechnung
    ...
    Rechnungsinhalt_zusammenstellen(loID)
```

Es wird von der Prozedur «Rechnung» aus eine andere Prozedur «Rechnungsinhalt_zusammenstellen» mit dem Primärschlüsselwert des aktuellen Datensatzes aufgerufen.

```
    WITH MailMerge
        .DataSourceName = "Beispiel_Datenbank_Serienbrief_direkt"
        .DocumentURL =  stDir+"Beispiel_Rechnung.odt"
        .Command = "Rechnung_einzeilig"
        ...
```

Der Aufruf von **`MailMerge`** unterscheidet sich nur in den Variablen, die das zu benutzenden Dokument und die in dem Dokument zu findende Abfrage beschreiben. Die Datenbank selbst ist oben nur der Vollständigkeit halber noch einmal aufgeführt. Die weiteren Variablen sind identisch.

```
    END WITH
    MailMerge.execute(arProps())
END SUB
```

Die HSQLDB bietet keine Funktion wie GroupConcat (MySQL) oder List (Firebird). Inhalte von gleichen Feldern in unterschiedlichen Datensätzen können deshalb am besten mit einem Makro in einer Zeile zusammengeführt werden. Das Ergebnis dieses Makros ist also das Erstellen eines einzigen Datensatzes aus der Abfrage beliebig Datensätze einer Tabelle.

```
SUB Rechnungsinhalt_zusammenstellen(loID AS LONG)
    DIM oDatenquelle AS OBJECT
    DIM oVerbindung AS OBJECT
    DIM oSQL_Anweisung AS OBJECT
    DIM oAbfrageergebnis AS OBJECT
    DIM stSql AS STRING
    DIM stText AS STRING
    stText = ""
    oDatenquelle = ThisComponent.Parent.CurrentController
    If NOT (oDatenquelle.isConnected()) THEN
        oDatenquelle.connect()
    END IF
    oVerbindung = oDatenquelle.ActiveConnection()
    oSQL_Anweisung = oVerbindung.createStatement()
```

Die Variablen werden deklariert. Die Variable **`loID`** wird beim Start der Prozedur direkt mitgegeben. Anschließend wird eine Verbindung zur Datenbank auf SQL-Ebene hergestellt. Der folgende SQL-Befehl ist dabei auf mehrere Zeilen verteilt, hier aber zur etwas besseren Lesbarkeit in eine Zeile zusammengefasst und gegliedert worden. Die Feldnamen der angesprochenen Tabelle müssen dabei in StarBasic mit doppelten Anführungszeichen eingegeben werden. Das doppelte Anführungszeichen bewirkt, dass das erste Anführungszeichen als Maskierung für das folgende sorgt und so ein Anführungszeichen tatsächlich aus dem Code heraus weiter gegeben wird.

```
    stSql = "SELECT ""Verkauf"".""Anzahl"" || CHAR( 9 ) ||
          ""Waren"".""Ware"" || CHAR( 9 ) ||
          REPLACE( ""Waren"".""Preis"" || ' €', '.', ',' ) || CHAR( 9 ) ||
          REPLACE( ""Verkauf"".""Anzahl"" * ""Preis"" || ' €', '.', ',' ) ||
              CHAR( 10 )
       FROM ""Verkauf"", ""Waren""
       WHERE ""Verkauf"".""Waren_ID"" = ""Waren"".""ID""
          AND ""Verkauf"".""Rechnung_ID"" = "+loID+""
```

Das Zeichen **`CHAR(9)`** steht für einen Tabulator. Das Zeichen **`CHAR(10)`** für den weichen Zeilenumbruch. Die Bedeutung der verschiedenen Steuerzeichen kann hier mit entsprechender Zahlenzuordnung nachgelesen werden: *http://de.wikipedia.org/wiki/Steuerzeichen* .

Die verschiedenen Felder werden wieder über **||** miteinander zu einem Feldinhalt verbunden. Bei den Feldern, die einen Preis ergeben sollen, wird der Dezimalpunkt, den sonst die Abfrage ergäbe, durch ein Dezimalkomma ersetzt. Außerdem wird die Währungseinheit «€» hinzugefügt.

Es werden nur die Felder der betroffenen Tabellen ausgewählt, die zu der entsprechenden Rechnungsnummer passen: **""Verkauf"."Rechnung_ID"" = "+loID+""**. Anschließend wird die Abfrage an die Datenbank gestellt.

```
oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql) ' Ergebnis auswerten
WHILE oAbfrageergebnis.next
   stText = stText + oAbfrageergebnis.getString(1)
WEND ' nächster Datensatz
```

Jeder Datensatz des Abfrageergebnis wird ausgelesen und an den vorhergehenden Datensatz angehängt. Das Abfrageergebnis ist ein Text und steht an der ersten Position der Abfrage. Deswegen steht in der Anweisung **getString(1)**. Die Abfrage gibt nich mehr als ein Feld wieder.

```
stSql = "DELETE FROM ""Rechnungsinhalt"" WHERE ""ID"" = "+loID+""
oSQL_Anweisung.executeUpdate(stSql)
```

Die Tabelle "Rechnungsinhalt" wird von allen vorhergehenden Eingaben gelöscht. Anschließend werden die gerade zusammengestellten Daten als eine einzelne Zeile eingefügt.

```
stSql = "INSERT INTO ""Rechnungsinhalt"" (""ID"",""Rechnungsinhalt"") VALUES
   ("+loID+", '"+stText+"')"
oSQL_Anweisung.executeUpdate(stSql)
END SUB
```

Zusammenführen der Druckdaten in einer Abfrage

Nach Durchführung der Makros steckt der Inhalt für die Rechnung jetzt jeweils in einer Zeile der entsprechenden Tabellen. Jetzt müssen für den Serienbrief die Informationen aus den Tabellen zusammen gezogen werden. Außerdem ist noch der Gesamtbetrag der Rechnung durch Addition der Einzelbeträge zu ermitteln. Das alles leistet die Abfrage «Rechnung_einzeilig».

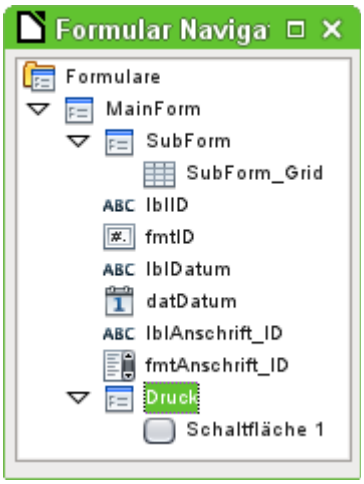
```
SELECT
    "Anschrift"."Vorname",
    "Anschrift"."Nachname",
    "Anschrift"."Straße_Nr",
    "Anschrift"."Postleitzahl",
    "Anschrift"."Ort",
    "a"."Datum" AS "Rechnungsdatum",
    "a"."ID",
    "Rechnungsinhalt"."Rechnungsinhalt",
    REPLACE( (
        SELECT SUM( "Verkauf"."Anzahl" * "Waren"."Preis" )
            FROM "Verkauf", "Waren"
            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.

Diese Abfrage wird mit dem Makro «Rechnung» als Datengrundlage für den Serienbrief «Beispiel_Rechnung.odt» genutzt.

Das Formular «Rechnung_Textfelder»

Das Formular «Rechnung_Textfelder» unterscheidet sich von dem Formular «Rechnung» nur in einem Punkt: Der Button liegt in einem weiteren Unterformular.



Das Unterformular «Druck» ist über das Feld "ID" mit dem Hauptformular verbunden. Art des Inhaltes ist eine Abfrage, und zwar die Abfrage «Rechnung_Textfelder».

Diese Abfrage stellt alle Einzeldaten der Rechnung zusammen, nicht aber den Rechnungsinhalt. Während die Einzeldaten anschließend in Textfelder übertragen werden, wird der Rechnungsinhalt in eine Writer-Tabelle eingetragen.

```
SELECT
    "Anschrift"."Vorname",
    "Anschrift"."Nachname",
    "Anschrift"."Straße_Nr",
    "Anschrift"."Postleitzahl",
    "Anschrift"."Ort",
    RIGHT( '0' || DAY( "a"."Datum" ), 2 ) || '.' || RIGHT( '0' || MONTH( "a"."Datum" ), 2 ) || '.' ||
        YEAR( "a"."Datum" ) AS "Rechnungsdatum",
    "a"."ID",
    REPLACE( (
        SELECT SUM( "Verkauf"."Anzahl" * "Waren"."Preis" )
            FROM "Verkauf", "Waren"
            WHERE "Verkauf"."Waren_ID" = "Waren"."ID"
                AND "Verkauf"."Rechnung_ID" = "a"."ID"
        ) || ' €', '.', ',' ) AS "Summe"
    FROM "Rechnung" AS "a", "Anschrift"
    WHERE "a"."Anschrift_ID" = "Anschrift"."ID"
```

Zuerst werden die Anschrifts-Daten erfasst.

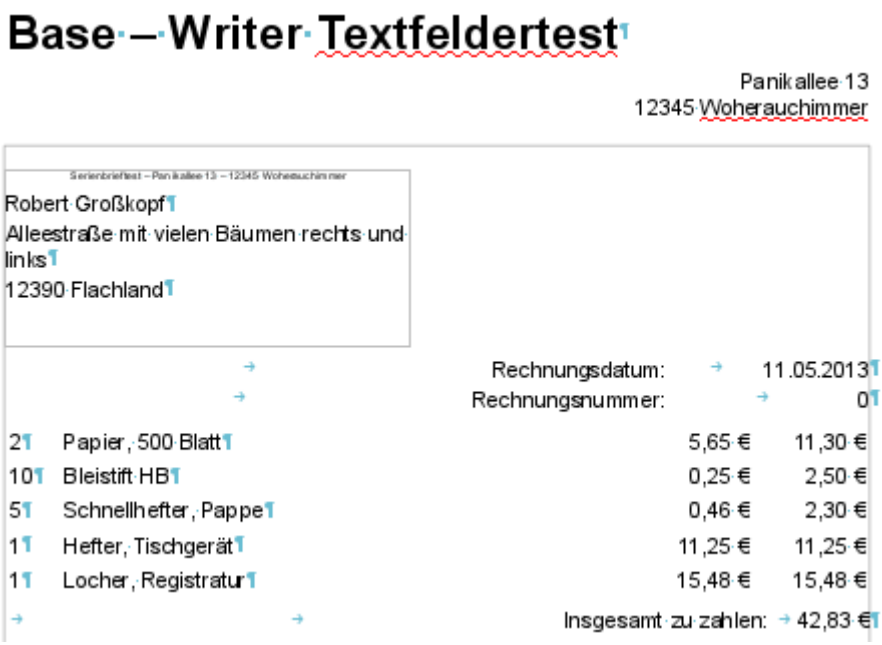
Beim Rechnungsdatum ist darauf zu achten, dass es, sofern in ein Textfeld eingelesen wird, in der korrekten Formatierung übergeben wird. Wird es nicht entsprechend vorformatiert, so erscheint es in der Schreibweise, die bei Datenbanken Standard ist: 2014-05-07. Mit **`DAY( "a"."Datum" )`** wird aus der 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' || DAY( "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.

Der weitere Inhalt der Abfrage wurde bereits für das vorhergehende Formular erläutert. Die Summe wird ermittelt, der Dezimalpunkt durch ein Komma ersetzt und schließlich noch ein «€»-Zeichen angefügt.

Ausdruck aus dem Formular «Rechnung_Textfelder»

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



Base – Writer Textfeldertest

Panikallee 13
12345 Woherauchimmer

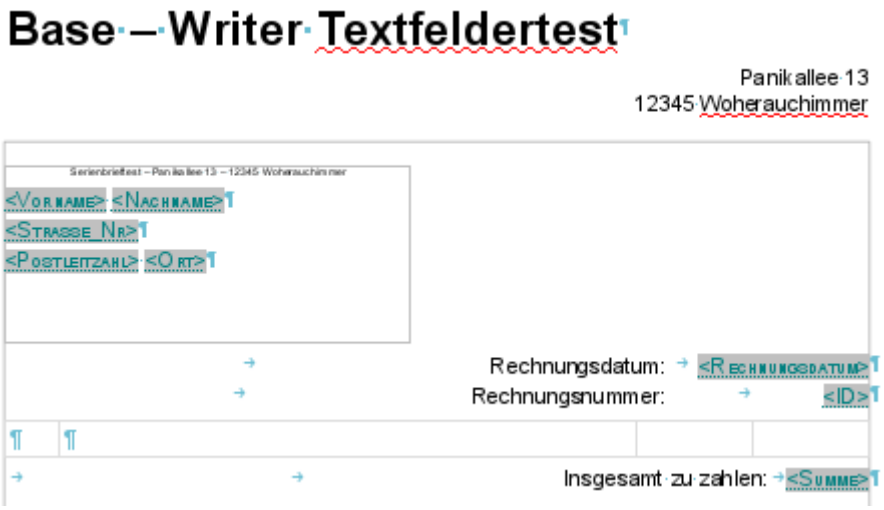
Robert Großkopf
Alleestraße mit vielen Bäumen rechts und links
12390 Flachland

Rechnungsdatum: 11.05.2013
Rechnungsnummer: 0

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

Insgesamt zu zahlen: 42,83 €

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.



Base – Writer Textfeldertest

Panikallee 13
12345 Woherauchimmer

<VORNAME> <NACHNAME>
<STRASSE_NR>
<POSTLEITZAHL> <ORT>

Rechnungsdatum: <RECHNUNGSDATUM>
Rechnungsnummer: <ID>

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 keine sichtbare Absatzendmarke. Die Tabelle ist hier nur zum besseren Verständnis 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.

Makrosteuerung zum Ausdruck der Rechnung mit Textfeldern

```
SUB Rechnungsinhalt_zusammenstellen_Tabelle
    DIM oDatenquelle AS OBJECT
    DIM oVerbindung AS OBJECT
    DIM oSQL_Anweisung AS OBJECT
    DIM oAbfrageergebnis AS OBJECT
    DIM oForm AS OBJECT
    DIM oColumns AS OBJECT
    DIM oDB AS OBJECT
    DIM oNewDoc AS OBJECT
    DIM oTabelle AS OBJECT
    DIM oRows AS OBJECT
    DIM oTextfields AS OBJECT
    DIM oTextfield AS OBJECT
    DIM stColumnname AS STRING
    DIM stDir AS STRING
    DIM stSql AS STRING
    DIM stText AS STRING
    DIM inIndex AS INTEGER
    DIM i AS INTEGER
    DIM loID AS LONG
    DIM doPreis AS DOUBLE
    DIM doAnzahlPreis AS DOUBLE
    oForm = thisComponent.Drawpage.Forms.MainForm.getByName("Druck")
    oForm.reload()
    oForm.last()
```

Das Formular, in dem sich auch der Button befindet, wird angesprochen. Der Inhalt der Abfrage, die diesem Formular zugrunde liegt, soll an die Textfelder weitergegeben werden. Damit sicher alle Datensätze angezeigt werden, wird die Abfrage noch einmal mit **`reload()`** neu geladen.

Der Datensatzzeiger muss hier wegen eines Bugs in der 4.3.0.0 Beta1 auf den letzten Datensatz gesetzt werden, da beim **`reload()`** der Zeiger einen Datensatz weiter bewegt wird.

Die Befüllung der Textfelder ist gleich der in der Prozedur «Textfelder_Fuellen». Lediglich die Vorlage hat hier einen anderen Namen, nämlich «Beispiel_Rechnung_Textfelder.ott».

Nach dem Befüllen der Textfelder beginnt der Teil, der für den Druck der Rechnungsinhalte einen Gewinn bringt. Hier wird die Tabelle befüllt und mit neuen Zeilen versehen.

```
    oTabellen = oNewDoc.getTextTables
    oTabelle = oTabellen.getByName("Rechnungsinhalt")
```

Das aus der Vorlage geöffnete Dokument wird angesteuert. Die Tabelle ist in den Tabelleneigenschaften mit dem Namen «Rechnungsinhalt» versehen worden. So kann genau diese Tabelle aus dem Dokument herausgesucht werden.

```
    oDatenquelle = ThisComponent.Parent.CurrentController
    If NOT (oDatenquelle.isConnected()) THEN
        oDatenquelle.connect()
    END IF
    oVerbindung = oDatenquelle.ActiveConnection()
```

Die Verbindung zur Datenbank wird erstellt. Anschließend wird über diese Verbindung ein SQL-Kommando abgesetzt, mit dem der Inhalt für die Tabelle «Rechnungsinhalt» zeilenweise gefüllt werden soll.

```
    oSQL_Anweisung = oVerbindung.createStatement()
    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+""
```

Aus den Tabellen "Verkauf" und "Waren" werden die für die jeweilige Rechnung notwendigen Informationen zusammengestellt. In der Tabelle "Verkauf" ist die Anzahl der einzelnen Artikel aufgelistet, in der Tabelle "Waren" die Informationen über die "Ware" und den "Preis". Schließlich wird noch die "Anzahl" mit dem "Preis" multipliziert.

```
    oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
    i = 0
    WHILE oAbfrageergebnis.next
        loAnzahl = oAbfrageergebnis.getInt(1)
        stText = oAbfrageergebnis.getString(2)
        doPreis = oAbfrageergebnis.getDouble(3)
        doAnzahlPreis = oAbfrageergebnis.getDouble(4)
```

Die Daten sind in dem Objekt «oAbfrageergebnis» gespeichert. Sie werden Zeile für Zeile ausgelesen. Die Ziffern hinter **`getInt()`**, **`getString()`** usw. verweisen auf die jeweiligen Spalten der Abfrage.

```
        oTabelle.getCellByPosition(0,i).setValue(loAnzahl)
        oTabelle.getCellByPosition(1,i).setString(stText)
        oTabelle.getCellByPosition(2,i).setValue(doPreis)
        oTabelle.getCellByPosition(3,i).setValue(doAnzahlPreis)
```

Mit **`i`** wird die Zeile der Tabelle festgelegt. Die Zählung beginnt hier bei 0. Deswegen ist **`i`** vorher auch auf 0 festgesetzt worden. Der erste Zahlenwert bestimmt die Spalte in der Tabelle. Auf diese Art werden sämtliche am Anfang existierenden Zellen mit Werten (für Zahlen) oder mit Text gefüllt.

```
        oRows = oTabelle.getrows()
        oRows.insertByIndex(oRows.getCount(),1)
```

Die Anzahl der Zeilen der Tabelle wird ermittelt und anschließend eine neue Tabellenzeile hinzugefügt, die anschließend wieder befüllt werden kann.

```
        i = i + 1
    WEND
```

Um den nächsten Datensatz in die neue Zeile zu schreiben muss zu **`i`** der Wert 1 addiert werden. Anschließend wird der nächste Datensatz ausgelesen.

```
    oRows.removeByIndex(oRows.getCount()-1,1)
END SUB
```

Da das Makro in der Schleife immer schon eine zusätzliche Zeile anlegt muss beim letzten Schleifendurchgang eine Tabellenzeile zu viel angelegt worden sein. Diese Zeile wird anschließend wieder entfernt. Danach ist die Rechnung fertig erstellt.

Das Formular «Rechnung_Textfelder_Uebertrag»

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

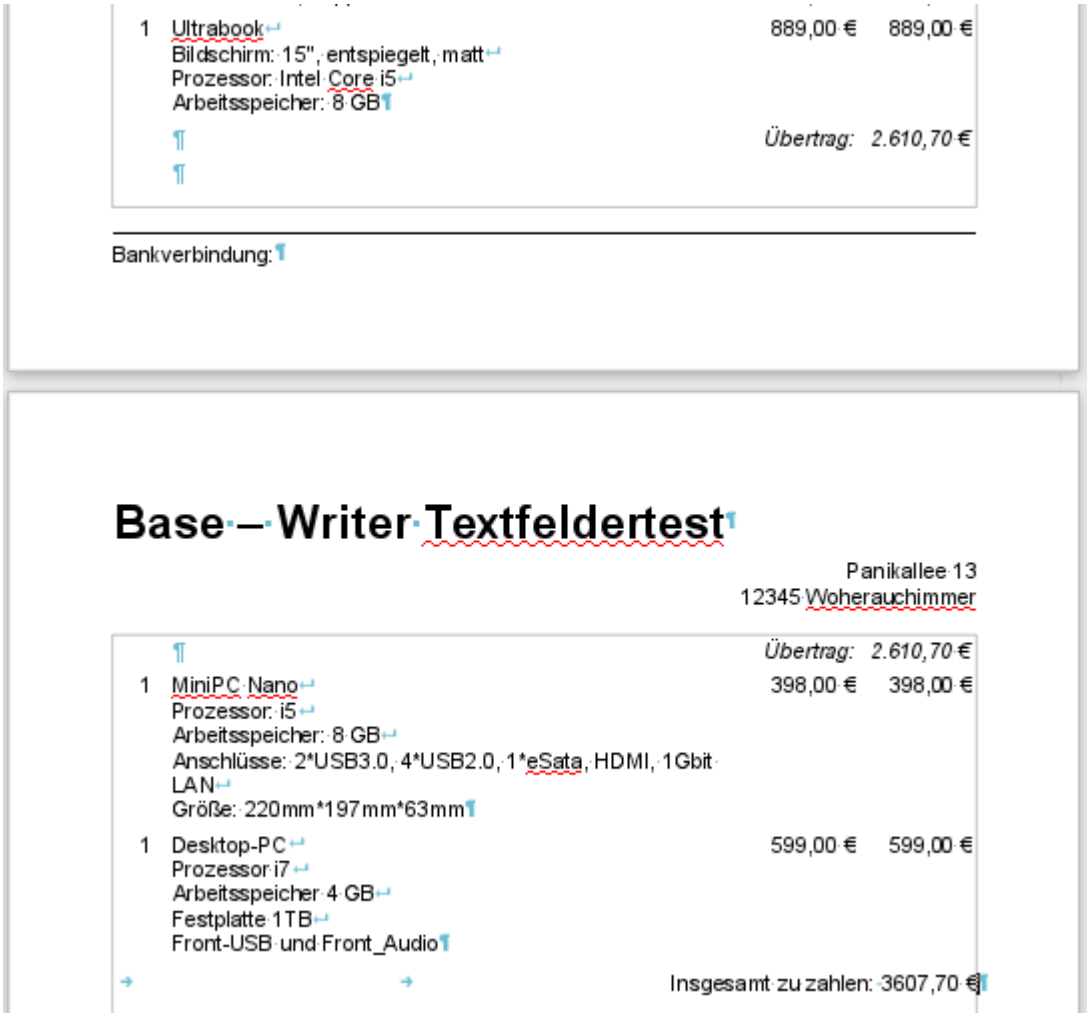
Für die mehrzeiligen Warenangaben war außerdem eine kleine Änderung im Listenfeld des Formulars notwendig. Der zu dem Tabellenkopf «Ware» gehörige **Listeninhalt** wurde etwas erweitert:

```
SELECT
    LEFT(REPLACE("Ware",CHAR(10),' ')|| SPACE(25), 25) || ' - ' ||
        REPLACE( RIGHT( SPACE(8) ||"Preis", 8),'.',',') || ' €',
        "ID"
    FROM "Waren"
    ORDER BY ("Ware" || ' - ' || "Preis" || ' €') ASC
```

In dem Feld "Ware" sind mehrzeilige Eingaben enthalten. Der Umbruch ist ein nicht sichtbares Zeichen. Das führt dazu, dass tatsächlich nur 24 und nicht 25 Zeichen angezeigt werden, wenn der anzuzeigende Inhalt z.B. einen Umbruch hat. Das nicht sichtbare Zeichen, hier **`CHAR(10)`**, muss also entfernt werden. In dem obigen SQL-Code wird es durch ein Leerzeichen ersetzt.

Ausdruck aus dem Formular «Rechnung_Textfelder_Uebertrag»

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.



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 Platze noch eine leere Tabellenzeile folgt.

Makrosteuerung zum Ausdruck der Rechnung mit Übertrag

Als zusätzliche Information bei einer Rechnung über mehrere Seiten wird jetzt noch ermöglicht, einen Übertrag in die unterste Zeile der aktuellen Seite und in die oberste Zeile der nächstfolgenden Zeile zu schreiben.

Der Code des Makros «Rechnungsinhalt_zusammenstellen_Tabelle» wurde dazu entsprechend erweitert. Hier werden nur die Änderungen aufgezeigt.

```
SUB Rechnungsinhalt_zusammenstellen_Tabelle_Uebertrag
    ...
    DIM oCursor AS OBJECT
    DIM oTxtCursor AS OBJECT
    DIM inStartSeite AS INTEGER
    DIM inFolgezeilen AS INTEGER
    DIM doUebertrag AS DOUBLE
    ...
```

```
doUebertrag = 0
inStartSeite = 1
inFolgezeilen = 1
oCursor = oNewDoc.CurrentController.ViewCursor
```

Die Variablen werden mit Anfangswerten belegt. Der Übertrag ist hat am Anfang den Wert 0, die erste Seite den Wert 1. In der Vorlage liegt eine Zeile unter der Tabelle, nämlich die Zeile mit dem Text «Insgesamt zu zahlen: ». Mit Hilfe des sichtbaren Cursors (**`ViewCursor`**) wird die Position innerhalb des Dokumentes bestimmt.

```
WHILE oAbfrageergebnis.next
    ...
    oCursor.JumpToLastPage()
    oCursor.JumpToEndOfPage()
    doUebertrag = doUebertrag + doAnzahlPreis
```

Der Cursor wird auf die letzte Seite und dort an das Ende bewegt. Der Übertrag wird laufend berechnet.

```
IF oCursor.Page > inStartSeite THEN
    oCursor.goUp(inFolgezeilen,False)
```

Der Cursor wird vom Ende um die Zeilen zurückbewegt, die außerhalb der Tabelle noch mit im Dokument enthalten sind. Befindet sich der Cursor dann wieder auf der ersten Seite, so liegt er genau in einer leeren Tabellenzeile, die mit dem Übertrag gefüllt werden kann.

Zusätzlich wird auch noch ein Textcursor gebildet, mit dem der gerade erstelle Eintrag markiert wird und auf den Schriftschnitt «Italic» (kursiv) gesetzt wird. Dadurch ist der Übertrag besser von den anderen Einträgen unterscheidbar.

Anschließend wird noch eine Zeile eingefügt. Diese Zeile liegt jetzt auf der Folgeseite und kann dort mit dem Übertrag für den Seitenanfang gefüllt werden.

```
IF oCursor.Page = inStartSeite THEN
    FOR ink = 1 TO 2
        i = i + 1
        oTabelle.getCellByPosition(2,i).setString("Übertrag:")
        oTxtCursor = oTabelle.getCellByPosition(2,i).createTextCursorByRange
            (oTabelle.getCellByPosition(2,i).text)
        oTxtCursor.CharPosture = com.sun.star.awt.FontSlant.ITALIC
        oTabelle.getCellByPosition(3,i).setValue(doUebertrag)
        oTxtCursor = oTabelle.getCellByPosition(3,i).createTextCursorByRange
            (oTabelle.getCellByPosition(3,i).text)
        oTxtCursor.CharPosture = com.sun.star.awt.FontSlant.ITALIC
        oRows = oTabelle.getRows()
        oRows.insertByIndex(oRows.getCount(),1)
    NEXT ink
ELSE
```

Befindet sich die letzte neu eingefügte Tabellenzeile allerdings nicht auf der vorhergehenden Seite, weil eben mehrzeilige Tabelleneinträge im Bereich «Ware» den Umbruch stören, so sind die Zeilen für den Übertrag vor die entsprechende Tabellenzeile der folgenden Seite zu legen.

```
    i = i + 1
    oRows = oTabelle.getRows()
    oRows.insertByIndex(oRows.getCount()-2,1)
    oTabelle.getCellByPosition(2,i-1).setString("Übertrag:")
    oTxtCursor = oTabelle.getCellByPosition(2,i-1).createTextCursorByRange
        (oTabelle.getCellByPosition(2,i-1).text)
    oTxtCursor.CharPosture = com.sun.star.awt.FontSlant.ITALIC
    oTabelle.getCellByPosition(3,i-1).setValue(doUebertrag - doAnzahlPreis)
    oTxtCursor = oTabelle.getCellByPosition(3,i-1).createTextCursorByRange
        (oTabelle.getCellByPosition(3,i-1).text)
    oTxtCursor.CharPosture = com.sun.star.awt.FontSlant.ITALIC
    WHILE oCursor.Page = inStartSeite + 1
```

Nachdem die erste Tabellenzeile eingefügt wurde kann es sein, dass die folgende Zeile noch nicht auf der folgenden Seite erscheint. Dies hängt davon ab, wie viele Zeilen die Ware eingenommen

hätte, für die eben auf der Vorseite nicht genug Platz war. In dieser Schleife sollen so viele Tabellenzeilen eingefügt werden, bis die nächste Zeile wirklich auf der Folgeseite liegt.

```
                oRows = oTabelle.getRows()
                oRows.insertByIndex(oRows.getCount()-2,1)
                oCursor.JumpToLastPage()
                oCursor.JumpToEndOfPage()
                oCursor.goUp(inFolgezeilen + 2,False)
                i = i + 1
            WEND
```

Nach dieser Schleife liegt die Tabellenzeile auf jeden Fall auf der Folgeseite. In Diese Tabellenzeile kann nun der Übertrag mit den entsprechenden Formatierungen eingetragen werden.

```
            i = i + 1
            oRows = oTabelle.getRows()
            oRows.insertByIndex(oRows.getCount()-2,1)
            oTabelle.getCellByPosition(2,i-1).setString("Übertrag:")
            oTxtCursor = oTabelle.getCellByPosition(2,i-1).createTextCursorByRange
                (oTabelle.getCellByPosition(2,i-1).text)
            oTxtCursor.CharPosture = com.sun.star.awt.FontSlant.ITALIC
            oTabelle.getCellByPosition(3,i-1).setValue(doUebertrag - doAnzahlPreis)
            oTxtCursor = oTabelle.getCellByPosition(3,i-1).createTextCursorByRange
                (oTabelle.getCellByPosition(3,i-1).text)
            oTxtCursor.CharPosture = com.sun.star.awt.FontSlant.ITALIC
        END IF
```

Schließlich wird der Wert für die Startseite um 1 erhöht, damit die Prozedur erst bei einem erneuten Seitenumbruch den nächsten Übertrag generiert.

```
            inStartSeite = inStartSeite + 1
        END IF
        i = i + 1
    WEND
    oRows.removeByIndex(oRows.getCount()-1,1)
END SUB
```

Zum Schluss wird wieder die letzte Zeile der Tabelle, die leer geblieben ist, gelöscht.

Das Formular «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.

Bilder in Base einbinden

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

Tabellen

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

In der Tabelle "Bilder" werden folgende Felder definiert:

Feldname	***Feldtyp***	***Beschreibung***
ID	Integer	Die ID ist Primärschlüssel dieser Tabelle.

Beschreibung	Text	Soll einen Text aufnehmen, der als Bildbeschreibung dienen kann
Pfad	Text	Nimmt den Pfad zu dem Bild auf.

In der Tabelle "Bilder_intern" werden folgende Felder definiert:

Feldname	***Feldtyp***	***Beschreibung***
ID	Integer	Die ID ist Primärschlüssel dieser Tabelle.
Beschreibung	Text	Soll einen Text aufnehmen, der als Bildbeschreibung dienen kann
Bild	Bild	Nimmt das Bild als Binärdatenstrom auf.
BildName	Text	Sollte gegebenenfalls den Namen der Datei mit Dateiendung speichern.

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

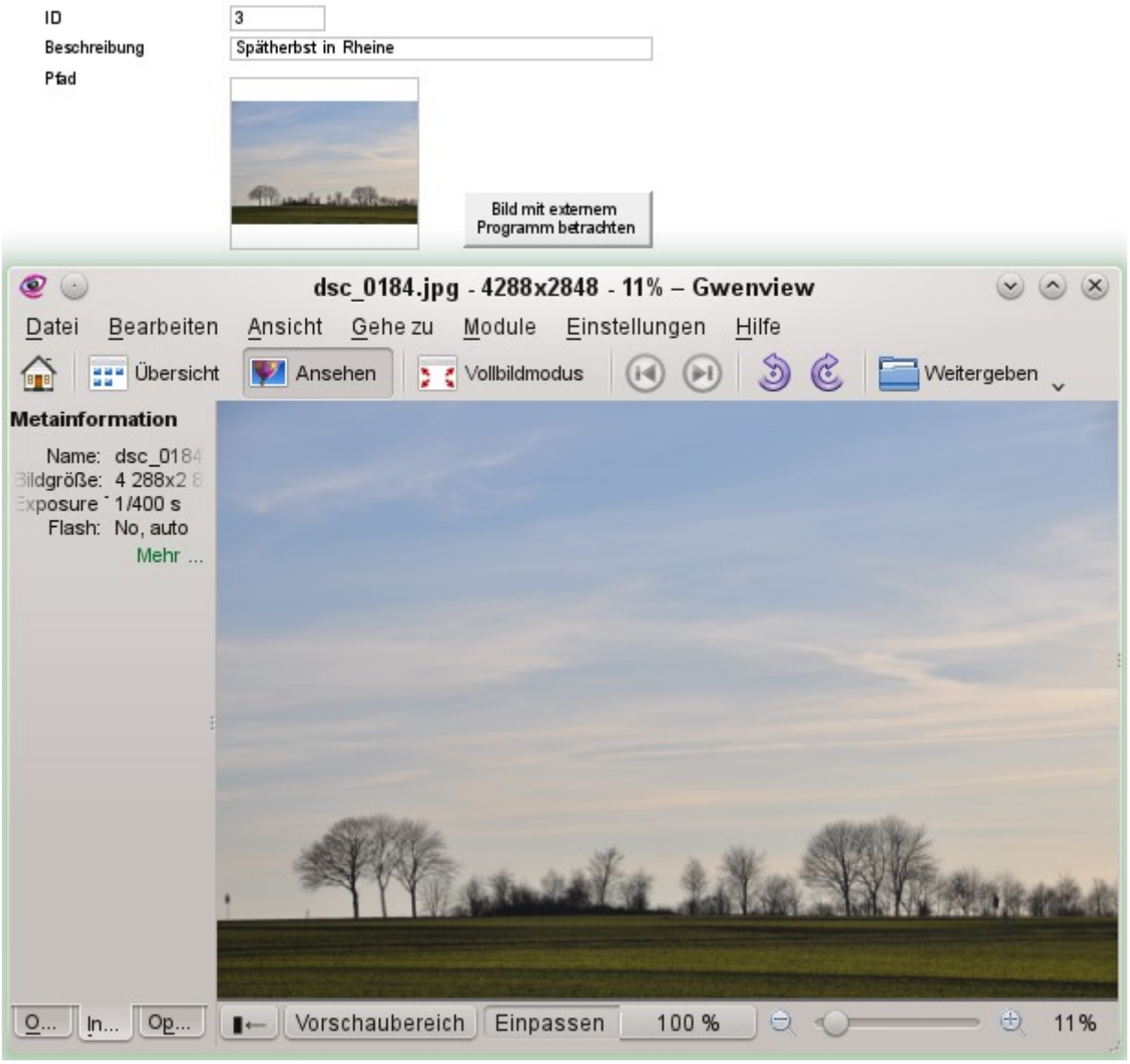
Formulare

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.

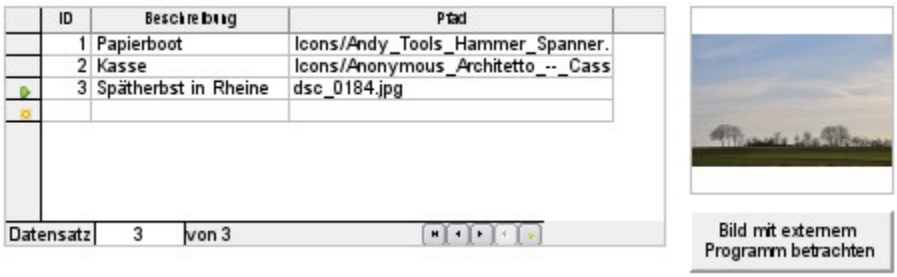
Das Formular «Pfadeingabe»

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 ein Bild aus dem Dateisystem ausgesucht, so wird nur der Pfad (relativ zur Datenbankdatei) gespeichert. Das Bild wird trotzdem in dem grafischen Steuerelement angezeigt.



Das grafische Steuerelement bietet nicht die Möglichkeit, Details des Bildes zu betrachten. Deshalb wird hier über Bild mit externem Programm betrachten der im System angegebenen Dateibetrachter für die entsprechende Datei gestartet. Jetzt können Details betrachtet und alle weiteren Funktionen des Viewers genutzt werden.

Das Formular «Pfadeingabe_Tabellenkontrollfeld»



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 Tabellen-

kontrollfeld im selben Formular angeordnet. Es zeigt das Bild zu dem aktuellen markierten Datensatz an.

Makrosteuerung für die Formulare mit Pfadspeicherung

```
SUB Betrachten
    DIM oDoc AS OBJECT
    DIM oDrawpage AS OBJECT
    DIM oForm AS OBJECT
    DIM oFeld AS OBJECT
    DIM oShell AS OBJECT
    DIM stUrl AS STRING
    DIM stFeld AS STRING
    DIM arUrl_Start()
    oDoc = thisComponent
    oDrawpage = oDoc.Drawpage
    oForm = oDrawpage.Forms.getByName("Formular")
    oFeld = oForm.getByName("GraphischesFeld")
    stUrl = oFeld.BoundField.getString
```

Das graphische Kontrollfeld im Formular wird aufgesucht. Da in der Tabelle nicht das Bild selbst, sondern nur der Pfad als Text gespeichert wird, wird hier über **`getString`** dieser Text ausgelesen.

Anschließend wird der Pfad zu der Datenbankdatei ermittelt. Mit **`oDoc.Parent`** wird die *.odb-Datei erreicht. Sie ist der Container für die Formulare. Über **`oDoc.Parent.Url`** wird schließlich die gesamte URL incl. Dateinamen ausgelesen. Der Dateiname ist auch zu sehen in **`oDoc.Parent.Title`**. Der Text wird jetzt mit der Funktion **`split`** aufgetrennt, wobei als Trenner der Dateiname benutzt wird. Die Auftrennung gibt so nur als erstes und einziges Element des Arrays den Pfad zur *.odb-Datei wieder.

```
    arUrl_Start = split(oDoc.Parent.Url,oDoc.Parent.Title)
    oShell = createUnoService("com.sun.star.system.SystemShellExecute")
    stFeld = convertToUrl(arUrl_Start(0) + stUrl)
    oShell.execute(stFeld,,0)
END SUB
```

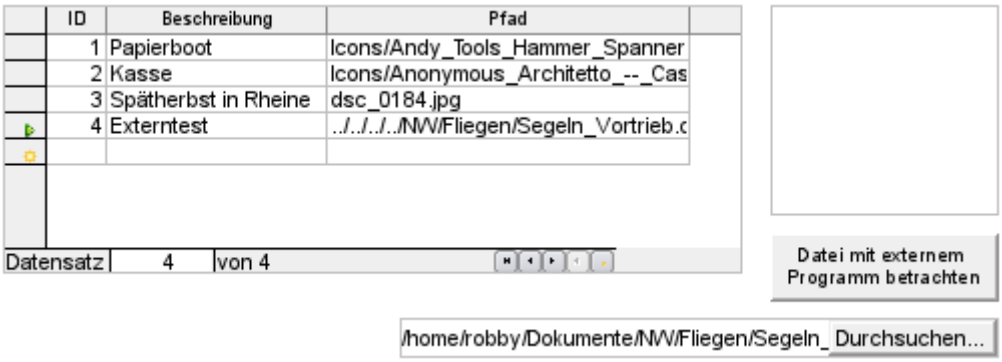
Externe Programme können über das Struct **`com.sun.star.system.SystemShellExecute`** gestartet werden. Dem externen Programm wird hier nur der Pfad zur Datei mitgegeben, der aus dem Pfad zur Datenbankdatei und dem intern gespeicherten relativen Pfad von der Datenbankdatei aus zusammengesetzt wurde. Die grafische Benutzeroberfläche des Betriebssystems entscheidet jetzt darüber, mit welchem Programm die entsprechende Datei zu öffnen ist.

Mit dem Kommando **`oShell.execute`** werden 3 Parameter übergeben. Als erstes wird eine ausführbare Datei oder der Pfad zu einer Datei aufgeführt, die im System mit einem Programm verbunden sind. Als zweites werden Parameter aufgeführt, mit denen das Programm gestartet werden soll. Als drittes wird über eine Ziffer mitgeteilt, wie mit Fehlermeldungen des Systems bei missglückter Ausführung umzugehen ist. Hier stehen 0 (Standard), 1 (keine Meldung anzeigen) und 2 (nur das Öffnen von absoluten URLs erlauben) zur Verfügung.

Das Formular «Pfadeingabe_Tabellenkontrollfeld_Pfadangabe»

Statt eines Bildes kann genauso gut der Pfad für eine Datei gespeichert werden. Wird anschließend der Button Datei mit externem Programm betrachten betätigt, so wird stattdessen das zu der Datei passende Programm des Betriebssystems ausgewählt.

Um den Pfad einer Datei aufzunehmen ist allerdings nicht das grafische Kontrollfeld geeignet. Hier wird mit dem Feld zur Dateiauswahl gearbeitet. Der Pfad zur ausgewählten Datei muss dann aber relativ zur Datenbank in dem entsprechenden Feld der Tabelle gespeichert werden.



Ergänzend zu dem vorhergehenden Formular ist hier ein Dateiauswahlfeld integriert. Dateiauswahlfelder zeigen die gerade erfolgte Auswahl an, sind aber nicht mit der Datenbank verbunden. Die Url des Dateiauswahlfeldes muss über ein Makro in einen relativen Link umgewandelt werden, damit die Datei passend über die Prozedur «Betrachten» geöffnet werden kann. Der Pfad in Zeile 4 zeigt an, dass von der Lage der Datenbankdatei aus 4 Verzeichnisse aufwärts gegangen werden muss und von dort aus das Verzeichnis «NW/Fliegen/» aufzusuchen ist.

Makrosteuerung für die relative Pfadangabe

Das folgende Makro ist an die Eigenschaft «Text modifiziert» des Dateiauswahlfeldes gebunden.

```
SUB PfadAuslesen
    DIM oDoc AS OBJECT
    DIM oDrawpage AS OBJECT
    DIM oForm AS OBJECT
    DIM oFeld AS OBJECT
    DIM oFeld2 AS OBJECT
    DIM arUrl_Start()
    DIM ar()
    DIM ar1()
    DIM ar2()
    DIM stText AS STRING
    DIM stUrl_gesamt AS STRING
    DIM stUrl_Text AS STRING
    DIM stUrl AS STRING
    DIM stUrl_weg AS STRING
    DIM ink AS INTEGER
    DIM i AS INTEGER
    oDoc = thisComponent
    oDrawpage = oDoc.Drawpage
    oForm = oDrawpage.Forms.getByName("Formular")
    oFeld = oForm.getByName("GraphischesFeld")
    oFeld2 = oForm.getByName("Dateiauswahl")
```

Zuerst werden, wie bei allen Prozeduren, die Variablen deklariert. Anschließend werden die entsprechenden Felder aufgesucht, die für die Aufnahme des Pfades wichtig sind. Der gesamte anschließende Code wird nur dann ausgeführt, wenn in der Dateiauswahl auch Inhalt steht, das Feld also z.B. nach einem Datensatzwechsel nicht einfach geleert wird.

```
    IF oFeld2.Text <> "" THEN
        arUrl_Start = split(oDoc.Parent.Url,oDoc.Parent.Title)
        ar = split(oFeld2.Text,"/")
        stText = ""
```

Der Pfad zur Datenbankdatei wird ausgelesen. Dies erfolgt, wie vorher schon erklärt, indem zuerst die gesamte Url gelesen wird und anschließend der Dateiname als Trenner eines Arrays dafür sorgt, dass im ersten teil des Arrays nur der direkte Pfad steht.

Anschließend werden alle Elemente des Pfades, der über die Dateiauswahl ausgesucht wurde, in das Array **`ar`** eingelesen. Als Trenner dient **`/`**. Dies dient dazu, für die Ermittlung des Pfades zu der Datei einfach nur den Dateinamen hinten abzuschneiden. Daher wird im nächsten Schritt der Pfad zu der aufzurufenden Datei in der Variablen **`stText`** wieder zusammengesetzt. Die Schleife endet

allerdings nicht mit dem höchsten Wert des vorher erstellten Arrays **ar**, sondern schon beim erreichen der vorhergehenden Wertes.

```
        FOR i = LBound(ar()) TO UBound(ar()) - 1
            stText = stText & ar(i) & "/"
        NEXT
        stText = Left(stText,Len(stText)-1)
        arUrl_Start(0) = Left(arUrl_Start(0),Len(arUrl_Start(0))-1)
```

Das zuletzt angehängte **/** wird wieder entfernt, da sonst im folgenden Array zum Schluss ein leerer Arraywert erzeugt wird, der den Pfadvergleich stören würde. Für einen richtigen Vergleich ist jetzt der Text noch zu einer tatsächlichen Url, beginnend mit **file:///**, umzuwandeln. Schließlich soll mit dem Pfad der Datenbankdatei verglichen werden, der genau auf dieser Basis zusammengesetzt ist.

```
        stUrl_Text = ConvertToUrl(stText)
        ar1 = split(stUrl_Text,"/")
        ar2 = split(arUrl_Start(0),"/")
        stUrl = ""
        ink = 0
        stUrl_weg = ""
```

In **ar1** werden alle Elemente gespeichert, die den Pfad zu der aufzurufenden Datei wieder geben. In **ar2** sind alle Elemente gespeichert, die den Pfad zur Datenbankdatei wieder geben. Anschließend wird das gesamte Array **ar2** schrittweise für einen Vergleich in einer Schleife durchlaufen.

```
        FOR i = LBound(ar2()) TO UBound(ar2())
            IF i <= UBound(ar1()) THEN
```

Nur wenn die Zahl **i** nicht größer ist als die Anzahl der in ar1 enthaltenen Elemente wird der folgende Code ausgeführt. Ist der Wert aus **ar2** gleich dem entsprechenden Wert aus **ar1** und ist bisher noch kein unterschiedlicher Wert aufgetaucht, dann wird der gemeinsame Inhalt in einer Variablen gespeichert, die zum Schluss von der Pfadangabe abgeschnitten werden kann.

```
                IF ar2(i) = ar1(i) AND ink = 0 THEN
                    stUrl_weg = stUrl_weg & ar1(i) & "/"
                ELSE
```

Ist irgendwann ein Unterschied zwischen den beiden Arrays aufgetaucht, so wird für jeden unterschiedlichen Wert die Bezeichnung für «setze ein Verzeichnis höher an» in der Variablen **stUrl** gespeichert.

```
                    stUrl = stUrl & "../"
                    ink = 1
                END IF
```

Sobald der Index, der durch die Variable **i** dargestellt wird, größer wird als die Anzahl der Elemente von **ar1**, wird für jeden weiteren Wert von **ar2** wieder ein weiteres **../** in der Variablen **stUrl** gespeichert.

```
            ELSE
                stUrl = stUrl & "../"
            END IF
        NEXT
        stUrl_gesamt = ConvertToUrl(oFeld2.Text)
        oFeld.boundField.UpdateString(stUrl & Right(stUrl_gesamt,Len(stUrl_gesamt)-
            Len(stUrl_weg)))
    END IF
END SUB
```

Ist die Schleife durch **ar2** komplett durchlaufen, so steht fest, ob und wie viele Verzeichnisse höher von der Datenbankdatei aus gesehen das Verzeichnis für die aufzurufende Datei liegt. Jetzt wird einmal die **stUrl_gesamt** aus dem Text im Dateiauswahlfeld erstellt. Diese enthält auch den Dateinamen. Anschließend wird in das graphische Feld der Wert für die Url übertragen. Der Wert für die Url beginnt mit **stUrl**, in der **../** in der erforderlichen Anzahl stehen. Dann wird von **stUrl_gesamt** der vordere Teil abgeschnitten, der bei den Pfaden zur Datenbankdatei und zur externen Datei gleich war. Der weg zu schneidende Inhalt wurde in **stUrl_weg** gespeichert.

Die Formulare «Bildaufnahme» und «Bildaufnahme_Name»

Äußerlich unterscheiden sich diese Formulare nicht vom Formular «Pfadeingabe». Das Formular «Bildaufnahme_Name» hat lediglich ein zusätzliches Feld, in dem der Name der Bilddatei abgespeichert werden kann, also z.B. «Haus.jpg». Dies könnte für Betriebssysteme, die nach der Dateiendung entsprechende Programme für das Öffnen von Dateien bereitstellen, wichtig sein, da der eigentliche Bildname nach dem Speichern der Bilddaten in der Datenbank nicht mehr nachvollzogen werden kann.

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

Makrosteuerung für das Formular «Bildaufnahme»

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.

```
SUB BildAuslesen
    DIM oDoc AS OBJECT
    DIM oDrawpage AS OBJECT
    DIM oForm AS OBJECT
    DIM oFeld AS OBJECT
    DIM oStream AS OBJECT
    DIM oShell AS OBJECT
    DIM oPath AS OBJECT
    DIM oSimpleFileAccess AS OBJECT
    DIM st AS STRING
    DIM stPfad AS STRING
    DIM stFeld AS STRING
    oDoc = thisComponent
    oDrawpage = oDoc.Drawpage
    oForm = oDrawpage.Forms.getByName("Formular")
    oFeld = oForm.getByName("GraphischesFeld")
    oStream = oFeld.BoundField.getBinaryStream
```

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

```
    oPath = createUnoService("com.sun.star.util.PathSettings")
    st = ""
```

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.

```
    stPfad = oPath.Temp & "/DbBild" & st
    oSimpleFileAccess = createUnoService("com.sun.star.ucb.SimpleFileAccess")
    oSimpleFileAccess.writeFile(stPfad, oStream)
```

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 Bilder.

```
    oShell = createUnoService("com.sun.star.system.SystemShellExecute")
    stFeld = convertToUrl(stPfad)
    oShell.execute(stFeld,,0)
END SUB
```

Makrosteuerung für das Formular «Bildaufnahme_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 «BildAuslesen»

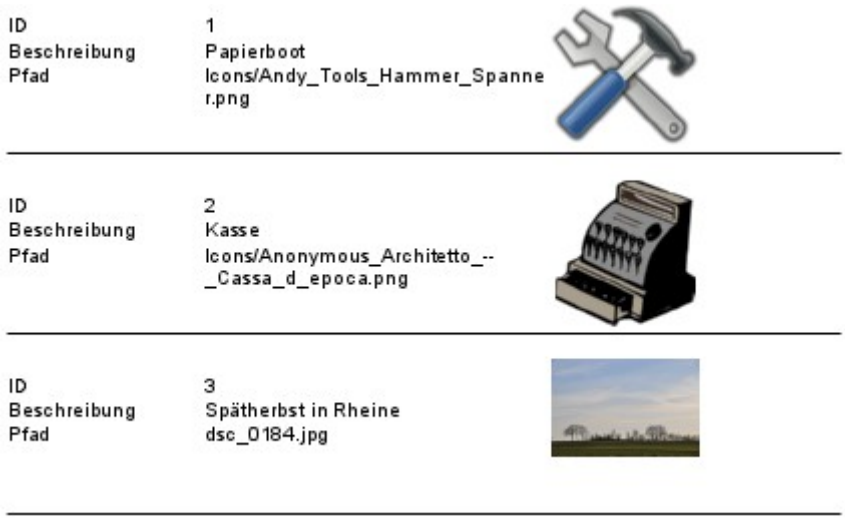
```
SUB BildAuslesen_mitName
    ...
    oFeld2 = oForm.getByName("BildName")
    stName = oFeld2.Text
    IF stName = "" THEN
        stName = "DbBild"
    END IF
```

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

```
    ...
    stPfad = oPath.Temp & "/" & stName
    ...
END SUB
```

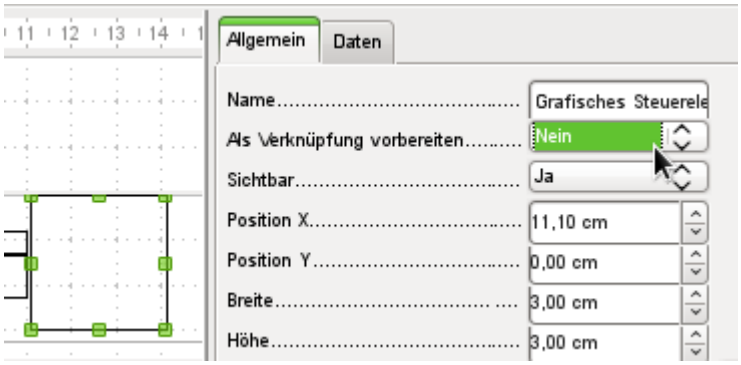
Berichte

Übersichtliche Ausdrucke der Daten zusammen mit Bildern sind ohne Probleme über den Report-Builder möglich.



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 *.pdf-Datei erscheint dann ein kleines Rechteck mit einer Meldung, dass diese Datei nicht geladen werden konnte.

Für eine kombinierte Speicherung von Bildern und Dateien kann stattdessen auch dafür gesorgt werden, dass die nicht anzeigbaren Dateien gar nicht erst an den Report-Builder weiter gegeben werden. Der Bericht könnte z.B. auf der folgenden Abfrage aufbauen:

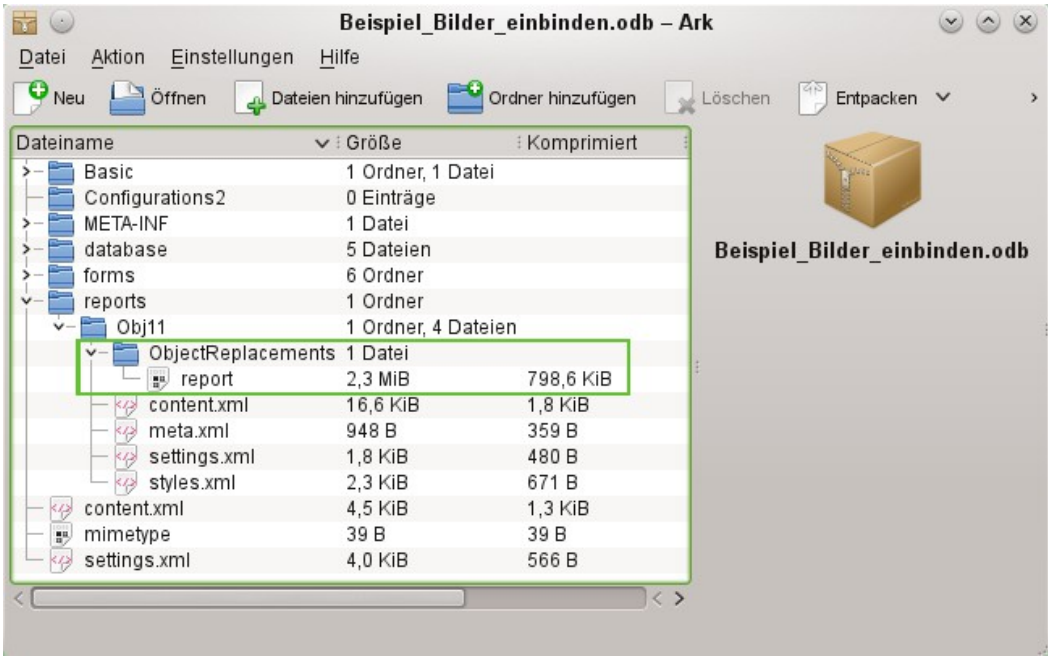
```
SELECT * FROM "Bilder"
    WHERE UPPER ( "Pfad" ) LIKE '%.JPG'
        OR UPPER ( "Pfad" ) LIKE '%.PNG'
```

Mit diesem Code würden nur *.jpg- und *.png-Bilder angezeigt. Damit wäre vermutlich schon ein Großteil der üblichen Formate erfasst.

Jetzt ist der Bericht wieder zum Bearbeiten zu öffnen. Über den Report-Navigator wird durch einen Klick auf den obersten Eintrag «Bericht» in den **Eigenschaften → Daten → Art des Inhaltes** auf «Abfrage» umgestellt und in **Daten → Inhalt** die entsprechende Abfrage ausgesucht.

Hinweis

Nach dem erneuten Editieren von Berichten mit Bildern fällt auf, dass die Datenbankdatei deutlich größer wird. Bei den Tests zu dieser Beschreibung mit großen Bilddateien entstand plötzlich eine Datei der Größe 16 MB. Innerhalb der *.odb-Datei legt Base aus nicht nachvollziehbaren Gründen im Berichtsverzeichnis einen Ordner «ObjectReplacements» an. Dieser Ordner enthält dann eine Datei «report», die für eine entsprechende Vergrößerung der *.odb-Datei sorgt.



Wenn die Datenbankdatei in einem Packprogramm geöffnet wird ist dieser Ordner mit Inhalt im Verzeichnis «reports» im Unterverzeichnis zu dem jeweiligen Bericht sichtbar. Dieses Verzeichnis kann über das Packprogramm gefahrlos gelöscht werden.

Wenn Berichte nicht wiederholt editiert werden, reicht ein einmaliges Löschen des Verzeichnisses aus.

Der Bug ist hier gemeldet: *https://bugs.freedesktop.org/show_bug.cgi?id=80320*

Mailversand aus einer Datenbank heraus

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. Zum Mailversand 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.

Tabellen

Die Datenbank besteht aus drei Tabellen.

In der Tabelle "Kontakte" werden folgende Felder definiert:

Feldname	***Feldtyp***	***Beschreibung***
ID	Integer	Primärschlüssel der Tabelle. Der Primärschlüssel muss eindeutig sein. Das Feld kann auch als Auto-Wert-Feld gesetzt wer-

		den.
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.

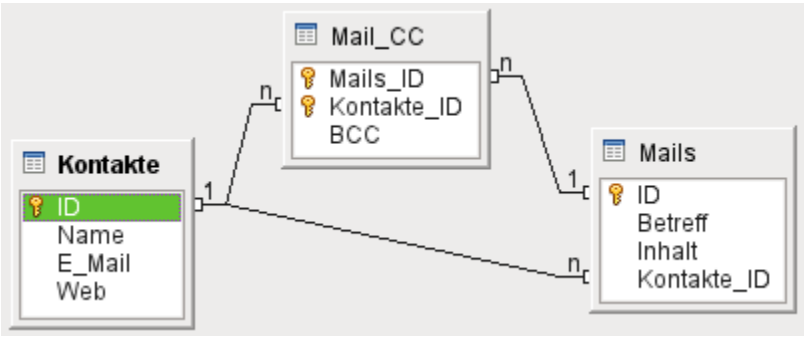
In der Tabelle "Mails" werden folgende Felder definiert:

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

In der Tabelle "Mail_CC" werden folgende Felder definiert:

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

Alle drei 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 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.

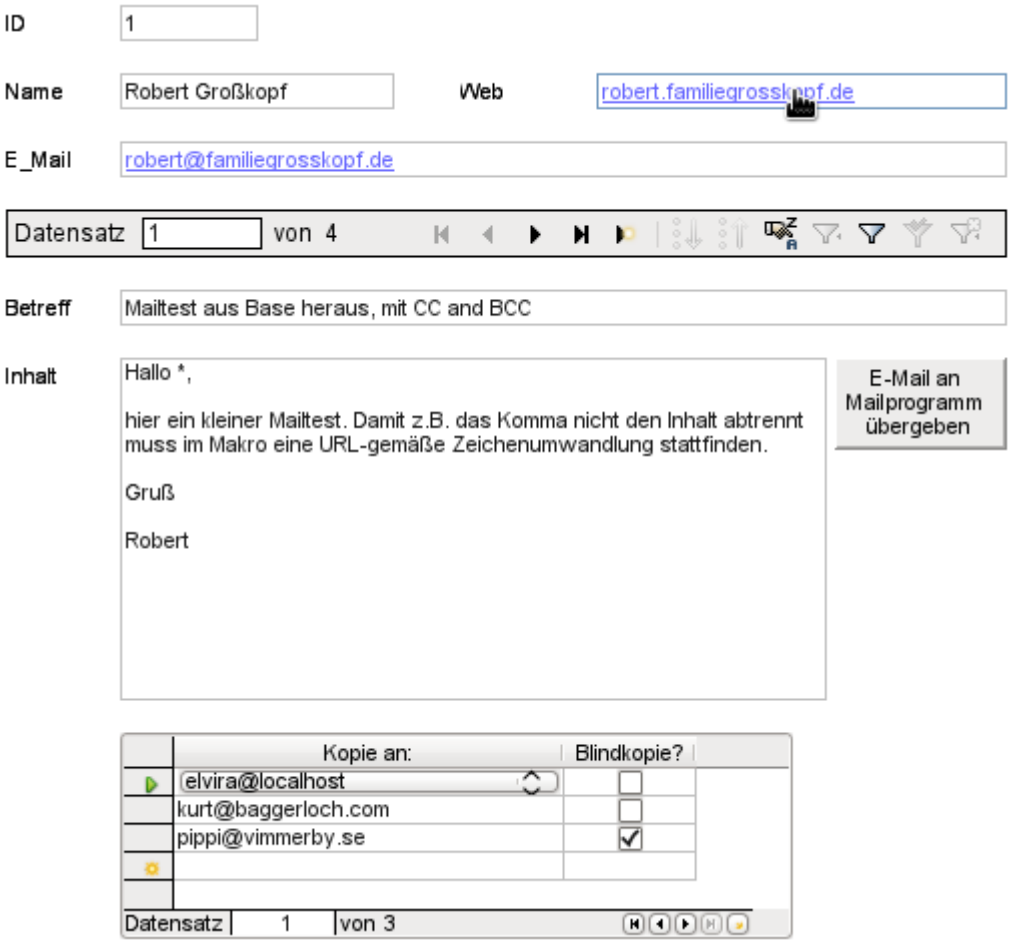
Ein Kontakt kann auch mehrmals als Kopieempfänger zugeordnet werden. Das Verhältnis von "Kontakte" zu "Mail_CC" ist 1:n.

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

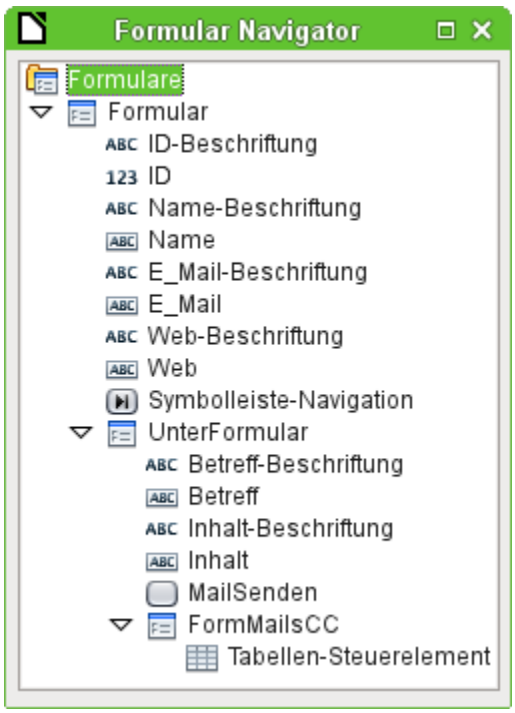
Formular

Das Beispiel ist in einem einzigen Formular zusammen gefasst. In dem Formular werden alle drei Tabellen benötigt, damit eine E-Mail mit allen zur Verfügung stehenden Funktionen an das Mailprogramm weiter gegeben werden kann.

Das folgende Bild zeigt dabei zwei Felder, die bereits durch die Formatierung der Schrift (unterstrichen und blau) als Felder mit einem Link gekennzeichnet sind. Felder, die mit einem Link direkt funktionieren, gibt es in Base nicht. Um trotzdem eine entsprechende Funktion bereit zu stellen ist ein Eingriff über Makros erforderlich.



Das Formular ist von oben nach unten in ein Hauptformular, ein Unterformular und ein Unterformular des Unterformulars gegliedert. Die entsprechenden Gliederung ist aus der Ansicht des Formularnavigators zu ersehen:



Sämtliche Einträge oberhalb der Navigationsleiste werden über «Formular» in der Tabelle «Kontakte» gespeichert. Das Formular «Formular» ist mit «UnterFormular» über das Feld "Kontakte"."ID" mit dem Feld "Mails"."Kontakte_ID" verknüpft. In «UnterFormular» werden die gesamten Mails abgespeichert und auch der Mailversand gestartet. Lediglich die Kopien der Mails müssen in einem Unterformular des Unterformulars, dem Formular «FormMailsCC», verwaltet werden. Schließlich soll es möglich sein, beliebig viele Mailadressen auszuwählen, an die eine Kopie der Mail gehen soll. «UnterFormular» und «FormMailsCC» sind über die Felder "Mails"."ID" und "Mail_CC"."Mails_ID" miteinander verknüpft.

Makrosteuerung für den Mauszeiger über einem Link

Der Mauszeiger wird über den entsprechenden **`UnoService`** beeinflusst. Die Typen für den Mauszeiger sind in **`com.sun.star.awt.SystemPointer`** verzeichnet. Die 27 verweist hier auf das Aussehen des Zeigers als Hand. 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`**.

```
SUB Mauszeiger(oEvent AS OBJECT)
    DIM oPointer AS OBJECT
    oPointer = createUnoService("com.sun.star.awt.Pointer")
    oPointer.setType(27)
    oEvent.Source.Peer.SetPointer(oPointer)
END SUB
```

Makrosteuerung für den Programmstart mit Link

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

```
SUB Website_Aufruf(oEvent AS OBJECT)
    DIM oFeld AS OBJECT
    DIM oShell AS OBJECT
    DIM stFeld AS STRING
```

```
    oFeld = oEvent.Source.Model
```

Ausgangspunkt ist wie beim Mauszeiger direkt das Feld, in dem der Klick mit der Maus erfolgt. Das Makro wird über **Textfeld → Eigenschaften → Ereignisse → Maustaste gedrückt** ausgelöst. Das Feld ist über diesen Weg bereits bekannt. Der Text, den das Feld anzeigt, kann ausgelesen werden. Enthält das Feld keinen Text, so braucht auch kein Browser oder Mailprogramm gestartet zu werden. Die Prozedur wird also bei einem leeren Feld abgebrochen.

```
    stFeld = oFeld.Text
    IF stFeld = "" THEN
        EXIT SUB
    END IF
```

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:`**.

```
    IF InStr(stFeld,"@") THEN
        stFeld = "mailto:"+stFeld
```

Beginnt der Text mit **`http://`**, so handelt es sich um eine vollständige Internetadresse. 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.

```
    ELSEIF InStr(stFeld,"http://") THEN
        stFeld = convertToUrl(stFeld)
    ELSE
        stFeld = "http://"+stFeld
        stFeld = convertToUrl(stFeld)
    END IF
```

Ist in dem Text weder @ noch **`http://`** enthalten, so wird hier davon ausgegangen, dass es sich um eine Internetadresse ohne den entsprechenden Vorspann handelt. 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 zum Schluss lediglich die entsprechende Adresse an die Benutzeroberfläche weiter gegeben, die dann ein dafür entsprechendes Programm sucht und öffnet.

```
    oShell = createUnoService("com.sun.star.system.SystemShellExecute")
    oShell.execute(stFeld,,0)
END SUB
```

Makrosteuerung für den Aufruf des Mailprogramms mit Parametern

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 mit **`mailto:Empfänger?subject=...&body=...&cc=...&bcc=...`** . 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 gelingen kann. Das Formular müsste dazu aber sinnvollerweise noch mindestens mit einem Dateiauswahlfeld ergänzt werden, da vermutlich nicht jede Person den gleichlautenden Mailanhang erhalten soll.

```
SUB Mail_Aufruf
    DIM oDoc AS OBJECT
    DIM oDrawpage AS OBJECT
    DIM oForm AS OBJECT
    DIM oFeld1 AS OBJECT
    DIM oFeld2 AS OBJECT
    DIM oFeld3 AS OBJECT
    DIM oFeld4 AS OBJECT
    DIM oShell AS OBJECT
```

```
DIM oColumns AS OBJECT
DIM oDatenquelle AS OBJECT
DIM oVerbindung AS OBJECT
DIM oSQL_Anweisung AS OBJECT
DIM oAbfrageergebnis AS OBJECT
DIM inIndex AS INTEGER
DIM i AS INTEGER
DIM loMails_ID AS LONG
DIM stFeld1 AS STRING
DIM stFeld2 AS STRING
DIM stFeld3 AS STRING
DIM stFeld4 AS STRING
DIM stSql AS STRING
DIM stMail_CC AS STRING
DIM stMail_BCC AS STRING
DIM st_CC AS STRING
DIM st_BCC AS STRING
DIM arSo_Zeichen()
DIM arSo_Ersatz()
oDoc = thisComponent
oDrawpage = oDoc.Drawpage
```

Nachdem alle Variablen deklariert sind wird der Pfad zu den verschiedenen Feldern geklärt. Anschließend wird in dem «UnterFormular», dem die Tabelle "Mails" zugrunde liegt, 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.

```
oForm = oDrawpage.Forms.getByName("Formular")
oFeld1 = oForm.getByName("E_Mail")
oSubForm = oForm.getByName("UnterFormular")
oFeld2 = oSubForm.getByName("Betreff")
oFeld3 = oSubForm.getByName("Inhalt")
stFeld1 = oFeld1.Text
oColumns = oSubForm.Columns
inIndex = oSubForm.findColumn("ID")
loMails_ID = oSubForm.getLong(inIndex)
IF stFeld1 = "" THEN
   msgbox "Keine Mailadresse vorhanden." & CHR(13) &
      "Das Mailprogramm wird nicht aufgerufen" , 48, "Mail senden"
   EXIT SUB
END IF
```

Enthält der Datensatz aus der Tabelle "Kontakte" keine E-Mail-Adresse, so wird die Prozedur direkt abgebrochen. Anschließend wird im «Betreff» und im «Inhalt» der Text 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 Ulr-konformen Textes.

```
stFeld3 = oFeld3.Text
stFeld2 = Mid(ConvertToUrl(oFeld2.Text),9)
stFeld3 = Mid(ConvertToUrl(oFeld3.Text),9)
```

Leider greift die Konvertierung zur Url bei einigen Sonderzeichen nicht. Nach Kommas bricht so z.B. der Text ab. Deswegen muss der bereits über die Funktion **`ConverToUrl`** bearbeitete Text noch einmal auf verschiedene Zeichen durchsucht werden. Über zwei Arrays werden hier noch diverse Sonderzeichen zusätzlich getestet und ersetzt. 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.

```
arSo_Zeichen = Array("\","^","~","[","]"," ","&","+",",",";",":","=","?","@")
arSo_Ersatz = Array("5C","5E","7E","5B","5D","20","26","2B","2C","3B",
   "3A","3D","3F","40")
FOR i = LBound(arSo_Zeichen()) TO UBound(arSo_Zeichen())
   stFeld3 = Join(Split(stFeld3,arSo_Zeichen(i)),"%" & arSo_Ersatz(i))
```

```
NEXT
```

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

```
stMail_CC = ""
stMail_BCC = ""
st_CC = ""
st_BCC = ""
oDatenquelle = ThisComponent.Parent.CurrentController
IF NOT (oDatenquelle.isConnected()) THEN
   oDatenquelle.connect()
END IF
oVerbindung = oDatenquelle.ActiveConnection()
oSQL_Anweisung = oVerbindung.createStatement()
stSql = "SELECT ""Kontakte"".""E_Mail"", ""Mail_CC"".""BCC"" FROM ""Mail_CC"",
   ""Kontakte"""
stSql = stSql + " WHERE ""Mail_CC"".""Kontakte_ID"" = ""Kontakte"".""ID"" AND
   ""Mail_CC"".""Mails_ID"" = '"+ loMails_ID +"'"
oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql) ' Ergebnis auswerten
WHILE oAbfrageergebnis.next
   IF oAbfrageergebnis.getBoolean(2) = TRUE THEN
      stMail_BCC = stMail_BCC + oAbfrageergebnis.getString(1) + ","
   ELSE
      stMail_CC = stMail_CC + oAbfrageergebnis.getString(1) + ","
   END IF
WEND
```

Über eine Abfrage werden die Mailadressen ermittelt, an die ein Kopie bzw. Blindkopie weitergereicht werden soll. 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. Hier ist eine alternative Formulierung für Thunderbird direkt mit dem Shell-Kommando aufgeführt.

```
IF stMail_BCC <> "" THEN
   st_BCC = Left(stMail_BCC,Len(stMail_BCC)-1)
   stMail_BCC = "&bcc="+st_BCC
   st_BCC = ",bcc='"+st_BCC+"'"
END IF
IF stMail_CC <> "" THEN
   st_CC = Left(stMail_CC,Len(stMail_CC)-1)
   stMail_CC = "&cc="+st_CC
   st_CC = ",cc='"+st_CC+"'"
END IF
oShell = createUnoService("com.sun.star.system.SystemShellExecute")
oShell.execute("mailto:"+stFeld1+"?subject="+stFeld2+"&body="+stFeld3
   +stMail_CC+stMail_BCC,,0)
```

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

Für Thunderbird wird hier 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*

```
Shell("thunderbird -compose ""to="+stFeld1+st_CC+st_BCC+",subject="+stFeld2+
   ",body="+stFeld3+",attachment='file:///home/user/Testdatei.pdf'""")
```

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.

```
    END SUB
```

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.

Suchen und Filtern von Daten ohne Makroeinsatz

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

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

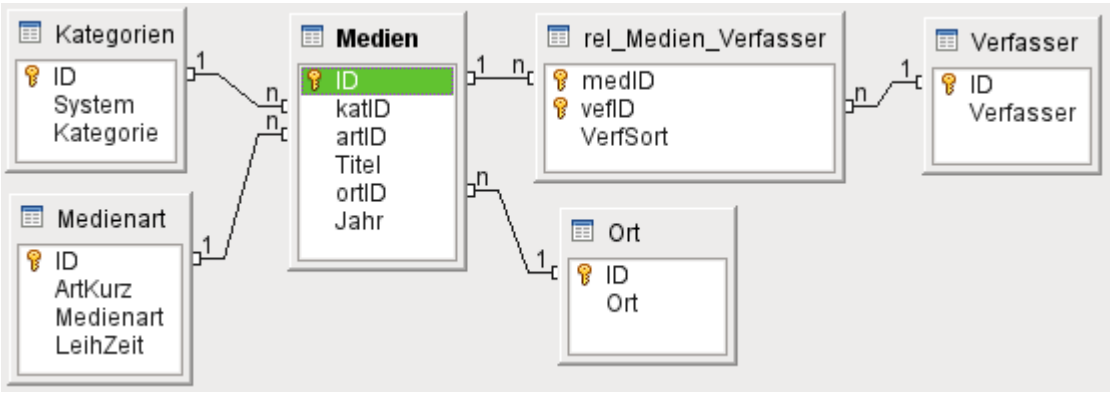
Beim Filtern wird durch einen oder mehrere Filterbegriffe direkt die Ergebnismenge aus der Tabelle heraus beeinflusst. Taucht das gewünschte Kriterium nur bei 3 von 100 Datensätzen auf, so werden eben nur 3 Datensätze angezeigt. Die Suchfunktion hingegen würde die 100 Datensätze durchlaufen und die einzelnen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

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

Als praktische Beispiel 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.

Tabellen



Über **Extras** → **Beziehungen** ist eine Übersicht der Tabellen zusammengefügt, durch die hindurch die Datensuche und Datenfilterung vorgenommen werden soll. Auf eine detaillierte Schilderung der einzelnen Felder soll hier verzichtet werden, da sie für die Suche nicht von Bedeutung ist.

Zentrale Tabelle ist die Tabelle "Medien". Die Tabelle "Kategorien", "Medienart" und "Ort" sind mit der Tabelle über Fremdschlüssel verbunden. Für einen Datensatz aus "Medien" kann es mehrere Datensätze aus "Verfasser" geben. Ebenso kann es für einen Datensatz aus "Verfasser" mehrere

Datensätze aus "Medien" geben. Aus dem Grunde ist über die Tabelle "rel_Medien_Verfasser" eine n:m-Beziehung erstellt worden.

Datenzusammenfassung in einer Ansicht

Damit sämtliche Daten direkt durchsucht werden können müssen sie zuerst in einer Abfrage oder Ansicht zusammengefasst werden. Der Code für eine Ansicht ist gleich dem Code für eine Abfrage. Base greift bei einer Ansicht aber auf die Daten zu, als ob sie direkt aus einer Tabelle kommen. Das läuft schneller.

Zentrale Tabelle ist die Tabelle "Medien":

```
SELECT * FROM "Medien"
```

Damit werden alle Datensätze aus der Tabelle Medien mit allen Feldern wieder gegeben. Die einzelnen Felder müssen nicht benannt werden und werden trotzdem angezeigt.

```
SELECT "Medien".* , "Kategorien"."System", "Kategorien"."Kategorie"
    FROM "Medien", "Kategorien"
    WHERE "Medien"."katID" = "Kategorien"."ID"
```

Damit bei zwei Tabellen noch klar ist, aus welcher Tabelle alle Felder angezeigt werden sollen, muss vor «*» die Tabelle verzeichnet sein: "Medien".* .

Aus der Tabelle "Kategorien" werden alle Felder bis auf den Primärschlüssel angezeigt. Das hat mehrere Gründe:

- Der Primärschlüssel aus der Tabelle "Kategorien" ist bereits als "katID" in der Tabelle "Medien" vertreten.
- Der Primärschlüssel aus der Tabelle "Kategorien" hat die gleiche Namensbezeichnung "ID" wie der Primärschlüssel aus der Tabelle "Medien". Während Abfragen so etwas noch in Base verarbeiten gibt die Datenbank eine entsprechende Fehlermeldung aus. "Kategorien"."ID" müsste mit Hilfe einer Aliasbezeichnung abgebildet werden, also z.B. "Kategorien"."ID" AS "KategorienID". Da dieses Feld aber ein Duplikat von "Medien"."katID" wäre ist darauf verzichtet worden.

Der obige Code hat jetzt einen Haken. Es werden nur die Datensätze angezeigt, bei denen "katID" aus der Tabelle "Medien" gleich "ID aus der Tabelle "Kategorien" ist. Ist allerdings bei einem Datensatz in der Tabelle "Medien" das Feld "katID" leer (**`NULL`**), dann ist dieser Datensatz aus der Anzeige ausgeschlossen. Da aber alle Datensätze aus "Medien" angezeigt werden sollen ist hier eine andere Formulierung der Bedingung notwendig:

```
SELECT "Medien".* , "Kategorien"."System", "Kategorien"."Kategorie"
    FROM "Medien"
    LEFT JOIN "Kategorien"
    ON "Medien"."katID" = "Kategorien"."ID"
```

Über **`LEFT JOIN`** werden aus der (links von der Verbindung liegenden) Tabelle "Medien" auf jeden Fall alle Datensätze abgebildet. Die Tabellenbezeichnungen folgen nicht direkt aufeinander. Sie werden durch **`LEFT JOIN`** von einander getrennt. Statt **`WHERE`** wird die Beziehung zwischen den Tabellen jetzt mit **`ON`** definiert.

```
SELECT "Medien".* , "Kategorien"."System", "Kategorien"."Kategorie", "Medienart"."ArtKurz",
"Medienart"."Medienart", "Medienart"."LeihZeit"
FROM "Medien"
LEFT JOIN "Kategorien"
ON "Medien"."katID" = "Kategorien"."ID"
LEFT JOIN "Medienart"
ON "Medien"."artID" = "Medienart"."ID"
```

Kommt eine weitere Tabelle dazu, so gilt immer noch: "Medien" ist die Tabelle, in der alle Datensätze liegen. Es werden zu allen Datensätzen aus "Medien" die "Kategorie" angezeigt. Wenn keine

"Kategorie" vermerkt ist wird der Datensatz aus "Medien" trotzdem angezeigt. Ebenso wird zu allen Datensätzen aus "Medien" eine "Medienart" angezeigt. Auch wenn die "Medienart" fehlt wird der Datensatz aus "Medien" weiter angezeigt.

Nur mittels eines **`LEFT JOIN`** (oder, bei umgekehrter Stellung, eines **`RIGHT JOIN`**) lassen sich wirklich alle Datensätze für eine Suche in einem Formular mit Unterformular wirkungsvoll zusammen fassen.

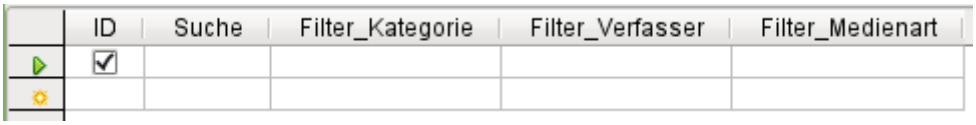
Die gesamte Zusammenfassung aller erforderlichen Daten in der Ansicht "SuchAnsicht" sieht schließlich folgendermaßen aus:

```
SELECT "Medien".* , "Kategorien"."System", "Kategorien"."Kategorie", "Medienart"."ArtKurz",
"Medienart"."Medienart", "Medienart"."LeihZeit", "Ort"."Ort", "Verfasser"."Verfasser",
"rel_Medien_Verfasser"."VerfSort", "rel_Medien_Verfasser"."vefID" ,
    IFNULL("rel_Medien_Verfasser"."vefID", 0 ) AS "VerfasserID",
    IFNULL("Medien"."katID", 0 ) AS "KategorieID",
    IFNULL("Medien"."artID", 0 ) AS "MedienartID"
FROM "Medien"
LEFT JOIN "Kategorien"
ON "Medien"."katID" = "Kategorien"."ID"
LEFT JOIN "Medienart"
ON "Medien"."artID" = "Medienart"."ID"
LEFT JOIN "Ort"
ON "Medien"."ortID" = "Ort"."ID"
LEFT JOIN "rel_Medien_Verfasser"
ON "rel_Medien_Verfasser"."medID" = "Medien"."ID"
LEFT JOIN "Verfasser"
ON "rel_Medien_Verfasser"."vefID" = "Verfasser"."ID"
```

Die mit einem Alias versehenen letzten drei Felder dienen dazu, dass für die anschließende Filterung nur Felder gefiltert werden, in denen auf jeden Fall ein Eintrag vorliegt. Ohne diese Zusätze könnte bei einer Filterung nicht mehr auf alle Datensätze zugegriffen werden. Siehe zu dieser Problematik die Abfrage «Filter_Form_Start».

Erstellung einer Filter-Tabelle

Die Filterung erfolgt, indem in eine Tabelle mit nur einem Datensatz über das Formular ein Wert eingetragen wird. Dieser Wert wird über eine Abfrage ausgewertet, so dass alle Datensätze mit diesem entsprechenden Wert angezeigt werden können.



	ID	Suche	Filter_Kategorie	Filter_Verfasser	Filter_Medienart
▶	☑				
✱					

Die Tabelle "Filter" hat als Primärschlüssel das Feld "ID". Der Primärschlüssel ist hier ein einfaches Ja/Nein-Feld. Der erste Wert für dieses Schlüsselfeld ist bereits gesetzt.

Daneben sind Felder für einen Suchbegriff in Textform sowie für verschiedene Fremdschlüsselfilter (Datenformat INTEGER) vorgesehen. In dem Feld "Filter_Kategorie" z. B. wird also nur ein Zahlenwert abgespeichert, nach dem dann die Datensätze durchsucht werden sollen.

Abfragen

Die meisten der folgenden Abfragen werden als Datengrundlage für die Formulare genutzt. Abfragen, die als Datengrundlage für Formulare dienen, sollten am besten editierbar sein. Dies wurde bei allen Abfragen berücksichtigt.

Eine Abfrage ist dann editierbar, wenn sie die Primärschlüssel aller Tabellen enthält, auf die sich die Abfrage bezieht.

Filter_Form_Start

Diese Abfrage ist lediglich in die Beispieldatenbank aufgenommen worden, um ein eventuell häufig auftauchendes Problem bei dem Einsatz von Filtern auf zu zeigen.

```
SELECT *
FROM "Medien"
WHERE "katID" =
    IFNULL(
    ( SELECT "Filter_Kategorie" FROM "Filter" WHERE "ID" = TRUE ),
    "katID" )
```

`SELECT * FROM "Medien"`: Zeige alle Datensätze aus der Tabelle "Medien" an.

`WHERE "katID" =`: Bei denen das Feld "katID" gleich dem folgenden Wert ist. Diese Bedingung schließt so bereits aus, dass das Feld "katID" **`NULL`** sein darf.

`IFNULL( <Bedingung> , "katID")`: Wenn die Bedingung keinen Datensatz zurück gibt, also **`NULL`** ist, dann soll stattdessen das Feld "katID" hinter dem Gleichheitszeichen erscheinen. Es würde also bei **`NULL`** lauten: **`WHERE "katID" = "katID"`**. Ist die Bedingung also leer, dann werden alle Datensätze angezeigt, bei denen in "katID" ein Eintrag vorliegt.

Die nachgefragte Bedingung ist eine Abfrage an das Feld "Filter_Kategorie" aus der Tabelle "Filter", bei der der Wert für den Primärschlüssel "ID" wahr (**`TRUE`**) ist. Die Filter-Tabelle wurde ja mit einem Primärschlüssel versehen, der aus einem Ja/Nein Feld (auch: wahr/falsch oder TRUE/FALSE oder 1/0) besteht. Mit dieser Unterabfrage wird höchstens ein Wert wieder gegeben. Da diese Unterabfrage lediglich in der Bedingung für die Datenauswahl erscheint ändert sie nichts daran, dass die Abfrage insgesamt editierbar bleibt.

Filter_Form

Die vorherige Abfrage hat das Problem, dass durch die Filterung von vornherein alle Datensätze ausgenommen sind, bei denen das Feld "katID" **`NULL`** ist. Deswegen muss der Code so ergänzt werden, dass in dem Vergleichsfeld aus der Abfrage immer ein Wert erscheint. Statt also bei einem leeren Feld "katID" keine Ausgabe zu erzeugen wird stattdessen ein Wert ausgegeben, der als "katID" nicht erscheint, da er nicht als Primärschlüssel in der Tabelle "Kategorien" vorgesehen ist.

```
SELECT "Medien".*, IFNULL( "katID", 0 ) AS "kat"
FROM "Medien" WHERE "kat" =
    IFNULL(
    ( SELECT "Filter_Kategorie" FROM "Filter" WHERE "ID" = TRUE ),
    "kat" )
```

Das Feld "katID" nimmt in der Beispieldatenbank mindestens den Wert '1' an. Durch die Zuweisung **`IFNULL( "katID", 0 ) AS "kat"`** wird einem neuen Feld mit der Aliasbezeichnug "kat" zuerst einmal immer der Wert von "katID" zugewiesen. Ist "katID" leer (**`NULL`**), dann wird in "kat" stattdessen '0' angezeigt. "kat" hat also im Gegensatz zu "katID" immer einen Wert vorzuweisen.

Mit diesem Feld "kat" wird jetzt der Vergleich durchgeführt. Ergibt die Unterabfrage **`SELECT "Filter_Kategorie" FROM "Filter" WHERE "ID" = TRUE`** keinen Wert, dann wird stattdessen "kat" mit "kat" verglichen. Alle Datensätze der Tabelle "Medien" (sowie außerdem "kat") werden angezeigt. Auch die Datensätze, bei denen "katID" NULL ist.

Diese Abfrage kann also komplett so in einem Formular genutzt werden, in dem nur nach der Kategorie gefiltert werden soll. Durch die Filterung passiert es nicht, dass von vornherein bestimmte Datensätze nicht angezeigt würden.

Filter_Form_Subform

Wird nicht nur ein einfaches Formular durchsucht, sondern z.B. der Inhalt eines Subformulars, so kann nicht direkt auf die Tabelle "Medien" des Hauptformulars Bezug genommen werden. Allerdings soll die Filterung des Unterformulars nicht, wie bei den eingebauten Filtern, nur das Unterformular filtern und immer noch alle Datensätze des Hauptformulars anzeigen. Stattdessen sollen die Datensätze ausgeschlossen werden, bei denen der gewünschte Begriff im Unterformular nicht vorhanden ist.

Für diese Art der Filterung wird die Ansicht "SuchAnsicht" durchsucht. Es wird in dem Beispiel nach einem Verfasser, genauer nach dem Fremdschlüssel des Verfassers "vefID" gesucht. Der Fremdschlüssel steht nicht in der Tabelle "Medien", da das Verhältnis von "Verfasser" zu "Medien" ein n:m-Verhältnis ist.

```
SELECT * FROM "Medien"
WHERE "ID" IN (
    SELECT DISTINCT "ID" FROM "SuchAnsicht" WHERE "VerfasserID" = IFNULL(
        ( SELECT "Filter_Verfasser" FROM "Filter" WHERE "ID" = TRUE ), "VerfasserID" )
    )
```

Durch die Abfrage werden alle Werte der Tabelle "Medien" angezeigt, auf die die Bedingung zutrifft: **`SELECT * FROM "Medien"`** .

Mit der Bedingung **`WHERE "ID" IN ( )`** kann über eine Unterabfrage der Inhalt in den Klammern auch deutlich mehr als nur einen Datensatz anzeigen. Nur die Bedingung **`IN ( )`** gilt für ganze Datengruppen, aus denen heraus der passende Wert gesucht wird.

Innerhalb der Klammern wird über **`SELECT DISTINCT "ID" FROM "SuchAnsicht"`** eine Gruppe von Primärschlüsselwerten der Tabelle "Medien" angegeben. Der Zusatz **`DISTINCT`** führt dazu, dass in diesem Fall kein Wert des Feldes "ID" doppelt erscheint.

Schließlich wird die Bedingung wieder, wie in der Abfrage «Filter_Form», in diesem Fall aber für das Feld "VerfasserID", formuliert. Das Feld "VerfasserID" ist in der Suchansicht bereits nach dem gleichen Verfahren erstellt worden, nach dem in der Abfrage «FilterForm" das Feld "kat" erstellt wurde: **`IFNULL("rel_Medien_Verfasser"."vefID", 0 ) AS "VerfasserID"`** . Damit ist sicher gestellt, dass bei einem nicht ausgewählten Verfasser auch die Datensätze angezeigt werden, die eventuell mit gar keinem Verfasser verbunden sind.

Filter_Form_Subform_3Filter

Mit Hilfe des gleichen Verfahrens wie bei "Filter_Form_Subform" können auch gleichzeitig mehrere Felder gefiltert werden.

```
SELECT * FROM "Medien"
WHERE "ID" IN (
    SELECT DISTINCT "ID" FROM "SuchAnsicht" WHERE "VerfasserID" = IFNULL(
        ( SELECT "Filter_Verfasser" FROM "Filter" WHERE "ID" = TRUE ), "VerfasserID" )
        AND "MedienartID" = IFNULL(
        ( SELECT "Filter_Medienart" FROM "Filter" WHERE "ID" = TRUE ), "MedienartID" )
        AND "KategorieID" = IFNULL(
        ( SELECT "Filter_Kategorie" FROM "Filter" WHERE "ID" = TRUE ), "KategorieID" )
    )
```

Wieder werden alle Felder in "SuchAnsicht" gelistet, die den Bedingungen entsprechen. Dabei werden die einzelnen Filter mit **`AND`** verknüpft. Es sollen also nur die Felder angezeigt werden, die wirklich allen eingestellten Filtern entsprechen. Damit wird die engste Auswahl getroffen. Hier könnte natürlich auch mit einem **`OR`** die Auswahl so dargestellt werden, dass entweder der eine oder der andere Wert enthalten sein muss. Nur würden in diesem Falle nicht ausgewählte Filter dazu führen, dass auf jeden Fall alle Datensätze aus "Medien" angezeigt werden.

Listenfeld_Kategorie_ohne_Eintrag

Sollen mit den Listenfeldern gegebenenfalls auch die Felder gefiltert werden, die gar keinen Eintrag haben (um z.B. diesen Fehler zu beheben …), so muss das Listenfeld neben den Fremdschlüsselwerten der eigentlichen Tabelle auch den Eintrag für gar keinen Fremdschlüsselwert aufzeigen. In der Abfrage "Filter_Form" wurde über den Alias "kat" Feldern ohne Fremdschlüssel der Wert '0' zugewiesen: **`IFNULL( "katID", 0 ) AS "kat"`** . Dieser Wert muss also bei der Filterung durch das Listenfeld gegebenenfalls auch in die Tabelle "Filter" eingetragen werden können.

```
SELECT "Kat", "ID"
FROM (
    SELECT '1' AS "C",
        'ohne Eintrag' AS "Kat",
        0 AS "ID"
        FROM "Kategorien"
    UNION
    SELECT "Kategorien"."System",
        LEFT( "Kategorien"."System" || SPACE( 7 ), 7 ) || ' - ' || "Kategorien"."Kategorie"
            AS "Kat",
        "Kategorien"."ID"
        FROM "Kategorien", "Medien" WHERE "Kategorien"."ID" = "Medien"."katID")
```

Der Start besteht, wie sonst bei Listenfelder üblich, aus der Aufzählung von zwei Felder. Das zuerst ermittelte Feld "Kat" zeigt den Text, der später im Listenfeld zu sehen ist. Das Feld "ID" wird nicht angezeigt. Es stellt die Verbindung zum Formularfeld und dem Feld der Tabelle bzw. Abfrage her, das dem Formularfeld zugrunde liegt.

In der Bedingung der Abfrage werden zwei Abfragen miteinander durch **`UNION`** verbunden. Diese Verbindung funktioniert nur, wenn die Abfragen sich auf Tabellen mit gleichen Feldtypen bezieht. Es kann auch nicht eine Tabelle mit 4 Feldern mit einer anderen mit 3 Feldern verbunden werden.

Standardmäßig werden keine doppelten Werte angezeigt. Dies entspricht dem Zusatz **`DISTINCT`**, wie er in den vorhergehenden Abfragen gebraucht wurde.

Auch eine Sortierung ist nicht möglich. Stattdessen wird fest nach dem ersten Feld der Abfragen/Tabellen sortiert.

Die erste Abfrage **`SELECT '1' AS "C", 'ohne Eintrag' AS "Kat", 0 AS "ID" FROM "Kategorien"`** schreibt in das Sortierfeld eine '1' und weist diesem Feld einen (beliebigen) Alias "C" zu. Mit dem Wert '1' wird sicher gestellt, dass in der späteren Sortierung dieser Eintrag ganz oben steht. Sämtliche Einträge aus "Kategorien"."System" beginnen nämlich mit Buchstaben, werden also nach einer Zahl eingeordnet.

Im Listenfeld erscheint also zuerst «ohne Eintrag». Wird «ohne Eintrag» ausgewählt, so wird der Wert '0' an das darunterliegende Feld der Tabelle weiter gegeben.

Die zweite Abfrage sucht über **`WHERE "Kategorien"."ID" = "Medien"."katID"`** nur die Kategorien heraus, die auch tatsächlich in der Tabelle "Medien" eingetragen sind. Dies ist hier notwendig, da die Kategorisierung der "Medien" einer allgemeinen Vorgabe für Bibliotheken entspricht und viele Kategorien bisher gar nicht genutzt werden.

Das erste Feld stellt das Systematikkürzel dar: **`"Kategorien"."System"`**. Nach diesem Feld wird durch **`DISTINCT`** sortiert.

Das zweite Feld kombiniert die Systematik mit der Kategorie: **`LEFT( "Kategorien"."System" || SPACE( 7 ), 7 ) || ' - ' || "Kategorien"."Kategorie" AS "Kat"`** . Da die Übersicht in dem Listenfeld tabellarisch dargestellt werden soll, wird die Länge des Eintrages für "Kategorien"."System" auf 7 Zeichen festgelegt. Dies entspricht dem längsten Eintrag in diesem Feld. Gegebenenfalls wird also über SPACE(7) mit maximal 7 Leerzeichen aufgefüllt. Mit der Darstellung von "Kategorien"."System" wird über einen Bindestrich «-» die Beschreibung

"Kategorien"."Kategorie" verbunden. Die **||** dienen hier als die Elemente, die die Verbindung von beliebig vielen Feldern zu einem einzigen Feld gestatten.

Im dritten Feld der zweiten Abfrage wird lediglich der Primärschlüssel der Tabelle "Kategorien" abgefragt, der zu dem jeweiligen Datensatz passt.

Die Hauptabfrage **SELECT "Kat", "ID"** fragt nun aus dieser Zusammensetzung der beiden Unterabfragen das zweite und das dritte Feld ab. Das erste Feld der Unterabfragen dient lediglich der Sortierung.

Suche_Form

Bei der Filterung wurden über Listenfelder nach festen Fremdschlüsseln Daten ausgefiltert. Die Suche in dieser Beispieldatenbank erlaubt die Eingabe eines beliebigen Begriffes in ein Textfeld. Dabei soll die Suche ähnlich funktionieren, wie es viele Nutzer von Suchmaschinen im Internet gewohnt sind.

Der Suchbegriff wird wie auch die Filter-Fremdschlüssel in der Datei "Filter" gespeichert und von dort aus für die Abfrage "Suche_Form" ausgelesen.

```
SELECT *
FROM "Medien"
WHERE LOWER ( "Titel" ) LIKE
    IFNULL( '%' || LOWER ( ( SELECT "Suche" FROM "Filter" WHERE "ID" = TRUE ) ) || '%',
        LOWER ( "Titel" ) )
```

Alle Datensätze aus der Tabelle "Medien" sollen angezeigt werden, die den Kriterien entsprechen: **SELECT * FROM "Medien"** . Die Suche wird nur für das Feld "Titel" durchgeführt, da in der Beispieltabelle nur im Feld "Titel" Text steht. Es ist allerdings genauso gut möglich, zusätzlich andere Felder, auch Zahlenfelder wie z.B. "Medien"."Jahr" oder auch "Medien"."ID" mit dem gleichen Begriff auf einmal zu filtern. Dies wird in der Abfrage «Suche_Form_Subform» gezeigt.

LOWER ("Titel") wird mit dem Suchbegriff verglichen. **LOWER** bedeutet hier lediglich, dass der Inhalt des Feldes "Titel" komplett in Kleinbuchstaben ausgelesen wird. Damit soll erreicht werden, dass Suchbegriffe unabhängig von Groß- und Kleinschreibung gefunden werden.

Die Bedingung **LIKE** durchsucht den Inhalt eines Feldes. Mit dem Zusatz «%» sind beliebig viele Zeichen gemeint. Vor dem Suchbegriff sowie nach dem Suchbegriff dürfen also beliebig viele Zeichen stehen: **'%' || LOWER ((SELECT "Suche" FROM "Filter" WHERE "ID" = TRUE)) || '%'** . Aus dem Feld "Suche" der Tabelle "Filter" wird der Begriff ausgelesen und gleichzeitig in Kleinbuchstaben umgewandelt. Ist kein Begriff dort eingetragen, so ist das Feld **NULL**. Damit wird auch die Zusammenfügung der Zeichen «%» mit **NULL** zu **NULL**. Sobald also kein Suchbegriff da ist soll statt der Auswertung des Eintrages in der Tabelle "Filter" mit **LOWER ("Titel")** verglichen werden. Da alle Datensätze in der Tabelle "Medien" mit einem Titel versehen wurden werden so alle Datensätze angezeigt, wenn in der Tabelle "Filter" kein Suchbegriff eingetragen wurde.

Wie die Abfrage "Filter_Form" funktioniert auch die Abfrage "Suche_Form" so nur in einem Formular, nicht auch im Unterformular.

Suche_Form_Subform

Der Ansatz, wie hier nicht nur im Hauptformular, sondern auch im Unterformular gesucht werden kann, entspricht dem Ansatz der Abfrage «Filter_Form_Subform».

```
SELECT *
FROM "Medien"
WHERE "ID" IN (
    SELECT DISTINCT "ID"
    FROM "SuchAnsicht"
```

```
    WHERE
        LOWER ( "Titel" ) LIKE
            IFNULL( '%' || LOWER ( ( SELECT "Suche" FROM "Filter" WHERE "ID" = TRUE ) )
            || '%', LOWER ( "Titel" ) )
        OR LOWER ( "Kategorie" ) LIKE
            '%' || LOWER ( ( SELECT "Suche" FROM "Filter" WHERE "ID" = TRUE ) ) || '%'
        OR LOWER ( "Medienart" ) LIKE
            '%' || LOWER ( ( SELECT "Suche" FROM "Filter" WHERE "ID" = TRUE ) ) || '%'
        OR LOWER ( "Ort" ) LIKE
            '%' || LOWER ( ( SELECT "Suche" FROM "Filter" WHERE "ID" = TRUE ) ) || '%'
        OR LOWER ( "Verfasser" ) LIKE
            '%' || LOWER ( ( SELECT "Suche" FROM "Filter" WHERE "ID" = TRUE ) ) || '%'
)
```

Mit **`SELECT * FROM "Medien" WHERE "ID" IN (SELCET DISTINCT "ID" FROM "SuchAnsicht WHERE … )`** werden alle Schlüsselwerte "ID" über die "SuchAnsicht" ermittelt. Aus der Tabelle "Medien" werden zu diesen Schlüsselwerten die Datensätze dargestellt.

Das Feld "Titel" wird genauso abgeglichen wie in der Abfrage «Suche_Form». Wenn also keine Eingabe in dem Feld "Suche" aus der Tabelle "Filter" steht, dann wird einfach **`LOWER ("Titel")`** mit **`LOWER ("Titel")`** verglichen. Dadurch werden alle Datensätze angezeigt.

Alle anderen durchsuchten Felder werden in der Bedingung mit **`OR`** verbunden. Das Suchwort soll also entweder in dem Feld "Titel" oder in dem Feld "Kategorie" oder in dem Feld "Medienart" usw. enthalten sein. Würden die Felder mit **`AND`** verbunden, so würden die Ergebnisse unnötig eingeschränkt.

Da die Bedingungen mit **`OR`** verbunden sind ist es außerdem nicht erforderlich, mehr als ein Feld so zu gestalten, dass bei einem leeren Feld "Suche" eben alle Datensätze angezeigt werden.

Formulare

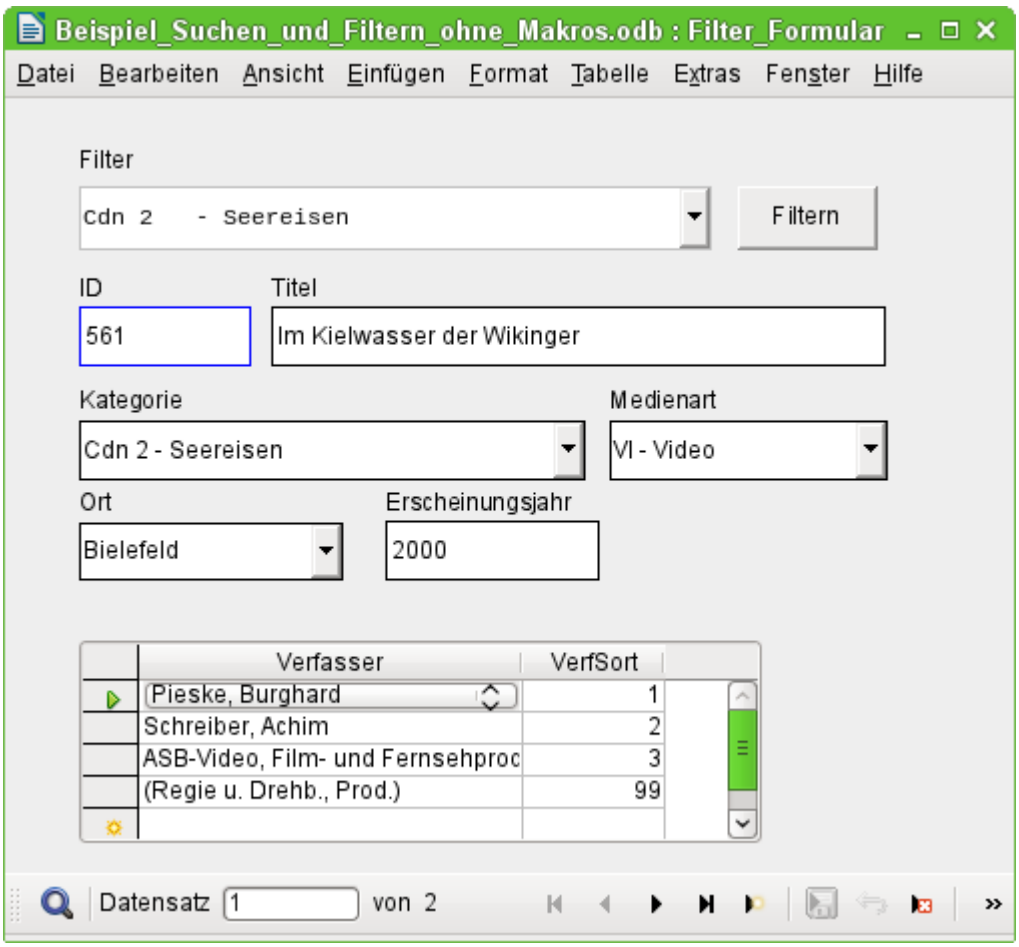
Bei den Formularen wird zuerst der zu ermittelnde Wert ausgesucht. Danach wird die Auswahl durch einen Button bestätigt.

Im darunter liegenden Formularteil wird das Filter- bzw. Suchergebnis angezeigt. Die Fundstellen in dem Formular werden nicht weiter markiert.

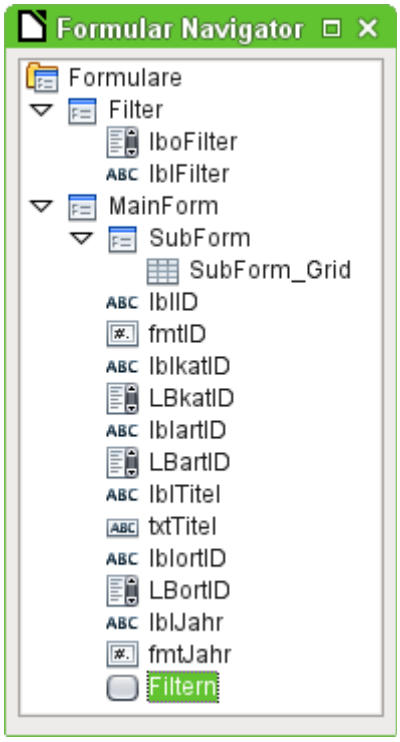
Filter_Formular

Äußerlich unterscheiden sich die Formulare «Filter_Formular», «Filter_Formular_Nullwerte» und «Filter_Formular_Subformular» nicht.

Über ein Listenfeld wird ein Wert ausgesucht. Der Button Filtern wird betätigt. Das Formular wird auf die ermittelten Datensätze eingestellt.



In diesem Beispiel wurde nach der Kategorie «Cdn 2» gesucht, die für den Begriff «Seereisen» steht. In der Datenbank gibt es insgesamt zwei Datensätze, die mit dieser Kategorie verzeichnet sind. Dies ist aus der Navigationsleiste ersichtlich.

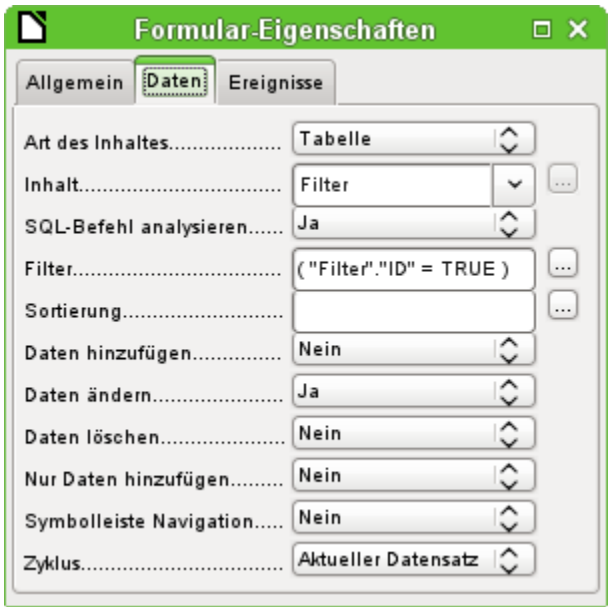


Der Formular Navigator gibt Aufschluss darüber, wie das Formular konstruiert ist. Die Konstruktion ist prinzipiell bei allen anderen Formularen dieser Beispieldatenbank gleich.

Es gibt zwei Formulare, die gleichberechtigt nebeneinander stehen.

Das Formular «Filter» enthält nur ein Listenfeld und das dazugehörige Beschriftungsfeld. Datengrundlage für dieses Formular ist die Tabelle "Filter". Mit dem Listenfeld ist das Feld "Filter_Kategorie" verbunden.

Das Formular «Filter» ist nicht für die Eingabe neuer Werte vorgesehen. Mit einem rechten Mausklick auf den Eintrag «Filter» im Formular Navigator und dem Menüpunkt «Eigenschaften» werden die Eigenschaften des Formulars sichtbar:



Das Formular zeigt nur den Datensatz an, für den aus der Tabelle "Filter" der Wert im Feld "ID" **TRUE** ist. Es werden keine Daten hinzugefügt, da dieser Datensatz ja bereits existiert. Es werden auch keine Daten gelöscht. Beides scheint hier etwas irritierend. Mit «Daten» ist hier aber der gesamte Datensatz gemeint. Einzelne Felder können dagegen geändert und auch geleert werden.

Eine Symbolleiste Navigation ist für dieses Formular nicht notwendig, da es sich immer im Datensatz 1 von 1 bewegt.

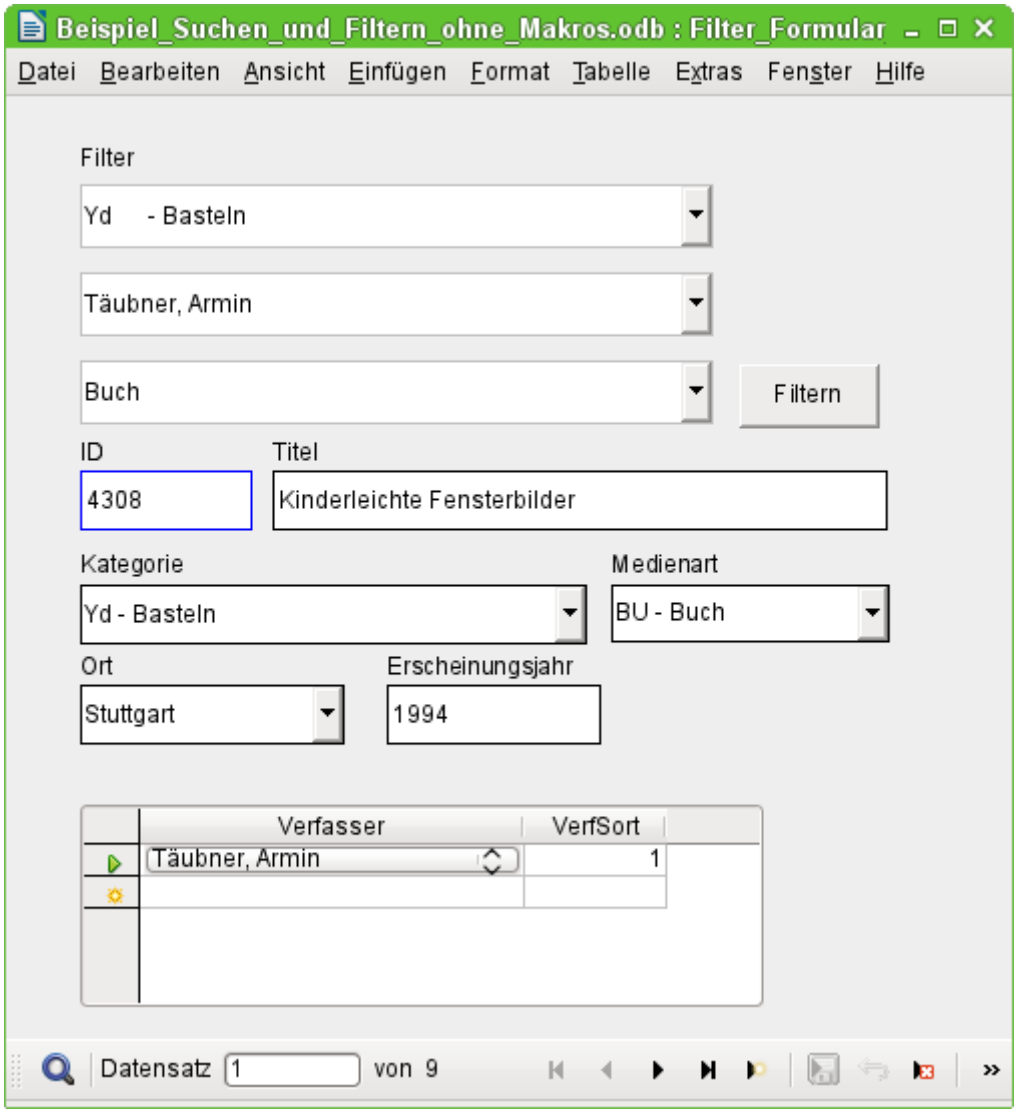
Das zweite Formular «MainForm» steht unabhängig neben dem Formular «Filter». Dieses Formular hat als Datenbasis eine Abfrage. Für das Gesamtformular «Filter_Formular» ist dies die Abfrage «Filter_Form». Diese Abfrage gibt die Datensätze nach dem im Nebenformular «Filter» eingetragenen Wert wieder.

Wichtigstes Element in dem Formular «MainForm» ist der Button «Filtern». Dieser Button ist mit der Aktion «Formular aktualisieren» verbunden. Er aktualisiert also die Anzeige des Formulars «MainForm» und des damit verknüpften Unterformulars «SubForm». Vor der Aktualisierung erledigt der Button allerdings noch eine andere, äußerlich nicht ersichtliches Aufgabe: Der Button liegt in einem anderen Formular als dem Formular, in dem der Filter eingestellt wurde. Eine Datenänderung im Formular «Filter» wird beim Verlassen des Formulars «Filter» und dem Betreten des Formular «MainForm» abgespeichert. Die Abspeicherung würde auch vorgenommen, wenn z.B. nur ein beliebiges Formularfeld des Formulars «MainForm» aufgesucht würde. Ein gesonderter Button zum Abspeichern des Wertes im Formular «Filter» ist also nicht erforderlich.

Das «Filter_Formular_Nullwerte» nutzt für das Listenfeld des Filters die Abfrage «Listenfeld_Kategorie_ohne_Eintrag». Dadurch können im Gegensatz zu dem vorhergehenden Formular auch nur die Datensätze gefiltert werden, die in der Tabelle "Medien" im Feld "katID" leer (**NULL**) sind.

Filter_Formular_Subformular_3Filter

Bereits das Formular «Filter_Formular_Subformular» filtert nach dem Verfasser im Subformular. Das untenstehende Formular birgt statt eines Listenfeldes für die Filterung gleich drei Listenfelder an.



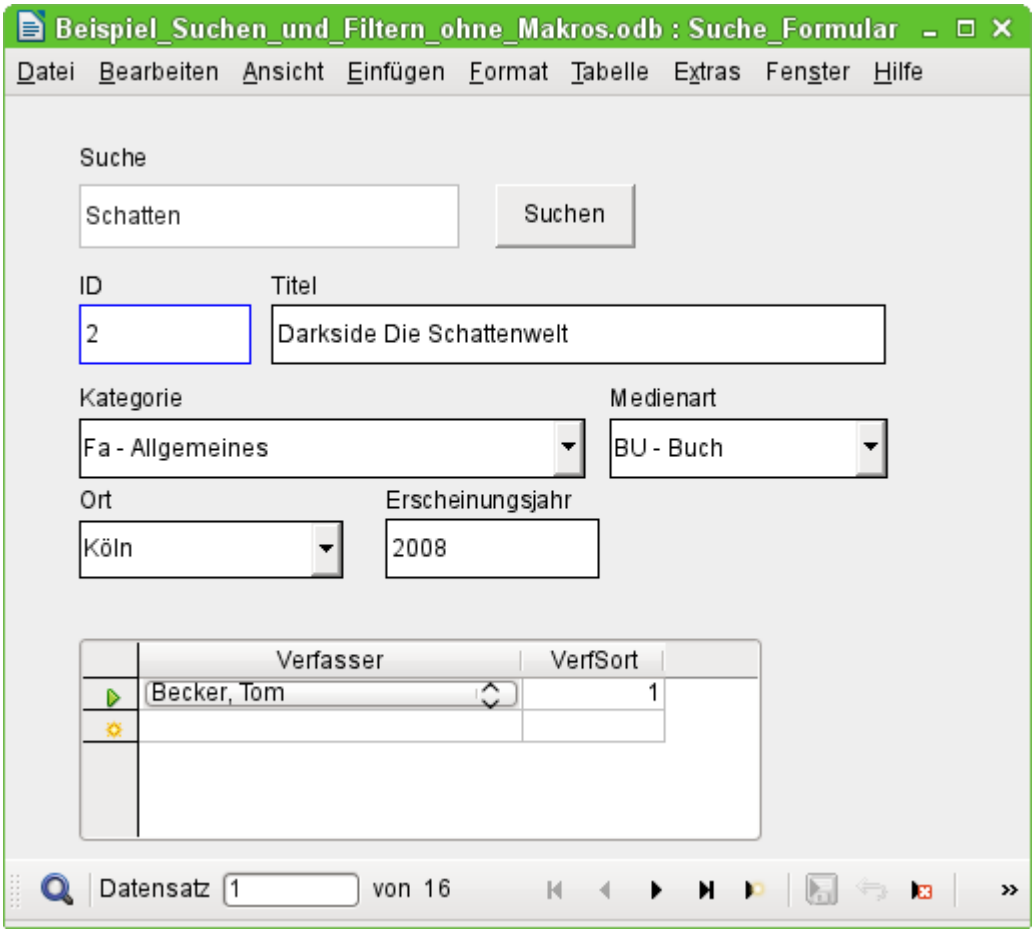
Die Listenfelder in diesem Formular sind nicht miteinander verbunden, d.h. der Eintrag des Wertes in einem Listenfeld sorgt nicht dafür, dass in dem anderen Listenfeld nur noch Werte angezeigt werden, die tatsächlich in der eingeschränkten Auswahl noch zur Verfügung stehen. Daher kann es bei direkter Betätigung aller drei Filter schnell dazu kommen, dass gar kein Datensatz mehr angezeigt wird, weil die Kombination einfach nicht vorhanden ist.

Es bietet sich für die Bedienung daher an, zuerst einmal nach einem Feld zu filtern und dann weiter zu sehen, welche Einträge überhaupt noch zur Auswahl stehen. Sich gegenseitig beeinflussende Listenfelder sind ohne den Einsatz von Makros nur mit vielen benutzerunfreundlichen Eingriffen in das Formular möglich. Dazu gehört vor allem eine laufende Betätigung des Buttons «Steuerelemente aktualisieren» in der Navigationsleiste für jedes verbleibende Listenfeld.

Datengrundlage für das Formular «MainForm» in diesem Formular «Filter_Formular_Subformular_3Filter» ist die oben vorgestellte Abfrage «Filter_Form_Subform_3Filter»

Suche_Formular

Äußerlich unterscheidet sich das «Suche_Formular» nur dadurch von dem «Filter_Formular», dass jetzt ein Texteingabefeld statt eines Listenfeldes zur Verfügung steht. Der Aufbau ist ansonsten identisch: Nebeneinander liegende Formulare «Filter» und «MainForm», Button zur Übernahme des Suchbegriffs als Aktualisierungsbutton in «MainForm» usw.



Das Nebenformular «MainForm» bezieht seine Daten aus der Abfrage «Suche_Form». Die Funktionsweise ist bei der Abfrage genauer erklärt.

In dem Formular «Suche_Formular_Subformular» ist lediglich die Datenquelle gegenüber dem gerade geschilderten Formular ausgetauscht. Die Datenquelle für dieses Formular ist die Abfrage «Suche_Form_Subform»