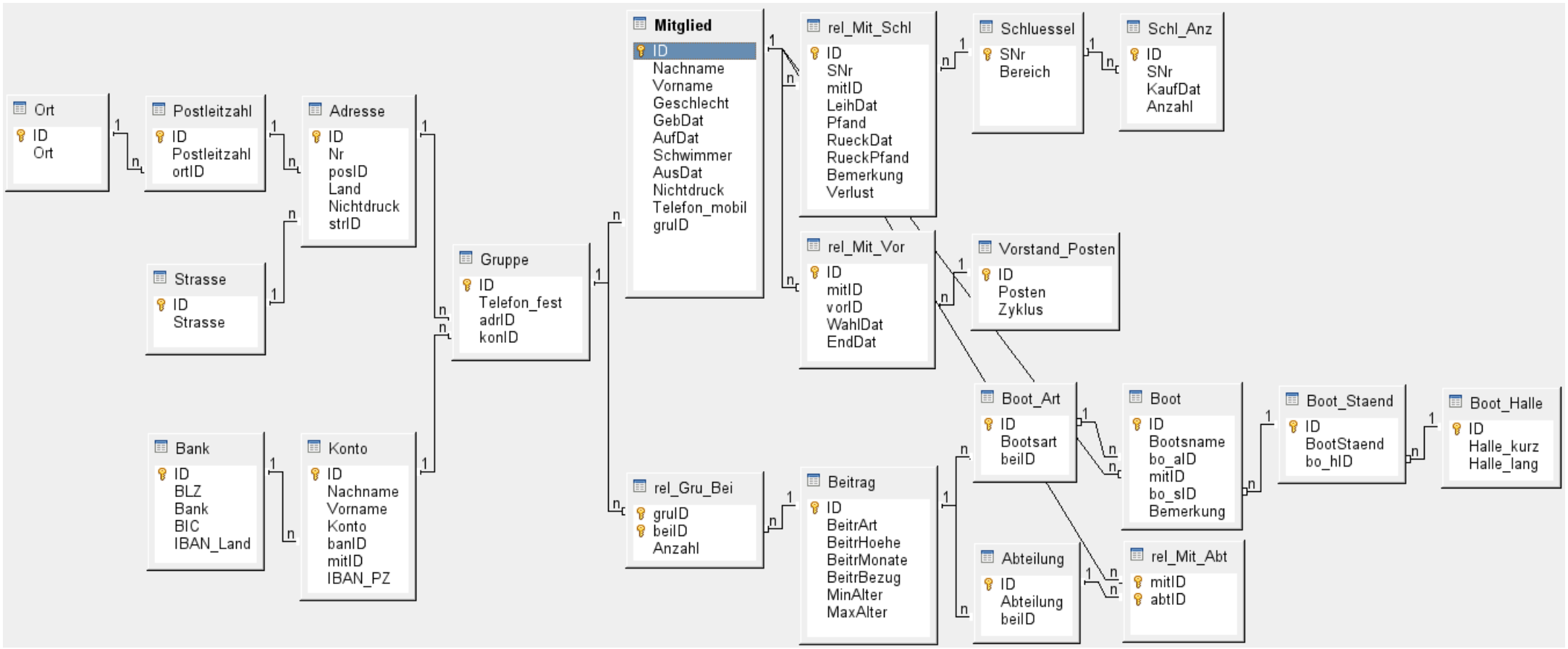
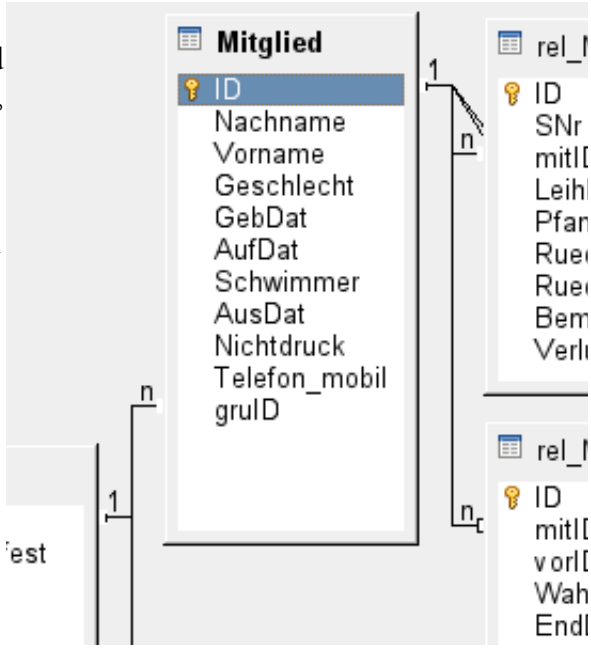
Datenbank OOo_Verein

Inhaltsverzeichnis

Datenbank „Verein“ - Relationen

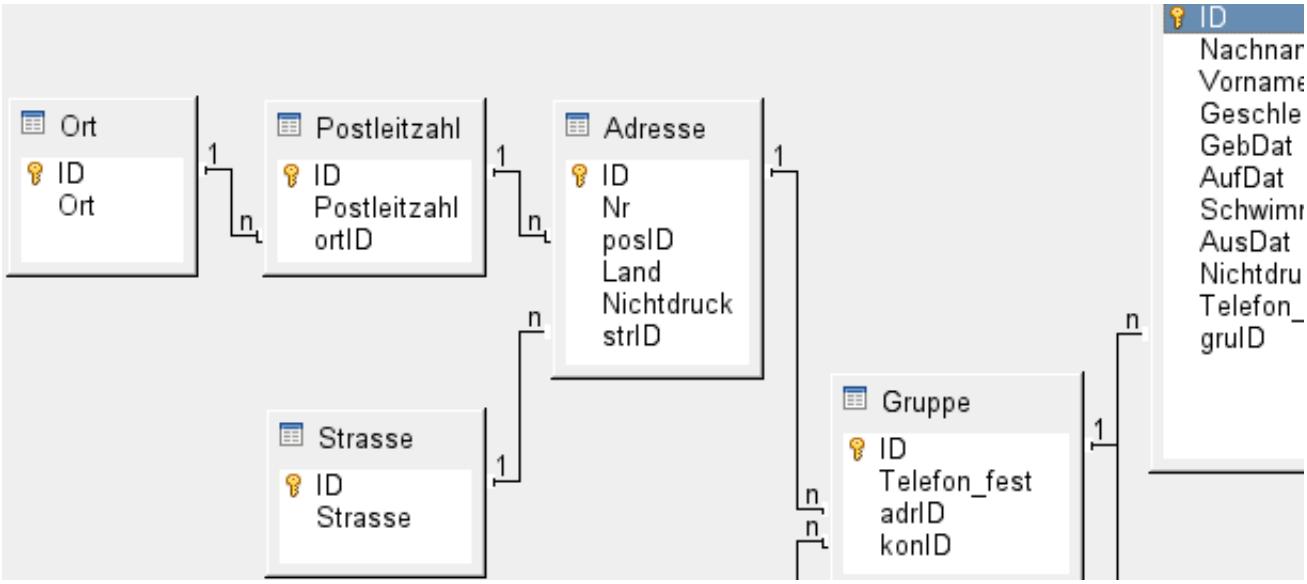


Zentrale Tabelle der Datenbank: die Mitgliedstabelle



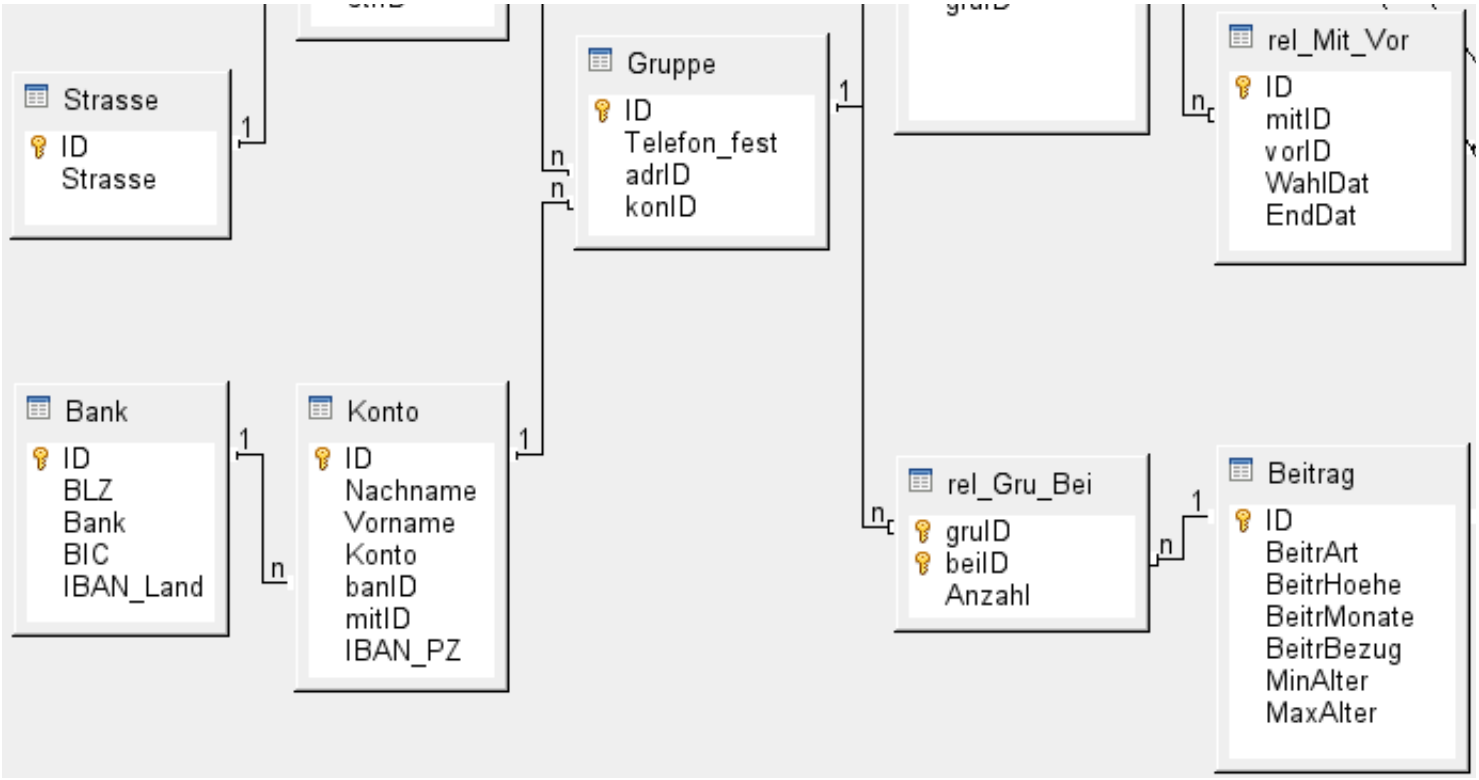
In der Mitgliedstabelle werden nur die Daten zusammengetragen, die auf genau dieses eine Mitglied zutreffen. „Nachname“ und „Vorname“ sind wohl klar, bei „Geschlecht“ wird lediglich „m“ und „w“ abgespeichert, um später Zuordnungen wie Anreden usw. zu ermöglichen. „GebDat“ ist das Geburtsdatum (für Vereine unabdingbar, es sei denn, sie legen keinen Wert auf Zuschüsse z.B. in der Jugendarbeit und wollen auch sonst nicht absichern, dass überhaupt jemand für einen Vorstandsposten vom Alter her wählbar ist). „AufDat“ ist das Aufnahmedatum in den Verein, „AusDat“ das Austrittsdatum. Natürlich kann auch einfach das Mitglied gelöscht werden – nur wie halte ich es dann mit eventuellen Jubiläumseinladungen, wenn Mitglieder in der Jugend im Verein waren und zum 25-Jährigen Jubiläum der Jugendabteilung gerne einmal die alten Gesichter wiedersehen würden?
„Schwimmer“ ist ein speziell für den dieser Datenbank zugrunde liegenden Verein notwendiger Eintrag. Wenn jemand nicht schwimmen kann gestaltet das die Teilnahme an Aktivitäten im Wassersport natürlich etwas problematischer. Bei manchen Aktivitäten können diese Personen dann schlicht nicht teilnehmen.
„Nichtdruck“ ist aus der leidigen Erfahrung erwachsen, dass Personen ihren Umzug beim Verein nicht melden und über den Postweg nicht erreichbar sind. Serienbriefe kommen nur dann zurück, wenn sie mit vollem Porto versehen sind – also nicht als Sammelzustellung laufen. Sammelzustellungen landen häufig irgendwo im Müll, wenn sie nicht zustellbar sind. Solange die Personen ihren Beitrag zahlen und sich über mangelnde Information nicht wundern kann der Verein also auf diese Weise Porto sparen.
„gruID“ ist über eine Gruppenzuweisung die Verbindung zu dem Tabellenstrang mit den Adressen sowie dem Tabellenstrang mit den Kontoinformationen. Hier zeigt sich die eine direkte n:1-Beziehung, gesehen vom Mitglied aus. Besteht ein Verein nur aus Junggesellen, so ist die Auslagerung der Adresse und des Kontos in großen Teilen nicht notwendig. Für Vereine mit vielen Ehepaaren und Familien macht das aber Sinn. Konkret: In dem zugrunde liegenden Verein gibt es ca. 500 Mitglieder und 270 verschiedene Adressen, an die die 2-monatlichen Rundschreiben gehen.
„Telefon_mobil“ wird zur Zeit noch nicht erfasst, macht aber Sinn direkt beim Mitglied, da über den Kontakt in der Regel nur eine Person zu erfassen ist.

Der Adressstrang



Sehr viele Orte haben mehrere Postleitzahlen. Für einen Verein in größeren Orten macht ist es deshalb sinnvoll, den Ortsnamen aus der Postleitzahltabelle auszulagern. Postleitzahlen wiederholen sich bei Adressen auch zur Genüge. Mein Beispielverein hat zwar 270 Adressen, aber nicht einmal 50 verschiedene Postleitzahlen – wobei einige vermutlich längst gelöscht werden könnten … Große Straßen werden bei der Adresseingabe häufiger vorkommen. Ob sich hier eine Auslagerung lohnt hängt ganz davon ab, wie die Eingabe machbar ist. Da ich mit Comboboxen arbeite, die den Wert direkt in die Tabelle speichern, die ID auslesen und in die Tabelle Adresse transportieren ist die Eingabe hier unproblematisch, also keine Hemmschwelle.
Die größere Tabelle „Adresse“ stellt über die Gruppenzuweisung die Verbindung zur Tabelle „Mitglied“ her. Hier ist sie getrennt aufgeführt wegen der Familien und der Ehepartner. Die Reihenfolge der Felder in der Tabelle entspricht nicht ganz der gewünschten Folge, lässt sich aber so nicht ohne weiteres rückgängig machen.
„Nr“ steht für die Hausnummer, kann also auch zum besseren Verständnis ausgeschrieben werden. „Land“ ist erforderlich, wenn, wie in meinem Beispielverein, die Post nicht nur in D, sondern auch in NL ausgetragen werden soll. „Nichtdruck“ wieder, wenn alle Personen, die mit der Adresse verknüpft sind, darüber nicht mehr erreichbar sind (Familie umgezogen, aber nicht die neue Adresse mitgeteilt – Porto für Rundschreiben sparen).
„Telefon_fest“ wird nicht mit der Adresse verbunden, sondern in einer gesonderten Gruppentabelle eingetragen, da in Mehrfamilienhäusern ja ohne weiteres auch mehrere Mietparteien gleichzeitig Mitglied im Verein sein können. In der Regel wird sich wohl die Gruppe mit den Personen decken, die über das gleiche Konto ihren Beitrag abbuchen lassen.

Der Kontostrang



Bank und Bankleitzahl sind in einer separaten Tabelle erfasst und werden dem jeweiligen Konto zugeordnet. Die Felder für den „Bank Identifier Code“ (BIC) und den Landesanteil der „International Bank Account Number“ (IBAN_Land) sind mit eingebaut, werden aber bisher im privaten Bereich weniger genutzt und daher im Formular bisher auch nicht abgefragt. Die Haupttabelle dieses Strangs ist die Tabelle „Konto“. Kontoinhaber kann dabei sowohl ein Mitglied sein (deshalb mitID in dieser Tabelle) als auch eine Person, die für ein Mitglied (meist bei Jugendlichen) den Beitrag bezahlt, selbst aber mit dem Verein sonst nichts zu tun hat. Deshalb hier die Felder für Nachname und Vorname. „Konto“ selbst soll die Kontonummer aufnehmen. Die Verbindung dieser Tabelle zur Mitgliedstabelle läuft über ein Feld „konID“ in der Gruppentabelle. In der „Konto“-Tabelle dann auch wieder ein Hinweis auf IBAN: IBAN_PZ stellt die Prüfziffer für die IBAN[1] dar.

Ein Konto wird natürlich nur deshalb in der Datenbank erfasst, weil von dem Konto ein Beitrag erhoben werden soll. Dieser Beitrag ist per Abfrage aus den Beitragssätzen des Vereins zu ermitteln. Die Tabelle „Beitrag“ ist hier wieder recht speziell auf den zugrunde liegenden Kanuverein zugeschnitten: Es gibt Beitragsarten für Erwachsene, Eheleute, Kinder und Jugendliche usw.; auch Abteilungsbeiträge oder, wie in obigem Verein, Liegeplatzgebühren für Boote.

Die Beitragshöhe ist natürlich der Betrag in €. Die Beitragsmonate haben etwas damit zu tun, dass es z.B. Sonderbeiträge gibt, die jährlich erhoben werden, während der Normalbeitrag monatlich erhoben wird. Das Alter hat mit den Beiträgen im Jugendlichen-Bereich etwas zu tun, ist also direkt

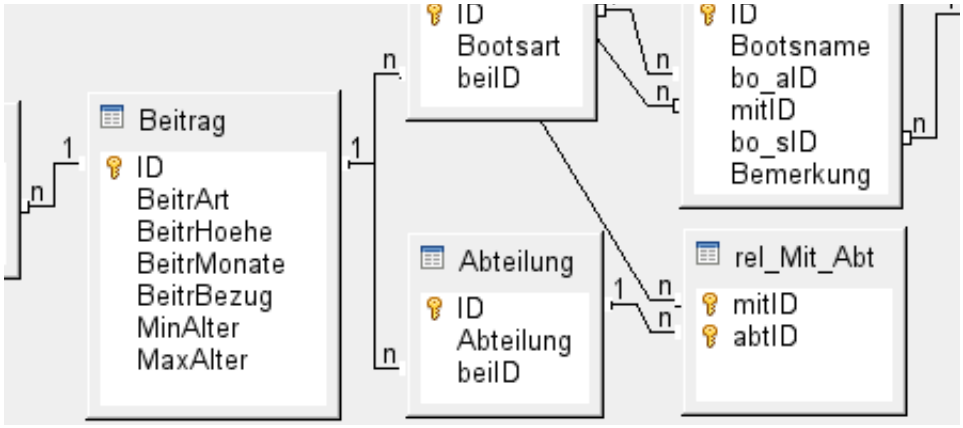
---

1 Eine IBAN beinhaltet
DEpp bbbb bbbb kkkk kkkk kk
Dabei bedeutet:
DE Länderkennzeichen für Deutschland
pp zweistellige Prüfziffer
bbbbbbbb die 8-stellige deutsche Bankleitzahl
kkkkkkkkkk die 10-stellige Kontonummer

in Verbindung mit dem Geburtsdatum zu sehen. Über den Beitragsbezug wird der Beitrag für ein einzelnes Mitglied berechnet. Die Tabellen für die Abteilungen sowie für die Bootsarten haben zwar eine direkte Verbindung zur „ID“ aus der Tabelle „Beitrag“, aber für altersabhängige Beiträge sowie Ermäßigungen usw. ist ein entsprechender Eintrag erforderlich.

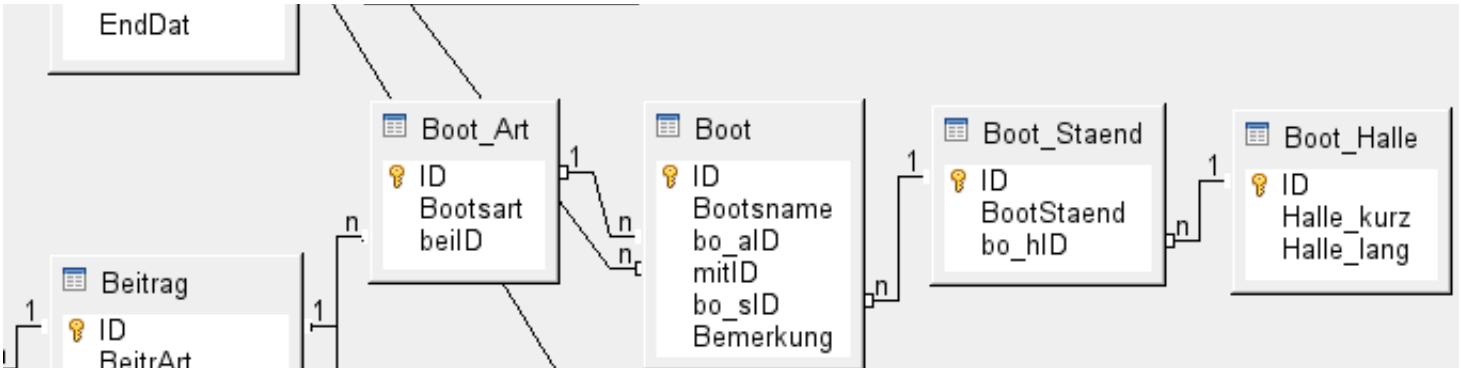
Da es ohne weiteres möglich ist, dass von einem Konto 2 Beiträge für Jugendliche unter 14 Jahren abgebucht werden müssen ist in der Verbindung von Beitrag zu Konto eine Anzahl vorgesehen. Der Beitrag wird zuerst durch eine Abfrage aufgrund der Einträge im Bereich Abteilung, Boot und Alter ermittelt. In der Tabelle „rel_Gru_Bei“ erfolgt dann gegebenenfalls eine Korrektur (Familienbeitrag usw.).

Der Abteilungsstrang



Vor allem in Sportvereinen gibt es verschiedene Abteilungen. Die Turnabteilung hat einen anderen Beitrag als die Volleyballer, die Rennsportabteilung im Kanuverein muss mehr aufbringen als jemand, der mit dem eigenen Boot unterwegs ist. Jedes Mitglied kann in mehreren Abteilungen aktiv sein, deshalb hier wieder die Relationentabelle.

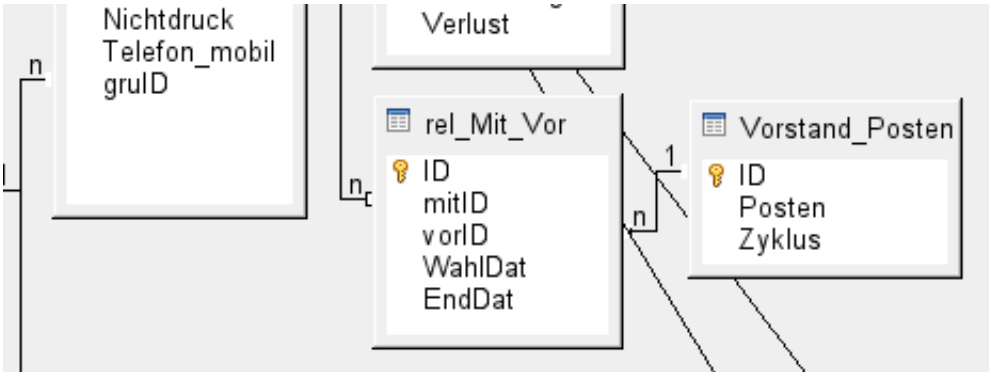
Der Bootsstrang



Jedes Mitglied kann mehrere Boote in den Hallen des Sportvereins liegen haben. Über diesen Strang sind Bootstypen und Lagerorte zu ermitteln. In der PHP-Ausführung wird daraus ein genauer Plan der verschiedenen Hallen mit den entsprechenden Bootsständern, Bootsarten und Eigentümern gezeichnet. Eine entsprechende Umsetzung in Form eines Berichtes innerhalb von OpenOffice-Base ist hier etwas komplizierter. Die Bootsarten sowie die Menge der Boote macht sich beim Beitrag bemerkbar.

Diese Strang ist natürlich sehr speziell auf den Fall eines Wassersport treibenden Vereins zugeschnitten. Ähnliches gibt es aber vermutlich auch bei anderen Vereinen.

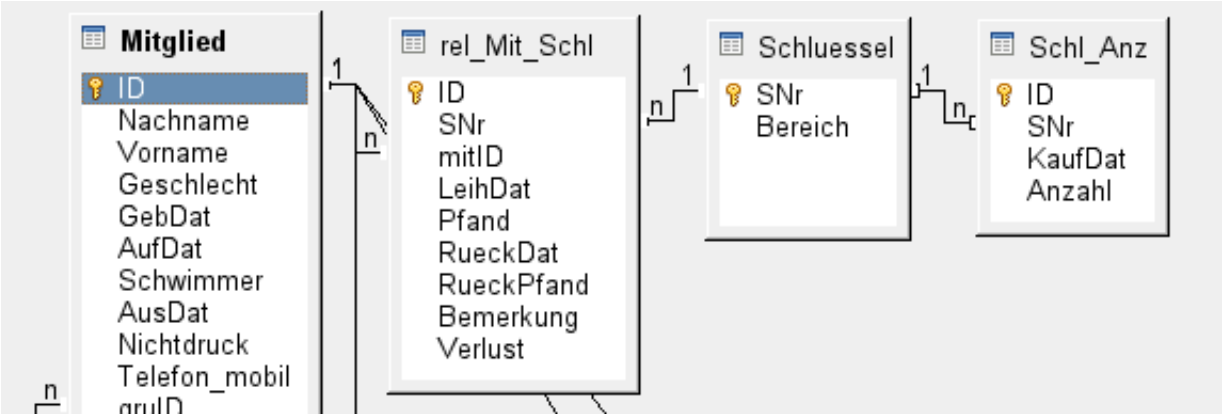
Der Vorstandsstrang



In einem bestimmten Zyklus wird neu gewählt. Dieser Zyklus ist zusammen mit den entsprechenden Posten aus der Tabelle rel_Mit_Vor ausgelagert. In dieser Tabelle ist dann verzeichnet, welches Mitglied (mitID) welchen Vorstandsposten (vorID) zu welcher Zeit inne hatte. Im Sinne einer Vereinschronik ist es daher sinnvoll, Mitglieder, die einmal im Vereinsvorstand gewesen sind, nicht durch Löschen aus der Mitgliederliste zu entfernen. Unter anderem deshalb dort auch das Austrittsdatum und anschließend auch keine Löschfunktion in den Formularen.

Durch die Eingabe des Wahldatums und des Enddatums als Datum des Ausscheidens aus dem Vorstand wird der Vorstand, der zu einer bestimmten Zeit aktiv war, auch auf Dauer abrufbar.

Der Schlüsselstrang



Vereine mit eigenen Vereinsheimen geben an Mitglieder Schlüssel aus. Dazu ist ein Schließsystem notwendig, das das einfache Duplizieren von Schlüsseln unmöglich macht. In dem vorliegenden Beispiel gibt es ein solches System mit Schlüsseln für verschiedene Schließbereiche. Nicht jedes Mitglied hat Zutritt zur Werkstatt (aus leidigen Erfahrungen), nicht jedes Mitglied kann einfach ein Rennboot aus der Halle nehmen oder sich an den PC im Vorstandszimmer setzen. Damit kontrolliert werden kann, wie viele Schlüssel denn im Umlauf sind, sind die Jeweiligen Einkäufe neuer Schlüssel in der Tabelle Schl_Anz zu verzeichnen. Nur in der Tabelle „Schluessel“ wurde übrigens für die Primärschlüsselnummer ein Feld gewählt, das nicht numerisch ist sondern der Nummer des Schlüssels selbst entspricht (GHS, GS1 usw.).

In der zentralen Verbindungstabelle zum Mitglied wird neben Schlüsselnummer und MitgliedsID

das Ausleihdatum, das erhobene Pfand, das Rückgabedatum und das zurückgegebene Pfand abgespeichert. Bei Bemerkungen wird notiert, ob der Schlüssel eventuell an ein minderjähriges Mitglied abgegeben wurde und die Unterschrift von einem Erziehungsberechtigten stammt, ob aus bestimmten Gründen kein Pfand erhoben wurde (z.B. beim Getränkelieferanten) etc. Verlust bedeutet im Vereinsleben immer eine Abwägungssache. Schließlich wird ein Schließsystem immer unsicher je mehr Schlüssel „verloren“ sind. Ein bei einer Bootsfahrt ins Wasser gefallener Schlüssel oder ein zerbrochener Schlüssel, der in Teilen abgegeben wurde, werden hier verzeichnet, damit schließlich die Anzahl der Schlüssel weiter stimmig nachvollziehbar bleibt.

Datenbank „Verein“ - Tabellen, die nicht in Relationen erfasst sind

In der Tabellenübersicht erscheinen einige Tabellen, die nicht direkt zur Eingabe von Daten genutzt werden. Dabei handelt es sich um folgende Tabellengruppen:

Ansichten, die durch Abfragen erstellt wurden und ihre Entsprechung auch in dem Abfragencontainer haben. Sie sind durch die zu normalen Tabelle unterschiedlichen Symbole sowie den entsprechenden Zusatz „_Ansicht“ erkennbar. Mehr dazu weiter unten.

Filtertabellen, die lediglich aus einem Datensatz bestehen und dafür sorgen sollen, dass in den Formularen beim Aufruf des folgenden Formulars auch der aktuelle Datensatz aktuell bleibt. Ähnlich den Tabellen mit dem Startbegriff „Filter“ ist die einzeilige Tabelle mit dem Startbegriff „Listfeld“.

Tabellen, die mit dem Begriff „tmp“ beginnen werden per Makro mit Daten gefüllt, die anschließend vor allem für Berichte zur Verfügung stehen.

Ansichten

Die Ansichten stellen eigentlich nur Abfragen im Tabellencontainer dar. Der Unterschied zu den Abfragen ist, dass die Ansichten auch beim Zugriff auf die Datenbank selbst verfügbar sind. Abfragen sind sozusagen mit der Benutzeroberfläche abgespeicherte Fragetexte, die erst an die darunter liegende HSQLDB geschickt werden, während Ansichten als „Views“ direkt in der HSQLDB abgelegt sind. Die Formulierung innerhalb der grafischen Benutzeroberfläche erscheint erst einmal gleich. Beim Abspeichern setzt die GUI, für den Benutzer unsichtbar, vor den SQL-Code allerdings ein „CREATE VIEW ...“, so dass daraus eben eine Tabellenansicht in der Datenbank festgeschrieben wird.

Die Ansichten werden hier meist dafür benutzt, auf entsprechend aufbereitete Daten aus den Grundtabellen über Makros zugreifen zu können. Die Makros arbeiten nämlich direkt mit der Datenbank, nicht mit der grafischen Benutzeroberfläche. Deshalb stehen auch in den Makros die Abfragen aus dem Abfragecontainer nicht zur Verfügung.

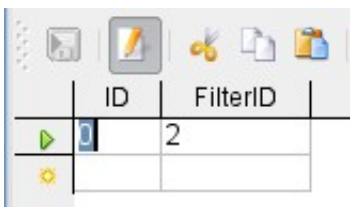
Auf Abfragen, die nur im direkten SQL-Modus funktionieren, kann Base nicht gefiltert zugreifen. Auch hier bietet der Umweg über Ansichten den entsprechenden Zugriff z.B. in Formularen.

Ansichten und Abfragen aktualisieren sich automatisch nach dem jeweils neuesten Datenstand in den ihnen zugrunde liegenden Tabellen. Dazu muss allerdings bei der Darstellung auf dem Bildschirm durch die Betätigung des Aktualisierungsbutton der geänderte Sachverhalt neu eingelesen werden.

Mehr zu Abfragen an entsprechender Stelle.

Filtertabellen

Diese Tabellen bestehen nur aus einem Datensatz.



Das Feld „ID“ beschreibt den für jede Tabelle notwendigen Primärschlüssel. Da die Tabelle nur einen Zeile enthält könnte grundsätzlich auch die „FilterID“ diese Eigenschaft übernehmen. Der Aufbau sollte so nur etwas besser in den Tabellenaufbau anderer Tabellen einzuordnen sein.

Der grüne Marker verweist darauf, dass gerade der 1. Datensatz mit der ID „0“ bearbeitet wird. Das Sternchen in der 2. Zeile teilt mit, dass hier ein neuer Datensatz eingetragen würde. In dem Feld „FilterID“ ist die Ziffer „2“ eingetragen. Dieses Feld weist auf einen entsprechenden Primärschlüssel in einer anderen Tabelle hin. In der Tabelle „Filter“ würde so auf das Mitglied mit der ID-Nummer 2 verwiesen, in der Tabelle „Filter_Vor“ auf den Vorstandsposten mit der Nummer „2“, die Tabelle „Filter_Etiketten“ wird zum Vorfiltern der auszudruckenden Etiketten nach z.B. Abteilungen genutzt.

Ist das Feld „FilterID“ leer, so werden alle Datensätze angezeigt. Dies ist durch eine besondere Abfragetechnik gewährleistet. Erst die Eingabe eines Wertes macht aus dem Feldinhalt einen Filterwert.

Tabellen für Berichte

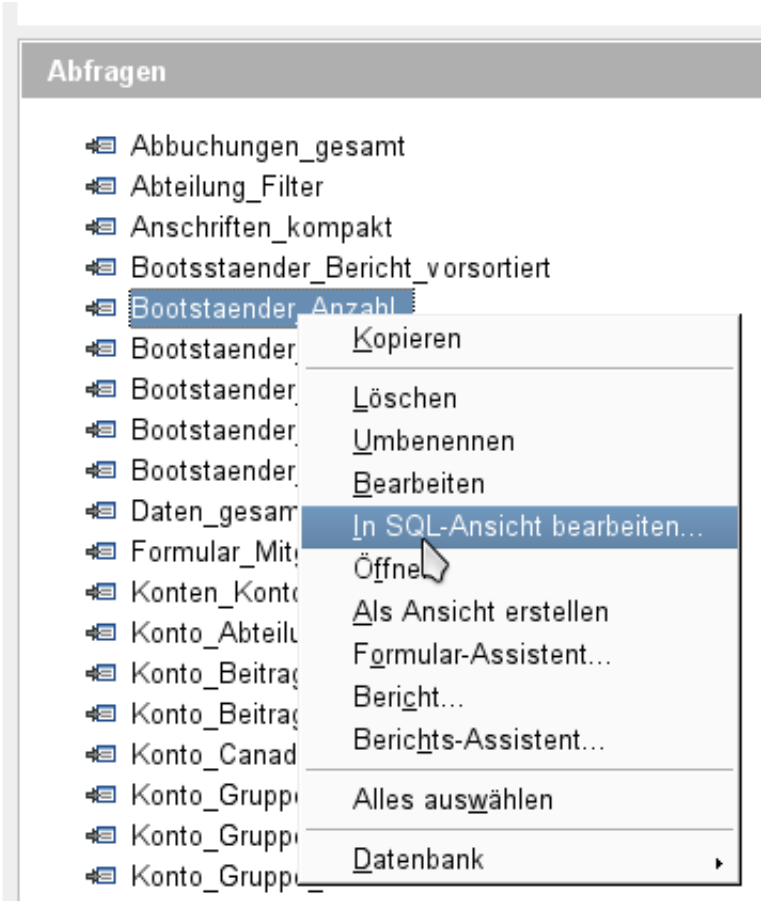
In den Eingabetabellen werden Datensätze naturgemäß untereinander geschrieben. Sollen aber z.B. aus den verschiedenen Eingaben für die Personen einer Familie Datensätze für Adressetiketten erstellt werden, so ist dies nur schwerlich über einfache Abfragen möglich. Schließlich müssen hier vorher in verschiedenen Datensätzen der gleichen Tabelle liegende Werte wie z.B. die Vornamen der einzelnen Familienmitglieder in einem Datensatz hintereinander liegen. Aus der vorherigen Spalte werden sozusagen Gruppen von Zeilen.

Diese Gruppierung zusammen mit bestimmten Zusatzinformationen (aus den Geschlechtern der Personen wird die Anrede, ab 3 Personen „Familie“) wird über das Einlesen der Datensätze in einem Makro und das dortige Zusammenstellen zu einzelnen Datensätzen erreicht. Das Ergebnis dieser Auswertung wird dann in den „tmp_“-Tabellen gespeichert.

Die „tmp_“-Tabellen sind schließlich die Grundlage für die Berichte, die aus den Formularen heraus aufgerufen werden. Nur wenn die Berichte aus den Formularen heraus aufgerufen werden sind sie also mit den neuesten Daten versorgt. Ansonsten lesen sie ihre Inhalte einfach aus den bereits bestehenden Tabellen aus.

Datenbank Verein – Abfragen

Im folgenden wird erst einmal das Grundprinzip einer Abfrage vorgestellt, bevor auf die Abfragen der Datenbank genauer eingegangen wird. Vor einem sei aber am Anfang bereits gewarnt: Das Öffnen einiger Abfragen im Bearbeitungsmodus der grafischen Benutzeroberfläche zumindest unter OOo 3.1.1 kann zum Zerstören der Konstruktion dieser Abfragen führen, wenn die Abfrage anschließend neu abgespeichert wird. Dies liegt wohl daran, dass die Automatik eine bessere Möglichkeit der Zusammenfassung der Inhalte zu erkennen meint als sie vom Autor vorgegeben ist – und im Nachhinein dann registriert, dass es so denn gar nicht mehr geht.



Wird über den Abfragen mit der rechten Maustaste ein Klick ausgeführt, so sind dort 2 verschiedene Bearbeitungsmöglichkeiten zur Auswahl gestellt. Gefahrlos ist die Möglichkeit, die Abfragen zum Bearbeiten in der SQL-Ansicht zu öffnen. Bei manchen Abfragen weist die GUI sogar darauf hin, dass nur in dieser Ansicht geöffnet wird, da die Abfrage nicht mit der grafischen Ansicht bearbeitet werden kann.

Erste Konstruktionen eigener Abfragen sind in der Regel nicht so komplex gehalten, wie es die teilweise später erläuterten Abfragen sind. Hierfür ist der erfolgversprechendste Weg die grafische Benutzeroberfläche. Unterabfragen, wie sie weiter unten vorkommen, werden in der grafischen Benutzeroberfläche so erstellt, dass statt einer zusätzlichen Tabelle einfach eine vorher erstellte Abfrage eingebunden wird.

Grundprinzip einer Abfrage

`SELECT "Tabellenfeld1" FROM "Tabelle"` – jede Abfrage beginnt mit einem „SELECT“, also „wähle aus“ und hat zudem in dem Text stehen, von welcher Tabelle, nämlich „FROM“ Tabelle. Mit diesem einfachen Befehl werden in der Abfrage alle Inhalte aus Tabellenfeld1 untereinander auf dem Bildschirm aufgelistet. In der GUI ist Tabellenfeld1 und Tabelle noch zusätzlich mit Anführungszeichen eingefasst.

`SELECT "Tabellenfeld1","Tabellenfeld2" FROM "Tabelle" ORDER BY "Tabellenfeld1" ASC` – jetzt ist die Abfrage um ein weiteres Tabellenfeld erweitert worden, das zur gleichen Tabelle gehört. Die Tabellenfelder werden durch ein Komma voneinander getrennt. Außerdem wird nach dem ersten Tabellenfeld sortiert (ORDER BY) und zwar aufsteigend (ASC-ascend) im Gegensatz zu absteigend (DESC-descend).

`SELECT "Tabelle1"."Tabellenfeld1","Tabelle2"."Tabellenfeld2" FROM "Tabelle1","Tabelle2" ORDER BY "Tabelle1"."Tabellenfeld1" DESC` – die Abfrage listet auch zwei Spalten auf. Nur wird sie wesentlich mehr Datensätze haben, da die Spalten aus 2 verschiedenen Tabellen kommen. Der Ursprung jedes Feldes ist durch die vorangestellte Tabellenbezeichnung, getrennt mit einem Punkt von dem Feld, gekennzeichnet. Außerdem muss auch noch der Verweis auf beide Tabellen im „FROM"-Bereich stehen. Was der Abfrage aber ganz entschieden gegenüber der vorherigen fehlt sind die klaren Zuordnungen von Tabellenfeld1 zu Tabellenfeld2. In der vorherigen Abfrage mussten beide zum gleichen Datensatz gehören. Es konnte also nicht mehr als die Gesamtzahl aller Datensätze angezeigt werden. Bei der obigen Abfrage aus den beiden Tabellen werden aber mangels Beziehungsdefinition alle Möglichkeiten ausgeschöpft. Wer gern eine Datenbank auf Dauer beschäftigen möchte braucht nur gut gefüllte Tabellen mit solchen Abfragen belästigen, denn jetzt werden alle Kombinationen von Tabelle1 mit Tabelle2 aufgelistet. Bei je 3 Datensätzen gibt es 9 Ergebnisse, bei 5 Datensätze 25 …. Es muss also eine Einschränkung her.

```
SELECT "Tabelle1"."Tabellenfeld1","Tabelle2"."Tabellenfeld2"
FROM "Tabelle1","Tabelle2"
WHERE "Tabelle1"."TabellenfeldX" = "Tabelle2"."TabellenfeldY"
ORDER BY "Tabelle1"."Tabellenfeld1" DESC.
```

Hier ist klar, dass nur die Felder aufgelistet werden, für die die Bedingung gilt, dass die angedeuteten anderen Felder TabellenfeldX und TabellenfeldY einander gleich sind. Von diesen Bedingungen können natürlich mehrere formuliert werden, so dass die Ergebnismenge immer weiter eingeschränkt wird. Daneben gibt es natürlich auch andere Beziehungsdefinitionen wie >, < usw., die in den folgenden Abfragen näher erläutert werden.

Die obige Abfrage zeigt nun nur die Ergebnisse an, bei denen die Bedingung zutrifft. Hat Tabelle2 keinen entsprechenden Datensatz zu bieten, so bleibt die Abfrage leer. Soll dies vermieden werden, so ist die Beziehung zwischen den Tabellen anders zu definieren:

```
SELECT "Tabelle1"."Tabellenfeld1","Tabelle2"."Tabellenfeld2"
FROM "Tabelle1"
LEFT JOIN "Tabelle2" ON "Tabelle1"."TabellenfeldX"="Tabelle2"."TabellenfeldY"
ORDER BY "Tabelle1"."Tabellenfeld1" DESC
```

Von der Tabelle 1 werden mit dem Befehl „LEFT JOIN" alle Datensätze angezeigt („Tabelle1" steht links von dem Begriff „LEFT JOIN"). Die Bedingung für die Verknüpfung mit der Tabelle2 wird jetzt über den Zusatz „ON" erstellt. Aus „Tabelle2" werden gegebenenfalls Zusatzinformationen geliefert. So kann dann auch häfiger in der Ergebnismenge "Tabelle2"."Tabellenfeld2" leer bleiben.

Weitere Details zu den Abfragen sind der Erklärung der einzeln aufgelisteten Abfragen zu entnehmen. Die Abfragen sind nicht nach Komplexität sondern nach dem Alphabet sortiert. Wiederholt vorkommende Ausdrücke werden später nicht noch einmal erklärt. Dennoch empfiehlt es sich für den Einsteiger eventuell, kürzere Abfragen zuerst einmal in Augenschein zu nehmen. Die „Monsterabfrage" `"Beitraege"` sollte nur demonstrieren, dass eine bestimmte Informationssammlung auch direkt in einer Abfrage möglich. In der Praxis bietet es sich an, solche Abfragen in viele übersichtlichere Teile aufzusplitten.

Abfragen der Datenbank Verein im Detail

Abteilung_Filter

```
SELECT "Anzeige", "ID", "Sort"
FROM "Filter_aus" AS "Filter_aus"
UNION ALL
SELECT "Abteilung" AS "Anzeige", "ID", "Abteilung" AS "Sort"
```

```
FROM "Abteilung" AS "Abteilung"
ORDER BY "Sort"
```

Funktion: Beschickung einer Listbox in einem Formular mit den Inhalten aus 2 Tabellen. Der Inhalt der Tabelle „Abteilung“ wird an den Inhalt der Tabelle „Filter_aus“ angehängt.

Beschreibung: Aus der Tabelle „Filter_aus“ werden alle Felder übernommen. Das Feld „Anzeige ist anschließend in der Listbox zu sehen, der Wert für das Feld „ID“ wird an die dem Formular zugrunde liegende Tabelle übergeben. Nach dem Feld „Sort“ werden die Inhalte der Listbox sortiert dargestellt. Die Tabelle „Filter_aus“ übergibt lediglich den Begriff „Alle“ sowie ein leeres „ID“-Feld. Wird die Listbox entsprechend eingestellt, so werden bei der Auswahl „Alle“ die dem Formular zugrunde liegenden Daten nicht mehr auf einen Filterwert beschränkt. Damit „Alle“ immer an erster Stelle der Listbox steht ist im Feld Sort „00000“ eingegeben.
An die Tabelle „Filter_aus“ wird mit „UNION ALL“ die Tabelle „Abteilung“ angehängt. Wichtig bei solchen Verbindungen ist, dass die Felder mit gleichem Namen und gleichem Format versehen sind. Der gleiche Name wird dadurch hergestellt, dass der ursprünglichen Feldbezeichnung mit dem Schlüsselwort „AS“ eine andere Bezeichnung als im Original übergeben wird. Auf diese Art wird bei der Tabelle „Abteilung“ zweimal das Feld „Abteilung“ aufgerufen. Einmal als Darstellungswert in der Listbox, zum anderen als Grundlage für die Sortierung der Listbox.
Die „ORDER BY“-Anweisung enthält keinen ausdrücklichen Hinweis auf die Reihenfolge. Wenn kein Hinweis wie „ASC“ oder „DESC“ existiert, wird automatisch aufsteigend sortiert („ASC“).

Anschriften_kompakt

```
SELECT "Anrede", "Vorname" || ' ' || "Nachname" AS "Name", "Strasse" || ' ' ||
  "Nr" AS "Str_Nr", "Postleitzahl" || ' ' || "Ort" AS "PLZ_Ort"
FROM "tmp_Anschriften"
```

Funktion: Vorformatierung für den Etikettendruck. Die Trennung der Felder führt sonst zu ungewollten Freiräumen zwischen den Feldern, wenn der Etikettendruck, wie hier geplant, mit dem Report-Designer zugänglich sein soll.

Beschreibung: Durch die senkrechten Strichpaare „||“ werden einzelne Begriffe in einer Abfrage miteinander verbunden. Diese Verbindungsmöglichkeit ist nicht bei allen Datenbanken Standard, funktioniert so aber in der Base zugrunde liegenden HSQLDB.
Auch neue Begriffe lassen sich so in eine Abfrage schreiben. Hier wird so z.B. „Vorname“ und „Nachname“ verbunden. Als neuer Begriff wird allerdings dazwischen eine Leerstelle mittels zweier Hochkommata (' ') eingefügt. Dem aus dieser Kombination entstehenden Feld wird über „AS“ die Bezeichnung „Name“ zugewiesen.

Beitraege

```
SELECT "Mitglied_aktuell_Ansicht"."ID", "Mitglied_aktuell_Ansicht"."gruID" AS
  "GruppenNr", "Mitglied_aktuell_Ansicht"."Nachname"||',
  '||"Mitglied_aktuell_Ansicht"."Vorname" AS "Name", "B_Alter"."Alter" AS
  "Beitragsart", "B_Alter"."Beitrag_Alter"/12 AS
  "Monatsbeitrag","B_Alter"."Beitrag_Alter"/4 AS
  "Quartalsbeitrag","B_Alter"."Beitrag_Alter" AS "Jahresbeitrag"
FROM
"Mitglied_aktuell_Ansicht",
( SELECT "Mitglied_aktuell_Ansicht"."ID", "Beitrag"."BeitrArt"||' - aktuelles
```

```
   Alter: '||CASEWHEN( DAYOFYEAR( "GebDat" ) > DAYOFYEAR( NOW( ) ),
   DATEDIFF( 'yy', "GebDat", NOW( ) ) - 1, DATEDIFF( 'yy', "GebDat", NOW( ) ) )
   AS "Alter", "Beitrag"."BeitrHoehe" * 12 / "Beitrag"."BeitrMonate" AS
   "Beitrag_Alter" FROM "Beitrag", "Mitglied_aktuell_Ansicht", "Gruppe" WHERE
   CASEWHEN( DAYOFYEAR( "GebDat" ) > DAYOFYEAR( NOW( ) ), DATEDIFF( 'yy',
   "GebDat", NOW( ) ) - 1, DATEDIFF( 'yy', "GebDat", NOW( ) ) ) BETWEEN
   "Beitrag"."MinAlter" AND "Beitrag"."MaxAlter" AND "Beitrag"."BeitrBezug" =
   'Mit' AND "Mitglied_aktuell_Ansicht"."gruID" = "Gruppe"."ID" AND
   "Gruppe"."ID"  NOT IN ( SELECT "rel_Gru_Bei"."gruID" FROM "rel_Gru_Bei",
   "Beitrag" WHERE "Beitrag"."ID" = "rel_Gru_Bei"."beiID" AND
   "Beitrag"."BeitrBezug" LIKE 'Mit%' ) ) AS "B_Alter"
WHERE "B_Alter"."ID" = "Mitglied_aktuell_Ansicht"."ID"
UNION ALL
SELECT "Mitglied_aktuell_Ansicht"."ID", "Mitglied_aktuell_Ansicht"."gruID" AS
   "GruppenNr", "Mitglied_aktuell_Ansicht"."Nachname"||',
   '||"Mitglied_aktuell_Ansicht"."Vorname" AS "Name", 'Abteilungbeitrag
   '||"B_Abteilung"."Abteilung" AS "Beitragsart",
   "B_Abteilung"."EinzelAbteilung"/12 AS
   "Monatsbeitrag","B_Abteilung"."EinzelAbteilung"/4 AS
   "Quartalsbeitrag","B_Abteilung"."EinzelAbteilung" AS "Jahresbeitrag"
FROM
"Mitglied_aktuell_Ansicht" ,
( SELECT "Mitglied_aktuell_Ansicht"."ID", "Abteilung"."Abteilung",
   "Beitrag"."BeitrHoehe" * 12 / "Beitrag"."BeitrMonate" AS "EinzelAbteilung"
   FROM "rel_Mit_Abt", "Abteilung", "Mitglied_aktuell_Ansicht", "Beitrag",
   "Gruppe"  WHERE "rel_Mit_Abt"."abtID" = "Abteilung"."ID" AND
   "rel_Mit_Abt"."mitID" = "Mitglied_aktuell_Ansicht"."ID" AND "Beitrag"."ID" =
   "Abteilung"."beiID" AND CASEWHEN( DAYOFYEAR( "GebDat" ) >
   DAYOFYEAR( NOW( ) ), DATEDIFF( 'yy', "GebDat", NOW( ) ) - 1, DATEDIFF( 'yy',
   "GebDat", NOW( ) ) ) BETWEEN "Beitrag"."MinAlter" AND "Beitrag"."MaxAlter"
   AND "Mitglied_aktuell_Ansicht"."gruID" = "Gruppe"."ID"  AND "Gruppe"."ID"
   NOT IN ( SELECT "rel_Gru_Bei"."gruID" FROM "rel_Gru_Bei", "Beitrag",
   "Abteilung" WHERE "Beitrag"."ID" = "rel_Gru_Bei"."beiID" AND ("Beitrag"."ID"
   = "Abteilung"."beiID" OR "Beitrag"."BeitrBezug" IN (SELECT
   "Beitrag"."BeitrBezug" FROM "Beitrag","Abteilung" WHERE "Beitrag"."ID" =
   "Abteilung"."beiID") ) ) ) AS "B_Abteilung"
WHERE "Mitglied_aktuell_Ansicht"."ID" = "B_Abteilung"."ID"
UNION ALL
SELECT "Mitglied_aktuell_Ansicht"."ID", "Mitglied_aktuell_Ansicht"."gruID" AS
   "GruppenNr", "Mitglied_aktuell_Ansicht"."Nachname"||',
   '||"Mitglied_aktuell_Ansicht"."Vorname" AS "Name", 'Bootständer für
   '||"B_Boot"."Bootsart" AS "Beitragsart", "B_Boot"."EinzelBoot"/12 AS
   "Monatsbeitrag","B_Boot"."EinzelBoot"/4 AS
   "Quartalsbeitrag","B_Boot"."EinzelBoot" AS "Jahresbeitrag"
FROM
"Mitglied_aktuell_Ansicht" ,
( SELECT "Mitglied_aktuell_Ansicht"."ID", "Boot_Art"."Bootsart",
   "Beitrag"."BeitrHoehe" * 12 / "Beitrag"."BeitrMonate" AS "EinzelBoot" FROM
   "Boot", "Mitglied_aktuell_Ansicht", "Boot_Art", "Beitrag", "Gruppe" WHERE
   "Boot"."mitID" = "Mitglied_aktuell_Ansicht"."ID" AND "Boot"."bo_aID" =
   "Boot_Art"."ID" AND "Beitrag"."ID" = "Boot_Art"."beiID" AND
   "Mitglied_aktuell_Ansicht"."gruID" = "Gruppe"."ID"  AND "Gruppe"."ID" NOT IN
   ( SELECT "rel_Gru_Bei"."gruID" FROM "rel_Gru_Bei", "Beitrag", "Boot_Art"
   WHERE "Beitrag"."ID" = "rel_Gru_Bei"."beiID" AND "Beitrag"."ID" =
   "Boot_Art"."beiID" )) AS "B_Boot"
WHERE "B_Boot"."ID" = "Mitglied_aktuell_Ansicht"."ID"
UNION ALL
SELECT "Mitglied_aktuell_Ansicht"."ID", "Mitglied_aktuell_Ansicht"."gruID" AS
   "GruppenNr", "Mitglied_aktuell_Ansicht"."Nachname"||',
   '||"Mitglied_aktuell_Ansicht"."Vorname" AS "Name", 'Zusatzbetrag für
   '||"B_Boot_Z"."AnzahlBoote"||' Boote' AS "Beitragsart",
```

```
   "B_Boot_Z"."ZusatzBoote"/12 AS "Monatsbeitrag","B_Boot_Z"."ZusatzBoote"/4 AS
   "Quartalsbeitrag","B_Boot_Z"."ZusatzBoote" AS "Jahresbeitrag"
FROM
"Mitglied_aktuell_Ansicht" ,
( SELECT "Mitglied_aktuell_Ansicht"."ID", COUNT( "Mitglied_aktuell_Ansicht"."ID"
   ) AS "AnzahlBoote", ( COUNT( "Mitglied_aktuell_Ansicht"."ID" ) *
   COUNT( "Mitglied_aktuell_Ansicht"."ID" ) -
   COUNT( "Mitglied_aktuell_Ansicht"."ID" ) ) * ( SELECT "Beitrag"."BeitrHoehe"
   * 12 / "Beitrag"."BeitrMonate" FROM "Beitrag" WHERE "BeitrBezug" = 'BooZ' ) /
   2 AS "ZusatzBoote" FROM "Boot", "Mitglied_aktuell_Ansicht", "Boot_Art",
   "Beitrag", "Gruppe" WHERE "Boot"."mitID" = "Mitglied_aktuell_Ansicht"."ID"
   AND "Boot"."bo_aID" = "Boot_Art"."ID" AND "Beitrag"."ID" = "Boot_Art"."beiID"
   AND "Mitglied_aktuell_Ansicht"."gruID" = "Gruppe"."ID" AND "Gruppe"."ID" NOT
   IN ( SELECT "rel_Gru_Bei"."gruID" FROM "rel_Gru_Bei", "Beitrag" WHERE
   "Beitrag"."ID" = "rel_Gru_Bei"."beiID" AND "Beitrag"."BeitrBezug" = 'BooZ' )
   GROUP BY "Mitglied_aktuell_Ansicht"."ID" ) AS "B_Boot_Z"
WHERE "B_Boot_Z"."ID" = "Mitglied_aktuell_Ansicht"."ID"
UNION ALL
SELECT "Mitglied_aktuell_Ansicht"."ID", "Mitglied_aktuell_Ansicht"."gruID" AS
   "GruppenNr", 'Gruppe ' || "Mitglied_aktuell_Ansicht"."Nachname" AS "Name",
   "rel_Gru_Bei"."Anzahl"||' * '||"Beitrag"."BeitrArt" AS "Beitragsart",
   "rel_Gru_Bei"."Anzahl" * "Beitrag"."BeitrHoehe" / "Beitrag"."BeitrMonate" AS
   "Monatsbeitrag","rel_Gru_Bei"."Anzahl" * "Beitrag"."BeitrHoehe"*3 /
   "Beitrag"."BeitrMonate" AS "Quartalsbeitrag","rel_Gru_Bei"."Anzahl" *
   "Beitrag"."BeitrHoehe"*12 / "Beitrag"."BeitrMonate" AS "Jahresbeitrag" FROM
   "rel_Gru_Bei", "Beitrag", "Mitglied_aktuell_Ansicht", "Gruppe" WHERE
   "rel_Gru_Bei"."beiID" = "Beitrag"."ID" AND "rel_Gru_Bei"."gruID" =
   "Gruppe"."ID" AND "Mitglied_aktuell_Ansicht"."gruID" = "Gruppe"."ID" AND
   "Mitglied_aktuell_Ansicht"."ID" IN (SELECT
   MIN( "Mitglied_aktuell_Ansicht"."ID" ) FROM "Mitglied_aktuell_Ansicht" GROUP
   BY "Mitglied_aktuell_Ansicht"."gruID")
ORDER BY "GruppenNr", "Name", "Beitragsart"
```

Funktion: Der Beitrag für jedes Mitglied wird anhand der bisher eingegebenen Daten ermittelt. Die Beitragsbearbeitung bei Einzelpersonen ist so überflüssig. Erst bei Gruppen wird aufgrund von bestimmten Ermäßigungen z.B. für Familien eine zusätzliche Eingabe dieser Ermäßigungen erforderlich sein, die diese Abfrage ebenfalls berücksichtigt.

Beschreibung: Die Abfrage ist natürlich erst einmal erschreckend groß geraten: Prinzipiell könnte sie in viele Unterabfragen gesplittet werden, was die Übersichtlichkeit im Abfragenverzeichnis senken würde. Base erstellt anschließend intern aus den einzelnen Abfragen genau so eine Abfrage wie oben zusammengestellt wurde. Grundsätzlich ist diese Abfrage nur in reinem SQL-Modus möglich. Das liegt vor allem an den teilweise verwendeten Formeln. Da Abfragen im SQL-Modus in einem Formular nicht gefiltert werden können wurde auf Basis dieser Abfrage anschließend eine Ansicht (View) erstellt. Hier steht die Filtermöglichkeit wieder zur Verfügung.

Grundsätzlich sind hier mehrere Abfragen durch `UNION ALL` miteinander verbunden worden.

```
SELECT "Mitglied_aktuell_Ansicht"."ID",
"Mitglied_aktuell_Ansicht"."gruID" AS "GruppenNr",
"Mitglied_aktuell_Ansicht"."Nachname"||',
'||"Mitglied_aktuell_Ansicht"."Vorname" AS "Name",
"B_Alter"."Alter" AS "Beitragsart", "B_Alter"."Beitrag_Alter"/12 AS
"Monatsbeitrag","B_Alter"."Beitrag_Alter"/4 AS
"Quartalsbeitrag","B_Alter"."Beitrag_Alter" AS "Jahresbeitrag"
```

Die Struktur für die gesamte Abfage wird gesetzt. Das 2. Feld wird in „GruppenNr“ umbenannt, damit zum Schluu danach sortiert werden kann. Das 4. Feld erhält die

Bezeichnung „Name“ zugewiesen, ebenfalls aus Sortiergründen, aus dem 5. Feld wird die „Beitragsart“ usw. Da der Beitrag erst einmal als Jahresbeitrag berechnet wird, muss für den Monatsbeitrag und den Quartalsbeitrag hier eine Neuberechnung stattfinden. Der Feldinhalt dieses Feldes wird also durch 12 bzw. 4 geteilt, damit Monats- und Quartalsbeitrag ermittelt werden.

Teilabfrage 1:

```
FROM
"Mitglied_aktuell_Ansicht",
( SELECT "Mitglied_aktuell_Ansicht"."ID", "Beitrag"."BeitrArt"||' -
aktuelles Alter: '||CASEWHEN( DAYOFYEAR( "GebDat" ) >
DAYOFYEAR( NOW( ) ), DATEDIFF( 'yy', "GebDat", NOW( ) ) - 1,
DATEDIFF( 'yy', "GebDat", NOW( ) ) ) AS "Alter",
"Beitrag"."BeitrHoehe" * 12 / "Beitrag"."BeitrMonate" AS
"Beitrag_Alter" FROM "Beitrag", "Mitglied_aktuell_Ansicht",
"Gruppe" WHERE CASEWHEN( DAYOFYEAR( "GebDat" ) >
DAYOFYEAR( NOW( ) ), DATEDIFF( 'yy', "GebDat", NOW( ) ) - 1,
DATEDIFF( 'yy', "GebDat", NOW( ) ) ) BETWEEN "Beitrag"."MinAlter"
AND "Beitrag"."MaxAlter" AND "Beitrag"."BeitrBezug" = 'Mit' AND
"Mitglied_aktuell_Ansicht"."gruID" = "Gruppe"."ID" AND
"Gruppe"."ID"  NOT IN ( SELECT "rel_Gru_Bei"."gruID" FROM
"rel_Gru_Bei", "Beitrag" WHERE "Beitrag"."ID" =
"rel_Gru_Bei"."beiID" AND "Beitrag"."BeitrBezug" LIKE 'Mit%' ) ) AS
"B_Alter"
WHERE "B_Alter"."ID" = "Mitglied_aktuell_Ansicht"."ID"
```

Die Daten für die Abfrage werden aus einer Tabelle und einer Unterabfrage ausgelesen, der mit AS „B_Alter“ die Bezeichnung „B_Alter“ zugewiesen wird. Das Feld „Alter“ der Abfrage „B_Alter“ setzt sich zusammen aus der Beitragsart sowie dem Verweis auf das aktuelle Alter der Person. Die Altersbestimmung wird im folgenden erklärt:

`DATEDIFF( 'yy', "GebDat", NOW( ) )` ergibt den Unterschied zwischen dem Datum aus „GebDat“ und dem heutigen Datum. Allerdings werden, zumindest bei der HSQLDB, auf diese Art und Weise nur die Jahreszahlen ('yy') verglichen. Liegt das Geburtsdatum im Vorjahr, so erscheint auf diese Weise auf jeden Fall eine „1“ bei der Berechnung – auch wenn heute gerade der 1.1. ist und die Person am 31.12. geboren wurde.

Daraus ergibt sich ein etwas umfassenderer Aufbau der Berechnung für das momentane Alter:

`CASEWHEN( DAYOFYEAR( "GebDat" ) > DAYOFYEAR( NOW( ) ), DATEDIFF( 'yy', "GebDat", NOW( ) ) - 1, DATEDIFF( 'yy', "GebDat", NOW( ) ) )` für den Fall, dass der Tag, gezählt als Tag im Jahr, von dem Geburtsdatum größer ist als der Tag heute, soll von der Jahresdifferenz 1 abgezogen werden. Ansonsten ist die Jahresdifferenz in das Feld zu schreiben.

Im nächsten Feld wird dann der Beitrag ermittelt. Da in der Tabelle Beitrag die Höhe des Beitrags sowie die Laufzeit für diesen Beitrag in Monaten angegeben ist wird der Betrag für ein Jahr ermittelt, indem der einzelne Betrag mit 12 multipliziert wird und anschließend durch die Monatsanzahl dividiert wird. Der Schritt über den Jahresbeitrag ist eventuell bei Rundungsfehlern der sicherere Weg, so dass schließlich innerhalb der Hauptabfrage wieder vom Jahresbeitrag auf den Monatsbeitrag heruntergerechnet wird.

`BETWEEN "Beitrag"."MinAlter" AND "Beitrag"."MaxAlter"` Das Alter ist ausschlaggebend für die Beitragshöhe. Also wird hier das ermittelt, ob das Alter zwischen dem Mindestalter und dem Maximalalter liegt. Vorher musste in der WHERE-Bedingung wieder das Alter ermittelt werden.

`AND "Beitrag"."BeitrBezug" = 'Mit'` nur die Datensätze werden berücksichtigt, in denen das Feld „BeitrBezug“ den genauen Wert 'Mit' hat. Hier

handelt es sich um die Bezeichnung für die altersabhängigen Mitgliedsbeiträge.

```
AND "Gruppe"."ID"  NOT IN ( SELECT "rel_Gru_Bei"."gruID" FROM
"rel_Gru_Bei", "Beitrag" WHERE "Beitrag"."ID" =
"rel_Gru_Bei"."beiID" AND "Beitrag"."BeitrBezug" LIKE 'Mit%' ) )
```

Die Datensätze sollen nur angezeigt werden, wenn nicht in der Tabelle `"rel_Gru_Bei"` ein Eintrag existiert, der sich auf die Gruppe bezieht und aus dem Bereich `'Mit'` stammt. Mit `LIKE 'Mit%'` ist hier erweitert worden auf alle Datensätze, die im `"BeitrBezug"` mit dem Wort `'Mit'` beginnen.
Wird also z.B. festgestellt, dass der Beitrag für mehrere Personen zusammen günstiger ist, so wird dies z.B. als Familienbeitrag in die Tabelle „rel_Gru_Bei“ über ein Formular eingetragen. Die Berechnung des Beitrages mit der obigen Abfrage wird dann einfach ausgesetzt.

```
WHERE "B_Alter"."ID" = "Mitglied_aktuell_Ansicht"."ID"
```

Das Feld "ID" aus der Tabelle "B_Alter" ist gleich dem Feld "ID" aus der Tabelle "Mitglied_aktuell_Ansicht".

Teilabfrage 2: Durch UNION ALL wird jetzt an die Teilabfrage 1 eine weitere Abfrage angehängt.

```
SELECT "Mitglied_aktuell_Ansicht"."ID",
"Mitglied_aktuell_Ansicht"."gruID" AS "GruppenNr",
"Mitglied_aktuell_Ansicht"."Nachname"||',
'||"Mitglied_aktuell_Ansicht"."Vorname" AS "Name",
'Abteilungbeitrag '||"B_Abteilung"."Abteilung" AS "Beitragsart",
"B_Abteilung"."EinzelAbteilung"/12 AS
"Monatsbeitrag","B_Abteilung"."EinzelAbteilung"/4 AS
"Quartalsbeitrag","B_Abteilung"."EinzelAbteilung" AS
"Jahresbeitrag"
FROM
"Mitglied_aktuell_Ansicht" ,
( SELECT "Mitglied_aktuell_Ansicht"."ID", "Abteilung"."Abteilung",
"Beitrag"."BeitrHoehe" * 12 / "Beitrag"."BeitrMonate" AS
"EinzelAbteilung" FROM "rel_Mit_Abt", "Abteilung",
"Mitglied_aktuell_Ansicht", "Beitrag", "Gruppe"  WHERE
"rel_Mit_Abt"."abtID" = "Abteilung"."ID" AND "rel_Mit_Abt"."mitID"
= "Mitglied_aktuell_Ansicht"."ID" AND "Beitrag"."ID" =
"Abteilung"."beiID" AND CASEWHEN( DAYOFYEAR( "GebDat" ) >
DAYOFYEAR( NOW( ) ), DATEDIFF( 'yy', "GebDat", NOW( ) ) - 1,
DATEDIFF( 'yy', "GebDat", NOW( ) ) ) BETWEEN "Beitrag"."MinAlter"
AND "Beitrag"."MaxAlter" AND "Mitglied_aktuell_Ansicht"."gruID" =
"Gruppe"."ID"  AND "Gruppe"."ID"  NOT IN ( SELECT
"rel_Gru_Bei"."gruID" FROM "rel_Gru_Bei", "Beitrag", "Abteilung"
WHERE "Beitrag"."ID" = "rel_Gru_Bei"."beiID" AND ("Beitrag"."ID" =
"Abteilung"."beiID" OR "Beitrag"."BeitrBezug" IN (SELECT
"Beitrag"."BeitrBezug" FROM "Beitrag","Abteilung" WHERE
"Beitrag"."ID" = "Abteilung"."beiID") ) ) ) AS "B_Abteilung"
WHERE "Mitglied_aktuell_Ansicht"."ID" = "B_Abteilung"."ID"
```

Die eigentliche SELECT-Anweisung birgt gegenüber der Anweisung in der vorherigen Teilabfrage nichts neues.
Auch hier liegt neben der Tabelle „Mitglied“ wieder eine Unterabfrage, diesmal unter der Bezeichnung „B_Abteilung“, vor. Die ID aus der Unterabfrage ist wieder gleich der ID aus der Tabelle „Mitglied“.
Der Abteilungsbeitrag kann eventuell erst ab einem bestimmten Alter erhoben werden. Deshalb enthält die Unterabfrage auch wieder eine Altersbestimmung mit entsprechendem Vergleich.
Da auch der Abteilungsbeitrag im Formular durch eine gesonderte Eingabe überschrieben werden kann gibt die Abfrage nur dann einen Datensatz wieder, wenn für die entsprechende Abteilung und die entsprechende Gruppe keine Eingabe über das Formular in „rel_Gru_Bei“ erstellt wurde. Die Kontrolle hierzu erfolgt über das

Feld „BeitrBezug“ in der „Beitrag“-Tabelle.

Teilabfrage 3: Hier werden die Gebühren für die Bootständer ermittelt. Diese Teilabfrage enthält nur die Gebühren, die sich direkt aus den Bootsarten ergeben. Für jede Bootsart wird die Gebühr mit dem entsprechenden Mitglied verknüpft.
Ansonsten birgt die Teilabfrage gegenüber den anderen Teilabfragen nichts neues.

Teilabfrage 4:
```
SELECT "Mitglied_aktuell_Ansicht"."ID",
"Mitglied_aktuell_Ansicht"."gruID" AS "GruppenNr",
"Mitglied_aktuell_Ansicht"."Nachname"||',
'||"Mitglied_aktuell_Ansicht"."Vorname" AS "Name", 'Zusatzbetrag
für '||"B_Boot_Z"."AnzahlBoote"||' Boote' AS "Beitragsart",
"B_Boot_Z"."ZusatzBoote"/12 AS
"Monatsbeitrag","B_Boot_Z"."ZusatzBoote"/4 AS
"Quartalsbeitrag","B_Boot_Z"."ZusatzBoote" AS "Jahresbeitrag"
FROM
"Mitglied_aktuell_Ansicht" ,
( SELECT "Mitglied_aktuell_Ansicht"."ID",
COUNT( "Mitglied_aktuell_Ansicht"."ID" ) AS "AnzahlBoote", ( COUNT(
"Mitglied_aktuell_Ansicht"."ID" ) *
COUNT( "Mitglied_aktuell_Ansicht"."ID" ) -
COUNT( "Mitglied_aktuell_Ansicht"."ID" ) ) * ( SELECT
"Beitrag"."BeitrHoehe" * 12 / "Beitrag"."BeitrMonate" FROM
"Beitrag" WHERE "BeitrBezug" = 'BooZ' ) / 2 AS "ZusatzBoote" FROM
"Boot", "Mitglied_aktuell_Ansicht", "Boot_Art", "Beitrag", "Gruppe"
WHERE "Boot"."mitID" = "Mitglied_aktuell_Ansicht"."ID" AND
"Boot"."bo_aID" = "Boot_Art"."ID" AND "Beitrag"."ID" =
"Boot_Art"."beiID" AND "Mitglied_aktuell_Ansicht"."gruID" =
"Gruppe"."ID" AND "Gruppe"."ID" NOT IN ( SELECT
"rel_Gru_Bei"."gruID" FROM "rel_Gru_Bei", "Beitrag" WHERE
"Beitrag"."ID" = "rel_Gru_Bei"."beiID" AND "Beitrag"."BeitrBezug" =
'BooZ' ) GROUP BY "Mitglied_aktuell_Ansicht"."ID" ) AS "B_Boot_Z"
WHERE "B_Boot_Z"."ID" = "Mitglied_aktuell_Ansicht"."ID"
```

Die Unterabfrage zeigt die Anwendung einer weiteren Funktion. Hier wird zusammengezählt, wie viele Werte die jeweiligen gruppierten Felder enthalten. Es wird gruppiert nach dem Feld `"Mitglied"."ID"` und dann gezählt, wie viele Boots-Datensätze für das Mitglied vorhanden sind, da von der Anzahl der Boote die Zusatzgebühr für Bootständer abhängt:

```
( COUNT( "Mitglied_aktuell_Ansicht"."ID" ) *
COUNT( "Mitglied_aktuell_Ansicht"."ID" ) -
COUNT( "Mitglied_aktuell_Ansicht"."ID" ) ) * ( SELECT
"Beitrag"."BeitrHoehe" * 12 / "Beitrag"."BeitrMonate" FROM
"Beitrag" WHERE "BeitrBezug" = 'BooZ' ) / 2 AS "ZusatzBoote"
```

Die in der Unterabfrage liegende Abfrage darf nur einen Datensatz mit genau einem Wert enthalten, da sie an einer Stelle steht, an der nur ein Wert zu präsentieren ist. Dies ist bei dem Beitrag mit dem Bezug 'BooZ' der Fall. Alternativ hätte hier ja auch direkt der Zusatzbeitrag in Höhe von 0,60 € stehen können. Nur müsste dann bei jeder Änderung die Abfrage nach diesem Wert durchsucht werden. So wird hingegen die Änderung aus der dafür zentralen Tabelle direkt übernommen.
Die Formel zur Berechnung für den Zusatzbeitrag ergibt sich aus der Beitragskonstruktion: Ab dem 2. Boot wird für jedes zusätzliche Boot ein steigender Zusatzbeitrag erhoben, da nur begrenzt Bootständer vorhanden sind. Für das 2. Boot also ein Zusatzbeitrag, für das 3. Boot 2 Zusatzbeiträge und für das 4. Boot 3 Zusatzbeiträge usw. Die Zusatzbeiträge könnten also auch entsprechend der Zahl der Datensätze aufsummiert werden, aber es fehlt eben in den Zeilen eine Zahl, die die bis zu dem Datensatz ausgelesenen Datensätze repräsentiert. Mit einem Makro eine einfache Sache, aber schließlich muss sich das ja auch von der Gesamtzahl der Boote aus ermitteln lassen. Bei Funktionen mit beständiger Steigung muss in der

Funktion mindestens ein Quadrat von x liegen. Uns so ist es auch hier: (x^2-x)*Zusatzbeitrag ergibt den entsprechenden Gesamtzusatzbeitrag.
Auch dieser ermittelte Zusatzbeitrag wird natürlich nur ausgelesen, wenn nicht in der Beitragskorrektur etwas anderes zum Zusatzbeitrag steht. So könnte es ja sein, dass die einer Person zugeschriebenen Boote auf die Familienmitglieder zu verteilen sind und kein Zusatzbeitrag zu zahlen ist.

Teilabfrage 5: In der 5. Teilabfrage wird jetzt zusätzlich zu der Abfrage des altersabhängigen Beitrags, des Beitrag für Abteilungen und des Beitrags für die Bootslagerung die Tabelle ausgelesen, die für die Korrektur dieser automatisch erstellten Beitragsermittlung verantwortlich ist: rel_Gru_Bei. Aller hier eingetragenen Werte werden über UNION ALL an die vorangegangenen Werte angehängt.
Im Anschluss wird die Abfrage noch sortiert nach „GruppenNr“ (Gruppenidentifikationsnummer), „Name“ und „Beitragsart“, so dass schließlich gar nicht mehr zu erkennen ist, dass die gesamte Abfrage eigentlich aus vielen verschiedenen Unterabfragen erstellt wurde.
Besonderes Kennzeichen der Sortierung: eine Sortierung noch „Mitglied_aktuell_Ansicht“.“gruID“ ist nicht so ohne weiteres möglich, da in der gesamten Abfrage, so auch direkt vor der letzten Sortieranweisung, genau nach diesem Feld gruppiert wurde. Gruppierungen haben ihre eigene Sortierung und schließen eine anschließende gesonderte Sortierung aus.

Beitraege_Bericht

```
SELECT "Bankverbindung"."Konto", "Bankverbindung"."BLZ",
   "Bankverbindung"."Bank", "Bankverbindung"."Kontoinhaber",
   "Beitraege_Ansicht"."Beitragsart", "Beitraege_Ansicht"."Monatsbeitrag",
   "Beitraege_Ansicht"."Quartalsbeitrag", "Beitraege_Ansicht"."Jahresbeitrag"
FROM ( SELECT "Konto"."ID", "Konto"."Konto", "Bank"."BLZ", "Bank"."Bank",
   CASEWHEN( "Konto"."mitID" > - 1, "Mitglied"."Nachname" || ', ' ||
   "Mitglied"."Vorname", "Konto"."Nachname" || ', ' || "Konto"."Vorname" ) AS
   "Kontoinhaber" FROM "Konto" LEFT JOIN "Bank" ON "Konto"."banID" = "Bank"."ID"
   LEFT JOIN "Mitglied" ON "Konto"."mitID" = "Mitglied"."ID" ) AS
   "Bankverbindung", "Beitraege_Ansicht", "Gruppe"
WHERE "Bankverbindung"."ID" = "Gruppe"."konID" AND "Gruppe"."ID" =
   "Beitraege_Ansicht"."GruppenNr"
ORDER BY "Bankverbindung"."Kontoinhaber" ASC, "Beitraege_Ansicht"."Beitragsart"
   ASC
```

Funktion: Die Beiträge für Mitgliedschaft, Abteilungen und Bootslagerung sollen zusammen mit den Kontoinformationen in Berichtsform übersichtlich dargestellt werden.

Beschreibung: Die Tabelle „Beitraege_Ansicht“ ist lediglich die Ansicht aus der Abfrage, die oben dargestellt wurde. Die Tabelle „Bankverbindung“ wird wieder durch eine Unterabfrage erstellt.
Diese Unterabfrage birgt dann auch noch eine zusätzliche Funktion aus dem Bereich der Funktionen der HSQLDB, die auch mit Base-OOo 3.1.1 lauffähig sind. Hier wird eine Fallunterscheidung eingebaut, da es sowohl möglich ist, dass ein Mitglied selbst Kontoinhaber ist, vielleicht aber auch ein anderes Mitglied als Kontoinhaber fungiert (Eltern?) oder die Person des Kontoinhabers separat mittels Vorname und Nachname erfasst wurde, da sie gar nicht Mitglied im Verein ist (häufig bei Jugendlichen des Fall).
`CASEWHEN( "Konto"."mitID" > - 1,` Falls der Fall eintritt, dass in der Tabelle „Konto“ das Feld „mitID“ mit einer Nummer größer als – 1 versehen ist (kleinste Mitgliedsnummer ist 0!), dann:
`"Mitglied"."Nachname" || ', ' || "Mitglied"."Vorname",` lese aus

der Tabelle Mitglied Nachname und Vorname aus und zeige sie, durch ein Komma getrennt, an, ansonsten
`"Konto"."Nachname" || ', ' || "Konto"."Vorname"` lese aus der Tabelle Konto Nachname und Vorname aus und zeige diese an
`) AS "Kontoinhaber"` und all dies unter der Bezeichnung „Kontoinhaber“.
Die Verknüpfung von der Tabelle Beitraege_Ansicht mit der Tabelle Bankverbindung erfolgt über
`"Bankverbindung"."ID" = "Gruppe"."konID" AND "Gruppe"."ID" = "Beitraege_Ansicht"."gruID"` , denn die Verbindung vom Konto zu den Mitgliedern erfolgt über die Gruppenzuordnung.

Beitraege_Summe

```
SELECT "GruppenNr", SUM( "Monatsbeitrag" ) AS "Monat_ges",
  SUM( "Quartalsbeitrag" ) AS "Quartal_ges", SUM( "Jahresbeitrag" ) AS
  "Jahr_ges"
FROM "Beitraege_Ansicht"
GROUP BY "GruppenNr"
```

Funktion: Aus den bisher einzeln aufgelisteten Beiträgen soll berechnet werden, wie hoch der Gesamtbeitrag ist, der von jedem Konto einzuziehen ist.

Beschreibung: Die Abfrage liest ihre Werte aus der Ansicht der Abfrage „Beitraege“. Die Berechnungen werden gruppiert nach dem Primärschlüssel der Tabelle Konto („konID“) durchgeführt. Das bedeutet, dass für jedes Konto alle Beträge im Feld Monatsbeitrag zusammengezählt aufgeführt werden. Gleiches gilt für den Quartalsbeitrag und den Jahresbeitrag.

Bootstaender_Anzahl

```
SELECT COUNT( * ) AS "Vorhanden", ( SELECT COUNT( * ) FROM "Boot" ) AS "Belegt"
  FROM "Boot_Staend"
```

Funktion: Die Anzahl aller Bootsständer sowie die der belegten Bootsständer werden für eine Anzeige im Formular gezählt

Beschreibung: COUNT bedeutet, dass die Anzahl der betroffenen Datensätze ausgegeben werden. Das erste Feld zählt dabei alle Datensätze aus der Tabelle „Boot:Staend“, während das 2. Feld durch eine Abfrage erstellt wird, die alle Datensätze aus der Tabelle „Boot“ zählt (in der die vergebenen Bootständer enthalten sind). Abfragen, die innerhalb einer Abfrage ein Feld bestücken, dürfen nur einen eindeutigen Wert ausgeben. Das ist beim Zählen der Werte der Fall, denn die Abfrage gibt genau eine Zahl zurück.
Mit „AS“ wird den beiden Feldern wieder ein Begriff zugeordnet, unter dem sie angezeigt werden.

Bootstaender_belegt

```
SELECT "Boot_Halle"."Halle_lang" || ' - ' || "Boot_Staend"."BootStaend" AS
  "Bootsstaender", "Boot_Staend"."ID"
FROM "Boot" AS "Boot", "Boot_Staend" AS "Boot_Staend", "Boot_Halle" AS
  "Boot_Halle"
WHERE "Boot"."bo_sID" = "Boot_Staend"."ID"
AND "Boot_Staend"."bo_hID" = "Boot_Halle"."ID"
```

Funktion: Die belegten Bootständer werden mit der genauen Ortsbezeichnung in einer Liste erfasst.

Beschreibung: Die Abfrage stellt die übliche Kombination für Listboxen dar: zuerst ein Begriff, der angezeigt wird, dann die dazugehörige Nummer des Primärschlüssels. Der Begriff wird wieder mit Hilfe der „||“ zu einem Begriff zusammengefasst. Die

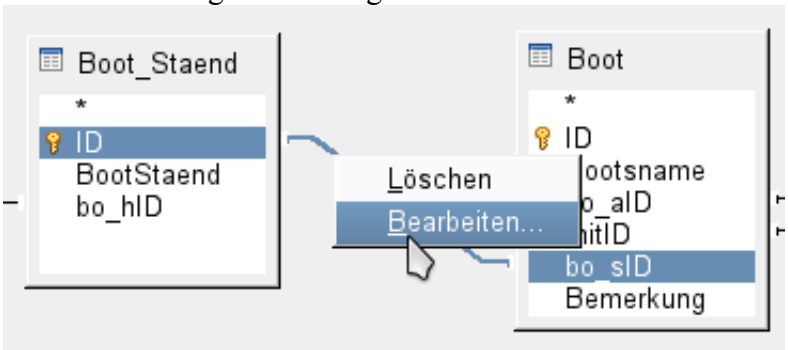
Tabelle „Boot“ ist bei den abgefragten Feldern nicht sichtbar, beschränkt aber das Ergebnis der Abfrage. Deshalb muss sie auch in der „FROM“-Liste verzeichnet sein. Dass hier, wie vorher schon üblich, jede Tabelle mit einem „AS“ wieder seinen eigenen Begriff zugewiesen bekommt liegt einfach an der Funktionsweise der grafischen Benutzeroberfläche unter OOo 3.1.1 und ist in diesem Zusammenhang eigentlich unnötig.
Die nach WHERE aufgelisteten Zusammenhänge zwischen den Tabellen sorgen dafür, dass nur die Datensätze angezeigt werden, die in allen 3 Tabellen verzeichnet sind, hier durch die Konstruktion der Tabellen also die Datensätze, die eine Entsprechung in der Tabelle „Boot“ haben. Schließlich kann ein Boot nur einen Bootständer zugewiesen bekommen haben, der als Bootständer einer Halle verzeichnet wurde.

Bootstaender_Bericht_einfach

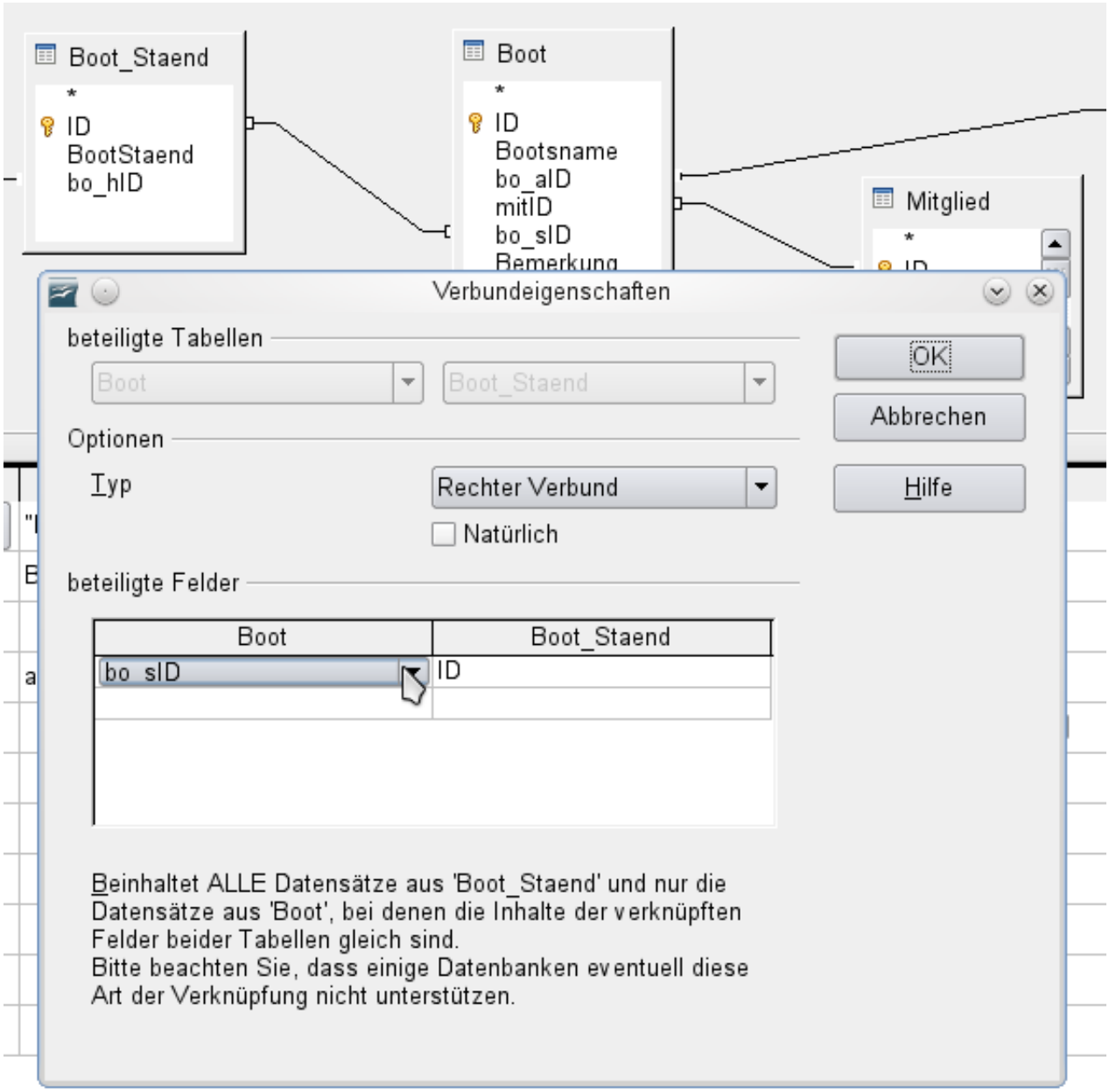
```
SELECT "Boot_Halle"."Halle_lang", "Boot_Halle"."Halle_kurz" || '-' ||
   "Boot_Staend"."BootStaend" AS "Bezeichnung", "Boot_Art"."Bootsart",
   "Boot"."Bootsname", "Mitglied"."Vorname" || ' ' || "Mitglied"."Nachname" AS
   "Eigentümer"
FROM { OJ "Boot" AS "Boot"
RIGHT OUTER JOIN "Boot_Staend" AS "Boot_Staend" ON "Boot"."bo_sID" =
   "Boot_Staend"."ID"
RIGHT OUTER JOIN "Boot_Halle" AS "Boot_Halle" ON "Boot_Staend"."bo_hID" =
   "Boot_Halle"."ID"
LEFT OUTER JOIN "Mitglied" AS "Mitglied" ON "Boot"."mitID" = "Mitglied"."ID"
LEFT OUTER JOIN "Boot_Art" AS "Boot_Art" ON "Boot"."bo_aID" = "Boot_Art"."ID" }
ORDER BY "Boot_Halle"."Halle_lang" ASC, "Bezeichnung" ASC
```

Funktion: Die wesentlichen Zuordnungen von Bootständern zu Personen sollen wie Adressen zusammengefasst werden. Hier wäre dann ein Ausdruck von Etiketten für die Bootständer möglich.

Beschreibung: Die infrage kommenden Felder sind teilweise aus mehreren Tabellen mittels „||“ zusammengesetzt, die FROM-Liste wurde hier im Schnellverfahren über die grafische Benutzeroberfläche erstellt. Sie besteht nur aus Verbindungen, die mit dem JOIN-Befehl zustande kommen. Im Verhältnis zur direkten Texteingabe ist die Zusammenstellung so allerdings etwas verwirrend.



Der Aufruf der Verbindungseigenschaften zwischen den Tabellen erfolgt mit einem Klick auf die rechte Maustaste über dem Verbindungssymbol. Hier wird gerade die Verbindung der Tabellen „Boot_Staend“ und „Boot“ zur Berabeitung geöffnet.



Alle mit dem JOIN-Befehl verbundenen Tabellen sind zusammen in {} gefasst, die mit dem Kürzel „OJ“ darauf verweisen, dass es sich um „OUTER JOIN“ handelt – wobei gleich vorweg zu sagen ist, dass der Begriff „OUTER“ genauso gut weggelassen werden kann. Bein einem „RIGHT OUTER JOIN“ werden alle Datensätze der Tabelle gezeigt, die rechts von dem JOIN steht. Die links von dem JOIN werden nur berücksichtigt, wenn sie dazu passen.
Prinzipiell hängt die Bezeichnung also lediglich von der Position ab, an der die jeweilige Tabelle steht, von der alle Datensätze berücksichtigt werden sollen.
`"Boot" AS "Boot" RIGHT OUTER JOIN "Boot_Staend" AS "Boot_Staend" ON "Boot"."bo_sID" = "Boot_Staend"."ID"` bedeutet also, dass alle „Boot_Staend“ Datensätze angezeigt werden, also alle Bootständer, aus der Tabelle „Boot“ dann nur die Datensätze, die hierzu passen. Da die Tabelle „Boot“ sowieso nur Boote enthält, die einen Liegeplatz haben, sind damit alle Boote erfasst – aber eben auch alle freien Liegeplätze.
Welche Tabelle nun als rechte und welche als linke in der grafischen

Benutzeroberfläche auftaucht ist mir bisher nicht klar geworden. Vielleicht hat dies etwas mit der Position zu tun, in der sie in dem allgemeinen Verbindungsentwurf erscheinen.

Bootstaender_Bericht_vorsortiert

```
SELECT * FROM "tmp_Bootstaender" ORDER BY "ID" ASC
```

Funktion: Vorsortierung eines Berichtes, damit der Bericht immer in der gleichen Sortierung ausgegeben wird.
Prinzipiell ist dies auch mit denb Mitteln des Berichtsdesigners machbar.
Merkwürdig ist bei dem Berichtsdesigner allerdings, dass er die automatische Sortierung der Tabelle erst einmal ignoriert (Version des Report-Designers für OOo 3.1.1).

Beschreibung: Neu gegenüber den vorhergehenden Abfragen ist lediglich das Sternchen (*) nach der SELECT-Anweisung. Es weist darauf hin, dass einfach alle Felder aus der Tabelle „tmp_Bootstaender“ auch in der Abfrage in der gleichen Spaltenanordnung wie in der Tabelle auftauchen. Nur die Sortierung nach dem Feld „ID“ erfordert diese Abfrage.

Bootstaender_frei

```
SELECT "Boot_Halle"."Halle_lang", "Boot_Staend"."BootStaend"
FROM "Boot_Staend", "Boot_Halle" WHERE "Boot_Staend"."bo_hID" =
   "Boot_Halle"."ID" AND NOT "Boot_Staend"."ID" IN ( SELECT "bo_sID" FROM "Boot"
   )
ORDER BY "Boot_Halle"."Halle_lang" ASC
```

Funktion: Alle freien Liegeplätze sollen aufgelistet werden.

Beschreibung: Die SELECT- und die FROM-Anweisung zeigen keine Besonderheiten gegenüber den vorhergehenden Abfragen. Unter den WHERE-Bedingungen befindet sich aber dieses Mal eine separate Abfrage, die eine (auch größere) Ergebnismenge haben kann.

```
AND NOT "Boot_Staend"."ID" IN ( SELECT "bo_sID" FROM "Boot" )
```

und der Primärschlüssel „ID“ aus der Tabelle „Boot_Staend“ darf nicht (Schlüsselwort „NOT“) als „bo_sID“ in (Schlüsselwort „IN“) der Tabelle „Boot“ auftauchen.

Bootstaender_vorhanden

```
SELECT "Boot_Halle"."Halle_lang" || ' - ' || "Boot_Staend"."BootStaend" AS
   "Bootsstaender", "Boot_Staend"."ID"
FROM "Boot_Staend" AS "Boot_Staend", "Boot_Halle" AS "Boot_Halle"
WHERE "Boot_Staend"."bo_hID" = "Boot_Halle"."ID"
ORDER BY "Bootsstaender" ASC
```

Funktion: Alle Bootständer sollen mit der üblichen Bezeichnung aufgelistet werden.

Beschreibung: Die Abfrage stellt lediglich die Verbindung von Boothalle und Bootständer als klare Bezeichnung für einen Bootständer in einem Feld her. Verknüpfungsmittel ist hier wieder „||“.

Daten_gesamt

```
SELECT "Mitglied_aktuell_Ansicht"."ID", "Mitglied_aktuell_Ansicht"."Nachname",
   "Mitglied_aktuell_Ansicht"."Vorname", "Mitglied_aktuell_Ansicht"."GebDat",
   "Mitglied_aktuell_Ansicht"."Schwimmer", "Strasse"."Strasse" || ' ' ||
   "Adresse"."Nr" AS "StrNr", "Postleitzahl"."Postleitzahl" || ' ' ||
   "Ort"."Ort" AS "PLZ_Ort", "Gruppe"."Telefon_fest", "Bankverbindung"."Konto",
   "Bankverbindung"."BLZ", "Bankverbindung"."Bank",
```

```
  "Bankverbindung"."Kontoinhaber",
  "tmp_Mitglied_Abteilungen_Boote"."Abteilung1",
  "tmp_Mitglied_Abteilungen_Boote"."Abteilung2",
  "tmp_Mitglied_Abteilungen_Boote"."Abteilung3",
  "tmp_Mitglied_Abteilungen_Boote"."Abteilung4",
  "tmp_Mitglied_Abteilungen_Boote"."Abteilung5",
  "tmp_Mitglied_Abteilungen_Boote"."Boot1",
  "tmp_Mitglied_Abteilungen_Boote"."Boot2",
  "tmp_Mitglied_Abteilungen_Boote"."Boot3",
  "tmp_Mitglied_Abteilungen_Boote"."Boot4",
  "tmp_Mitglied_Abteilungen_Boote"."Boot5", "Mitglied_aktuell_Ansicht"."gruID",
  "Gruppe"."adrID"
FROM "Mitglied_aktuell_Ansicht"
LEFT JOIN "tmp_Mitglied_Abteilungen_Boote" ON
  "tmp_Mitglied_Abteilungen_Boote"."ID" = "Mitglied_aktuell_Ansicht"."ID"
LEFT JOIN "Gruppe" ON "Gruppe"."ID" = "Mitglied_aktuell_Ansicht"."gruID"
LEFT JOIN "Adresse" ON "Adresse"."ID" = "Gruppe"."adrID"
LEFT JOIN ( SELECT "Konto"."ID", "Konto"."Konto", "Bank"."BLZ", "Bank"."Bank",
  CASEWHEN( "Konto"."mitID" > - 1, "Mitglied"."Nachname" || ', ' ||
  "Mitglied"."Vorname", "Konto"."Nachname" || ', ' || "Konto"."Vorname" ) AS
  "Kontoinhaber" FROM "Konto" LEFT JOIN "Bank" ON "Konto"."banID" = "Bank"."ID"
  LEFT JOIN "Mitglied" ON "Konto"."mitID" = "Mitglied"."ID" ) AS
  "Bankverbindung" ON "Bankverbindung"."ID" = "Gruppe"."konID"
LEFT JOIN "Strasse" ON "Adresse"."strID" = "Strasse"."ID"
LEFT JOIN "Postleitzahl" ON "Adresse"."posID" = "Postleitzahl"."ID"
LEFT JOIN "Ort" ON "Postleitzahl"."ortID" = "Ort"."ID"
```

Funktion: Der Mitgliederbestand wird für die Darstellung innerhalb eines Berichtes aufbereitet.

Beschreibung: Im SELECT-Bereich sticht lediglich die Sammlung an fast gleich lautenden Feldern aus der Tabelle „tmp_Mitglied_Abteilungen_Boote“ hervor. Hierdurch wird vermieden, dass ein Mitglied in dieser Abfrage mehrmals auftaucht, wenn es denn in mehreren Abteilungen ist oder mehrere Boote in den Bootshallen liegen hat. Im FROM-Bereich ist wieder der JOIN-Befehl auffällig, der dafür sorgt, dass alle Datensätze aus „tmp_Mitglied_Abteilungen_Boote“ dargestellt werden. Die besondere Unterabfrage zur Bankverbindung entspricht der Unterabfrage in der Abfrage „Abbuchungen_gesamt“.

Daten_gesamt_Filter

```
...
WHERE "Mitglied_aktuell_Ansicht"."ID" IN ( SELECT "FilterID" FROM
  "Filter_Daten_gesamt" )
```

Funktion: Filterung des gesamten Datenbestandes nach Eingabe in einem Textfeld.

Beschreibung: Die obige Abfrage wurde lediglich durch eine Filterbedingung erweitert, damit die gesamten Daten nach einem Suchbegriff durchsucht und angezeigt werden können.

Formular_Mitglied_gefiltert

```
SELECT * FROM "Mitglied"
WHERE "ID" = IFNULL( ( SELECT "FilterID" FROM "Filter" ), "ID" ) AND "ID" IN
  (SELECT "ID" FROM "tmp_Mitglied_Filter")
```

Funktion: Die Datensätze, die in dem Eingabeformular für Mitglieder angezeigt werden, sollen auch beim Sprung von einem Formular zum anderen erhalten bleiben. Diese Abfrage ist Basis für die Formulare zur Mitgliedereingabe.

Beschreibung: Alle Felder der Tabelle „Mitglied“ werden gelesen. Bedingung für die auszulesenden Datensätze: Wenn im Feld „FilterID“ nichts steht (IFNULL), dann soll das Feld „ID“ gleich dem Feld „ID“ sein – heißt, dass eben einfach alle

Datensätze dargestellt werden. Ansonsten wird die „FilterID“ ausgelesen und nur der Datensatz angezeigt, dessen „ID“ zu der „FilterID“ passt. Die Anzeige wird eingeschränkt durch die vorgefilterten Werte aus der Tabelle „tmp_Mitglied_Filter“, die als Grundlage für die Filterlistbox der Formulare erstellt wird.

Gruppe_Abteilung

```
SELECT "Mitglied_aktuell_Ansicht"."gruID",
   "Mitglied_aktuell_Ansicht"."Nachname", "Mitglied_aktuell_Ansicht"."Vorname",
   "Mitglied_aktuell_Ansicht"."GebDat", "Abteilung"."Abteilung"
FROM "rel_Mit_Abt", "Mitglied_aktuell_Ansicht", "Abteilung"
WHERE "rel_Mit_Abt"."abtID" = "Abteilung"."ID" AND "rel_Mit_Abt"."mitID" =
   "Mitglied_aktuell_Ansicht"."ID" AND NOT "Abteilung"."beiID" IS NULL
```

Funktion: Für das Formular „Beitrag“ wird eine Zusammenstellung aller Mitglieder mit allen für die Betragsberechnung relevanten Abteilungen angefertigt.

Beschreibung: Die SELECT-Anweisung birgt nichts neues, in der WHERE-Bedingung wird zum Schluss klar, dass nur die Mitlieder mit entsprechenden Abteilungen angegeben werden, bei denen die Abteilungen einen Bezug zum Beitrag haben, also einen Zusatzbeitrag ergeben.

Gruppe_Alter

```
SELECT "gruID", "Nachname", "Vorname", "AufDat", CASEWHEN( DAYOFYEAR( "GebDat" )
   > DAYOFYEAR( NOW( ) ), DATEDIFF( 'yy', "GebDat", NOW( ) ) - 1,
   DATEDIFF( 'yy', "GebDat", NOW( ) ) ) AS "Alter", DATEDIFF( 'yy', "GebDat",
   NOW( ) ) AS "Alter_aktuelles_Jahr"
FROM "Mitglied_aktuell_Ansicht"
ORDER BY "gruID" ASC,"Alter" DESC
```

Funktion: Für das Formular „Beitrag“ ist das Alter der betreffenden Personen von Bedeutung. Das Alter wird hier einmal als momentanes Alter und als in dem aktuellen Jahr zu erwartendes Alter (hiernach werden z.B. Wettkampfklassen eingeteilt) dargestellt.

Beschreibung: Die Datumsberechnungen in der SELECT-Anweisung sind neu.
`DATEDIFF( 'yy', "GebDat", NOW( ) )` ergibt den Unterschied zwischen dem Datum aus „GebDat“ und dem heutigen Datum. Allerdings werden, zumindest bei der HSQLDB, auf diese Art und Weise nur die Jahreszahlen ('yy') verglichen. Liegt das Geburtsdatum im Vorjahr, so erscheint auf diese Weise auf jeden Fall eine „1“ bei der Berechnung – auch wenn heute gerade der 1.1. ist und die Person am 31.12. geboren wurde.
Daraus ergibt sich ein etwas umfassenderer Aufbau der Berechnung für das momentane Alter:
`CASEWHEN( DAYOFYEAR( "GebDat" ) > DAYOFYEAR( NOW( ) ), DATEDIFF( 'yy', "GebDat", NOW( ) ) - 1, DATEDIFF( 'yy', "GebDat", NOW( ) ) )` für den Fall, dass der Tag, gezählt als Tag im Jahr, von dem Geburtsdatum größer ist als der Tag heute, soll von der Jahresdifferenz 1 abgezogen werden. Ansonsten ist die Jahresdifferenz in das Feld zu schreiben.

Gruppe_Bezeichnung

```
SELECT "Mitglied_aktuell_Ansicht"."Nachname" ||', '||"Vorname"|| ' - Tel: ' ||
   IFNULL( "Gruppe"."Telefon_fest", 'keins' ) AS "Name", "Gruppe"."ID",
   "Gruppe"."konID", "Gruppe"."adrID"
FROM "Mitglied_aktuell_Ansicht", "Gruppe"
WHERE "Mitglied_aktuell_Ansicht"."gruID" = "Gruppe"."ID" AND
   "Mitglied_aktuell_Ansicht"."ID" IN ( SELECT
   MIN( "Mitglied_aktuell_Ansicht"."ID" ) FROM "Mitglied_aktuell_Ansicht"
GROUP BY "Mitglied_aktuell_Ansicht"."gruID" )
```

```
ORDER BY "Name" ASC
```

Funktion: Die Gruppentabelle birgt neben Zahlenfeldern nur die Festnetz-Telefonnummer. Hier soll ein lesbarer Name aus der Mitgliedstabelle zugeordnet werden. Der Name des ältesten Mitglieds ist manchmal mangels eindeutiger Eingabe nicht zu ermitteln, so dass stattdessen das zuerst eigegebene Mitglied mit Nachnamen angezeigt wird.

Beschreibung: Das mit „||“ zusammengefasste Feld „Name“ wird aus dem Nachnamen und, sofern eine Telefonangabe existiert, der Telefonnummer zusammengesetzt.
`IFNULL("Gruppe"."Telefon_fest",'keins')` sofern ein Eintrag in „Telefon_fest“ existiert, wird dieser genommen, ansonsten 'keins' geschrieben. Dieses Verfahren ist notwendig, da sonst in zusammengefügten Feldern keine Anzeige erfolgt, wenn ein Element keinen Inhalt hat.
Die Ansicht „Mitglied_aktuell_Ansicht“ gibt nur die aktuellen Mitglieder wieder. Siehe hierzu die Abfrage „Mitglied_aktuell“ weiter unten.

Gruppe_Boot

```
SELECT "Mitglied"."gruID", "Mitglied"."Nachname", "Mitglied"."Vorname",
   "Boot_Art"."Bootsart", "Boot"."Bootsname"
FROM "Boot", "Boot_Art", "Mitglied"
WHERE "Boot"."bo_aID" = "Boot_Art"."ID" AND "Boot"."mitID" = "Mitglied"."ID"
```

Funktion: Für das Formular „Beitrag“ werden die Boote dargestellt.

Beschreibung: Gegenüber den vorhergehenden Abfragen keine Besonderheiten.

Gruppe_Groesse

```
SELECT COUNT( "ID" ) AS "Gruppengroesse", "gruID"
FROM "Mitglied_aktuell_Ansicht"
WHERE "gruID" IS NOT NULL
GROUP BY "gruID"
```

Funktion: Die Gruppengrößen werden festgestellt, damit bei der Beitragsermittlung gegebenenfalls nach den größeren Gruppen mit Familienermäßigungen usw. gefiltert werden kann.

Beschreibung: COUNT ist der Zählbefehl. Gezählt werden alle gleichen Gruppennummern. Dies geht aus der GROUP BY – Bedingung (gruppiert nach …) hervor. Einzige Zählungsausnahme: Es ist keine Gruppe angegeben, das Feld also NULL.

Jubilaeum

```
SELECT "ID", "Vorname" || ' ' || "Nachname" AS "Name", 'für ' || DATEDIFF( 'YY',
   "AufDat", NOW( ) ) || '-jährige Mitgliedschaft' AS "Mitgliedsjahre"
FROM "Mitglied" AS "Mitglied"
WHERE ( "AusDat" IS NULL AND DATEDIFF( 'YY', "AufDat", NOW( ) ) = 70 OR
   DATEDIFF( 'YY', "AufDat", NOW( ) ) = 60 OR DATEDIFF( 'YY', "AufDat", NOW( ) )
   = 50 OR DATEDIFF( 'YY', "AufDat", NOW( ) ) = 40 OR DATEDIFF( 'YY', "AufDat",
   NOW( ) ) = 25 )
ORDER BY "Mitgliedsjahre" DESC
```

Funktion: Die Mitgliedsdauer wird festgestellt und für die entsprechend definierten Jubiläumsjahre ein Text für eine Urkunde vorbereitet. Diese Abfrage ist Grundlage für den Urkundendruck über einen Bericht.

Beschreibung: Über DATEDIFF('YY'...) wird die Differenz in Jahren gegenüber dem Beitrittsdatum ermittelt. Mitglieder, die in dem aktuellen Jahr ein Jubiläum haben werden somit berücksichtigt.

Konten_Kontoinhaber

```
SELECT "Konto"."ID", "Konto"."Konto", "Bank"."BLZ", "Bank"."Bank",
```

```
  CASEWHEN( "Konto"."mitID" > - 1, "Mitglied"."Nachname" || ', ' ||
  "Mitglied"."Vorname", "Konto"."Nachname" || ', ' || "Konto"."Vorname" ) AS
  "Kontoinhaber"
FROM "Konto"
LEFT JOIN "Bank" ON "Konto"."banID" = "Bank"."ID"
LEFT JOIN "Mitglied" ON "Konto"."mitID" = "Mitglied"."ID"
```

Funktion: Übersicht über die Kontenverbindung für Abbuchungen

Beschreibung: Diese Abfrage ist integriert in die Abfragen „Abbuchungen_gesamt“ und „Daten_gesamt“

Mitglied_Adresse

```
SELECT "Anzeige", "ID", "Sort" FROM "Listfeld_neu" AS "Listfeld_neu"
UNION ALL SELECT "Strasse"."Strasse"||' '||"Adresse"."Nr"||'
  '||"Adresse"."Land"||' '||"Postleitzahl"."Postleitzahl"||' '||"Ort"."Ort" AS
  "Anzeige", "Adresse"."ID","Strasse"."Strasse"||' '||"Adresse"."Nr" AS "Sort"
FROM "Adresse" AS "Adresse", "Postleitzahl" AS "Postleitzahl", "Ort" AS "Ort",
  "Strasse" AS "Strasse"
WHERE "Adresse"."posID" = "Postleitzahl"."ID"
AND "Postleitzahl"."ortID" = "Ort"."ID"
AND "Adresse"."strID" = "Strasse"."ID"
ORDER BY "Sort" ASC
```

Funktion: Für das Formular „Adresse“ wird ein Auswahlfeld erstellt. Das Listfeld zeigt als obersten Wert „Neu“ an. Anschließend erfolgt die Ausgabe der kompletten Adressen in dem sichtbaren Feld, sortiert nach den Straßennamen.

Beschreibung: Die Abfrage „Abteilung_Filter“ erklärt die Grundlage zu dieser Erstellung von Inhalten für eine Listbox. Alle anderen Inhalte der Abfrage sind weiter oben bereits erläutert.

Mitglied_Adressen

```
SELECT "Mitglied_aktuell_Ansicht"."Nachname",
  "Mitglied_aktuell_Ansicht"."Vorname", "Mitglied_aktuell_Ansicht"."GebDat",
  "Gruppe"."adrID", "Gruppe"."ID", "Gruppe"."Telefon_fest"
FROM "Mitglied_aktuell_Ansicht", "Gruppe"
WHERE "Mitglied_aktuell_Ansicht"."gruID" = "Gruppe"."ID"
ORDER BY "Gruppe"."adrID" ASC, "Gruppe"."ID" ASC,
  "Mitglied_aktuell_Ansicht"."GebDat" ASC,
  "Mitglied_aktuell_Ansicht"."Nachname" ASC,
  "Mitglied_aktuell_Ansicht"."Vorname" ASC
```

Funktion: Zeigt im Formular „Adressen“ alle Mitglieder mit der gleichen Adresse an. Bei Gruppen mit gleichen Adressen werden diese durch die Gruppennummer und die Telefonnummer unterschieden.

Beschreibung: Gegenüber den vorhergehenden Abfragen keine Besonderheiten.

Mitglied_aktuell

```
SELECT * FROM "Mitglied" WHERE ( "AusDat" IS NULL OR "AusDat" > NOW( ) )
```

Funktion: Ersetzt die in vielen Abfragen sonst nötige Zusatzabfrage, ob die Person überhaupt noch im Verein ist. Ist in der gleichen Form als View abgespeichert.

Beschreibung: Alle Elemente der Tabelle Mitglied werden wiedergegeben.
`"AusDat" IS NULL OR "AusDat" > NOW( )`
„IS NULL“ bedeutet, dass das Feld leer ist (also auch keine 0 darin steht!), bisher hier keine Eingabe erfolgte. „OR“ sagt nur aus, dass die Bedingung davor oder die dahinter oder eben auch ruhig beide erfüllt sein dürfen. `„AusDat“ > NOW( )` heißt nur, dass ein eventuell eingetragenes Austrittsdatum größer als das heutige Datum sein soll, d.h. es muss in der Zukunft liegen.

Mitglied_Anschrift

```
SELECT "Mitglied"."gruID", "Mitglied"."Nachname", "Mitglied"."Vorname",
   "Strasse"."Strasse", "Adresse"."Nr", "Adresse"."Land",
   "Postleitzahl"."Postleitzahl", "Ort"."Ort", "Mitglied"."Geschlecht",
   DATEDIFF( 'yy', "Mitglied"."GebDat", NOW( ) ) AS "Alter_aktuelles_Jahr",
   "Mitglied"."ID" AS "mitID", "Mitglied"."AusDat"
FROM "Mitglied", "Gruppe", "Adresse", "Strasse", "Postleitzahl", "Ort"
WHERE "Mitglied"."gruID" = "Gruppe"."ID" AND "Gruppe"."adrID" = "Adresse"."ID"
   AND  "Adresse"."strID" = "Strasse"."ID" AND "Adresse"."posID" =
   "Postleitzahl"."ID" AND "Postleitzahl"."ortID" = "Ort"."ID" AND
   "Adresse"."ID" NOT IN (SELECT "ID" FROM "Adresse" WHERE "Nichtdruck" = 1) AND
   "Mitglied"."ID" NOT IN (SELECT "ID" FROM "Mitglied" WHERE "Nichtdruck" = 1)
ORDER BY "Mitglied"."gruID" ASC, "Alter_aktuelles_Jahr" DESC
```

Funktion: Jedes Mitglied wird mit seiner kompletten Anschrift gelistet, damit auch z.B. Abteilungsrundschreiben erfolgen können. Mitglieder werden nur gelistet, wenn der Datensatz für den Ausdruck nicht gesperrt ist (Vermeidung von Postrückläufern wegen falscher Adressen ...)

Beschreibung: Umfangreicher gegenüber vorhergehenden Abfragen hier die Bedingung für den Nichtdruck:
`AND "Adresse"."ID" NOT IN (SELECT "ID" FROM "Adresse" WHERE "Nichtdruck" = 1) AND "Mitglied"."ID" NOT IN (SELECT "ID" FROM "Mitglied" WHERE "Nichtdruck" = 1)` das Feld „Nichtdruck“ existiert sowohl für die ganze Adresse als auch für das einzelne Mitglied. Ist die Adresse auf „Nichtdruck“ gestellt, da von der Adresse grundsätzlich jede Post zurückkommt, so ist die Nichtdruck-Einstellung beim Mitglied egal. Bei den Ja/Nein-Feldern entspricht 0 einem nicht angekreuzten Wert (Nein), 1 einem angekreuzten Wert. Ist ein Ja/Nein-Feld bisher nicht betätigt worden, so ist der dazugehörige Feldinhalt „NULL“, d.h. leer. Da das Feld so 3 Zustände annehmen kann ist die Abfrage des Nichtdrucks unproblematisch (schließlich ist das Feld angekreuzt), aber die Abfrage des Gegenteils wegen der 2 möglichen Varianten etwas umständlich. Da die „Nichtdruck“-Anweisungen nur eine geringe Zahl einnehmen werden wird hier einfach nachgesehen, ob die jeweiligen Nummern nicht in der Liste der Nichtdrucknummern vorkommen.

Mitglied_Filter

```
SELECT "Anzeige", "ID", "Sort" FROM "Filter_aus" AS "Filter_aus"
UNION ALL SELECT "Nachname" || ', ' || "Vorname" AS "Anzeige", "ID", "Nachname"
   || ', ' || "Vorname" AS "Sort"
FROM "Mitglied" AS "Mitglied"
ORDER BY "Sort"
```

Funktion: Für die Mitgliedereingabe wird hier das Listenfeld bestückt. Die Anzeige „Alle“ sorgt dafür, dass im Formular alle Mitgliederdatensätze mittels Navigationsleiste angesprungen werden können.

Beschreibung: Gegenüber den vorhergehenden Abfragen keine Besonderheiten.

Mitglied_Formular

```
SELECT "Mitglied"."ID", "Mitglied"."ID" AS "mitID", "Mitglied"."Nachname",
   "Mitglied"."Vorname", "Mitglied"."Geschlecht", "Mitglied"."GebDat",
   "Mitglied"."AufDat", "Mitglied"."Schwimmer", "Mitglied"."AusDat",
   "Mitglied"."Nichtdruck", "Mitglied"."Telefon_mobil", "Mitglied"."gruID",
   "Gruppe"."Telefon_fest", "Gruppe"."adrID", "Gruppe"."konID", "Adresse"."Nr",
   "Adresse"."posID", "Adresse"."Land", "Adresse"."Nichtdruck" AS
   "Nichtdruck_Adr", "Adresse"."strID", "Konto"."Nachname" AS "Nachname_Kon",
   "Konto"."Vorname" AS "Vorname_Kon", "Konto"."Konto", "Konto"."banID",
```

```
  "Konto"."mitID" AS "mitID_Kon", "Konto"."IBAN_PZ"
FROM "Mitglied" LEFT JOIN "Gruppe" ON "Mitglied"."gruID" = "Gruppe"."ID" LEFT
  JOIN "Adresse" ON "Gruppe"."adrID" = "Adresse"."ID" LEFT JOIN "Konto" ON
  "Gruppe"."konID" = "Konto"."ID"
```

Funktion: Alle notwendigen Daten für ein Mitglied werden zusammengefasst, damit in dem Mitgliedsformular nicht mit zu vielen Unterformularen gearbeitet werden muss. Diese Abfrage ist gleich der, die als Ansicht daraus erstellt wurde, um per Makro schließlich auf die Ansicht zugreifen zu können.
Der Inhalt eines Datensatzes wird anschließend per Makro in die Tabelle „Filter_Formular“ übertragen.

Mitglied_Gruppe

```
SELECT "Anzeige", "ID", "Sort" FROM "Listfeld_neu" WHERE "Anzeige" = 'Neu'
UNION ALL SELECT "Mitglied_aktuell_Ansicht"."Nachname" || ', ' || "Vorname" || '
  - Tel: ' || IFNULL( "Gruppe"."Telefon_fest", 'keins' ) AS "Anzeige",
  "Gruppe"."ID", "Mitglied_aktuell_Ansicht"."Nachname" || ', ' || "Vorname" ||
  ' - Tel: ' || IFNULL( "Gruppe"."Telefon_fest", 'keins' ) AS "Sort" FROM
  "Mitglied_aktuell_Ansicht", "Gruppe" WHERE "Mitglied_aktuell_Ansicht"."gruID"
  = "Gruppe"."ID" AND "Mitglied_aktuell_Ansicht"."ID" IN ( SELECT
  MIN( "Mitglied_aktuell_Ansicht"."ID" ) FROM "Mitglied_aktuell_Ansicht" GROUP
  BY "Mitglied_aktuell_Ansicht"."gruID" )
ORDER BY "Sort" ASC
```

Funktion: Beim Formular der Mitgliedereingabe wird eine Gruppenzuweisung erstellt, um z.B. Familien klar filtern zu können.

Beschreibung: An die Tabelle „Listfeld_neu“ wird der Inhalt einer Abfrage angehängt. Mit dem Schlüsselwort „DISTINCT“ werden alle gleichlautenden Werte des nachfolgenden Datensatzes von der Anzeige ausgeschlossen. Da in Gruppen mehrere Mitglieder existieren würde sonst eine Gruppe so häufig angezeigt, wie Mitglieder für die Gruppe vorhanden sind. In der Anzeige werden wieder Feldinhalte und andere Inhalte miteinander verbunden – hier zum ersten Mal ein Text, der dann erscheint, wenn für die Gruppe kein gemeinsames Festnetztelefon existiert. Dann erscheint nämlich der Text 'keins'. Da die Zuweisung der Gruppe aber Vorbedingung für eine Verbindung zur Adresse und zum Konto ist, wird durch das Formular statt leerer Telefonangaben ein '-' gesetzt. Sonst wäre die Gründung einer neuen Gruppe nicht möglich, da in dem neuen Datensatz kein Inhalt stehen würde.

Mitglied_Gruppen

```
SELECT "Nachname", "Vorname", "GebDat", "gruID"
FROM "Mitglied_aktuell_Ansicht"
ORDER BY "gruID" ASC, "GebDat" ASC, "Nachname" ASC, "Vorname" ASC
```

Funktion: Für das Formular der Mitgliedseingabe wird als Anzeige eine Aufstellung der Gruppenmitglieder erstellt.

Beschreibung: Gegenüber den vorhergehenden Abfragen keine Besonderheiten.

Mitglied_Konten

```
SELECT "Mitglied_aktuell_Ansicht"."Nachname",
  "Mitglied_aktuell_Ansicht"."Vorname", "Mitglied_aktuell_Ansicht"."GebDat",
  "Gruppe"."konID", "Gruppe"."ID", "Gruppe"."Telefon_fest"
FROM "Mitglied_aktuell_Ansicht", "Gruppe"
WHERE "Mitglied_aktuell_Ansicht"."gruID" = "Gruppe"."ID"
ORDER BY "Gruppe"."konID", "Gruppe"."ID", "Mitglied_aktuell_Ansicht"."GebDat",
  "Mitglied_aktuell_Ansicht"."Nachname", "Mitglied_aktuell_Ansicht"."Vorname"
```

Funktion: Für das Formular „Konto“ wird aufgelistet, welche Mitglieder vom gleichen Konto

ihren Beitrag einziehen lassen.

Beschreibung: Gegenüber den vorhergehenden Abfragen keine Besonderheiten. Die Sortierung ist hier ohne eine Richtung angegeben. Unter diesen Umständen wählt die Datenbank immer die aufsteigende Sortierung.

Mitglied_Konto

```
SELECT "Anzeige", "ID", "Sort"
FROM "Listfeld_neu" AS "Listfeld_neu"
UNION ALL SELECT IFNULL( ( 'Konto:' || "Konto"."Konto" || ' BLZ:' ||
  "Bank"."BLZ" || ', ' || "Bank"."Bank" || ', Kontoinhaber:' ||
  CASEWHEN( "Konto"."mitID" > - 1, "Mitglied"."Nachname" || ', ' ||
  "Mitglied"."Vorname", "Konto"."Nachname" || ', ' || "Konto"."Vorname" ) ),
  'Fehler in der Angabe' ) AS "Anzeige", "Konto"."ID", "Konto"."Konto" AS
  "Sort"
FROM "Konto"
LEFT JOIN "Bank" ON "Konto"."banID" = "Bank"."ID"
LEFT JOIN "Mitglied" ON "Konto"."mitID" = "Mitglied"."ID"
ORDER BY "Sort" ASC
```

Funktion: Das Formular „Konto“ benötigt für die Zuweisung ein Listfeld, in dem alle notwendigen Kontoeingaben zusammengefasst werden. Bei Auswahl von „Neu“ wird ein neues Konto eingegeben, bei anderen Werten wird das alte Konto bearbeitet. Alle vorhandenen Kontoeintragungen werden angezeigt.

Beschreibung: Wenn das zusammengefasste Feld „Anzeige“ nur einen leeren Wert enthält bleibt bei der Konstruktion mit „||“ die Anzeige leer. Deswegen wird in diesem Fall 'Fehler in der Angabe' in dem Listenfeld eingeblendet.
Das Konto ist mit der Bank mit einem LEFT JOIN verbunden, damit auf jeden Fall alle Konten angezeigt werden. Im Gegensatz zu „Mitglied_Konto_Formular“ zeigt diese Abfrage alle Datensätze der Konto-Tabelle an, also auch die der bereits ausgetretenen Mitglieder. Dies ist z.B. bei einem Neueintritt von Vorteil.

Mitglied_Konto_Formular

```
SELECT CASEWHEN( "Konto"."mitID" > - 1, "Mitglied"."Nachname" || ', ' ||
  "Mitglied"."Vorname", "Konto"."Nachname" || ', ' || "Konto"."Vorname" ) || '
  (Kontoinhaber) ' || 'Konto:' || "Konto"."Konto" || ' BLZ:' || "Bank"."BLZ" ||
  ', ' || "Bank"."Bank" AS "Anzeige", "Konto"."ID",
  "Gruppe_Groesse"."Gruppengroesse"
FROM "Konto"
LEFT JOIN "Bank" ON "Konto"."banID" = "Bank"."ID"
LEFT JOIN "Mitglied" ON "Konto"."mitID" = "Mitglied"."ID", "Gruppe",
  "Gruppe_Groesse"
WHERE "Konto"."ID" = "Gruppe"."konID" AND "Gruppe"."ID" =
  "Gruppe_Groesse"."gruID"
AND "Konto"."ID" = IFNULL((SELECT "FilterID" FROM "Filter_Konto"),"Konto"."ID")
ORDER BY "Anzeige" ASC
```

Funktion: Die Abfrage wird für die Kontenansicht im Formular benötigt, wenn der Beitrag für eine Gruppe überarbeitet werden soll. Die Gruppengröße kann im Formular gefiltert werden. Sie soll bereits auf ein Konto vorgefiltert werden, wenn aus der Mitgliedstabelle direkt die Beitragsermittlung aufgerufen wird.

Beschreibung: Lediglich 3 Felder werden dargestellt: Das Feld des Kontos mit zusätzlichen Informationen (siehe oben), das ID-Feld, damit die ID an die darunterliegenden Formulare weitergegeben werden kann und die Gruppengröße, damit vor allem die Gruppen-Beitragsermäßigung zügig überprüft werden kann. Es werden nur die Datensätze ausgegeben, die mit aktuellen Mitgliedern zusammenhängen. Das stellt die Abfrage `Gruppe_Groesse` sicher, die über die Gruppe mit dem Konto

verbunden ist.
Zur Sortierung wird der Kontoinhaber herangezogen. Damit auch direkt ein Konto angesteuert werden kann, wenn das Formular geöffnet wird, wird in der Tabelle „Filter_Konto“ eine entsprechende „FilterID“ abgelegt. Diese „FilterID“ wird nur genutzt, wenn sie nicht NULL ist (IFNULL).

Mitglied_Kontoinhaber

```
SELECT DISTINCT '' AS "Anzeige", NULL AS "ID", '00000' AS "Sort"
FROM "Listfeld_neu" AS "Listfeld_neu"
UNION ALL SELECT "Mitglied"."Nachname" || ', ' || "Mitglied"."Vorname" AS
   "Anzeige", "Mitglied"."ID" AS "ID", "Mitglied"."Nachname" || ', ' ||
   "Mitglied"."Vorname" AS "sort"
FROM "Konto", "Mitglied" WHERE "Konto"."mitID" = "Mitglied"."ID"
ORDER BY "Sort" ASC
```

Funktion: Inhaltserstellung für ein Listfeld, das alle Kontoinhaber auflistet, die gleichzeitig Mitglieder im Verein sind.

Mitglied_Mitgliedsdauer

```
SELECT "ID", "Nachname", "Vorname", "AufDat", DATEDIFF( 'YY', "AufDat", NOW( ) )
   AS "Mitgliedsjahre"
FROM "Mitglied" AS "Mitglied" WHERE "AufDat" IS NOT NULL AND "AusDat" IS NULL
   AND DATEDIFF( 'YY', "AufDat", NOW( ) ) > 20
ORDER BY "AufDat" ASC
```

Funktion: Auflistung aller Mitglieder, die länger als 20 Jahre Mitglied im Verein sind. Erzeugt eine Übersicht über die in Zukunft zu ehrenden Mitglieder.

Mitgliederbewegung

```
SELECT "Mitglied"."ID", "Mitglied"."Nachname", "Mitglied"."Vorname",
   "Mitglied"."GebDat", "Mitglied"."AufDat", "Mitglied"."AusDat",
   "Mitglied"."Schwimmer", "Strasse"."Strasse" || ' ' || "Adresse"."Nr" AS
   "StrNr", "Postleitzahl"."Postleitzahl" || ' ' || "Ort"."Ort" AS "PLZ_Ort",
   "Gruppe"."Telefon_fest", "Bankverbindung"."Konto", "Bankverbindung"."BLZ",
   "Bankverbindung"."Bank", "Bankverbindung"."Kontoinhaber",
   "tmp_Mitglied_Abteilungen_Boote"."Abteilung1",
   "tmp_Mitglied_Abteilungen_Boote"."Abteilung2",
   "tmp_Mitglied_Abteilungen_Boote"."Abteilung3",
   "tmp_Mitglied_Abteilungen_Boote"."Abteilung4",
   "tmp_Mitglied_Abteilungen_Boote"."Abteilung5",
   "tmp_Mitglied_Abteilungen_Boote"."Boot1",
   "tmp_Mitglied_Abteilungen_Boote"."Boot2",
   "tmp_Mitglied_Abteilungen_Boote"."Boot3",
   "tmp_Mitglied_Abteilungen_Boote"."Boot4",
   "tmp_Mitglied_Abteilungen_Boote"."Boot5", "Mitglied"."gruID",
   "Gruppe"."adrID"
FROM "Mitglied"
LEFT JOIN "tmp_Mitglied_Abteilungen_Boote" ON
   "tmp_Mitglied_Abteilungen_Boote"."ID" = "Mitglied"."ID"
LEFT JOIN "Gruppe" ON "Gruppe"."ID" = "Mitglied"."gruID"
LEFT JOIN "Adresse" ON "Adresse"."ID" = "Gruppe"."adrID"
LEFT JOIN ( SELECT "Konto"."ID", "Konto"."Konto", "Bank"."BLZ", "Bank"."Bank",
   CASEWHEN( "Konto"."mitID" > - 1, "Mitglied"."Nachname" || ', ' ||
   "Mitglied"."Vorname", "Konto"."Nachname" || ', ' || "Konto"."Vorname" ) AS
   "Kontoinhaber" FROM "Konto" LEFT JOIN "Bank" ON "Konto"."banID" = "Bank"."ID"
   LEFT JOIN "Mitglied" ON "Konto"."mitID" = "Mitglied"."ID" ) AS
   "Bankverbindung" ON "Bankverbindung"."ID" = "Gruppe"."konID"
LEFT JOIN "Strasse" ON "Adresse"."strID" = "Strasse"."ID"
LEFT JOIN "Postleitzahl" ON "Adresse"."posID" = "Postleitzahl"."ID"
```

```
LEFT JOIN "Ort" ON "Postleitzahl"."ortID" = "Ort"."ID"
```

Funktion: Sämtliche Daten über Mitglieder werden zusammengefasst, um anschließend eine Übersicht über Veränderungen in der Mitgliederbewegung zu erzeugen. Hier geht es um die Austritte und Beitritte in jedem Quartal o.ä.
Diese Abfrage existiert in gleicher Form als Ansicht, um sie über Makros auslesen und weiterverarbeiten zu können.
Je nach Vorauswahl im Formular „Berichte“ wird die Tabelle „tmp_Mitgliederbewegung_rein“ bzw. „tmp_Mitgliedrbewegung_raus“ mit dem entsprechenden Inhalt gefüllt und ein Bericht zu den jeweiligen Tabelleninhalten erstellt.

Schluessel_Anzahl

```
SELECT "Schl_Anz"."SNr", SUM( "Schl_Anz"."Anzahl" ) -
   IFNULL( "Schluesselausleihe"."Verliehen",0) AS "Bestand"
FROM  "Schluessel"
LEFT JOIN "Schl_Anz" ON "Schl_Anz"."SNr" = "Schluessel"."SNr"
LEFT JOIN (SELECT "SNr", COUNT( "SNr" ) AS "Verliehen" FROM "rel_Mit_Schl" WHERE
   "RueckDat" IS NULL GROUP BY "SNr") AS "Schluesselausleihe" ON
   "Schluesselausleihe"."SNr" = "Schluessel"."SNr"
GROUP BY "Schl_Anz"."SNr", "Schluesselausleihe"."Verliehen"
```

Funktion: Die Summe aller noch zur Ausgabe vorliegenden Schlüssel für die jeweilige Schlüsselnummer soll ermittelt werden.

Beschreibung: Wenn kein Schlüssel entliehen wurde kann keine Berechnung durchgeführt werden. Deshalb muss in diesem Fall statt des NULL-Ergebnisses der Abfrage eine „0“ zur Berechnung gesetzt werden.
Die Anzahl der entliehenen Schlüssel wird in der gesonderten Abfrage „Schluesselausleihe“ gezählt. Würde dies innerhalb einer Abfrage erledigt, so würde für jeden Datensatz aus „rel_Mit_Schl“ eine neue Verknüpfung in der Abfrage erstellt. Bei z.B. 3 entliehenen Schlüsseln gäbe es auch die Anzahl anschließend dreifach. Die Rechnung wäre also nicht stimmig.

Schluessel_Bestand

```
SELECT DISTINCT "Schluessel"."SNr" || '  - Schließbereich: ' ||
   "Schluessel"."Bereich" AS "Schliessbereich", "Schluessel"."SNr",
   "Schluessel_Anzahl"."Bestand"
FROM { OJ "rel_Mit_Schl" AS "rel_Mit_Schl" RIGHT OUTER JOIN "Schluessel" AS
   "Schluessel" ON "rel_Mit_Schl"."SNr" = "Schluessel"."SNr" },
   "Schluessel_Anzahl" AS "Schluessel_Anzahl"
WHERE "Schluessel"."SNr" = "Schluessel_Anzahl"."SNr"
```

Funktion: Der Schlüsselbestand wird ausführlich lesbar dargestellt.

Beschreibung: Gegenüber den vorhergehenden Abfragen keine Besonderheiten.

Schluessel_Bestand_0

```
SELECT "Schluessel"."SNr" || '  - Schließbereich: ' || "Schluessel"."Bereich" AS
   "Schliessbereich", "Schluessel"."SNr", "Schluessel_Anzahl"."Bestand"
FROM { OJ "rel_Mit_Schl" AS "rel_Mit_Schl" RIGHT OUTER JOIN "Schluessel" AS
   "Schluessel" ON "rel_Mit_Schl"."SNr" = "Schluessel"."SNr" },
   "Schluessel_Anzahl" AS "Schluessel_Anzahl"
WHERE "Schluessel"."SNr" = "Schluessel_Anzahl"."SNr"
AND "Schluessel_Anzahl"."Bestand" = 0
```

Funktion: Alle Schlüsselnummern werden gelistet, für die keine Schlüssel zur Ausgabe mehr vorhanden sind.

Beschreibung: Gegenüber den vorhergehenden Abfragen keine Besonderheiten.

Schluessel_Uebersicht

```
SELECT "rel_Mit_Schl"."SNr", "Mitglied"."Nachname" || ', ' ||
   "Mitglied"."Vorname" AS "Name"
FROM "rel_Mit_Schl" AS "rel_Mit_Schl", "Mitglied" AS "Mitglied"
WHERE "rel_Mit_Schl"."mitID" = "Mitglied"."ID" AND "rel_Mit_Schl"."RueckDat" IS
   NULL AND ( "rel_Mit_Schl"."Verlust" = 0 OR "rel_Mit_Schl"."Verlust" IS NULL )
ORDER BY "rel_Mit_Schl"."SNr" ASC, "Name" ASC
```

Funktion: Anhand der Abfrage wird ein Bericht erstellt, mit dem augegeben wird, welche Personen über einen Schlüssel verfügen.

Beschreibung: Es werden nur Schlüssel angegeben, die bisher nicht zurückgegeben wurden (RueckDat IS NULL) und die nicht als verloren oder beschädigt abgeschrieben wurden. Da letzteres über ein Ja/Nein-Feld mit 3 unterschiedlichen Einstellungen abgefragt wird, ist hier nach Feldern mit „0“ oder leeren Feldern „IS NULL“ gefragt.

Statistik_Alter

```
SELECT "Beitrag"."BeitrArt" AS "Alter", COUNT( "Mitglied_aktuell_Ansicht"."ID" )
   AS "Anzahl" FROM "Mitglied_aktuell_Ansicht", "Beitrag"
WHERE CASEWHEN( DAYOFYEAR( "GebDat" ) > DAYOFYEAR( ( SELECT "Stichtag" FROM
   "Filter_Statistik" WHERE "ID" = 0 ) ), DATEDIFF( 'yy', "GebDat", ( SELECT
   "Stichtag" FROM "Filter_Statistik" WHERE "ID" = 0 ) ) - 1, DATEDIFF( 'yy',
   "GebDat", ( SELECT "Stichtag" FROM "Filter_Statistik" WHERE "ID" = 0 ) ) )
   BETWEEN "Beitrag"."MinAlter" AND "Beitrag"."MaxAlter" AND
   "Beitrag"."BeitrBezug" = 'Mit'
GROUP BY "Beitrag"."BeitrArt"
```

Funktion: Zu Meldungen an Landesverbände usw. sind häufig statistische Informationen wichtig. Hier wird beispielhaft nach den Altersstrukturen aus den Beiträgen die jeweilige Zahl der Mitglieder ermittelt. Diese Abfrage ist im Berichtsformular untergebracht.

Beschreibung: Zuerst werden die verschiedenen Beitragsstrukturen mit dem BeitrBezug 'Mit' aufgelistet. Nach diesen Strukturen wird die ganze Abfrage gruppiert. Gezählt werden die Datensätze aus „Mitglied_aktuell_Ansicht“, hier einfach der Primärschlüssel „ID“. Die Altersbestimmung geht im Gegensatz zu den vorherigen Abfragen nicht von dem momentanen Datum aus. Stattdessen wird das Alter zu einem bestimmten Stichtag ermittelt. Dieser Stichtag ist abgespeichert in der Tabelle „Filter_Statistik“ mit dem Primärschlüssel 0. Im Formular wird dieser Wert lediglich laufend überschrieben. Die Altersermittlung stellt den Zusammenhang zwischen „Beitrag“ und „Mitglied_aktuell_Ansicht“ her.

Statistik_Alter_Detail

```
SELECT "Altersgruppen"."MinAlter" || ' bis ' || "Altersgruppen"."MaxAlter" AS
   "Alter", "Mitglied_aktuell_Ansicht"."Geschlecht",
   COUNT( "Mitglied_aktuell_Ansicht"."ID" ) AS "Anzahl"
FROM "Mitglied_aktuell_Ansicht", "Altersgruppen" WHERE
   CASEWHEN( DAYOFYEAR( "GebDat" ) > DAYOFYEAR( ( SELECT "Stichtag" FROM
   "Filter_Statistik" WHERE "ID" = 0 ) ), DATEDIFF( 'yy', "GebDat", ( SELECT
   "Stichtag" FROM "Filter_Statistik" WHERE "ID" = 0 ) ) - 1, DATEDIFF( 'yy',
   "GebDat", ( SELECT "Stichtag" FROM "Filter_Statistik" WHERE "ID" = 0 ) ) )
   BETWEEN "Altersgruppen"."MinAlter" AND "Altersgruppen"."MaxAlter" AND
   ( "Mitglied_aktuell_Ansicht"."AusDat" > ( SELECT "Stichtag" FROM
   "Filter_Statistik" WHERE "ID" = 0 ) OR "Mitglied_aktuell_Ansicht"."AusDat" IS
   NULL )
GROUP BY "Alter", "Mitglied_aktuell_Ansicht"."Geschlecht"
ORDER BY "Mitglied_aktuell_Ansicht"."Geschlecht", "Altersgruppen"."MinAlter"
```

Funktion: Genauere Abfrage nach Geschlecht und Altersgruppen als Grundlage für einen Bericht und die weitere Auswertung z.B. für die Meldung zu Verbänden.

Beschreibung: Die Altersabfrage wurde bereits weiter oben erklärt. In der Tabelle „Altersgruppen“ finden sich die Daten, nach denen die Übersicht differenziert werden soll.

Vorstand_aktuell

```
SELECT "Vorstand_Posten"."Posten", "Mitglied"."Nachname", "Mitglied"."Vorname",
  "Gruppe"."Telefon_fest" AS "Telefon", "rel_Mit_Vor"."WahlDat"
FROM  "Vorstand_Posten"
LEFT JOIN "rel_Mit_Vor" ON "rel_Mit_Vor"."vorID" = "Vorstand_Posten"."ID" AND
  "rel_Mit_Vor"."EndDat" IS NULL
LEFT JOIN "Mitglied" ON "rel_Mit_Vor"."mitID" = "Mitglied"."ID"
LEFT JOIN "Gruppe" ON "Gruppe"."ID" = "Mitglied"."gruID"
ORDER BY "Vorstand_Posten"."ID" ASC
```

Funktion: Der aktuelle Vorstand wird für eine Anzeige im Bericht aufgelistet. Sämtliche Posten werden dargestellt, auch eventuell nicht besetzte Posten

Beschreibung: Die Anzeige sämtlicher Posten wird sicher gestellt, indem die Posten über LEFT JOIN an die anderen Tabellen gebunden werden. Leider führt dies bei den Versionen OOo 3.2.1, OOo 3.3 und LibreOffice 3.3 zu einer fehlerhaften Abfrage, die keine leeren Datumsfelder ausgibt. Die Version OOo 3.1.1 hingegen zeigt die Felder bei nicht besetzten Posten als leer an.

Vorstand_aktuell_lt_Satzung

```
SELECT "Vorstand_Posten"."Posten", "Mitglied"."Nachname", "Mitglied"."Vorname",
  "Gruppe"."Telefon_fest" AS "Telefon", "Strasse"."Strasse"||'
  '||"Adresse"."Nr"||', '||"Postleitzahl"."Postleitzahl"||' '||"Ort"."Ort" AS
  "Anschrift"
FROM  "Vorstand_Posten"
LEFT JOIN "rel_Mit_Vor" ON "rel_Mit_Vor"."vorID" = "Vorstand_Posten"."ID" AND
  "rel_Mit_Vor"."EndDat" IS NULL LEFT JOIN "Mitglied" ON "rel_Mit_Vor"."mitID"
  = "Mitglied"."ID"
LEFT JOIN "Gruppe" ON "Gruppe"."ID" = "Mitglied"."gruID"
LEFT JOIN "Adresse" ON "Adresse"."ID" = "Gruppe"."adrID"
LEFT JOIN "Strasse" ON "Strasse"."ID" = "Adresse"."strID"
LEFT JOIN "Postleitzahl" ON "Postleitzahl"."ID" = "Adresse"."posID"
LEFT JOIN "Ort" ON "Ort"."ID" = "Postleitzahl"."ortID"
ORDER BY "Vorstand_Posten"."ID" ASC LIMIT 17
```

Funktion: Weitergabe der Vorstandsdaten an Bezirks- und Landesverbände, für Rundschreiben an die Mitglieder etc. Posten, Personen, Adressen und Telefonverbindungen werden für einen Berciht aufbereitet.

Beschreibung: Mit „Limit“ werden die wiedergegebenen Daten auf die Posten begrenzt, die ausgegeben werden sollen. Die ersten 17 Posten in der Vorstands-Posten-Übersicht sind die Posten, die in diesem Fall für den eigentlichen Vorstand stehen und keine Übungsleiter o.ä. Beinhalten.