LibreOffice
The Document Foundation

Base

Kapitel 2
Datenbank erstellen

Copyright

Mitwirkende/Autoren

Robert Großkopf | Jost Lange | Michael Niedermair

Jochen Schiffers

Rückmeldung (Feedback)

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

Vorsicht

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

Datum der Veröffentlichung und Softwareversion

Veröffentlicht am 01.02.2023. Basierend auf der LibreOffice Version 7.5.

Inhalt

Allgemeines bezüglich der Erstellung einer Datenbank

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

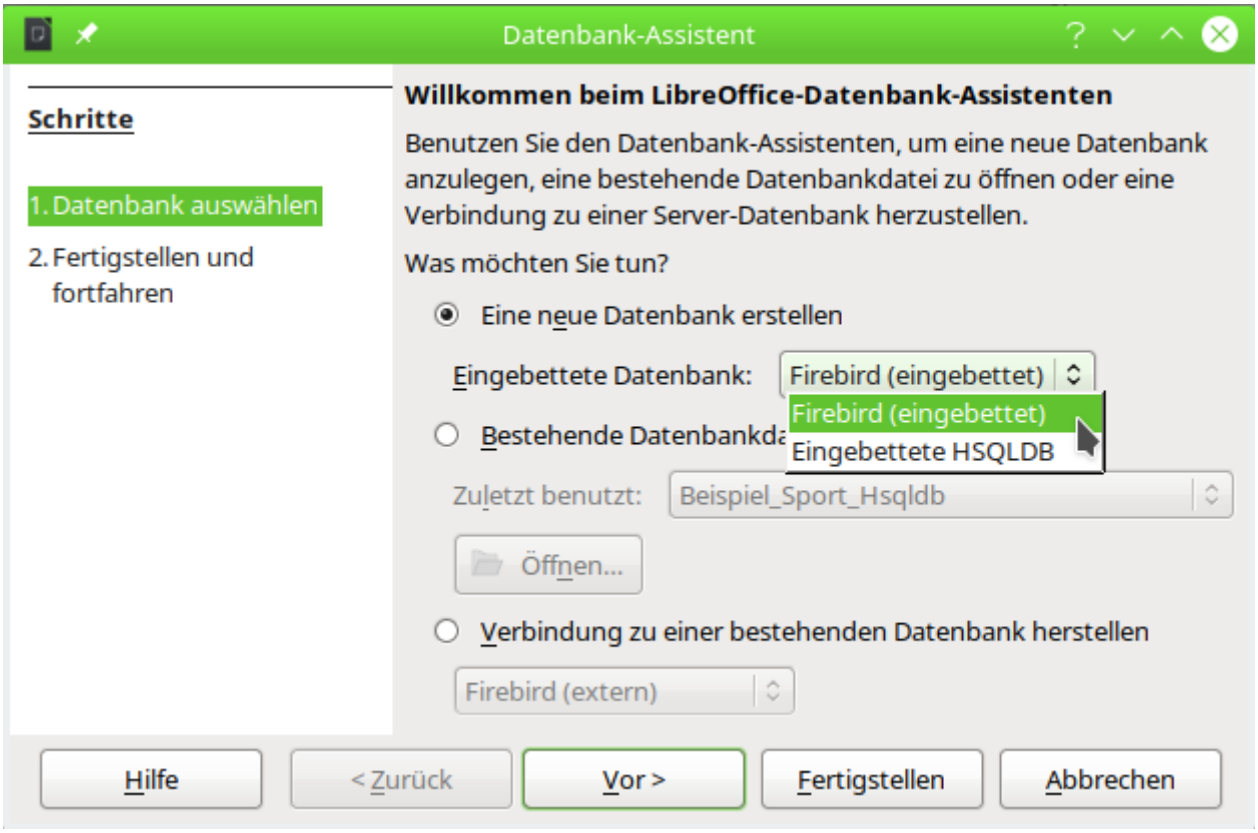
Neue Datenbank als interne Datenbank

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

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

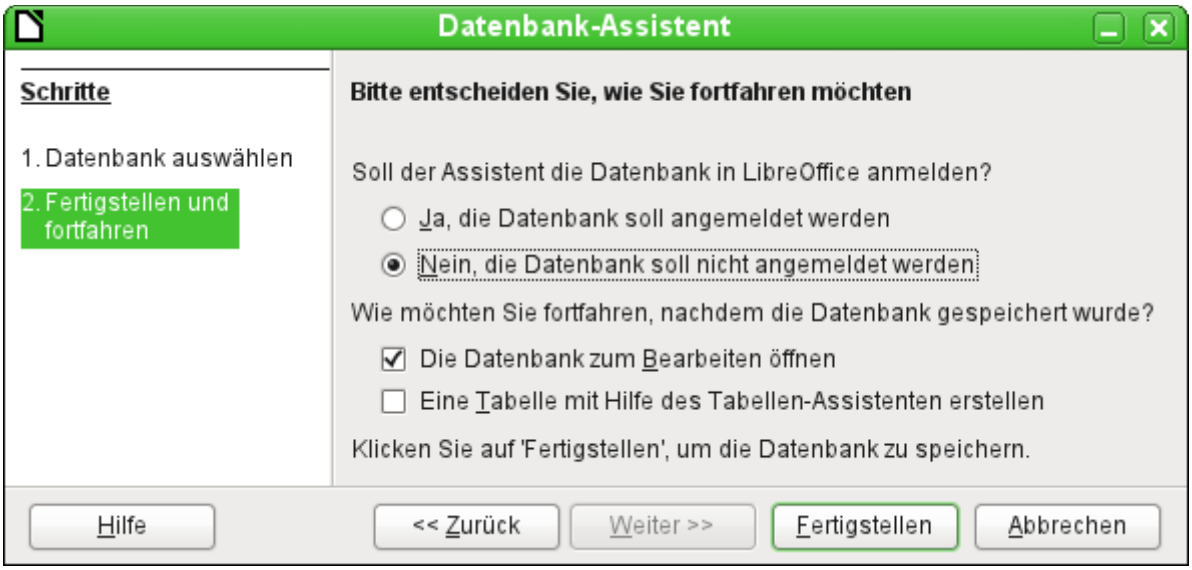
Hinweis	Die Nutzung großer Datenmengen kann bei **Java** zu Geschwindigkeitsverlusten führen. Dies kann von der verwendeten Java-Version abhängen. Ein einfacher Test bei direktem Ausführen von SQL ergibt für eine Tabelle von 15 Spalten und 30.000 Datensätzen deutliche Unterschiede für die Zeit, die für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

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



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

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



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

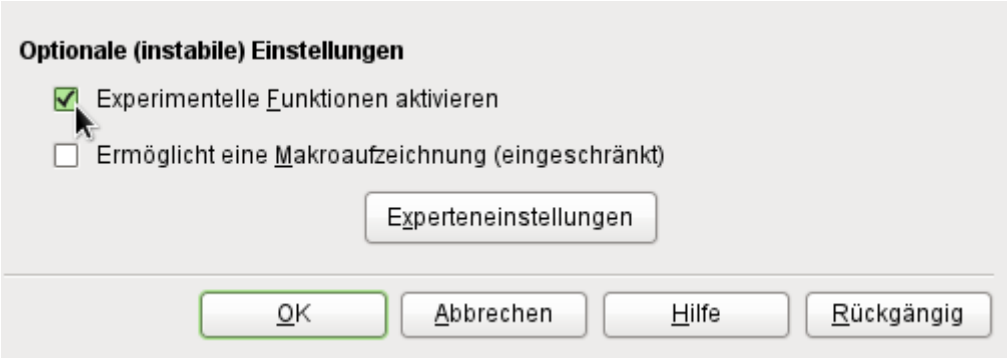
auch zu einem späteren Zeitraum erfolgen. Deshalb ist an dieser Stelle erst einmal das Optionsfeld **Nein, die Datenbank soll nicht angemeldet werden**, ausgewählt.

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

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

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

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



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

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

Vorsicht

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

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

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

Zugriff auf externe Datenbanken

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

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

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

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

- Firebirddatei
- Tabellendokument
- Text
- sowie verschiedene Adressbücher

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

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

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

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

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

MySQL/MariaDB-Datenbanken

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

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

Erstellen eines Nutzers und einer Datenbank

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

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

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

Direkte Verbindung zu MySQL/MariaDB

Für die **MariaDB** und die **MySQL-Datenbank** existiert eine direkte Verbindung mit Base. Diese Verbindung funktioniert auf jeden Fall mit der MariaDB, eventuell aber nicht einwandfrei mit neueren Versionen von MySQL ab Version MySQL 8. Das liegt daran, dass aus Lizenzgründen vom MySQL-C++-Verbinder zum MariaDB-C-Verbinder gewechselt wurde. Der MariaDB-C-Verbinder steht unter der LGPL-Lizenz, die kompatibel zu der LibreOffice-Lizenz ist.[1]

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

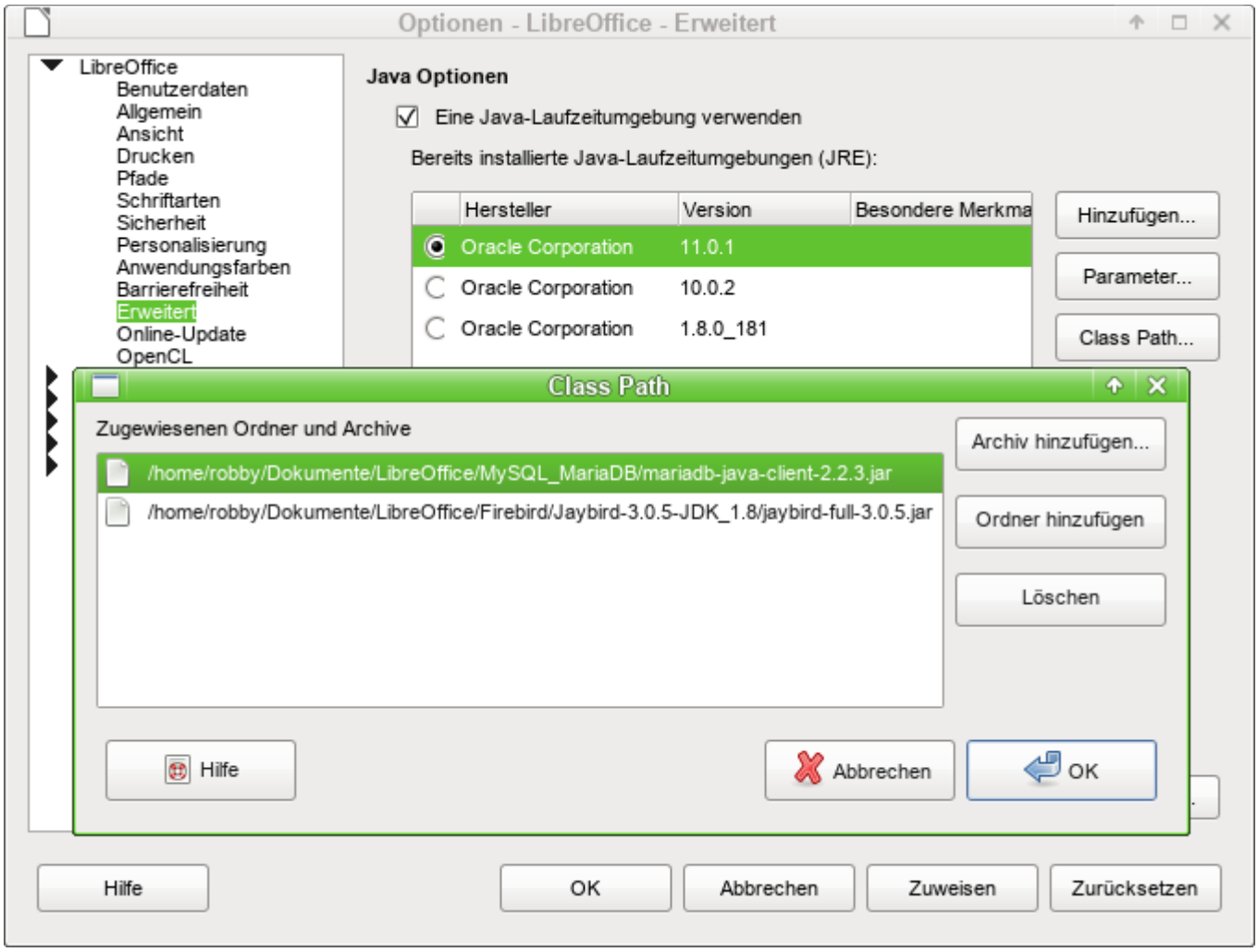
MySQL/MariaDB-Verbindung über JDBC

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

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

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

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



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

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

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

MySQL/MariaDB-Verbindung über ODBC

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

Nach Installation der entsprechenden Software kann es passieren, dass LO den Dienst verweigert, weil es die libodbc.so.1 nicht findet. Hier existiert in den meisten Systemen inzwischen die

libodbc.so.2 . Auf diese Datei muss ein entsprechender Verweis mit dem Namen libodbc.so.1 in dem gleichen Verzeichnis abgespeichert werden.

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

odbcinst.ini

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

odbc.ini

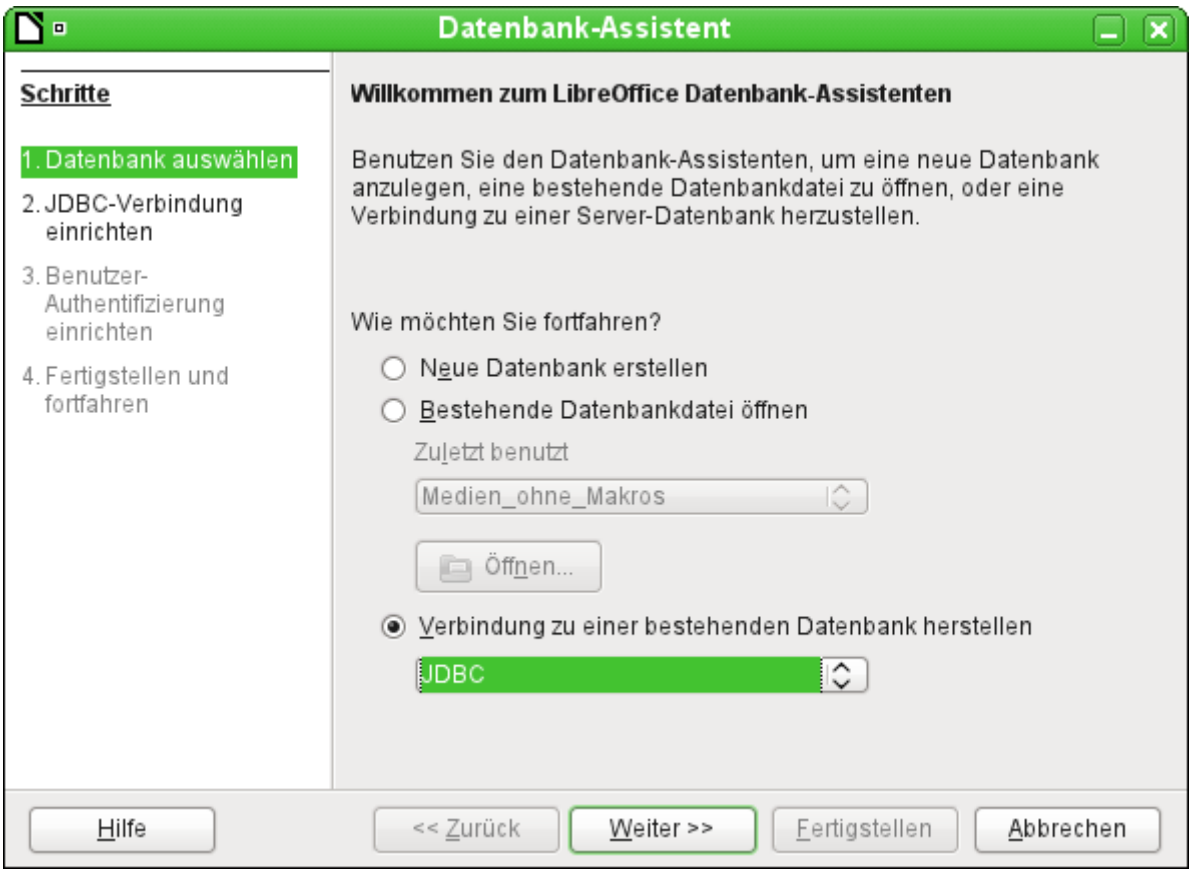
```
001 [MySQL-test]
002 Description = MySQL database test
003 Driver = MySQL
004 Server = localhost
005 Database = libretest
006 Port = 3306
007 Socket =
008 Option = 3
009 Charset = UTF8
```

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

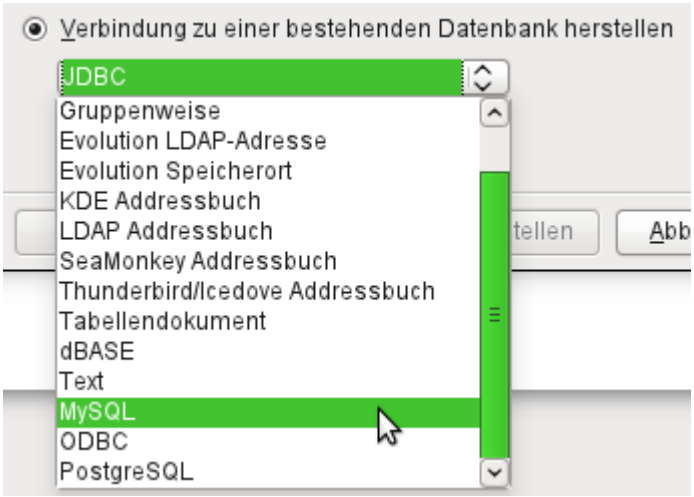
Details zu den Verbindungsparametern sind im *MySQL-Referenzhandbuch* zu finden.

Verbindung zur MySQL-Datenbank über den Datenbank-Assistenten

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



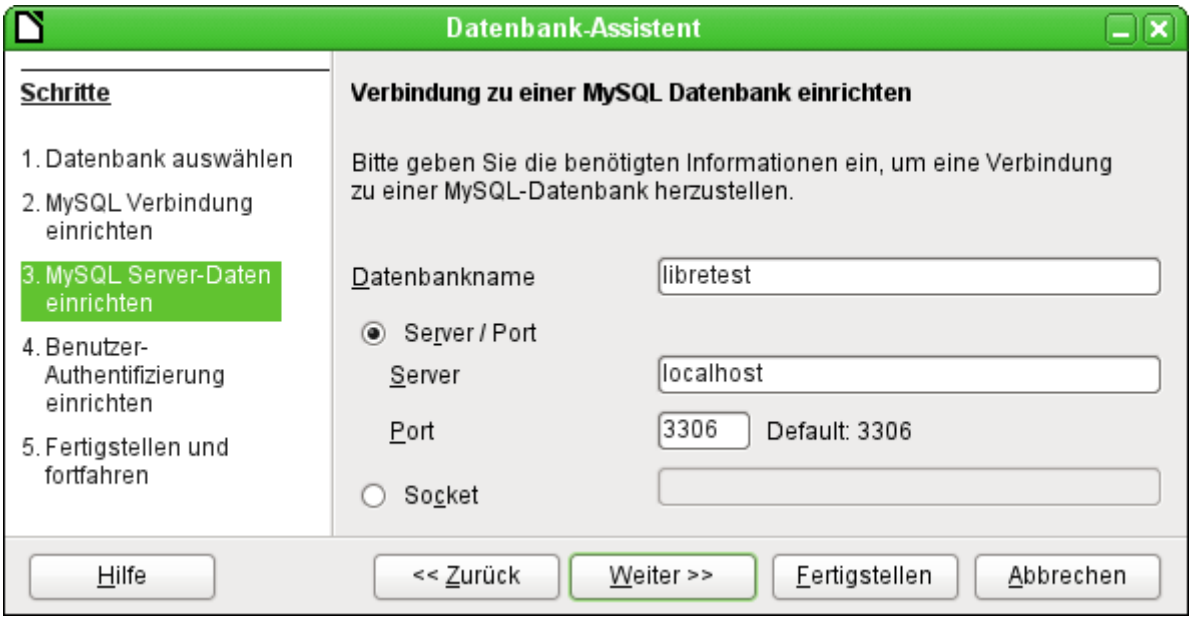
Über «*Neue Datenbank erstellen*» ist nur die Erstellung einer Datenbank im internen HSQLDB-Format möglich. Die Zusammenarbeit mit anderen Datenbanken kann nur erfolgen, wenn die Datenbank selbst bereits existiert. Dazu muss also **Verbindung zu einer bestehenden Datenbank herstellen** gewählt werden.



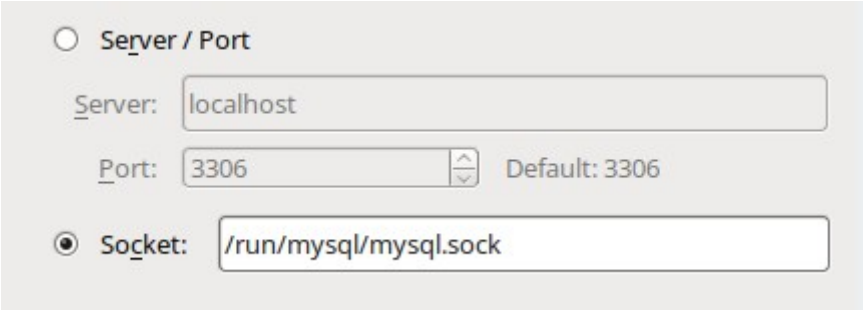
Hier wird aus den, teilweise betriebssystemspezifischen, Datenbanken die MySQL-Variante ausgewählt.

Die direkte Verbindung

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



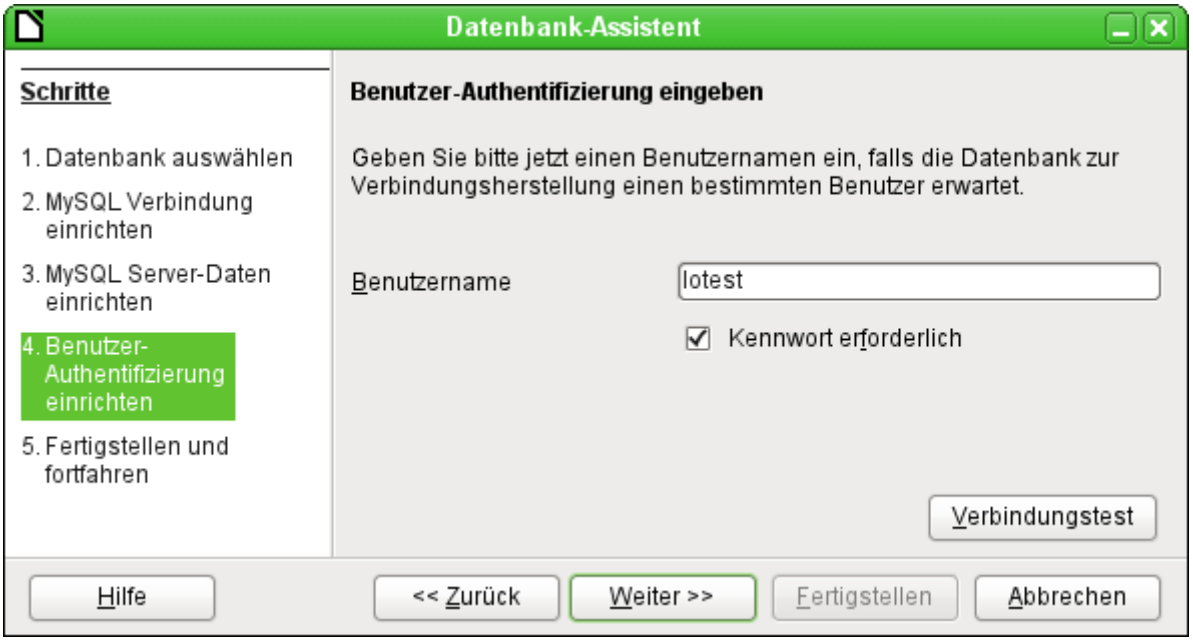
Der Datenbankname muss bekannt sein. Befindet sich der Server auf dem gleichen Rechner wie die Benutzeroberfläche, unter der die Datenbank erstellt wird, so kann als Server «localhost» gewählt werden. Ansonsten kann die IP-Adresse (z.B. 192.168.0.1) oder auch je nach Netzwerkstruktur der Rechnername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



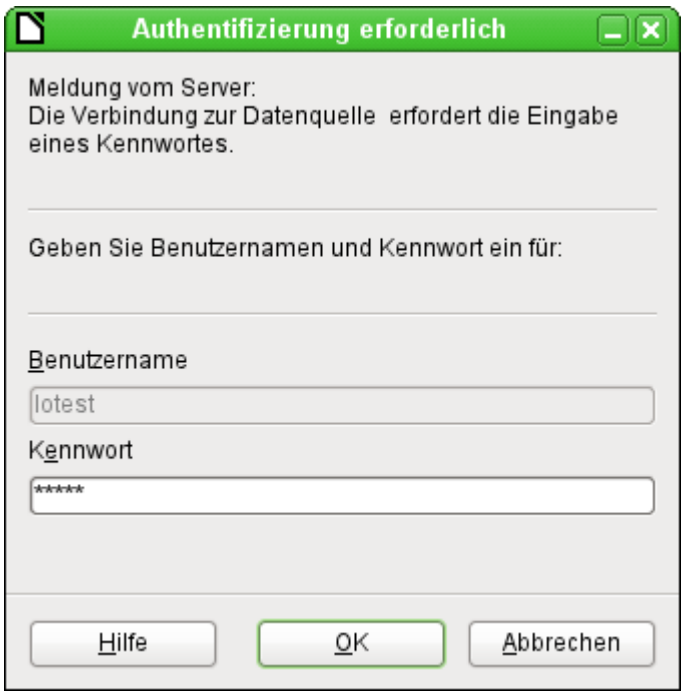
Alternativ zu den Serverangaben kann (unter Linux) auch direkt der Socket eingegeben werden. Der Standardsocket kann dafür aus der Datei /etc/my.cnf ausgelesen werden.

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

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

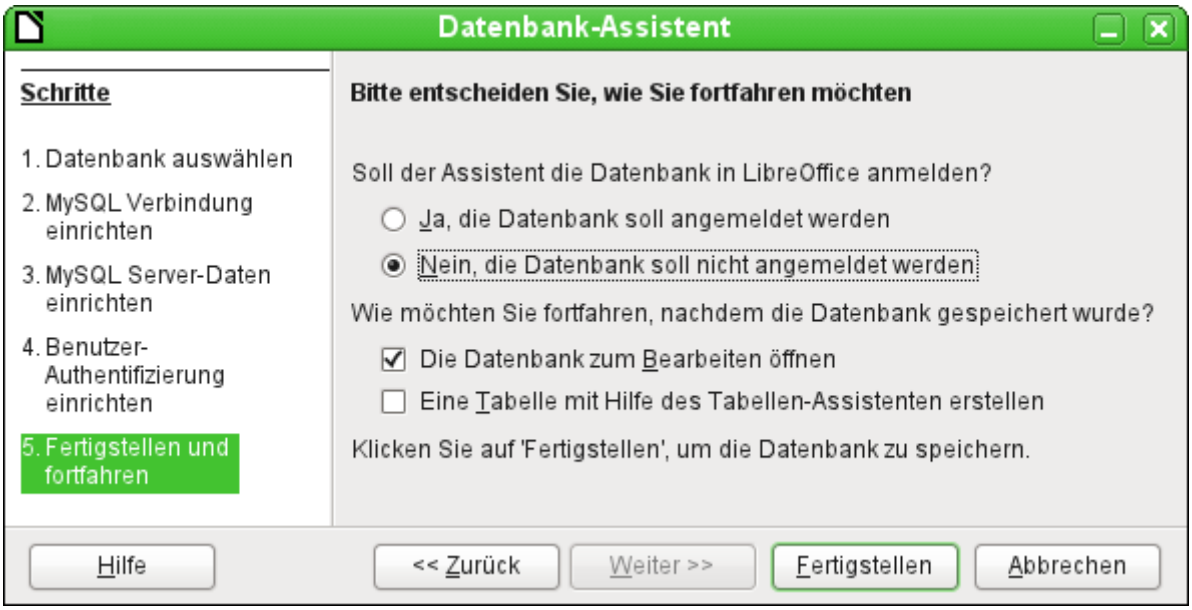


Jede über das Netz erreichbare Datenbank sollte mit einem Benutzernamen und einem Kennwort geschützt sein. Hier wird direkt getestet, ob die Verbindung klappt. Wichtig ist natürlich, dass der entsprechende Benutzer in MySQL bzw. der MariaDB entsprechend für den benannten Server eingerichtet wurde.



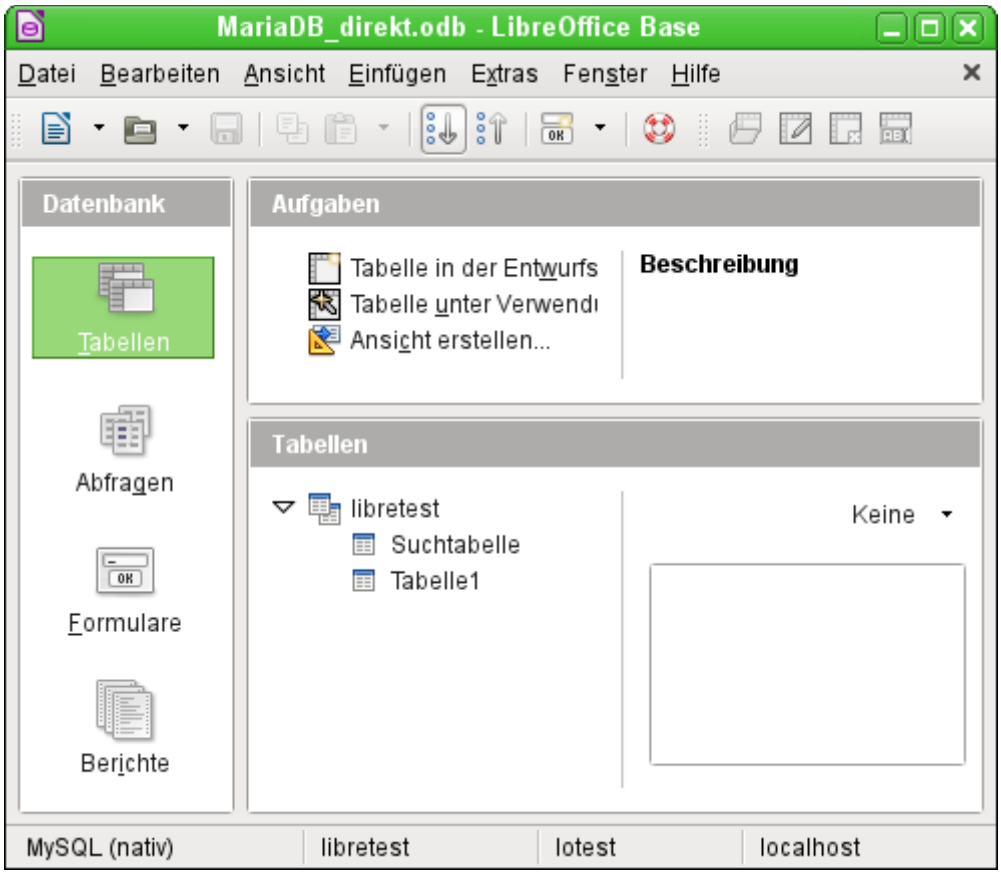
Dieses Fenster erscheint anschließend bei jedem Start der Datenbankdatei, wenn das erste Mal auf die MySQL-Datenbank zugegriffen wird.

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



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

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



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

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

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

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

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

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

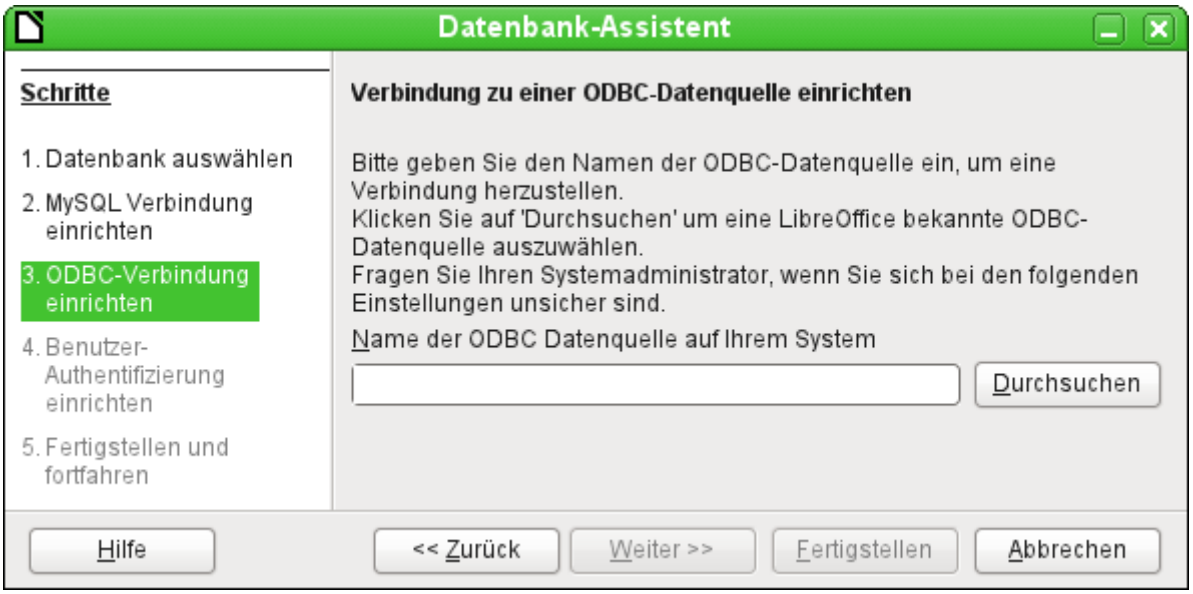
Hier ist es also auf jeden Fall notwendig, der Kombination aus Datenbankname und Tabellenname mit dem Zusatz «AS» einen alternativen Alias-Namen zuzuweisen. Genaueres siehe dazu in dem Kapitel «Verwendung eines Alias in Abfragen».

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

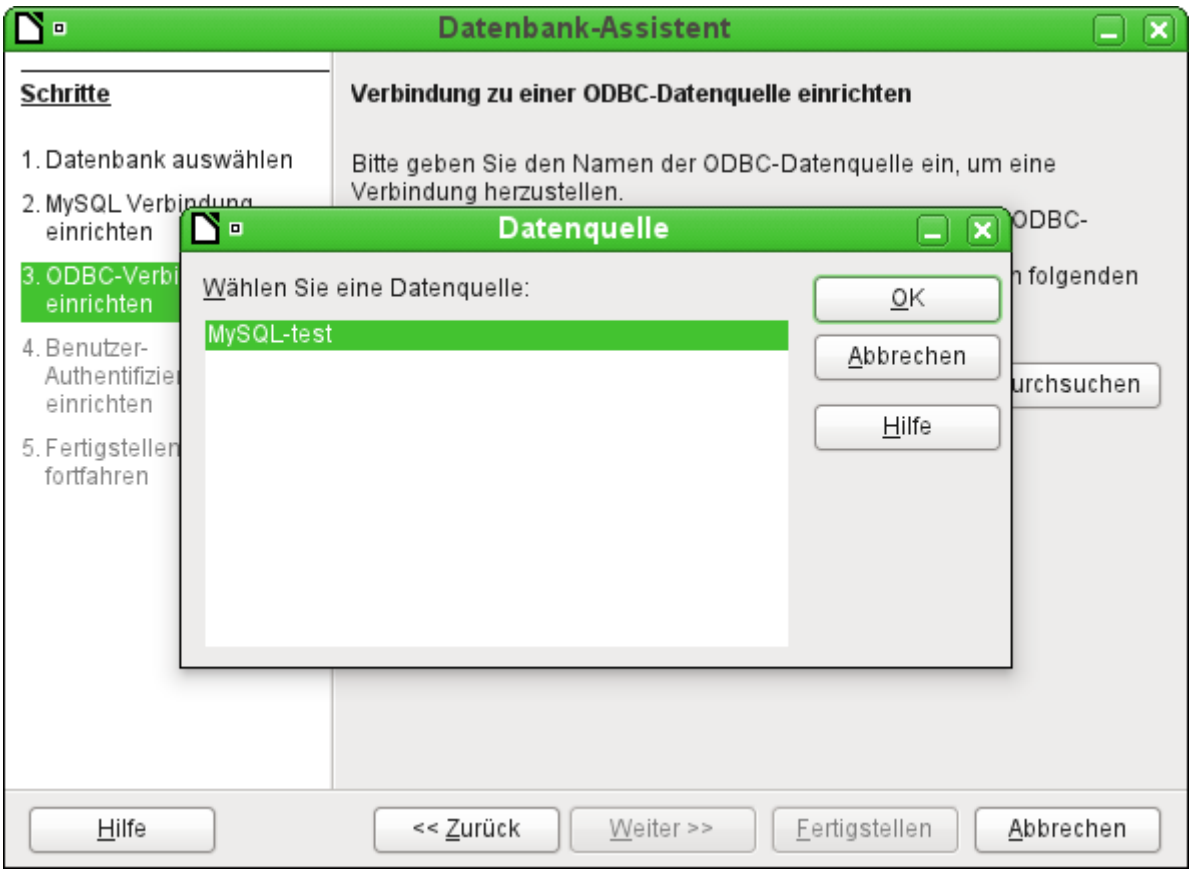
Tabellen können in der Datenbank erstellt und gelöscht werden. Automatisch hoch zählende Zahlenfelder («Autowert») funktionieren und lassen sich auch bei der Tabellenerstellung auswählen. Sie starten bei MySQL mit dem Wert '1'.

Die ODBC-Verbindung

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



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

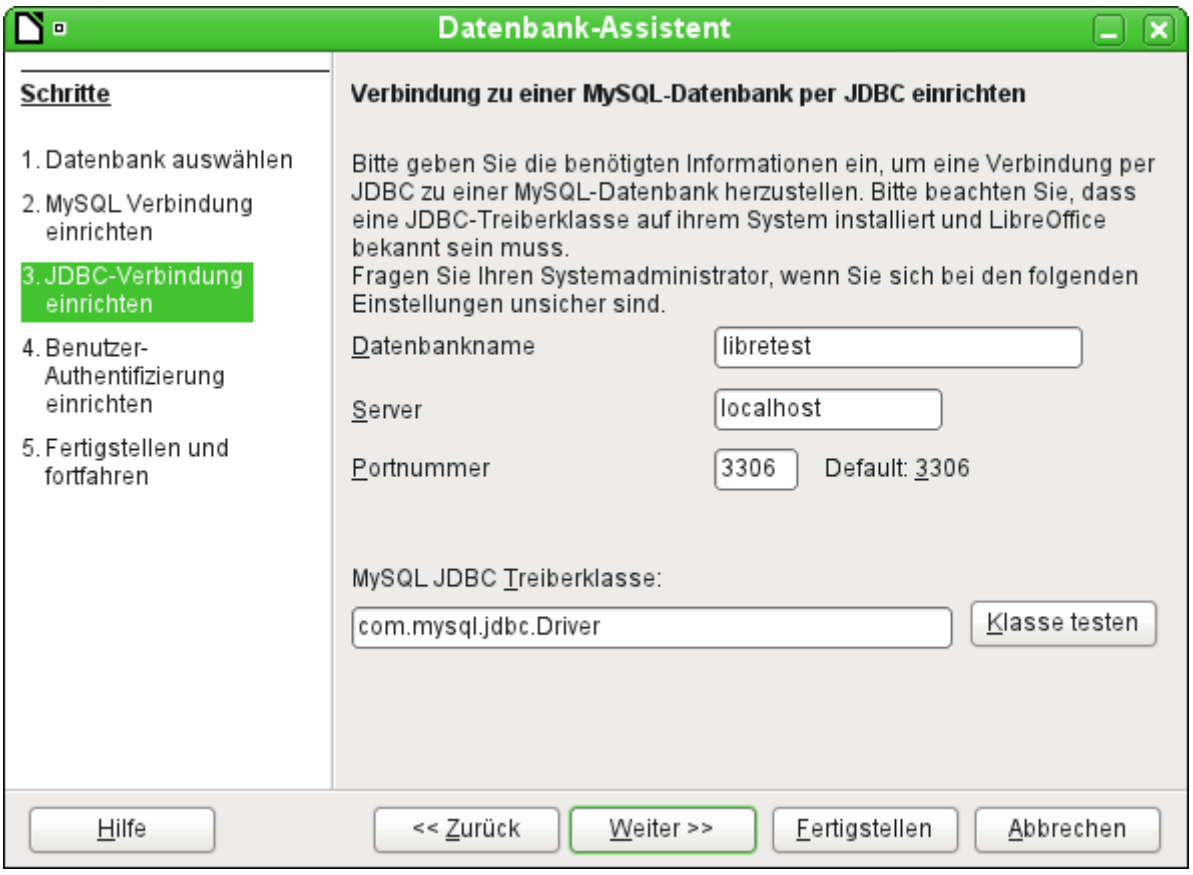


Beim Test erscheint der Name aus der «odbc.ini». Auch wenn hier wieder nur mit einer Datenbank verknüpft wird, können andere Tabellen des MySQL-Servers sehr wohl gelesen werden.

Schritt 4 und 5 laufen wieder identisch zur Direktverbindung ab.

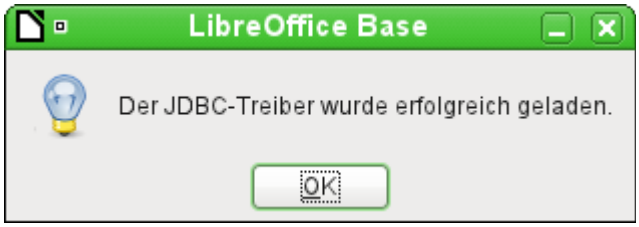
Die JDBC-Verbindung

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



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

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



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

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

Hinweis

Der JDBC-Zugang mit MySQL/MariaDB kann auch direkt über die Auswahl **JDBC** statt des Untermenüs **MySQL → JDBC** erfolgen. Dies kann besonders dann sinnvoll sein, wenn dem Treiber Parameter mitgegeben werden sollen.

Ohne Parameter trennt z.B. der JDBC-Treiber wie auch der direkte Treiber zu einer MySQL-Datenquelle im Internet nach recht kurzen Pausenzeiten die Verbindung. Mit der folgenden Einstellung wird dies vermieden:

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

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

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

Dies bietet alle erforderlichen Einstellungen, zusätzlich auch noch die Einstellung des Zeichensatzes.

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

Verbindung zu MySQL/MariaDB über das Internet

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

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

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

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

```
001 GLOBAL boStop AS BOOLEAN
```

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

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

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

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

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

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

Zugriff auf gespeicherte Prozeduren in MySQL/MariaDB

Auf dem Datenbankserver können neben Tabellen und Ansichten auch Prozeduren gespeichert werden. Werden diese Prozeduren direkt auf der Konsole von MySQL aufgerufen, so können dort zum Teil Tabellenansichten ähnlich einer sonst in Base sichtbaren Ansicht («View») abgerufen werden. In Base sind diese Tabellenansichten nicht sichtbar. Ein Aufruf der Prozeduren unter **Extras → SQL** ist problemlos möglich, nur erfolgt leider keine Ausgabe der anzuzeigenden Inhalte. Um Inhalte der gespeicherten Prozeduren («Stored Procedures») zu sehen kann der folgende Weg beschritten werden:

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

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

PostgreSQL

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

Erstellen eines Nutzers und einer Datenbank

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

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

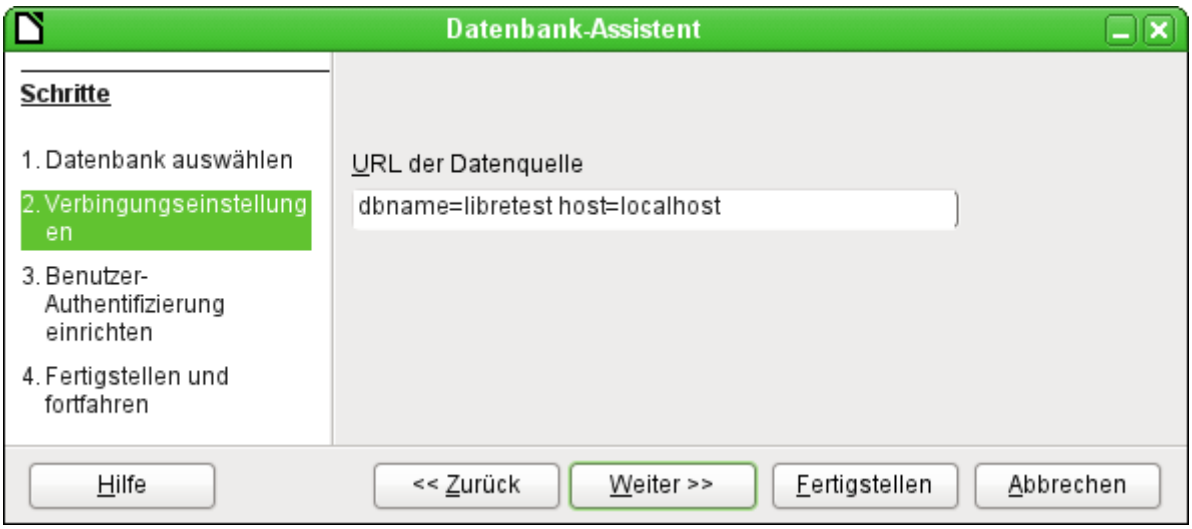
Der Nutzer «lotest» hat nicht die gleichen Rechte wie der Nutzer «postgres». Meldet sich der Nutzer «postgres» direkt beim System über **`su postgres`** an, so kann er beispielsweise zusätzliche Schemas für eine Datenbank erstellen:

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

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

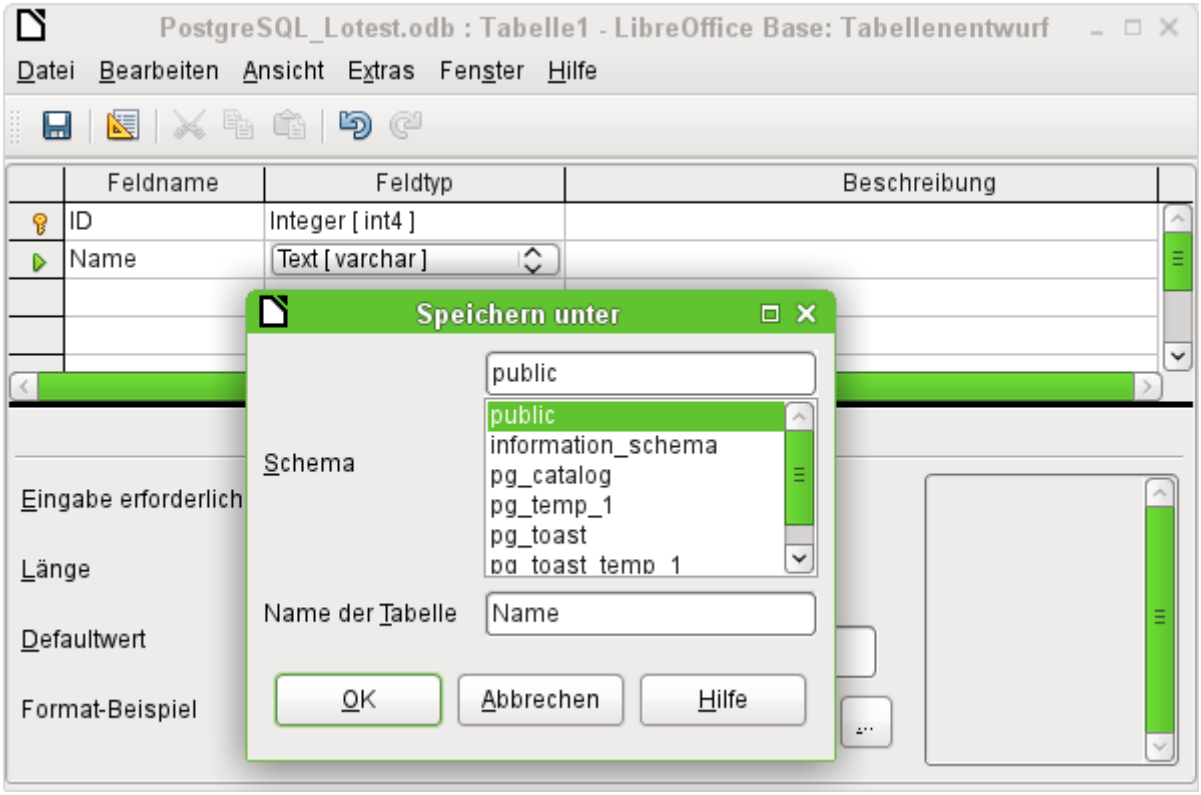
Direkte Verbindung zu PostgreSQL

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



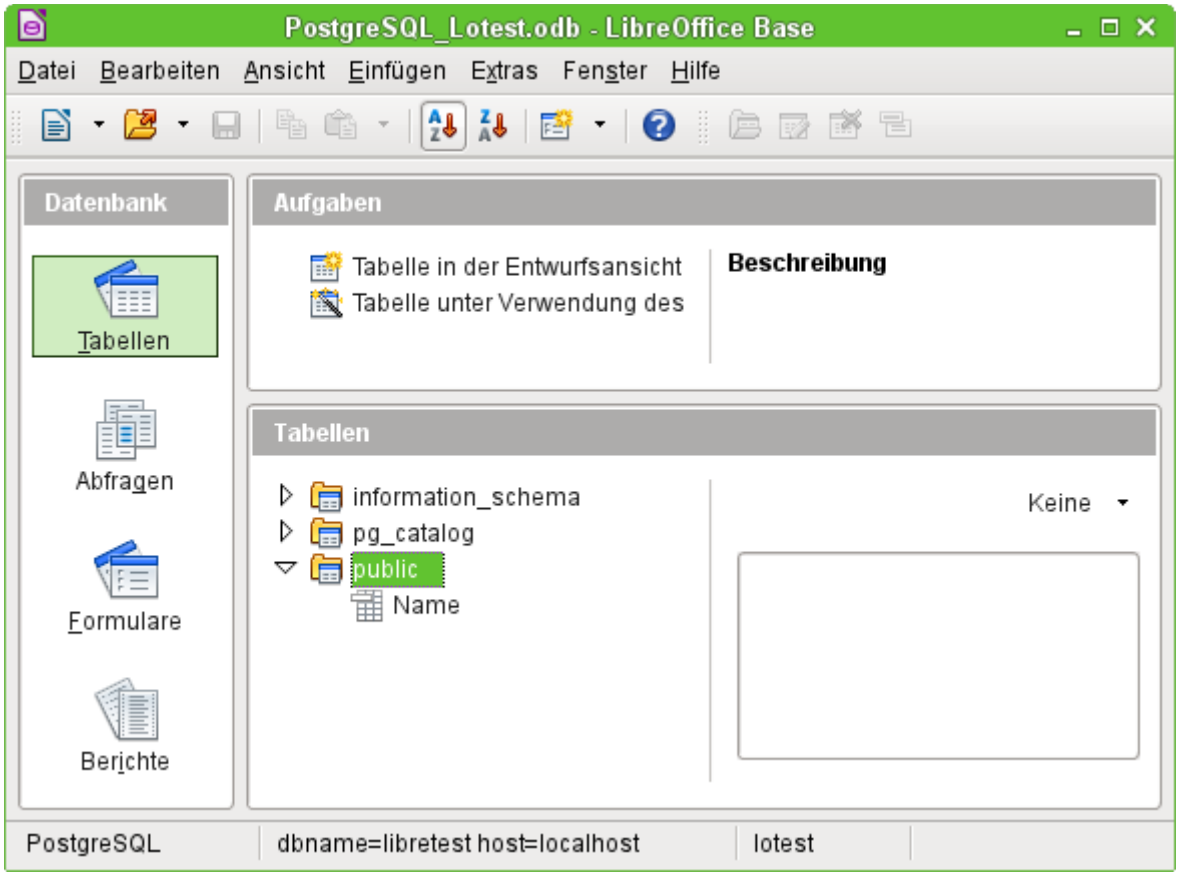
Die Benutzer-Authentifizierung sieht genauso aus wie die unter MySQL. Neben «dbname» und «host» können hier noch «port», «connect_timeout» und andere Einträge erstellt werden.

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



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

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



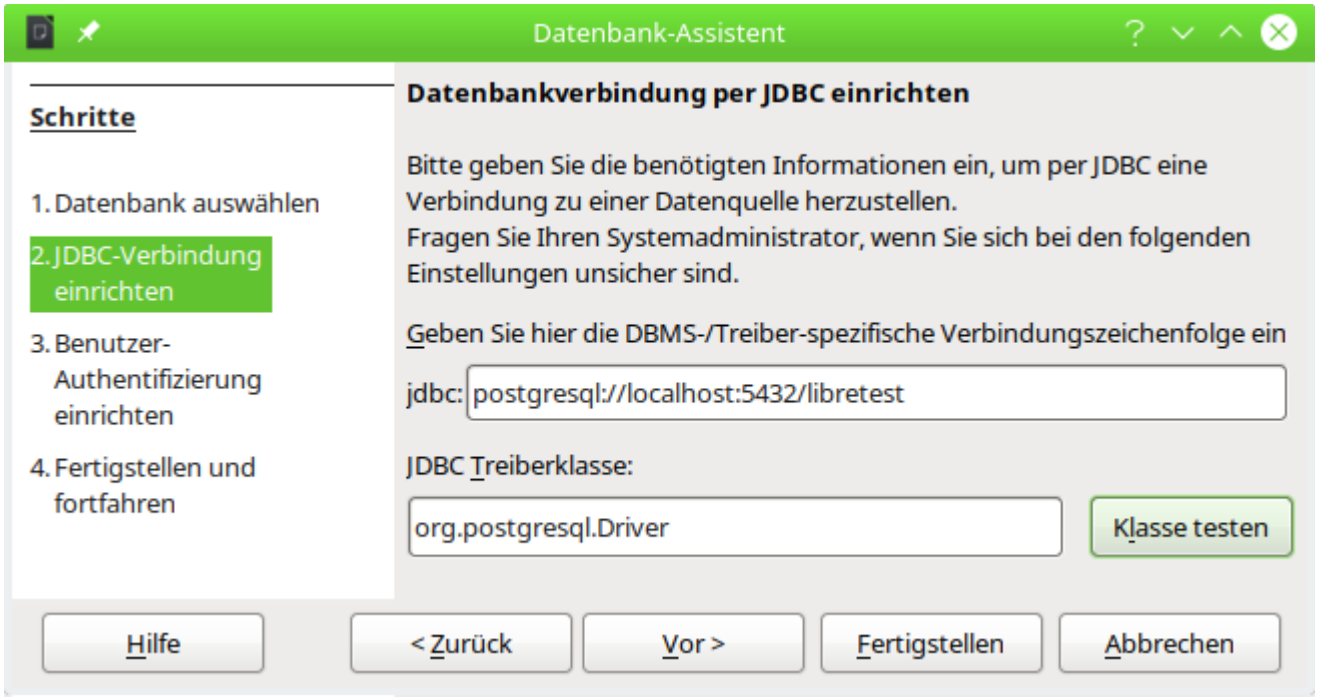
In der Tabellenübersicht von PostgreSQL erscheinen die unterschiedlichen Schemata. Im Schema «public» ist die abgespeicherte Tabelle «Name» zu sehen.

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

PostgreSQL-Verbindung über JDBC

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

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



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

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

PostgreSQL-Verbindung über und ODBC

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

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

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

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

odbcinst.ini

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

odbc.ini

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

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

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

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

Verbindung zu PostgreSQL über das Internet

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

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

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

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

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

Autoincrement-Werte bei PostgreSQL

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

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

Auch über

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

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

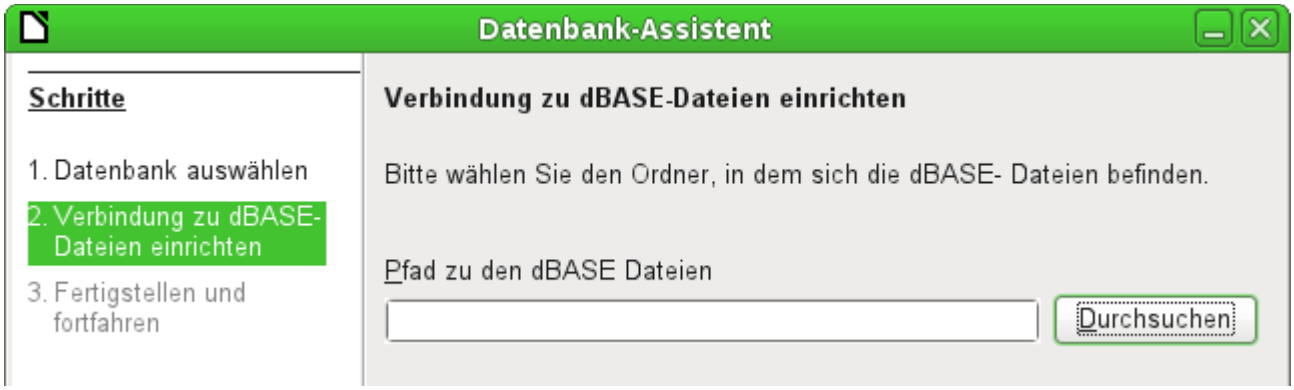
Hier wird ein automatisch hoch zählendes Feld "ID" als Primärschlüssel definiert. Die entsprechende Eingabe unter **Extras → SQL** erzeugt die Tabelle "Test". Wird anschließend über **Ansicht → Tabellen aktualisieren** die Anzeige der Tabelle ermöglicht, so kann die Tabelle in dem GUI-Editor anschließend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

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

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

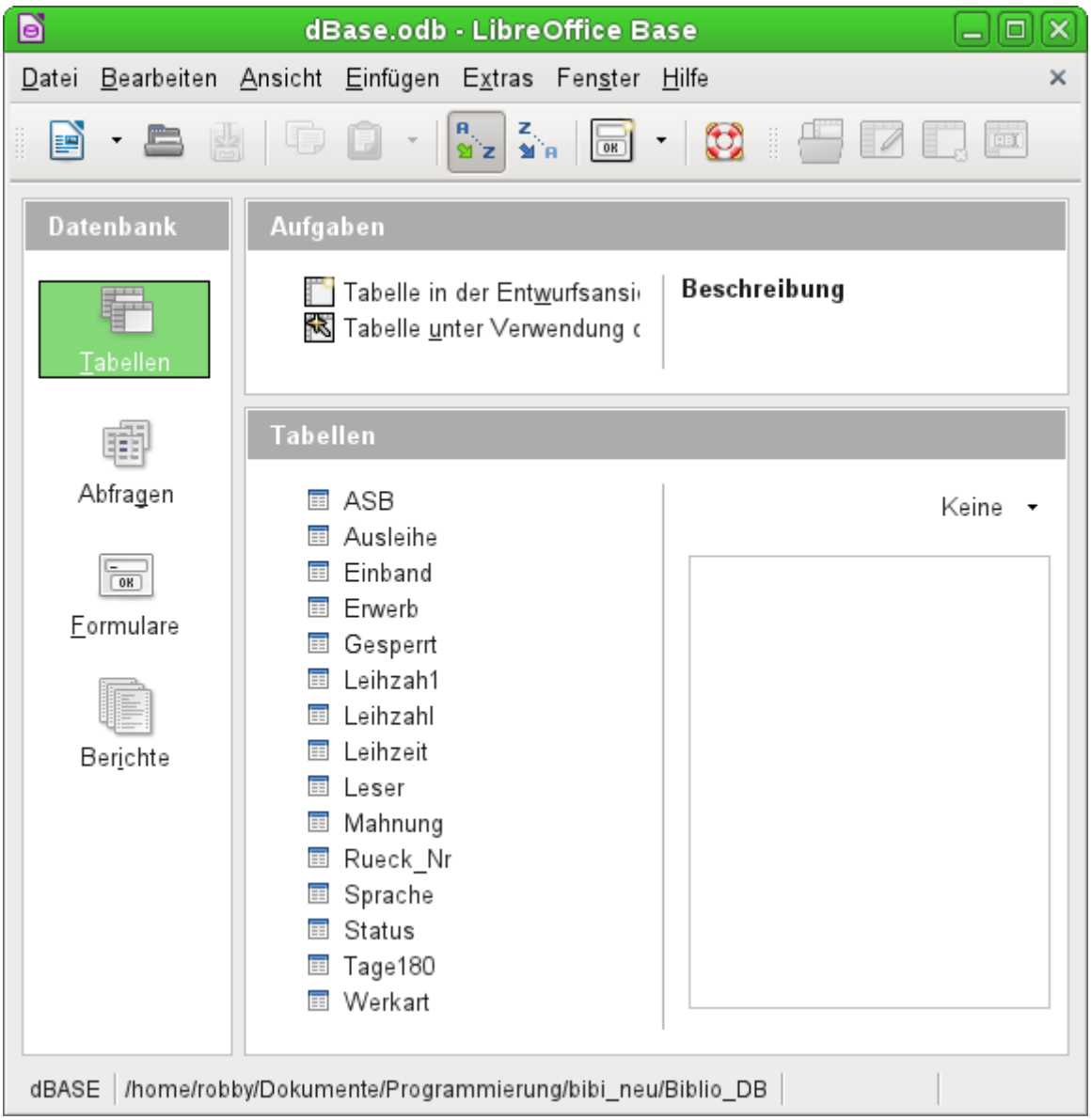
dBase-Datenbanken

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



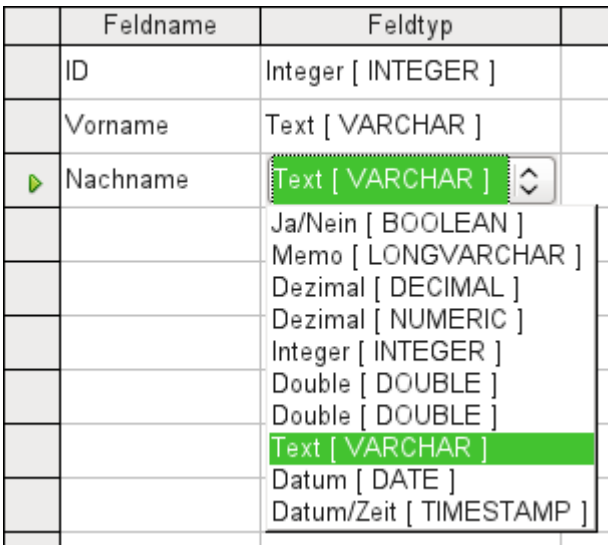
Die Verbindung wird zu einem Verzeichnis hergestellt. Alle *.dbf-Dateien, die sich in diesem Verzeichnis befinden, werden anschließend angezeigt und können über Formulare miteinander verbunden werden.

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



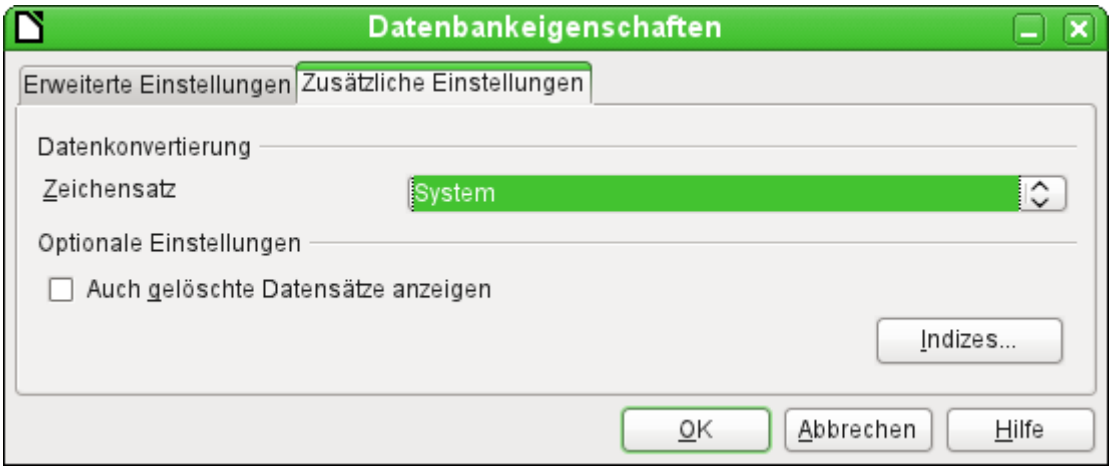
Tabellen können in Base neu erstellt werden. Sie werden dann einfach als neue Dateien in das angegebene Verzeichnis geschrieben.

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



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

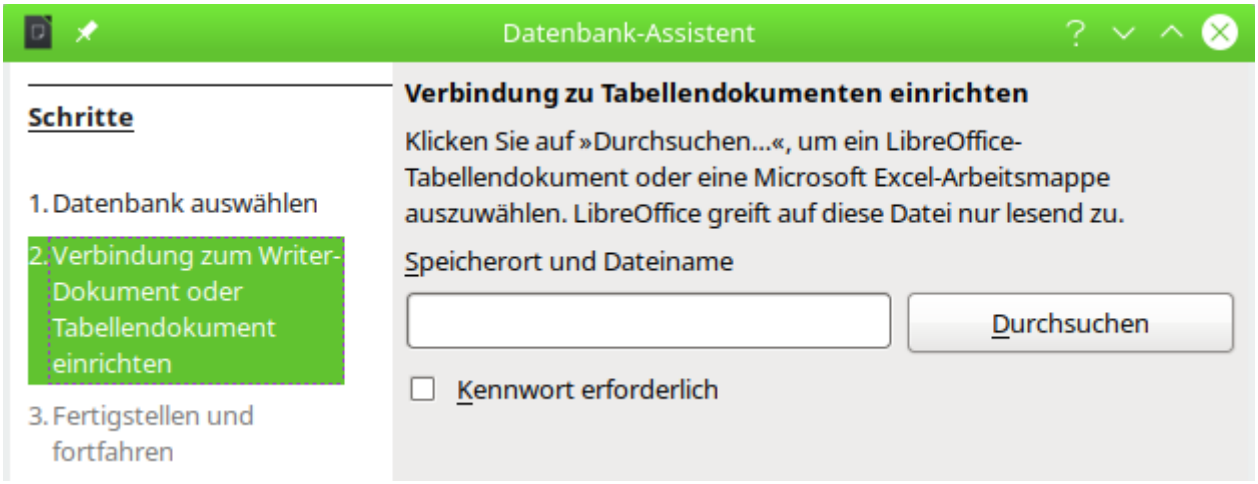
Base übernimmt einfach die Codierung, die dem Betriebssystem entspricht. Alte dBase-Dateien weisen dadurch leicht Fehler beim Import von Sonderzeichen auf. Der Zeichensatz kann anschließend über **Bearbeiten → Datenbank → Eigenschaften → Zusätzliche Einstellungen** entsprechend korrigiert werden:



Tabellendokumente und Tabellen in Writer-Dokumenten

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

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



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

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

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

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

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

Daten in Calc bearbeiten und in Base aktualisieren

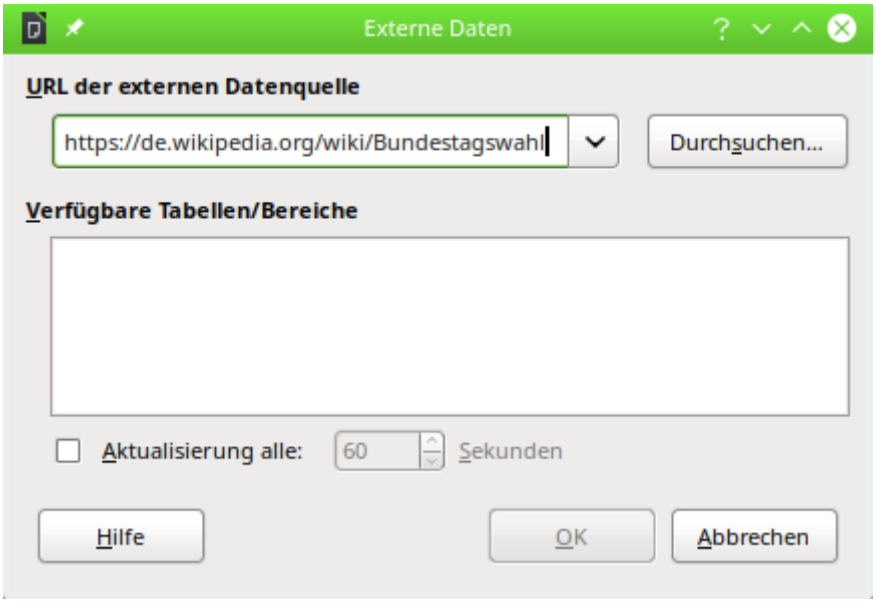
Innerhalb von Base sind Daten aus Calc nicht editierbar. Sollen trotzdem Daten verändert werden, so sind die folgenden Schritte zu beachten:

1. Die Base-Datei muss mit einem Zugriff auf die Tabellen geöffnet sein, bevor die damit verbundene Calc-Datei geöffnet wird.
2. Die Calc-Datei wird mit einem Schreibschutz geöffnet. Hier auf Dokument bearbeiten klicken.
3. Werden Daten im vorhandenen Datenbereich geändert, so muss in der Base-Tabelle der Button zum Aktualisieren (oder **Daten → Aktualisieren**) betätigt werden. Die geänderten Daten sind jetzt in der Base-Datei zu sehen. Vorsicht! Die Daten sind in Calc nicht abgespeichert.
4. Werden Daten unterhalb des Bereiches in einer neuen Zeile hinzugefügt, so bekommt die Tabelle in Base von der zusätzlichen Zeile nichts mit. In diesem Fall die Tabelle schließen und **Ansicht → Tabellen aktualisieren** im Hauptmenü der Base-Datei betätigen. Danach die Tabelle wieder öffnen und die zusätzliche Zeile erscheint.

Tabellen aus dem Internet über Calc automatisch aktualisiert in Base auslesen

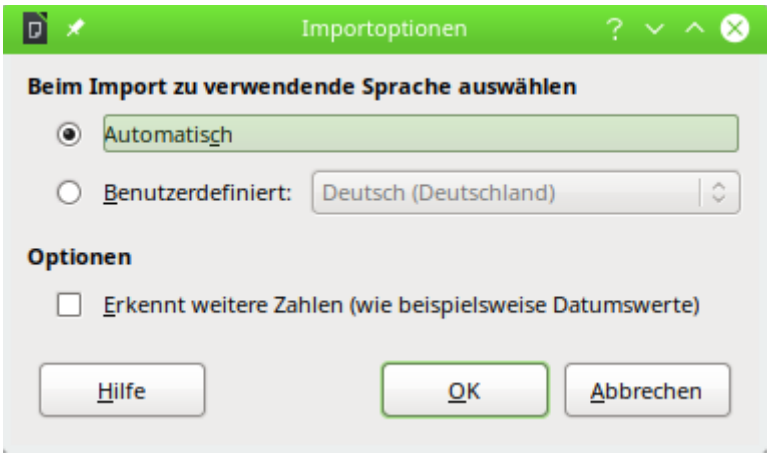
In Calc ist es möglich, auf externe Daten aus dem Internet zuzugreifen. Diese Daten lassen sich mit einer beständigen Aktualisierung versehen. So ist es auch möglich, mit Base eben über Calc auf solche Daten zuzugreifen.

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

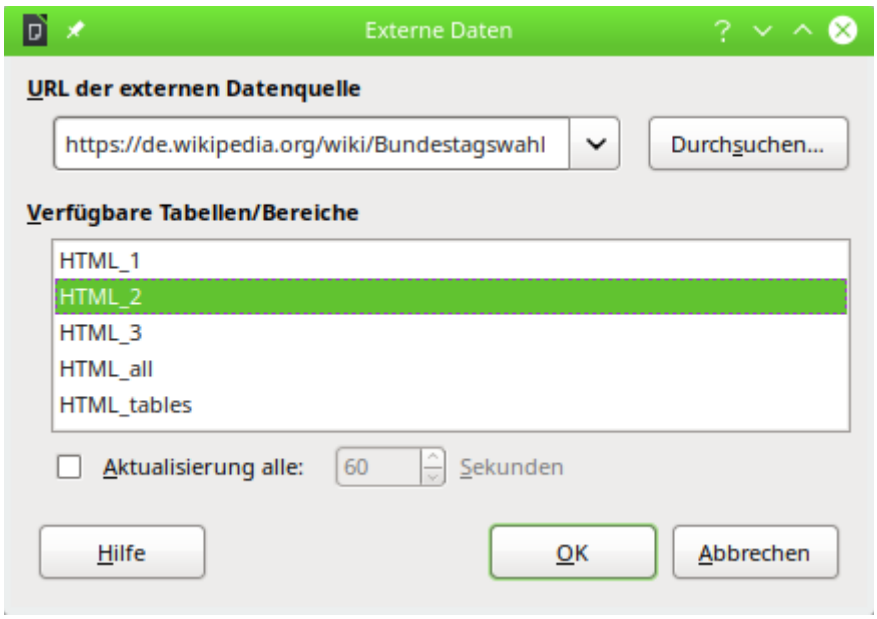


Hier wurde als Quelle der Wikipediaartikel zu Bundestagswahlen ausgesucht. Bei dem Artikel lohnt sich zwar nicht die Aktualisierung, aber das Prinzip lässt sich daran auch nachvollziehbar erklären.

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

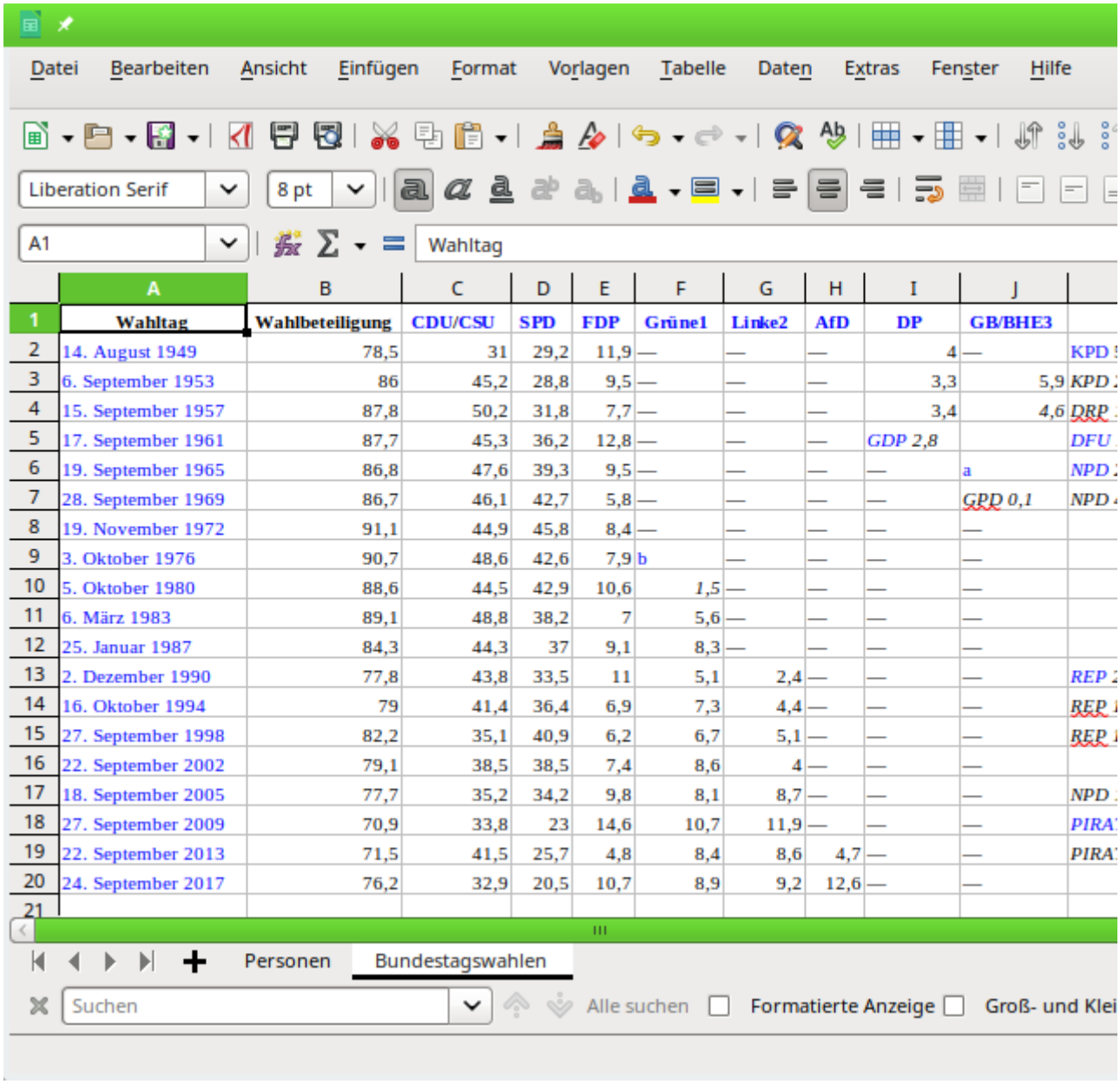


Wenn Calc entsprechende Informationen aus der Seite geladen hat erscheinen zuerst die Importoptionen. Da die Tabellen in Wikipedia in dem gewählten Beispiel sowieso eine Mischung aus Text und Zahlen darstellen wird auch Base daraus nur Textfelder machen. Deswegen ist hier z. B. das Erkennen weiterer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



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

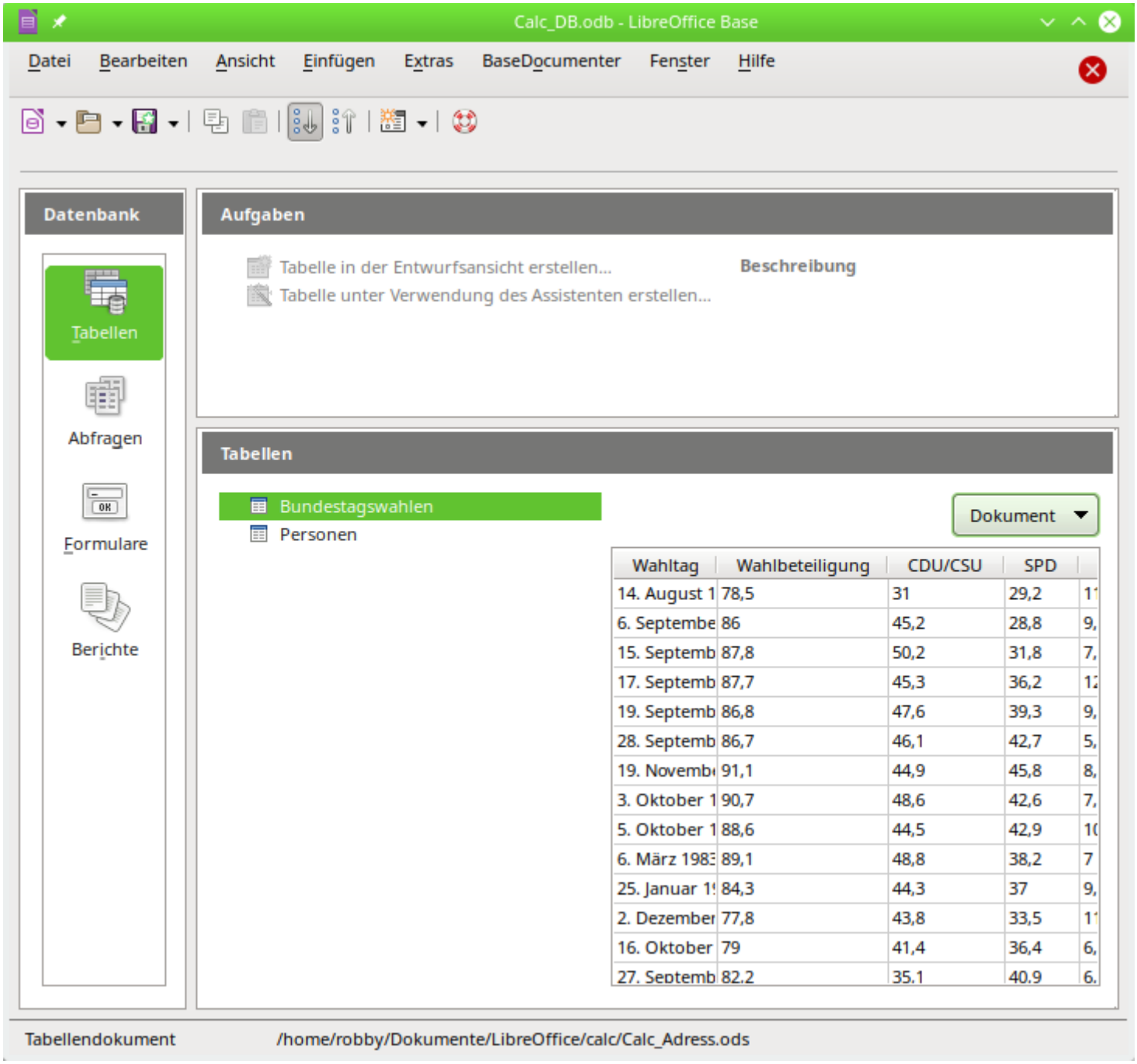
Jetzt könnte noch die automatische Aktualisierung eingestellt werden, so dass z. B. alle 60 Sekunden im Netz nachgeschaut wird, ob die Daten noch Bestand haben. So etwas macht wohl eher bei Aktienkursen als bei einer Tabelle zur Bundestagswahl Sinn.



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

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

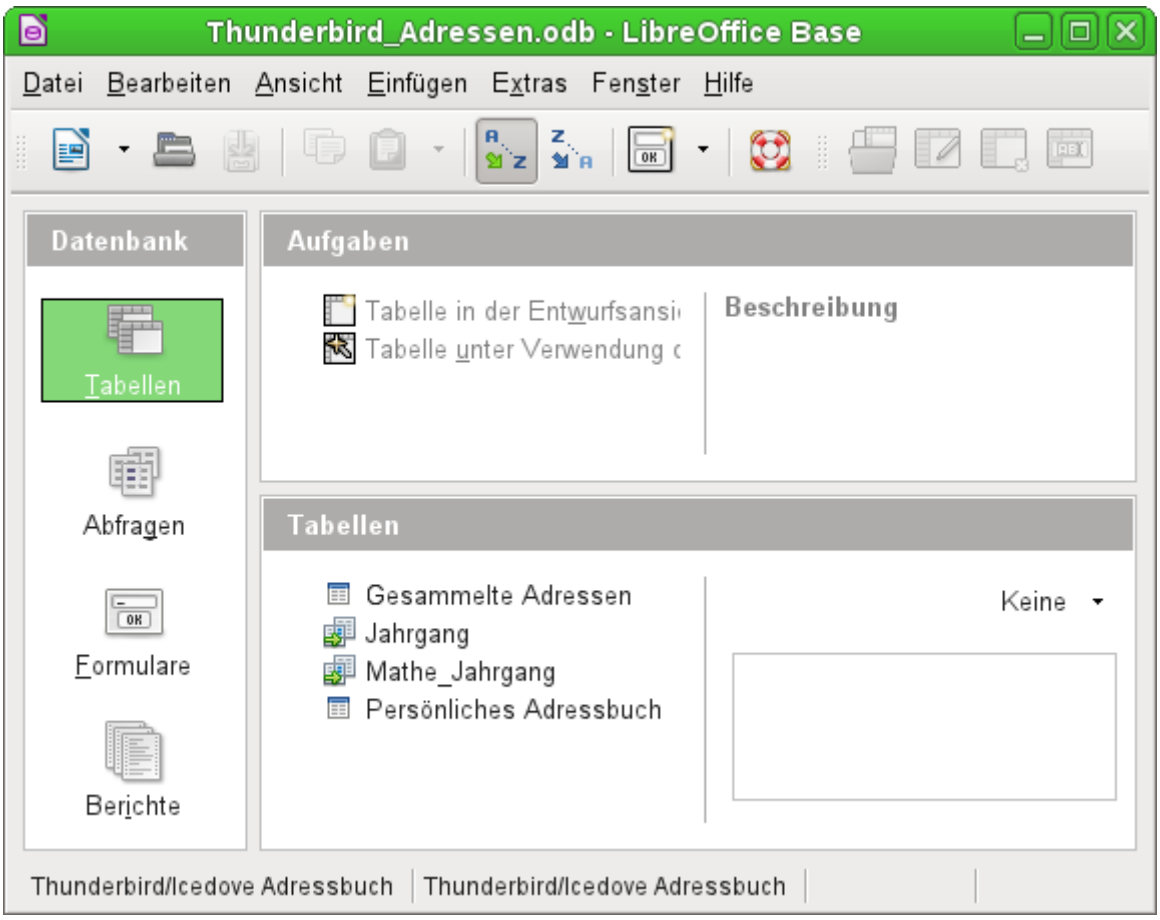
Bei Tabellen, die so in Calc eingelesen werden, werden die besten Ergebnisse erzielt, wenn die Vorlage im Internet die Daten sauber in Zeilen und Spalten darstellt. Zeilenumbrüche in einer Zelle führen zu zusätzlichen Datenzeilen, die dann leere Felder in den anderen Spalten hervorrufen. Zusätzliche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



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

Thunderbird Adressbuch

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



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

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

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

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

Treiberbezeichnung: **org.sqlite.JDBC**

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

Leider ist die Aufmachung der Tabellen in der SQLite-Version des Adressbuches nicht ganz so einfach gestaltet. Etwas problematisch ist auch, dass die Daten jetzt beschreibbar sind. Das kann bei einigen der Codes dazu führen, dass bei falscher Verknüpfung die Daten innerhalb des Adressbuches nicht mehr stimmig sind. Hier ist also Vorsicht geboten. Am besten ist es, die erforderlichen Daten durch Abfragen zusammen zu fassen und eine Datenänderung innerhalb von Thunderbird vor zu nehmen. Hier eine Abfrage, die Namen, Mailadressen und Listenzugehörigkeit anzeigt:

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

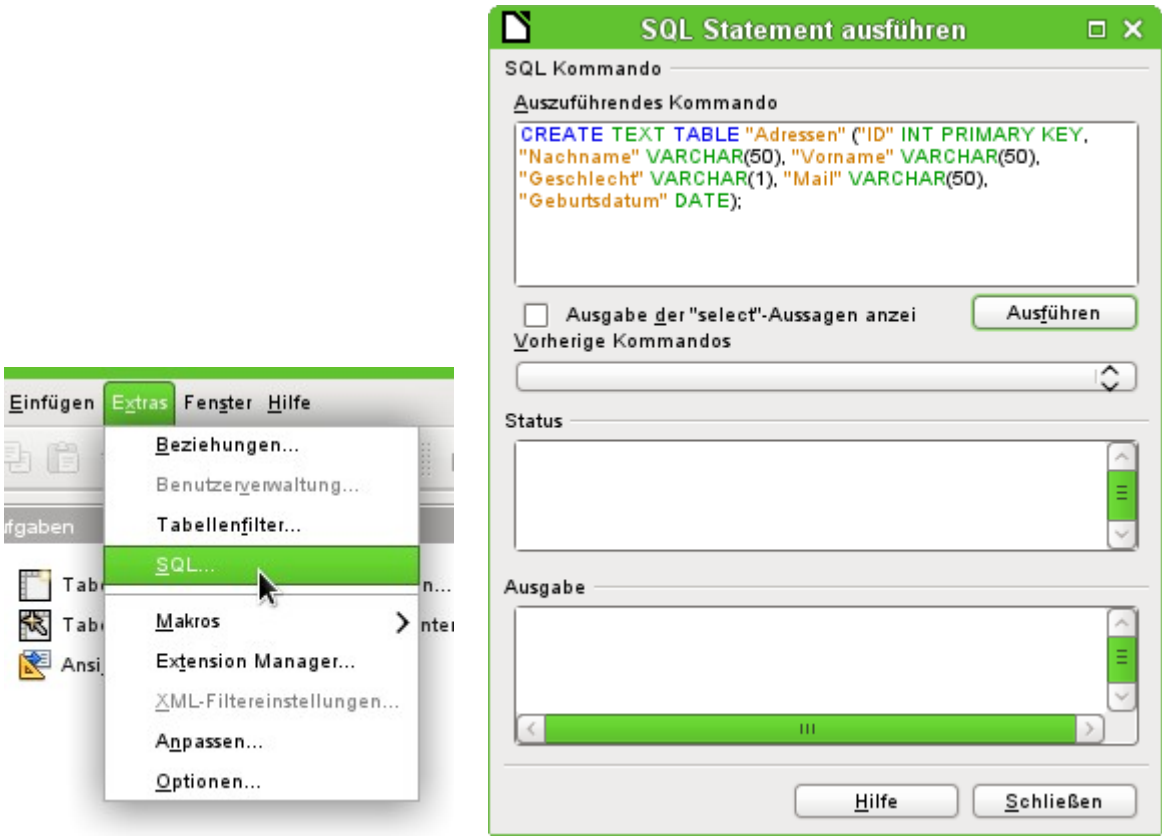
Texttabellen

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

Texttabellen innerhalb einer internen HSQLDB-Datenbank

Ein gängiges Austauschformat zwischen Datenbanken ist das *.csv-Format[2]. Daten werden hier in einer durch einfache Textbearbeitungsprogramme les- und schreibbaren Form gespeichert. Einzelne Felder werden durch Kommata getrennt. Enthält ein Feld Text mit einem Komma, so werden die Textfelder mit doppelten Anführungszeichen versehen. Ein neuer Datensatz beginnt nach einem Zeilenumbruch.

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

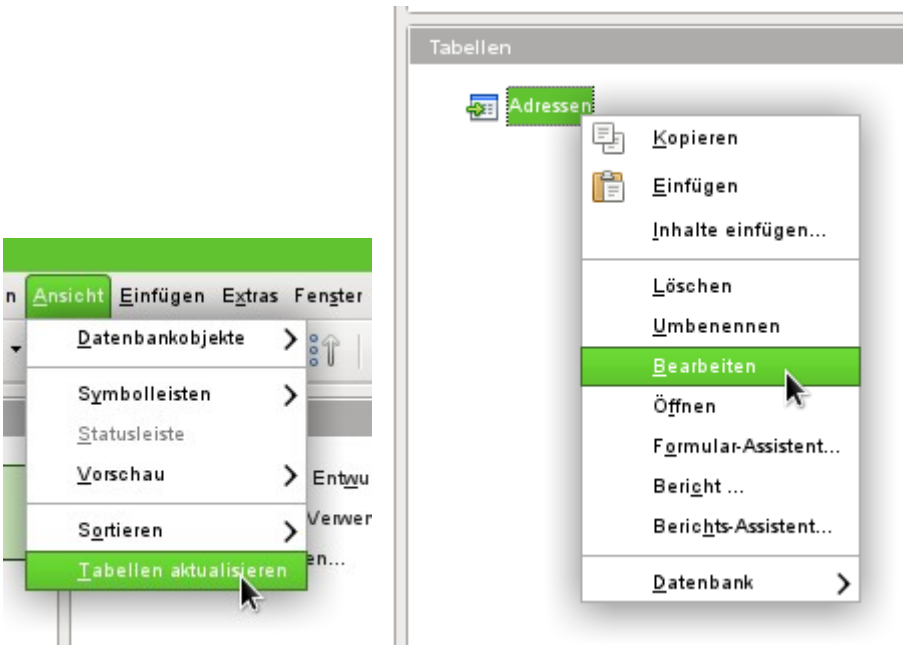


Der Zugriff zu einer Texttabelle kann nicht über die grafische Benutzeroberfläche erstellt werden.[3] Stattdessen wird über **Extras → SQL** eine Texttabelle erstellt. Die Felder in der Texttabelle müssen dem entsprechen, was die Texttabelle in der entsprechenden Reihenfolge liefert. Das Feld "ID" muss z. B. positive Ganzzahlen enthalten, das Feld "Geburtsdatum" z. B. ein Datum in der Schreibweise Jahr – Monat – Tag.

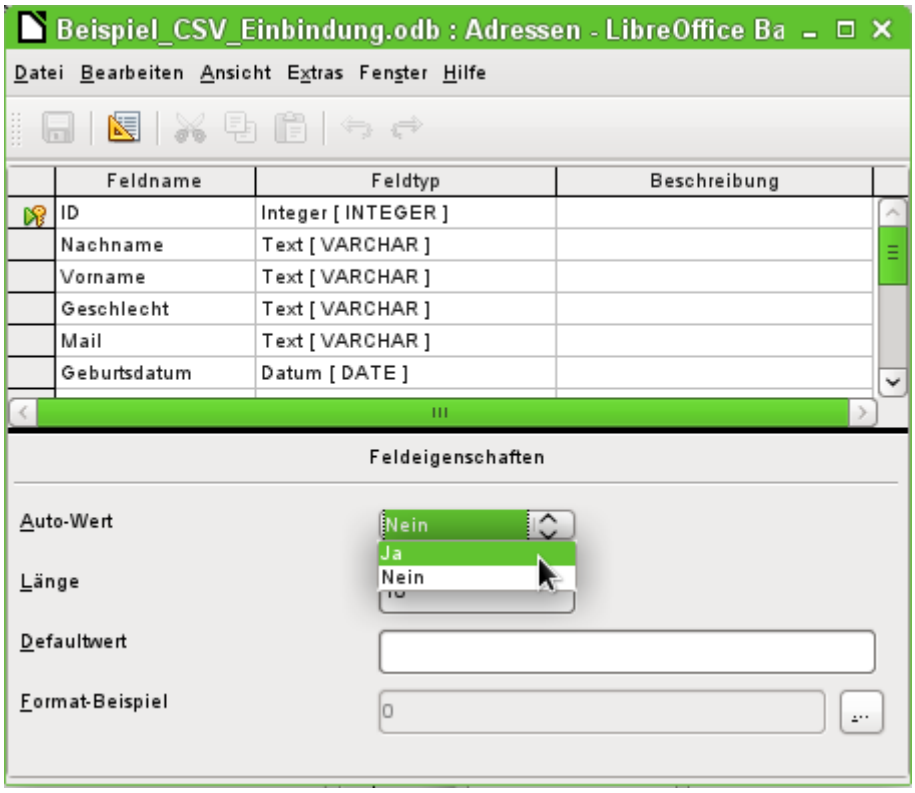
---

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

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



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



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

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

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

```
001 SET TABLE "Adressen" SOURCE "Adressen.csv;encoding=UTF-8";
```
[4]

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

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

Für die Verbindung zur Textdatei stehen verschiedene Parameter zur Verfügung.

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

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

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

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

Mit

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

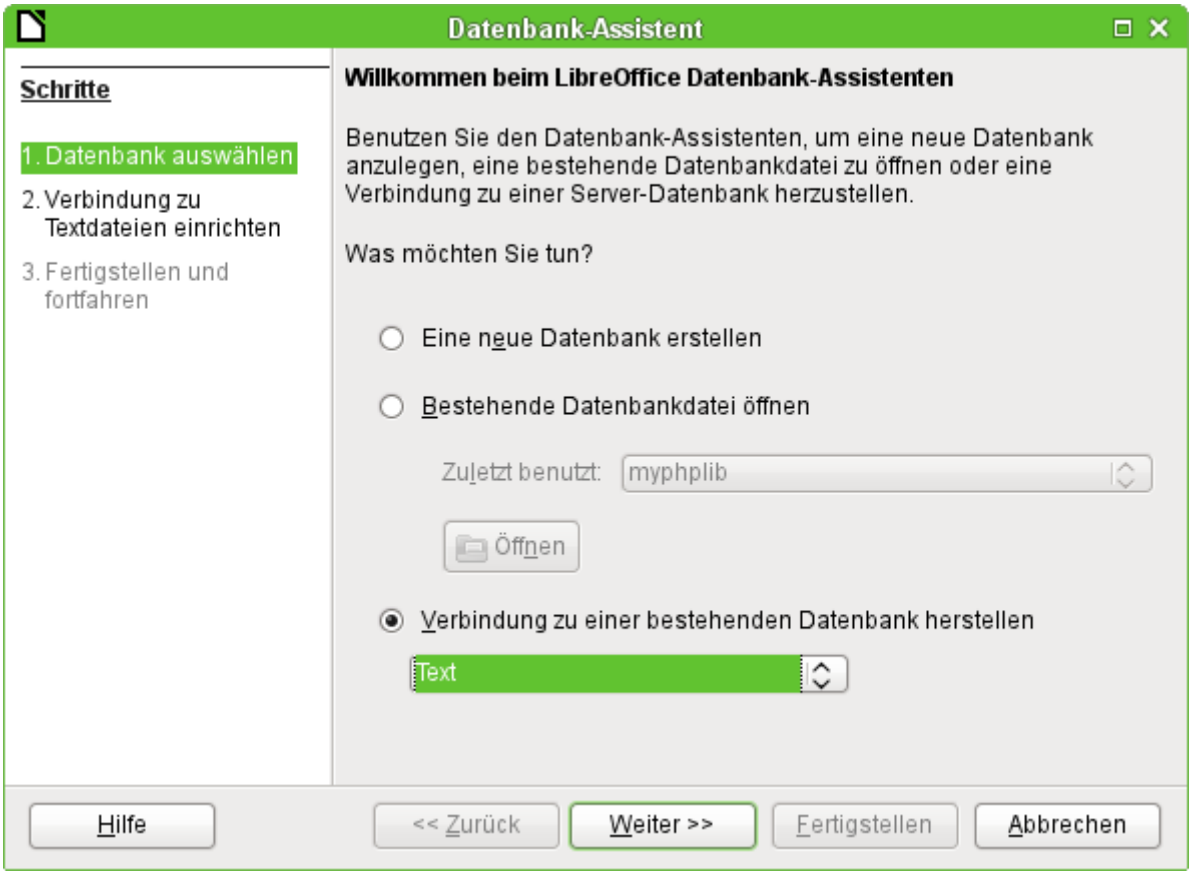
wird schließlich davor geschützt, dass in die Tabelle überhaupt etwas geschrieben wird. So steht dann die Tabelle als normale Adresstabelle wie eine Tabelle aus Adressbüchern z.B. von Mailprogrammen schreibgeschützt zur Verfügung. Die gesonderte Erstellung eines Schreibschutzes ist allerdings nur notwendig,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

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

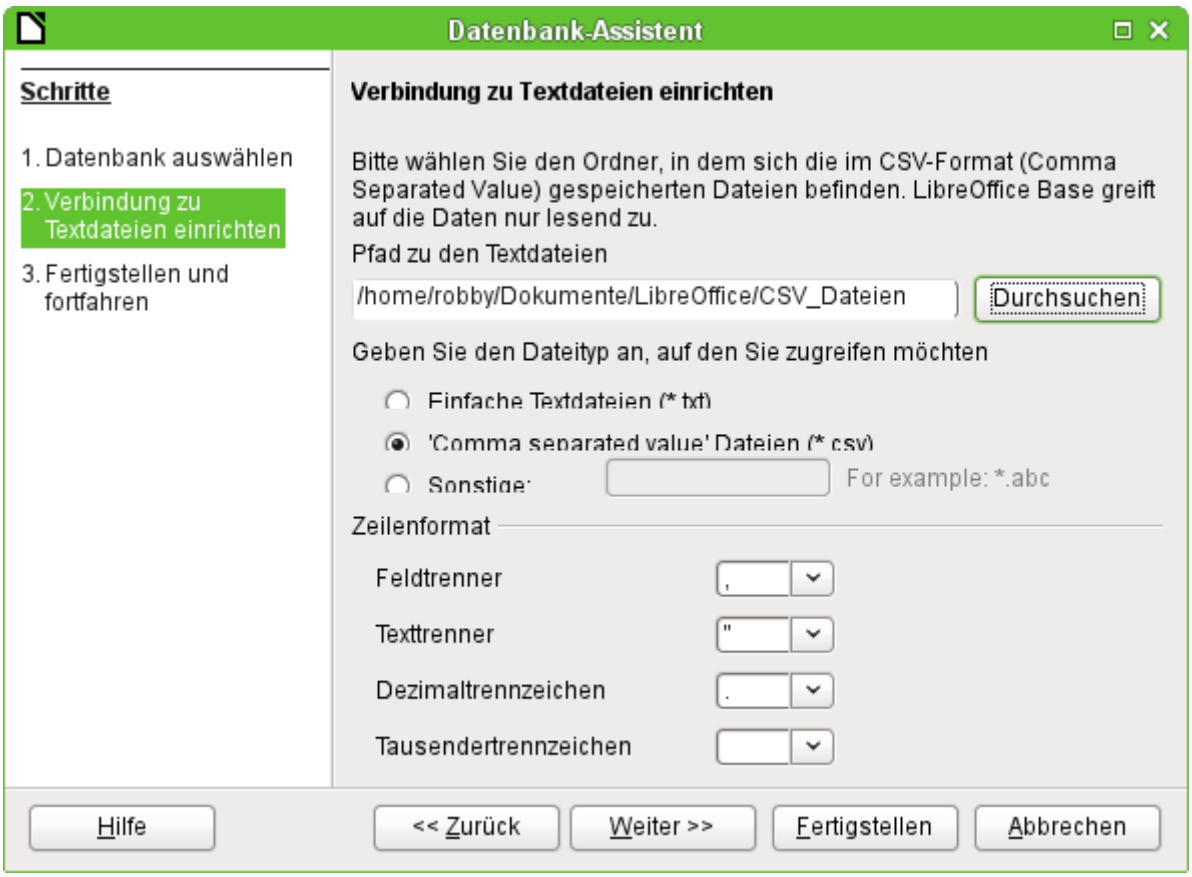
Hinweis	Texttabellen innerhalb der internen HSQLDB scheinen zur Zeit die einzige Möglichkeit zu sein, Zeilenumbrüche aus externen Tabellen wie z.B. einer Calc-Tabelle beim Import in eine HSQLDB zu erhalten: Die Calc-Tabellen müssen jeweils als einzelne *.csv-Dateien exportiert werden. Zeilenköpfe sind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

Texttabellen als Grundlage für eine eigenständige Datenbankdatei

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

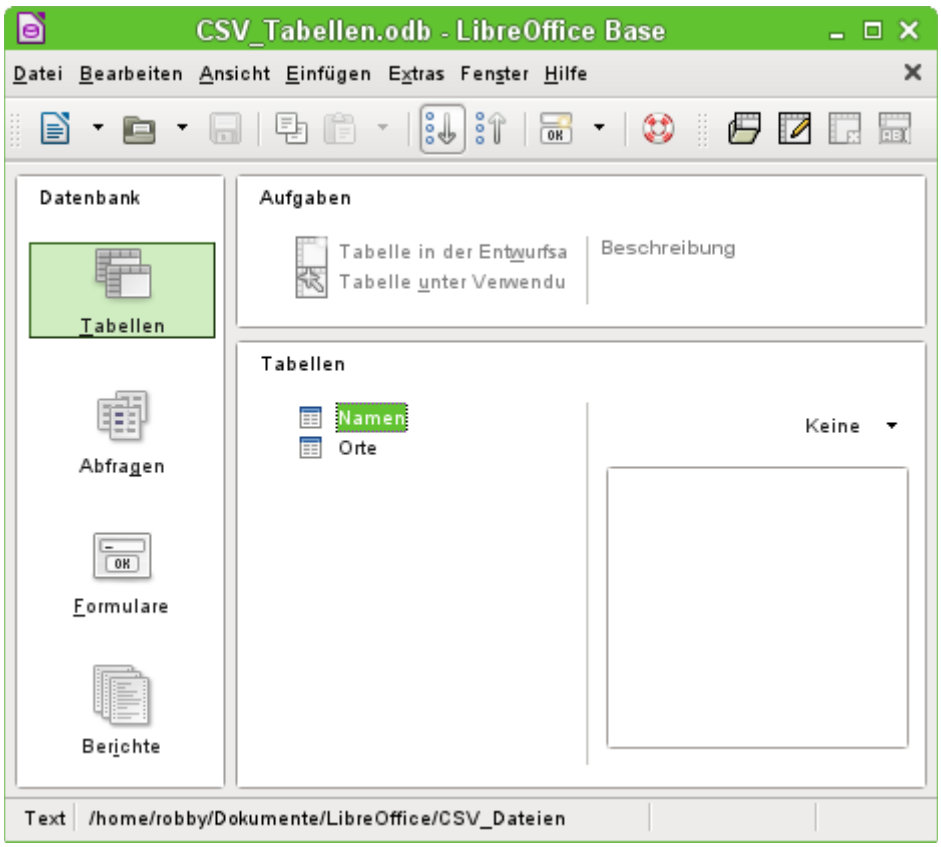


Startpunkt ist die Verbindung zu einer bestehenden Datenbank. Hier ist als Format «Text» gewählt.



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

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



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

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

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

Firebird

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

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

Erstellen eines Nutzers und einer Datenbank

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

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

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

Direkte Verbindung zu einem Firebird Server

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

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

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

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

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

Firebird-Verbindung über JDBC

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

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

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

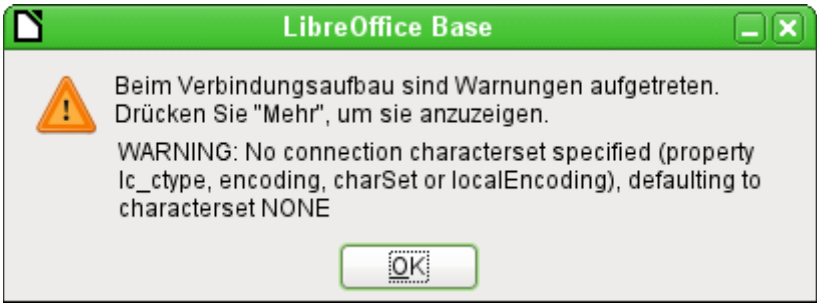
JDBC URL: **jdbc:firebirdsql:oo://host[:port]/<path_or_alias>**

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

In obigen Beispiel wurde dies zu der URL

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

Ohne Zeichensatzangabe erscheint die folgende Fehlermeldung:



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

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

Firebird-Verbindung über ODBC

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

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

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

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

Weitere benötigte Dateien lassen sich mit

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

über die Konsole von Linux ermitteln.

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

odbcinst.ini

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

odbc.ini

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

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

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

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

Direkte Verbindung zu einer Firebird-Datei

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

Aus einer internen Datenbankdatei lässt sich eine externe Datenbankdatei erstellen:

- Das temporäre Verzeichnis, das in LibreOffice definiert ist, muss mit dem Dateimanager aufgesucht werden. Dort erscheint automatisch ein neues Unterverzeichnis, wenn LibreOffice gestartet wird. Dieses Unterverzeichnis ist nicht an dem Namen, sondern an dem Zeitpunkt der Erstellung zu erkennen.
- Die Datenbank mit der internen Firebird-Datenbankdatei muss gestartet werden.[5]
- Mit einem Klick auf **Datenbank → Tabellen** muss der Tabellencontainer geöffnet werden, damit die Verbindung zur internen Datenbankdatei erstellt wird.
- Jetzt werden in dem neu erstellten temporären Unterverzeichnis mehrere Unterverzeichnisse erstellt.
- In einem dieser Unterverzeichnisse steckt neben «firebird.fbk» die daraus erstellte «firebird.fdb». Diese Datei kann jetzt kopiert und als externe Datenbankdatei weiter verwandt werden. Ein Passwortschutz für diese Datei besteht nicht.
- Nach dem Schließen der internen Datenbankdatei wird das temporäre Verzeichnis automatisch gelöscht.

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

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

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

wird zu

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

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

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

Von der externen Firebird-Datei zur Serverdatenbank

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

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

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

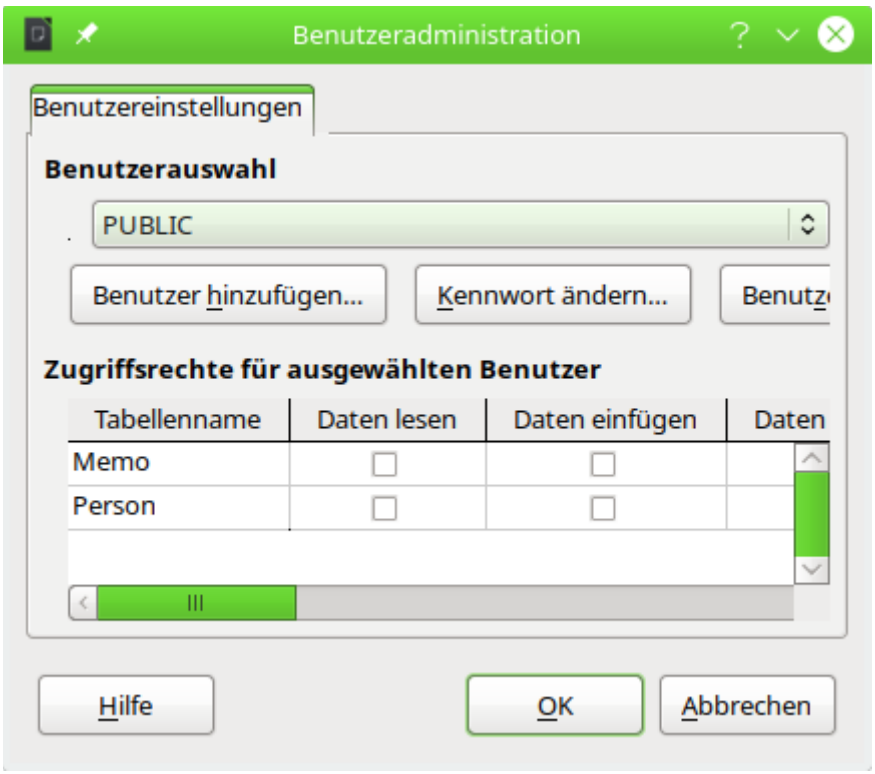
11. Über **Extras → SQL** wird mit
    `CREATE USER neuerUser PASSWORD 'neuesUserPasswort';`
    ein neuer Nutzer erstellt.
12. Verbindet sich der neue Nutzer mit der Serverdatenbank, so kann er zwar die Tabellen sehen, bei einem Klick auf die Tabellen erscheint aber, dass er keine Rechte zum Lesen der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

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

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

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

Benutzerverwaltung bei der externen Firebird Datenbank



Unter Firebird und einigen anderen externen Datenbanken, soweit sie das zulassen, steht über **Extras → Benutzerverwaltung...** ein Dialog zur Verwaltung der Benutzerrechte für die verschiedenen Tabellen zur Verfügung. Der Screenshot zeigt leider schon, dass der Dialog nicht so gut zu der deutschsprachigen Übersetzung passt: Der Button Benutzer löschen ist nicht einmal zur Hälfte sichtbar und lässt sich zur Zeit auch nicht durch Größeneinstellung des Dialogs sichtbar machen.

Serverdatenbanken wie MySQL/MariaDB oder auch PostgreSQL arbeiten nicht mit dieser Benutzerverwaltung zusammen.

SQLite

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

Erstellen einer Datenbank

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

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

```
001 sqlite3 libretest.db
```

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

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

SQLite-Verbindung über ODBC

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

odbcinst.ini

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

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

odbc.ini

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

SQLite-Verbindung über JDBC

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

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

Treiberbezeichnung: **org.sqlite.JDBC**

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

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

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

Zugriff auf Access

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

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

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

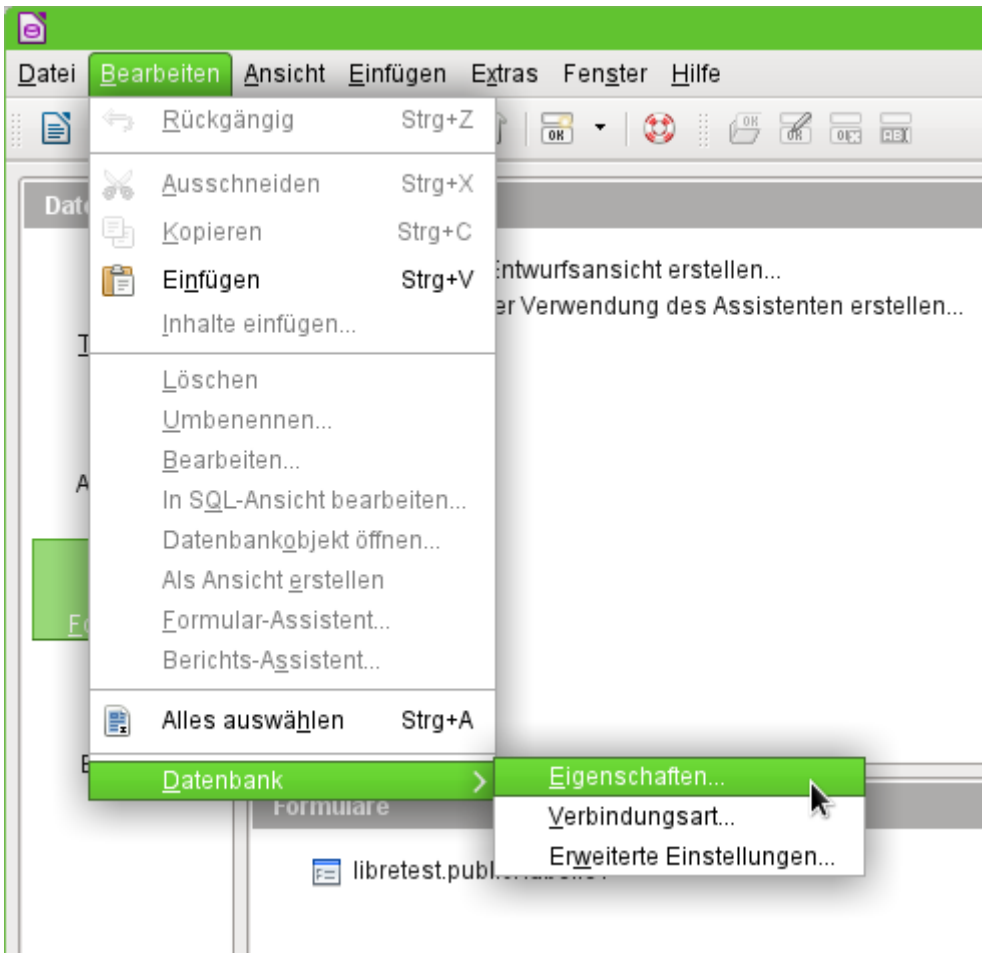
JDBC URL: **`jdbc:ucanaccess:///home/<path_or_alias>`**

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

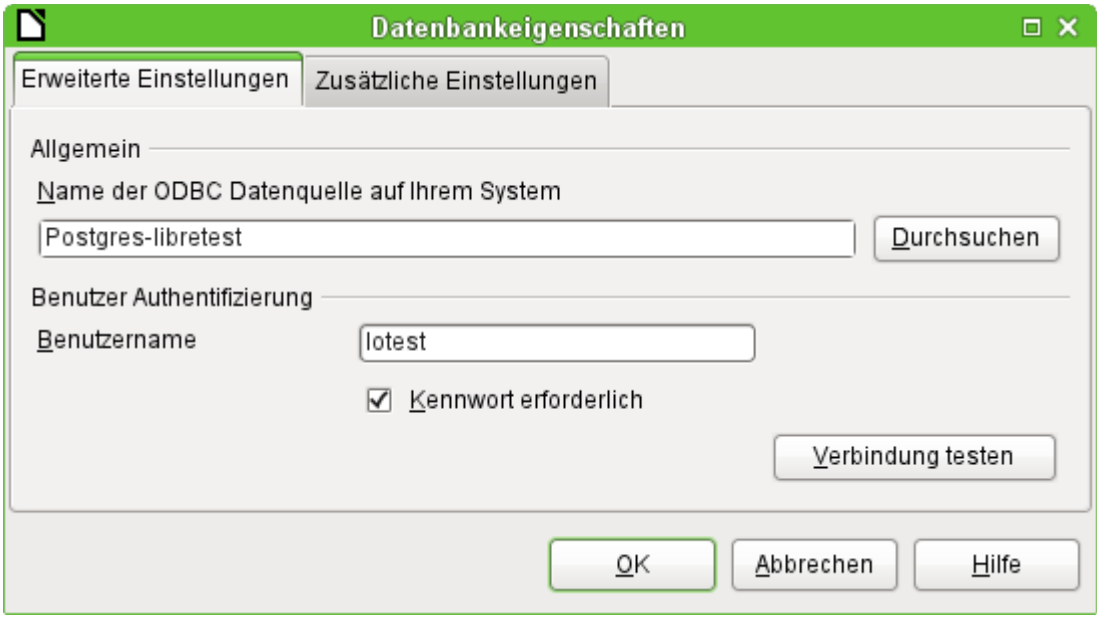
Nachträgliche Bearbeitung der Verbindungseigenschaften

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

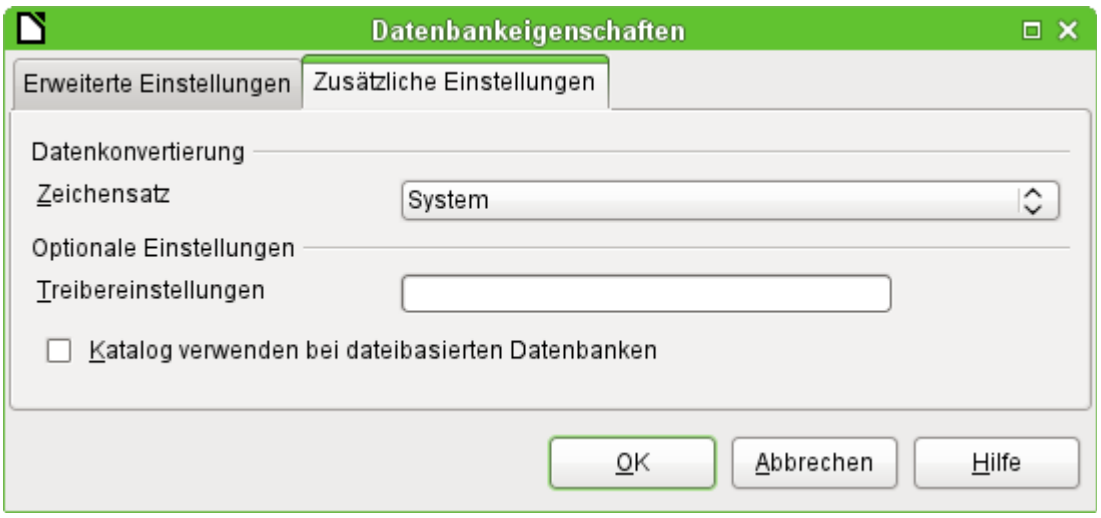
Die folgenden Screenshots zeigen die Möglichkeit auf, wie z.B. die Verbindungsparameter zu einer externen PostgreSQL-Datenbank im Nachhinein geändert werden können.



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

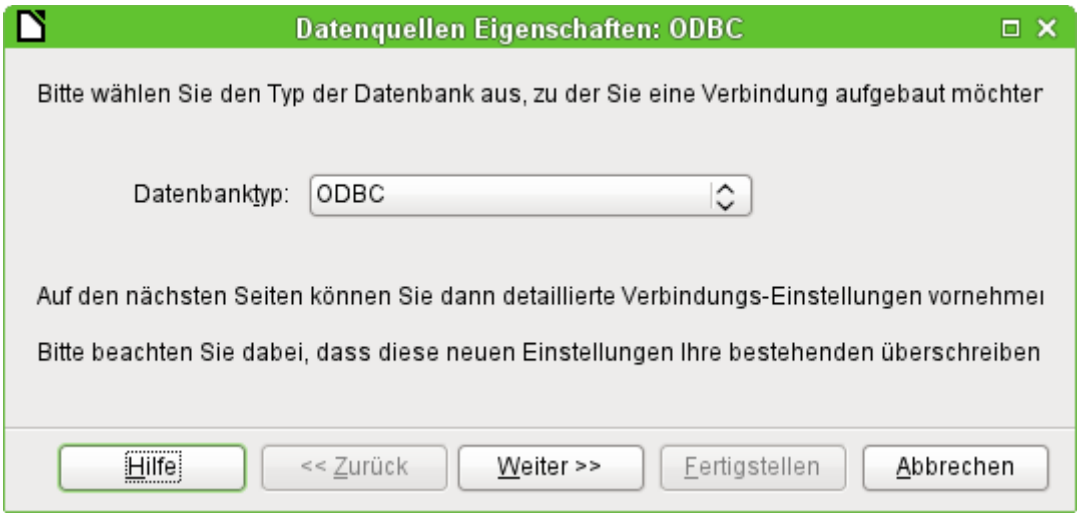


Falls sich der Name der Datenquelle geändert hat, kann dies hier geändert werde. Bei einer Verbindung über ODBC wird ja der Name, mit dem die Datenbank aufgerufen werden kann, in der odbc.ini festgelegt. Der Name ist in der Regel nicht gleich dem eigentlichen Datenbanknamen in PostgreSQL.

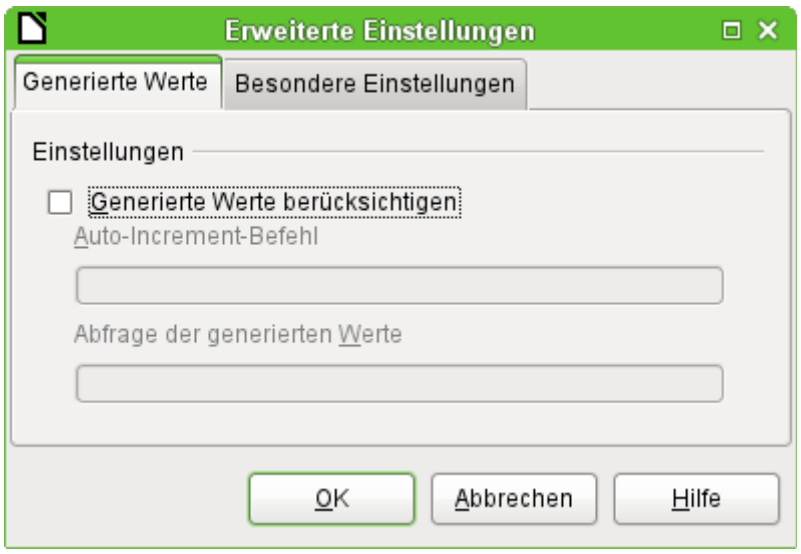


Gibt es Probleme mit dem Zeichensatz? Diese Probleme können eventuell in dem zweiten Reiter der Datenbankeigenschaften gelöst werden.

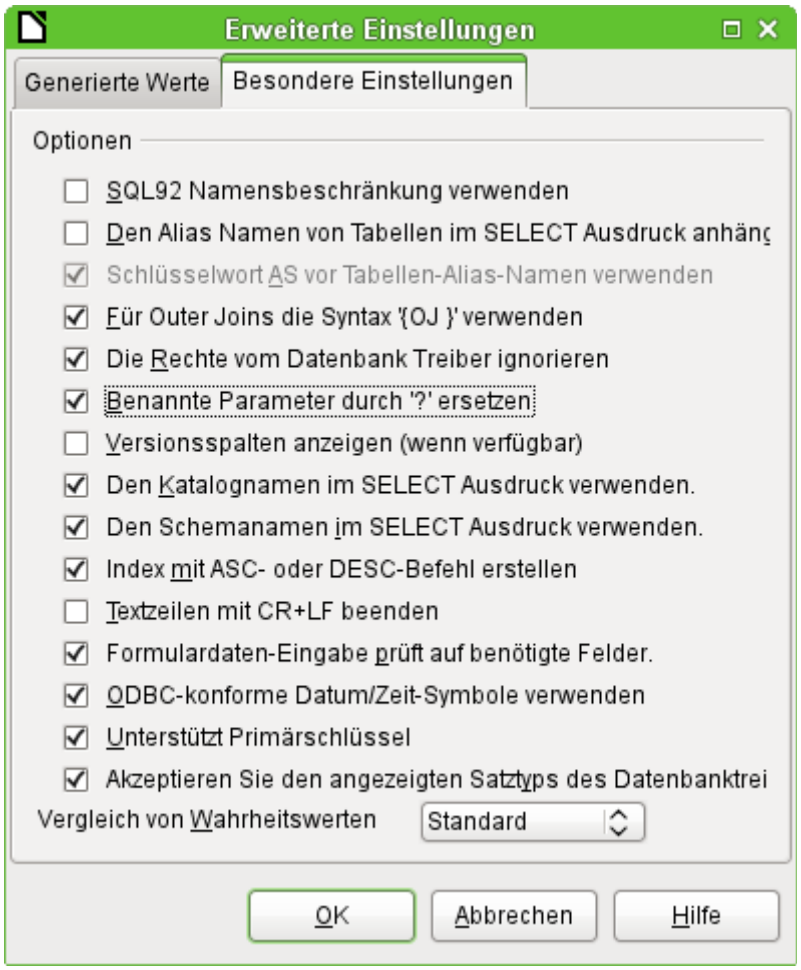
Auch eine zusätzliche spezielle Einstellung des Treibers ist möglich, wenn eventuell ein Parameter ausgeführt werden soll, der zur Zeit nicht bereits in der odbc.ini steht.



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



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



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

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

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

Der nicht funktionierende Code:

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

Wird dieser Code geändert auf

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

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

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

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

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

Maskierung von Tabellennamen und Feldnamen

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

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

funktioniert genauso wie

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

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

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

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

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

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

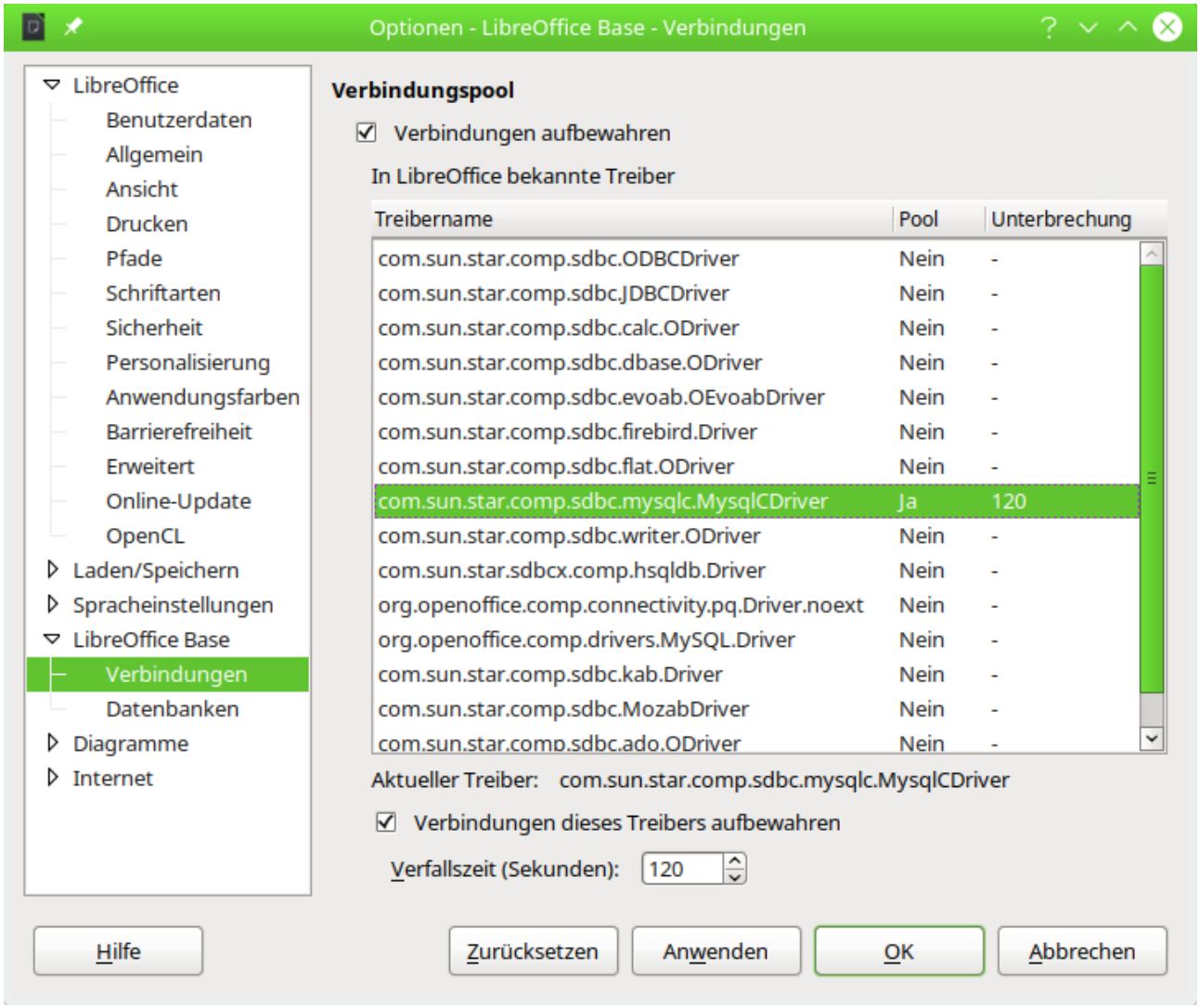
Bei externen Datenbanken hängt die Maskierung von dem entsprechenden Datenbanksystem ab. Für **MySQL/MariaDB** schreibt Base je nach Treiber direkt die passende Maskierung in den SQL-Code, der durch den Abfrageeditor erstellt wird:

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

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

Treiberverbindungen aufbewahren

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



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

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

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

nach

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

gewechselt werden.

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

Datenbankverbindung über SSH

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

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

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

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

localhost:63333:libretest:dblotest:MeinPasswort
Die Datei wird unter Linux als **.pgpass** direkt im Homeverzeichnis abgespeichert. Sie sollte für andere Nutzer nicht lesbar sein.
Ist diese Datei für die entsprechende Datenbank vorhanden, so entfällt in Base die Angabe von Benutzername und Passwort.

Tipp

Die Angabe eines Passwortes kann nach einem Schlüsselaustausch entfallen. Auf der Konsole des Clients (nicht ssh) wird dazu angegeben:
ssh-keygen -t rsa
Alle Abfragen zu Dateiname und Passphrase, die nach diesem Befehl folgen, werden mit einem Return bestätigt. Wird der Dateiname anders gewählt, so klappt eventuelle das folgende Kommando nicht.
ssh-copy-id 192.168.178.32
kopiert den Schlüssel zum Server. Zum letzten Mal wird hier das Passwort abgefragt. Danach erfolgt das Einloggen per ssh ohne Passworteingabe.

Damit erscheint die Passwortabfrage nicht mehr und die Verbindung über ssh kann mit Hilfe von Startdateien erfolgen.

Unter **Linux** enthält die Datei **db_ssh.sh** den folgenden Inhalt:

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

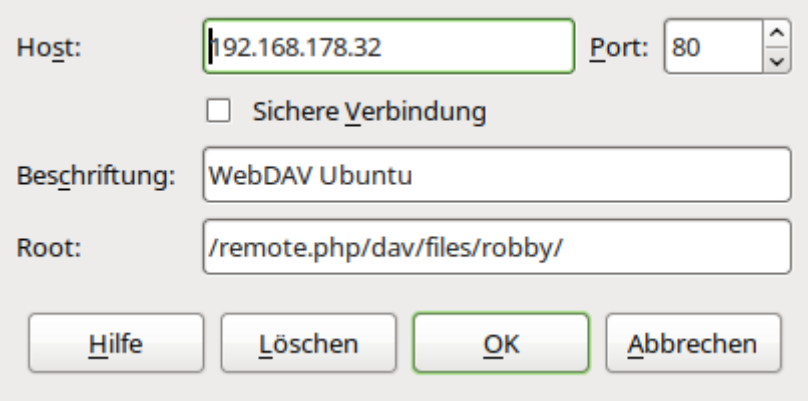
Die Datei muss anschließend ausführbar gemacht werden.

Unter **Windows** enthält die **Date db_ssh.bat** den folgenden Inhalt:

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

Datenbanken in der Cloud

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



Verbindung im lokalen Netz zu einer Nextcloud-Installation

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

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

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

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

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

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

Bugs und Workarounds bei verschiedenen Datenbankverbindungen

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

MySQL/MariaDB

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

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

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

PostgreSQL

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

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

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

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

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

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

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

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

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

```
001 GLOBAL boNew AS BOOLEAN
```

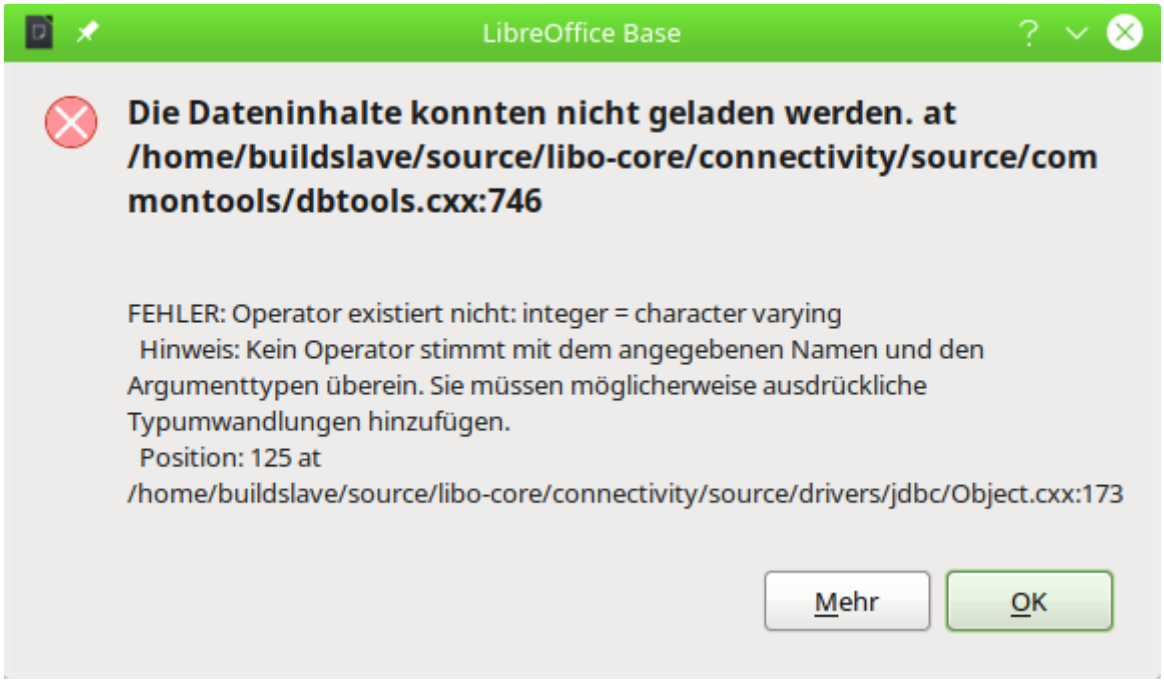
```
001 SUB OldNew(oEvent AS OBJECT)
002    DIM oForm AS OBJECT
003    oForm = oEvent.Source
004    IF hasUnoInterfaces(oForm, "com.sun.star.form.XForm" ) THEN
005       IF oForm.getInt(1) = 0 THEN
006          boNew = True
007       ELSE
008          boNew = False
009       END IF
010    END IF
011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

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

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

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

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



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

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

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

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

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

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

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

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

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

dBase

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

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