Dietmar Horn

# Programmieren kinderleicht
## mit
# XProfan

**für Windows 9.x, Windows 2000, XP, Vista, Windows 7/8**

## Teil 3: „SQL-Datenbanken mit Firebird“

## „Das verstehe sogar ich ...“

# Inhaltsverzeichnis

# 1. Vorbemerkungen

## 1.1. Was ist SQL?

SQL ist eine Datenbanksprache zur Definition von Datenstrukturen in relationalen Datenbanken sowie zum Bearbeiten (Einfügen, Verändern, Löschen) und Abfragen von darauf basierenden Datenbeständen.

Die Syntax von SQL ist relativ einfach aufgebaut und an die englische Umgangssprache angelehnt. Ein gemeinsames Gremium von ISO und IEC standardisiert die Sprache unter Mitwirkung nationaler Normungsgremien wie ANSI oder DIN. Fast alle gängigen Datenbanksysteme unterstützen SQL – allerdings in unterschiedlichem Umfang und leicht voneinander abweichenden „Dialekten“. Durch den Einsatz von SQL strebt man die Unabhängigkeit der Anwendungen vom eingesetzten Datenbankmanagementsystem an.

Die Bezeichnung SQL wird im allgemeinen Sprachgebrauch als Abkürzung für „Structured Query Language“ aufgefasst, obwohl sie laut Standard ein eigenständiger Name ist. Die Bezeichnung leitet sich von dem Vorgänger SEQUEL (Structured English Query Language) ab, welche von Edgar F. Codd (IBM) in den 1970er Jahren entworfen wurde. SEQUEL wurde später in SQL umbenannt, weil SEQUEL ein eingetragenes Warenzeichen der Hawker Siddeley Aircraft Company ist.

## 1.2. Was ist Firebird?

Firebird ist ein freies Server-Programm und Teil eines Systems zum Betreiben von Datenbanken (DBMS). Es ist der freie Ableger des weiterhin kommerziell von Embarcadero (ehemals CodeGear, Borland) vertriebenen relationalen Datenbankmanagementsystems InterBase. Die Abspaltung erfolgte im Jahre 2000, als kurz vor Freigabe der Version 6 des kommerziellen Vorgängers Interbase bei Borland ernsthafte Überlegungen im Gange waren, die Weiterentwicklung einzustellen.

Aus Interbase 6.0 wurde Firebird 1.0, wobei dies als eine Bugfix-Version mit nur wenigen Erweiterungen angesehen werden kann. Eine Erweiterung von Firebird 1.0 ist der 64-Bit File I/O, sodass auch Datenbankdateien von mehr als 2 GB Größe erzeugt werden können.

Grundlegende Erweiterungen erfolgen im Firebird-2.0-Zweig. Der gesamte Quellcode von Interbase wurde aus der Sprache C nach C++ übersetzt. Ab Februar 2004 stand die erste Produktiv-Version aus dem Firebird 2.0-Zweig als Firebird 1.5 zur Verfügung. Im November 2006 wurde Firebird 2.0 freigegeben, im Oktober 2011 die derzeit aktuelle Version 2.5.1.

Die Firebird-Datenbank gibt es in drei Versionen mit unterschiedlichem Funktionsumfang. Es gibt die SuperServer-, ClassicServer- und EmbeddedServer-Variante. Nicht alle Varianten sind für alle Plattformen erhältlich.

★ **SuperServer**
Die SuperServer-Variante ist ein Multithreaded Server-Prozess. Der SuperServer verwaltet alle Benutzeranfragen und Verbindungen mittels voneinander unabhängigen Threads innerhalb eines Prozesses. Unter Windows (ab NT4) kann der SuperServer als Dienst laufen oder generell auch als Applikation.

★ **ClassicServer**
Mit dem ClassicServer werden alle Verbindungen in getrennten Prozessen verwaltet. Jeder Prozess verwaltet dabei seinen eigenen Datenbank-Cache. Durch die Trennung nach Prozessen eignet sich der ClassicServer gut für Multiprozessor-Umgebungen. Er verbraucht allerdings mehr Arbeitsspeicher.

★ **Embedded**
Mit der Embedded-Variante ist es möglich, einer einzelnen Applikation exklusiven Zugriff auf eine Datenbank zu ermöglichen. Diese Servervariante eignet sich sehr gut für Einzelanwendungen, die mit einer eigenen Datenbank laufen und keinen Mehrbenutzermodus benötigen. Die Embedded-Version benötigt keine Installation und hat ausschließlich nur eine Programmbibliothek für die verfügbaren Plattformen.

★ **32- und 64-bit-Unterstützung**

Seit der Version 2.1.3 vom September 2009 sind für Windows und Linux getrennte 32- und 64-bit-Versionen verfügbar.

## 1.3. Was ist ein BLOB?

**B**inary **L**arge **Ob**jects (**BLOB**s) sind große binäre Objekte wie z.B. Bild- oder Audiodateien.

Für eine Datenbank sind BLOBs nicht weiter strukturierte Objekte beziehungsweise Felddaten. Einige Datenbanken gestatten, dass die Feldtypen große Datenmengen (quasi komplette Dateien) als Feldinhalt abspeichern können.

## 1.4. XProfan und SQL

**Auszug aus der XProfan-X2-Hilfedatei:**

XProfan unterstützt auch die einfach zu handhabende ODBC-Schnittstelle, um auf beliebige andere Datenbanken zugreifen zu können, die über ODBC verfügen. So empfiehlt sich XProfan auch als Frontend bzw. Client in einer Client-Server-Umgebung.

ODBC ist die windowseigene Schnittstelle zu SQL-fähigen Datenbanken. Dahinter steht der Gedanke, dass es der Anwendung völlig egal sein kann, auf welche Datenbank zugegriffen wird. Die Schnittstelle zu allen SQL-fähigen Datenbanken ist identisch. Will ich eine andere Datenbank benutzen, muss ich nur den Treiber auswechseln.

SQL ist ein Standard, um mit den gleichen Befehlen auf beliebige Datenbanken zugreifen zu können, um deren Daten auszuwerten. Im Idealfall bedeutet das, dass man mit denselben Befehlen auf die unterschiedlichsten Datenbanken zugreifen kann, unabhängig von Datenbank oder Betriebssystem. Einmal erstellte komplexe Abfragen können weiter verwendet werden, solange die Tabellenstruktur der Datenbank gleich bleibt. Ein Wechsel des Datenbankherstellers oder des Betriebssystems führt dann nicht zwangsweise zu einer kompletten Neuentwicklung der Datenbank und der Abfragen.

## 1.5. XProfan und Firebird

**Auszug aus der XProfan-X2-Hilfedatei:**

**Firebird** ist eine mächtige zu **Interbase** kompatible Datenbank. Es gibt sie sowohl in einer - Client/Server Version und einer "embedded" Version.

Der Vorteil der Embedded-Version: Sie muss nicht komplex installiert werden und es werden keine Treiber benötigt. Einfach das Paket in ein beliebiges Verzeichnis packen und es funktioniert. Die Anwendung kommt ins gleiche Verzeichnis und ruft direkt die Funktionen der DLL auf.

Zunächst holen wir uns eine aktuelle Version. Die Version, mit der die XProfan-Implementation getestet wurde, gibt es hier als ZIP-Datei: http://www.xprofan.de/download//FirebirdEmbed.zip

Diese entpacken wir in ein beliebiges Verzeichnis. Das war es dann auch schon. Wollen wir eine bestehende Anwendung für den Firebird-Client damit ausführen, müssen wir lediglich die **fbembed.dll** in **fbclient.dll** umbenennen. Der Funktionsumfang ist derselbe.

Zum Entwickeln einer Firebird Embedded Anwendung also ein Verzeichnis anlegen und darin Firebird embedded komplett hineinkopieren, so wie es von Firebird zur Verfügung gestellt wird. In exakt diesem Verzeichnis können wir die Anwendung entwickeln.

Ach ja: Und der Name der Datenbankdatei darf einen Pfad enthalten. Das heißt: Die Datenbank muss nicht im Verzeichnis der Anwendung liegen. Allerdings funktioniert Firebird Embedded nicht auf Netzlaufwerken und auch die Datenbank darf nicht auf einem solchen liegen.

Hinweise:

- ★ Um schnell loslegen zu können, wird bei der XProfan-Installation ein Firebird-Embedded-Verzeichnis mit installiert. Um dieses "sauber" zu halten, empfiehlt es sich vor der Entwicklung eines Firebird-Programms für dieses ein neues Verzeichnis anzulegen und den Inhalt des Firebird-Verzeichnisses dort hinein zu kopieren.

- ★ Es ist natürlich möglich, dass es inzwischen eine aktuellere Firebird-Version gibt. Diese findet sich auf der Firebird-Produkt-Seite: http://www.firebirdsql.org/index.php

## 1.6. Zum Anliegen dieses Lehrbuches

### 1.6.1. Allgemein

Anliegen dieses Tutorials ist es keinesfalls, Programmier-Anfängern, -Einsteigern oder -Umsteigern Hinweise und Anregungen zum Programmieren mit XProfan zu geben.
Hierfür gibt es bereits seit Jahren mein „XProfan-Lehrbuch“ und z.B. „Programmieren kinderleicht mit XProfan [Einführung]“.

Vielmehr geht es darum, Grundlagen zum Erstellen einer SQL-Datenbankanwendung mit XProfan und dem DBMS Firebird zu vermitteln.

### 1.6.2. Voraussetzungen zum Arbeiten mit XProfan und Firebird

- ★ Zunächst benötigen Sie das kostenlose **Firebird ab Version 2.5.** Hinweise zum Download und zur Installation finden Sie in den Kapiteln 1.5. und 2.1. dieses Tutorials.
- ★ Die Firebird-Befehle sind in XProfan ab **XProfan X2** enthalten.
- ★ Für Anwender von XProfan 11 stellt der XProfan-Autor, Roland G. Hülsmann, eine kostenlose XProfan-Unit **Firebird.pcu** mit weitestgehend identischem Funktionsumfang zum Download zur Verfügung: http://www.xprofan.de/download/FirebirdXPrf.zip
  Im offiziellen XProfan-Support-Forum finden Sie Hinweise zur Firebird.pcu:
  http://www.rgh-soft.de/forum01/read.php?f=7&i=14281&t=14281

### 1.6.3. Und wie geht es weiter?

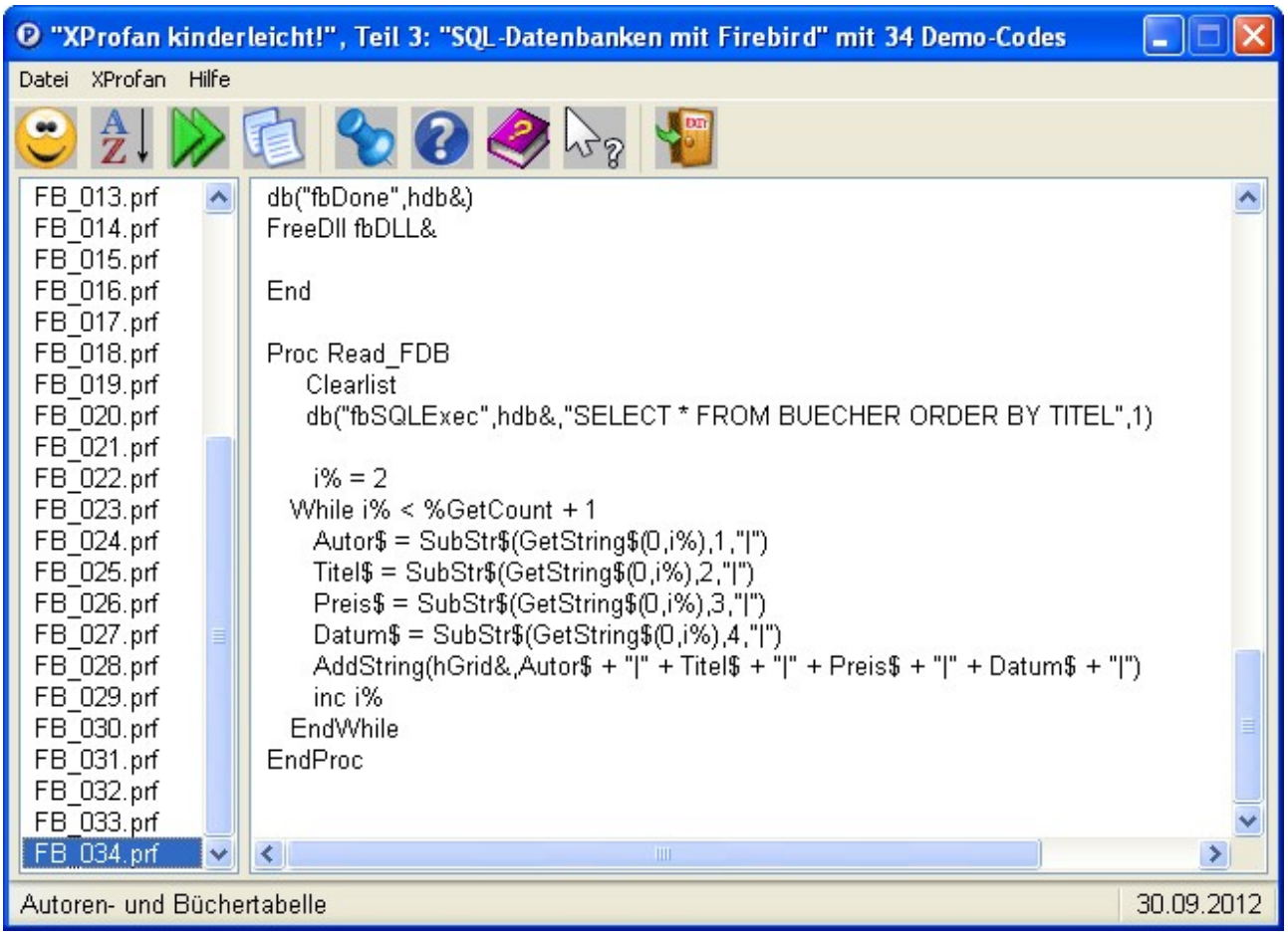
Dies und vieles mehr können Sie auf den folgenden Seiten dieses Tutorials nachlesen.

Ausgehend vom Anlegen einer Datenbank und dem Erstellen von Tabellen in dieser Datenbank wird demonstriert, wie Datensätze in die Tabelle eingefügt, verändert, ausgelesen, gefiltert und ausgelesen werden.

Im Kapitel 6. wird beispielsweise Schritt für Schritt (aufbauend auf vorherige Kapitel des Tutorials) eine kleine Bücherliste mit XProfan und Firebird entwickelt, die Sie Ihren Bedürfnissen gerne anpassen können.

In der Vollversion dieses Lehrbuches wird ein installiertes **Firebird 2.5. embedded** mitgeliefert. Wenn Sie in diesen Ordner Interpreter, Runtime, Compiler und die Hilfedatei von XProfan X2 kopieren, sowie den von Ihnen benutzten XProfan-Editor, dann können Sie sofort loslegen.

Zusätzlich ist ein Lehrbuch-Helfer enthalten, der alle Beispielcodes aus dem Lehrbuchtext auflistet, die Sie über die Windows-Zwischenablage in Ihren XProfan-Editor zum Testen und zur weiteren Bearbeitung kopieren können.




Eine komplette Adressenverwaltung (mit Quellcode) und eine Firebird-Adressen-Datenbank runden dieses Tutorial ab.

### 1.6.4. Welche Firebird-Dateien müssen einer Anwendung beigelegt werden?

Zusätzlich zum selbst erstellten Programm (Exe-Datei und ggf. der Datenbank) werden folgende Dateien aus dem Firebird-Installationsverzeichnis benötigt, damit Ihre Anwendung auch auf Computern läuft, auf denen kein Firebird embedded installiert ist:

⇒ fbclient.dll bzw. fbembed.dll
⇒ firebird.msg
⇒ ib_util.dll
⇒ icudt30.dll
⇒ icuuc30.dll

# 2. Erste Schritte mit Firebird embedded

## 2.1. Installation

- ★ Downloaden Sie das Archiv **FirebirdEmbed.zip** von der Homepage des XProfan-Herstellers: http://www.xprofan.de/download//FirebirdEmbed.zip
- ★ Entpacken Sie alle Dateien des Archivs in einen Ordner der Festplatte. Dieser Ordner wird der Entwicklungsordner für unser Firebird-Projekt.
- ★ Benennen Sie die Datei **fbembed.dll** in **fbclient.dll** um.
- ★ Kopieren Sie die wichtigsten Dateien von XProfan X2 in diesen Ordner:
  - **Profan.exe** (Interpreter)
  - **PrfRun32.exe** (Runtime-Modul)
  - **ProfComp.exe** (Compiler)
  - **Profan.chm** (XProfan-Hilfedatei)

  und zusätzlich die Dateien des von Ihnen verwendeten XProfan-Editors.

## 2.2. Das Erzeugen einer Datenbank

Vor jeder Verwendung von Firebird ist der Name der Firebird-DLL anzugeben. Bei Firebird embedded ist dies in der Regel die "fbembed.dll". Wenn man den Client der Client/Server Version von Firebird installiert hat und diese nutzen möchte, ist hier die Firebird-Client-DLL anzugeben, die "**fbclient.dll**" heißt.

Für die Beispiele in diesem Tutorial verwenden wir die **fbclient.dll**.

### db("fbUseDLL",S)

| | | |
|---|---|---|
| S: | String | - Firebird-DLL |
| Ergebnis: | Integer | - Ergebnis: Handle der DLL |

Die zu benutzende Firebird-DLL wird ausgewählt und initialisiert.

Im Normalfall ist das bei Firebird Embedded die "fbembed.dll". Eine Pfadangabe erfolgt nicht, da die Anwendung im gleichen Verzeichnis wie die DLLs von FireBird Embedded liegen soll.

Sollen auch Anwendungen für die Client-Server-Version laufen, ist die DLL in "fbclient.dll" umzubenennen.

Eine neue Datenbank erzeugen wir mit **db("fbCreate"**. Damit wird lediglich eine leere Firebird-Datenbank erzeugt, die noch keine Tabelle und demzufolge auch noch keine Anwenderdaten enthält.

## db("fbCreate",U,P,D)

| | | |
|---|---|---|
| U: | String | - Username |
| P: | String | - Passwort |
| D: | String | - Name der Datenbankdatei |
| | | |
| Ergebnis: | Integer | - Handle der Datenbank |

Eine neue Firebird-Datenbank wird erstellt, und die notwendigen Systemtabellen werden angelegt. In Firebird/Interbase befinden sich alle Tabellen (auch Systemtabellen) in einer physikalischen Datei. Die Datenbank wird auch gleich geöffnet. Username und Passwort werden bei späteren Zugriffen über **db("fbInit"** benötigt.

Ist eine Datei mit diesem Namen bereits im Verzeichnis vorhanden, erfolgt eine Fehlermeldung.

FB_001.PRF:

```
Var fbDLL& = db("fbUseDLL", "fbclient.dll")

declare hdb& 'Handle der Datenbank

ifnot fileexists("XAdressen.fdb")
    hdb& = db("fbCreate", "SYSDBA", "masterkey", "XAdressen.fdb")
endif
```

Mit diesem Code wird eine leere Datenbank namens **XAdressen.fdb** erzeugt. Die Extension **.fdb** steht für **F**irebird-**D**aten**b**ank.

Falls Sie versuchen sollten, eine bereits existierende Datenbank mit **db("fbCreate"** neu zu erzeugen, werden Sie mit einer Messagebox auf Ihren Fehler hingewiesen, und Sie können programmiertechnisch darauf bereits im Vorfeld entsprechend reagieren.

## 2.3. Das Öffnen einer vorhandenen Datenbank

Eine vorhandene Datenbank wird mit **db("fbInit"** geöffnet und muss spätestens zum Programmende mit **db("fbDone"** geschlossen werden.

**db("fbInit",U,P,D)**

| | | |
|---|---|---|
| U: | String | - Username |
| P: | String | - Passwort |
| D: | String | - Name der Datenbankdatei |
| Ergebnis: | Integer | - Handle der Firebird-Datenbank |

Eine bestehende Firebird- oder Interbase-Datenbank wird geöffnet. In Firebird/Interbase befinden sich alle Tabellen (auch Systemtabellen) in einer physikalischen Datei. Ist die Datenbank nicht vorhanden, erfolgt eine Fehlermeldung. Sind Username oder Passwort falsch, erfolgt beim Zugriff eine Fehlermeldung. Ist die Datenbank bereits geöffnet, erfolgt auch eine Fehlermeldung, da unter Firebird Embedded nur ein User gleichzeitig auf eine Datenbank zugreifen kann. Für die Client-Server-Version gilt die Einschränkung nicht.

Es können in einer Anwendung aber durchaus mehrere Datenbanken geöffnet werden.

**db("fbDone",H)**

| | | |
|---|---|---|
| H: | Integer | - Handle einer Firebird-Datenbank |

Die Firebird-Datenbank mit dem Handle **H** wird geschlossen.

FB_002.PRF:

```
var fbDLL& = db("fbUseDLL", "fbclient.dll")
declare hdb& ' Handle der Datenbank

ifnot fileexists("XAdressen.fdb")
        hdb& = db("fbCreate", "SYSDBA", "masterkey", "XAdressen.fdb")
    Else
        hdb& = db("fbInit", "SYSDBA", "masterkey", "XAdressen.fdb")
endif

Var tmp$ = "Die Datenbank XAdressen.fdb ist geöffnet.\n\n" + \
           "Handle der Datenbank: " + str$(hdb&)
Messagebox(tmp$,"Info",4160)

db("fbDone",hdb&)

tmp$ = "Die Datenbank XAdressen.fdb wurde geschlossen.\n\n" + \
           "Handle der Datenbank: " + str$(hdb&)
Messagebox(tmp$,"Info",4160)
```


## 2.4. Das Erstellen einer Tabelle in der Datenbank

In eine SQL- bzw. Firebird-Datenbank werden alle Tabellen in die Datenbank-Datei eingebettet.

Datenbank-Befehle werden von XProfan mittels SQL-Befehlen an die Datenbank gesendet.

Das Senden von SQL-Befehlen an eine Firebird-Datenbank erfolgt in XProfan mit der Funktion **db("fbSQLExec"**.

**db("fbSQLExec",H,S,N)**

| | | |
|---|---|---|
| H: | Integer | - Handle der Datenbank |
| S: | String | - SQL-Statement |
| N: | Integer | - Modus |
| Ergebnis: | Integer | - Anzahl gefundener bzw. bearbeiteter Datensätze |

Ein SQL-Befehl wird ausgeführt. **N** hat die gleiche Bedeutung wie bei der XProfan-Funktion **db("SQLExec"**. Es werden alle Modi unterstützt.

**Modus N:**

- ★ **N = 0**: Das Ergebnis wird angezeigt. Jeder Datensatz ist eine Zeile.
- ★ **N = 1**: Das Ergebnis wird in die Listbox-Liste geschrieben und kann dann z.B. mit der **LISTBOX$-Funktion** betrachtet werden. Jeder Datensatz ist ein Eintrag in der Liste und kann maximal 32767 Zeichen enthalten. Es passen maximal 262143 Datensätze in die Liste.
- ★ **N = 2**: Das Ergebnis wird in eine Datei geschrieben. Jeder Datensatz ist eine Zeile. Die Datei wird mit **Set("SQLFile"** festgelegt. Wird kein Dateiname bestimmt, wird "SQL.DAT" im aktuellen Verzeichnis genommen.
- ★ **N > 2**: Wenn das Handle einer Gridbox angegeben wird, werden deren Spaltenüberschriften mit den Feldnamen gefüllt und die Gridbox mit dem Inhalt der Tabelle (etwaige vorherige Spalten und Inhalte der Gridbox werden gelöscht).

**Set("SQLFile",S)**

| | | |
|---|---|---|
| S: | String | - Dateiname, ggf. mit Pfad |

Hiermit wird der Name der Datei festgelegt, in die im Modus 2 die Ergebnisse geschrieben werden.

**Hinweis:**

Firebird erlaubt die Verwendung unterschiedlicher SQL-Dialekte. Es wird derzeit und somit auch in XProfan Dialekt 3 verwendet. Datenbanken, die mit früheren Dialekten erstellt wurden, sollten problemlos bearbeitet werden können. Umgekehrt geht das nicht. Dialekt 1 unterstützt z.B. noch keine 64-Bit-Integerwerte, sodass dort die Länge von NUMERIC oder DECIMAL auf 9 Stellen beschränkt ist.

Wir erzeugen in der Datenbank **XAdressen.fdb** eine (zunächst noch leere) Tabelle namens **ADRESSEN**.

Hierfür verwenden wir den SQL-Befehl **CREATE TABLE**:

```
CREATE TABLE <Tabellenname> (<Feldname> <Feldtyp> [,...])
```

Und in XProfan umgesetzt:

```
db("fbSQLExec", hdb&, "CREATE TABLE ADRESSEN \
                       ( NAME CHAR(30),\
                         VORNAME CHAR(30),\
                         STRASSE CHAR(30),\
                         PLZ CHAR(5),\
                         ORT CHAR(30),\
                         GEBDAT CHAR(10) )",1)
```

FB_003.PRF:

```
var fbDLL& = db("fbUseDLL", "fbclient.dll")
declare hdb& 'Handle der Datenbank

ifnot fileexists("XAdressen.fdb")
      hdb& = db("fbCreate", "SYSDBA", "masterkey", "XAdressen.fdb")
   Else
      hdb& = db("fbInit", "SYSDBA", "masterkey", "XAdressen.fdb")
endif

db("fbSQLExec", hdb&, "CREATE TABLE ADRESSEN \
                       ( NAME CHAR(30),\
                         VORNAME CHAR(30),\
                         STRASSE CHAR(30),\
                         PLZ CHAR(5),\
                         ORT CHAR(30),\
                         GEBDAT CHAR(10) )",1)
db("fbDone",hdb&)
```

## 2.5. Das Hinzufügen von Datensätzen

Um der Tabelle einen Datensatz hinzuzufügen, wird der SQL-Befehl INSERT verwendet:

```
INSERT INTO <Tabellenname> (<Feldliste>) VALUES (<Werteliste>)
```

In XProfan und auf unsere Tabelle bezogen:

```
db("fbSQLExec",hdb&,"INSERT INTO ADRESSEN (NAME,VORNAME,STRASSE,PLZ,ORT,GEBDAT) \
    VALUES ('Horn','Dietmar','Dorfstr. 1a','02994','Bernsdorf','18.12.1955')",1)
```

FB_004.PRF:

```
var fbDLL& = db("fbUseDLL", "fbclient.dll")
declare hdb& 'Handle der Datenbank

ifnot fileexists("XAdressen.fdb")
        hdb& = db("fbCreate", "SYSDBA", "masterkey", "XAdressen.fdb")
    Else
        hdb& = db("fbInit", "SYSDBA", "masterkey", "Adressen.fdb")
endif

db("fbSQLExec",hdb&,"INSERT INTO ADRESSEN (NAME,VORNAME,STRASSE,PLZ,ORT,GEBDAT) \
   VALUES ('Horn','Dietmar','Dorfstr. 1a','02994','Bernsdorf','18.12.1955')",1)

db("fbDone",hdb&)
```

Damit wir für die folgenden Kapitel eine etwas größere Datenbank zur Verfügung haben, schreiben wir auf dieselbe Art und Weise noch einige weitere Datensätze in die Tabelle.

FB_005.PRF:

```
var fbDLL& = db("fbUseDLL", "fbclient.dll")
declare hdb& 'Handle der Datenbank

hdb& = db("fbInit", "SYSDBA", "masterkey", "XAdressen.fdb")

db("fbSQLExec",hdb&,"INSERT INTO ADRESSEN (NAME,VORNAME,STRASSE,PLZ,ORT,GEBDAT) \
   VALUES ('Horn','Marion','Dorfstr. 1b','02994','Bernsdorf','16.10.1957')",1)

db("fbSQLExec",hdb&,"INSERT INTO ADRESSEN (NAME,VORNAME,STRASSE,PLZ,ORT,GEBDAT) \
   VALUES ('Beloch','Lothar','Hölperweg 15','02345','Nirgendwo','26.05.1965')",1)

db("fbSQLExec",hdb&,"INSERT INTO ADRESSEN (NAME,VORNAME,STRASSE,PLZ,ORT,GEBDAT) \
   VALUES ('Beloch','Marion','Holperweg 15','02345','Nirgendwo','08.01.1971')",1)

db("fbSQLExec",hdb&,"INSERT INTO ADRESSEN (NAME,VORNAME,STRASSE,PLZ,ORT,GEBDAT) \
   VALUES ('Mustermann','Max','Musterstr. 11a','04532','Musterdorf','11.11.1955')",1)

db("fbSQLExec",hdb&,"INSERT INTO ADRESSEN (NAME,VORNAME,STRASSE,PLZ,ORT,GEBDAT) \
   VALUES ('Mustermann','Angelika','Musterstr. 11a','04532','Musterdorf','01.01.1977')",1)

db("fbDone",hdb&)
```

## 2.6. Das Anzeigen von Datensätzen

Für das Anzeigen von Datensätzen ist der SQL-Befehl **SELECT** zuständig. Allein über den Select-Befehl könnte man ein ganzes Buch schreiben.

```
SELECT [ALL|DISTINCT] <Feldliste>
        FROM <Tabellenname>
        [WHERE <Bedingung>]
        [GROUP BY <Feldliste>]
        [HAVING <Bedingung>]
        [ORDER BY <Feldliste>]
```

In eckigen Klammern stehende Teile des Befehls können weggelassen werden. **ALL** oder **DISTINCT** kann wahlweise eingesetzt werden. Anstelle der ersten Feldliste kann auch ein ***** stehen, womit dann alle Felder der Tabelle angezeigt werden.

Wir werden uns hier zunächst mit einfachen Datenbankabfragen beschäftigen und möchten uns zunächst alle Datensätze unserer Tabelle **ADRESSEN** anzeigen lassen.

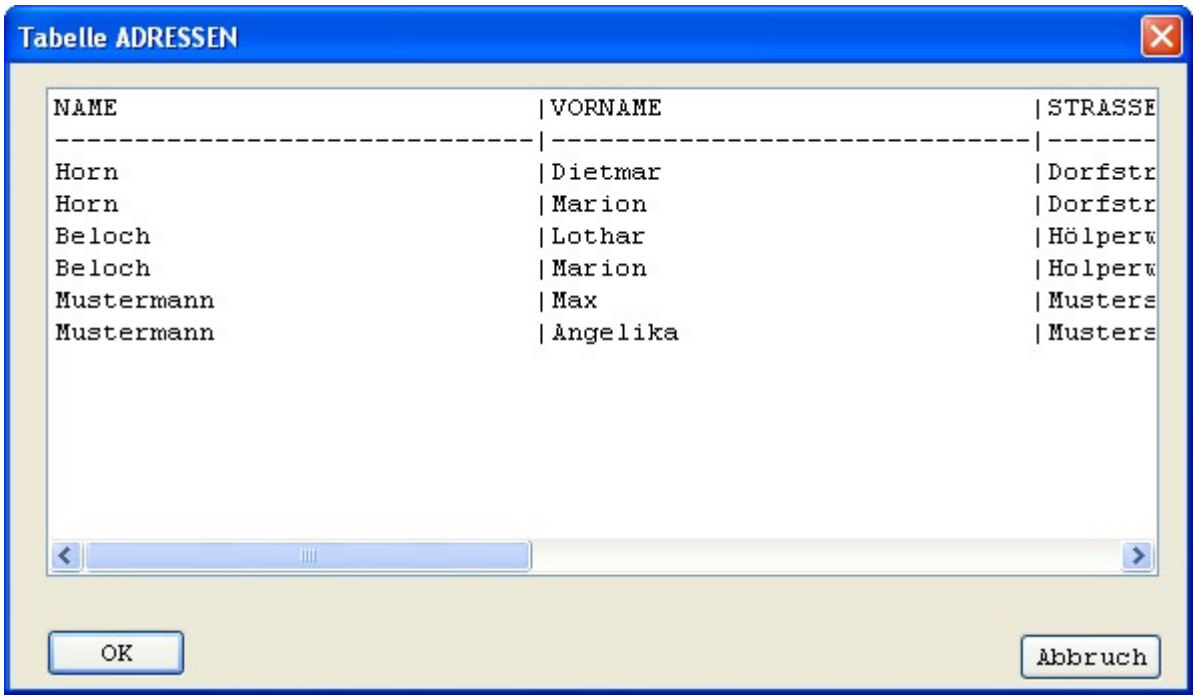
```
db("fbSQLExec",hdb&,"SELECT * FROM ADRESSEN",1
```

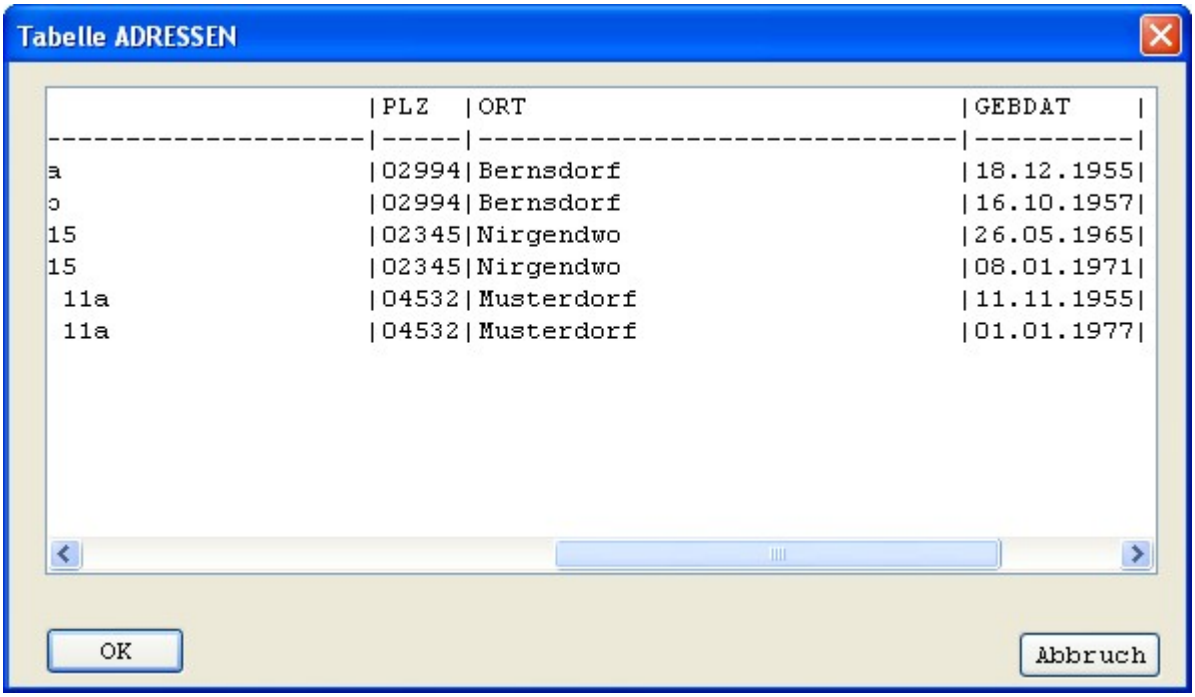
Weil wir für **db("fbSQLExec"** den **Modus 1** verwenden, schreibt XProfan das Ergebnis der Select-Anweisung in seine interne Listbox-Liste. Jedem Programmierer ist es natürlich freigestellt, den Inhalt dieser Liste nach eigenen Vorstellungen weiter zu verwenden.

Der Einfachheit halber zeigen wir im folgenden Beispiel die Datensätze unserer Tabelle mit dem vorgefertigten Listbox-Dialog von XProfan an.

FB_006.PRF:

```
var fbDLL& = db("fbUseDLL", "fbclient.dll")
declare hdb& 'Handle der Datenbank

hdb& = db("fbInit", "SYSDBA", "masterkey", "XAdressen.fdb")

db("fbSQLExec",hdb&,"SELECT * FROM ADRESSEN",1)
Listbox$("Tabelle ADRESSEN",2)

db("fbDone",hdb&)
```




Besonders hübsch sieht die grafische Anzeige der Datensätze zwar noch nicht aus, aber wir wollten ja lediglich den Inhalt unserer Adressentabelle zunächst mal sehen.

Etwas können wir die Anzeige der Datensätze in der Liste noch verbessern, indem wir nicht je Spalte die von uns in der Tabelle voreingestellten 30 Zeichen ausgeben. Für die bisher eingetragenen Adressen würden vielleicht auch maximal 15 Zeichen je Spalte in der Anzeige ausreichen.

Die Einstellung der Anzeige der maximal auszugebenden Zeichen je Spalte in der Listbox erfolgt mit dem Befehl **Set("SQLColWidth"**.

## Set("SQLColWidth",N)

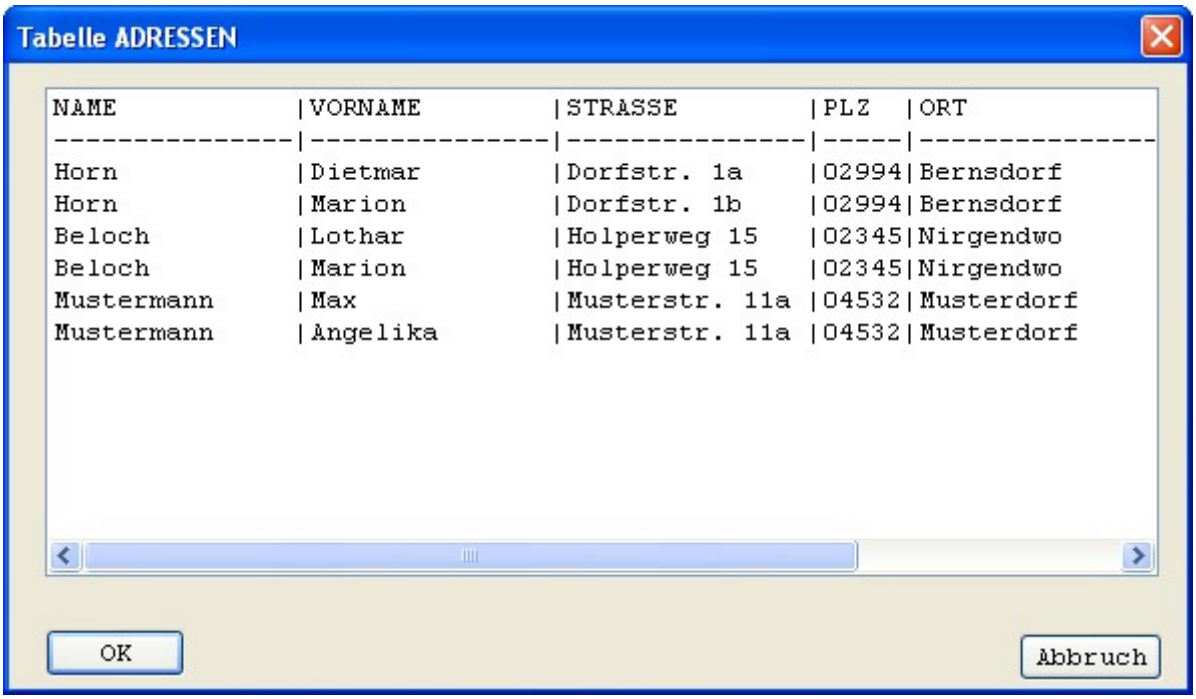
N: Integer

Mit dieser Funktion kann die maximale Spaltenbreite beim Auslesen von SQL-Daten eingestellt werden. Das ist dann besonders interessant, wenn man diese Daten in einer Tabelle oder Listbox darstellen möchte. N ist die Spaltenbreite in Zeichen. Hat N den Wert -1, gibt es keine maximale Spaltenbreite.

**Hinweis**: Unabhängig davon gibt es die Funktion **Set("SQLWidth"**, die lediglich die maximale Ausleselänge von LongVarChar (Memo-Felder) beeinflusst.

FB_009.PRF:

```
var fbDLL& = db("fbUseDLL", "fbclient.dll")
declare hdb& 'Handle der Datenbank

hdb& = db("fbInit", "SYSDBA", "masterkey", "XAdressen.fdb")

Set("SQLColWidth",15)

db("fbSQLExec",hdb&,"SELECT * FROM ADRESSEN",1)
Listbox$("Tabelle ADRESSEN",2)

db("fbDone",hdb&)
```

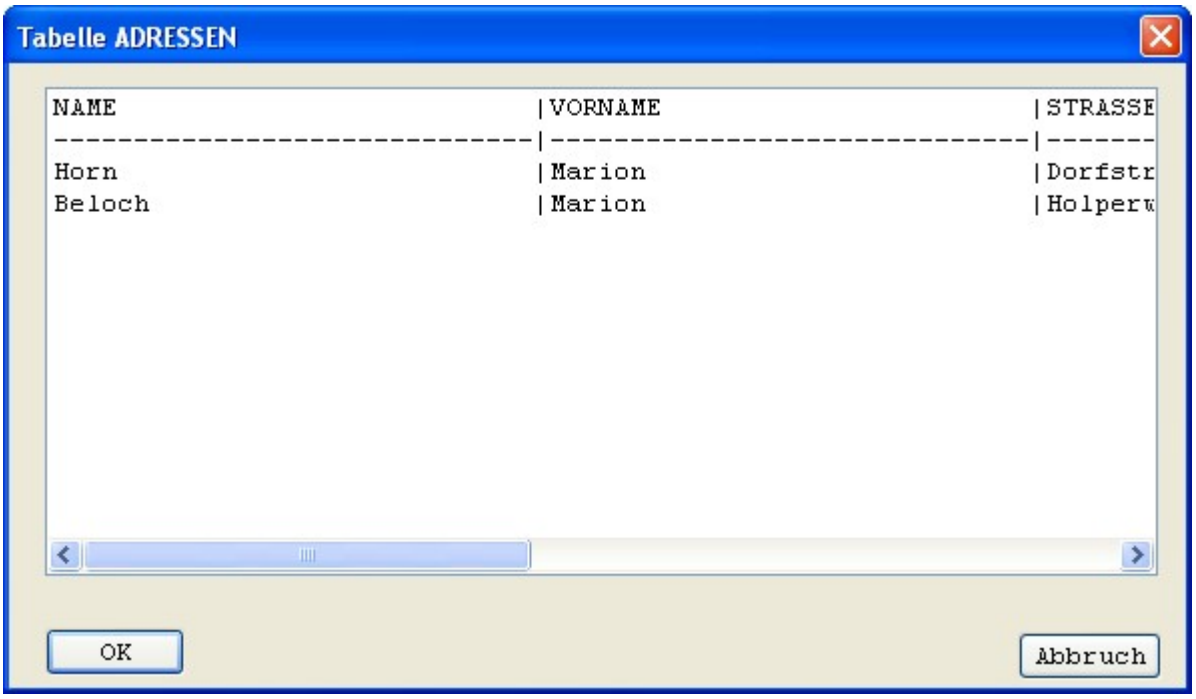


Wenn wir nur ausgewählte Datensätze angezeigt haben möchten, dann ändern wir die SELECT-Anweisung ab. Zum Beispiel möchten wir nur alle Personen mit dem Vornamen „Marion“ angezeigt bekommen:

```
db("fbSQLExec",hdb&,"SELECT * FROM ADRESSEN WHERE VORNAME = 'Marion'",1)
```

FB_007.PRF:

```
var fbDLL& = db("fbUseDLL", "fbclient.dll")
declare hdb& 'Handle der Datenbank

hdb& = db("fbInit", "SYSDBA", "masterkey", "XAdressen.fdb")

db("fbSQLExec",hdb&,"SELECT * FROM ADRESSEN WHERE VORNAME = 'Marion'",1)
Listbox$("Tabelle ADRESSEN",2)

db("fbDone",hdb&)
```



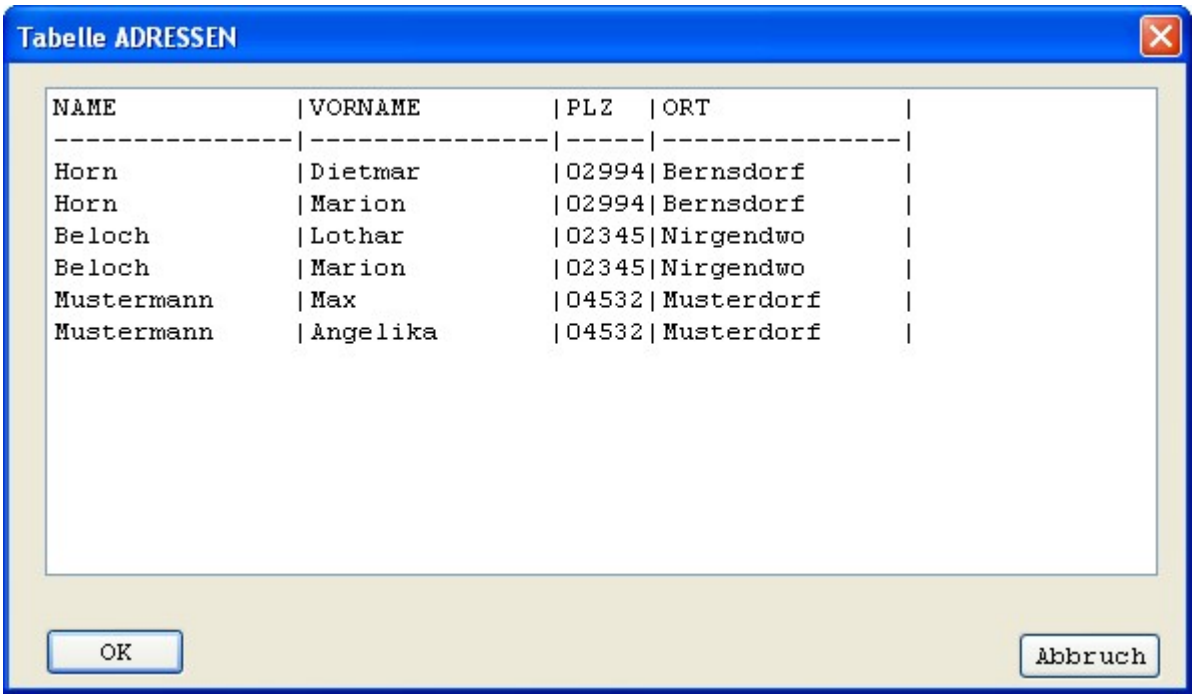
Nicht immer ist es sinnvoll, alle Spalten der Datensätze anzuzeigen.

Im folgenden Beispiel sollen nur Name, Vorname, Postleitzahl und Wohnort der Personen aus unserer Tabelle angezeigt werden.

```
db("fbSQLExec",hdb&,"SELECT NAME,VORNAME,PLZ,ORT FROM ADRESSEN",1)
```

FB_010.PRF:

```
var fbDLL& = db("fbUseDLL", "fbclient.dll")
declare hdb& 'Handle der Datenbank

hdb& = db("fbInit", "SYSDBA", "masterkey", "XAdressen.fdb")

Set("SQLColWidth",15)

db("fbSQLExec",hdb&,"SELECT NAME,VORNAME,PLZ,ORT FROM ADRESSEN",1)
Listbox$("Tabelle ADRESSEN",2)

db("fbDone",hdb&)
```

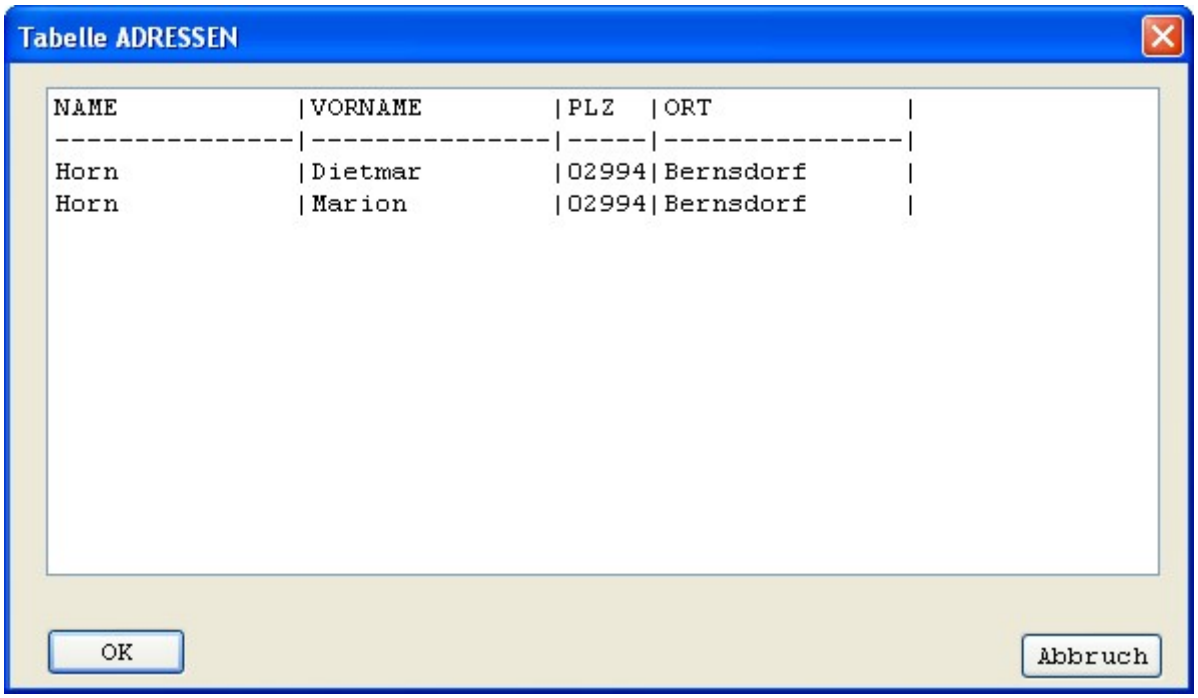


Jetzt sollen uns nur alle Mitglieder der Familie Horn angezeigt werden.

```
db("fbSQLExec",hdb&,"SELECT NAME,VORNAME,PLZ,ORT FROM ADRESSEN WHERE NAME = 'Horn'",1)
```

FB_011.PRF:

```
var fbDLL& = db("fbUseDLL", "fbclient.dll")
declare hdb& 'Handle der Datenbank

hdb& = db("fbInit", "SYSDBA", "masterkey", "XAdressen.fdb")

Set("SQLColWidth",15)

db("fbSQLExec",hdb&,"SELECT NAME,VORNAME,PLZ,ORT FROM ADRESSEN WHERE NAME = 'Horn'",1)
Listbox$("Tabelle ADRESSEN",2)

db("fbDone",hdb&)
```

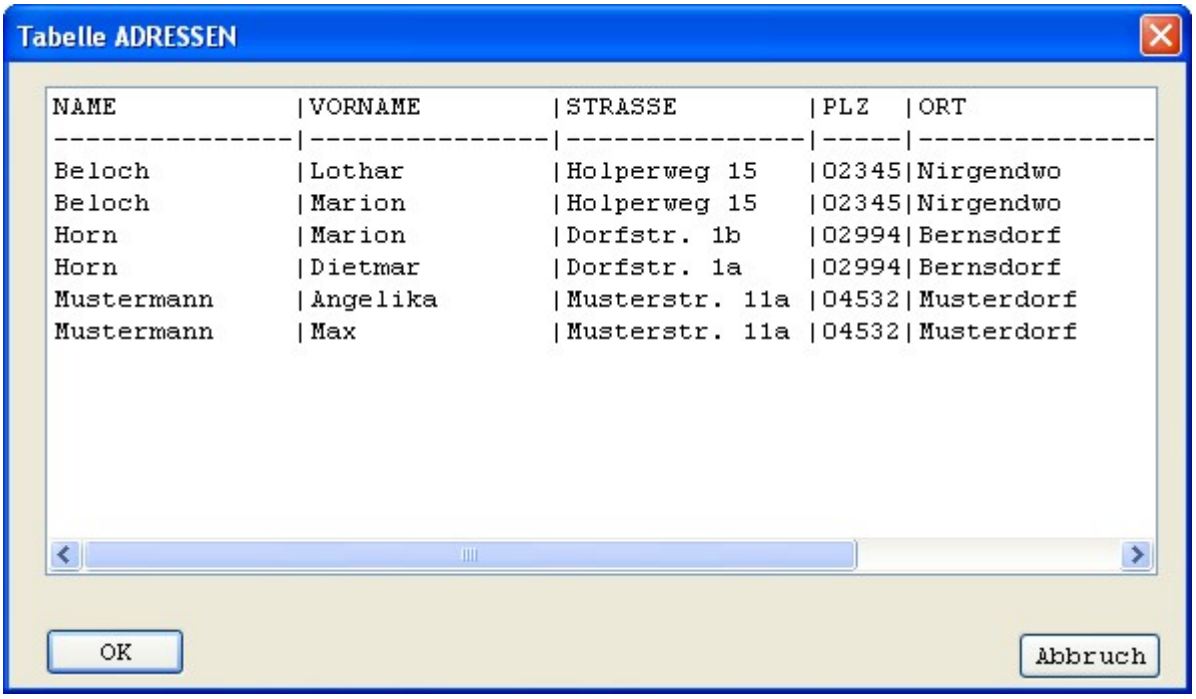


Möchten Sie alle Datensätze der Adressen-Tabelle vielleicht lieber z.B. nach den Nachnamen alphabetisch sortiert sehen? Mit der SQL-Anweisung **ORDER BY** ist auch dies kein Problem.

```
db("fbSQLExec",hdb&,"SELECT * FROM ADRESSEN ORDER BY NAME",1)
```

FB_012.PRF:

```
var fbDLL& = db("fbUseDLL", "fbclient.dll")
declare hdb& 'Handle der Datenbank

hdb& = db("fbInit", "SYSDBA", "masterkey", "XAdressen.fdb")

Set("SQLColWidth",15)

db("fbSQLExec",hdb&,"SELECT * FROM ADRESSEN ORDER BY NAME",1)
Listbox$("Tabelle ADRESSEN",2)

db("fbDone",hdb&)
```



## 2.7. Das Editieren von Datensätzen

In der Tabelle hat sich ein Tippfehler eingeschlichen. Natürlich wohnt Herr Lothar Beloch nicht im „H**ö**lperweg 15“, sondern zusammen mit seiner Familie im „H**o**lperweg 15“ in „02345 Nirgendwo“

Wie können wir dies korrigieren, ohne alle bisher eingetragenen Datensätze neu eingeben zu müssen?

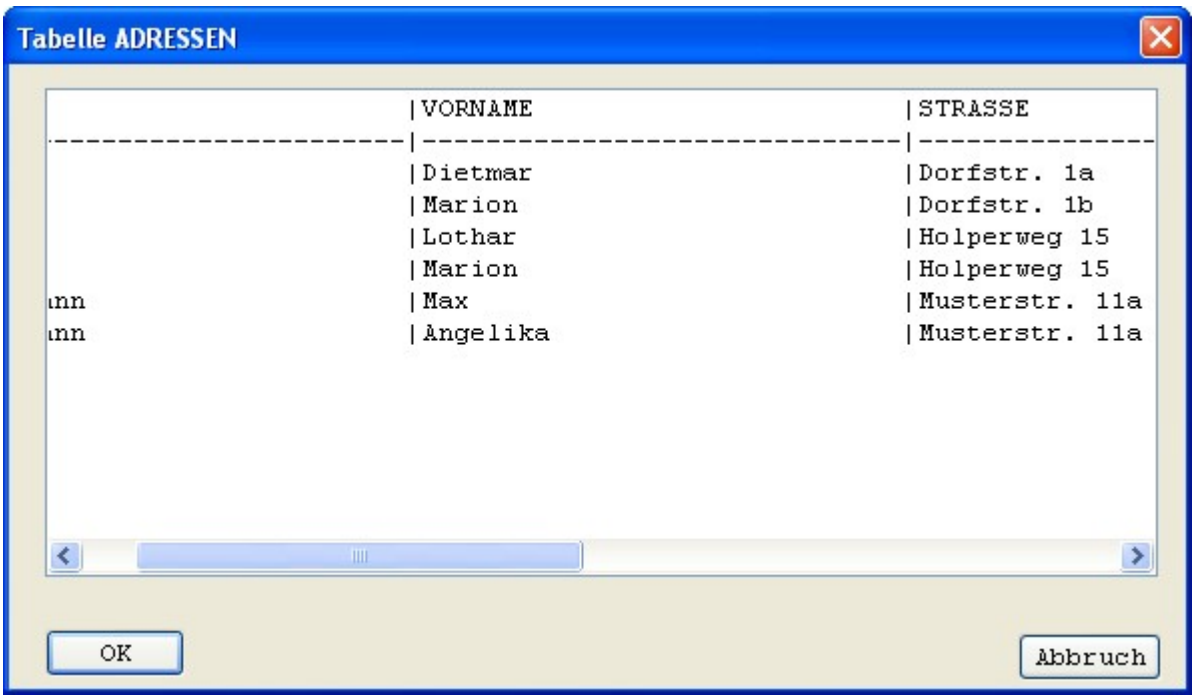
Datensätze werden mit der SQL-Anweisung **UPDATE** geändert.

```
UPDATE <Tabellenname> SET <Feldname>=Inhalt [,...] WHERE <Bedingung>
```

```
db("fbSQLExec", hdb&, "UPDATE ADRESSEN SET STRASSE = 'Holperweg 15' \
WHERE VORNAME = 'Lothar'",1
```

FB_008.PRF:

```
Var fbDLL& = db("fbUseDLL", "fbclient.dll")
declare hdb& 'Handle der Datenbank

hdb& = db("fbInit", "SYSDBA", "masterkey", "XAdressen.fdb")

db("fbSQLExec", hdb&, \
  "UPDATE ADRESSEN SET STRASSE = 'Holperweg 15' WHERE VORNAME = 'Lothar'",1)

db("fbSQLExec",hdb&,"SELECT * FROM ADRESSEN",1)
Listbox$("Tabelle ADRESSEN",2)

db("fbDone",hdb&)
```



## 2.8. Das Löschen von Datensätzen

Der SQL-Befehl DELETE löscht alle Datensätze, auf die eine oder mehrere der angeführten Bedingungen zutreffen.

```
DELETE tabelle WHERE bedingung
```

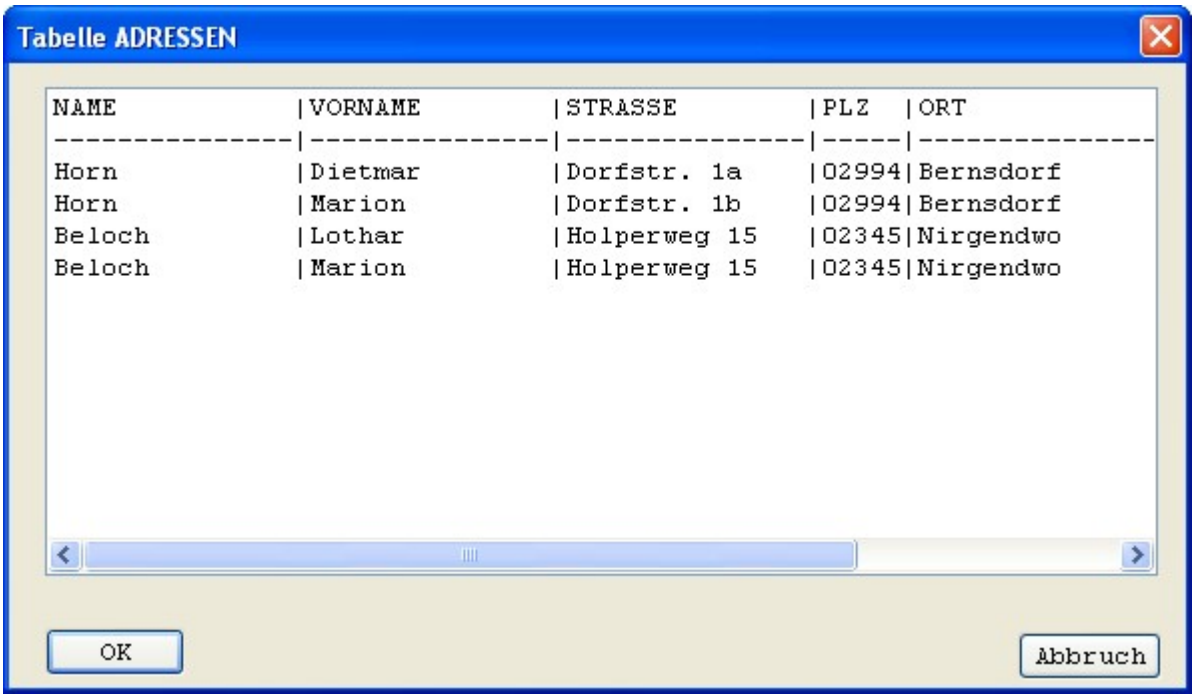
In unserer Beispieltabelle sollen alle Personen mit dem Nachnamen „Mustermann“ gelöscht werden.

```
db("fbSQLExec",hdb&,"DELETE FROM ADRESSEN WHERE NAME = 'Mustermann'",1)
```

Sichern Sie bitte unbedingt vor der Ausführung der folgenden beiden Demos jeweils die Datenbank XAdressen.fdb, damit Sie für weitere Beispiele nicht immer die gelöschten Datensätze neu eingeben müssen.

FB_013.PRF:

```
var fbDLL& = db("fbUseDLL", "fbclient.dll")
declare hdb& 'Handle der Datenbank

hdb& = db("fbInit", "SYSDBA", "masterkey", "XAdressen.fdb")

Set("SQLColWidth",15)

db("fbSQLExec",hdb&,"DELETE FROM ADRESSEN WHERE NAME = 'Mustermann'",1)

db("fbSQLExec",hdb&,"SELECT * FROM ADRESSEN",1)
Listbox$("Tabelle ADRESSEN",2)

db("fbDone",hdb&)
```



Wie Sie sehen, wurden alle Daten der Familie Mustermann aus der Tabelle entfernt.
Kopieren Sie nun Ihre Sicherheitskopie XAdressen.fbd zurück.

Soll gezielt nur ein einziger Datensatz entfernt werden, dann sind mehrere Bedingungen zu formulieren, sodass der gewünschte zu löschende Datensatz eindeutig zugeordnet werden kann.
Auch in solchen Fällen können in SQL, wie in XProfan, mehrere Bedingungen mit **AND** (und) bzw. **OR** (oder) verknüpft werden.

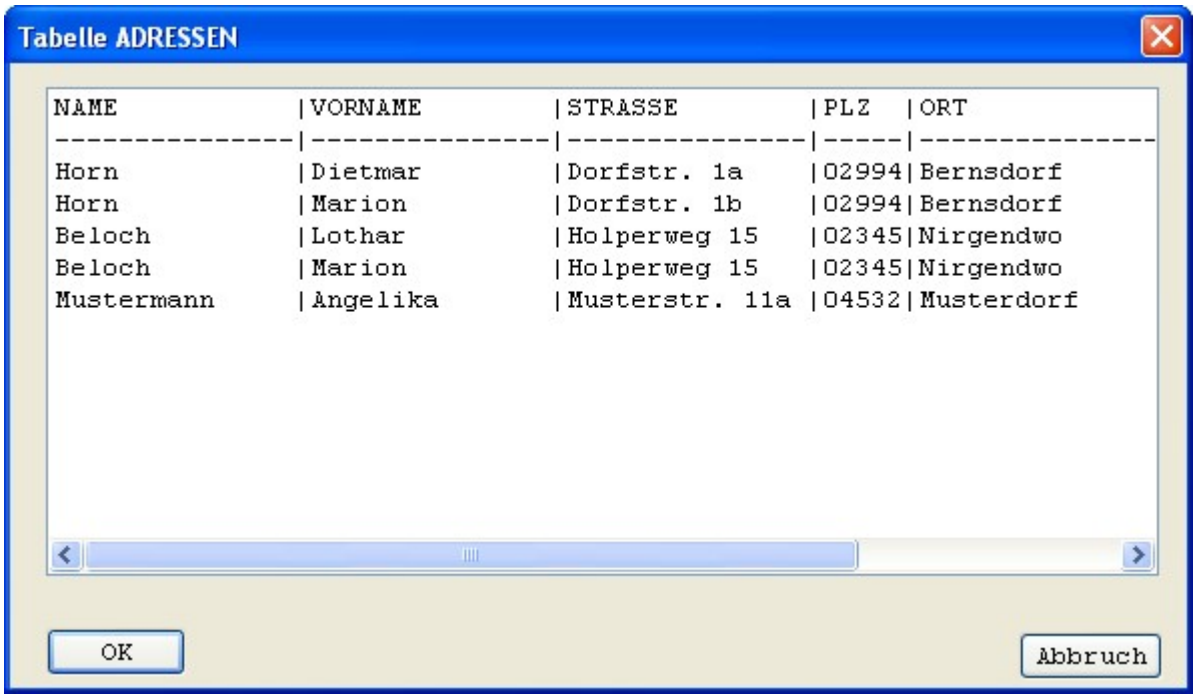
```
DELETE FROM tabelle WHERE bedingung1 AND bedingung2
```

Es sollen nur die Daten von Max Mustermann gelöscht werden.

```
Var sql$ = "DELETE FROM ADRESSEN WHERE NAME = 'Mustermann' AND VORNAME = 'Max'"
db("fbSQLExec",hdb&,sql$,1
```

FB_014.PRF:

```
var fbDLL& = db("fbUseDLL", "fbclient.dll")
declare hdb& 'Handle der Datenbank

hdb& = db("fbInit", "SYSDBA", "masterkey", "XAdressen.fdb")

Set("SQLColWidth",15)

Var sql$ = "DELETE FROM ADRESSEN WHERE NAME = 'Mustermann' AND VORNAME = 'Max'"
db("fbSQLExec",hdb&,sql$,1)

db("fbSQLExec",hdb&,"SELECT * FROM ADRESSEN",1)
Listbox$("Tabelle ADRESSEN",2)

db("fbDone",hdb&)
```



## 2.9. Das Löschen einer Tabelle

Soll eine Tabelle unwiderruflich aus einer Datenbank gelöscht werden (aus welchen Gründen auch immer), ist die SQL-Anweisung **DROP** zu verwenden.

```
DROP TABLE tabelle
```

Natürlich sind nach dem Löschen einer Tabelle auch alle darin enthaltenen Daten verloren!

```
db("SQLExec","DROP TABLE ADRESSEN",0)
```

Dies kann sinnvoll sein, wenn z.B. alle Datensätze einer Tabelle gelöscht werden sollen:

- ★ Tabelle mit DROP löschen.
- ★ Tabelle neu erstellen.

FB_015.PRF:

```
var fbDLL& = db("fbUseDLL", "fbclient.dll")
declare hdb&,sql$

hdb& = db("fbInit", "SYSDBA", "masterkey", "XAdressen.fdb")

Set("SQLColWidth",15)

sql$ = "DROP TABLE ADRESSEN"
db("fbSQLExec",hdb&,sql$,1)

sql$ = "CREATE TABLE ADRESSEN \
     ( NAME CHAR(30),\
       VORNAME CHAR(30),\
       STRASSE CHAR(30),\
       PLZ CHAR(5),\
      ORT CHAR(30),\
      GEBDAT CHAR(10) )"
db("fbSQLExec",hdb&,sql$,1)

db("fbSQLExec",hdb&,"SELECT * FROM ADRESSEN",1)
Listbox$("Tabelle ADRESSEN",2)

db("fbDone",hdb&)
```

# 3. XProfan-Variablen in SQL-Anweisungen verwenden

In einem fertigen Programm können wir dem Anwender unseres Programmes nicht zumuten, z.B. neue Datensätze im SQL- bzw. XProfan-Code einzugeben, sondern wir programmieren einen Eingabedialog und speichern die eingegebenen Daten in Variablen.

Doch wie übergeben wir diese Variablen-Inhalte z.B. an die SQL-INSERT-Anweisung?

Auch hier hat der XProfan-Autor clever mitgedacht und uns die Möglichkeit geschaffen, Variableninhalte durch Angabe der Variablennamen mit vorangestelltem Doppelpunkt an die SQL-Anweisung zu übergeben.

```
INSERT INTO <Tabellenname> (<Feldliste>) VALUES (<Werteliste>)
```

Statt:

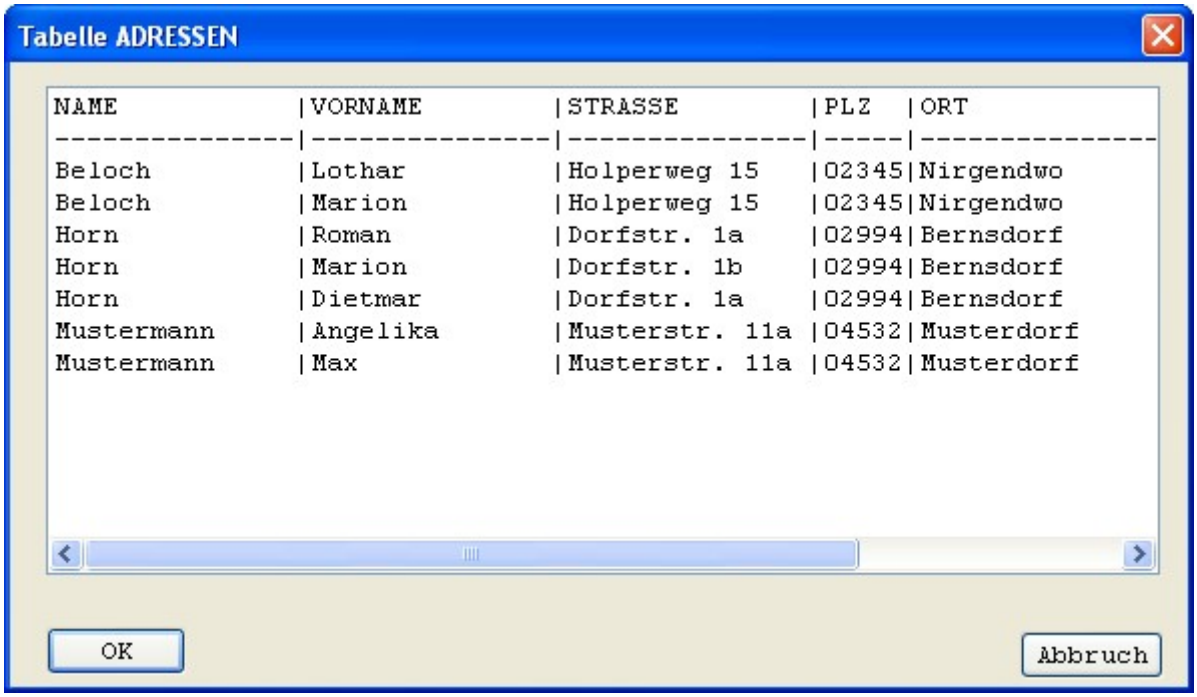
```
db("fbSQLExec",hdb&,"INSERT INTO ADRESSEN (NAME,VORNAME,STRASSE,PLZ,ORT,GEBDAT) \
    VALUES ('Horn','Roman','Dorfstr. 1a','02994','Bernsdorf','11.11.1997')",1)
```

können wir schreiben:

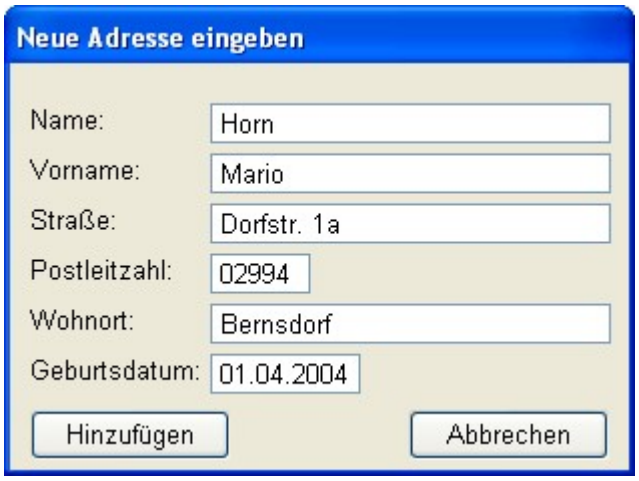
```
db("fbSQLExec",hdb&,"INSERT INTO ADRESSEN (NAME,VORNAME,STRASSE,PLZ,ORT,GEBDAT) \
   VALUES (:Nane$,:Vorname$,:Strasse$,:Plz$,:Ort$,:Gebdat$)",1)
```

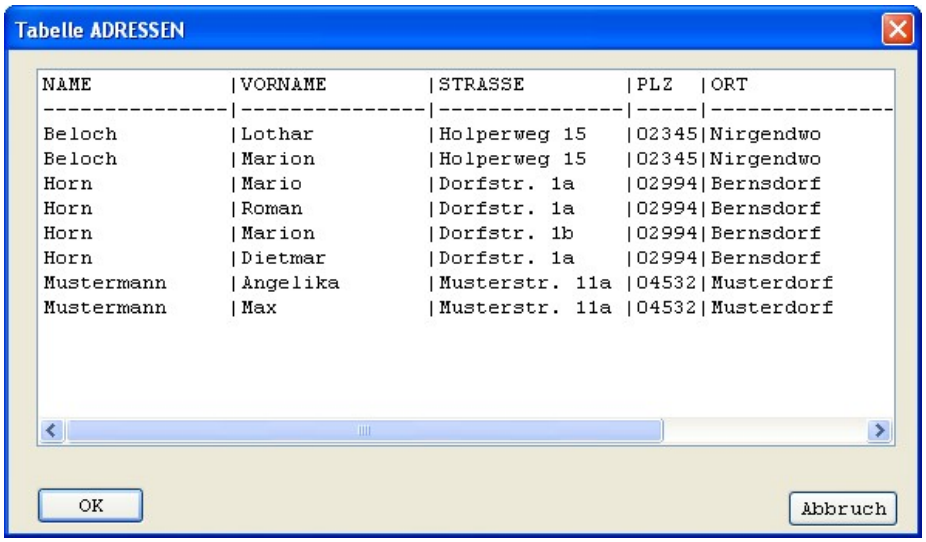
FB_016.PRF:

```
var fbDLL& = db("fbUseDLL", "fbclient.dll")
declare hdb& 'Handle der Datenbank
Declare Name$,Vorname$,Strasse$,Plz$,Ort$,Gebdat$

Name$       = "Horn"
Vorname$    = "Roman"
Strasse$    = "Dorfstr. 1a"
Plz$        = "02994"
Ort$        = "Bernsdorf"
Gebdat$     = "11.11.1997"

Set("SQLColWidth",15)

hdb& = db("fbInit", "SYSDBA", "masterkey", "XAdressen.fdb")

db("fbSQLExec",hdb&,"INSERT INTO ADRESSEN (NAME,VORNAME,STRASSE,PLZ,ORT,GEBDAT) \
   VALUES (:Name$,:Vorname$,:Strasse$,:Plz$,:Ort$,:Gebdat$)",1)

db("fbSQLExec",hdb&,"SELECT * FROM ADRESSEN ORDER BY NAME",1)
Listbox$("Tabelle ADRESSEN",2)

db("fbDone",hdb&)
```




Mit den bisher erworbenen SQL- bzw. Firebird-Kenntnissen programmieren wir uns einen Eingabe-Dialog zur Erweiterung unserer Adressen-Tabelle.



FB_017.PRF:

```
var fbDLL& = db("fbUseDLL", "fbclient.dll")
declare hdb& 'Handle der Datenbank
Declare Name$,Vorname$,Strasse$,Plz$,Ort$,Gebdat$
Declare Name&,Vorname&,Strasse&,Plz&,Ort&,Gebdat&
Declare e%,x%,y%
Declare Save&,Exit&

Def GetSysColor(1) !"USER32","GetSysColor"

Set("SQLColWidth",15)
hdb& = db("fbInit", "SYSDBA", "masterkey", "XAdressen.fdb")

x% = 315
y% = 235

WindowTitle "Neue Adresse eingeben"
WindowStyle 48 + 512
Window (%MaxX - x%) / 2,(%MaxY - y%) / 2 - x%,y%
UseFont "Arial",16,0,0,0,0
SetDialogFont 1
CLS GetSyscolor(15)

Create("Text",%HWnd,"Name:",10,20,75,20)
Name& = Create("Edit",%HWnd,"",100,20,200,20)

Create("Text",%HWnd,"Vorname:",10,20 + 25,75,20)
Vorname& = Create("Edit",%HWnd,"",100,20 + 25,200,20)

Create("Text",%HWnd,"Straße:",10,20 + 2 * 25,75,20)
Strasse& = Create("Edit",%HWnd,"",100,20 + 2 * 25,200,20)

Create("Text",%HWnd,"Postleitzahl:",10,20 + 3 * 25,75,20)
Plz& = Create("Edit",%HWnd,"",100,20 + 3 * 25,50,20)

Create("Text",%HWnd,"Wohnort:",10,20 + 4 * 25,75,20)
Ort& = Create("Edit",%HWnd,"",100,20 + 4 * 25,200,20)

Create("Text",%HWnd,"Geburtsdatum:",10,20 + 5 * 25,85,20)
Gebdat& = Create("Edit",%HWnd,"",100,20 + 5 * 25,75,20)

Save& = Create("Button",%HWnd,"Hinzufügen",10,Height(%HWnd) -30,100,25)
Exit& = Create("Button",%HWnd,"Abbrechen",Width(%HWnd) - 110,Height(%HWnd) -30,100,25)
SetFocus(Name&)

WhileNot e%
        Waitinput
        If Clicked(exiT&) or (%Key= 2)
                e% = 1
                db("fbDone",hdb&)
          ElseIf Clicked(Save&)
                SPEICHERN
        EndIf
EndWhile

Proc SPEICHERN
    Name$       = trim$(GetText$(Name&))
    Vorname$    = trim$(GetText$(Vorname&))
    Strasse$    = trim$(GetText$(Strasse&))
    Plz$        = trim$(GetText$(Plz&))
    Ort$        = trim$(GetText$(Ort&))
    Gebdat$     = trim$(GetText$(Gebdat&))

    db("fbSQLExec",hdb&,"INSERT INTO ADRESSEN (NAME,VORNAME,STRASSE,PLZ,ORT,GEBDAT) \
    VALUES (:Name$,:Vorname$,:Strasse$,:Plz$,:Ort$,:Gebdat$)",1)

    db("fbSQLExec",hdb&,"SELECT * FROM ADRESSEN ORDER BY NAME",1)
    Listbox$("Tabelle ADRESSEN",2)

    e% = 1
    db("fbDone",hdb&)
EndProc
```

# 4. Das Anzeigen der Datensätze in einer XProfan-Gridbox

Die bisherige grafische Darstellung der Datensätze war recht dürftig gewesen.
Mit einer XProfan-GridBox können wir das jedoch ändern. Eine XProfan-GridBox ist nichts anderes als ein normales Windows-ListView.

Mit dee SQL-SELECT-Anweisung werden die Datensätze aus der Tabelle ausgelesen und durch den Parameter 1 bei **db("fbSQLExec"** in die interne (unsichtbare) XProfan-Listboxliste kopiert, wobei die Felder jedes Datensatzes durch „|“ getrennt sind. Mit **ListboxItem$(** lesen wir die einzelnen Datensätze aus der Listboxliste in einer **While**-Schleife aus. Da uns das „|“ als Trennzeichen zwischen den einzelnen Feldern bekannt ist, können wir die Feld-Inhalte mit Substr$( nacheinander auslesen und die zuvor deklarierten Stringvariablen **Name$**, **Vorname$**, **Strasse$**,**Ort$** mit den Feld-Inhalten zur Anzeige in der Gridbox bestücken. Den durch die Variablen-Inhalte zusammengesetzten String (wiederum mit „|“ als Trennzeichen) übergeben wir mit **AddString(** der Gridbox nit dem Handle **hGrid&**,

Der Befehl

```
SelectString(hGrid&,0,"")
```

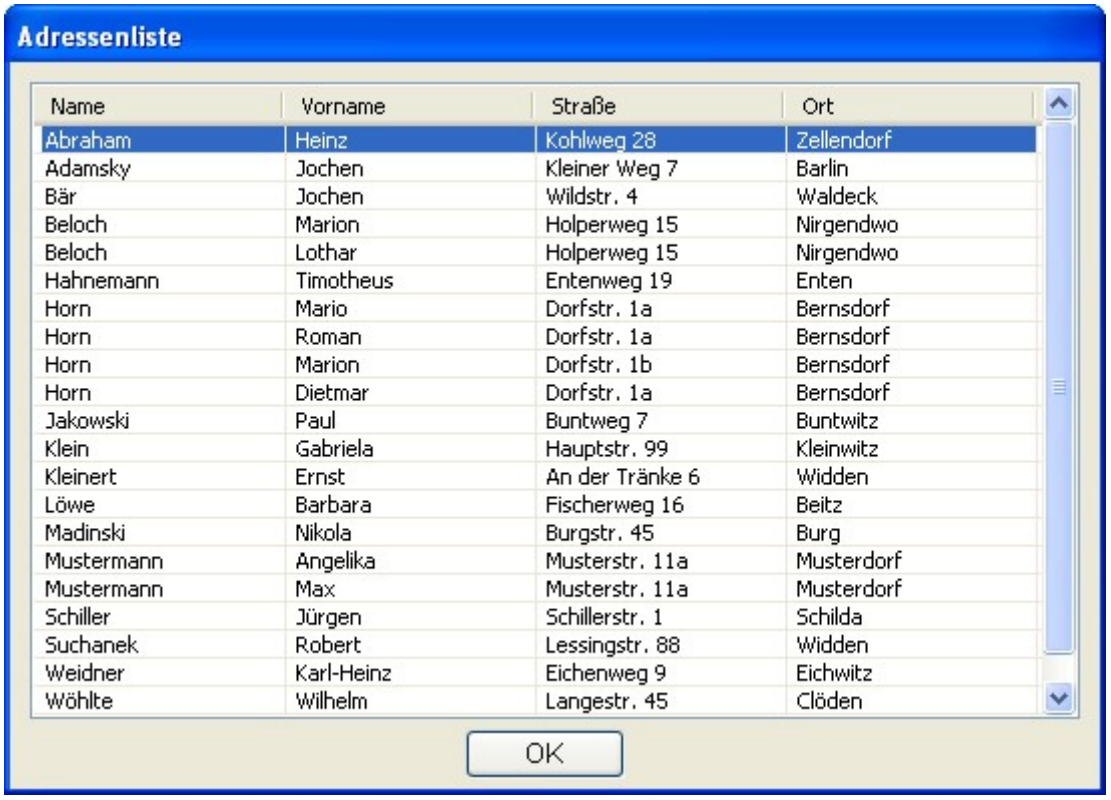
dient lediglich dazu, den ersten Eintrag in der Gridbox mit dem Index 0 zu markieren.

FB_018.PRF:

```
var fbDLL& = db("fbUseDLL", "fbclient.dll")
Declare hdb& 'Handle der Datenbank
Declare e%,x%,y%,Nr%,i%,tmp$
Declare Exit&,hGrid&
Declare Name$,Vorname$,Strasse$,Ort$

Def GetSysColor(1) !"USER32","GetSysColor"

Set("SQLColWidth",15)
hdb& = db("fbInit", "SYSDBA", "masterkey", "XAdressen.fdb")

x% = 550
y% = 395

WindowTitle "Adressenliste"
WindowStyle 48 + 512
Window 5 * %MaxX + (%MaxX - x%) / 2,(%MaxY - y%) / 2 - x%,y%
UseFont "Arial",16,0,0,0,0
SetDialogFont 1
CLS GetSyscolor(15)

Var gt$ = "Name;0;125;Vorname;0;125;Straße;0;125;Ort;0;125"
hGrid& = Create("Gridbox",%Hwnd,gt$,1,10,10,Width(%HWnd) - 20,Height(%HWnd) - 45)

Exit& = Create("Button",%HWnd,"OK",(Width(%HWnd) - 90) / 2,Height(%HWnd) -30,80,25)
```

Fortsetzung FB_018.PRF:

```
db("fbSQLExec",hdb&,"SELECT * FROM ADRESSEN",1)

Nr% = %Getcount
i% = 2
While i% < Nr% + 1
    Name$    = Substr$(Listboxitem$(i%),1,"|")
    Vorname$ = Substr$(Listboxitem$(i%),2,"|")
    Strasse$ = Substr$(Listboxitem$(i%),3,"|")
    Ort$     = Substr$(Listboxitem$(i%),5,"|")

    AddString(hGrid&,Name$ + "|" + Vorname$ + "|" + Strasse$ + "|" + Ort$)
    inc i%
EndWhile

SetWindowpos %HWnd = (%MaxX - x%) / 2,(%MaxY - y%) / 2 - x%,y%;0
SelectString(hGrid&,0,"")

WhileNot e%
        Waitinput
        If Clicked(exit&) or (%Key= 2)
                e% = 1
                db("fbDone",hdb&)
        EndIf
EndWhile
```



# 5. BLOBs bzw. binäre Dateien in die Firebird-Datenbank einbinden

Dieses 5. Kapitel enthält Beispiele und Texte aus der aktuellen XProfan-X2-Hilfedatei (Autor: **Roland G. Hülsmann**).

Für die Demos verwenden wir hier weiterhin unsere bereits vorhandene Datenbank **XAdressen.fdb**.

## 5.1. Allgemeines zu BLOBs

**B**inary **L**arge **Ob**jects (**BLOB**s) sind große binäre Objekte wie z.B. Bild- oder Audiodateien. Text-BLOBs hingegen enthalten Zeichenketten.

Ein großer Vorteil von BLOBs besteht darin, dass die Dateien der Datenbank nicht einzeln beigelegt werden müssen, sondern als Felder einer Tabelle in die Datenbank-Datei direkt eingebettet sind.

In XProfan sind für das Einbinden und Lesen von BLOBs in Verbindung mit Firebird im Wesentlichen die Funktionen **db("fbPutBlob"** und **db("fbGetBlob"** zuständig.

### db("fbPutBlob",H,S1,[S2|B])

| | | |
|---|---|---|
| H: | Integer | - Handle der Datenbank |
| S: | String | - SQL-Statement |
| S1: | String: | - Text-Blob |
| S2: | Bereich: | - binäres Blob |

Mit dem SQL-Befehl wird ein Blob (Binary large object) in die Datenbank geschrieben. Der SQL-Befehl muss ein **INSERT** oder **UPDATE** sein, der exakt ein Blob-Feld enthält. Der Wert des Blob-Feldes wird dabei durch ein Fragezeichen ersetzt. Der Blob selbst folgt im nächsten Parameter als Zeichenkette oder Bereich. Auf diese Weise zählt die Länge des Blobs nicht zur Länge des SQL-Befehles, die in Firebird auf 64 kByte beschränkt ist.

**Beispiele:**

```
db("fbPutBlob",db&,"UPDATE bilderliste SET bild = ? WHERE bildnr = 1", Bild#)

db("fbPutBlob", db&, "INSERT INTO bilderliste (kommentar) VALUES (?)", Kommentar$)

db("fbPutBlob", db&, "INSERT INTO bilderliste VALUES ('Helloween Muffins','Annemarie Hülsmann',1,?)", Bild#)
```

Wie das dritte Beispiel zeigt, dürfen die Statements durchaus auch weitere Felder des Satzes behandeln, aber nur ein Blob-Feld. Das Blob kann ein Bereich oder ein String (Zeichenkette) sein. Da in XProfan Zeichenketten alle Zeichen enthalten dürfen, könnte auch eine Zeichenkette binäre Daten enthalten.

## db("fbGetBlob",H,S)

| | | |
|---|---|---|
| H: | Integer | - Handle der Datenbank |
| S: | String | - SQL-Statement |
| Ergebnis: | String | - Blob (binär oder Text) |

Der SQL-Befehl ist ein **SELECT** und darf nur exakt ein Feld abfragen, und dies muss ein Blob-Feld sein. Anderenfalls erfolgt eine Fehlermeldung. Sollte der SELECT mehrere Zeilen als Ergebnis haben, wird nur das Blob der ersten Zeile ermittelt.

**Beispiel:**

```
Bild$ = db("fbGetBlob",db&,"SELECT bild FROM bilderliste WHERE bildnr = 1")
```

Das Blob wird immer als String zurückgegeben. Wenn es ein binäres Blob ist, kann es einfach einer Bereichsvariablen übergeben werden, etwa um es als Datei abzuspeichern:

```
Dim Bild#,len(Bild$)
Char Bild#,0 = Bild$
```

Das Abspeichern als Datei könnte mit **BlockWrite** erfolgen:

```
BlockWrite "Bild.jpg",Bild#,SizeOf(Bild#)
```

Reine Text-Blobs können auch per SELECT in einem normalen db("fbSQLExec" ermittelt werden.

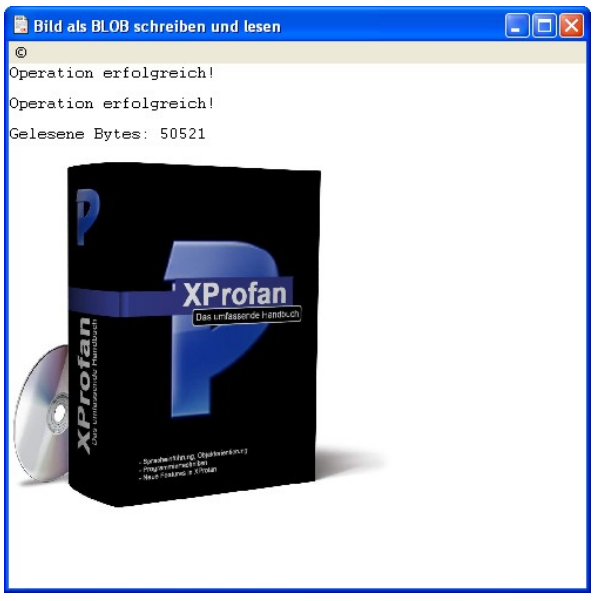
## 5.2. Binäre BLOBs

Im folgenden Beispiel wird gezeigt, wie ein Bild in einer Tabelle als BLOB gespeichert wird.

**db("fbGetBlob"** liefert einen String zurück, der die binären Daten enthält. Zur Weiterverarbeitung ist es sinnvoll, diesen in einen Bereich zu übertragen, wie dies im Demo gezeigt wird.

In XProfan können Strings beliebige Zeichen, einschließlich des Null-Bytes beinhalten.

Als Beispiel verwenden wir hier das Titelbild meines XProfan-Lehrbuches **XLehrbuch.jpg**.

FB_019.PRF:

```
declare sql$, hdb&, bild#

WindowTitle "Bild als BLOB schreiben und lesen"
window 500,500
cls

db("fbUseDLL","fbclient.dll")

' Datenbank erzeugen, falls nicht vorhanden, und Verbindung herstellen
ifnot fileexists("XAdressen.fdb")
  hdb& = db("fbCreate", "SYSDBA", "masterkey", "XAdressen.fdb")
     else
  hdb& = db("fbInit", "SYSDBA", "masterkey", "XAdressen.fdb")
endif

Set("Errorlevel",-1)
sql$="DROP TABLE Bilder"
db("fbSQLExec",hdb&,sql$, 0)
Set("Errorlevel", 0)

sql$="CREATE TABLE Bilder (Bild BLOB)"
db("fbSQLExec",hdb&,sql$,0)

' Bild einlesen
var datei$ = "XLehrbuch.jpg"

Dim bild#, FileSize(datei$)
BlockRead(datei$, bild#, 0, FileSize(datei$))

sql$ = "INSERT INTO Bilder (bild) VALUES (?)"
db("fbPutBlob",hdb&,sql$,Bild#)

sql$ = "SELECT bild FROM Bilder"
Var text$ = db("fbGetBlob",hdb&,sql$)

print "Gelesene Bytes:", len(Text$)

Dim bild#, len(Text$)
Char bild#, 0 = Text$
blockwrite "XLehrbuch2.jpg", bild#, 0, SizeOf(bild#)

DrawSizedPic "XLehrbuch2.jpg", 0, 80 - 325, 300, 0

db("fbDone", hdb&)

waitinput
end
```

## 5.3. Text-BLOBs

Das Lesen von Textblobs funktioniert mit **db("fbGetBlob"** aber auch mit **db("fbSQLExec"**.

Damit der Blob auch mit **db("fbSQLExec"** gelesen werden kann, muss er als Textblob angelegt worden sein.

Um ein Feld als Textblob zu erzeugen, muss es als "**BLOB SUB_TYPE 1**" definiert werden. Wenn man nur "BLOB" angibt, wird es als binäres Blob angelegt.

FB_020.PRF:

```
declare sql$,hdb&

WindowTitle "Text-BLOB schreiben und lesen"
window 500,500
cls
Var fbDLL& = db("fbUseDLL", "fbclient.dll")

' Datenbank erzeugen, falls nicht vorhanden, und Verbindung herstellen
IfNot fileexists("XAdressen.fdb")
  hdb& = db("fbCreate", "SYSDBA", "masterkey", "XAdressen.fdb")
     else
  hdb& = db("fbInit", "SYSDBA", "masterkey", "XAdressen.fdb")
endif
Set("Errorlevel",-1)
sql$="DROP TABLE Test"
db("fbSQLExec",hdb&,sql$,0)
Set("Errorlevel",1)

sql$="CREATE TABLE Test (Feld1 BLOB SUB_TYPE 1)"
db("fbSQLExec",hdb&,sql$, 0)

' Teststring 4 554 030 Bytes
var Text$ = "Das ist ein extrem l" + mkstr$("a",4554000) + "nger Text!"

' BLOB schreiben
sql$ = "INSERT INTO Test (Feld1) VALUES (?)"
db("fbPutBlob",hdb&,sql$,Text$)

' BLOB mit Blob-Funktion lesen
sql$ = "SELECT Feld1 FROM Test"
text$ = db("fbGetBlob", hdb&, sql$)

print "Gelesene Bytes:", len(Text$)
print "Erste 30      :", left$(Text$,30)
print "Letzte 30     :", right$(Text$,30)

' BLOB mit Select lesen
sql$="SELECT Feld1 FROM Test"
db("fbSQLExec",hdb&,sql$,1)

' erste beide Einträge aus Listboxliste entfernen
DeleteString(0,0)
DeleteString(0,0)

' erstes Feld aus erstem Eintrag auslesen
Text$=trim$(substr$(listboxitem$(0),1,"|"))

print " "
print "Gelesene Bytes:", len(Text$)
print "Erste 30      :", left$(Text$,30)
print "Letzte 30     :", right$(Text$,30)

db("fbDone", hdb&)
FreeDll fbDLL&

waitinput
end
```

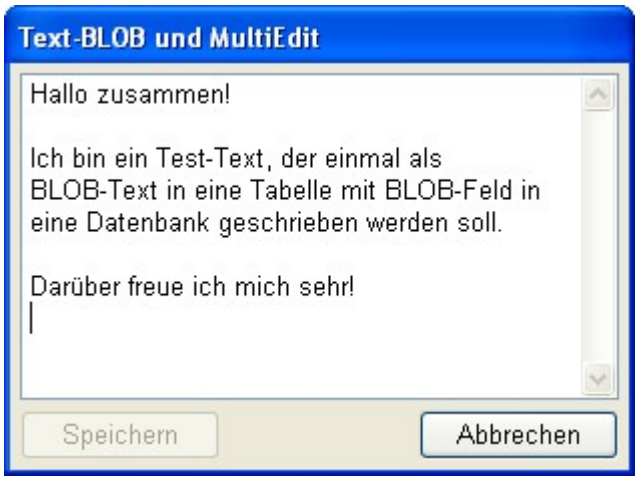

## 5.4. Text-BLOBs und MultiEdit

So richtig interessant wird die Sache mit den Text-BLOBs, wenn man diese vom eigenen Programm (z.B. von einem MultiEdit aus) in ein BLOB-Feld einer Tabelle in die Datenbank schreiben und auch wieder aus der Datenbank in das MultiEdit einlesen kann (z.B. für Notizfelder in einer Adressenverwaltung)..

Für die nachfolgenden Beispiele erstellen wir zunächst ein Fenster mit einem MultiEdit.

FB_021.PRF:

```
Declare e%,x%,y%
Declare MEdit&,Save&,Exit&

Def GetSysColor(1) !"USER32","GetSysColor"

x% = 315
y% = 235

WindowTitle "Text-BLOB und MultiEdit"
WindowStyle 48 + 512
Window (%MaxX - x%) / 2,(%MaxY - y%) / 2 - x%,y%
UseFont "Arial",16,0,0,0,0
SetDialogFont 1
CLS GetSyscolor(15)

MEdit& = Create("MultiEdit",%HWnd,"",5,5,Width(%HWnd) -10,(Height(%HWnd) - 40) * (-1))

Save& = Create("Button",%HWnd,"Speichern",5,Height(%HWnd) - 30,100,25)
EnableWindow Save&,0
Exit& = Create("Button",%HWnd,"Abbrechen",Width(%HWnd) - 105,Height(%HWnd) - 30,100,25)
SetFocus(MEDit&)

Clear e%
WhileNot e%
    Waitinput
    If Clicked(Exit&) or (%Key = 2)
                e% = 1
    EndIf
EndWhile
end
```




Zur weiteren Vorgehensweise:

- ★ In der bereits vorhandenen Datenbank **XAdressen.fdb** erzeugen wir eine neue Tabelle namens **TEXTBLOBS**.
- ★ Als Tabellenfelder beschränken wir uns zunächst auf je ein Feld **TEXTNR** und **BLOBTEXT**.
- ★ In den ersten Datensatz schreiben wir in das Feld TEXTNR "Text 1“ und in das Feld BLOB-TEXT einen Leerstring.

Weil unsere bisherigen Tests mit binären BLOBs und Text-BLOBs die Datenbank XAdressen.fdb aufgebläht haben, löschen wir zunächst die darin enthaltenen Tabellen namens **Bilder** und **Test**. Diese Tabellen benötigen wir im Folgenden nicht mehr – lediglich die Tabelle **Adressen** mit schon einer Menge eingegebener Datensätzen lassen wir unangetastet.

FB_022.PRF:

```
declare sql$, hdb&

db("fbUseDLL","fbclient.dll")

' Datenbank erzeugen, falls nicht vorhanden, und Verbindung herstellen
ifnot fileexists("XAdressen.fdb")
  hdb& = db("fbCreate", "SYSDBA", "masterkey", "XAdressen.fdb")
     else
  hdb& = db("fbInit", "SYSDBA", "masterkey", "XAdressen.fdb")
endif

WindowTitle "Tabellen löschen"
cls

Set("Errorlevel",-1)
sql$="DROP TABLE Bilder"
db("fbSQLExec",hdb&,sql$, 0)

sql$="DROP TABLE Test"
db("fbSQLExec",hdb&,sql$, 0)
Set("Errorlevel", 0)

db("fbDone", hdb&)

Messagebox("Fertig","Info",4160)
End
```

Wir erzeugen die Tabelle **TEXTBLOBS** mit dem Feld **BLOBTEXT**. Damit es zu keiner Fehlermeldung kommt, falls diese Tabelle bereits existiert, setzen wir den **Errorlevel** auf **-1** und danach wieder auf den normalen **Errorlevel 0** zurück:

```
Set("Errorlevel",-1)
db("fbSQLExec", hdb&, "CREATE TABLE TEXTBLOBS \
                       ( BLOBTEXT BLOB SUB_TYPE 1)",1)
Set("Errorlevel",0)
```

Spätestens zum Programmende ist die Datenbank zu schließen:

```
db("fbDone",hdb&)
```

### 5.4.1. MultiEdit auslesen und den Inhalt als Text-BLOB speichern

Wir lesen den Inhalt des MultiEdits in einer While-Schleife aus und schreiben ihn in eine temporäre Textdatei:

```
assign #1,MTempDat$
rewrite #1

Clear i%
While i% < GetCount(MEdit&)
   print #1,GetString$(MEDit&,i%)
   inc i%
EndWhile

close #1
```

Anschließend wird der Inhalt der temporären Textdatei in eine Bereichsvariable eingelesen und der Inhalt in einen String umgewandelt, den wir dann mit **db("fbPutBlob"** in das BLOB-Feld der Tabelle speichern:.

```
Dim BBereich#, FileSize(MTempDat$) + 2
ReadText BBereich#,MTempDat$
BString$ = STRING$(BBereich#,0)

sql$ = "INSERT INTO TEXTBLOBS (BLOBTEXT) VALUES (?)"
db("fbPutBlob",hdb&,sql$,BString$)

db("fbDone",hdb&)
dispose BBereich#
```

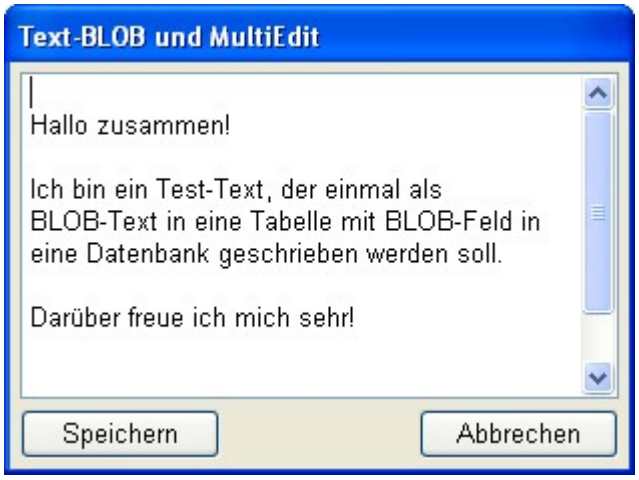
Und hier der gesamte Code der Prozedur **BLOB_SPEICHERN**:

```
Proc BLOB_SPEICHERN
   assign #1,MTempDat$
   rewrite #1

   Clear i%
   While i% < GetCount(MEdit&)
      print #1,GetString$(MEDit&,i%)
         inc i%
   EndWhile
   close #1

   Dim BBereich#, FileSize(MTempDat$) + 2
   ReadText BBereich#,MTempDat$
   BString$ = STRING$(BBereich#,0)

   sql$ = "INSERT INTO TEXTBLOBS (BLOBTEXT) VALUES (?)"
   db("fbPutBlob",hdb&,sql$,BString$)

   db("fbDone",hdb&)
   dispose BBereich#

   If FileExists(MTempDat$)
         assign #1,MTempDat$
         Erase #1
   EndIf
   e% = 1
```

## 5.4.2. Text-BLOB lesen und in einem MultiEdit anzeigen

Wir lesen den BLOB-Text mit **db("fbGetBlob"** in eine Stringvariable ein und weisen den Inhalt der Stringvariablen mit **SetText** dem MultiEdit zu.

```
Proc BLOB_LESEN
    sql$ = "SELECT BLOBTEXT FROM TEXTBLOBS"
    BString$ = db("fbGetBlob", hdb&, sql$)
    SetText MEdit&,BString$
EndProc
```




FB_023.PRF:

```
Declare e%,i%,x%,y%
Declare MEdit&,Save&,Exit&
declare sql$, hdb&
Declare MTempDat$
Declare BBereich#,BString$

MTempDat$ = $ProgDir + "MTempDat.txt"

Def GetSysColor(1) !"USER32","GetSysColor"

x% = 315
y% = 235

db("fbUseDLL","fbclient.dll")

ifnot fileexists("XAdressen.fdb")
  hdb& = db("fbCreate", "SYSDBA", "masterkey", "XAdressen.fdb")
     else
  hdb& = db("fbInit", "SYSDBA", "masterkey", "XAdressen.fdb")
endif

Set("Errorlevel",-1)
db("fbSQLExec", hdb&, "CREATE TABLE TEXTBLOBS \
                        (BLOBTEXT BLOB SUB_TYPE 1)",1)
Set("Errorlevel",0)
```

Fortsetzung FB_023.PRF:

```
WindowTitle "Text-BLOB und MultiEdit"
WindowStyle 48 + 512
Window (%MaxX - x%) / 2,(%MaxY - y%) / 2 - x%,y%
UseFont "Arial",16,0,0,0,0
SetDialogFont 1
CLS GetSyscolor(15)

MEdit& = Create("MultiEdit",%HWnd,"",5,5,Width(%HWnd) -10,(Height(%HWnd) - 40) * (-1))

BLOB_LESEN

Save& = Create("Button",%HWnd,"Speichern",5,Height(%HWnd) - 30,100,25)
Exit& = Create("Button",%HWnd,"Abbrechen",Width(%HWnd) - 105,Height(%HWnd) - 30,100,25)
SetFocus(MEDit&)

Clear e%
WhileNot e%
    Waitinput
    If Clicked(Exit&) or (%Key = 2)
        db("fbDone", hdb&)
        dispose BBereich#
            e% = 1
        ElseIf Clicked(Save&)
            BLOB_SPEICHERN
    EndIf
EndWhile

end

Proc BLOB_SPEICHERN
   assign #1,MTempDat$
   rewrite #1

   Clear i%
   While i% < GetCount(MEdit&)
      print #1,GetString$(MEDit&,i%)
         inc i%
   EndWhile
   close #1

   Dim BBereich#, FileSize(MTempDat$) + 2
   ReadText BBereich#,MTempDat$
   BString$ = STRING$(BBereich#,0)

   sql$ = "INSERT INTO TEXTBLOBS (BLOBTEXT) VALUES (?)"
   db("fbPutBlob",hdb&,sql$,BString$)

   db("fbDone",hdb&)
   dispose BBereich#

   If FileExists(MTempDat$)
        assign #1,MTempDat$
        Erase #1
   EndIf

   e% = 1
EndProc
```

Fortsetzung FB_023.PRF:

```
Proc BLOB_LESEN
    sql$ = "SELECT BLOBTEXT FROM TEXTBLOBS"
    BString$ = db("fbGetBlob", hdb&, sql$)
    SetText MEdit&,BString$
EndProc
```

# 6. Autoren und Büchertabelle

## 6.1. Das Erstellen, Füllen und Anzeigen der Tabelle

In unsere Adressen-Datenbank soll nun zusätzlich noch eine Büchertabelle integriert werden. Wie bereits eingangs erwähnt, können in Firebird alle Tabellen in einer einzigen Datenbankdatei untergebracht werden. Diese Büchertabelle integrieren wir wieder in die Datenbank **XAdressen.fdb**.

Die Vorgehensweise:

- ★ Prüfen, ob die Datenbank **XAdressen.fdb** existiert. Sollte dies nicht der Fall sein, dann wird sie erzeugt.
- ★ Prüfen, ob die Tabelle **BUECHER** existiert. Sollte dies nicht der Fall sein, dann wird sie erzeugt, und es werden die Datensätze eingefügt.

Prüfen, ob die Datenbank **XAdressen.fdb** existiert und ggf. erzeugen:

```
Declare fbdll&, hdb&

fbdll& = db("fbUseDLL","fbclient.dll")

ifnot fileexists("XAdressen.fdb")
  hdb& = db("fbCreate", "SYSDBA", "masterkey", "XAdressen.fdb")
     else
  hdb& = db("fbInit", "SYSDBA", "masterkey", "XAdressen.fdb")
endif

db("fbDone", hdb&)
FreeDll fbDLL&
```

Eine bereits vorhandene Tabelle namens **BUECHER** wird gelöscht.

```
Set("Errorlevel",-1)
sql$="DROP TABLE BUECHER"
db("fbSQLExec",hdb&,sql$, 1)
Set("Errorlevel",0)
```

Die Tabelle **BUECHER** wird erzeugt.

```
db("fbSQLExec", hdb&, "CREATE TABLE BUECHER \
                       ( AUTOR VARCHAR(20),\
                         TITEL VARCHAR(50),\
                         PREIS NUMERIC(2,2),\
                         DATUM DATE )",1)
```

Verwendete Datentypen:

**VARCHAR(n)**: Feld für Zeichenketten (Strings, Texte) mit maximaler Länge für n Zeichen. Bei kürzeren Inhalten wird entsprechend weniger Platz belegt. CHAR(n) da gegen belegt dagegen immer den Platz für n Zeichen, auch wenn der Inhalt des Feldes kürzer ist.

**NUMERIC(n1,n2):** Feld für Dezimalzahlen mit n1 Vorkommastellen und n2 Nachkommastellen. Der Dezimaltrenner ist als Punkt und nicht als Komma einzugeben.

**DATE:** Feld für Datumsangaben.

Die Tabelle wird mit einigen Beispiel-Datensätzen gefüllt.

```
db("fbSQLExec",hdb&,"INSERT INTO BUECHER (AUTOR,TITEL,PREIS,DATUM) \
   VALUES ('Dietmar Horn','Das Große XProfan-Lehrbuch',20.00,'01.03.2008')",1)

db("fbSQLExec",hdb&,"INSERT INTO BUECHER (AUTOR,TITEL,PREIS,DATUM) \
   VALUES ('Dietmar Horn','XProfan kinderleicht: Einführung',10.00,'01.01.2012')",1)

db("fbSQLExec",hdb&,"INSERT INTO BUECHER (AUTOR,TITEL,PREIS,DATUM) \
   VALUES ('Dietmar Horn','XProfan kinderleicht: dBase-Tabellen',10.00,'01.03.2012')",1)

db("fbSQLExec",hdb&,"INSERT INTO BUECHER (AUTOR,TITEL,PREIS,DATUM) \
   VALUES ('Dietmar Horn','XProfan kinderleicht: SQL-Datenbanken mit Firebird',10.00,'21.10.2012')",1)

db("fbSQLExec",hdb&,"INSERT INTO BUECHER (AUTOR,TITEL,PREIS,DATUM) \
   VALUES ('Dietmar Horn','Tabellenkalkulation kinderleicht mit Open-Office',9.70,'01.08.2012')",1)

db("fbSQLExec",hdb&,"INSERT INTO BUECHER (AUTOR,TITEL,PREIS,DATUM) \
   VALUES ('Roland G. Hülsmann','XProfan für alle',29.90,'01.12.2003')",1)

db("fbSQLExec",hdb&,"INSERT INTO BUECHER (AUTOR,TITEL,PREIS,DATUM) \
   VALUES ('Rolf Koch','ROC - der Fensterhelfer für XProfan',19.95,'01.02.2006')",1)

db("fbSQLExec",hdb&,"INSERT INTO BUECHER (AUTOR,TITEL,PREIS,DATUM) \
   VALUES ('Lothar Beloch','Tabellenkalkulation kinderleicht mit Open-Office',9.70,'01.08.2012')",1)

db("fbSQLExec",hdb&,"INSERT INTO BUECHER (AUTOR,TITEL,PREIS,DATUM) \
   VALUES ('Lothar Beloch','BWL mit anderen Worten',9.10,'01.01.2012')",1)
```

Der Inhalt der Tabelle BUECHER wird mit der SQL-Anweisung EXEC im vorgefertigten XProfan-Listbox-Dialog angezeigt.

```
Clearlist

db("fbSQLExec",hdb&,"SELECT * from BUECHER ORDER BY AUTOR", 1)

Listbox$("Tabelle BUECHER",2)
```

FB_024.PRF:

```
Var fbDLL& = db("fbUseDLL","fbclient.dll")
Declare i%
declare sql$,hdb&

ifnot fileexists("XAdressen.fdb")
        hdb& = db("fbCreate", "SYSDBA", "masterkey", "XAdressen.fdb")
     Else
        hdb& = db("fbInit", "SYSDBA", "masterkey", "XAdressen.fdb")
endif

Set("Errorlevel",-1)
sql$="DROP TABLE BUECHER"
db("fbSQLExec",hdb&,sql$, 1)
Set("Errorlevel",0)

db("fbSQLExec", hdb&, "CREATE TABLE BUECHER \
                            ( AUTOR VarCHAR(20),\
                                  TITEL VARCHAR(50),\
                                  PREIS NUMERIC(2,2),\
                                  DATUM DATE )",1)

db("fbSQLExec",hdb&,"INSERT INTO BUECHER (AUTOR,TITEL,PREIS,DATUM) \
   VALUES ('Dietmar Horn','Das Große XProfan-Lehrbuch',20.00,'01.03.2008')",1)

db("fbSQLExec",hdb&,"INSERT INTO BUECHER (AUTOR,TITEL,PREIS,DATUM) \
   VALUES ('Dietmar Horn','XProfan kinderleicht: Einführung',10.00,'01.01.2012')",1)

db("fbSQLExec",hdb&,"INSERT INTO BUECHER (AUTOR,TITEL,PREIS,DATUM) \
   VALUES ('Dietmar Horn','XProfan kinderleicht: dBase-Tabellen',10.00,'01.03.2012')",1)

db("fbSQLExec",hdb&,"INSERT INTO BUECHER (AUTOR,TITEL,PREIS,DATUM) \
   VALUES ('Dietmar Horn','XProfan kinderleicht: SQL-Datenbanken mit Firebird',10.00,'21.10.2012')",1)

db("fbSQLExec",hdb&,"INSERT INTO BUECHER (AUTOR,TITEL,PREIS,DATUM) \
   VALUES ('Dietmar Horn','Tabellenkalkulation kinderleicht mit Open-Office',9.70,'01.08.2012')",1)

db("fbSQLExec",hdb&,"INSERT INTO BUECHER (AUTOR,TITEL,PREIS,DATUM) \
   VALUES ('Roland G. Hülsmann','XProfan für alle',29.90,'01.12.2003')",1)

db("fbSQLExec",hdb&,"INSERT INTO BUECHER (AUTOR,TITEL,PREIS,DATUM) \
   VALUES ('Rolf Koch','ROC - der Fensterhelfer für XProfan',19.95,'01.02.2006')",1)

db("fbSQLExec",hdb&,"INSERT INTO BUECHER (AUTOR,TITEL,PREIS,DATUM) \
   VALUES ('Lothar Beloch','Tabellenkalkulation kinderleicht mit Open-Office',9.70,'01.08.2012')",1)

db("fbSQLExec",hdb&,"INSERT INTO BUECHER (AUTOR,TITEL,PREIS,DATUM) \
   VALUES ('Lothar Beloch','BWL mit anderen Worten',9.10,'01.01.2012')",1)

Clearlist

db("fbSQLExec",hdb&,"SELECT * from BUECHER ORDER BY AUTOR", 1)

Listbox$("Tabelle BUECHER",2)

db("fbDone",hdb&)

FreeDll fbDLL&
```

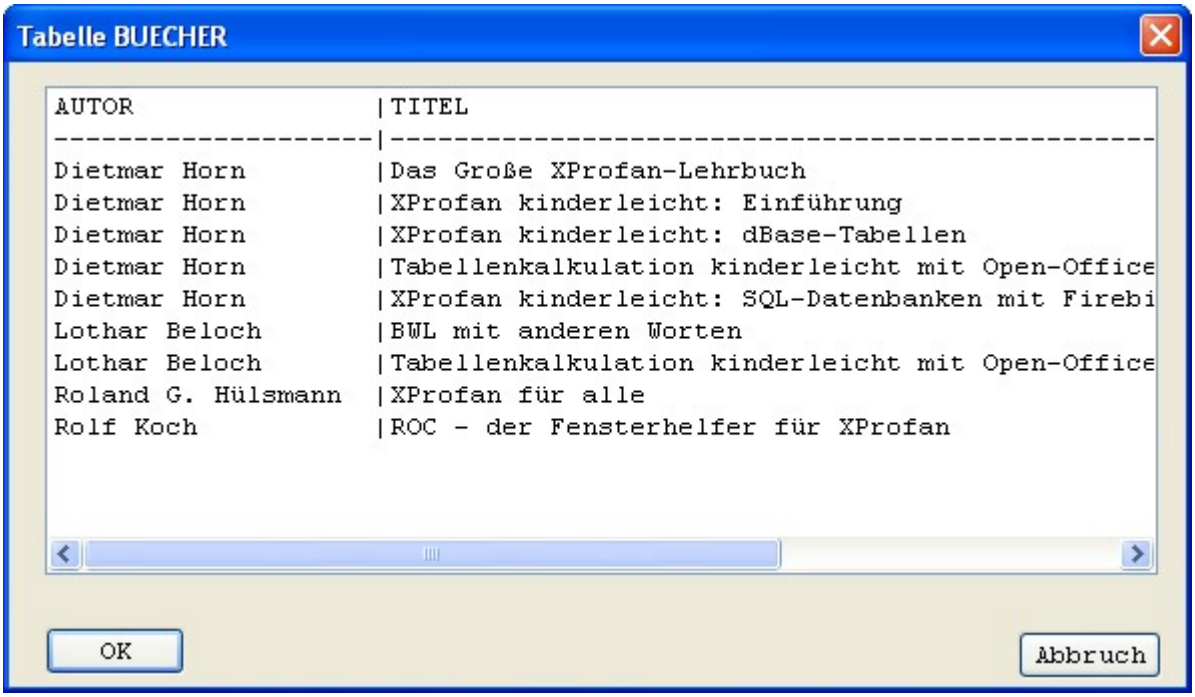


Tabelle BUECHER
AUTOR |TITEL
Dietmar Horn |Das Große XProfan-Lehrbuch
Dietmar Horn |XProfan kinderleicht: Einführung
Dietmar Horn |XProfan kinderleicht: dBase-Tabellen
Dietmar Horn |Tabellenkalkulation kinderleicht mit Open-Office
Dietmar Horn |XProfan kinderleicht: SQL-Datenbanken mit Firebi
Lothar Beloch |BWL mit anderen Worten
Lothar Beloch |Tabellenkalkulation kinderleicht mit Open-Office
Roland G. Hülsmann |XProfan für alle
Rolf Koch |ROC - der Fensterhelfer für XProfan
OK
Abbruch

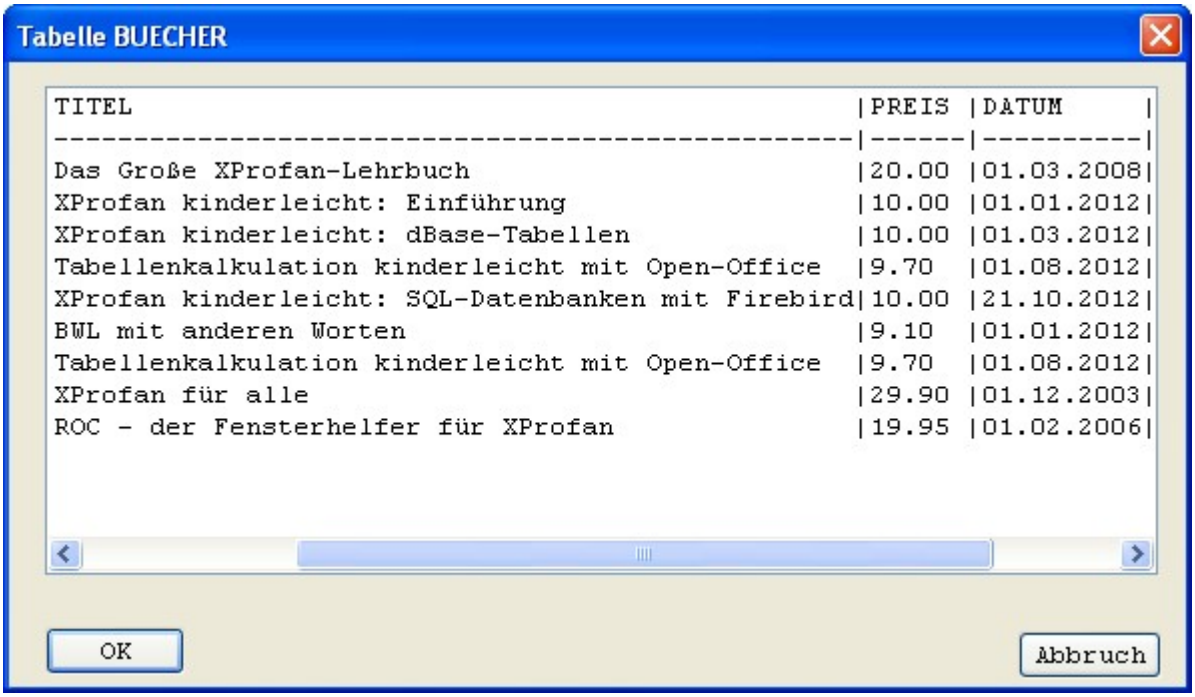


Tabelle BUECHER
TITEL |PREIS |DATUM
Das Große XProfan-Lehrbuch |20.00 |01.03.2008
XProfan kinderleicht: Einführung |10.00 |01.01.2012
XProfan kinderleicht: dBase-Tabellen |10.00 |01.03.2012
Tabellenkalkulation kinderleicht mit Open-Office |9.70 |01.08.2012
XProfan kinderleicht: SQL-Datenbanken mit Firebird |10.00 |21.10.2012
BWL mit anderen Worten |9.10 |01.01.2012
Tabellenkalkulation kinderleicht mit Open-Office |9.70 |01.08.2012
XProfan für alle |29.90 |01.12.2003
ROC - der Fensterhelfer für XProfan |19.95 |01.02.2006
OK
Abbruch

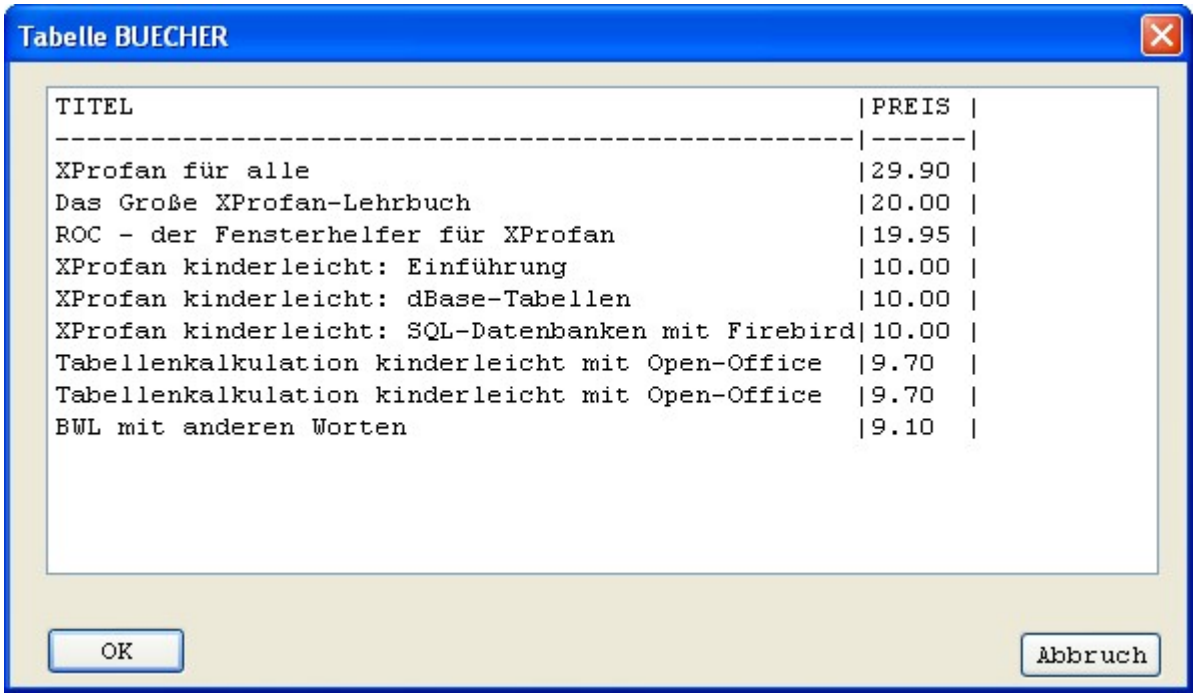
## 6.2. Das Filtern der Datensätze zum Anzeigen

In diesem Kapitel des Tutorials sollen einfache Möglichkeiten des Listens ausgewählter Datensätze bzw. Datenfelder am Beispiel unserer Tabelle **BUECHER** demonstriert werden.

**Nur Titel und Preise der Bücher nach ihrem Preis absteigend anzeigen:**

```
db("fbSQLExec",hdb&,"SELECT TITEL,PREIS FROM BUECHER ORDER BY PREIS DESC",1)
Listbox$("Tabelle BUECHER",2)
```

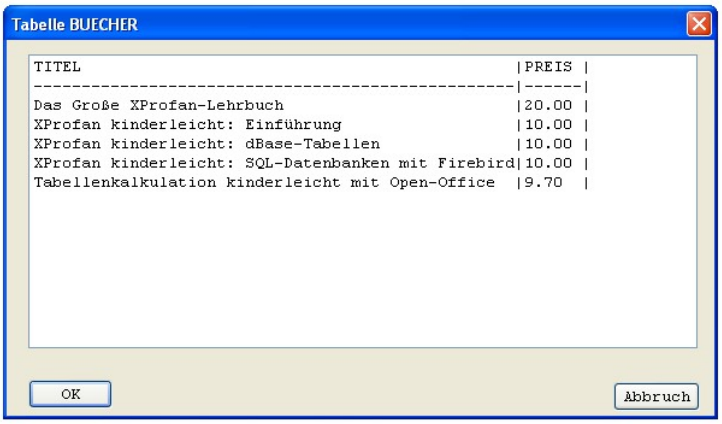
Siehe: FB_025.PRF




**Nur Titel und Preise der Bücher von Dietmar Horn anzeigen:**

```
db("fbSQLExec",hdb&,"SELECT TITEL,PREIS FROM BUECHER WHERE AUTOR = 'Dietmar Horn' ",1)
Listbox$("Tabelle BUECHER",2)
```

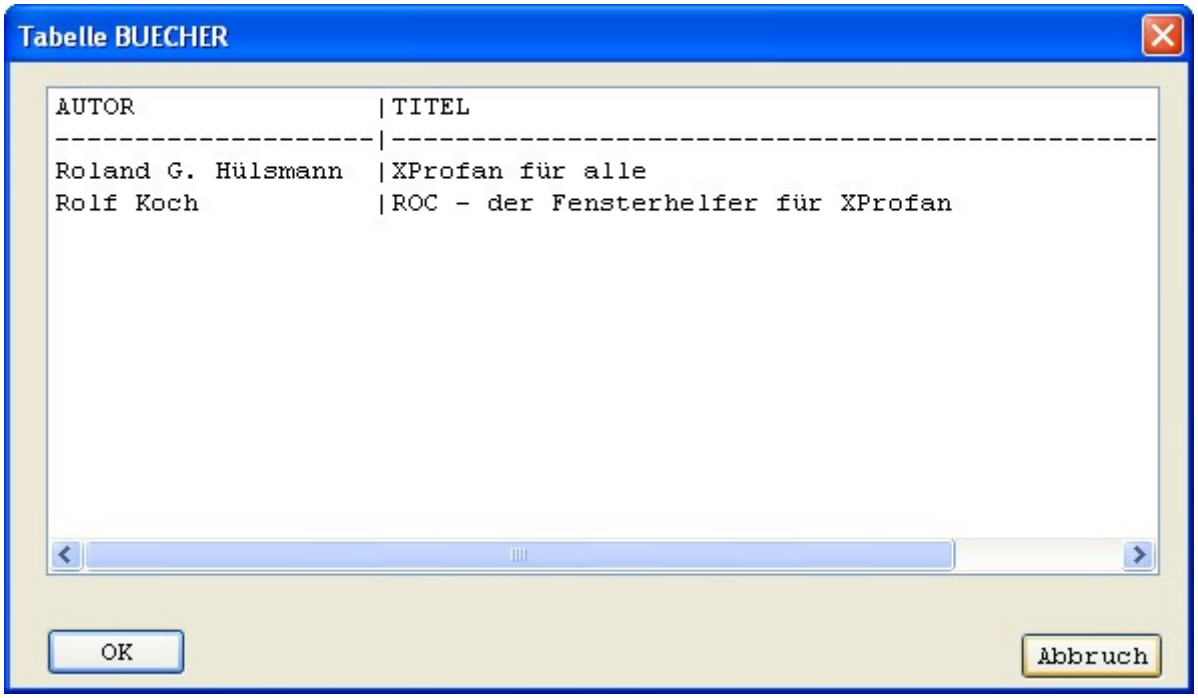
Siehe: FB_026.PRF



**Alle Bücher von Rolf Koch und Roland G. Hülsmann:**

```
db("fbSQLExec",hdb&,"SELECT AUTOR,TITEL FROM BUECHER WHERE AUTOR = 'Rolf Koch' OR AUTOR = 'Roland G. Hülsmann'",1)
Listbox$("Tabelle BUECHER",2)
```

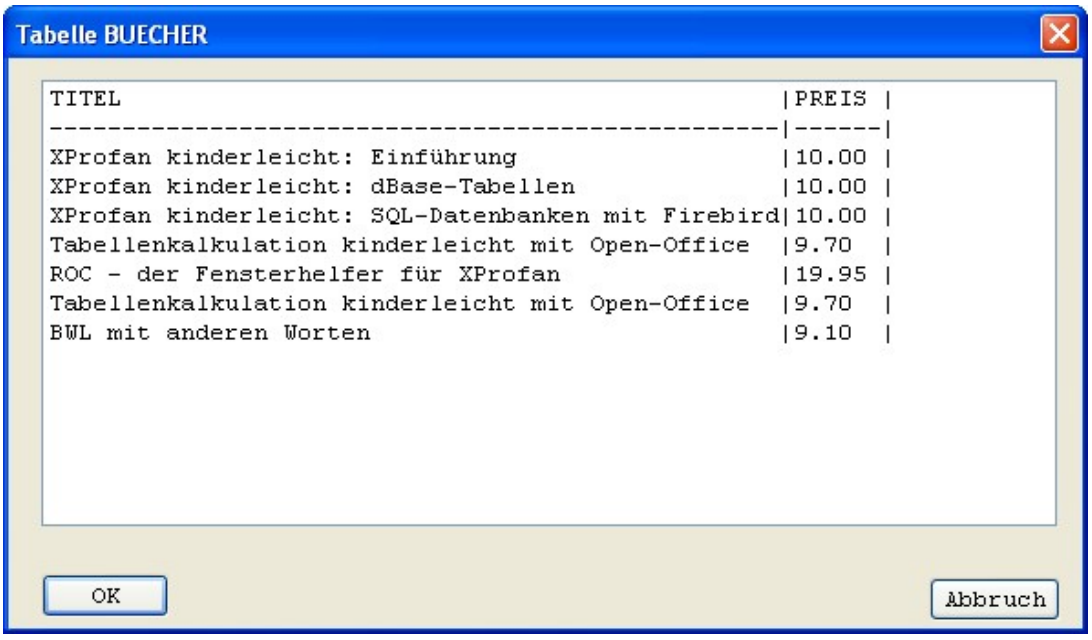
Siehe: FB_027.PRF




**Alle Bücher mit einem Preis unter 20.00 Euro:**

```
db("fbSQLExec",hdb&,"SELECT TITEL,PREIS FROM BUECHER WHERE PREIS < 20.00",1)
Listbox$("Tabelle BUECHER",2)
```

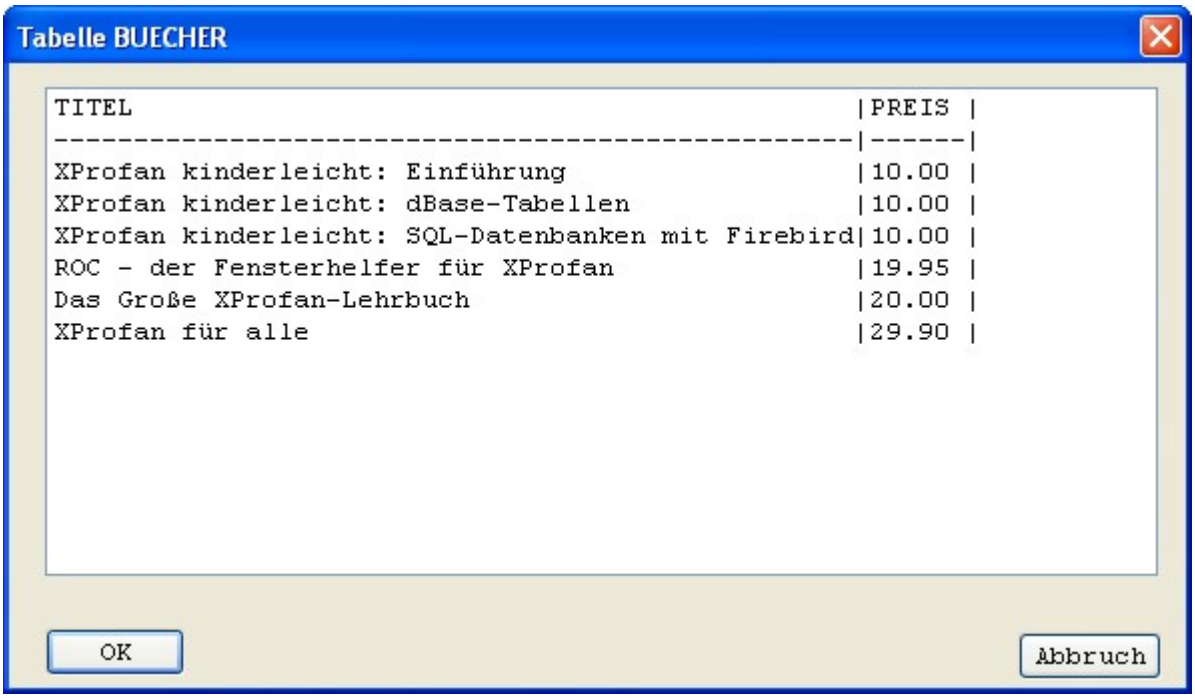
Siehe: FB_028.PRF



**Alle Bücher ab einem Preis von 10.00 Euro:**

```
db("fbSQLExec",hdb&,"SELECT TITEL,PREIS FROM BUECHER WHERE PREIS >= 10.00 ORDER BY PREIS",1)
Listbox$("Tabelle BUECHER",2)
```

Siehe: FB_029.PRF



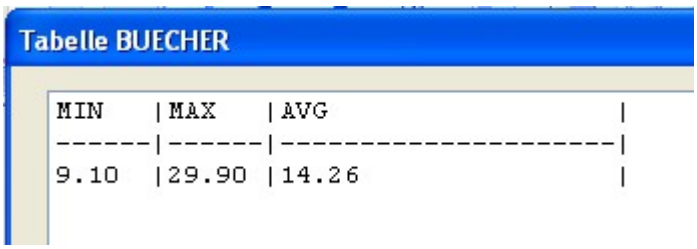

**Den kleinsten Preis, den größten Preis und den Durchschnittspreis aller Bücher ermitteln:**

Spaltenfunktionen bzw. Aggregatfunktionen gehen über alle Einträge einer Spalte (aggregate = gesamt).

**MIN(PREIS)**: Der kleinste Preis.
**MAX(PREIS)**: Der größte Preis.
**AVG(PREIS)**: Der Durchschnittspreis.

```
db("fbSQLExec",hdb&,"SELECT MIN(PREIS), MAX(PREIS), AVG(PREIS) FROM BUECHER",1)
Listbox$("Tabelle BUECHER",2)
```

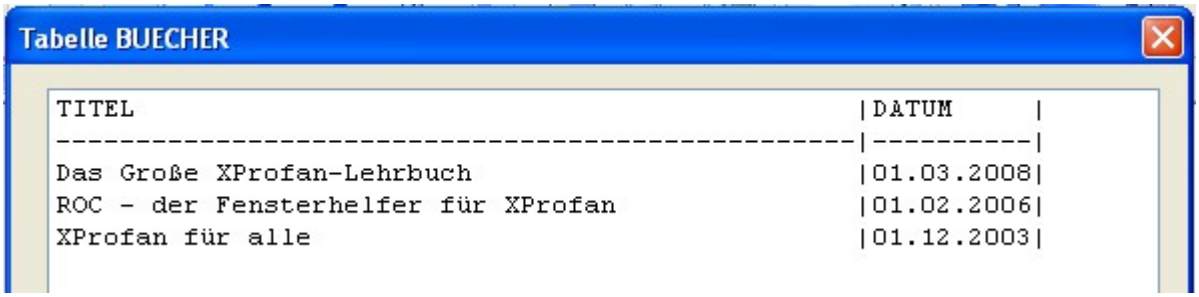
Siehe: FB_030.PRF



**Alle Bücher, die vor 2012 erschienen sind:**

```
db("fbSQLExec",hdb&,"SELECT TITEL,DATUM FROM BUECHER WHERE DATUM < '01.01,2012' ORDER BY DATUM DESC",1)
Listbox$("Tabelle BUECHER",2)
```

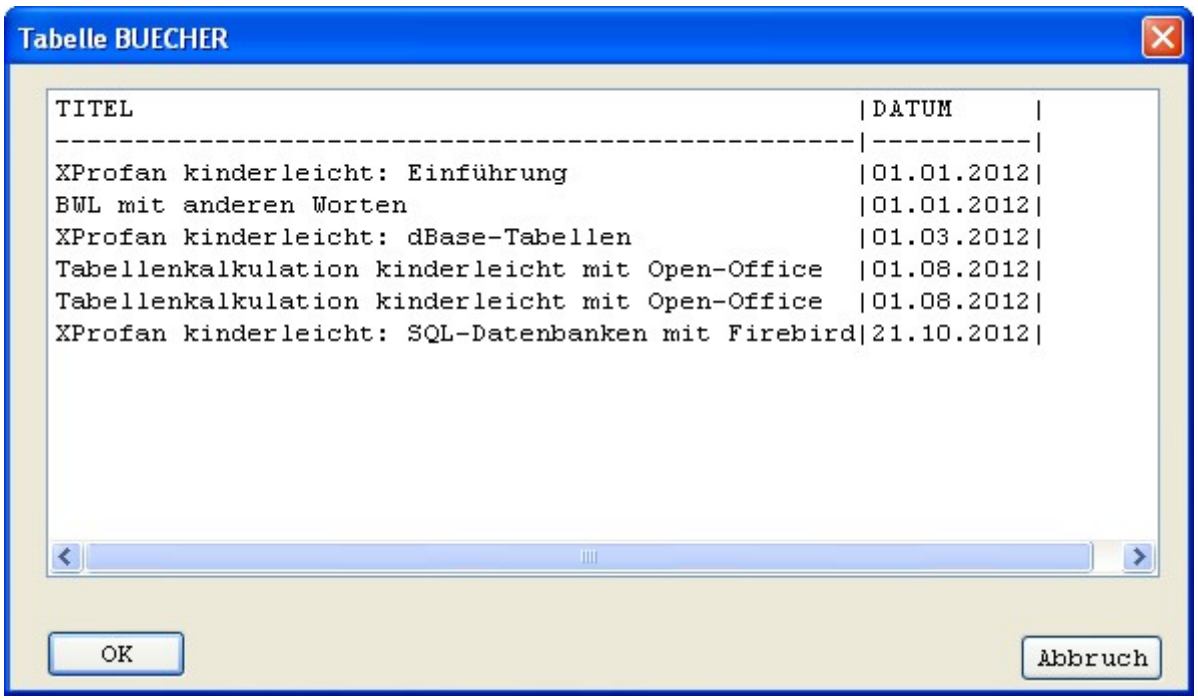
Siehe: FB_031.PRF




**Alle Bücher, die ab 2012 erschienen sind:**

```
db("fbSQLExec",hdb&,"SELECT TITEL,DATUM FROM BUECHER WHERE DATUM >= '01.01,2012' ORDER BY DATUM",1)
Listbox$("Tabelle BUECHER",2)
```

Siehe: FB_032.PRF

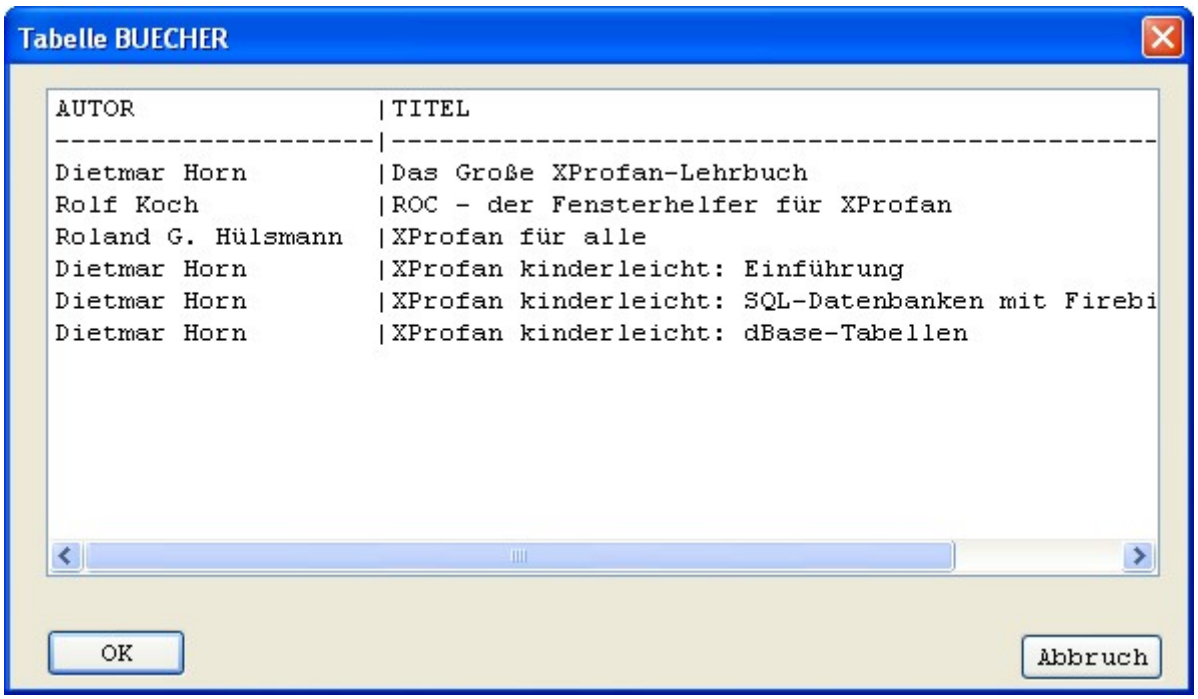


**Alle XProfan-Lehrbücher anzeigen:**

```
db("fbSQLExec",hdb&,"SELECT AUTOR, TITEL FROM BUECHER WHERE TITEL LIKE '%XProfan%' ORDER BY TITEL",1)
Listbox$("Tabelle BUECHER",2)
```

Das Prozentseichen im String **'%XProfan%'** steht als Platzhalter für beliebig viele Zeichen.

Siehe: FB_033.PRF



## 6.3. Das Anzeigen der Datensätze in einer XProfan-Gridbox

Zunächst erstellen wir uns mit „XProfan-Bordmitteln“ ein Fenster mit einer Gridbox (Listview) zur späteren Anzeige der in der Bücher-Tabelle enthaltenen Datensätze (siehe FB_034.PRF).


Das Kernstück zum Füllen der Gridbox mit den Datensätzen der Tabelle BUECHER ist dieses Unterprogramm (Prozedur):

```
Proc Read_FDB
    Clearlist
    db("fbSQLExec",hdb&,"SELECT * FROM BUECHER ORDER BY TITEL",1)

    i% = 2
  While i% < %GetCount + 1
    Autor$     = SubStr$(GetString$(0,i%),1,"|")
    Titel$     = SubStr$(GetString$(0,i%),2,"|")
    Preis$     = SubStr$(GetString$(0,i%),3,"|")
    Datum$     = SubStr$(GetString$(0,i%),4,"|")
    AddString(hGrid&,Autor$ + "|" + Titel$ + "|" + Preis$ + "|" + Datum$ + "|")
    inc i%
  EndWhile
EndProc
```

FB_034.PRF:

```
Declare e%,i%,x%,y%
Declare tmp$
Declare exit&,hGrid&
Declare Autor$,Titel$,Preis$,Datum$

x% = 510
y% = 185

DEF ___GSM(1) !"USER32","GetSystemMetrics"
DEF CaptionX(1) @%(1) + ___GSM(7) * 2
DEF CaptionY(1) @%(1) + CaptionX(0) + @___GSM(4)
DEF CenterX(1) %MaxX / 2 - CaptionX(@%(1) / 2)
DEF CenterY(1) %MaxY / 2 - CaptionY(@%(1) / 2)

Def GetSysColor(1) !"USER32","GetSysColor"

Var fbDLL& = db("fbUseDLL","fbclient.dll")
declare sql$,hdb&

ifnot fileexists("XAdressen.fdb")
        hdb& = db("fbCreate", "SYSDBA", "masterkey", "XAdressen.fdb")
     Else
        hdb& = db("fbInit", "SYSDBA", "masterkey", "XAdressen.fdb")
endif

Proc Fenster
    WindowStyle 48 + 8 + 512 + 2
    WindowTitle "Autoren- und Büchertabelle"
    Window 5 * %MaxX + CenterX(x%),CenterY(y%) - CaptionX(x%),CaptionY(y%)
    UseIcon "COMPUTER"
    UseFont "Arial",16,0,0,0,0
    SetDialogFont 1
    CLS GetSysColor(15)

    tmp$ = "Autor;0;115;Titel;0;275;Preis;2;45;Datum;0;70"
    hGrid& = Create("Gridbox",%HWnd,tmp$,0,0,0,Width(%HWnd),Height(%HWnd)-35)

    Read_FDB

    exit& = Create("Button",%HWnd,"Beenden",(Width(%HWnd) - 100) / 2,Height(%HWnd) -30,100,25)
EndProc

'#########################################################################

Fenster

SetWindowPos %HWnd = CenterX(x%),CenterY(y%) - CaptionX(x%),CaptionY(y%);0
SelectString(hGrid&,0,"")

Clear e%
WhileNot e%
                Waitinput
                If Clicked(exit&) or (%Key = 2)
                                        e% = 1
                EndIf
EndWhile

db("fbDone",hdb&)
FreeDll fbDLL&
End

Proc Read_FDB
    Clearlist
    db("fbSQLExec",hdb&,"SELECT * FROM BUECHER ORDER BY TITEL",1)

    i% = 2
  While i% < %GetCount + 1
    Autor$ = SubStr$(GetString$(0,i%),1,"|")
    Titel$ = SubStr$(GetString$(0,i%),2,"|")
    Preis$ = SubStr$(GetString$(0,i%),3,"|")
    Datum$ = SubStr$(GetString$(0,i%),4,"|")
    AddString(hGrid&,Autor$ + "|" + Titel$ + "|" + Preis$ + "|" + Datum$ + "|")
    inc i%
  EndWhile
EndProc
```

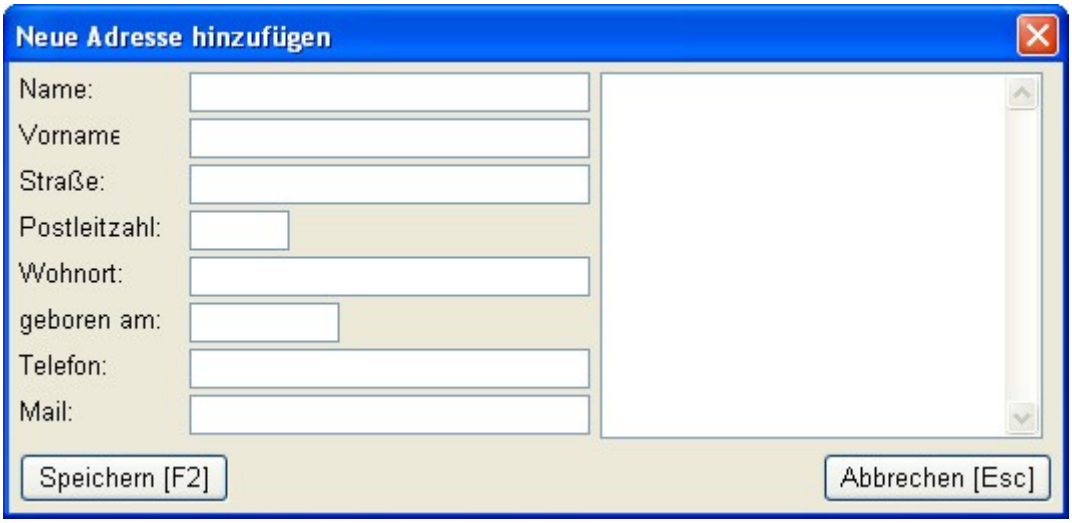
Zum Weitergeben eines solchen selbst erstellten Programmes müssen Sie mindestens folgende Dateien mitgeben, damit es auch Computern funktioniert, auf denen kein Firebird installiert ist.:

⇒ buecherliste.exe (bzw. FB_034.exe)
⇒ fbclient.dll (bzw. fbembed.dll)
⇒ ib_util.dll
⇒ icudt30.dll
⇒ icuuc30.dll

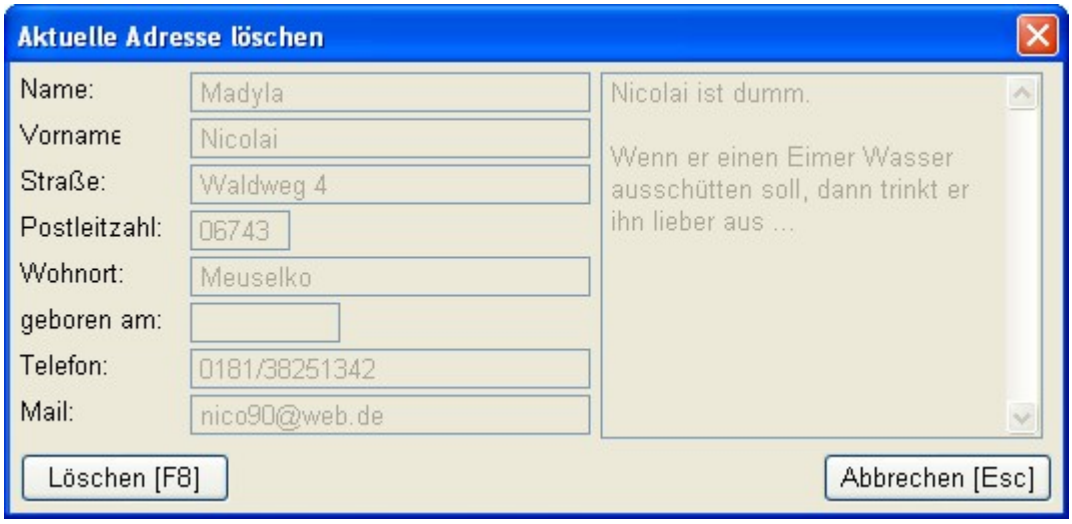
# 7. Eine kleine Adressenverwaltung mit XProfan und Firebird

Ein erstes kleines Projekt – eine minimale Adressenverwaltung, die Sie nach Ihren Bedürfnissen erweitern und anpassen können. Der Quellcode des Programmes ist der Vollversion dieses Lehrbuches beigelegt.

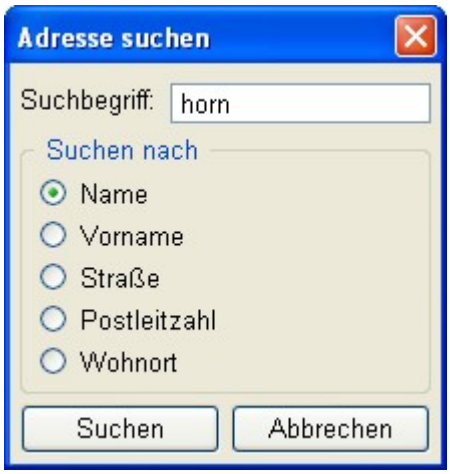

Neue Adressen können natürlich hinzugefügt werden.




Selbstverständlich kann man Adressen auch wieder aus der Tabelle entfernen.

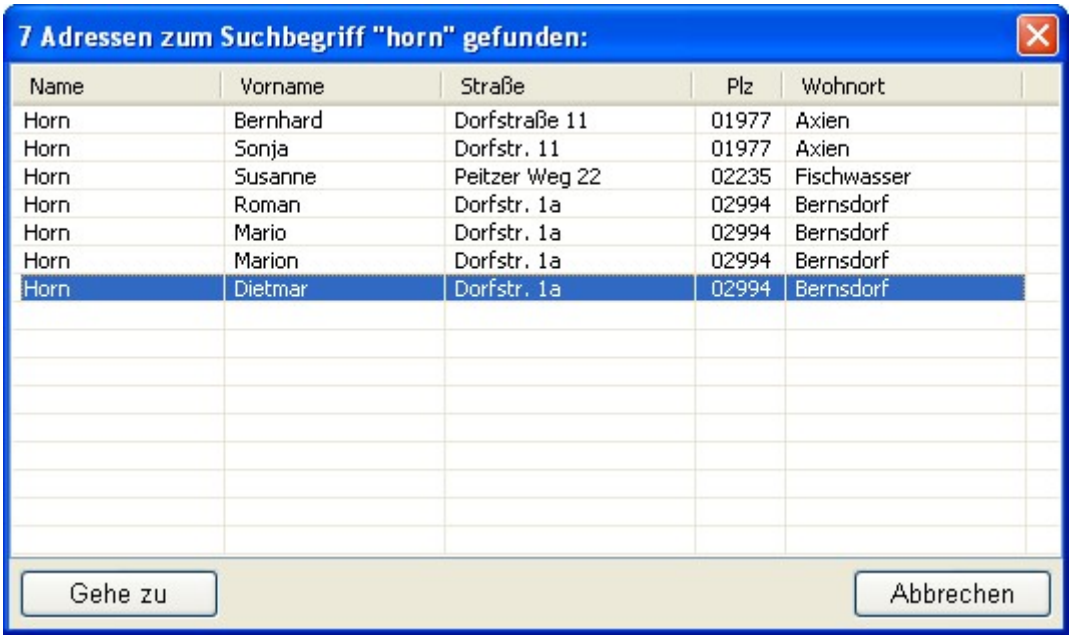



Und eine Suchen-Funktion ist in diesem kleinen Programm ebenfalls bereits enthalten.




Wird der Suchbegriff nur ein einziges Mal gefunden, dann wird im Hauptprogramm der gesamte Datensatz angezeigt.

Werden zum Suchbegriff mehrere Entsprechungen gefunden, dann werden diese dem Anwender zur Auswahl angezeigt.



# Anhang 1

## 1. Firebird-Funktionen in XProfan

Die FireBird-Funktionen wurden als Unterfunktionen in die Containerfunktion **db()** von XProfan aufgenommen.

### db("fbCreate"

**db("fbCreate",U,P,D)**

| | | |
|---|---|---|
| U: | String | - Username |
| P: | String | - Passwort |
| D: | String | - Name der Datenbankdatei |
| Ergebnis: | Integer | - Handle der Datenbank |

Eine neue Firebird-Datenbank wird erstellt, und die notwendigen Systemtabellen werden angelegt. In Firebird/Interbase befinden sich alle Tabellen (auch Systemtabellen) in einer physikalischen Datei. Die Datenbank wird auch gleich geöffnet. Username und Passwort werden bei späteren Zugriffen über **db("fbInit"** benötigt.

Ist eine Datei mit diesem Namen bereits im Verzeichnis vorhanden, erfolgt eine Fehlermeldung.

### db("fbDone"

**db("fbDone",H)**

| | | |
|---|---|---|
| H: | Integer | - Handle einer Firebird-Datenbank |

Die Firebird-Datenbank mit dem Handle **H** wird geschlossen.

### db("fbDrop"

**db("fbDrop",H)**

| | | |
|---|---|---|
| H: | Integer | - Handle einer Firebird-Datenbank |

Die Datenbank mit dem Handle **H** wird ohne weitere Rückfrage gelöscht. Alle Daten gehen dabei natürlich verloren. Die Datenbankdatei wird physikalisch gelöscht. Eine Datenbank gleichen Namens kann nun neu erzeugt werden.

## db("fbGetBlob"

### db("fbGetBlob",H,S)

| | | |
|---|---|---|
| H: | Integer | - Handle der Datenbank |
| S: | String | - SQL-Statement |
| Ergebnis: | String | - Blob (binär oder Text) |

Der SQL-Befehl ist ein **SELECT** und darf nur exakt ein Feld abfragen, und dies muss ein Blob-Feld sein. Anderenfalls erfolgt eine Fehlermeldung. Sollte der SELECT mehrere Zeilen als Ergebnis haben, wird nur das Blob der ersten Zeile ermittelt.

**Beispiel:**

```
Bild$ = db("fbGetBlob",db&,"SELECT bild FROM bilderliste WHERE bildnr = 1")
```

Das Blob wird immer als String zurückgegeben. Wenn es ein binäres Blob ist, kann es einfach einer Bereichsvariablen übergeben werden, etwa um es als Datei abzuspeichern:

```
Dim Bild#,len(Bild$)
Char Bild#,0 = Bild$
```

Das Abspeichern als Datei könnte mit **BlockWrite** erfolgen:

```
BlockWrite "Bild.jpg",Bild#,SizeOf(Bild#)
```

Reine Text-Blobs können auch per SELECT in einem normalen db("fbSQLExec" ermittelt werden.

## db("fbInit"

### db("fbInit",U,P,D)

| | | |
|---|---|---|
| U: | String | - Username |
| P: | String | - Passwort |
| D: | String | - Name der Datenbankdatei |
| Ergebnis: | Integer | - Handle der Firebird-Datenbank |

Eine bestehende Firebird- oder Interbase-Datenbank wird geöffnet. In Firebird/Interbase befinden sich alle Tabellen (auch Systemtabellen) in einer physikalischen Datei. Ist die Datenbank nicht vorhanden, erfolgt eine Fehlermeldung. Sind Username oder Passwort falsch, erfolgt beim Zugriff eine Fehlermeldung. Ist die Datenbank bereits geöffnet, erfolgt auch eine Fehlermeldung, da unter Firebird Embedded nur ein User gleichzeitig auf eine Datenbank zugreifen kann. Für die Client-Server-Version gilt die Einschränkung nicht.

Es können in einer Anwendung aber durchaus mehrere Datenbanken geöffnet werden.

## db("fbPutBlob"

### db("fbPutBlob",H,S1,[S2|B])

| | | |
|---|---|---|
| H: | Integer | - Handle der Datenbank |
| S: | String | - SQL-Statement |
| S1: | String: | - Text-Blob |
| S2: | Bereich: | - binäres Blob |

Mit dem SQL-Befehl wird ein Blob (Binary large object) in die Datenbank geschrieben. Der SQL-Befehl muss ein **INSERT** oder **UPDATE** sein, der exakt ein Blob-Feld enthält. Der Wert des Blob-Feldes wird dabei durch ein Fragezeichen ersetzt. Der Blob selbst folgt im nächsten Parameter als Zeichenkette oder Bereich. Auf diese Weise zählt die Länge des Blobs nicht zur Länge des SQL-Befehles, die in Firebird auf 64 kByte beschränkt ist.

**Beispiele:**

```
db("fbPutBlob",db&,"UPDATE bilderliste SET bild = ? WHERE bildnr = 1", Bild#)

db("fbPutBlob", db&, "INSERT INTO bilderliste (kommentar) VALUES (?)", Kommentar$)

db("fbPutBlob", db&, "INSERT INTO bilderliste VALUES ('Helloween Muffins','Annemarie Hülsmann',1,?)", Bild#)
```

Wie das dritte Beispiel zeigt, dürfen die Statements durchaus auch weitere Felder des Satzes behandeln, aber nur ein Blob-Feld. Das Blob kann ein Bereich oder ein String (Zeichenkette) sein. Da in XProfan Zeichenketten alle Zeichen enthalten dürfen, könnte auch eine Zeichenkette binäre Daten enthalten.

## db("fbSQLExec"

### db("fbSQLExec",H,S,N)

| | | |
|---|---|---|
| H: | Integer | - Handle der Datenbank |
| S: | String | - SQL-Statement |
| N: | Integer | - Modus |
| Ergebnis: | Integer | - Anzahl gefundener bzw. bearbeiteter Datensätze |

Ein SQL-Befehl wird ausgeführt. **N** hat die gleiche Bedeutung wie bei der XProfan-Funktion **db("SQLExec"**. Es werden alle Modi unterstützt.

**Hinweis:**

Firebird erlaubt die Verwendung unterschiedlicher SQL-Dialekte. Es wird derzeit und somit auch in XProfan Dialekt 3 verwendet. Datenbanken, die mit früheren Dialekten erstellt wurden, sollten problemlos bearbeitet werden können. Umgekehrt geht das nicht. Dialekt 1 unterstützt z.B. noch keine 64-Bit-Integerwerte, sodass dort die Länge von NUMERIC oder DECIMAL auf 9 Stellen beschränkt ist.

## db("fbUseDLL"

### db("fbUseDLL",S)

| | | |
|---|---|---|
| S: | String | - Firebird-DLL |
| Ergebnis: | Integer | - Ergebnis: Handle der DLL |

Die zu benutzende Firebird-DLL wird ausgewählt und initialisiert.

Im Normalfall ist das bei Firebird Embedded die "fbembed.dll". Eine Pfadangabe erfolgt nicht, da die Anwendung im gleichen Verzeichnis wie die DLLs von FireBird Embedded liegen soll.

Sollen auch Anwendungen für die Client-Server-Version laufen, ist die DLL in "fbclient.dll" umzubenennen.

# 2. SQL-Funktionen und Systemvariablen in XProfan

## db("SQLInit"

### db("SQLInit",S)

S: String - Initialisierungsstring
Ergebnis: Integer

Die Funktion verbindet das Programm mit dem Datenbankserver.

S ist der Initialisierungsstring, um den Datenbankserver zu öffnen und zu initialisieren. Ist auf Ihrem System ODBC nicht installiert, kommt die Fehlermeldung "ODBC.DLL nicht gefunden". Geben Sie als S einen Leerstring an, wird bei den meisten ODBC-Treibern eine Dialogbox geöffnet, mit der Sie den Datenbankserver wählen können.

## db("SQLExec"

### db("SQLExec",S,N)

| | | |
|---|---|---|
| S: | String: | - SQL-Befehl bzw. Zusatzbefehl mit "#" |
| N: | Integer | - Ergebnismodus |
| Ergebnis: | Integer | - Anzahl der bearbeiteten bzw. gefundenen Sätze |

Sendet einen SQL-Befehl an die Datenbank. S ist das SQL-Statement und N der Ergebnismodus:

- ★ **N = 0**: Das Ergebnis wird angezeigt. Jeder Datensatz ist eine Zeile.
- ★ **N = 1**: Das Ergebnis wird in die Listbox-Liste geschrieben und kann dann z.B. mit der LISTBOX-Funktion betrachtet werden. Jeder Datensatz ist ein Eintrag in der Liste und kann maximal 32767 Zeichen enthalten; maximal 262143 Datensätze passen in die Liste.
- ★ **N = 2**: Das Ergebnis wird in eine Datei geschrieben. Jeder Datensatz ist eine Zeile. Die Datei wird mit **Set("SQLFile"** festgelegt. Wird kein Dateiname bestimmt, wird "SQL.DAT" im aktuellen Verzeichnis genommen.
- ★ **N > 2**: Wenn das Handle einer Gridbox angegeben wird, werden deren Spaltenüberschriften mit den Feldnamen gefüllt, und die Gridbox wird mit dem Inhalt der Tabelle (etwaige vorherige Spalten und Inhalte der Gridbox werden gelöscht) gefüllt.

In **&SQLCount** wird zurückgeliefert, wie viele Datensätze bearbeitet wurden oder -1, wenn ein Fehler auftrat.

## Set("SQLColWidth"

### Set("SQLColWidth",N)

N: Integer

Mit dieser Funktion kann die maximale Spaltenbreite beim Auslesen von SQL-Daten eingestellt werden. Das ist dann besonders interessant, wenn man diese Daten in einer Tabelle oder Listbox darstellen möchte. N ist die Spaltenbreite in Zeichen. Hat N den Wert -1, gibt es keine maximale Spaltenbreite.

**Hinweis**: Unabhängig davon gibt es die Funktion **Set("SQLWidth"**, die lediglich die maximale Ausleselänge von LongVarChar (Memo-Felder) beeinflusst.

## Set("SQLDel"

### Set("SQLDel",S)

S: String

Legt fest, mit welchem Zeichen die einzelnen Felder im Ergebnis getrennt werden. Voreingestellt ist "|". Wenn S zwei Zeichen groß ist, legt das zweite Zeichen fest, mit welchem Zeichen die Feldinhalte begrenzt werden. Voreingestellt ist hier der Leerstring. Das zweite Zeichen wirkt nur, wenn das Ergebnis des SQL-Befehles mit **SQL("Exec“** in eine Datei oder die Listboxliste ausgegeben wird (Modus 1 oder 2).

Will man zum Beispiel erreichen, dass in der Ergebnisdatei die Feldinhalte in Anführungszeichen stehen und die Felder mit einem Komma getrennt werden, müsste **S** die Zeichen **,** und **"** enthalten:

Set("SQLDEL", ",\q")

## Set("SQLEmbedded"

### Set("SQLEmbedded",N)

N: Integer

Schaltet die XProfan-Variablen in SQL-Statements ein oder aus.

N = 1: (default) "embedded SQL" (XProfan-Variablen in SQL-Statetements) ist eingeschaltet.
N = 0: "embedded SQL" ist ausgeschaltet.

Das Ausschalten ist dann von Interesse, wenn die Syntax mit einer Syntax-Erweiterung des SQL-Treibers kollidiert.

### Set("SQLNull"

### Set("SQLNull",S)

S: String

Legt fest, dass **NULL** als S im Ergebnis dargestellt wird. Ein Datensatzfeld hat den Wert NULL, wenn ihm kein Wert zugewiesen wurde. **NULL** steht also für **nichts**.

### Set("SQLWidth"

### Set("SQLWidth",N)

N: Integer

MEMO-Felder (in dBase-Dateien) werden unter SQL als "LongVarChar" behandelt. Mit der Funktion **Set("SQLWidth"** kann die beim Auslesen genutzte Länge nahezu beliebig (max. 65535 Zeichen) eingestellt werden.

## $SQLError

### $SQLError

Der von der ODBC-Schnittstelle zurückgegebene Fehlertext.

Wenn man den Errormodus auf 0 gestellt hat, werden die Messageboxen mit der ausführlichen ODBC-Fehlermeldung unterdrückt, und nur das SQL-Ergebnis in **&SQLCount** wird auf -1 gesetzt.

Über **$SQLError** kann man dennoch den Fehlertext ermitteln.

## &SQLCount

### &SQLCount

Anzahl der vom letzten SQL-Befehl bearbeiteten Datensätze.

Hat **&SQLCount** den Wert -1, so ist ein Fehler aufgetreten.
Ein Fehlertext steht in **$SQLError**.

## 3. Ausgewählte SQL-Anweisungen

Hier werden einige oft verwendete SQL-Anweisungen aufgeführt.

### Datenbank erstellen

CREATE DATABASE

```
hdb& = db("fbCreate", "SYSDBA", "masterkey", "XAdressen.fdb")
```

### Datenbank löschen

DROP DATABASE

```
sql$="DROP TABLE BUECHER"
db("fbSQLExec",hdb&,sql$,1)
```

### Tabelle erstellen

CREATE TABLE

```
db("fbSQLExec", hdb&, "CREATE TABLE BUECHER \
                       ( AUTOR VarCHAR(20),\
                         TITEL VARCHAR(50),\
                         PREIS NUMERIC(2,2),\
                         DATUM DATE )",1)
```

### Tabelle löschen

DROP TABLE

```
sql$="DROP TABLE BUECHER"
```

**ALTER TABLE**
Tabellenstruktur ändern.

**CREAT INDEX**
Index anlegen.

**SELECT**
Tabelle abfragen.

**DELETE**
Daten löschen.

**INSERT**
Daten hinzufügen.

**UPDATE**
Daten ändern.

# 4. Datentypen für Datenfelder in Firebird

## SMALLINT

Größe: 2 Byte
Wertebereich: -32,768 bis +32,767

## INTEGER

Größe: 4 Byte
Wertebereich: -2,147,483,648 bis +2,147,483,647

## BIGINT

Größe: 64 Byte
Wertebereich: -9.223.372.036.854.775.808 bis 9.223.372.036.854.775.807

## FLOAT

Größe: 4 Byte
Wertebereich: -3.4 x 10^-38 bis 3.4 x 10^38
(7 signifikante Dezimalstellen)

## DOUBLE PRECISION

Größe: 8 Byte
Wertebereich: -1.7 x 10^-308 bis 1.7 x 10^308
(15 signifikante Dezimalstellen)

## DECIMAL

## NUMERIC

## DATE

## CHAR

Wertebereich: 1 bis 32767 Zeichen

## VARCHAR

Wertebereich: 1 bis 32767 Zeichen

## TIME

## TIMESTAMP

## BLOB

# Anhang 2

## 1. Copyright und Lizenzrechtliches



**Autor**: **Dietmar Horn**
Mail: dietmar_horn@gmx.de

## 2. Quellen-Nachweise

Zum Erarbeiten dieses Tutorials wurden folgende Quellen verwendet:

- ★ Eigene Scripte und jahrzehntelange Erfahrungen bei der Durchführung von Programmierkursen für Kinder, Jugendliche und Erwachsene.
- ★ Die Vollversion der Windows-Programmiersprache **XProfan** von **Roland G. Hülsmann,** http://xprofan.de.
- ★ Ausgewählte Hilfetexte der offiziellen Hilfedatei zu **XProfan** von **Roland G. Hülsmann**, http://xprofan.de.
- ★ Das große **XProfan-Lehrbuch** von **Dietmar Horn**, http://mmj.mxii.com/download/xlehrbuch.zip

## 3. Bezugsmöglichkeiten von XProfan

### XProfan X2

★ **Vollversion**
★ Compiler: ja
★ Preis: 59,90 €
★ Bestellung: http://www.jds-online-shop.de/contents/de/d8.html

### XProfan 11

★ **Freewareversion,** ohne Datenbankunterstützung (dBase, SQL)
★ Compiler: nein
★ Preis: 0,00 Euro
★ Download: http://www.xprofan.de/download/xprofanfree.exe

### OGL Basic

★ **Vollversion**, basierend auf XProfan 10, mit eingeschränktem Befehlsumfang
★ Compiler: ja
★ Preis: 0,00 Euro
★ Download: http://xprofan.de/download/oglbasic10.zip

### XProfan 9.1.

★ **Vollversion**
★ Compiler: ja
★ Preis: 0,00 Euro
★ Download: http://www.xprofan.de/download/xprofan91.zip

# 4. Publikationen

## 4.1. Das Große XProfan-Lehrbuch

**Autor: Dietmar Horn**
**Mail:** dietmar_horn@gmx.de


Dieses elfbändige Programmierhandbuch mit einem Umfang von ca. 3000 Seiten richtet sich an Anfänger, Umsteiger und Fortgeschrittene gleichermaßen.

Mehrere ausführliche Beiträge von Gast-Autoren zu unterschiedlichen Programmierthemen runden dieses umfangreiche Kompendium ab.

Ein komfortables Lehrbuchhelfer-Programm ist ebenso enthalten, wie Hunderte von Demo-Codes und zahlreiche Zusatzprogramme, sodass Sie sofort mit dem Programmieren beginnen können.

Mit XProfan können Sie kinderleicht eigene Windowsprogramme und Spiele erstellen, die unter Windows 95, 98, ME, 2000, XP, Vista sowie unter Windows 7 und Windows 8 lauffähig sind.

Kostenlose Testversion mit Kopier- und Druckschutz:
http://mmj.mxii.com/download/xlehrbuch.zip

Die Vollversion des umfangreichen Lehrbuches bzw. XProfan-Kompendiums (ohne Kopier- und ohne Druckeinschränkungen) als PDF-Dateien mit komfortablem Installationsprogramm und Lehrbuchhelfer ist erhältlich für nur 20 Euro.

Bestellung über: L.Beloch@gmx.de
oder über: http://xprofan.com/intl/de/lehrbuch-bestellen

### 4.2. „XProfan kinderleicht!“, Teil 1: „Einführung“

**Autor:** **Dietmar Horn**

**Mail:** dietmar_horn@gmx.de

**Freewareversion:** http://mmj.mxii.com/download/xlb.zip

**Vollversion (nur 10 Euro):** dietmar_horn@gmx.de
L.Beloch@gmx.de

**Die Zusendung der Vollversion erfolgt als Downloadlink per Mail (Zip-Archiv).**

## Dietmar Horn

# Programmieren kinderleicht
## mit
# XProfan

**für Windows 9.x, Windows 2000, XP, Vista, Windows 7/8**

## Teil 1: „Einführung“

### 4.3. „XProfan kinderleicht!“, Teil 2: „dBase-Tabellen“

**Autor:** **Dietmar Horn**

**Mail:** dietmar_horn@gmx.de

**Freewareversion:** http://mmj.mxii.com/download/xlbdbf.zip

**Vollversion (nur 10 Euro):** dietmar_horn@gmx.de
L.Beloch@gmx.de

**Die Zusendung der Vollversion erfolgt als Downloadlink per Mail (Zip-Archiv).**

## Dietmar Horn

# Programmieren kinderleicht
## mit
# XProfan

**für Windows 9.x, Windows 2000, XP, Vista, Windows 7/8**

## Teil 2: „dBase-Tabellen“

### 4.4. „XProfan kinderleicht!“, Teil 3: „SQL mit Firebird“

**Autor:** **Dietmar Horn**

**Mail:** dietmar_horn@gmx.de

**Freewareversion:** http://mmj.mxii.com/download/xlbdbfb.zip

**Vollversion (nur 10 Euro):** dietmar_horn@gmx.de
L.Beloch@gmx.de

**Die Zusendung der Vollversion erfolgt als Downloadlink per Mail (Zip-Archiv).**

## Dietmar Horn

# Programmieren kinderleicht
## mit
# XProfan

**für Windows 9.x, Windows 2000, XP, Vista, Windows 7/8**

## Teil 3: „SQL mit Firebird“

## 4.5. „Über 150 Excel-Tabellen für jedermann“

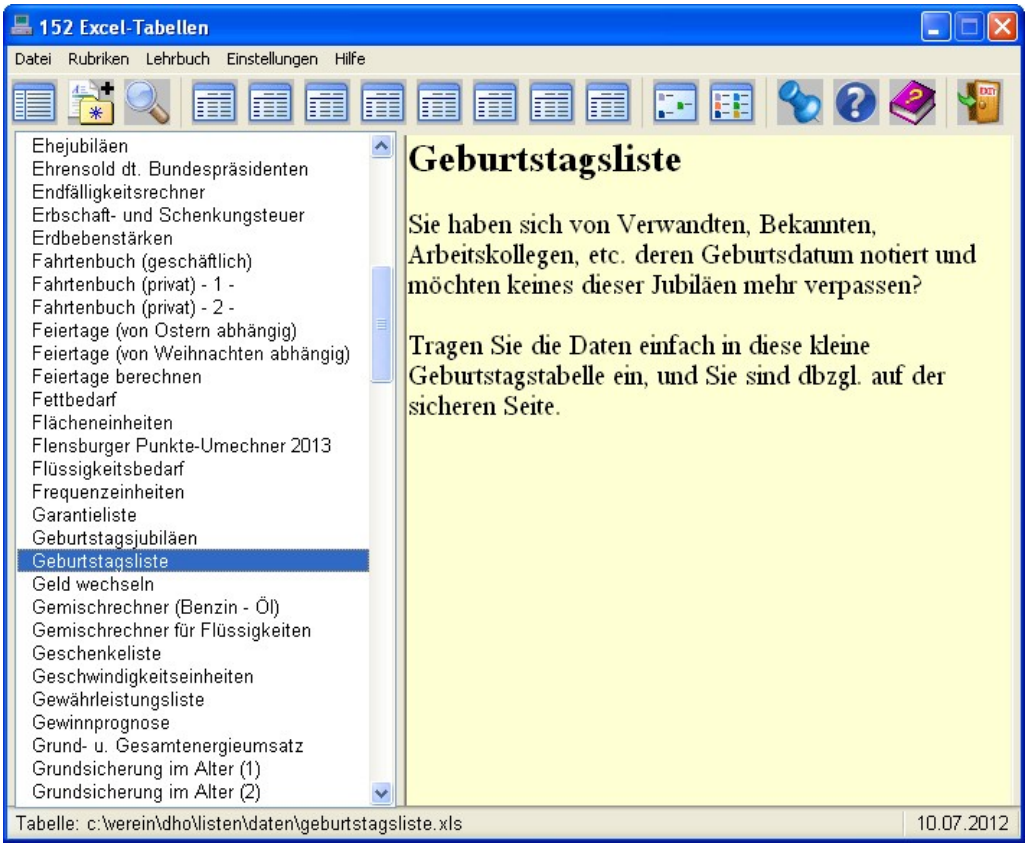
**Autor:** **Dietmar Horn**

**Mail:** dietmar_horn@gmx.de

**Freewareversion:** http://mmj.mxii.com/download/xls_listen.zip

**Vollversion (nur 10 Euro):** dietmar_horn@gmx.de
L.Beloch@gmx.de

**Die Zusendung der Vollversion erfolgt als Downloadlink per Mail (Zip-Archiv).**




**Geburtstagsliste**

| Vorname | Nachname | Geburtsdatum | Wochentag | Aktuelles Alter | Nächster Geburtstag | Wochentag |
|---|---|---|---|---|---|---|
| Dieterle | Bornowski | 18.12.1955 | Sonntag | 56 | 18.12.2012 | Dienstag |
| Lolle | Belowski | 26.05.1960 | Donnerstag | 52 | 26.05.2013 | Sonntag |
| Kunigunde | Kunowski | 25.10.1948 | Montag | 63 | 25.10.2012 | Donnerstag |
| Nicolai | Madjewski | 14.09.1990 | Freitag | 21 | 14.09.2012 | Freitag |
| Thomasius | Zielowski | 13.02.1990 | Dienstag | 22 | 13.02.2013 | Mittwoch |
| Maximilian | Winkjewitsch | 23.08.1995 | Mittwoch | 16 | 23.08.2012 | Donnerstag |
| Paolo | Glatzowski | 17.12.1998 | Donnerstag | 13 | 17.12.2012 | Montag |

### 4.6. „Tabellenkalkulation kinderleicht mit Open-Office“

**Autoren:** **Dietmar Horn & Lothar Beloch**

**ISBN-13:** **978-3-95488-028-7**

**1. Auflage 2012 Engelsdorfer Verlag**

**Paperback, Format: 21x15, 123 Seiten**

**Preis:** **9,70 €**

# Tabellenkalkulation kinderleicht

## „Das verstehe sogar ich ...“

# mit Open Office

Mit diesem Tutorial versuchen wir, Ihnen die Grundlagen des Arbeitens mit einem aktuellen Tabellenkalkulationsprogramm zu vermitteln. »Open Office« ist eine Office-Suite, die Sie kostenfrei aus dem Internet downloaden können. Die in diesem Office-Programm enthaltene Tabellenkalkulation Calc erfüllt inzwischen auch professionelle Ansprüche und Bedürfnisse. Sobald sich ein Ausgangs- oder Zwischenwert verändert, wird automatisch alles in Echtzeit aktualisiert. Die Ausführungen in diesem Lehrbuch gelten weitestgehend auch für andere Tabellenkalkulationen, wie z.B. Excel von Microsoft Office, Calc von Libre Office oder Planmaker von Softmaker Office. Sämtliche Beispiele und Bildschirmfotos dieses Lehrbuches wurden mit der Tabellenkalkulation Calc von Apache Open Office 3.4 erstellt.

Es besteht die Bezugsmöglichkeit des Buches direkt bei den Autoren für 10.00 € inklusive Porto und Versand. Die Kontodaten werden auf Anfrage per Mail mitgeteilt.

**Mail:** dietmar_horn@gmx.de
L.Beloch@gmx.de

## 4.7. „BWL mit anderen Worten“

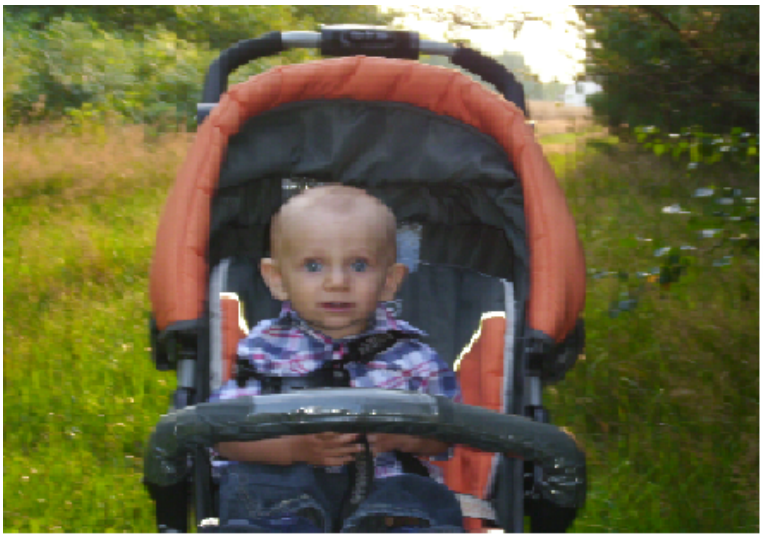
**Autor:** **Lothar Beloch**

**ISBN-13:** **978-3-86268-611-7**

**1. Auflage 2012 Engelsdorfer Verlag**

**Paperback, Format: 21x15, 114 Seiten**

**Preis:** **9,10 €**



**Das verstehe sogar ich ...**

Dies ist der Versuch, aus dem Bereich der Betriebswirtschaftslehre einige Begriffe mit anderen Worten zu erklären. Dieses Buch ist sowohl für Schüler der 9. und 10. Klasse als auch für Azubis und Umschüler gedacht.

Erläutert werden Themen, welche mir während meiner Dozententätigkeit bei den Schülern als Schwerpunkte aufgefallen sind.

Es besteht die Bezugsmöglichkeit des Buches direkt beim Autor für 8.00 € inklusive Porto und Versand. Die Kontodaten werden auf Anfrage per Mail mitgeteilt.

Mail: L.Beloch@gmx.de