

Dbase III-Dateien unter Profan verwenden.

Dieser Kurs soll für Anfänger in Profan die ersten Schritte zum Erstellen und verarbeiten von DBF-Dateien zeigen.

Alle benutzten Beispiele laufen sicher ab Profan Version 7.5.

Die Beispiele erheben keinen Anspruch auf absolute Richtigkeit und Vollständigkeit. Sie sind durch ausprobieren - in Verbindung mit der Profan - Bedienerhilfe entstanden.

Dieser hier vorliegende Kurs ist gegenüber der Vorgängerversion komplett überarbeitet worden und sollte jetzt wirklich auch für Anfänger, das heißt für Profaner die noch nie mit einer dBase-Datei unter Profan gearbeitet haben, geeignet sein.

Für Vorschläge zum Verbessern, Ergänzen oder Korrigieren dieses Kurses schreiben Sie bitte an:

Autor: Gerhard Putschalka, [Email](#)

Sie finden auch weitere Programmbeispiele auf meiner [HomePage](#)

Gerhard Putschalka, 1.6.2010

Voraussetzung für diesen Kurs:

Profan ist auf Ihrem PC bereits installiert (mindestens Version 7.5)

Sie sind mit der Profan - Entwicklungsumgebung z.B. Xprofed oder Profed32 vertraut.

Alle Beispiele können in dieser Entwicklungsumgebung ausgeführt (=interpretiert) werden.

Vorwort.

In Profan werden (neben anderen Dateien) dBase III Dateien, hier als DBF-Dateien bezeichnet, verwendet.

Eine DBF-Datei hat einen festen Satzaufbau, das heißt eine feste Satzlänge. Jeder Satz besteht aus genau definierten Feldern.

Für diese Dateiart wird auch der Begriff Tabelle benutzt. Die Zeilen einer Tabelle entsprechen den Sätzen, die Spalten der Zeile entsprechen den Feldern eines Satzes.

Wesentlich ist, dass bei einer DBF-Datei die einzelnen Sätze in beliebiger Reihenfolge geladen sein können und trotzdem ist aus einem Programm der Zugriff zu bestimmten Sätzen problemlos möglich ohne erst lange eine Suchroutine programmieren zu müssen.

Wie wird eine DBF Datei erstellt?

In der Hilfe zu Profan finden Sie unter Inhalt > Helfer (Assistenten) > Datenbankstruktur eine Anleitung zum Erstellen einer Strukturdatei.

Die Strukturdatei ist die Basis zum Erstellen einer DBF-Datei.

Um eine Strukturdatei zu erstellen gibt es 2 Möglichkeiten:

- Sie benützen in der Profan-Entwicklungsumgebung im Menü "Helfer" den Strukturhelfer
- oder Sie benützen einen normalen Texteditor (z.B. NotePad oder Word Pad), da eine Strukturdatei eine gewöhnliche Textdatei ist

In der Strukturdatei werden alle Felder mit ihren Attributen beschrieben. Der Aufbau ist grundsätzlich:

Name;Art;Länge;Dezimal

Erklärung:

Name	ist der Name des Feldes. Länge 1 bis 10 Zeichen in Großbuchstaben. Die erste Stelle darf nur Buchstaben, die Stellen 2 bis 10 dürfen Buchstaben und Ziffern enthalten.
Art	<p>C = Zeichenfeld. Das Feld kann alle Zeichen enthalten (Buchstaben, Ziffern, Sonderzeichen). Ein Zeichenfeld kann 1 bis 254 Stellen lang sein.</p> <p>N = numerisches Feld. Das Feld kann nur Ziffern und einen Punkt (als Dezimalzeichen) enthalten. Vor dem ersten Zeichen ist auch ein Minuszeichen erlaubt. Laut dBase III Spezifikationen darf ein numerisches Feld 1 bis 19 Stellen lang sein. In der Länge ist zu berücksichtigen, daß ein Minuszeichen und ein Dezimalpunkt ebenfalls je eine Stelle belegen.</p> <p>L = ein logisches Feld das nur 2 Zustände kennt: Wahr oder Nichtwahr (True oder False). Ein logisches Feld ist 1 Stelle lang</p> <p>D = ein Datumsfeld. Es kann nur Ziffern enthalten. Das Datum wird in der Form JJJMMTT gespeichert. Es ist 8 Stellen lang.</p> <p>M = ein Memofeld. Ein Memofeld ist immer 10 Stellen lang und enthält eine Referenz zu einem Satz in einer Memodatei. Der Satz in der Memodatei dient zur Aufnahme längerer Texte.</p>
Länge	beschreibt die Anzahl der Stellen des Feldes. Bestimmte Feldarten haben eine fix vorgegebene Länge. Bei einem numerischen Feld ist auch die Anzahl der Dezimalstellen in der Länge enthalten. Bei der Anzahl Dezimalstellen >0 ist auch noch eine Stelle im Feld für den Dezimalpunkt reserviert.
Dezimal	ist nur bei numerischer Feldart wichtig und beschreibt die Anzahl der Dezimalstellen des Feldes. Der Wert kann 0 bis 15 sein. Ist Dezimal größer als 0, muß von der Länge des Feldes die Anzahl der Dezimalstellen UND noch eine Stelle (für den Dezimalpunkt) abgezogen werden. Bei allen anderen Feldarten wird hier 0 eingesetzt.

Zwischen Name, Art, Länge und Dezimal muss als Trennzeichen ein Semikolon eingesetzt sein.

Beispiel einer Strukturdatei:

KUNDENNUMM;N;5;0
ZUNAME;C;35;0
VORNAME;C;40;0
GEBDAT;D;8;0
ANREDE;C;8;0
AKTIV;L;1;0
NOTIZEN;M;10;0

Mit dieser Struktur kann eine DBF-Datei erstellt werden. Jeder Satz enthält 6 Felder und die Satzlänge ist 97 Stellen.

Mit dem Profanbefehl

dbCreate S1 > S2

wird eine leere DBF Datei erstellt.

S1 ist der Dateiname (inkl. Pfad) der Strukturdatei
S2 ist der Dateiname (inkl. Pfad) der zu erstellenden DBF Datei (die Dateierweiterung (Extension) kann weggelassen werden es wird automatisch .DBF benutzt.

Beispiel: **dbCreate C:\DbFiles\Kunden.str > C:\DbFiles\Kunden**

Damit wird im Verzeichnis DbFiles auf dem Laufwerk C: die Datei Kunden.dbf erstellt.

Falls ein oder mehrere Memofelder in der Struktur definiert sind, wird automatisch auch eine Memodatei erstellt mit dem selben Namen wie die DBF-Datei, die Dateierweiterung ist jedoch .DBT.

Der Befehl **dbCreate** sowie alle weiteren Befehle kann sowohl beim Interpretieren eines Sourceprogramms als auch aus einem compilierten Programm ausgeführt werden.

Jetzt ist der Zeitpunkt für die erste Übung: [Lektion1](#)

Memofelder

In der ersten Übung haben wir eine DBF-Datei erstellt. Im Satzaufbau wurde auch ein Memofeld angegeben.

Felder in DBF-Dateien können höchstens 254 Stellen lang sein. Um längere Texte speichern zu können wurden Memofelder erfunden. Solche Memotexte sind nicht in der DBF-Datei sondern in einer eigenen Memodatei gespeichert. Eine Memodatei hat den selben Namen wie die DBF-Datei - mit dem Unterschied, daß die Dateinamenerweiterung nicht .dbf sondern .dbt lautet.

Warum speichern wir die Texte nicht gleich direkt in der DBF-Datei?

Dafür gibt es zwei Gründe.

Der erste Grund ist die Begrenzung der Textlänge auf 254 Stellen. Man könnte natürlich den Text

auf mehrere Felder splitten.

Der zweite Grund ist der, dass wahrscheinlich nicht jeder DBF-Satz einen Memotext hat. Das würde bedeuten, dass wir in jedem DBF-Satz die Felder bereitstellen müssen und vielleicht viele davon leer sind. In der Memodatei wird nur soviel Platz belegt, als Memotexte gegeben sind.

Die Verbindung des Textes in der Memodatei zum zugehörigen Satz in der DBF-Datei stellen die Memofelder her.

Zusammengefasst: Im DBF-Satz können 1 oder mehrere Memofelder enthalten sein. In den Memofeldern der DBF-Datei steht nur eine Referenznummer zum Text, der sich in der Memodatei befindet. Memofelder werden, zum Unterschied von normalen Feldern in der DBF-Datei, mit eigenen Befehlen bearbeitet.

Indexdateien.

Beim Ansehen der in der ersten Übung erstellten DBF-Datei (mit dem Programm **Anzeig1.prf**) werden Sie festgestellt haben, dass die Sätze in der Datei, auf die Kundennummer bezogen, unsortiert geladen wurden. Es ist wohl klar, dass diese Kundennummer einmalig ist und dass damit ein bestimmter Kunde eindeutig identifiziert werden kann, was über den Namen schon schwieriger wäre.

Daher wäre es naheliegend diese Datei nach der Kundennummer zu sortieren. Vielleicht wäre es auch erforderlich die Datei nach Postleitzahl oder auch nach Kundenname sortiert zu haben.

Das würde bedeuten dass die Datei jedesmal entsprechend umsortiert werden müsste. Oder wir haben die Datei mehrfach in verschiedener Sortierung vorliegen.

Das ist aber nicht sinnvoll und auch gefährlich. Denn:

Die selben Daten des Kunden dürfen nur in einer einzigen Datei und nicht in mehreren Dateien gespeichert sein. Jede Änderung in den Daten müsste immer gleichzeitig in allen diesen Dateien erfolgen.

Dieses Schema widerspricht einem ordentlichen Datenbankkonzept!

Es gibt in Profan keinen Befehl, mit dem eine DBF-Datei sortiert werden könnte.

Es gibt aber einen Ausweg in Form von Indexdateien.

Eine Indexdatei ist eine Hilfsdatei zu einer DBF-Datei und ermöglicht es, eine unsortierte DBF-Datei so zu verarbeiten als ob sie nach einem bestimmten Gesichtspunkt sortiert wäre.

Eine Indexdatei wird mit dem Profanbefehl **dbCreateIndex Feldname > Dateiname** erstellt. Für Feldname wird der Name des DBF-Feldes angegeben, nach dem die "Sortierung" erfolgen soll. Der Dateiname ist frei wählbar. Er bekommt automatisch die Dateiendung **.ndx**.

Beispiel: **dbCreateIndex Kundennumm > Kunde**

Es wird hier nach aufsteigender Sortierung der Kundennummer die Datei **Kunde.ndx** erstellt (Die DBF Datei bleibt dabei unverändert!).

Über einer DBF-Datei können mehrere Indexdateien (auch als Zugriffswege bezeichnet) aufgebaut werden. Damit besteht die Möglichkeit die DBF-Datei z.B. wahlweise (auch im selben Programm) nach Kundennummer oder nach Name oder nach Postleitzahl etc. zu verarbeiten.

Jetzt ist der Zeitpunkt für die nächste Übung: [Lektion2](#)

Wir haben in Lektion 1 eine kleine DBF-Datei mit dem Namen **Kunden.dbf** erstellt und haben jetzt in der Lektion 2 auch Memotexte zu einigen Kundensätzen hinzugefügt.

In der nächsten Lektion werden wir mit einem weiteren Programm die Sätze der Datei anzeigen und ändern.

Um die Inhalte der Datei **Kunden.dbf** (und **Kunden.dbt**) zu sehen und eventuell zu verändern:

gehen wir zur nächsten Übung: .
[Lektion3](#)

In der Übung 4 werden die Dateien **Kunden.dbf** und **Kunden.dbt** reorganisiert
Das ist notwendig, wenn Memotexte gelöscht wurden. Der Platz eines gelöschten Memotextes verbleibt als „Leiche“ in der Datei und der Platz kann auch nicht mehr verwendet werden.

Führen wir die [Lektion4](#) **aus.**

In der letzten Lektion wird in einem Programm gezeigt, wie eine Datei aufgebaut wird, nach verschiedenen Kriterien zu den Sätzen zugegriffen wird und was zu tun ist, wenn neue Sätze hinzugefügt werden damit auch zugehörige Indexdateien aktualisiert werden.

Springen wir zur letzten Übung: [Lektion5](#)

Damit sind wir am Ende des DBF-Kurses angelangt. Es sind nicht alle DBF-Profanfunktionen hier besprochen worden, doch sollte es mit dem Wissen aus diesem Kurs möglich sein die Schritte zum Erstellen und Bearbeiten von DBF-Dateien zu verstehen.

Mit dem Programm **Reorg.prf** können alle Dateien die während des Workshops erstellt wurden wieder entfernt werden, wenn Sie diesen Workshop wiederholen wollen.

So wünsche ich noch ein gutes Gelingen.

Gerhard Putschalka im Juni 2010

Lektion 1.

Machen wir gleich die erste Übung.

Lassen Sie diesen Hilfetext stehen und öffnen Sie über das Startmenü (Start in der Taskleiste links unten >Programme >Profan >Profanentwicklung - ist von der jeweiligen Windows Version abhängig) den Editor, den sie zum Bearbeiten von Profansources verwenden (Xprofed, oder alternativ ProfEdit32, Profan-Pad ...).

Sie haben damit die Möglichkeit Sources (d.h. die Übungsprogramme) zu laden und interpretiert auszuführen. Gleichzeitig können Sie zwischen Programm und dieser Hilfe hin- und herblättern.

In dieser ersten Übung erstellen wir eine DBF Datei und laden die Daten aus einer Textdatei.

Laden Sie das Programm **Progr1.prf**

Zu Beginn finden wir die Anweisungen:

```
OutFile$ = "Kunden.dbf"  
Assign #1,"Kunden.txt"  
dbCreate "Kunden.str" > OutFile$  
Reset #1  
@dbOpen(#2,OutFile$)  
@dbUse(#2)
```

Der Name der DBF-Datei kommt hier mehrmals vor (in **dbCreate** und in **dbOpen**). Daher wurde der Name der DBF-Datei in einer Variablen (OutFile\$) gespeichert und diese Variable bei den erwähnten Befehlen angegeben. Genausogut kann man aber auch den Dateinamen bei den Befehlen **dbCreate** und **dbOpen** direkt als Stringkonstante anführen.

Gleiches gilt für den Befehl **Assign**, wo der Dateiname auch in einer Stringvariablen angegeben werden kann.

Um mit einer Datei arbeiten zu können muss sie geöffnet werden. Je nach Dateart erfolgt dies mit unterschiedlichen Befehlen.

Die Textdatei wird zum Lesen mit **Reset #1** geöffnet, nachdem zuvor dem Dateikennzeichen **#1** mit **Assign** der Name der Datei zugeordnet wurde.

Die DBF-Datei wird mit **dbOpen** geöffnet wobei die Zuordnung des Dateikennzeichens (**#2**) gleich hier und nicht mit **Assign** erfolgt.

Während Textdateien gezielt zum Lesen oder zum Schreiben geöffnet werden bewirkt ein **dbOpen**, dass die DBF-Datei (und nur für diese ist **dbOpen** gültig) immer zum Lesen, Ändern oder Schreiben eines neuen Satzes geöffnet wird.

Im weiteren Programmablauf werden die Dateien nicht mehr mit dem Namen sondern nur mit dem Dateikennzeichen (hier **#1** und **#2**) angesprochen.

Nach dem **dbOpen** wird noch mit **dbUse** die DBF-Datei zur Verarbeitung aktiviert (wird nur eine DBF-Datei im Programm benutzt hat **dbUse** keine Bedeutung).

Nun wird in einer Schleife Satz für Satz von der Textdatei gelesen bis das Dateiende erreicht wird. Jeder Satz der Eingabedatei hat alle Daten eines Satzes für die DBF-Datei.

Für jeden eingelesenen Text-Satz wird jetzt mit `@dbAppendBlank()` ein zunächst leerer Satz in die DBF-Datei geschrieben.

Im Eingabesatz sind alle Satzfelder, durch Semikolons getrennt, aneinandergereiht. Das heißt, dass jedes Feld einen Substring darstellt. Deshalb muss jedes Satzfeld aus dem Eingabesatz einzeln herausgelöst werden und kann dann in das betreffende Ausgabefeld übernommen werden.

Um Daten in ein DBF-Feld zu schreiben wird die Funktion `@dbPut` (z.B. `@dbPut("KUNDENNUMM",Daten$)`) angewendet. Der erste Parameter ist der Feldname, entweder als Konstante wie hier gezeigt oder in einer Stringvariablen gespeichert. Der zweite Parameter enthält die Daten. Diese können ebenfalls Konstanten sein, oder in einer Variablen stehen. In diesem Programm wird der Inhalt als Teil (Substring) einer Variablen übernommen, was durch `@Substr$(Zeile$,1,";")` ausgedrückt wird. Es wird der 1. Substring aus Zeile\$ übernommen, wobei ";" das Trennzeichen darstellt.

In dieser Art werden alle DBF-Satzfelder, mit Ausnahme des Memofeldes, gefüllt.

Nachdem nun ein Satz fertiggestellt ist muß er noch in die Datei geschrieben werden. Genauer gesagt, es wird der mit `@dbAppendBlank()` zuvor geschriebene Satz geändert. Das geschieht mit `@dbPutRec(0)`. Weil Null als Satznummer im Parameter angegeben ist, wird hier der gerade aktuelle Satz verändert.

Zuletzt, vor dem Programmende, müssen die Dateien wieder geschlossen werden. Auch hier wird wieder zwischen DBF-Datei und Textdatei unterschieden.

Als Nachtrag steht hier nochmals der DBF_Satzaufbau, wie er in der Strukturdatei gespeichert ist:

```
KUNDENNUMM;C;4;0
NAME;C;30;0
STRASSE;C;30;0
STADT;C;15;0
PLZ;C;6;0
VORWAHL;C;6;0
TELEFON;C;10;0
DATUM;D;8;0
UMSATZ;N;10;2
LIQUID;L;1;0
NOTIZ;M;10;0
```

Beenden Sie das Programm `Progr1.prf` und laden Sie das Programm `Anzeig1.prf` in den Editor. Wenn Sie das Programm ausführen, können Sie den Inhalt der soeben erstellten DBF-Datei ansehen.

Beachten Sie, daß die Sätze nicht nach der Kundennummer sortiert sind. Wir kommen in der nächsten Übung darauf zurück.

[zurück.](#)

Lektion 2.

Nachdem wir nun erfolgreich in der ersten Übung eine DBF-Datei erstellt haben, fahren wir mit der 2. Übung fort.

In dieser Übung werden Memotexte zu bestimmten Kundensätzen zugeordnet.

In der ersten Übung wurde die DBF-Datei Kunden.dbf erstellt, dabei wurde auch im Satz ein Memofeld NOTIZEN erstellt, aber es wurden keine Memodaten gespeichert. Das soll nun in dieser Übung geschehen.

Es gibt die Datei MemoTxt.txt (schon wieder eine Textdatei) in der einige Texte gespeichert sind. Die Verbindung, das heißt welcher Text gehört zu welchem DBF-Satz, geschieht über die Kundennummer.

Im Ablauf dieser Übung mit dem Programm Progr2.prf wird die Datei MemoTxt.txt sequentiell gelesen und pro Satz wird der DBF-Satz mit der gleichen Kundennummer mit den Memodaten ergänzt.

Das ist ganz einfach, wenn die Sätze in der DBF-Datei sortiert geladen sind. Sie sind es aber nicht!

Auch in der Datei MemoTxt.txt sind die Sätze nicht nach der Kundennummer sortiert.

Wie finden wir nun den richtigen DBF-Satz?

Wir könnten für jeden Memotext-Satz, die DBF-Datei vom ersten Satz beginnend bis zum gefundenen Satz lesen und diesen dann ändern. Das ist die schlechteste Methode. Bei großen Dateien würde das viel zu lange dauern.

Es gibt in Profan die Funktion **@dbSeek** mit der in der Datei nach einem bestimmten Begriff in einem Satzfeld gesucht werden kann. In diesem Fall die Kundennummer. Das wäre zwar schneller, ist aber dafür auch nicht die geeignete Methode.

Oder, und das ist die Lösung, wir erstellen einen Index über die Kundennummer.

Einen Index zu erstellen heißt, es wird mit dem Profanbefehl **dbCreateIndex** eine Indexdatei erstellt.

Falls nicht bereits geschehen, öffnen Sie wieder über das Startmenü den Editor, den sie zum Bearbeiten von Profansources verwenden (ProfEdit, Profan-Pad, ProfanWriter ...). Die Programme **Progr1.prf** und **Anzeig1.prf** werden nicht mehr benötigt. Sollte also der Editor von der ersten Übung noch immer offen und diese Programme noch geladen sein dann schließen Sie diese jetzt und laden das **Progr2.prf**.

Im Programm werden die beiden Dateien, so wie im ersten Programm, geöffnet. Neu ist hier der Befehl **dbCreateIndex "KUNDENNUMM" > "NUMMER"**

Mit **dbUse** wurde zuvor die DBF-Datei Kunden.dbf aktiviert und deshalb wird die Indexdatei über diese aktive Datei erstellt (das ist wichtig, wenn mehrere DBF-Dateien im Programm bearbeitet werden). Und zwar wird die Indexdatei NUMMER.ndx über dem Begriff KUNDENNUMM erstellt.

Mit der Funktion `@DBIndex("NUMMER")` sagen wir dem Programm, daß alle Zugriffe zur Datei Kunden.dbf über die Indexdatei NUMMER.ndx erfolgen sollen.

Beim Lesen der MemoTxt-Datei kommen die Sätze, nach Kundennummer gesehen, unsortiert daher. Deshalb müssen wir bei jedem gelesenen Satz mit einer geeigneten Funktion gezielt den richtigen DBF-Satz suchen (und finden). Die Funktion `@dbGo()` ist hier nicht geeignet (sie liest sequentiell – in diesem Fall nach der Indexdatei), es gibt dafür die Funktion `@dbFind`.

Der Suchbegriff (auch Schlüssel oder Key genannt) wird in einer Variablen an die Funktion übergeben. Die Funktion stellt dann, so wie bei `@dbGo`, den Satz für das Programm bereit. Natürlich nur, wenn der gesuchte Satz existiert.

Als Ergebnis der Funktion `@dbFind` erhalten wir die Nummer des Satzes in der DBF-Datei. Diese Nummer (nicht zu verwechseln mit der Kundennummer!) brauchen wir dann, wenn der geänderte Satz wieder in die Datei geschrieben wird. Sollte die Nummer Null sein, wurde der Satz nicht gefunden.

Wir dürfen daher eine Änderung nur durchführen, wenn die Nummer nicht Null ist.

Die Bearbeitung von Memofeldern erfolgt unter zu Hilfenahme der Listboxliste.

Wird ein Memofeld mit der Funktion `@dbGetMemo` eingelesen, wird der Text (aus der Memodatei!) automatisch in die Listboxliste übernommen. Von dort kann der Text mit verschiedenen Möglichkeiten angesehen oder bearbeitet werden und wird danach mit der Funktion `@dbPutMemo` wieder aus der Listboxliste in die Memodatei zurückgeschrieben.

Memofelder (und nur diese Feldart) können nur mit den Funktionen `@dbGetMemo` und `@dbPutMemo` gelesen bzw. geschrieben werden.

Im **Progr2.prf** werden die Memotexte nur aus der Textdatei in die DBF-Datei geschrieben. Lesen und schreiben von Memotexten kommt in einem späteren Programm.

Wesentlich ist, daß wir in diesem Programm die Texte aus einer Textdatei einlesen, aus den Stellen 1 - 4 die Kundennummer und aus den Stellen 5 - Satzende den Memotext in die Variablen Nummer\$ und MText\$ stellen. In Nummer\$ ist jetzt der Suchbegriff (also die Kundennummer) um den DBF-Satz zu finden. MText\$ stellen wir in die Listboxliste von der sie dann mit `dbPutMemo` in die Memodatei geschrieben wird. Profan vermerkt dabei automatisch im Memofeld NOTIZEN die Adresse des Textes in der Memodatei.

Führen Sie jetzt das Programm **Progr2.prf** aus.

Ob die Memotexte jetzt übernommen wurden, werden wir in einer späteren Übung feststellen. Beenden Sie wieder das Programm **Progr2.prf** und laden Sie wieder das Programm **Anzeig1.prf** und speichern es sofort als **Anzeig2.prf**.

Führen Sie **Anzeig2.prf** aus.

Die Datei Kunden erscheint noch immer unsortiert. Das ist auch richtig, weil die Datei durch das Erstellen der Indexdatei ja nicht verändert wurde.

Bauen Sie nun in **Anzeig2.prf** die Funktion `@dbIndex` (so wie in **Progr2.prf** gezeigt) hinter der

Funktion `@dbUse` ein. `@dbCreateIndex` ist jetzt nicht erforderlich, weil die Indexdatei ja schon besteht. Führen Sie das Programm `Anzeig2.prf` aus. Wenn Sie die Funktion richtig eingebaut haben müßte die Datei jetzt nach Kundennummer sortiert angezeigt werden. Wieso funktioniert das Ganze doch mit `@dbGo` anstatt mit `@dbFind`?

`@dbGo` wird normalerweise für ein fortlaufendes Lesen, oder gezielt zum Lesen eines bestimmten Satzes mit Hilfe der Satznummer (nicht die Kundennummer!) verwendet. In `Progr2.prf` wollten wir jeweils einen bestimmten Satz mit Hilfe des Suchbegriffes (= die Kundennummer) lesen.

Im Programm `Anzeig2.prf` wird die Datei fortlaufend vom 1. bis zum letzten Satz gelesen. Durch den Einbau der Funktion `@dbFind` wird hier der Index über die Kundennummer berücksichtigt, obwohl die DBF-Datei nach wie vor unsortiert besteht.

`@dbFind` funktioniert nur in Verbindung mit einer Indexdatei.

[Zurück](#)

Lektion 3.

In dieser Übung kann die Datei Kunden.dbf angesehen und auch geändert werden.

Falls nicht bereits geschehen beenden Sie alle bisherige Programme und laden Sie das Programm **Progr3.prf**.

Dieses Programm ist schon umfangreicher.

Nach dem Start des Programms wird ein Dialog angezeigt.

Im oberen Teil kann die Kundennummer des zu suchenden Satzes eingegeben werden. Mit dem Button **suchen** wird, wenn die Nummer stimmt, der Inhalt des Kundensatzes angezeigt.

Da die Kundennummer oft nicht gleich bekannt ist, gibt es auch die Möglichkeit über den Kundennamen zu suchen. Dabei muss nicht der ganze Name eingegeben werden, es genügt auch, nur den ersten/die ersten Buchstaben einzugeben. Um das Ganze noch mehr einzuschränken kann zusätzlich auch die Postleitzahl eingegeben werden (ebenso auch nur ein Teil der Nummer möglich).

Wichtig ist jedoch, beide Suchbegriffe müssen mit der ersten Stelle beginnen. Um z.B. alle "Berger" zu suchen muß B oder Be oder Ber oder Berg (u.s.w.) eingegeben werden. Die Eingabe von "erger" ist nicht gültig!

Wurde also ein Name eingegeben wird, wenn es zumindest einen Satz zum Suchbegriff gibt, eine Listbox angezeigt und hier kann dann der entsprechende Kundensatz ausgewählt werden.

Bei der darauf folgenden Anzeige der Kundendaten ist das Feld mit der Kundennummer grau. Das heißt, dass dieses Feld nicht geändert werden kann.

Es ist auch nicht sinnvoll eine Kundennummer zu ändern!

Alle anderen Daten können geändert werden. Gibt es zu diesem Kundensatz auch einen Memotext, so ist der entspr. Button mit **Notizen ändern** beschriftet. Gibt es keinen Memotext so lautet die Beschriftung **Notizen eingeben**.

Werden Daten im angezeigten Satz geändert und/oder der Memotext geändert oder erstellt, muss der angezeigte Satz mit dem Button **Satz ändern** in die Datei zurückgeschrieben werden - wenn der Satz tatsächlich geändert werden soll.

Anmerkung.

In der vorliegenden Form ist es ohne Bedeutung ob der Suchbegriff in Groß- oder Kleinschreibung eingegeben wird. In der Funktion **@dbSeek** kann mit dem 3. Parameter bestimmt werden ob der (Teil-) Suchbegriff irgendwo, oder ab der ersten Stelle im gesuchten Feld stehen muss. Die Suche mit **@dbSeek** ist hier deshalb angebracht, weil hier in jedem Feld (in diesem Beispiel Name und Postleitzahl) gesucht werden kann – unabhängig davon ob es eine Indexdatei (.ndx) über dieses Feld gibt.

Siehe Profanhilfe.

Es gibt auch die Funktion **@dbFind** die, wenn nach der Kundennummer gesucht wird (und wenn

über dieses Feld eine Indexdatei existiert!), besser geeignet ist.

. Sollte das Programm Progr3.prf mit einer älteren Profanversion ausgeführt werden so ist in der Prozedur Suche_mit_Name die Funktion @dbSeek durch die Funktion @dbFind zu ersetzen (siehe Kommentar im Programm. In diesem Fall muß für den Suchbegriff die Groß- und Kleinschreibung beachtet werden.

Satz löschen.

In DBF-Dateien werden Sätze bei der Bearbeitung nur zum Löschen markiert. Bei allen Sätzen die als gelöscht markiert sind, werden alle Eingabefelder grau (gesperrt) angezeigt und der entsprechende Button wechselt den Text von Satz löschen auf Satz aktivieren. Damit besteht die Möglichkeit einen als gelöscht markierten Satz wiederzubeleben.

Als "gelöscht" markierte Sätze bleiben solange in der Datei bestehen bis sie mit der Funktion dbPack() aus der Datei entfernt werden. Erfolgt dies nicht, existieren diese Sätze auch nach dem Ende des Programms weiter in der Datei. Behalten jedoch das „gelöscht“ Kennzeichen.

Ob ein Satz als gelöscht markiert ist kann mit der Profanvariablen %dbDeleted festgestellt werden.

Wurde im Programm Progr3.prf einmal der Button Satz löschen gedrückt, so wird am Programmende gefragt ob gelöschte Sätze aus der Datei entfernt werden sollen.

Memodaten.

Die Art wie bei dBase III Memodaten verwaltet werden ist nicht sehr komfortabel. Die Memodaten bzw. Memotexte werden, wie schon erwähnt, in einer eigenen Datei gespeichert. In der DBF-Datei wird im Memofeld nur eine Referenznummer zum Text gespeichert.

Das Problem besteht darin, daß ein Textblock der einmal angelegt wurde, nicht mehr aus der Memodatei entfernt werden kann. Man kann zwar den Text entfernen, doch bleibt das leere Textfeld in der Memodatei bestehen und der Platz kann nicht für einen anderen Kundensatz genutzt werden. Es kann jedoch in das leere Textfeld - für den selben Kundensatz - später ein anderer Text eingegeben werden.

Aus diesem Grund erscheint nach dem Leeren eines Memotextes bei der Anzeige des Kundensatzes auf dem Button weiterhin der Text "Notizen ändern".

Die eingefügten Kommentare im Progr3.prf sollten genügen um die Funktion des Programmes zu verstehen.

[Zurück](#)

Lektion 4.

Wie schon in der vorigen Übung erwähnt bleiben geleerte Memotexte weiterhin als "Leichen" in der Memodatei bestehen.

Jeder Textblock belegt 1024 Stellen!.

Wurde ein DBF-Satz gelöscht so bleibt ein allfällig erstellter Memotext ebenfalls bestehen!

Die einzige Möglichkeit diese "Zombies" zu entfernen besteht darin die DBF-Datei inkl. der Memodatei zu reorganisieren.

Mit dem Programm **Progr4.prf** kann dies erfolgen.

Im Programm werden zuerst die Kunden.dbf- und die Kunden.dbt Dateien umbenannt. Danach werden die Dateien Kunden.dbf und Kunden.dbt leer neu erstellt. Nun wird die alte, umbenannte DBF-Datei gelesen und jeder Satz in die zuvor neu erstellte DBF-Datei übertragen. Es wird geprüft ob ein Memosatz existiert und ob dieser existierende Memotext auch einen Inhalt hat. Nur dann wird auch der Memotext in die Memodatei übernommen. Das heißt, ein komplett leerer Memotext wird ignoriert.

Bei der Übernahme aller Felder vom Eingabesatz in den Ausgabesatz tritt folgende Unbequemlichkeit auf:

Auf Satzfelder einer DBF-Datei kann nur zugegriffen werden, wenn der Satzpuffer mit **@dbUse** aktiviert ist. Zur selben Zeit kann immer nur ein Puffer aktiviert sein. Somit kann der Inhalt eines Satzfeldes nicht direkt vom Eingabesatz in den Ausgabesatz übertragen werden. Er muß deshalb zwischengelagert werden.

als Beispiel:

```
@dbUse(#1)           ' bezieht sich auf die Eingabedatei
Zw$ = @dbGet$("NAME")
@dbUse(#2)           ' bezieht sich auf die Ausgabedatei
@dbPut("NAME",Zw$)
@dbUse(#1)
Zw$ = @dbGet$("DATUM")
@dbUse(#2)
@dbPut("DATUM",Zw$)
' u.s.w.
```

Alternativ können aber auch alle Eingabefelder zuerst in jeweils eigenen Stringvariablen gespeichert und danach (mit **@dbUse(#2)**) in den Ausgabesatz übertragen werden.

Es gibt aber eine elegantere Methode.

Wir sprechen die Satzfelder nicht mit dem Namen sondern mit der Nummer des Feldes im Satzaufbau an!

In einer Schleife verwenden wir eine Laufvariable und lesen ein Feld mit Hilfe dieser Laufvariablen und übertragen den Inhalt als Zeile in die Listboxliste. Nachdem alle Feldinhalte (eines Satzes) in

die Liste übertragen wurden wechseln wir mit **@dbUse** zum Ausgabe-Satzpuffer. Jetzt lesen wir, wieder in einer Schleife, die Zeilen der Listboxliste aus und übertragen sie in die Ausgabefelder, die wir wieder über die Laufvariable mit der Feldnummer ansprechen.

Das ist wenig Programmieraufwand, es gibt aber Einschränkungen:

der Satzaufbau der Ausgabedatei muß gleich dem Satzaufbau der Eingabedatei sein. Allerdings können die Feldnamen im Ausgabesatz anders lauten. Die Feldarten sollten gleich sein, die Längen der einzelnen Felder können auch differieren

Memofelder können auf diese Art nicht behandelt werden. Das bedeutet, dass, wenn sich mitten im Satzaufbau ein Memofeld befindet, dieses nicht in der Schleife behandelt werden kann. Memofelder müssen in der Form verarbeitet werden, dass mit dem Feldnamen und nicht mit der Nummer des Feldes gearbeitet wird.

Da im Satzaufbau der Datei Kunden.DBF das (einzige) Memofeld als letztes angeordnet ist, kann daher im Programm **Progr4.prf** die Schleife ohne Unterbrechung die Felder 1 bis 10 problemlos verarbeiten.

[Zurück](#)

Lektion 5.

Im **Progr5.prf** ist eine Zusammenfassung der Behandlung einer dBase-Datei mit 2 zugeordneten Indexdateien.

Das Beispielprogramm benutzt folgende Funktionen:

- erstellen einer dBase III Datei DbTest.DBF
- schreiben von Sätzen in die Datei
- listen der Datei nach Satzfolge (Satzfolge = wie die Sätze in die Datei geschrieben wurden)
- bilden von 2 Indexdateien (DbTest1.NDX und DbTest2.NDX)
- listen der Datei nach Index1 und nach Index2
- hinzufügen von 2 neuen Sätzen (mit gleichzeitiger Wartung der Indexdateien)
- erneut listen der Datei nach Satzfolge !!
- erneut listen nach Indexdateien (fortlaufend nach Index)
- listen von bestimmten Sätzen (nach Schlüsselbegriff in einer Indexdatei)
- suchen und listen eines Satzes. Mit Suchbegriff - nicht über den Index

Ablauf:

Zuerst wird die DBF-Datei erstellt, 5 Sätze hinzugefügt und die DBF-Datei gelistet (nach Satzfolge).

Nun werden die 2 Indexdateien erstellt und gelistet (jede Indexdatei nach dem Schlüssel aufsteigend).

Die DBF-Datei wird wieder nach Satzfolge gelistet (dies soll zeigen, wie das erfolgen kann, wenn (mit **@dbIndex**) ein Index zugeordnet ist.

Anschließend werden 2 neue Sätze hinzugefügt und wieder die Dateien gelistet:

- nach Satzfolge
- nach Index1 und Index2 (nach dem Index fortlaufend)
- nach ausgewählten Schlüsseln im Index1 (mit **@dbFind**)
- und ein Satz nach einem Suchbegriff wobei hier unabhängig von einer Indexdatei, der Satz (mit **@dbSeek**) in einem nicht-indexierten Feld gesucht wird.

Zu beachten ist: bei jeder Zuordnung einer Indexdatei (mit **@dbIndex**) müssen alle benutzten Indexdateien angeführt werden! Es wird aber immer nur die erste der angeführten Indexdateien für die nächsten Dateizugriffe wirksam (leider auch beim Hinzufügen eines neuen Satzes - es wird nur die erste angeführte Indexdatei updated).

Zum Hinzufügen eines neuen Satzes sollte die Prozedur SatzAus benutzt werden. Dies ist grundsätzlich richtig. Der Satz wird zur DBF-Datei hinzugefügt. Ist aber ein Index (mit **@dbIndex**) zugeordnet funktioniert das nur mit einer Indexdatei richtig.

In der Datenbankbearbeitung (ein zu Profan zugekauft Produkt) steckt ein Bug. Sind zwei Indexdateien in Verwendung wird nur die erste der mit **@dbIndex** zugeordneten Indexdateien updated, die zweite Indexdatei bleibt unverändert! Damit das Update in beiden Indexdateien erfolgt,

wird die Prozedur [Add_Satz](#) an Stelle der Prozedur [SatzAus](#) verwendet. So ist es möglich den Fehler zu umgehen.

Würden nach jedem hinzufügen mit der Prozedur [SatzAus](#) die Indexdateien mit [@dbCreateIndex](#) neu erstellt so wären auch alle Indexdateien auf dem letzten Stand. Dass dies langsam und daher nicht sinnvoll ist, liegt auf der Hand.

Dieses Beispielprogramm sollte wesentliche Dateizugriffe zeigen.

Natürlich ist das Beispiel auch anwendbar, wenn nur eine Indexdatei verwendet wird (bei [@dbIndex](#) ist dann eben nur die eine Indexdatei anzuführen und zum Schreiben eines neuen Satzes wird die Prozedur [SatzAus](#) statt [Add_Satz](#) verwendet).

[zurück](#)