Datenbankprogrammierung mit Visual Basic

Einführung

Im ersten Teil möchte ich mich mit der Datenbankprogrammierung mit Hilfe des Datensteuerelements widmen. Der zweite Teil wird sich mit der Programmierung des Datenbankzugriffs von Hand befassen.

Einführung zum VB-Projekt Adressverwaltung

Ich habe vor, ein kleines Adressverwaltungsprogramm zu programmieren, das seine Datensätze in deiner Access-Datenbank speichert. Für dieses Vorhaben setze ich das Datensteuerelement ein.

Datensteuerelement

Man muss gegenüber der Programmierung von Hand etwas weniger programmieren. Es reicht das Datensteuerelement in der Form zu platzieren und die richtigen Einstellungen zu treffen. Doch die sogenannte Bindung von Steuerelementen (z.B. Textfeldern) kann praktisch ohne Programmieraufwand schnell eine Datenbankanwendung erstellt werden. Den Zugriff auf die Datenbank übernimmt größtenteils das Datensteuerelement.
Die Benutzung des Datensteuerelements ist für Einsteiger empfehlenswert, da bei der Programmierung von Hand mehr Verständnis der Datenbank vorhanden sein muss und man mehr selbst von Hand programmieren muss. Den Einsteiger schreckt dies erst einmal ab. Aus diesem Grund mache ich den Datenbankzugriff erst einmal mit dem Datensteuerelement. Im nächsten Workshop werde ich dann die selbsterstellte Lösung verwenden, die auch einige Vorteile bietet, die ich dann erläutern werde.

Datenbank für das Projekt anlegen

Zuallererst einmal muss eine Datenbank erstellt werden, in die die Daten gespeichert werden. Das kann auf zwei Wege gehen: Mit Access oder mit dem Visual Data Manager der bei Visual Basic mit geliefert wird.
Da jeder Visual Basic-Programmierer den Visual Data Manager besitzt, werde ich die Erstellung einer Datenbank mit diesem Programm erklären.
Den "*Visual Data Manager*" findest du in der IDE unter "*Add-Ins/Visual Data Manager*". Vorher musst du natürlich Visual Basic startet und ein "*Standart-EXE"-Projekt*" erstellen. Ein Klick auf den Menüpunkt startet das Programm. Wähle dann "*Datei/Neu/Microsoft Access/Version 7.0 MDB*" um eine neue Datenbank zu erstellen. Du musst der Datenbank noch einen Namen geben. In dem dann erscheinenden Fenster, steht der Eintrag "*Properties*", klicke mit der rechten Maustaste darauf, ein Kontextmenü öffnet sich, klicke dort auf "*Neue Tabelle*". Eine Eingabemaske erscheint in der du den Tabellennamen noch eingeben musst. Gib "*Adress*" ein, klicke dann auf "*Feld hinzufügen*", eine weitere Eingabemaske erscheint, wo du nun den Feldnamen "*Vorname*" eingibst und die Voreinstellung "*Text*" beibehältst.
Selbstverständlich sind je nach Zweck andere Felddatentypen sinnvoll, wenn zum Beispiel in ein Feld nur Zahlen kommen sollen lohnt sich die Einstellung "*Zahl*". Durch diesen Datentyp lassen sich Ressourcen sparen, denn der "*Text*"-Datentyp verbraucht mehr als der "*Zahl*"-Datentyp. Es gibt nämlich mehr verschiedene Buchstaben als Zahlen, den die Datenbank und später die Jet-Engine (Verbindungsglied zwischen deinem Programm und der Datenbank) im Speicher reservieren muss. Bei wenigen Datensätzen fällt dies nicht ins Gewicht, doch wenn man 500 Datensätze hat, merkt man dies deutlich.

Klicke, wenn du den Namen eingegeben hast, auf "*OK*". Im Hintergrund siehst du das neuerstellte Feld. Wiederhole den Vorgang für "*Nachname*", "*Strasse*", "*PLZOrt*", "*Telefon*" und "*Bemerkungen*". Wenn du all die Felder eingegeben hast, klicke auf "*Schließen*". Das Fenster sollte unter "*Felder*" jetzt so aussehen:

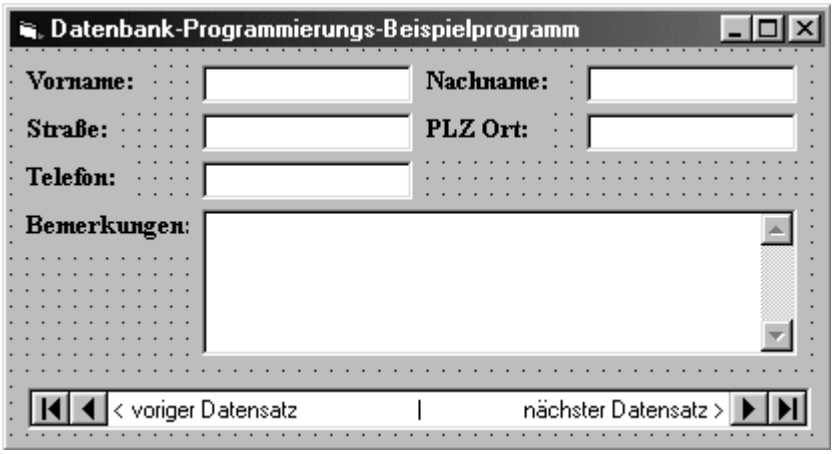
Drücke nun auf "*Tabelle erstellen*" um die Tabelle zu erstellen. Mit dem Schließen vom Visual Data Manager ist die Datenbank erstellt.

Datenbank in das VB-Projekt einbinden

Nun, wenn du die Datenbank hast, kann es in Visual Basic mit der Programmierung des Programms weitergehen. Zuerst werden wir das Datensteuerelement () in unser Formular einfügen. Ziehe dazu das Steuerelement auf bis es etwa so breit ist wie das Formular in das das Steuerelement eingefügt wird. Dann gehe in den Eigenschaften-Dialog und wähle die Eigenschaft "*Database Name*" aus, in dem Feld dahinter erscheint eine Schaltfläche mit drei Punkten, wenn du auf diese Schaltfläche klickst, öffnet sich ein Fenster in dem du den Pfad angeben musst, unter dem sich die Datenbank befindet. Wähle den Pfad der Datenbank aus. Drücke dann auf "*Öffnen*", hinter der Eigenschaft *Database Name* erscheint nun der Pfad der ausgewählten Datenbank. Dann musst du Visual Basic noch sagen in welche Tabelle die Daten eingetragen werden sollen. Dies geschieht mit der Eigenschaft "*RecordSource*" (recht weit unten in dem Dialog). Klicke auf das Pfeilchen am des Feldes, und wähle "*Adress*" aus, eine andere Auswahl dürfte es nicht geben. Es wäre darüber hinaus auch noch schön, wenn die Aufschrift des Daten-Steuerelements etwas schöner wäre. Dies kann man mit der Eigenschaft "*Caption*" verändern, diese Eigenschaft dürfte dir noch von den Bezeichnungssteuerelementen aus der Lektion "Programmieren mit VB" bekannt sein, gib hinter das Eigenschaftsfeld die Beschriftung "*< voriger Datensatz | nächster Datensatz >*" ein. Nun weiß das Daten-Steuerelement wo sich die Datenbank befindet, in welche Tabelle die Daten geschrieben werden sollen und es hat eine schönere Beschriftung, doch es können noch keine Daten eingegeben werden. Wo soll man die denn eingeben? Ganz einfach! In ein Textfeld, dafür sind sie da! Und dies geht ganz ohne programmieren, und zwar durch Bindung. Dies bedeutet, das alles was in das Textfeld eingegeben wird AUTOMATISCH in EIN Feld in der Datenbank übertragen wird, z.B. der Vorname, der in das Textfeld für den Vornamen kommt, wird sofort in das Feld "*Vorname*" der Datenbank geschrieben.

Formulardesign

Also erzeuge erst einmal für jedes erstellte Feld der Datenbank ein Bezeichnungsfeld und ein Textfeld. Die "*Caption*"-Eigenschaften der Bezeichnungsfelder sind die jeweiligen Namen der Felder der Datenbank, die Text-Eigenschaften der Textfelder sind leer. Das Formular sollte jetzt etwa so aussehen:



Die Schriftart und -größe veränderst du mit Hilfe der Eigenschaft "*Font*". Bei dem Feld "*Bemerkungen*", handelt es sich um ein Feld in das auch mehrzeilige Eingaben möglich sind. Dies kann man mit Hilfe der Eigenschaft "*MultiLine*" einstellen. Setze die Eigenschaft auf "*True*". Außerdem hat die Textbox eine Scroll-Bar für Texte die über dieses Feld hinausgehen. Dies wird mit der Eigenschaft "*ScrollBars*" erreicht. Setze die Eigenschaft auf "*2 - Vertikal*" damit der Balken am rechten Rand erscheinen.

Textfelder an Datenbank binden

Nun müssen die Textfelder noch an die Felder der Datenbank gebunden werden. Als erstes muss hierfür die Eigenschaft "*DataSource*" aller Textfelder auf "*Data1*" gesetzt werden (So heißt das Datensteuerelement). Dies geschieht mit dem Pfeil am Rand des Feldes. Wenn du nun bei der Eigenschaft "*DataField*" auf den Pfeil klickst erscheinen in einer Liste alle verfügbaren Felder der Datenbank. Wähle für jedes Textfeld das zugehörige Feld der Datenbank aus. Nun sind die Textfelder an die Datenbank gebunden. Doch die Datenbank enthält noch keine Datensätze, die muss der Benutzer erst eingeben. Damit er dies tun kann, bauen wir diese Funktionen ein. Dies geht leider nicht per drag & drop. Diese Funktion muss programmiert werden.

Die Buttons und ihre zukünftigen Funktionen

Ich möchte in das Programm zwei Buttons einfügen, mit dem einen soll man Datensätze löschen können, mit dem anderen neue erstellen. Also müssen zuerst zwei Knöpfe eingefügt werden, den einen mit der Aufschrift "*Löschen*" den anderen mit der Aufschrift "*Einfügen*" ("*Caption*"-Eigenschaft). Mache dein Formular größer, wenn die Knöpfe nicht hereinpassen. Nun werden wir die Knöpfe mit Funktionen ausstatten.

Damit wir beim Programmieren noch wissen welches Feld welches ist, geben wir dem Feld "*Vorname*" einen neuen Namen. Das Feld soll "*txtVName*" heißen, du wirst dich sicher fragen warum das Feld einen so komischen Namen hat, *txt* in dem Namen ist ein Präfix, das nach der ungarischen Notation dem Namen vorangestellt wird. Diese Namenskonvention besagt, das jedes Objekt ein Präfix hat, so z.B. cmd (=Befehlsschaltfläche), lbl (=Bezeichnungsfeld), dat (=Daten-Steuerelement), frm (=Formular), mnu (=Menü) und eben txt (=Textfeld).

Also gib hinter der Eigenschaft "N*ame"* des Textfeldes in das der Vorname soll, "*txtVName*" ein.

Der Quelltext des ersten Buttons

Doppelklicke dann auf den "*Einfügen*"-Button um Quelltext einzugeben, gib dann ein:

txtVName.SetFocus
Data1.Recordset.AddNew

Die erste Anweisung weist Visual Basic an, den Cursor in das Feld "*txtVName*" zu setzen. Die zweite weist den Recordset des Datensteuerelements an, einen Datensatz anzulegen.
Jetzt kannst du das Programm schon mal ausprobieren.

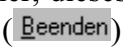
Der Quelltext des zweiten Buttons

Wenn der Benutzer in das Programm einen Freund eingetragen hat, den er jetzt nicht mehr ausstehen kann, dann möchte er diese Adresse auch wieder löschen. Diese Funktion wird der "*Löschen*"-Knopf übernehmen. Also doppelklicke auf den "*Löschen*"-Knopf und gib ein:

```
Dim prompt$ as String
Dim reply as String
prompt$ = "Wollen Sie diesen Datensatz WIRKLICH löschen?"
reply = MsgBox(prompt$, vbOKCancel, "Datensatz löschen")
If reply = vbOK Then
Data1.Recordset.Delete
Data1.Recordset.MoveNext
End If
```

Die ersten zwei Anweisungen nach der Deklaration der erforderlichen Variablen geben eine Meldung aus, die sicherstellt das sich der Benutzer nicht verklickt hat. Wenn der "*OK*"-Knopf (vbOK) dann gedrückt wird, wird der Datensatz gelöscht und der Zeiger einen Datensatz weiter geschoben. Zeiger heißt, die Markierung des aktuell angezeigten Datensatzes. Wenn der Zeiger auf dem gerade gelöschten Datensatz steht, den es dann nicht mehr gibt, stehen bleibt kommt es zu einem Laufzeitfehler.

Das Menü einbauen

Das Programm ist im Grunde fertig, doch die Art das Programm zu beenden (über Schließen-Kreuzchen) ist noch etwas schlecht. Wir werden noch ein Menü einfügen mit dem man das Programm beenden kann. Also öffne den Menü-Editor () aus der Symbolleiste durch einen Klick darauf. Nun gib bei Caption "&Beenden" ein. Nein, & ist kein Schreibfehler, dieses Kaufmannsund macht einen kleinen Strich unter den Buchstaben danach (

). So kann man hier beispielsweise durch Druck von [Alt/B] den Menüpunkt öffnen und muss nicht erst extra darauf klicken. Gebe dann unter "*Name*" "*mnuEnd*" ein. Der Namenspräfix *mnu* vor "*mnuEnd*" bedeutet das es sich um einen Menüpunkt handelt. Mit einem Klick auf "*OK*" wird deinem Fenster eine Menüleiste hinzugefügt. Nun müssen wir noch dem Menüpunkt sagen, was er machen soll, wenn auf ihn geklickt wird. Klicke hierfür auf den Menüpunkt, als wolltest du ihn öffnen. Das Code-Fenster öffnet sich und wartet auf deine Quelltexteingabe.

Gib ein:

End

Mit dieser Anweisung wird dein Programm geschlossen.

Was ist nochmal...

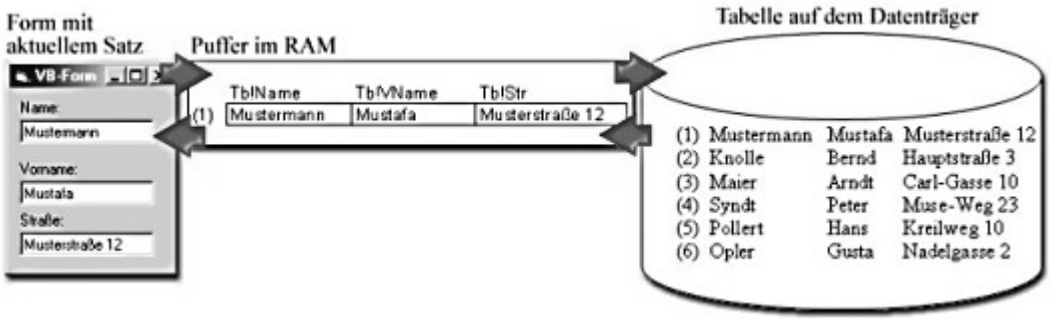
- Namenskonvention, ungarische Notation
  Diese Konvention soll helfen die Objekte später besser wieder an ihrem Namen erkennen zu können. txtBeispiel lässt sich besser merken als Text11. Die Notation vereinfachlicht das Ganze durch genormte Präfixe (txt, lbl, cmd).
- Binden
  Binden heißt, ein Ausgabefeld (Textfeld, Bezeichnungsfeld) mit einem Felder einer Datenbank zu verbinden. Als Vermittler zwischen Programm und Datenbank agiert die Jet-Engine, die Datenbankfunktionen wie erstellen, löschen, suchen, Zeiger bewegen, usw. für das Programm bereit stellt.
- Zeiger
  Ein Zeiger ist die Markierung des aktuell aufgerufenen Datensatzes. Beim Löschen muss darauf geachtet werden, das der Zeiger nicht auf ein nicht mehr vorhandenes Feld zeigt, ansonsten kommt es zu einem Laufzeitfehler.

Einführung in die zweite Art des Datenbankzugriffs

Bevor ich mit dem Thema Datenbankzugriff selbst programmieren beginne, möchte ich noch etwas Datenbanktheorie einschieben, da man ansonsten einige Dinge beim späteren Programmieren nicht verstehen kann.
Sehr fleißige werden schon im Access-Workshop geschaut haben (und ihn hoffentlich durchgemacht haben), doch auch diese werden in dieser Datenbankeinführung einige Neuigkeiten erfahren.

Der Datenbankzugriff

Erst mal eine grafische Darstellung des Datenbankzugriffs:



Zuerst einmal das Lesen von Daten: Aus der Tabelle einer Datenbank wird der mit dem Datensatzzeiger aktuell markierte Datensatz in den Hauptspeicher geladen. Dort befindet sich ein Puffer der die Daten zwischenspeichert. Einer der Aufgaben des Programmierers ist es die Daten in den Puffer zu bekommen und von dem Puffer aus korrekt in der Form anzeigen zu lassen.
Beim Speichern wird der umgekehrte Weg gegangen. Änderungen werden erst in den

Puffer im RAM gespeichert. Von da aus werden die Änderungen in den Feldern der Datenbank gesichert.

Der Lesezugriff

Ich möchte nun den lesenden Zugriff auf die Datenbank genau behandeln.
Die Datenbank liegt dem Programmierer vor, zuerst einmal muss er sich um das Puffern des aktuellen Datensatzes in den RAM kümmern. Dies ist eigentlich recht simpel; man benötigt zwei Variablen, die die Verbindung zur Datenbank verstellen. Die eine Variable ist für das öffnen der Datenbank zuständig und die andere für den Zugriff auf die Felder der Datenbank.
Die Variable werden folgendermaßen deklariert:
Dim Db as Database
Dim Tb as Recordset

Die erste Variable ist also sinnigerweise eine Variable vom Typ Database. Der Zugriff auf die Felder der Datenbank wird mit einer Variable vom Typ Recordset vollzogen. Nun zu der Zuweisung der beiden Variablen:
Set Db = OpenDatabase ("db.mdb")
Set Tb = Db.OpenRecordset ("Tab1", dbOpenDynaset)

Bei der ersten Zuweisung ist db.mdb der Name der Datenbank und Db, die vorhin deklarierte Variable. Die zweite Zuweisung sieht schon etwas komplizierter aus. Tab1 ist hier der Tabellenname und Tb die vorhin deklarierte Variable. Das öffnen der Tabelle geht mit dem soeben definierten Objekt DB, dessen Attribut OpenRecordset für das Öffnen einer Tabelle zuständig ist. Das Attribut dbOpenRecordset sorgt dafür, dass die Tabelle als Dynaset geöffnet wird, d.h. Änderungen werden erst beim nochmaligen laden des Dynasetobjekts sichtbar. Dies geschieht, wenn man z.B. einen Datensatz weitergeht.
Mit diesen beiden Anweisungen ist a) die Datenbank geöffnet worden und b) der RAM mit dem aktuellen Datensatz gefüllt worden. Nun müssen die Datensätze des Puffers auf die Form gebracht werden. Dies geht in dem man dem Textfeld folgendes zuweist:
txtName.Text = Tb!Name

Dem Textfeld wird der Inhalt des Datenfeldes im RAM zugewiesen. Die Inhalte der Datenfelder werden im RAM im folgenden Format gespeichert:
Tb!Datenfeldname

Wobei Tb selbstredend die oben deklarierte Variable ist. Mit ! und dem Datenfeldname aus der Tabelle wird der Inhalt in das Textfeld kopiert.

Schreibzugriff

Etwas komplizierter ist da schon der schreibende Zugriff. Im Grunde genommen geht man zwar den entgegengesetzten Weg des lesenden Zugriffs, also:
Tb!Name = txtName.Text
Jedoch sind noch weitere Befehle notwendig um die Daten dann auch in der Datenbank zu speichern. Diese möchte ich nun im Folgenden behandeln.
Das Ändern wird mit zwei Befehlen vollzogen:
Tb.Edit
Tb!Name = txtName.Text

Tb.Update
Tb ist hier wieder die Variable von oben. Mit der ersten Anweisung wird die Tabelle in den Bearbeitungsmodus versetzt, d.h. erst nachdem die Tabelle im Bearbeitungsmodus ist, können Felder verändert werden. Zuvor ist die Tabelle für den schreibenden Zugriff gesperrt. Danach wird die oben beschriebene Zuweisung in das Datenfeld gemacht und mit der letzten Anweisung wird die Änderung mit Update gespeichert und ab dann angezeigt. Der letzte Befehl hängt auch damit zusammen, dass wir den Recordset als Dynaset geöffnet haben.
Das war die Änderung bestehender Datensätze; natürlich kann man auch neue Datensätze in die Tabelle einfügen. Dies geht so:
Tb.AddNew

Jetzt ist an die Tabelle ein neuer Datensatz angehängt worden. Wenn man die Felder dann mit Inhalt gefüllt hat, müsste man die letzte Anweisung incl. der Zuweisung noch mal durchlaufen, damit die Änderungen wirksam werden.
Natürlich ist es auch notwendig Datensätze wieder zu entfernen, die geht damit:
Tb.Delete

Bei diesem Befehl muss man jedoch noch einiges beachten, zum Einen ob es sinnvoll ist, den Befehl ohne Sicherheitsabfrage auszuführen und zum Anderen, dass man den Datensatzzeiger um eins weiterverschieben muss. Vor allem das Letztere birgt Fehlerpotential in sich.

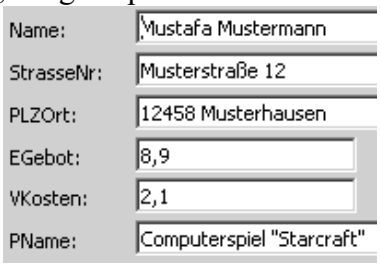
Einführung
Die meisten werden den ersten Teil lesen und sagen, dass doch alles wunderbar funktioniert hat und man hat mit dem schönen Steuerelement eigentlich wenig Arbeit. Eigentlich perfekt. Doch richtige Programmierer setzen bei ihren Programmen nicht das Datensteuerelement ein, da dies einige Probleme mit sich bringt. So werden mehr Dateien für eine Installation benötigt (das Steuerelement und die DAO-Dateien), außerdem ist das Datensteuerelement viel zu langsam und es funktioniert nicht in allen Fällen, so wie es der Programmier sich wünscht.
Warum also von dem Datensteuerelement einschränken lassen, wenn es alles auch selbstgemacht geht. Darum möchte ich euch in diesem Workshop in die Kunst der Datenbankprogrammierung aus der Sicht des richtigen Programmierers zeigen.

Theorie: Fertig!
Der selbstprogrammierte Datenbankzugriff ist bedeutend schneller als der Zugriff über ein Data-Steuerelement, außerdem ermöglicht er später fortgeschrittene Methoden des Datenbankzugriffs, z.B. EmbeddedSQL in Visual Basic.
Ich habe mir vorgestellt, eine Datenbankanwendung zu erstellen, die die Daten für den eBay-Verkauf beinhalten. Also eine Tabelle die die Adresse des Käufers speichert, das Endgebot, die Versandkosten und den Namen des Produkts.

Datenbank für das Projekt anlegen
Wie bei allen Datenbankanwendungen muss zuallererst eine Datenbank erstellt werden. Ich möchte die Erstellung diesmal jedoch ganz schnell abtun, für alle, denen es jetzt zu schnell geht, sei gesagt, dass die Erstellung der Datenbank mit dem Visual Data Manager im ersten Workshop sehr ausführlich beschrieben ist.
Wir erstellen eine *Microsoft Access 7.0 MDB*, gib der Datenbank einen einprägsamen Namen (z.B. ebay). Erstelle dann mit einem Rechtsklick auf "*Properties*" und der

Auswahl von "*Neue Tabelle*" aus dem Kontextmenü eine neue Tabelle.
Gebe der Tabelle den Namen "*ebay*". Erstelle dann folgende Felder: "*Name*" (Text), "*StrasseNr*" (Text), "*PLZOrt*" (Text), "*EGebot*" (Currency), "*VKosten*" (Currency) und "*PName*" (Text). Mit "*Feld hinzufügen*" wird das Feld jeweils in der Datenbank gespeichert und ein neues Datenfeld vorbereitet.
Das Fenster sollte unter "*Felder*" jetzt so aussehen:

Nachdem du auf "*Tabelle erstellen*" geklickt hast, ist die neue Tabelle erstellt worden. Bevor du nun den Visual Data Manager schließt, wollen wir noch einen Datensatz einfügen, damit wir später sofort sehen, ob die Programmierung funktioniert hat. Doppelklicke auf den Namen der Tabelle. Im nächsten Fenster klickst du erst einmal auf "Hinzufügen", danach erst gibst du die Daten in die Felder ein. Lasse dir irgend etwas einfallen, was ist egal, für ganz phantasielose hier mein Eintrag:



Schließe danach alles, nun geht es ans richtige programmieren!

Formulardesign
Doch bevor wir wirklich ans Werk gehen, designen wir noch unsere Form. Erzeuge für jedes erstellte Feld der Datenbank ein Bezeichnungsfeld und ein Textfeld. Die "*Caption*"-Eigenschaften der Bezeichnungsfelder sind die jeweiligen Namen der Felder der Datenbank, die Text-Eigenschaften der Textfelder sind leer.
Nenne Textfeld des Produktnamens *txtPName*, das des Namen des Käufers *txtName*, das mit der Straße *txtStrNr*, das mit PLZ und Ort *txtPLZOrt*, das mit dem Höchstgebot *txtHGeb* und das mit den Versandkosten *txtVKosten*. Nun brauchen wir noch 7 Buttons. Die ersten vier für das Navigieren in der Datenbank und die anderen drei für die Aktionen Hinzufügen, Ändern und Löschen. Die Beschriftungen und wie die Form jetzt aussehen könnte hier:

Den Buttons haben wir noch keine eindeutigen Namen gegeben. Nenne den ganz linken Knopf der vier Navigationsknöpfe *cmdFirst*, den nächsten *cmdPrev*, den nächsten *cmdNext* und den letzten der vier *cmdLast*. Den Hinzufügen-Button nennst du *cmdAdd*, den Ändern-Button *cmdEdit* und den Löschen-Button *cmdDel*.

Grundgerüst der Datenbankanwendung erstellen

Gut aussehen tut unser Programm nun schon einmal, jedoch kann es bis jetzt noch nichts machen.
Bevor wir jedoch mit dem programmieren beginnen können, müssen wir dem Programm noch einen Verweis hinzufügen. Gehe unter dem Menüpunkt *Projekt* auf *Verweise* suche dort den Eintrag "Microsoft DAO 3.6 Object Library" (eine andere Version ist auch OK!) und mache einen Hacken davor.
Als erstes wollen wir nun die Anbindung an die Datenbank realisieren. Definiere die Variablen Db und Tb:

```
Dim Db as Database
Dim Tb as Recordset
```

Gehe dann in die Prozedur Form_Load und tippe folgendes ein:

```
Set Db = OpenDatabase(App.Path & "\ebay.mdb")
Set Tb = Db.OpenRecordset("ebay", dbOpenDynaset)
```

Die Anweisung App.Path und das \ vor dem Datenbanknamen ist notwendig, dass die Datenbankanbindung auch unter der IDE von Visual Basic läuft. Ansonsten werden alle Dateien im VB-Verzeichnis gesucht. Mit App.Path wird in das Verzeichnis der Anwendungsdatei gewechselt.
So nun ist die Datenbankverbindung zum Puffer im RAM realisiert. Nun ist es noch unsere Aufgabe die Daten von dort auf unsere Form zu bekommen. In der Theorie haben wir gelernt, dass dies mit *txtName = Tb!Name* geht, doch es wäre umständlich bei jeder Änderung der Anzeige alle Verknüpfungen herunter zu rattern. Dieses Problem wollen wir mit einer Prozedur lösen:

```
Private Sub Load()
txtPName.Text = Tb!PName
txtName.Text = Tb!Name
txtStrNr.Text = Tb!StrasseNr
txtPLZOrt.Text = Tb!PLZOrt
txtHGeb.Text = Tb!EGebot
txtVKosten.Text = Tb!VKosten
End Sub
```

Mit der Anweisung ***Load*** wird die Prozedur nun immer dann ausgeführt, wenn man im Programm die aktuellen Inhalte des Puffers benötigt.
Damit muss also der Form_Load-Prozedur noch der Befehl:

```
Load
```

angehängt werden. Nun könntest du das Programm schon mal ausprobieren. Wenn alles funktioniert, dann wird der erste Datensatz schon jetzt angezeigt. Damit haben wir das aller gröbste Grundgerüst schon einmal erstellt. Doch wirdlich viel kann das Programm noch nicht, weshalb wir im folgenden noch einige Funktionen hinzufügen werden. Schließlich haben wir noch unbelegte Buttons die noch auf Quellcode war-

ten.

Vorbereitung für weitere Schritte

Die zunächst wichtigste Eigenschaft nach der Anzeige der Datensätze ist die Navigation in der Datenbank. Die wollen wir nun in unsere Form einbauen. Das Problem ist jedoch, dass wir bis jetzt nur einen Datensatz haben, so dass eine Navigation noch nicht möglich ist.
Gehe also wieder in den Visual Data Manager und füge noch mindestens zwei weitere Datensätze hinzu. Beende danach den Visual Data Manager wieder.

Navigation einbauen

So wo wir nun mehrere Datensätze haben, macht die Navigation auch Sinn. Nun aber wirklich zum Programmieren. Der Zum-Ersten-Springen-Button-Quelltext muss so aussehen:

```
Tb.MoveFirst
Load
```

Mit MoveFirst wird der Datensatzzeiger auf den ersten Datensatz verschoben und mit Load dann die Datenfelder vom Puffer im RAM aus auf die Form ausgegeben.
Der Zum-Vorherigen-Springen-Button-Quelltext muss so aussehen:

```
Tb.MovePrevious
If Tb.BOF Then
Tb.MoveFirst
End If
Load
```

Ich habe mal das Problem, dass sich ergeben wir vorweggenommen und gelöst, denn was ist, wenn der Datensatzzeiger sich schon am Anfang der Datenbank (=BOF; **B**eginn **O**f **F**ile)befindet? Dann kann man nicht mehr zum vorherigen gehen. Ich habe das Problem so gelöst, dass ich überprüfe, ob der Datensatzzeiger am sich am BOF befindet, falls ja, bewege ich ihn wieder auf den ersten Datensatz zurück.
Der Zum-Nächsten-Springen-Button-Quelltext muss so aussehen:

```
Tb.MoveNext
If Tb.EOF Then
Tb.MoveLast
End If
Load
```

Das gleiche Problem wie bei dem vorherigen Button besteht auch bei diesem Button: Was tun wenn man am Ende der Datenbank (= EOF; **E**nd **O**f **F**ile) angelangt ist? Meine Lösung ist ähnlich wie bei dem vorherigen Button, dass ich den Datensatzzeiger auf den letzten Datensatz zurücksetze.
Der Zum-Letzer-Springen-Button-Quelltext muss so aussehen:

```
Tb.MoveLast
Load
```

Mit MoveLast wird der Datensatzzeiger auf den letzten Datensatz verschoben.
Nun ist das Programm vollfunktionsfähig, man kann in den Datensätzen navigieren. Nur wenn man die Datenbasis ohne den Visual Data Manager verändern will, hat man bis jetzt noch ein Problem. Aber es sind ja auch noch drei Buttons unbelegt. Den

schreibenden Zugriff wollen wir nun realisieren.

Die restlichen Buttons

Nun sollen auch die letzten drei Buttons mit Funktionen ausgestattet werden. Der erste davon ist der Hinzufügenbutton. Mit den folgenden Anweisungen wird aber nur ein leerer Datensatz gespeichert, d.h. die Eingaben werden nicht gespeichert. Darum muss man nach der Eingabe in den neuen Datensatz noch einmal den im späteren Verlauf erstellten Ändern-Button drücken. Nun aber erst einmal der Quellcode:

```
Tb.AddNew
Clear
txtPName.SetFocus
```

Das erste Kommando sollte dir aus der Theorie bekannt sein. Bei dem zweiten handelt es sich wiederum eine selbsterstellte Prozedur, die ist folgendermaßen aufgebaut:

```
Private Sub Clear()
txtPName.Text = ""
txtName.Text = ""
txtStrNr.Text = ""
txtPLZOrt.Text = ""
txtHGeb.Text = ""
txtVKosten.Text = ""
End Sub
```

Die Prozedur sorgt also dafür, dass alle Felder frei sind, mit dem SetFocus-Befehl wird der Cursor in das Textfeld PName gesetzt, so dass man gleich loslegen kann mit dem Eingeben.

Ändern-Button

Der Ändern-Button wird, wie soeben erwähnt, nicht nur zum bearbeiten bestehender Datensätze benötigt, sondern auch zum abspeichern der Eingaben in einen neuen Datensatz.
So wird die Funktion realisiert:

```
If Tb.EditMode = dbEditAdd Then
MsgBox ("Datensatz wurde angehängt")
Else
Tb.Edit
End If
Send
Tb.Update
MsgBox ("Datensatz erfolgreich geändert!")
```

Die Bedingungsabfrage überprüft, ob der Benutzer die Änderungen bei einem neuen Datensatz speichern will oder einen bestehenden Datensatz ändern will. Im letzteren Fall muss man die Tabelle noch in den Edit-Mode versetzen.
Wieder habe ich eine Prozedur vorweggenommen - Send - mit der Prozedur Send schreibe ich die Änderungen in den Puffer, analog zu der Prozedur Load beim Lesen der Datensätze.
Die Prozedur Send ist folgendermaßen aufgebaut:

```
Private Sub Send()
```

Tb!PName = txtPName.Text
Tb!Name = txtName.Text
Tb!StrasseNr = txtStrNr.Text
Tb!PLZOrt = txtPLZOrt.Text
Tb!EGebot = txtHGeb.Text
Tb!VKosten = txtVKosten.Text
End Sub

Es handelt sich ganz offensichtlich um das Gegenteil der Load-Prozedur. Nun noch zu den oberen Anweisungen: Mit Tb.Update schließlich werden die Daten gesichert, das haben wir ja schon genauer in der Theorie besprochen.

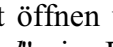
Der Löschen-Button

Nun ist es nur noch ein Button bis zur fertigen Datenbankanwendung. Dieser letzte Button ist der Löschen-Button. Der Quelltext dieses Buttons sieht so aus:

Dim OK As Integer
OK = MsgBox ("Datensatz wirklich löschen?", vbYesNo, "Löschen")
If OK = 6 Then
Tb.Delete
cmdNext_Click
End If

Zuerst mache ich eine Sicherheitsabfrage mit einer MsgBox, mit dem Attribut vbYesNo lege ich fest, dass die Knöpfe Ja und Nein zur Auswahl stehen. Wird Ja gedrückt, so ergibt sich der Rückgabewert 6 der MsgBox-Funktion. Ob als Rückgabewert 6 herauskommt, überprüfe ich mit der folgenden Bedingungsabfrage. Sollte Ja geklickt worden sein, dann wird mit Tb.Delete der Datensatz gelöscht. Der Datensatzzeiger muss noch um eins weiterverschoben werden, da er sonst ins Lerre zeigt; was wäre dazu besser geeignet als die cmdNext_Click Prozedur des Springe-Zum - Nächsten-Buttons. Damit ist auch gleich sicher gestellt, dass die Ausnahmefälle mit EOF und BOF beachtet werden.

Das Menü einbauen

Das Programm ist im Grunde fertig, doch die Art das Programm zu beenden (über Schließen-Kreuzchen) ist noch etwas schlecht. Wir werden noch ein Menü einfügen mit dem man das Programm beenden kann. Also öffne den Menü-Editor () aus der Symbolleiste durch einen Klick darauf. Nun gib bei Caption "&Beenden" ein. Nein, & ist kein Schreibfehler, dieses Kaufmannsund macht einen kleinen Strich unter den Buchstaben danach (

). So kann man hier beispielsweise durch Druck von [Alt/B] den Menüpunkt öffnen und muss nicht erst extra darauf klicken. Gebe dann unter "*Name*" "*mnuEnd*" ein. Der Namenspräfix *mnu* vor "*mnuEnd*" bedeutet das es sich um einen Menüpunkt handelt. Mit einem Klick auf "*OK*" wird deinem Fenster eine Menüleiste hinzugefügt. Nun müssen wir noch dem Menüpunkt sagen, was er machen soll, wenn auf ihn geklickt wird. Klicke hierfür auf den Menüpunkt, als wolltest du ihn öffnen. Das Code-Fenster öffnet sich und wartet auf deine Quelltexteingabe. Gib ein:

Tb.Close
Db.Close
End

Zuerst wird der Recordset und dann auch die Datenbank geschlossen, danach wird das ganze Programm beendet.

Einführung in embeddedSQL

Im letzten Workshop haben wir den Praxisteil zum selbstprogrammierten Datenzugriff behandelt. Nun wollen wir unser Programm noch etwas verfeinern, in dem wir eine weitere Technologie einsetzen, die mit dem selbstprogrammierten Datenzugriff einfach zu nutzen ist. Es handelt sich um SQL, das ist eine Abfragesprache, die nur dafür entworfen wurde Abfragen in Datenbanken zu erstellen. SQL heißt ausgeschrieben, Structured Query Language, also strukturierte Abfragesprache, wie ich eben gesagt habe. Abfragen sind - für alle die den Access-Workshop nicht gelesen haben - eine Schablone von Bedingungen die die Datensätze erfüllen müssen um angezeigt zu werden, so könnte eine Bedingung lauten, dass alle Datensätze angezeigt werden sollen die mit S beginnen.
Solche Anzeigeroutinen kann man natürlich auch mit Visual Basic erstellen, doch SQL bietet auch noch einige Features wie eine Sortierroutine, die in Visual Basic einige Arbeit bedeuten würde.

SQL in Visual Basic embedden

SQL Befehle, die für Abfragen erstellt werden, beginnen immer mit SELECT, so erkennt Visual Basic automatisch, dass es sich um eine SQL-Anweisung handelt.
Um also eine SQL-Anweisung in Visual Basic einzubinden definiert man eine Variable vom Typ String. Dieser weist man dann anschließend die SQL-Befehlsphrase zu. Danach wird die SQL-Anweisung ausgeführt, indem man beim Zuweisen einer Recordsetvariable statt des Tabellennamens die Variable einträgt. Eine typische SQL-Anweisung incl Aufruf sieht also so aus:

```
Dim sql as String
sql = "SELECT Name, EGebot FROM ebay WHERE Name LIKE '"& eingabe & "*'"
Set Tb = Db.OpenRecordset(sql)
```

In diesem Beispiel wird eine Tabelle erstellt, bei der alle Datensätze mit einer Zeichenfolge die in *eingabe* gespeichert ist übereinstimmen müssen, es dürfen aber noch Zeichen folgen.
Wenn du die Tabelle nun auf der Form ausgeben möchtest, so schreibe eine Do While-Schleife, die so aussehen könnte:

```
Do While Not (Tb.EOF)
Print Tb!Name, Tb!EGebot
Tb.MoveNext
Loop
```

Die Syntax einer SQL-Anweisung

Ich habe zwar eben ein Beispiel mit einer SQL-Anweisung gemacht, trotzdem dürftest du davon noch nicht sehr viel verstanden haben. Nun mal im Einzelnen, wie eine SQL-Anweisung aufgebaut ist:
SELECT *Datenfelder* ***FROM*** *Tabelle* ***WHERE*** *Bedingung*

Bei Datenfeldern handelt es sich um solche, die später in der erstellten Tabelle vorkommen sollen. Die Tabelle ist dann selbstverständlich die Tabelle in der sich die Felder befinden.
Mit INNER JOIN kann man mehrere Tabellen einbeziehen, die ein Verknüpfungsfeld

haben, die Syntax sähe so aus:
[..] FROM ebay INNER JOIN tab2 ON ebay.Name = tab2.Name [...]

[...] bedeutet, dass die obigen Befehle weggelassen wurden.

Sortierroutine
Ich sprach in der Einführung den Sortieralgorithmus an, den bindet man folgendermaßen ein:
[...] ORDER BY Name

Hänge das einfach an die bisherige SQL-Anweisung an und jetzt werden die Datensätze ausgehend von dem Feld Name automatisch nach dem Alphabet sortiert. Wenn du absteigend nach dem Alphabet sortieren möchtest, hänge hinten ***DESC*** an.

War das alles?
Nein, wenn du die vollen Möglichkeiten von SQL entdecken willst, dann werfe mal einen Blick in die msdn-Library. Dort findest du noch einige Optionen und auch die restlichen möglichen Befehle von SQL: ADD NEW mit dem neue Tabellen erzeugt werden können, DELETE mit dem Tabellen gelöscht werden können, INSERT mit dem eine Insertabfrage (beim Vergleich von zwei Tabellen fehlende Datensätze ergänzen) gestartet wird. Es gibt aber noch einige mehr.