durch sie im Rahmen einer Access-Anwendung oder im Datenbankfenster nicht direkt »anklickbar« wären). Das war es aber auch schon. In Zukunft müssen Datenbank-Management-Systeme in der Lage sein, komplexere Datentypen, wie Medienformate (etwa einen Asf- oder Real-Audio-Datenstrom) oder Objekte, direkt zu speichern. Dann wird es in VBA möglich sein, ein Objekt auf der Basis einer Klasse zu definieren, den Eigenschaften Werte zuzuweisen und das Objekt inklusive seiner Werte in einer Datenbank zu speichern[1].

2.4 Zusammenfassung

Eine Datenbank besteht aus Tabellen, die wiederum verschiedene Felder enthalten. Die Frage, wie die zu speichernden Daten auf die einzelnen Tabellen verteilt werden, ist von grundlegender Bedeutung für das Design einer Datenbank. Ein wichtiger Teilaspekt beim Entwurf einer Datenbank ist die Normalisierung. Unter Normalisierung versteht man das Aufteilen der einzelnen Felder auf verschiedene Tabellen, so daß das resultierende Datenbankmodell den Regeln relationaler Datenbanken gehorcht und die Redundanz der Daten auf ein »vernünftiges« Minimum reduziert wird. Damit der Normalisierungsprozeß überschaubarer wird, wird er in insgesamt fünf Stufen unterteilt, die auch als 1NF (1. Normalform), 2NF usw. bezeichnet werden. In der Praxis spielen allerdings lediglich die Normalformen 1NF, 2NF und 3NF eine Rolle. Eine »Übernormalisierung« ist allerdings auch nicht optimal, da diese sich negativ auf die Performance auswirken und in ungünstigen Fällen auch zu Datenverlusten führen kann (wenn die Beziehungen zwischen Daten durch die Aufteilung auf zu kleine Tabellen verloren geht – dieser Aspekt wurde in dem Kapitel aber nicht behandelt). Wie so oft im Leben kommt es auch beim Datendesign auf viel Erfahrung und den »goldenen Mittelweg« an.

2.5 Wie geht es weiter?

Den Theorieteil haben Sie nun absolviert, jetzt beginnt die Praxis. In Kapitel 3 bauen Sie die Beispieldatenbank des Buches (*Fuhrpark.mdb*) Schritt für Schritt auf. In Kapitel 4 lernen Sie mit dem ADO-Datensteuerelement die erste von mehreren Alternativen kennen, den Inhalt der Datenbank in einem Visual-Basic-Programm sichtbar zu machen.

---

1 Kommen wird diese Möglichkeit (Stichwort: »Objektorientierte Datenbank-Management-Systeme«) mit Sicherheit, doch vermutlich nicht bei der Jet-Engine, sondern beim SQL-Server.

2.6 Fragen

Frage 1:

Was sind die wichtigsten Merkmale einer relationalen Datenbank?

Frage 2:

Welche Vorteile bringt es, die zu speichernden Daten auf verschiedene Tabellen zu verteilen?

Frage 3:

Welche Hilfsmittel bietet Microsoft Access für das Datenbankdesign an?

Frage 4:

Eine Datenbank besteht aus der Tabelle *Autoren*, die folgende Felder enthält:

ISBN	Titel	Autoren
0-7821-2322-8	Standard C	Kernighan, Ritchie

Welchen (kleinen) Schwachpunkt besitzt diese Feldanordnung und wie läßt sich diese Tabelle in die 1. Normalform bringen?

Frage 5:

Welche Nachteile kann eine zu strenge Normalisierung, d.h. die Aufteilung auf zu viele kleine Tabellen, nach sich ziehen?

Frage 6:

Ein Datenbankprogrammierer möchte eine Datenbank für niedergelassene Ärzte programmieren. Da er keine Erfahrung im Datenbankdesign besitzt, packt er erst einmal alle Felder in eine Tabelle:

- PatientenNr
- Name
- Adresse
- Telefon
- DatumBesuch1
- DiagnoseBesuch1
- AbrechnungBesuch1
- DatumBesuch2

- ✘ DiagnoseBesuch2
- ✘ AbrechnungBesuch2

usw.

Kopfzerbrechen bereitet ihm allerdings die Frage, wie viele Felder für Arztbesuche vorgesehen sein sollen. Was ist, wenn der Patient mehr als zwei Besuche macht, wo liegt die Obergrenze der Besuche, wann kommt die nächste Gesundheitsreform? Wären in der Tabelle etwa Einträge für 100 Besuche vorgesehen, so würden die meisten Tabelle viele leere Felder enthalten, da die Durchschnittszahl bei den Besuchen bei 4,5 liegt. Können Sie dem Programmierer helfen?

Die Antworten zu den Fragen finden Sie in Anhang D.

Kapitel 3

Die erste Datenbank im »Eigenbau«

Nachdem die ersten beiden Kapitel des Buches relativ ausführlich auf den Aufbau und die »Theorie« einer Datenbank eingingen, erstellen Sie in diesem Kapitel Ihre erste Datenbank mit dem Visual Data Manager, einem überaus nützlichen Add-In von Visual Basic. Es wird eine sehr kleine (Access-)Datenbank sein, da es lediglich darum geht, die generelle Vorgehensweise beim Anlegen einer Datenbank zu veranschaulichen. Da die Struktur der Datenbank vorgegeben ist, müssen Sie sich also noch keine Gedanken um Normalisierungsregeln oder Beziehungen machen. Auch die SQL-Abfragen heben wir uns für später (Kapitel 7) auf. Auch wenn Sie vielleicht bereits Pläne und sicherlich jede Menge Ideen für eigene Datenbanken im Kopf haben dürften, sollten Sie sich in diesem Kapitel noch an die Vorgaben halten, da anhand des kleinen Beispiels der elementare Aufbau einer Datenbank veranschaulicht werden soll. Die in diesem Kapitel aufgebaute Datenbank finden Sie übrigens auch auf der Webseite des Buches (siehe Einleitung) in Gestalt der Datei *Fuhrpark.mdb*, wenngleich der Lerneffekt dieses Kapitels darin besteht, die Datenbank Schritt für Schritt umzusetzen.

Sie lernen in diesem Kapitel etwas über folgende Themen:

- ✘ Den Visual Database Manager
- ✘ Das Anlegen einer neuen Datenbank
- ✘ Das Anlegen einer Tabelle in einer Datenbank
- ✘ Das Anlegen von Feldern in einer Tabelle

- ✘ Die Eingabe von Daten
- ✘ Das Erstellen einer »Eingabemaske« für das Bearbeiten der Daten
- ✘ Das Hinzufügen einer weiteren Tabelle

3.1 Der Visual Data Manager

In diesem Abschnitt soll, ohne große Vorankündigung, der *Visual Data Manager* (kurz VDM, wenngleich diese Abkürzung nicht offiziell ist und daher im weiteren Verlauf nur an einigen Stellen verwendet wird[1]) vorgestellt werden. Visual Basic verfügt (bereits seit der Version 5.0) über eine Reihe komfortabler Add-Ins und seit der Version 6.0 auch über eine komfortable Möglichkeit, diese zu laden bzw. zu entladen. Der Visual Data Manager ist eines von ihnen. Mit seiner Hilfe können Sie unter anderem:

- ✘ Neue Datenbanken in verschiedenen Datenbankformaten erstellen
- ✘ Bereits vorhandene Datenbanken in verschiedenen Formaten öffnen
- ✘ Neue Tabellen und Felder anlegen
- ✘ Die Struktur der Tabellen und die Attribute der vorhandenen Felder ändern und ergänzen
- ✘ SQL-Abfragen erstellen, in der Datenbank speichern und ausführen
- ✘ Die Benutzer und Gruppen einer Access-Datenbank bearbeiten
- ✘ Formulare mit Datenbankfeldern erstellen
- ✘ Den Inhalt einer Datenbank zwecks Bearbeitung anzeigen

Sie müssen den Visual Data Manager nicht verwenden, er ist lediglich eine von mehreren Optionen. Wenn Sie über Microsoft Access verfügen, sollten Sie es bei etwas größeren Datenbanken dem Visual Data Manager vorziehen, da es als richtiges DBMS einfach über mehr Komfort verfügt (der Visual Data Manager ist nichts anderes ein mittelgroßes Visual-Basic-Programm). Dennoch können auch Access-Besitzer vom VDM profitieren, da

1 Unter dem Namen VisData gab es den Visual Data Manager schon bei Visual Basic 3.0, damals lediglich als gut verstecktes Beispielprogramm. In einem Unterverzeichnis liegt der VDM übrigens auch heute noch in Quelltextform vor, was ein hervorragendes Anschauungsmaterial für die Datenbankprogrammierung auf der Basis der DAOs darstellt. Man muß nur die nötige Zeit dafür haben, dann kann man durch Studieren der einzelnen Module (die meistens sehr gut kommentiert sind) eine Menge lernen.

es oft einfacher ist und schneller geht, den VDM aufzurufen, als Microsoft Access zu starten. Geht es darum, das Anlegen einer Datenbank Schritt für Schritt kennenzulernen, ist der VDM vielleicht sogar besser geeignet, da Microsoft Access mit seinen reichhaltigen Dialogfeldern und seinen Assistenten manchmal ein wenig zu sehr vom »Kern der Sache« ablenkt.

3.1.1 Der Aufruf des Visual Data Managers

Der Visual Data Manager kann wahlweise als eigenständiges Programm mit dem Namen *Visdata.exe* oder als Add-In gestartet werden.

1. Starten Sie Visual Basic, und öffnen Sie das ADD-INS-Menü.
2. Wählen Sie den Eintrag VISUAL DATA MANAGER.

Sollte der Eintrag nicht angeboten werden, müssen Sie zuerst den Eintrag ADD-IN-MANAGER wählen und aus der Liste der aufgeführten Add-Ins den Eintrag »Visual Data Manager« auswählen. Sollte dieser Eintrag wider Erwarten nicht aufgeführt sein, wurde Visual Basic offenbar nicht vollständig installiert[1].

Bild 3.1: Der Visual Data Manager wird über das Add-Ins-Menü gestartet

Für alle, die den Visual Data Manager in ihr Herz geschlossen haben: In der Registry lassen sich im Zweig *HKEY_CURRENT_USER\Software\VB and VBA Program Settings\Microsoft Visual Basic AddIns\VisData6* eine Reihe von Arbeitseinstellungen treffen.

3.1.2 Die Fenster des Visual Data Managers

Der VDM arbeitet mit zwei Fenstern: dem Datenbankfenster und einem Fenster mit dem Titel »SQL-Anweisung«. Während das Datenbankfenster die Struktur der aktuell geladenen Datenbank anzeigt (Sie können immer nur eine Datenbank auf einmal laden und bearbeiten), können Sie im SQL-Anweisungsfenster SQL-Anweisungen eintippen und ausführen. Mehr zum Thema SQL in Kapitel 7.

1 Dieser Fall sollte wirklich nur sehr selten vorkommen. Mir ist er jedenfalls noch nicht untergekommen.

Der Umgang mit dem Visual Data Manager ist praktisch selbsterklärend, gewisse Datenbankgrundkenntnisse natürlich vorausgesetzt. Datenbankkenner dürfte es interessieren, daß der Visual Data Manager auch bei Visual Basic 6.0 noch mit den *Data Access Objects* (DAOs) arbeitet, also keine ADOs und deren Besonderheiten kennt (auch ein Zugriff auf Access 2000-Datenbanken ist aufgrund des geänderten Dateiformats nicht möglich). Geht es nur um das Erstellen und Bearbeiten einer Datenbank, spielt dieser Aspekt keine Rolle, denn wenn die Datenbank erst einmal als Mdb-Datei auf der Festplatte vorhanden ist, ist es (zumindest bei kleineren Datenbanken) unerheblich, auf welche Weise sie erstellt wurde.

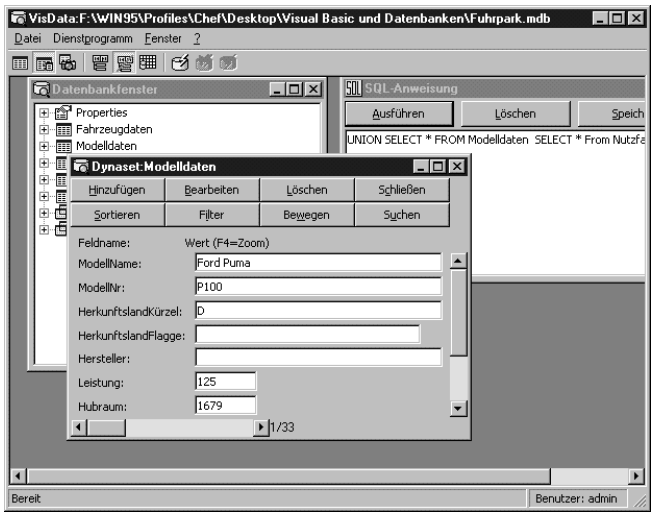


Bild 3.2: Der Visual Data Manager ist ein einfaches Programm

3.1.3 Die Menüs des Visual Data Managers

Die Menüs werden am Anfang nur selten benötigt, doch wer neugierig ist (und wer ist das nicht?), wird sicherlich schon einen Blick in drei Menüs geworfen haben. Im DATEI-Menü finden Sie die Kommandos zum Öffnen und Neuanlegen einer Datenbank. Außerdem werden hier Kommandos zum Importieren und Exportieren von Teilen der Datenbank zur Verfügung gestellt. Im Menü DIENSTPROGRAMM finden Sie den Abfragegenerator zum Erstellen von SQL-Kommandos, den Formulardesigner und ein paar sehr spezielle Kommandos, die Sie nur selten benötigen werden. Alle Menükommandos sind in Tabelle 3.1 zusammengefaßt.

Menü	Kommando	Bedeutung
DATEI	DATENBANK ÖFFNEN	Ermöglicht das Öffnen einer bereits vorhandenen Datenbank.
DATEI	NEU	Legt eine neue Datenbank in einem der zur Verfügung stehenden Formate an. Auf diese Weise ist es z.B. möglich, Datenbanken im alten dBase- oder Paradox-Format oder auch Textdateien im Datenbankformat anzulegen. Voraussetzung ist allerdings, daß der benötigte ISAM-Treiber vorhanden ist.
DATEI	SCHLIESSEN	Schließt die aktuell geöffnete Datenbank.
DATEI	IMPORTIEREN/ EXPORTIEREN	Ermöglicht das Importieren einer Tabelle aus einer Datenbank in die aktuelle Datenbank oder das Exportieren von Tabellen und SQL-Abfragen in eine vorhandene Datenbank. Leistungsfähige Funktion.
DATEI	WORKSPACE	Öffnet die Datenbank mit einem bestimmten Benutzer- und Workspace-Namen.
DATEI	FEHLER	Listet die zuletzt aufgetretenen Fehler auf.
DATEI	MDB KOMPRIMIEREN	Komprimiert eine bereits vorhandene Access-Datenbank.
DATEI	MDB REPARIEREN	Repariert die Struktur einer vorhandenen Access-Datenbank.
DATEI	BEENDEN	Beendet den Visual Data Manager.
DIENSTPROGRAMM	ABFRAGEGENERATOR	Startet den SQL-Abfragegenerator, mit dem sich SQL-Kommandos durch Auswahl der Elemente erstellen lassen.
DIENSTPROGRAMM	DATENFORMULAR-DESIGNER	Startet den Datenformular-Designer, der ein Visual-Basic-Formular anlegt, auf dem Steuerelemente für alle in einer Tabelle oder Abfrage enthaltene Felder angeordnet werden.
DIENSTPROGRAMM	GLOBAL ERSETZEN	Ersetzt den Wert eines Feldes in einer der vorhandenen Tabellen durch einen anderen Wert, wobei der zu ersetzende Werte durch ein Kriterium eingegrenzt werden kann.

Tabelle 3.1: Die Menükommandos des Visual Data Managers und ihre Bedeutung

Tabelle 3.1: Die Menükommandos des Visual Data Managers und ihre Bedeutung (Fortsetzung)

Menü	Kommando	Bedeutung
DIENSTPROGRAMM	EINGEBUNDENE TABELLEN	Ermöglicht das Einbinden (im engl. »attachment«) einer Tabelle aus einer beliebigen Datenbank an die aktuelle Datenbank. Damit lassen sich Zugriffe auf ODBC-Datenbanken in der Regel am schnellsten durchführen.
DIENSTPROGRAMM	GRUPPEN/ BENUTZER...	Ermöglicht den Zugriff auf die in einer Sicherheitsdatei enthaltenen Gruppen und Benutzer.
DIENSTPROGRAMM	SYSTEM.MD?	Lädt eine Access-Sicherheitsdatei.
DIENSTPROGRAMM	VOREINSTELLUNGEN	Erlaubt das Verändern gewisser allgemeiner Voreinstellungen (z.B. daß die zuletzt geöffnete Datenbank beim Starten automatisch geöffnet wird).

3.1.4 Die Symbolleiste des Visual Data Managers

In diesem Buch sollen Ihnen keine Fakten vorenthalten werden, daher lernen Sie in diesem Abschnitt auch die Symbole der kleinen Symbolleiste kennen. Die Tabelle 3.2 faßt die insgesamt neun Symbole zusammen, die Sie am Anfang zwar ignorieren können, deren Bedeutung Sie aber kennen sollten.

Tabelle 3.2: Die Symbole der Symbolleiste

Symbol	Bedeutung
	Legt fest, daß die geöffneten Recordsets vom Typ »Table« sind.
	Legt fest, daß die geöffneten Recordsets vom Typ »Dynaset« sind.
	Legt fest, daß die geöffneten Recordsets vom Typ »Snapshot« sind.
	Legt fest, daß auf einem vom Formularassistenten erstellten Formular das Daten-Steuerelement verwendet wird.
	Legt fest, daß auf einem vom Formularassistenten erstellten Formular das Daten-Steuerelement nicht verwendet wird.
	Legt fest, daß auf einem vom Formularassistenten erstellten Formular das DBGrid-Steuerelement verwendet wird. Außerdem werden die Ergebnisse von SQL-Abfragen in einer Tabelle dargestellt, was vor allem für das Auprobieren von SQL-Kommandos praktisch ist.
	Bricht eine Transaktion wieder ab (Rollback).
	Bestätigt eine Transaktion (CommitTrans).

Begriffe wie *Dynaset* oder *Snapshot* spielen im Zusammenhang mit den ADO-Objekten keine Rolle. Sie werden in Anhang C kurz erklärt, wenn es um die DAO-Objekte geht.

3.2 Das Anlegen einer neuen Datenbank

Der VDM ist ein kleines und schlichtes Programm, das ohne Schnörkel und besondere Hilfestellungen auskommt. Am besten, Sie starten den VDM und legen gleich los. Zuvor soll aber in aller Kürze die Datenbank vorgestellt werden, die in diesem und den folgenden Abschnitten Schritt für Schritt umgesetzt werden soll.

Hier ist das Szenario, das bereits in Kapitel 2 begonnen wurde und sich wie ein roter Faden durch die restlichen Kapitel des Buches ziehen wird. Stellen Sie sich vor, Sie sind der Verwalter eines Fuhrparks bei einem mittelgroßen Unternehmen und möchten die Wagen des Fuhrparks mit einer kleinen Datenbank verwalten. Im Moment soll es lediglich um die wichtigsten Aufgaben, wie das Erfassen des Fahrzeugbestandes und von Entleihvorgängen, gehen, wobei neben den Fahrzeugdaten auch die Modelldaten der Fahrzeuge gespeichert werden sollen. In den nächsten Ausbaustufen (die in diesem Buch allerdings nicht besprochen werden), könnte man z.B. verschiedene Wagenkategorien einführen (etwa Nutzfahrzeuge oder Luxuswagen) oder ein Abrechnungssystem hinzufügen. Das sind jedoch alles »Variationen« des Kernthemas.

Da am Anfang nur die allerwichtigsten Daten der einzelnen Fahrzeuge (Erstzulassung, Km-Stand, Farbe usw.) sowie der Modelle (Modellname, PS, Hubraum usw.) erfaßt werden, käme man auch mit einem kleinen Karteikasten prima aus, doch man soll bekanntlich mit der Zeit gehen, und schließlich möchten Sie in diesem Buch etwas über Datenbanken und deren Programmierung mit Visual Basic lernen[1].

1 Der überaus wichtige Grundsatz »Das richtige Werkzeug für den richtigen Zweck« soll damit zwar nicht außer Kraft gesetzt werden, doch eine Zettelkastenlösung wäre langfristig nicht sehr befriedigend.

3.2.1 Der Aufbau der Datenbank

Damit es am Anfang übersichtlich bleibt, wird die Datenbank klein gehalten, denn der Datenbestand soll in den folgenden Kapiteln lediglich dazu dienen, elementare Datenbankoperationen und einfache SQL-Kommandos ausführen zu können. In der ersten Ausbaustufe soll die Datenbank *Fuhrpark.mdb* daher lediglich zwei Tabellen enthalten:

- Die Tabelle *Modelldaten*
- Die Tabelle *Fahrzeugdaten*

In Kapitel 9 kommen mit den Tabellen *Ausleihdaten und Mitarbeiterdaten* zwei weitere Tabellen hinzu, durch die das »Ausleihen« von Fahrzeugen möglich wird. In diesem Kapitel werden auch die für das Erfassen von Datensätzen benötigten Formulare vorgestellt. Die Aufteilung der Daten auf mehrere Tabellen ist nicht zwingend erforderlich. Doch wie es Kapitel 2 hoffentlich anschaulich dargestellt hat, bringt dies eine Reihe von Vorteilen (oder, um es andersherum zu formulieren, es vermeidet eine Reihe von Nachteilen).

3.2.2 Der Aufbau der Tabelle »Modelldaten«

In der Tabelle *Modelldaten* werden die technischen Daten der einzelnen Modelle gespeichert. Beachten Sie, daß es sich hier lediglich um Modelle, nicht um die tatsächlich vorhandenen Fahrzeuge handelt. Doch um nicht später zu jedem vorhandenen Fahrzeug in der Tabelle *Fahrzeugdaten* auch die technischen Daten führen zu müssen (denken Sie an die in Kapitel 2 besprochene Notwendigkeit der Vermeidung von doppelt vorkommenden Feldern), werden diese Daten in einer separaten Tabelle gehalten. Dadurch kann es zwar passieren, daß es zu einem Fahrzeugmodell kein tatsächlich im Fuhrpark vorhandenes Fahrzeug gibt, doch ist dieser Fall nicht weiter tragisch. Nicht vorkommen darf dagegen, daß zu einem Fahrzeug in der Tabelle *Fahrzeugdaten* kein Eintrag in den Tabellen *Modelldaten* existiert.

Auch wenn jedes Fahrzeug ein hochtechnisches Gefährt ist, soll die Datenbank nur die wichtigsten Eckdaten erfassen:

- Eine Modellnummer
- Den (Modell-) Namen
- Den Namen des Herstellers
- Das Herkunftsland als Kürzel
- Das Herkunftsland als Flaggensymbol

- ✘ Die Leistung in PS (oder kW)
- ✘ Der Hubraum in ccm3
- ✘ Die Anzahl an Zylinder
- ✘ Die Geschwindigkeit in km/h
- ✘ Die Beschleunigung 0 auf 100 km/h
- ✘ Das Gewicht in kg

Es steht Ihnen natürlich frei, weitere Felder einzufügen. Jede Datenbank ist prinzipiell beliebig erweiterbar. Die Beschränkung auf relativ wenige Felder geschieht einzig und allein aus dem Grund, daß die ganze Angelegenheit in diesem Buch überschaubar bleibt.

Insgesamt werden also pro Modell elf Angaben gespeichert. Oder anders formuliert: Jeder Datensatz der Tabelle umfaßt elf Felder. Die einzelnen Felder der Tabelle *Modelldaten* sind damit festgelegt. Nun muß jedes Feld einen Datentyp erhalten. Diese Festlegung wird zwar erst bei der Umsetzung der Datenbank getroffen, doch ist es besser, sich bereits beim Entwurf der einzelnen Tabellen darüber Gedanken zu machen. Alle Datentypen beziehen sich übrigens auf die Jet-Engine (der »Access-Datenbank«). Würde man etwa eine Datenbank für den SQL-Server vorbereiten, wären andere Datentypen im Spiel, denn jedes DBMS besitzt seinen eigenen Satz von Datentypen.

Beginnen wir mit dem Feld *ModellNr.* Hier handelt es sich um ein Feld mit dem Datentyp *Text*, wobei vier Zeichen Umfang genügen (die Standardvorgabe ist 50 Zeichen). Die Modelle werden bei P1000 beginnend in »Einerschritten«, P1001, P1002 usw., durchnumeriert[1]. Da das Feld später als Primärschlüssel vereinbart wird, muß die Modellnummer stets eindeutig sein. Die Felder *ModellName* und *Hersteller* sind ebenfalls vom Typ *Text*, allerdings werden hier 50 Zeichen reserviert. Sollte ein Name sich später als länger erweisen, werden die überzähligen Zeichen abgeschnitten (der Visual Data Manager verweigert in diesem Fall schlicht und ergreifend die Eingabe). Auch das Feld *HerkunftslandKürzel* ist vom Typ *Text*, wobei hier diesmal drei Buchstaben (die international gültigen Landeskennzeichen) genügen. Auch wenn es nicht notwendig ist, bei kleinen Datenbanken »sparsam« mit dem Platz umzugehen, sollte sich die Feldgröße stets an dem tatsächlichen Bedarf orientieren. Man kann nicht immer ausschließen, daß

1 Dies ist eine rein willkürliche Festlegung. Das Präfix »P« steht übrigens für Personenfahrzeuge und soll die Möglichkeit offenhalten, später auch Nutzfahrzeuge in die Datenbank einfügen zu können und dabei eine einheitliche Numerierung beizubehalten.

eine Datenbank wächst, und bei 100000 Datensätzen können 40 eingesparte Byte pro Datensatz bereits etwas ausmachen.

Welchen Datentyp das Feld *HerkunftslandFlagge* besitzen soll, können Sie zu diesem Zeitpunkt vermutlich noch nicht erahnen. Da in diesem Feld ein Bitmap (also eine Aneinanderreihung von Bildpunkten gespeichert wird), könnte man erwarten, daß ein entsprechender Datentyp zur Auswahl steht. Doch wie Sie im weiteren Verlauf noch erfahren werden, gibt es in einer Access-Datenbank keinen solchen Feldtyp. Statt dessen wird der Universaldatentyp *Binary* verwendet, der immer dann zur Anwendung kommt, wenn Daten gespeichert werden sollen, zu denen keiner der anderen Felddatentypen paßt. Anzumerken ist in diesem Zusammenhang, daß dieser Datentyp innerhalb von Microsoft *OLE-Object* heißt.

Tabelle 3.3: Die Felder der Tabelle »Modelldaten« und ihre Datentypen

Feld	Datentyp	Besonderheit
ModellNr	*Text*	4 Zeichen
Modellname	*Text*	50 Zeichen
Hersteller	*Text*	50 Zeichen
HerkunftslandKürzel	*Text*	3 Zeichen
HerkunftslandFlagge	*Binary*	Wird erst in Kapitel 10 benutzt.
Leistung	*Integer*	Wert kann zwischen –32768 und +32767 liegen.
Hubraum	*Integer*	Wert kann zwischen –32768 und +32767 liegen.
Zylinder	*Byte*	Wert kann zwischen 0 und 255 liegen.
Geschwindigkeit	*Integer*	Wert kann zwischen –32768 und +32767 liegen.
Beschleunigung100	*Single*	Es sollen auch Nachkommastellen erlaubt sein.
Gewicht	*Integer*	Wert kann zwischen –32768 und +32767 liegen.

Zur Darstellung von Bitmaps, die in Microsoft Access gespeichert wurden, auf einem Visual-Basic-Formular muß anstelle des Bildfeldes entweder das OLE-Steuerelement (das allerdings nicht ADO-fähig ist) verwendet oder ein wenig zusätzliche Programmierarbeit betrieben werden. Mehr dazu in Kapitel 10.

Bei der Festlegung des Feldes *Leistung*, das (entgegen aller Normen, aber Kraft der Gewohnheit) eine Zahl mit der Einheit PS erhalten wird, ist die Sache wieder einfach. Da PS- bzw. kW-Angaben nur ganze Zahlen sind, die zudem nicht größer als 32767 werden (bereits beim Entwurf der Datenbank sollten »vernünftige« Annahmen zugrunde gelegt werden), kann hier der Datentyp *Integer* zum Einsatz kommen. Dieser Datentyp paßt auch für die Felder *Hubraum*, *Zylinder, Geschwindigkeit* und *Gewicht*, wenngleich für das Feld *Zylinder* der Datentyp *Byte* verwendet wird, da hier der Zahlenbereich zwischen 0 und 255 vollkommen ausreichend ist. Lediglich die Beschleunigung von 0 auf 100 km/h (Feld *Beschleunigung100*) kann »krumme« Werte annehmen, hier ist daher der Datentyp *Single* angebrachter.

Tabelle 3.3 faßt die eben getroffene Zuordnung (hoffentlich) übersichtlich zusammen. Sie sollten sich in diesem Zusammenhang lediglich merken, daß jedes Feld einer Tabelle zwei Merkmale besitzt: seinen Namen und seinen Datentyp. Letztere Angabe ist deswegen so wichtig, weil der Datentyp direkt den Platzbedarf eines Feldes in der Tabelle festlegt. Ein *Single*-Wert belegt 4 Byte, ein *Integer*-Wert 2 Byte und ein *Text*-Wert soviele Bytes, wie Zeichen vorgesehen wurden. Eine Übersicht über die sehr wichtige Beziehung zwischen Datentyp und Feldgröße gibt Tabelle 3.4. Diese Tabelle gibt gleichzeitig eine Übersicht über die bei der Jet-Engine zur Verfügung stehenden Datentypen.

Die Jet-Engine arbeitet mit einem eigenen Satz an Datentypen. Die weitestgehende Übereinstimmung mit den Datentypen von VBA ist »zufällig«. Nicht für alle VBA-Datentypen existiert ein Pendant bei der Jet-Engine (eine Ausnahme ist z.B. *Decimal*) und umgekehrt (VBA kennt z.B. keinen *Hyperlink*-Datentyp oder kein Pendant für die Replikations-ID, diese müssen in VBA durch die vorhandenen Datentypen »nachgebildet« werden). Neben den Jet-Datentypen gibt es noch die SQL-Datentypen, die eigene Namen und eigene Größen besitzen. Mit den SQL-Datentypen, die durch den SQL-Standard bestimmt werden, werden Sie im Zusammenhang mit der Jet-Engine nicht konfrontiert.

Beachten Sie bitte, daß diese Angaben nur für die Access-Datenbanken (also die Mdb-Datenbanken) gelten. Bei einer anderen Datenbank, z.B. Microsoft SQL Server, kann der Felddatentyp *Integer* (sofern dieser Name dort überhaupt verwendet wird) auch 4 Byte umfassen. Standardisiert sind diese Angaben nicht.

Tabelle 3.4: Die Feldgröße der verschiedenen Datentypen der Jet-Engine

Felddatentyp	Bedeutung	Größe in Byte
Boolean	Umfaßt die Werte True und False	2
Byte	Werte von 0 bis 255	1
Integer	Werte von –32768 bis +32767	2
Long	Werte von –2.147.483.648 bis 2.147.483.647	3
Single	Werte von –3.402823E38 bis –1.401298E–45 für negative Zahlen, und von 1.401298E–45 bis 3.402823E38 für positive Zahlen.	4
Double	Werte von –1.79769313486231E308 bis –4.94065645841247E–324 für negative Zahlen, und von 1.79769313486231E308 bis 4.94065645841247E–324 für positive Zahlen.	8
Currency	Werte von –922.337.203.685.477,5808 bis 922.337.203.685.477,5807 mit vier festen Nachkommastellen.	8
DateTime	Datum/Zeitangabe von 1.1.100 bis 32.12.9999	8
Text	Abhängig von der max. Anzahl an Zeichen	0 bis 255
Binary	Abhängig vom Inhalt	bis zu 1 Gbyte
Memo	Abhängig vom Inhalt	wird durch Größe der Datenbank begrenzt

3.2.3 Der Aufbau der Tabelle Fahrzeugdaten

Die Tabelle *Modelldaten* enthält absichtlich nur die reinen Modelldaten, aber nicht die Daten der tatsächlich im Fuhrpark vorhandenen Fahrzeuge. Diese Daten werden in der Tabelle *Fahrzeugdaten* gespeichert. Bei dieser Tabelle kommt ein wichtiges Element ins Spiel, die Relation. Damit ein Datensatz der Tabelle *Fahrzeugdaten* nicht auch noch einmal die technischen (Modell-)Daten des Fahrzeugs enthalten muß, besitzt ein Datensatz ein Feld mit dem Namen *ModellNr*, dessen Inhalt auf einen Datensatz in der Tabelle *Modelldaten* verweist. Auf diese Weise wird zwischen der Tabelle *Fahrzeugdaten* und der Tabelle *Modelldaten* eine n:1-Beziehung hergestellt. Mit anderen Worten, 1 Datensatz in der Tabelle *Modelldaten* kann auf n Datensätze in der Tabelle *Fahrzeugdaten* verweisen, denn ein und dasselbe Modell kann beliebig oft als Fahrzeug vorhanden sein. Dies ist das erste konkrete Beispiel für eine Relation, das Sie in diesem Buch kennenlernen. Hergestellt wird diese Relation (wie es in Kapitel 2 bereits angesprochen wurde) einfach dadurch, daß das Feld *ModellNr* in der Tabelle *Fahrzeugda-*

ten Werte enthält, die Inhalten des entsprechenden Feldes *ModellNr* (die Namensübereinstimmung ist zufällig) in der Tabelle *Modelldaten* entspricht. Das Feld *ModellNr* in der Tabelle *Modelldaten* ist daher der Primärschlüssel, das Feld *ModellNr* in der Tabelle *Fahrzeugdaten* entsprechend der Fremdschlüssel. Eine besondere Kennzeichnung dieser Felder ist zwar nicht notwendig, doch da die Jet-Engine einen Primärschlüssel automatisch mit einem Index belegt, der den Zugriff deutlich beschleunigt, werden Primärschlüssel explizit vereinbart. Mehr dazu in Kapitel 3.8.1. Es ist eine notwendige Bedingung, daß die Werte von Primärschlüsseln einzigartig (engl. »unique«) sind.

Datenbankfeld	**Datentyp**	**Besonderheit**
FahrzeugNr	*Text*	4 Zeichen
ModellNr	*Text*	4 Zeichen. Legt einen Datensatz in der Tabelle *Modelldaten* fest.
AngeschafftAm	*DateTime*	Datum, wann das Fahrzeug angeschafft und damit in den Fahrzeugbestand übernommen wurde.
Kennzeichen	*Text*	10 Zeichen
Farbe	*Text*	–
Preis	*Currency*	–
Baujahr	*Long*	Ist zwar eine Datumsangabe, doch da es nur um Jahreszahlen geht, tut es auch der Datentyp *Long*.
Bild	*Binary*	Wird erst in Kapitel 10 verwendet.
Bemerkung	*Memo*	Hier können allgemeine Anmerkungen zum Fahrzeug eingetragen werden – wird für die Beispiele in diesem Buch allerdings nicht verwendet.

Tabelle 3.5: Der Aufbau der Tabelle »Fahrzeugdaten«

3.3 Das Anlegen von Tabellen in der Datenbank

Sie kennen jetzt den Aufbau der Datenbank *Fuhrpark.mdb* in ihrer ersten Ausbaustufe mit ihren beiden Tabellen *Modelldaten* und *Fahrzeugdaten* und den in den einzelnen Tabellen enthaltenen Feldern. In diesem Abschnitt geht es um die Umsetzung der Datenbank, die bislang nur auf dem Papier existiert.

Bevor es losgeht, noch eine wichtige Anmerkung. Auch wenn die folgende Schrittanleitung halbwegs narrensicher sein sollte, es kann immer passieren, daß eine Beschreibung doch mißverständlich oder nicht eindeutig ist, Sie

aus irgendeinem Grund abgelenkt werden oder schlicht und ergreifend einen Fehler machen. Daher an dieser Stelle bereits der Hinweis, daß Sie jederzeit Festlegungen in einer Access-Datenbank ändern oder rückgängig machen können. Nichts ist endgültig, bei der Datenbankprogammierung schon gar nicht. Das gilt auch später, wenn Sie »richtige« Datenbanken erstellen, die vielleicht mehrere hundert oder gar tausend Datensätze erfassen, und nun die Struktur der Datenbank an einigen Stellen ändern möchten. Den Daten macht das gar nichts aus. Sie dürfen natürlich nur keine Daten löschen, Änderungen an der Datenbankstruktur sind daher im allgemeinen kein Problem. Und sollten Sie bei der folgenden Schrittanleitung völlig durcheinander kommen, fangen Sie einfach wieder von vorne an (unter Umständen müssen Sie eine bereits angelegte Mdb-Datei im Explorer wie eine gewöhnliche Datei löschen).

Schritt 1: Starten des Visual Data Managers

Starten Sie Visual Basic und anschließend über das ADD-INS-Menü den Visual Data Manager.

Schritt 2: Anlegen einer neuen Datenbank

Wählen Sie aus dem DATEI-Menü den Eintrag NEU und nacheinander die Einträge MICROSOFT ACCESS und ACCESS 7.0. Damit sagen Sie dem VDM, daß Sie eine Access-Datenbank im Format »Access 95« (die Version 7.0) anlegen möchten. Der Umstand, daß hier keine Einträge für Access 97 (Version 8.0) und Access 2000 (Version 9.0) angeboten werden, liegt einfach daran, daß das alte Access 95-Format von den Nachfolgeversionen »verstanden« wird und es in Bezug auf das Format der eigentlichen Datenbank nach oben keine Unterschiede zwischen den Versionen gibt. Außerdem ist der VDM, das wurde bereits erwähnt, ein eher schlichtes Programm, das auf Feinheiten nur wenig Rücksicht nimmt.

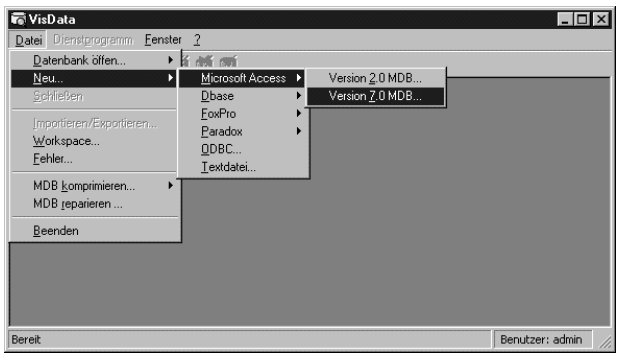


Bild 3.3: Nach dem Start des Visual Data Managers wird eine neue Datenbank angelegt

Der Visual Data Manager beschränkt Sie beim Anlegen einer neuen Datenbank nicht auf das Access-Format. Sie können alle Datenbankformate verwenden, für die ein ISAM- oder ODBC-Treiber vorhanden ist. Das bedeutet konkret, daß Sie neben dem Access-Format z.B. auch dBase-Datenbanken (ISAM-Format) oder sogar Microsoft SQL-Server-Datenbanken (ODBC-Datenbank) anlegen können. Diese Option ist allenfalls für jene Situationen interessant, in denen aus irgendeinem Grund das alte dBase-Format benötigt wird. Bei allen übrigen Datenbanktypen wird man jene Anwendungen verwenden, die mit dem Datenbanksystem ausgeliefert werden.

Schritt 3: Festlegen eines Datenbanknamens

Im nächsten Schritt möchte der VDM den Namen der Access-Datei wissen. Tun Sie ihm den Gefallen, und geben Sie als Namen »Fuhrpark« ein. Die Erweiterung *.Mdb* wird automatisch angehängt.

Schritt 4: Einfügen einer Tabelle

Seien Sie nicht enttäuscht, wenn Sie zu diesem Zeitpunkt noch nicht allzu viel sehen. Auch wenn Sie soeben eine neue Datenbank angelegt haben, gibt es noch nichts zu sehen, denn die Datenbank verfügt über keine einzige Tabelle. (Sie können lediglich auf das Pluszeichen vor dem Eintrag PROPERTIES klicken und sehen so die verschiedenen Eigenschaften der Datenbank und ihre aktuellen Einstellungen, die aber alle recht spezieller Natur sind und daher an dieser Stelle auch noch nicht erläutert werden[1].) Das soll im folgenden nachgeholt werden. Allerdings versteckt der VDM den dafür zuständigen Befehl. Anstatt ihn über das Menü oder die Symbolleiste anzubieten, müssen Sie eine leere Fläche im Datenbankfenster mit der rechten Maustaste anklicken und aus dem Kontextmenü den Eintrag NEUE TABELLE wählen (darauf muß man erst einmal kommen).

Nach Ausführung des Befehls öffnet sich ein auf den ersten Blick recht unübersichtliches Dialogfeld, in dem Sie im folgenden alle Daten der Tabelle eingeben sollen.

1 Ehrlich gesagt werden diese Angaben sogar im ganzen Buch nicht zusammenhängend erläutert, da es am Anfang kaum einen Grund gibt, sie zu ändern. Sie finden aber eine mehr als befriedigende Erklärung in der MSDN-Hilfe.

Schritt 5: Legen Sie den Tabellennamen fest

Geben Sie in das Eingabefeld »Tabellenname« den Namen der neuen Tabelle an. Geben Sie hier den Namen »Modelldaten« ein, klicken Sie aber noch nicht auf die *Schließen*-Schaltfläche.

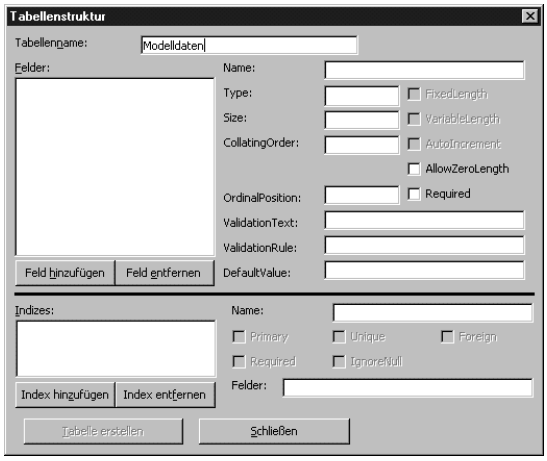


Bild 3.4: Als erstes wird im Dialogfeld Tabellenstruktur *der Name der neuen Tabelle festgelegt*

3.4 Das Anlegen von Feldern in einer Tabelle

Wie eine leere Datenbank stellt auch eine leere Tabelle lediglich eine Hülle dar, die zwar einen Rahmen zur Verfügung stellt, aber noch keine Daten aufnehmen kann. Damit eine Tabelle Daten speichern kann, muß sie über eines oder mehrere Felder verfügen. Im folgenden sollen Sie die einzelnen Felder gemäß Tabelle 3.3 erstellen, wobei Sie neben den Feldnamen auch den passenden Datentyp festlegen sollen.

Schritt 6: Anlegen des Feldes »ModellNr«

Nun wird die neue (und zu diesem Zeitpunkt noch völlig leere) Tabelle Schritt für Schritt um neue Felder ergänzt. Klicken Sie für jedes neue Feld auf die Schaltfläche *Feld hinzufügen*. Es öffnet sich ein weiteres Dialogfeld, in dem Sie nun alle Daten des Feldes eingeben sollen. Das Ganze ist einfacher, als es zunächst den Anschein haben mag, denn alle Felder benötigen nur eine einzige Angabe zwingend: den Feldnamen. Alle anderen Angaben sind optional und können jederzeit nachträglich wieder geändert werden. Geben Sie keinen Datentyp an, erhält das Feld den Datentyp Text mit einer

Begrenzung auf 50 Zeichen. Das ist zwar nicht immer optimal, doch als »Notlösung« in Ermangelung einer Alternative in Ordnung.

Geben Sie für den Feldnamen des ersten Feldes »ModellNr« ein. Das Einstellen eines Datentyps über die Auswahlliste *Type* ist nicht notwendig, denn der Datentyp *Text* soll beibehalten werden. Ändern Sie die vorgegebene Länge von 50 Zeichen im Feld *Size* auf 5 Zeichen.

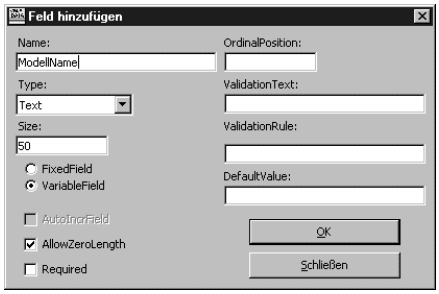


Bild 3.5: Im Dialogfeld Feld hinzufügen wird als erstes der Name des neuen Feldes festgelegt

Klicken Sie aber noch nicht auf die *Schließen*-Schaltfläche, denn es sollen noch weitere Felder zur Tabelle hinzugefügt werden. Klicken Sie statt dessen auf die *OK*-Schaltfläche. Sie werden feststellen, daß die Felder im Dialogfeld *Feld hinzufügen* lediglich gelöscht werden. Sollten Sie bereits auf *Schließen* geklickt haben, müssen Sie lediglich erneut auf die Schaltfläche *Feld hinzufügen* klicken, um weitere Felder hinzufügen oder auch löschen zu können. Das ist alles.

Schritt 7: Anlegen des Feldes »ModellName«

Jetzt ist das nächste Feld an der Reihe. Es wird auf die gleiche Weise angelegt. Auch *ModellName* besitzt den Typ *Text*, nur daß diesmal 50 Zeichen reserviert werden sollen und sich daher an der Vorgabe nichts ändert. Klikken Sie auf *OK*, um das nächste Feld festlegen zu können.

Kurzer Exkurs: Feldattribute

Das Dialogfeld des VDM, das zur Festlegung der Feldeigenschaften dient, offenbart, daß es noch eine Reihe weiterer Feldattribute gibt. Eine Übersicht gibt Tabelle 3.6. Einer besonderen Erwähnung bedarf der Feldtyp *AutoIncrField*. Diese Felder (sie besitzen den Datentyp *Long*) werden von der Jet-Engine beim Hinzufügen eines Datensatzes entweder automatisch um eins erhöht oder zufällig (aber stets eindeutig) vergeben, was in einigen Fällen recht praktisch sein kann, denn auf diese Weise übertragen Sie die Verwaltung des Primärschlüssels der Jet-Engine. Der Nachteil ist, daß mit dem Löschen eines Datensatzes oder einem »verunglückten« Aktualisierungsversuch die Zählwerte nicht angepaßt werden, so daß Lücken entstehen und es keine fortlaufende Numerierung gibt. Außerdem kann man den Zählfeldern keine Werte zuweisen, so daß ihr Einsatzbereich begrenzt ist. Ebenfalls mit Vorsicht zu genießen ist die Möglichkeit, eine Validierungsregel auf Datenbankebene festzulegen. Was sich zunächst praktisch anhören mag (und in einigen Fällen auch sehr praktisch ist), erweist sich beim Testen einer Da-

tenbankanwendung erst einmal als Ärgernis, da Sie beim Hinzufügen neuer Datensätze das betreffende Feld nicht einfach leer lassen können und eine Aktualisierung an der Validierungsregel scheitert1.

Tabelle 3.6: Die Attribute eines Datenbankfeldes bei der Jet-Engine in der Übersicht

Feldeigenschaft	Bedeutung
OrdinalPosition	Reihenfolge des Feldes innerhalb des Datensatzes.
ValidationText	Meldung, die angezeigt werden soll, wenn der zugewiesene Wert gegen die Validierungsregel des Feldes verstößt.
ValidationRule	Stellt eine Validierungsregel dar (z.B. <-1), die vor jedem Zuweisen eines Wertes an das Feld überprüft wird. Verstößt der Wert gegen die Regel, wird das Feld nicht aktualisiert. Statt dessen ist ein Laufzeitfehler die Folge.
DefaultValue	Ermöglicht die Festlegung eines Wertes, der immer dann eingesetzt wird, wenn das Feld vor dem Aktualisieren keinen Wert erhalten hat. Auf diese Weise läßt sich ohne großen Aufwand verhindern, daß ein Feld einen *NULL*-Wert erhält.
AllowZeroLength	Gibt an, ob ein Feld vom Typ *Text* auch keine Zeichen enthalten darf.
Required	Legt fest, daß das Feld einen Wert besitzen muß und nicht *NULL* sein darf.
AutoIncrField	Ein solches Feld wird von der Jet-Engine direkt verwaltet. Es erhält für jeden neuen Datensatz entweder einen um eins höheren Wert oder einen Zufallswert (letzteres läßt sich aber nicht beim VDM einstellen).

Die einzelnen Attribute stehen (selbstverständlich) während der Programmausführung über die Eigenschaften eines (ADO-/DAO-)*Field*-Objekts zur Verfügung und können daher auch programmgesteuert gesetzt werden.

Schritt 8: Anlegen des Feldes »HerkunftslandKürzel«

Das nächste Feld trägt den Namen »HerkunftslandKürzel«, besitzt den Typ *Text* und soll nur drei Zeichen umfassen. Geben Sie daher in das Feld Size die Zahl 3 ein. Klicken Sie auf *OK*, um das nächste Feld festlegen zu können.

1 Besonders ärgerlich ist es natürlich, wenn man die Validierungsregel vergißt und sich wundert, warum das Datenbankprogramm »spinnt«.

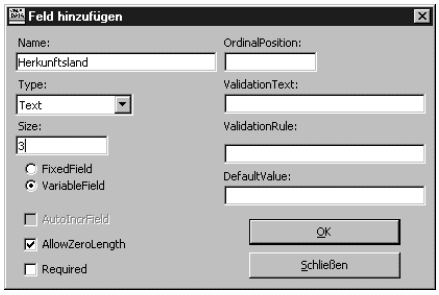


Bild 3.6: Im Dialogfeld Feld hinzufügen werden die Daten des Feldes »HerkunftslandKürzel« festgelegt

Schritt 9: Anlegen des Feldes »HerkunftslandFlagge«

Jetzt gibt es ein wenig Abwechslung, denn das nächste Feld heißt »HerkunftslandFlagge« und besitzt den Typ *Binary*. Hier sollen später einmal die Flaggensymbole (am besten aus dem Unterverzeichnis *\Graphics\Icons\Flags*) gespeichert werden. Klicken Sie auf OK, um das nächste Feld festlegen zu können.

Schritt 10: Anlegen des Feldes »Leistung«

Nun ist das Feld »Leistung« an der Reihe. Inzwischen wissen Sie, wie es geht, und müssen nur noch wissen, daß dieses Feld den Typ *Integer* besitzt. Dadurch wird indirekt die Größe des Feldes bestimmt. (Sie erkennen das daran, daß in dem Feld Size keine Eingaben möglich sind.) Klicken Sie auf *OK*, um das nächste Feld festlegen zu können.

Schritt 11: Anlegen des Feldes »Hubraum«

Bei diesem Feld lernen Sie nichts Neues mehr hinzu. Es besitzt ebenfalls den Typ *Integer*. Klicken Sie auf *OK*, um das nächste Feld festlegen zu können.

Schritt 12: Anlegen des Feldes »Zylinder«

Dieses Feld besitzt den Typ *Byte* (also Zahlen zwischen 0 und 255), wenngleich auch der Typ *Integer* in Frage gekommen wäre. Er belegt zwar 1 Byte mehr, doch wird diese »Platzverschwendung« kaum ins Gewicht fallen. Klicken Sie auf *OK*, um das nächste Feld festlegen zu können.

Schritt 13: Anlegen des Feldes »Geschwindigkeit«

Dieses Feld besitzt wieder den Typ *Integer*, da die Geschwindigkeiten im allgemeinen zwischen 100 und 300 km/h liegen können und der Datentyp *Byte* nicht ausreichend wäre1. Klicken Sie auf *OK*, um das nächste Feld festlegen zu können.

Schritt 14: Anlegen des Feldes »Beschleunigung100«

Da die Beschleunigung von 0 auf 100 km/h ein Wert ist, der in Sekunden gemessen wird und meistens auch eine Nachkommastelle besitzt, wird für das Feld *Beschleunigung100* der Typ *Single* gewählt. Klicken Sie auf *OK*, um das nächste Feld festlegen zu können.

Schritt 15: Anlegen des Feldes »Gewicht«

Da das Gewicht nur ganze Zahlen annehmen kann, besitzt es den Typ Integer. Klicken Sie auf OK, um das nächste Feld festlegen zu können.

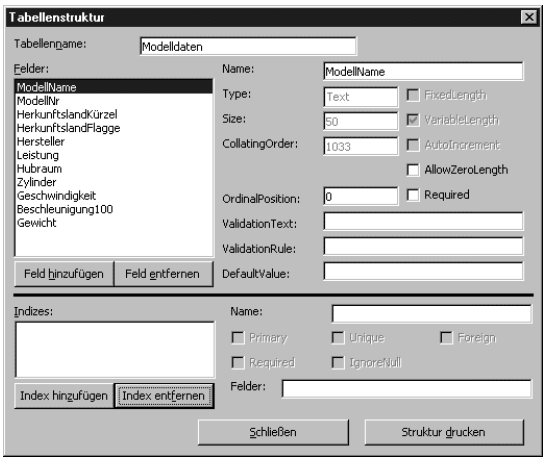


Bild 3.7: Die Tabellenstruktur der Tabelle »Modelldaten« ist endlich fertig

Schritt 16: Fertigstellen der Tabelle

Nun kommt der große Augenblick, denn alle Felder sind beieinander, und die Tabelle kann endlich erstellt werden. Klicken Sie dazu zunächst im Dialogfeld *Feld hinzufügen* auf die Schaltfläche *Schließen* und im Dialogfeld *Tabellenstruktur* auf die Schaltfläche *Tabelle erstellen*.

---

1 Zwar gibt es eine herstellerübergreifende Einigung auf eine Höchstgeschwindigkeit von 250 km/h, doch soll dieser Aspekt an dieser Stelle keine Rolle spielen.

Klicken Sie versehentlich auf die Schaltfläche Schließen, müssen Sie unbedingt die Frage »Schließen ohne zu speichern« mit »Nein« beantworten, da ansonsten alle Eingaben verloren sind (der VDM kann richtig grausam sein). Das gleiche gilt für das Drücken der Esc-Taste. Wenn Sie Pech haben und zu schnell auf »Nein« klicken, müssen Sie die komplette Tabelle noch einmal anlegen.

Wie bereits erwähnt, können Sie jederzeit Änderungen an der Tabellenstruktur vornehmen. Im VDM ist es aber (anders als z.B. in Microsoft Access) nicht möglich, die Eigenschaften eines Feldes zu ändern. Beim VDM müssen Sie das Feld erst löschen und anschließend neu anlegen. Das ist ein wenig umständlich, im allgemeinen aber kein Nachteil, da man große Datenbanken ohnehin nicht mit dem VDM anlegen wird1.

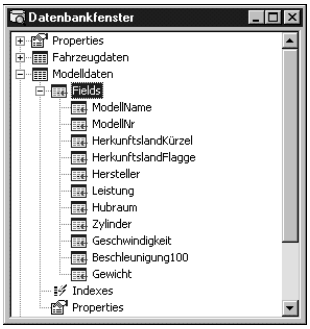


Bild 3.8: Das Datenbankfenster zeigt alle Felder der neuen Tabelle an

3.5 Die Eingab e von Daten

Die Tabellenstruktur der Tabelle *Modelldaten* steht, die Tabelle ist damit fertig, doch sie enthält noch keine Daten. Das soll im folgenden nachgeholt werden. Doch wo sollen die Daten herkommen? Die wenigsten Leserinnen und Leser dürften begeisterte Autokenner sein oder das große Lexikon der Personen- und Nutzfahrzeuge in Griffweite haben. Kein Problem, die Tabelle 3.7 enthält die Daten von ca. 30 Fahrzeugen, die im folgenden als Grundlage dienen soll. Machen Sie sich bitte einmal die Mühe, und tippen Sie alle Daten ein. Zwar müssen Sie nicht zum »Datentypisten« werden, doch ein wenig Fingertraining kann nicht schaden. Außerdem sind es nur wenige Daten, die sich in einigen Minuten eingeben lassen2.

1 Keine Angst, der Autor ist nicht prozentual an den Verkaufserlösen von Microsoft Access beteiligt (schön wär's). Aber wer richtig in die Datenbankprogrammierung einsteigen will, wird wahlweise um Visual Basic 6.0 Enterprise, Microsoft Access oder gleich den Microsoft SQL Server 7.0 nicht herumkommen.

2 Die Daten wurden übrigens rein willkürlich zusammengestellt und enthalten keinerlei Vorlieben oder Präferenzen des Autors bezüglich einzelner Modelle oder gar Hersteller.

3.5.1 Die Rolle der Stammdaten

Bei Datenbanken wird (allerdings nicht hochoffiziell) zwischen Stammdaten und Produktionsdaten unterschieden. Stammdaten sind jene Daten, die das »Fundament« der Datenbank bilden und daher von großer Wichtigkeit sind. Die Kundenadressen bei einem Versandhaus oder einer Versicherung sind typische Stammdaten. Produktionsdaten sind jene Daten, die während des täglichen Betriebs anfallen, wie z.B. Aufträge, Vorgänge und andere Dinge. Die bisher angelegten Tabellen *Modelldaten* und *Fahrzeugdaten* fallen in die Kategorie Stammdaten, die in Kapitel 9 anzulegende Tabelle *Ausleihdaten* eher in die Kategorie der Produktionsdaten, wenngleich es hier einen fließenden Übergang gibt. Der Grund, warum diese Unterscheidung getroffen wurde, ist, daß man in der Praxis die Produktionsdaten nicht auf die gleiche Weise erfaßt wie die Stammdaten. Letztere entsprechen 1:1 einer bereits vorliegenden Tabelle, wie dem Datenblatt des Herstellers. Bei der Eingabe der Ausleihdaten würden Sie jedoch ein kleines Problem feststellen: Sie erhalten den Namen eines Mitarbeiters, sollen aber dessen Mitarbeiternummer eingeben, da dies der Fremdschlüssel auf die Tabelle mit den Mitarbeiterdaten ist. Sie erhalten den Namen eines Modells, sollen aber die Modellnummer eingeben, da dies der Fremdschlüssel auf die Tabelle mit den Modelldaten ist. Zwar ist dies kein unlösbares Problem, denn man könnte sich die *Modelldaten*-Tabelle z.B. ausdrucken oder in einem zweiten Fenster öffnen, bei größeren Datenmengen oder mehreren (Fremd-)Schlüsselfeldern ist das nicht mehr praktikabel. In der Praxis werden daher die Produktionsdaten nicht direkt in die Datenbank eingegeben (auch wenn das möglich ist), sondern vielmehr mit Hilfe einer Eingabemaske erfaßt. Eine Eingabemaske ist in diesem Fall ein Formular, das für jedes Datensatzfeld ein entsprechendes Eingabefeld enthält und selbständig anstelle des Fremdschlüssels *ModellNr* den dazugehörigen Modellnamen anzeigt. Beim Erfassen eines neuen Datensatzes kann der Anwender das Modell bequem aus einem Listenfeld mit allen in der *Modelldaten*-Tabelle enthaltenen Modellnamen auswählen und muß keine mehr oder weniger kryptische Modellnummer eingeben. Wie sich solche Eingabemasken mit den Mitteln von Visual Basic erstellen lassen, erfahren Sie in Kapitel 9.

Modellname	Herkunfts-land	Leistung (PS)	Geschwin-digkeit	Zylinder	Hub-raum	Gewicht	Beschleuni-gung100
Cadillac Seville	USA	305	241	8	4565	1755	6.4
Ford Puma	D	125	203	4	1679	1033	9.2
Saab 9-5 3.0	S	200	235	6	2962	1545	8.0
Polo Variant 75	D	75	167	4	1598	1130	13.3
Mercedes Benz A 160	D	102	182	4	1598	1095	10.8
Smart	D	55	130	3	600	680	12.2
BMW Roadster Z3	D	321	250	6	3201	1425	5.4
Citroen Xsara	F	110	195	4	1761	1190	10.7
Volvo C 70 Coupé	S	193	230	5	2435	1480	7.8
Renault Kangoo	F	75	153	4	1390	1015	13.9
Plymouth Prowler	USA	243	190	6	3518	1350	7.2
Audi A6 2.8 Avant	D	193	234	6	2771	1510	8.4
Opel Astra 1.6	D	75	170	4	1598	1210	15.2
Opel Vectra Caravan 2.5 V6	D	170	222	6	2498	1475	9.9
Fiat Palio Weekend	I	165	100	4	1580	1050	13.0
Rover Freelander 1.8	GB	120	165	4	1796	1380	11.9
Volkswagen Golf 1.6	D	101	188	4	1595	1151	10.9
Seat Arosa	ESP	50	151	4	998	939	17.4
Mercedes Benz M 320	D	218	180	6	3199	1930	9.5
Honda Integra Type R	J	200	242	4	1797	1060	8.2
Volkswagen Passat 1.8	D	125	200	4	1781	1390	11.2
Audi S4 Avant	D	265	250	6	2671	1540	5.7
Nissan Primera Traveller	J	115	198	6	1998	1270	9.9
Madza 626	J	136	208	4	1991	1315	9.6
Alfa Romeo	I	155	216	4	1970	1250	8.6
Citroen Berlingo	F	90	160	4	1760	1240	14.6
Volvo V40 T4	S	200	235	4	1855	1410	7.3
Citroen Jumpy 1.6i	F	90	157	4	1592	1180	-
BMW 316i	D	115	190	4	1895	1250	11.9
Mercedes C180	D	125	193	4	1799	1350	12.0
BMW 318tds	D	90	175	4	1665	1290	13.9
BMW 323i	D	177	231	6	2494	1445	8.0

Tabelle 3.7: Die Musterdaten für die Tabelle »Modelldaten«

Schritt 17: Eingabe des ersten Datensatzes

Um eine Tabelle mit Daten zu füllen, klicken Sie die Tabelle im Datenbankfenster mit der rechten Maustaste an und wählen den Eintrag ÖFFNEN (ein Doppelklick tut es natürlich auch). Es öffnet sich ein neues Dialogfeld, in dem jedes Feld der Tabelle durch ein Eingabefeld dargestellt wird. Je nach Art des Datentyps werden Textfelder für die Eingabe von Zahlen und Texten und Kontrollkästchen für die Eingabe von Ja/Nein-Werten (Datentyp *Boolean*) angeboten. Lediglich für die Felder vom Typ *Binary* gibt es keine Eingabehilfe. Hier muß der VDM grundsätzlich passen (die Eingabe des Pfadnamens einer Bitmap-Datei bringt leider nicht den gewünschten Effekt). Sie werden in Kapitel 10 eine Möglichkeit kennenlernen, auch Bitmaps in einem Datenbankfeld zu speichern.

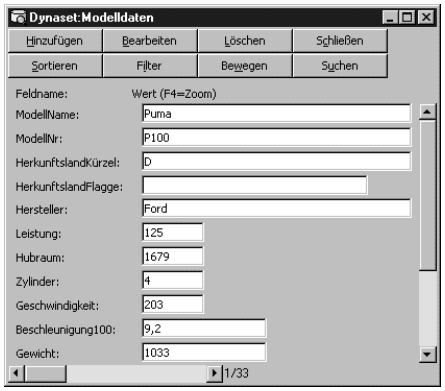


Bild 3.9: In diesem Dialogfeld wird der Inhalt eines Datensatzes der Tabelle »Modelldaten« angezeigt

Fürs erste müssen mir uns daher auf die übrigen Felder beschränken. Doch Vorsicht, eine direkte Eingabe in die Felder des Dialogfeldes ist noch nicht möglich. Da die Tabelle noch keinen Datensatz enthält, muß dieser zunächst über die Schaltfläche *Hinzufügen* hinzugefügt werden. Sie werden feststellen, daß sich das Dialogfeld nur geringfügig ändert. Wenn Sie genau hinschauen, werden Sie feststellen, daß der Name der linken oberen Schaltfläche nun *Aktualisieren* heißt. Diese Schaltfläche klicken Sie, nachdem Sie alle Daten in die einzelnen Felder (gemäß Tabelle 3.7) eingegeben haben (Sie können einen Datensatz beliebig oft aktualisieren). Anschließend werden die eingegebenen Daten in die Tabelle übernommen und damit in der Datenbank auf der Festplatte des PC gespeichert.

Ganz kurz eine Anmerkung zur Bildlaufleiste mit den zwei »Pfeilschaltflächen«, die Ihnen am unteren Rand des Dialogfeldes angeboten wird. Mit Hilfe dieser Schaltflächen, die Sie auch noch einmal im Zusammenhang mit dem ADO-Datensteuerelement in Kapitel 4 kennenlernen werden, können

Sie in einer Gruppe von Datensätzen vor und zurück gehen (der Fachausdruck lautet »Browsen«). Das Vor- und Zurückbewegen kommt erst dann in Frage, wenn die Tabelle über mehrere Datensätze verfügt.

3.6 Das Erstellen einer »Eingabemaske« für das Bearbeiten der Daten

Bislang spielte sich alles im Visual Data Manager ab, das soll sich nun ändern. Im folgenden wird ein Formular angelegt, mit dem Sie den Inhalt der Tabelle *Modelldaten* etwas komfortabler bearbeiten können. Sie müssen den Visual Data Manager noch nicht einmal verlassen, denn ein einfacher Formulardesigner ist fest eingebaut. Öffnen Sie das Menü DIENSTPROGRAMM, und wählen Sie den Eintrag DATENFORMULAR-DESIGNER. Es öffnet sich ein Dialogfeld, in dem Sie zwei Dinge eingeben müssen:

1. Den Namen des Formulars, das erstellt werden soll.
2. Den Namen der Tabelle oder Abfrage, deren Felder auf dem Formular erscheinen sollen.

Wählen Sie aus der Auswahlliste die Tabelle *Modelldaten* aus (hier wird zur Zeit nur ein Eintrag angeboten). Ging alles gut, erscheinen die in der Tabelle enthaltenen Felder in einer Auswahlliste. Klicken Sie auf das Feld mit dem Doppelrechtspfeil, um alle Felder in das Listenfeld *Aufgenommene Felder* zu übertragen. Wenn Sie möchten, können Sie über die Pfeile die Reihenfolge der Felder verändern, im Moment ist das allerdings nicht notwendig.

Bild 3.10: Der Datenformular-Designer ist bereit, ein Formular zu erstellen

Klicken Sie nun auf die Schaltfläche *Formular erstellen*, um das Formular zu erstellen. Der Assistent fängt an zu arbeiten, und nach einem kurzen Augenblick ist das Dialogfeld verschwunden. Was ist passiert? Nun, schalten

Sie einfach auf Visual Basic um. Im aktuellen Projekt sollte ein neues Formular enthalten sein, das dem Aufbau der Tabelle verblüffend ähnlich sieht. Wenn Sie dieses Formular zum Startformular machen und das Projekt über die F5-Taste starten, sollten Sie etwas später den Inhalt der Tabelle *Modelldaten* in einem Visual-Basic-Formular sehen.

So sieht Datenbankprogrammierung mit Visual Basic in der einfachsten Form aus. Dabei spielt es überhaupt keine Rolle, ob die Datenbank als Access-Datenbank auf der Festplatte Ihres PC vorliegt oder sich als SQL-Server-Datenbank in einem Netzwerk befindet. Das Prinzip des Zugriffs ist in beiden Fällen identisch.

3.7 Das Hinzufügen weiterer Tabellen

In diesem Abschnitt sollen Sie selbst aktiv werden. Ihre Aufgabe lautet: Legen Sie mit dem Visual Data Manager in der Datei *Fuhrpark.mdb* eine Tabelle mit dem Namen *Fahrzeugdaten* an. Der Aufbau der Tabelle (d.h. ihr Schema) ist in Tabelle 3.5 zu sehen, den (möglichen) Inhalt entnehmen Sie dagegen der Tabelle 3.8. Irgendwelche Besonderheiten gibt es nicht zu beachten, d.h., Sie können sich bei der Umsetzung an der im letzten Abschnitt vorgestellten Schrittfolge orientieren, wobei es Ihnen natürlich freisteht, eigene oder zusätzliche Datensätze hinzufügen.

Fahrzeug Nr	Modell Nr	Angeschafft Am	Farbe	Preis	Baujahr	Kennzeichen	Bild	Anmerkung
9001	P1001	1.1.99	Rot	18400	1998	VB-XX-123	–	–
9002	P1001	1.1.99	Gelb	38400	1997	VB-XX-124	–	–
9003	P1001	1.2.99	Weiß	19200	1998	VB-XX-123	–	–
9004	P1019	4.5.98	Silber	32100	1996	VB-XX-125	–	–
9005	P1012	7.8.82	Schwarz	19800	1994	VB-XX-126	–	–

Tabelle 3.8: Ein paar Musterdatensätze für die Tabelle »Fahrzeugdaten«

3.8 Die Datenbank wird erweitert

Die Datenbank *Fuhrpark.mdb* besteht bis zu diesem Zeitpunkt aus drei Tabellen und insgesamt (wenn Sie fleißig waren) ca. 30 bis 40 Datensätzen. Auch wenn eine solche Datenbank aufgrund ihrer bescheidenen Größe noch in die Kategorie »Mikrodatenbank« fällt, handelt es sich um eine »richtige« Access-Datenbank. Unabhängig von ihrer Größe wurden wichtige Arbeitsschritte beim Erstellen einer Datenbank noch ausgelassen:

1. Das Festlegen eines Primärschlüssels pro Tabelle.
2. Das Festlegen von Indizes.

Beim Visual Data Manager werden beide Schritte zusammengefaßt, das heißt, beim Hinzufügen eines Index kann angegeben werden, ob dieser die Rolle eines Primärschlüssels spielen soll. Bei Microsoft Access verhält es sich anders herum. Hier wird durch Anklicken eines Feldes in der Entwurfsansicht mit der rechten Maustaste und Auswahl des Eintrags PRIMÄRSCHLÜSSEL ein Index definiert.

3.8.1 Festlegen eines Primärschlüssels

Ein Primärschlüssel (engl. »primary key«) ist dazu da, einen einzelnen Datensatz in einer Tabelle eindeutig zu identifizieren. Ein Primärschlüssel setzt sich aus den Namen eines oder mehrerer Felder einer Tabelle zusammen. Schlüssel, weil so alle Felder genannt werden, die für einen gezielten Zugriff auf einen Datensatz verwendet werden. Primär, weil dies der erste und wichtigste Schlüssel ist (eine Tabelle kann durchaus mehrere Schlüssel besitzen, es können auch mehrere Felder zu einem Schlüssel kombiniert werden). Jedes Feld kann die Rolle des Primärschlüssels spielen, doch nicht jedes Feld ist dazu geeignet. Die wichtigste Forderung an den Primärschlüssel ist nämlich, daß der Inhalt des Feldes eindeutig ist. Ein Name kommt dagegen nur selten in Frage, da Namen mehrfach vorkommen können. Eindeutig ist dagegen die vollständige Personalausweisnummer (in den USA ist die Sozialversicherungsnummer der bekannteste Primärschlüssel) oder eine interne Nummer, die nur einmal vergeben wird. Der Primärschlüssel kann sich auch aus mehreren Feldern zusammensetzen. So wäre ein Feld Kontonummer nicht zwingend eindeutig, da es bei zwei Banken auch zwei identische Nummern geben kann. Wird zum Primärschlüssel dagegen die Bankleitzahl hinzugezogen, ergeben beide Felder eine stets eindeutige Kombination. Wird ein Feld mit dem Attribut »Unique« versehen, was sich indirekt durch die Festlegung als Primärschlüssel ergibt, achtet die Jet-Engine dar

auf, daß kein Wert doppelt vorkommt. Natürlich muß die Eindeutigkeit des Primärschlüssels nur im Kontext der Datenbank (bzw. der Tabelle) erfüllt sein. Wenn sich etwa eine Finanzsoftware nur auf ein Konto beschränkt, kommt die Kontonummer für den Primärschlüssel in Frage, auch wenn sie in einem globalen Kontext nicht eindeutig ist.

Jede Tabelle sollte einen Primärschlüssel besitzen. Ein Primärschlüssel sind ein oder mehrere Felder der Tabelle, die im Datenbanksystem besonders gekennzeichnet werden.

Bei der Jet-Engine muß ein Primärschlüssel nicht zwingend gekennzeichnet sein. Wenn Sie z.B. das Feld *ModellNr* in der Tabelle *Modelldaten* zum Primärschlüssel auserkoren haben, so muß dies nirgends hinterlegt werden. Das Feld wird zum Primärschlüssel, indem Sie dafür sorgen, daß keine zwei Felder den gleichen Inhalt besitzen und daß andere Tabellen Felder mit Modellnummern enthalten, die sich im Feld *ModellNr* der *Modelldaten*-Tabelle wiederfinden, so daß diese Felder automatisch zu Fremdschlüsseln werden. Dennoch wird man in der Praxis auch bei der Jet-Engine einen Primärschlüssel besonders kennzeichnen und zwar aus zwei Gründen:

1. Die Jet-Engine sorgt bei einem Primärschlüssel dafür, daß kein Feld den Wert eines anderen Feldes besitzen kann.
2. Die Jet-Engine legt einen Index auf das bzw. die Felder des Primärschlüssels, was eine Sortierung nach dem Primärschlüssel ergibt und generell den Zugriff beschleunigt.

Im folgenden soll mit Hilfe des Visual Data Managers für die Tabelle *Modelldaten* ein Primärschlüssel für das Feld *ModellNr* angelegt werden.

Schritt 18: Start des Visual Data Managers

Starten Sie Visual Basic, legen Sie ein Standard-Exe-Projekt an, starten Sie den Visual Data Manager über das ADD-INS-Menü, und öffnen Sie die Datenbank *Fuhrpark.mdb*.

Schritt 19: Öffnen der Tabellenstruktur

Selektieren Sie im Datenbankfenster die Tabelle *Modelldaten* mit der rechten Maustaste, und wählen Sie den Eintrag ENTWERFEN.

Schritt 20: Anlegen eines Index

Klicken Sie auf die Schaltfläche *Index hinzufügen* in der linken unteren Ecke. Es erscheint ein kleines Dialogfeld, in dem Sie drei Dinge einstellen können:

1. Den Namen des Index.
2. Die zu dem Index gehörenden Felder.
3. Bestimmte Attribute wie *Primary*, *Unique* und *IgnoreNulls*.

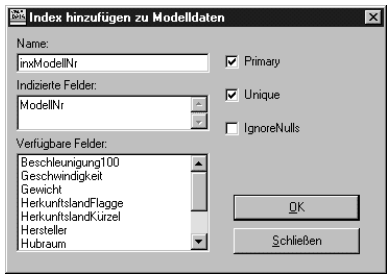


Bild 3.11: In diesem Dialogfeld wird ein neuer Index angelegt, der gleichzeitig die Rolle des Primärschlüssels spielen kann (aber nicht muß)

Schritt 21: Der Index erhält einen Namen

Geben Sie für den Namen des Index »inxModellNr« an. Der Präfix »inx« soll andeuten, daß es sich um einen Index handelt.

Schritt 22: Die Indexfelder werden ausgewählt

Wählen Sie aus der Auswahlliste der verfügbaren Felder das Feld *ModellNr* aus. Es erscheint dadurch in der Auswahlliste der indizierten Felder. Sollten Sie ein Feld zuviel ausgewählt haben, müssen Sie den Namen im Feld der indizierten Felder löschen.

Schritt 23: Das Indexfeld wird zum Primärschlüssel

Stellen Sie sicher, daß die Optionen *Primary* und *Unique* gesetzt sind (die Option *Primary* setzt immer auch die Option *Unique*, da ein Primärschlüssel einzigartig sein muß).

Klicken Sie auf *Schließen*, um das Dialogfeld zu schließen. Der neu angelegte Index, der gleichzeitig die Rolle des Primärschlüssels zeigt, wird mit seinen Attributen in der Auswahlliste *Indizes* angezeigt. Sie können jederzeit neue Indizes hinzufügen oder bestehende Indizes löschen.

Bild 3.12: Die Tabelle »Modelldaten« hat einen Index mit dem Namen »inxModellNr« erhalten

Tabelle 3.9: Die verschiedenen Eigenschaften eines Index

Index-Eigenschaft	Bedeutung
Primary	Der Index ist ein Primärschlüssel, dessen Wert in der Tabelle nur einmal vorkommen darf.
Unique	Der Index darf in der Tabelle nur einmal vorkommen. Diese Option wird automatisch aktiviert, wenn die Option *Primary* gewählt wurde.
IgnoreNulls	Das Feld darf keine NULL-Werte besitzen.

Vielleicht ist es Ihnen schon aufgefallen, daß Sie Microsoft Access vor dem Abspeichern einer neu angelegten Tabelle fragt, ob Sie einen Primärschlüssel erstellen möchten, sofern Sie dies unterlassen haben. Bestätigen Sie diese Frage mit *Ja*, sucht sich Microsoft Access nicht etwa ein passendes Feld aus, sondern fügt ein Zählfeld (mit dem Namen *ID*) zur Tabelle hinzu, das als Primärschlüssel (d.h. mit dem Attribut »indiziert ohne Duplikate«) vereinbart wird.

3.8.2 Sichtbarmachen einer Beziehung in Microsoft Access

Es ist recht lehrreich, sich die Beziehung zwischen zwei oder mehreren Tabellen anzuschauen. Leider geht das nicht mit dem Visual Data Manager, Sie benötigen dazu z.B. Microsoft Access. Starten Sie Microsoft Access, laden Sie die Datenbank *Fuhrpark.mdb*, und führen Sie den Menübefehl EXTRAS/BEZIEHUNGEN aus. Falls Sie noch keine Beziehungen betrachtet haben, gibt es noch kein Layout, und Sie müssen zunächst einmal die beiden Tabellen *Modelldaten* und *Fahrzeugdaten* auswählen und durch Anklicken der *Hinzufügen*-Schaltfläche zum Layout hinzufügen. Allerdings sind Sie noch nicht am Ziel, Microsoft Access zeigt keine Beziehung an. Sie werden lediglich feststellen, daß das Feld *ModellNr* in der Tabelle *Modelldaten* fett dargestellt wird, da es sich hier um einen Primärschlüssel handelt. Daß den-

noch keine Beziehung angezeigt wird, liegt daran, daß diese Beziehung zwar in den beiden Tabellen existiert, jedoch noch nicht in der Datenbank vermerkt wurde. Dies müssen Sie erst nachholen, indem Sie das Feld *ModellNr* aus der Tabelle *Modelldaten* auf das gleichnamige Feld in der Tabelle *Fahrzeugdaten* ziehen. Es öffnet sich ein Dialogfeld, in dem Sie genauere Angaben über die Art der Beziehung treffen können. Setzen Sie die Option *Referentielle Integrität*, auch wenn diese zum jetzigen Zeitpunkt noch nicht benötigt wird.

Bild 3.13: In diesem Dialogfeld wird die Art der Beziehung zwischen zwei Feldern festgelegt

Klicken Sie nun auf *Erstellen*, um die Beziehung der Datenbank zu »verewigen«. Sie werden feststellen, daß Microsoft Access jetzt eine 1:n-Beziehung zwischen der Tabelle *Modelldaten* und der Tabelle *Fahrzeugdaten* (das »n« wird bei Access 97 durch ein Unendlichzeichen dargestellt) erkennt.

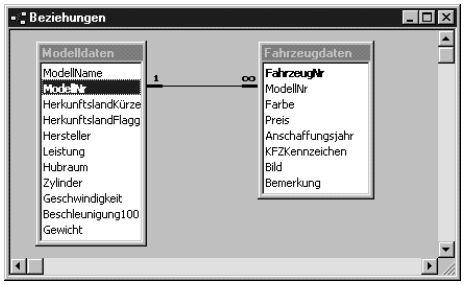


Bild 3.14: Zwischen beiden Feldern besteht eine 1:n-Beziehung

Das explizite Anlegen einer Beziehung bewirkt bei Microsoft Access, daß in der Datenbank ein *Relation*-Objekt (DAO) angelegt wird, dessen Eigenschaften die Attribute der Beziehung enthalten. Die Möglichkeit, *Relation*-Objekte explizit anzulegen, gibt es beim Visual Data Manager nicht.

Aus dem Beziehungsdiagramm entnehmen Sie, daß zwischen dem Feld *ModellNr* in der Tabelle *Modelldaten* und dem gleichnamigen Feld in der Tabelle *Fahrzeugdaten* eine 1:n-Beziehung besteht. Das Feld *ModellNr* in *Modelldaten* ist der Primärschlüssel, das Pendant in *Fahrzeugdaten* entsprechend der Fremdschlüssel. Fremdschlüssel können ohne weiteres mehrfach vorkommen. Würde jedes Feld *ModellNr* der Tabelle *Fahrzeugdaten* etwa den Wert *P1001* enthalten, würde das bedeuten, daß der gesamte Fuhrpark aus dem gleichen Fahrzeugtyp besteht.

3.9 Anlegen einer UDL-Datei

Der Umstand, daß eine Datenbank auf der Festplatte vorliegt, genügt für OLE DB noch nicht, um die Datenbank ansprechen zu können. Aus der Sicht von OLE DB gibt es keine Datenbanken, sondern lediglich Datenquellen, die über sogenannte Verbindungszeichenfolgen angesprochen werden. Aus der Verbindungszeichenfolge (engl. »connection string«) erfährt OLE DB u.a., wo sich die Datenquelle physikalisch befindet und über welchen OLE DB-Provider sie angesprochen wird. Die Verbindungszeichenfolge wird entweder unmittelbar vor dem Zugriff auf die Datenquelle angegeben, sie kann aber auch vorab »irgendwo« physikalisch gespeichert werden. Dieses »Irgendwo« kann die Registry, es kann aber auch eine speziell dafür vorgesehene Datei sein. Entscheidet man sich für die Datei, muß man eine sogenannte Verknüpfungsdatei oder UDL-Datei (für *Universal Data Link*) anlegen. UDL-Dateien tragen die Erweiterung *.Udl* und werden in der Regel im Unterverzeichnis *Data Links* (einem Unterverzeichnis des Unterverzeichnisses *Gemeinsame Dateien\System\OLE DB*) zusammengefaßt, wenngleich dies keine Voraussetzung ist. Im folgenden wird eine solche UDL-Datei für die frisch erstellte Access-Datenbank *Fuhrpark.mdb* angelegt. Das bringt den Vorteil, daß Sie später (z.B. in Kapitel 4) nur noch die UDL-Datei auswählen müssen und damit automatisch alle Informationen zur Verfügung stellen, die OLE DB für den Zugriff auf die Datenquelle benötigt. Indem Sie die UDL-Datei auf einen anderen PC kopieren, stehen dort ebenfalls die benötigten Zugriffsdaten zur Verfügung.

Die UDLs sind die Nachfolger der *Data Source Names* (DSN), in denen bei ODBC alle Verbindungsinformationen zusammengefaßt wurden und die ebenfalls wahlweise als Registry-Einträge oder als unabhängige Dateien vorliegen.

Schritt 1: Öffnen des Verzeichnisses

Öffnen Sie das Verzeichnis, in dem die UDL-Datei angelegt werden soll.

Schritt 2: Anlegen der UDL-Datei

Klicken Sie die Innenfläche des Verzeichnisses mit der rechten Maustaste an, und wählen Sie nacheinander die Einträge NEU und MICROSOFT DATENLINK.

Schritt 3: Benennen der UDL-Datei

Es erscheint ein neuer Dateieintrag mit dem Namen »Neu Microsoft Datenlink.udl«. Benennen Sie die Datei in *Fuhrpark.udl* um.

Schritt 4: Festlegen der Eigenschaften

Klicken Sie die UDL-Datei mit der rechten Maustaste an, und wählen Sie EIGENSCHAFTEN.

Schritt 5: Auswahl des OLE DB-Providers

Wechseln Sie auf die Registerkarte *Provider*, und wählen Sie dort den Eintrag »Microsoft Jet 3.51 OLE DB Provider«, da es sich um eine Access-Datenbank handelt.

Schritt 6: Auswahl der Datenquelle

Wechseln Sie auf die Registerkarte *Verbindung,* und tragen Sie im obersten Feld den kompletten Pfad von *Fuhrpark.mdb* ein.

Schritt 7: Fertigstellen der UDL

Klicken Sie auf *OK*, um das Dialogfeld wieder zu schließen. Sie haben damit eine funktionsfähige UDL-Datei angelegt, die Sie für künftige Zugriffe (per OLE DB) auf die Fuhrpark-Datenbank anstelle einer Verbindungszeichenfolge verwenden können (aber nicht müssen).

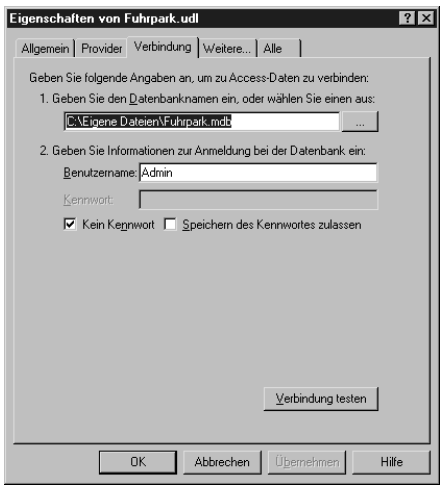


Bild 3.15: Auf der Registerkarte Verbindung *wird die Access-Datenbank für die UDL-Datei ausgewählt*

3.10 Zusammenfassung

In diesem Kapitel haben Sie mit Hilfe des Visual Data Managers aus Visual Basic Ihre (vermutlich) erste Datenbank erstellt. Am Anfang bestand sie lediglich aus einer Tabelle, im Rahmen einer Erweiterung wurde eine weitere Tabelle hinzugefügt. Mit Hilfe des Datenformularassistenten wurde ein Visual-Basic-Formular erstellt, das in der Lage ist, den Inhalt eines Datensatzes anzuzeigen und zu bearbeiten. Bislang fand alles praktisch unter Ausschluß der Programmierung statt. Das ist jedoch nicht Sinn der Sache. Der Visual Data Manager kann (und soll) zwar gewisse Routineaufgaben erleichtern, die Programmierung überflüssig machen soll er jedoch nicht.

Im nächsten Kapitel lernen Sie mit dem ADO-Datensteuerelement eine einfache Möglichkeit kennen, die Verbindung zu einer (beliebigen) Datenbank herzustellen und einzelne Felder mit Steuerelementen zu verbinden. Auch hier spielt die Programmierung noch keine Rolle. Das ändert sich erst in Kapitel 5, in dem die allgegenwärtigen ADO-Objekte vorgestellt werden. Unter deren Mitwirkung erhalten Visual-Basic-Programmierer die volle Kontrolle über die Datenbank.

3.11 Wie geht es weiter?

Die kleine Datenbank steht, sie enthält ca. 20 bis 50 Datensätze (je nach Fleiß und Ausdauer), sie besteht aus zwei Tabellen und kann wahlweise mit dem Visual Data Manager oder mit einem kleinen Visual-Basic-Programm bearbeitet werden. Für den Anfang ist das nicht schlecht. Richtige Datenbankprogrammierung bedeutet natürlich mehr. Richtig spannend wird es, wenn sich richtige Abfragen durchführen lassen. Bezogen auf die Tabelle mit den Modelldaten könnten solche Abfragen z.B. lauten »Zeige mir alle Modelle, die schneller als 160 km/h fahren und aus den USA kommen« oder »Welches ist das schnellste Fahrzeug mit 6 Zylinder?«. Dieses und vieles mehr läßt sich mit SQL, der universellen Datenbankabfragesprache, relativ einfach realisieren. Bis es soweit ist, müssen Sie in Kapitel 5 jedoch zuerst die ADO-Objekte kennenlernen, um z.B. das Ergebnis einer SQL-Abfrage einem *Recordset*-Objekt zuweisen zu können. Natürlich müssen Sie auch über SQL Bescheid wissen, das in Kapitel 7 vorgestellt wird. In Kapitel 4 lernen Sie zunächst mit dem ADO-Datensteuerelement eine relativ einfache Methode kennen, den Inhalt einer Tabelle auf einem Formular sichtbar zu machen.

3.12 Fragen

Frage 1:

Nennen Sie drei Anwendungen, mit denen sich Datenbanken erstellen lassen.

Frage 2:

Wo liegen die Stärken und die Schwächen des Visual Data Managers?

Frage 3:

Lassen sich mit dem Visual Data Manager auch ODBC-Datenbanken anlegen?

Frage 4:

Ist es möglich, eine mit dem Visual Data Manager angelegte Tabellenstruktur nachträglich zu ändern?

Frage 5:

Wie wird beim Visual Data Manager ein Primärschlüssel festgelegt?

Frage 6:

Die Datenbank *Fuhrpark.mdb* wurde mit dem Visual Data Manager erstellt. Außerdem wurden in der Tabelle *Modelldaten* das Feld *ModellNr* als Primärschlüssel und in der Tabelle *Fahrzeugdaten* das Feld *ModellNr* als Index (allerdings nicht als Primärschlüssel) festgelegt. Wird diese 1:n-Beziehung angezeigt, wenn die Datenbank mit Access 97 geöffnet wird?

Die Antworten zu den Fragen finden Sie in Anhang D.

Kapitel 4

Das ADO-Datensteuerelement und die gebundenen Steuerelemente

In diesem Kapitel lernen Sie mit dem ADO-Datensteuerelement die erste von (mindestens) drei Möglichkeiten kennen, den Inhalt eines Datenbankfeldes mit einem Steuerelement zu verbinden. Diese Bindung bewirkt, daß der Inhalt des Datenbankfeldes automatisch im Steuerelement angezeigt, also einer bestimmten Eigenschaft des Steuerelements, etwa der *Text*-Eigenschaft bei einem Textfeld, zugeordnet wird und sich Änderungen in dem Textfeld in der Datenbank auswirken. Die Bindung ist eine sehr praktische Angelegenheit, da Sie sich nicht um die »Abstimmung« zwischen einem Steuerelement und einem Datenbankfeld kümmern müssen, denn dies übernimmt das ADO-Datensteuerelement. Nach dem Durcharbeiten von Kapitel 5 werden Sie allerdings feststellen, daß das ADO-Datensteuerelement gegenüber den ADO-Objekten keine neue Funktionalität einführt, sondern lediglich das, was über die ADO-Objekte zur Verfügung gestellt wird, in einer etwas »handlicheren« Form verpackt. Aus diesem Grund werden die Eigenschaften, Methoden und Ereignisse des ADO-Datensteuerelements in diesem Kapitel nur kurz erwähnt. Eine ausführliche Vorstellung ist in Kapitel 5 an der Reihe, wenn es um die ADO-Objekte geht.

Sie erfahren in diesem Kapitel etwas über folgende Themen:

- ✘ Gebundene Steuerelemente
- ✘ Das Prinzip der »Bindung«
- ✘ Das ADO-Datensteuerelement mit seinen wichtigsten Methoden, Eigenschaften und Ereignissen

- ✘ Das Anzeigen einer Tabelle mit dem ADO-Datensteuerelement
- ✘ Die Steuerelemente DataList, DataCombo und DataGrid

4.1 Steuerelemente können gebunden sein

Die meisten Steuerelemente können an ein Datenbankfeld, genauer gesagt an das Feld eines *Recordset*-Objekts, gebunden werden. Diese Bindung (engl. »binding«) bedeutet, daß der Inhalt des Datenbankfeldes automatisch jener Eigenschaft zugewiesen wird, die für die Darstellung des Feldinhalts zuständig ist. Binden Sie etwa das Textfeld *txtModellname* an das Feld *Modellname* der Tabelle *Modelldaten*, wird der Inhalt des Feldes des aktuellen Datensatzes stets der *Text*-Eigenschaft des Textfeldes zugewiesen und damit angezeigt (die Zuordnung zwischen dem Feldinhalt und dem Textfeld kann nachträglich geändert werden). Aufgrund der bestehenden Bindung werden auch alle Änderungen, die Sie im Textfeld vornehmen, in die Datenbank übernommen, sobald der Datensatzzeiger seine Position ändert und ein anderer Datensatz zum aktuellen Datensatz wird. Das Textfeld wird damit zum Ein- und Ausgabefenster eines bestimmten Datenbankfeldes. Steuerelemente, die das können, werden als »gebundene Steuerelemente« (engl. »bound controls«) bezeichnet. Besonders praktisch ist das natürlich bei Gitternetz-Steuerelementen (den Grids), da hier nicht nur ein Feld, sondern gleich eine ganze Datensatzgruppe auf einmal angezeigt wird.

Auch bei in Visual Basic erstellten (ActiveX-)Steuerelementen können eine oder mehrere Eigenschaften »bindungsfähig« gemacht werden, so daß sich auch diese mit einer Datenbank verbinden können. Über die *Bindings*-Eigenschaft eines ActiveX-Steuerelements läßt sich jene Eigenschaft des Steuerelements auswählen, über die die Bindung hergestellt werden soll.

Allerdings sind nicht alle Steuerelemente automatisch gebunden. Ein gebundenes Steuerelement erkennen Sie daran, daß es über die Eigenschaften *DataSource*, *DataField* und (seit Visual Basic 6.0) *DataMember* und *DataFormat* verfügt. Bei Visual Basic sind die allermeisten Standardsteuerelemente, wie das Textfeld, das Bezeichnungsfeld oder das Bildfeld, gebunden. Nicht gebunden sind dagegen das Optionsfeld und der Zeitgeber, weil es hier keinen Sinn ergeben würde. Grundvoraussetzung ist, daß das Steuerelement eine Eigenschaft besitzt, deren Datentyp mit einem Datenbankdatentyp übereinstimmt. Denn ansonsten ließe sich ein Datenbankinhalt in dem Steuerelement nicht darstellen.

Ein Steuerelement muß nicht gebunden sein, damit es den Inhalt eines Datenbankfeldes anzeigen kann. Es erleichtert nur die ganze Angelegenheit, da Sie ansonsten die Zuweisung programmgesteuert übernehmen müssen, was gerade bei umfangreichen Datensatzgruppen Programmierarbeit bedeuten kann, die im Grunde unnötig ist. Es kann aber Situationen geben, in denen man von den mit einer automatischen Bindung einhergehenden Mechanismen, wie z.B. den verschiedenen ADO-Ereignissen, unabhängig sein möchte. In diesem Fall greift man direkt über ein *Recordset*-Objekt auf die Datenquelle zu und weist den Inhalt der einzelnen *Field*-Objekte den einzelnen Steuerelementen zu. Das war bis Visual Basic 5.0 eine gängige Praxis, da die Mechanismen der Datenbindung noch nicht so vielseitig waren und eine automatische Bindung zudem nur über das alte Datensteuerelement möglich war.

Steuerelement	**Gebundene Eigenschaft**
Anzeige	*Picture*
Bezeichnungsfeld	*Caption*
Bildfeld	*Picture*
Kombinationsfeld	*Text*
Kombinationsfeld	*List* (Liste wird nicht automatisch gefüllt)
Kontrollkästchen	*Value*
Listenfeld	*List* (Liste wird aber nicht automatisch gefüllt)
Textfeld	*Text*

Tabelle 4.1: Die gebundenen Steuerelemente in der Übersicht

4.1.1 Alte und neue gebundene Steuerelemente

Gebundene Steuerelemente gibt es bereits seit Visual Basic 3.0. Sie wurden damals von den Visual-Basic-Programmierern freudig begrüßt, versprachen sie doch den gleichen Komfort, der bei Microsoft Access schon immer selbstverständlich war. Allerdings, das wurde in diesem Buch bereits mehrfach angesprochen, hat es mit der Version 6.0 einen wichtigen Einschnitt gegeben. Die Datenbankschnittstelle DAO wurde gegen ADO ausgetauscht. Das bedeutet, daß auch die Fähigkeiten der Steuerelemente erweitert werden mußten. Die meisten Steuerelemente von Visual Basic ab Version 6.0 können sowohl an eine ADO- als auch an eine DAO-Datenquelle gebunden werden (eine Ausnahme sind jene Steuerelemente, die den Zusatz »OLE DB« im Namen tragen, sie können nur an ADO gebunden werden). Bei

älteren Steuerelementen, oder jenen, die mit Visual Basic 5.0 erstellt wurden, ist dies nicht der Fall. Diese lassen sich nur an eine DAO-Datenquelle binden. Und da ADO die Zukunft ist, können Sie mit diesen Steuerelementen bei der Datenbankprogrammierung nicht viel anfangen.

Doch wie erkenne ich, ob ein Steuerelement »ADO-tauglich« ist? Zum einen durch Ausprobieren (wenn es nicht geht, erhalten Sie eine Fehlermeldung), zum anderen an der Eigenschaft *DataMember*, die es nur bei den ADO-Steuerelementen gibt. Über die *DataMember*-Eigenschaft wird aus der über die *DataSource*-Eigenschaft festgelegten Datenquelle eine Datensatzgruppe ausgewählt.

4.1.2 Das Datensteuerelement als »Vermittler«

Nun ist es aber nicht so, daß ein Steuerelement über seine *DataSource*-Eigenschaft direkt an eine Access-Datenbank (oder an eine andere Datenquelle) gebunden werden kann. Diese Eigenschaft muß entweder den Namen eines ADO-Datensteuerelements, einer Datenumgebung oder direkt den Namen eines *Recordset*-Objekts erhalten, das wiederum mit einer OLE DB-Datenquelle verbunden ist. Erst dann ist es möglich, über die *DataField*-Eigenschaft den Namen eines Feldes in der Datensatzgruppe auszuwählen.

Und welche Rolle spielt die *DataMember*-Eigenschaft? Ganz einfach, sobald ein Steuerelement an eine Datenumgebung (mehr dazu in Kapitel 6) gebunden wird, kann es mehrere Datensatzgruppen enthalten, da eine Datenumgebung meistens mehrere *Command*-Objekte umfaßt. In diesem Fall wird die Datensatzgruppe über *DataMember* ausgewählt.

Tabelle 4.2: Diese Eigenschaften sind für die Bindung eines Steuerelements zuständig

Eigenschaft	Bedeutung
DataSource	Legt die Datenquelle fest. Hier kann der Name eines Datensteuerelements, einer Datenumgebung oder eines *Recordset*-Objekts zugewiesen werden.
DataField	Legt den Namen des Felds in der zugewiesenen Datensatzgruppe fest.
DataMember	Wurde die Datenquelle über eine Datenumgebung zugewiesen, muß hier der Name des *Command*-Objekts angegeben werden.

4.2 Das ADO-Datensteuerelement stellt sich vor

Nachdem Sie wissen, auf welche Weise der Inhalt einer Datenquelle in einem (gebundenen) Steuerelement dargestellt werden kann, sollen Sie (endlich) das ADO-Datensteuerelement kennenlernen. Es ist kein fester Bestandteil der Werkzeugsammlung, sondern muß erst über den Menübefehl PROJEKT/KOMPONENTEN und Auswahl des Eintrags »Microsoft ADO Data Control 6.0 (OLED6)« zu einem Projekt hinzugefügt werden. Anschließend erscheint es in der Werkzeugsammlung und kann auf dem Formular angeordnet werden. Auch wenn es üblicherweise im unteren Teil eines Formulars plaziert wird, gibt es auch Situationen, in denen es unsichtbar wird (z.B. dann, wenn es als zusätzliches Datensteuerelement auf einem Formular die Verbindung zu einer weiteren Tabelle oder Datenbank herstellt). Über die *Align*-Eigenschaft läßt sich erreichen, daß es sich automatisch an die Größe des Formulars anpaßt.

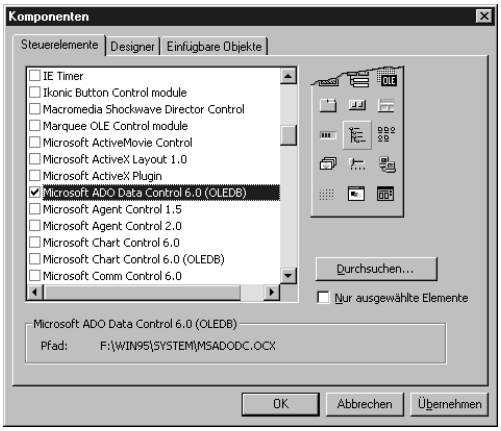


Bild 4.1: Das ADO-Datensteuerelement muß als ActiveX-Steuerelement zunächst zu einem Projekt hinzugefügt werden

Die Anbindung an eine Datenquelle erfolgt stets in mehreren Schritten:

✘ Auswahl eines OLE DB-Providers und damit verbunden einer Datenquelle über die *ConnectionString*-Eigenschaft. Sie müssen hier nichts eintippen, sondern klicken einfach auf die Schaltfläche mit den drei Punkten. Das öffnet ein Dialogfeld, aus dem Sie OLE DB-Provider und Datenquelle auswählen.

- ✘ Auswahl einer Tabelle, Abfrage oder eine SQL-Anweisung über die *RecordSource*-Eigenschaft. Dieser Schritt ist notwendig, um dem ADO-Datensteuerelement mitzuteilen, wo der Inhalt der Datensatzgruppe herkommt. Sobald über *ConnectionString* eine gültige Datenquelle ausgewählt wurde, erscheint im Eintrag *RecordSource* eine Auswahlliste mit allen zur Verfügung stehenden Tabellen, Abfragen, gespeicherten Prozeduren usw. Was hier erscheint, hängt natürlich von der Datenquelle ab.

Die Bezeichnung »ADO-Datensteuerelement« wird von Microsoft bewußt gewählt. Es gibt nämlich noch das alte Datensteuerelement (aus Visual Basic 3.0), das in die Werkzeugsammlung fest eingebaut ist. Dieses Datensteuerelement kann nur an DAO-Datenbanken gebunden werden und wird in diesem Buch auch nicht behandelt. Eine Verwechslungsgefahr besteht übrigens nicht, denn wenn Sie es mit einer ADO-Datenquelle zu verbinden versuchen, wird es ganz einfach nicht funktionieren.

Da es in diesem Buch ausschließlich um das ADO-Datensteuerelement geht, wird es im folgenden meistens nur als Datensteuerelement bezeichnet.

4.3 Das ADO-Datensteuerelement und die gebundenen Steuerelemente in der Praxis

Sie kennen die Aufgabe des Datensteuerelements und wissen, wie es zu einem Projekt hinzugefügt und mit einer Datenquelle verbunden wird. Jetzt soll das Datensteuerelement am Beispiel der in Kapitel 3 angelegten Fuhrpark-Datenbank in Aktion treten. Im folgenden wird ein Formular erstellt, mit dem Sie den Inhalt der Tabelle *Modelldaten* betrachten und bearbeiten können. Weitergehende Operationen, wie das Suchen nach einem Datensatz oder das Löschen eines Datensatzes, werden in den folgenden Kapiteln hinzugefügt. Allerdings nicht im Zusammenhang mit dem Datensteuerelement, sondern direkt mit den ADO-Objekten, was allerdings kein Widerspruch ist, da sich beide ergänzen.

Für die Umsetzung der folgenden Übungen benötigen Sie Visual Basic (ab Version 6.0) und die in Kapitel 3 vorgestellte Datenbank *Fuhrpark.mdb*, die mindestens die Tabelle *Modelldaten* enthalten sollte.

Schritt 1: Anlegen eines neuen Projekts

Starten Sie Visual Basic, legen Sie ein Standard-Exe-Projekt an, und fügen Sie über den Menübefehl PROJEKT/KOMPONENTEN und Auswahl des Eintrags »Microsoft ADO Data Control 6.0 (OLED6)« ein ADO-Datensteuerelement zur Werkzeugsammlung hinzu.

Schritt 2: Anordnen des ADO-Datensteuerelements auf dem Formular

Ordnen Sie das ADO-Datensteuerelement auf dem Formular an, und geben Sie ihm den Namen »adoModelldaten« (der genaue Name spielt keine Rolle, er sollte allerdings den Präfix »ado« besitzen).

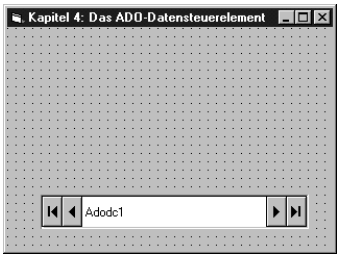


Bild 4.2: Das ADO-Datensteuerelement wurde auf dem Formular angeordnet

Auch wenn sich theoretisch beliebig viele ADO-Datensteuerelemente (auch in Kombination mit dem alten Datensteuerelement) auf einem Formular anordnen lassen, sollten es nicht zu viele sein, da ein einzelnes ADO-Datensteuerelement bezogen auf die aufrechterhaltenen Datenbankverbindungen relativ ressourcenintensiv ist (für das erste Steuerelement werden zwei, für jedes weitere Steuerelement wird eine Datenbankverbindung benötigt) und es nur wenige Anwendungen geben dürfte, in denen mehr als zwei Datensteuerelemente auf einem Formular erforderlich sind.

Schritt 3: Umbenennen des Formulars und des Projekts

Geben Sie dem Formular den Namen »frmADODatensteuerelement« und die Überschrift »Beispiel: Das ADO-Datensteuerelement«. Geben Sie dem Projekt den Namen »ADODatensteuerelement«, und speichern Sie das Projekt ab, damit Sie es später wieder laden können.

Schritt 4: Festlegen der *ConnectionString*-Eigenschaft

Wählen Sie beim ADO-Datensteuerelement die Eigenschaft *ConnectionString* aus, und klicken Sie auf die drei Punkte.

Schritt 5: Auswahl der OLE DB-Datenquelle

Das Dialogfeld bietet Ihnen drei Auswahlmöglichkeiten. Welche Sie davon wählen, hängt davon ab, ob bereits ein sogenanntes *Data Link* in Gestalt einer UDL-Datei angelegt wurde, Sie eine bereits vorhandene ODBC-Datenquelle verwenden oder anlegen möchten, oder ob (das dürfte am Anfang die Regel sein) Sie einen Verbindungsstring neu anlegen möchten. Wählen Sie für diese Übung die Option *Verbindungszeichenfolge verwenden*, und klicken Sie auf die *Erstellen*-Schaltfläche.

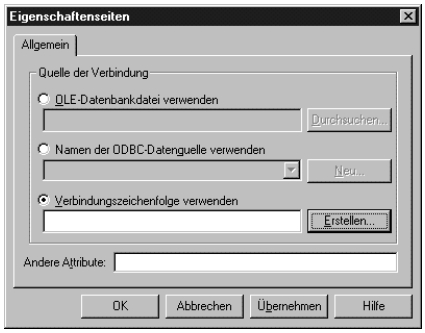


Bild 4.3: In diesem Dialogfeld wird die Verbindung zur Datenbank ausgewählt

Schritt 6: Auswahl des OLE DB-Providers für Access 97

Es erscheint das typische OLE DB-Auswahldialogfeld, in dem Sie a) einen OLE DB-Provider und b) eine Datenquelle auswählen müssen. Wählen Sie in der Registerkarte *Provider* als OLE DB-Provider »Microsoft Jet 3.51 OLE DB Provider« aus. Dies ist der OLE Provider für alle Microsoft-Access-97-Datenbanken.

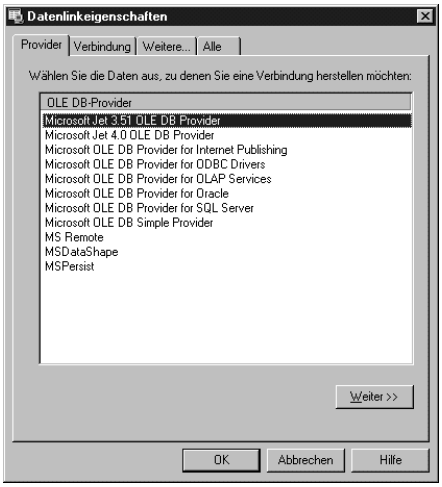


Bild 4.4: In diesem Dialogfeld wird der OLE DB-Provider ausgewählt

Schritt 7: Auswahl der Datenbank *Fuhrpark.mdb*

Schalten Sie auf die Registerkarte *Verbindung* um. Hier wird die Verbindung zwischen dem zuvor ausgewählten OLE DB-Provider und einer Datenquelle festgelegt. Da Sie im letzten Schritt den Access 97-Provider gewählt haben, paßt sich die Registerkarte entsprechend an und erlaubt Ihnen nun die Auswahl einer Mdb-Datei. Klicken Sie auf die Schaltfläche mit den drei Punkten, und wählen Sie die Datei *Fuhrpark.mdb* aus.

Sollten Sie diese Datei nicht besitzen, müssen Sie sie sich entweder wie in Kapitel 3 beschrieben anlegen oder von der Webseite des Buches herunterladen.

Bestätigen Sie Ihre Auswahl durch Anklicken der *OK*-Schaltfläche. Ein Testen der Verbindung durch Anklicken der Schaltfläche *Verbindung testen* ist nicht notwendig, kann aber sinnvoll sein, um festzustellen, ob die ausgewählte Datenquelle existiert (das ist vor allem bei Datenquellen praktisch, die über das Netzwerk angesprochen werden).

Schritt 8: Schließen des Dialogfeldes zur Auswahl einer OLE DB-Datenquelle

Das war es auch schon. Klicken Sie zweimal auf die *OK*-Schaltfläche, um die beiden Dialogfelder zu schließen. Sie haben damit (über OLE DB) das Datensteuerelement mit einer Datenquelle verbunden.

Schritt 9: Auswahl einer Tabelle

Nun müssen Sie das Datensteuerelement mit einer Tabelle, gespeicherten Abfrage usw. der Datenquelle verbinden. Wählen Sie dazu die *RecordSource*-Eigenschaft des Datensteuerelements aus. Haben Sie bislang alles richtig gemacht, sollte ein Dialogfeld mit einer Registerkarte mit dem Titel *RecordSource* erscheinen. Hier müssen Sie zwei Dinge einstellen: den Typ des anzusprechenden Objekts (z.B. Tabelle oder Ansicht) und den Namen des Objekts. Wählen Sie für Befehlstyp »2 – adCmdTable«. In der unteren Auswahlliste sollten kurz darauf die Namen aller in der Datenbank *Fuhrpark.mdb* enthaltenen Tabellen angezeigt werden[1]. Wählen Sie den Eintrag »Modelldaten«, und schließen Sie das Dialogfeld über die *OK*-Schaltfläche. Sie haben damit das ADO-Datensteuerelement mit der Tabelle *Modelldaten* verbunden.

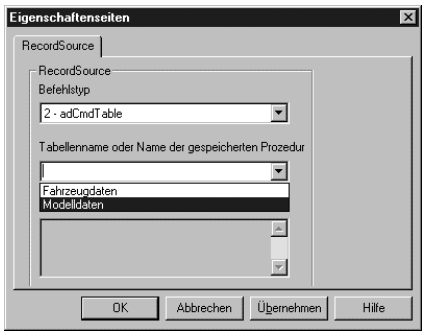


Bild 4.5: Auf der Registerkarte RecordSource *wird eine Tabelle aus der Fuhrpark-Datenbank ausgewählt*

Schritt 10: Anordnen eines gebundenen Textfeldes

Viel zu sehen gibt es noch nicht. Was noch fehlt, sind die (gebundenen) Steuerelemente. Ordnen Sie auf dem Formular ein Textfeld an, und geben Sie ihm den Namen »txtModellname«. Setzen Sie außerdem die *Locked*-Eigenschaft auf *True*, damit sein Inhalt später nicht versehentlich überschrieben wird (das ist keine Voraussetzung).

Schritt 11: Verbinden des Textfeldes mit dem Datensteuerelement

Jetzt wird die Verbindung zwischen dem Textfeld und dem Datensteuerelement hergestellt. Wählen Sie als erstes die *DataSource*-Eigenschaft des Textfeldes im Eigenschaftsfenster aus. Haben Sie bislang alles richtig ge-

1 Wenn dabei die Festplatte ein wenig ins Rotieren gerät, ist das kein Problem.

macht, sollte eine Auswahlliste erscheinen, in der der Eintrag »adoDaten« angeboten wird. Dies ist der Name des ADO-Datensteuerelements. Wählen Sie diesen Namen aus, um das Textfeld mit dem ADO-Datensteuerelement und der Datenquelle, die über dieses Datensteuerelement angesprochen wird, zu verbinden.

Schritt 12: Verbinden des Textfeldes mit einem Datenbankfeld

Nun muß noch die Verbindung zwischen dem Textfeld und dem Datenbankfeld *ModellName* hergestellt werden. Wählen Sie dazu im Eigenschaftsfenster die Eigenschaft *DataField* aus. Haben Sie bislang alles richtig gemacht, sollte eine Auswahlliste mit allen Feldnamen der Tabelle *Modelldaten* erscheinen. Wählen Sie in der Liste den Feldnamen *Modellname* aus. Sie haben damit das Textfeld mit dem Datenbankfeld verbunden.

Bild 4.6: In der Auswahlliste der »DataField«-Eigenschaft werden alle Feldnamen der Tabelle »Modelldaten« angezeigt

Sollte die Auswahlliste der *DataField*-Eigenschaft nicht erscheinen, müssen Sie noch einmal alle Schritte in Ruhe durchgehen.

Schritt 13: Das Programm wird getestet

Starten Sie das Programm, um das Datensteuerelement in Aktion zu erleben. Sie werden feststellen, daß die vier Funktionsschaltflächen aktiv geworden sind und Sie sich durch Anklicken der Schaltflächen in der Datensatzgruppe bewegen können.

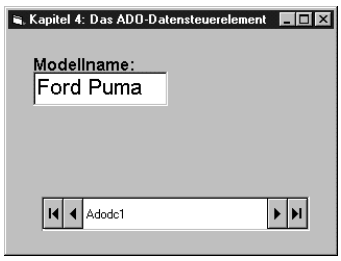


Bild 4.7: Der Inhalt des Datenbankfeldes »Modell-Name« wird im Textfeld angezeigt

Prinzipiell wäre es auch möglich, Änderungen an den Daten vorzunehmen, doch da die *Locked*-Eigenschaft des Textfeldes den Wert *True* erhalten hat, ist diese Option im Moment gesperrt. Diese Maßnahme soll verhindern, daß unbedachte Eingaben den Inhalt der Tabelle überschreiben. Wie sich in einer Datenbank kontrolliert Änderungen vornehmen lassen, erfahren Sie in Kapitel 5 in Zusammenhang mit den Ereignissen der ADO-Objekte.

Schritt 14: Hinzufügen weiterer Steuerelemente

Nach dem gleichen Schema können Sie nun für jedes weitere Datenbankfeld ein gebundenes Steuerelement hinzufügen. Welches Steuerelement Sie wählen, hängt vom Datentyp des Feldes ab (Tabelle 4.3 enthält eine entsprechende Gegenüberstellung). So wird man für Datenbankfelder mit einem Textinhalt meistens das Textfeld oder das Bezeichnungsfeld, für boolesche Daten (Ja/Nein) das Kontrollkästchen und für Bilder (also für Binärfelder) entweder die Anzeige oder das Bildfeld verwenden. Das OLE-Steuerelement, das die bei Microsoft Access im OLE-Format vorliegenden Bilder direkt anzeigen kann, funktioniert beim ADO-Datensteuerelement leider nicht. Passen Datentyp und Steuerelement nicht zusammen, kommt es nach dem Programmstart zwangsläufig zu einem Laufzeitfehler. Eine Unterscheidung zwischen Textdaten und numerischen Daten gibt es im allgemeinen nicht. Auch Datenbankfelder, die Zahlen beinhalten und z.B. vom Typ *Long* sind, werden in einem Textfeld angezeigt. Es sei denn, Sie hätten ein Steuerelement, das speziell für numerische Daten da ist. Vorausgesetzt, es ist (per OLE DB) gebunden, kann man es selbstverständlich als Alternative zum Textfeld verwenden.

Normalerweise wird bei der Anzeige eines Wertes dessen numerisches Format nicht berücksichtigt. Ein Geldbetrag wird in einem Textfeld nicht als solcher dargestellt, auch wenn das Datenbankfeld den Datentyp *Curreny* besitzt. Über die *DataFormat*-Eigenschaft eines Steuerelements läßt sich ein spezielles Darstellungsformat auswählen[1].

Noch ein kleiner Hinweis: Sollten Sie in die Tabelle *Modelldaten* keine Bilder eingefügt haben (was wahrscheinlich ist), werden auch keine angezeigt. In diesem Fall müssen Sie keine Anzeige auf dem Formular anordnen. Wie sich auch Bitmaps in eine Access-Datenbank einfügen lassen, wird in Kapitel 10 angesprochen.

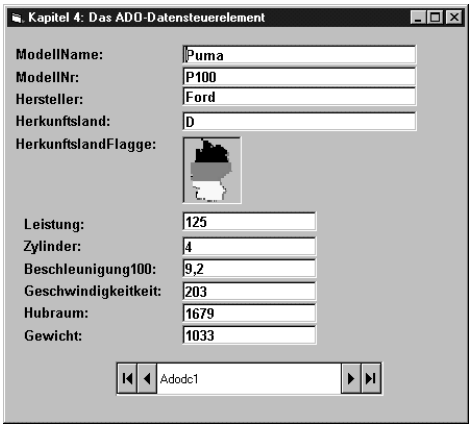


Bild 4.8: Das fertige Formular zeigt nun alle Felder der Tabelle »Modelldaten« an

Feld	**Passendes Steuerelement**
ModellName	Textfeld
ModellNr	Textfeld
Hersteller	Textfeld
HerkunftslandKürzel	Textfeld
HerkunftslandFlagge	Anzeige oder Bildfeld
Leistung	Textfeld

Tabelle 4.3: Die Felder der Tabelle »Modelldaten« und die passenden gebundenen Steuerelemente

1 Die Möglichkeit der nachträglichen Formatierung muß vom OLE DB-Provider unterstützt werden. Es funktioniert also nicht bei allen Datenquellen.

Tabelle 4.3: Die Felder der Tabelle »Modelldaten« und die passenden gebundenen Steuerelemente (Fortsetzung)

Feld	Passendes Steuerelement
Hubraum	Textfeld
Zylinder	Textfeld
Geschwindigkeit	Textfeld
Bild	Anzeige oder Bildfeld
ModellNr	Textfeld
Beschleunigung100	Textfeld
Gewicht	Textfeld

4.4 Mehr über das ADO-Datensteuerelement

Bislang haben Sie das ADO-Datensteuerelement lediglich als einfach zu handhabendes »Plug-In« kennengelernt. Es verfügt aber auch über Eigenschaften, Methoden und Ereignisse, die im folgenden kurz vorgestellt werden. Da die meisten Eigenschaften und Ereignisse aber zum *Recordset*-Objekt gehören, das durch ein ADO-Datensteuerelement repräsentiert wird, werden sie auf Kapitel 5 vertagt. Es sei aber an dieser Stelle bereits erwähnt, daß die *Recordset*-Eigenschaft des Datensteuerelements der Schlüssel ist, um die fehlende Funktionalität, wie z.B. das Löschen von oder die Suche nach Datensätzen zu ergänzen.

4.4.1 Die Eigenschaften des ADO-Datensteuerelements

Die meisten Eigenschaften des ADO-Steuerelements sind, wie bereits erwähnt, Eigenschaften des *Connection*- bzw. des *Recordset*-Objekts. Dazu gehören z.B. Eigenschaften wie *CursorLocation*, *ConnectionString*, *CommandType*, *LockType* oder *MaxRecords*. Die wichtigste Eigenschaft ist die *ConnectionString*-Eigenschaft, denn sie gibt an, aus welcher Datenquelle/Datenbank die Daten stammen. An dieser Eigenschaft wird sehr schön der grundsätzliche Unterschied zwischen den neuen ADOs und den alten DAOs deutlich. Während bei den DAOs die Datenbank über die *DatabaseName*-Eigenschaft ausgewählt wird, deren Bedeutung bereits der Name verrät, wird dieses im Grunde simple Prinzip bei den ADOs durch die *ConnectionString*-Eigenschaft und die Notwendigkeit, eine Verbindungszeichenfolge angeben zu müssen, »verklausuliert«. Dafür können Sie aber

auch beliebige Datenquellen auswählen, was bei den DAOs nur im Zusammenhang mit der *Connect*-Eigenschaft (und dann auch nur beschränkt auf ODBC) möglich ist.

Die »zweitwichtigste« Eigenschaft des Datensteuerelements ist die *Recordset*-Eigenschaft, über die wiederum alle Eigenschaften und Methoden des *Recordset*-Objekts zur Verfügung stehen. Und da wir gerade bei der Rangfolge sind. Die drittwichtigste Eigenschaft ist die *RecordSource*-Eigenschaft, die am Anfang gerne mit der *Recordset*-Eigenschaft verwechselt wird. Erstere dient dazu, entweder im Eigenschaftsfenster oder programmgesteuert ein *Recordset*-Objekt in der Datenquelle auszuwählen. Die *RecordSource*-Eigenschaft erhält entweder den Namen einer Tabelle oder Abfrage oder ein SQL-Kommando als Wert zugewiesen. Die *RecordSource*-Eigenschaft stellt damit die Bindung zwischen der über *ConnectionString* ausgewählten Datenquelle und der *DataField*-Eigenschaft der gebundenen Steuerelemente her.

Die wichtigsten »eigenen« Eigenschaften des ADO-Datensteuerelements sind: *Align* (legt die Ausrichtung des Datensteuerelements im Formular fest), *BOFAction*, *EOFAction* und *Orientation* (legt fest, ob das Datensteuerelement horizontal oder vertikal angezeigt wird). Die Eigenschaften *BOFAction* und *EOFAction* legen fest, wie das Datensteuerelement auf den Umstand reagiert, daß der Datensatzzeiger über den Beginn (*BOF=True*) bzw. das Ende (*EOF=True*) einer Datensatzgruppe hinaus bewegt wurde. Zur Auswahl steht bei der *BOF*-Eigenschaft (Anfang der Datensatzgruppe erreicht) das Ausführen einer *MoveFirst*-Methode, um den Datensatzzeiger auf den ersten Datensatz zu bewegen, oder lediglich das Setzen der *BOF*-Eigenschaft auf *True*. Bei der EOF-Eigenschaft gibt es drei Alternativen. Neben dem Ausführen einer *MoveLast*-Methode und dem Anzeigen eines »End Of File«-Zustands durch Setzen der gleichnamigen Eigenschaft steht als dritte Alternative die Ausführung einer *AddNew*-Methode zur Disposition, um einen neuen Datensatz an die Datensatzgruppe anzuhängen. Warum soll ich bei *EOF=True* ein *MoveLast* ausführen, der Datensatzzeiger steht doch schon beim letzten Datensatz? Nun, nicht ganz. Ein *EOF* signalisiert, daß der Datensatz bereits über den letzten Datensatz hinausbewegt wurde und damit nicht mehr gültig ist. Die *MoveLast*-Methode hebt diesen Zustand auf, indem sie den Datensatzzeiger wirklich auf den letzten Datensatz positioniert.

4.4.2 Die Methoden des ADO-Datensteuerelements

Diese Thema läßt sich schnell abhandeln, denn das ADO-Datensteuerelement besitzt, neben den Standardmethoden, keine eigenen Methoden. Und was ist z.B. mit der *Find*-Methode, die einen Datensatz im aktuellen Recordset ansteuert? Nun, das ist eine Methode des *Recordset*-Objekts, die folglich über die *Recordset*-Eigenschaft angesprochen wird:

```
adoData.Recordset.Find Criteria:="Name='Huhn'"
```

Mehr zur *Find*-Methode in Kapitel 8, in dem das sehr wichtige *Recordset*-Objekt im Mittelpunkt steht.

4.4.3 Die Ereignisse des ADO-Datensteuerelements

Ereignisse spielen (anders als bei den alten DAOs) bei den ADOs und damit auch beim ADO-Datensteuerelement eine wichtige Rolle. Sowohl das *Recordset*- als auch das *Connection*-Objekt können auf eine Reihe von Ereignissen reagieren, wobei das Datensteuerelement aber keine eigenen, sondern die *Recordset*-Ereignisse zur Verfügung stellt. Da diese in Kapitel 5 ausführlicher besprochen werden, bleibt es in diesem Abschnitt bei einer kurzen tabellarischen Auflistung.

Tabelle 4.4: Die Ereignisse des ADO-Datensteuerelements

Ereignis	Wann tritt es auf?	Was zeigt es an?
WillMove	Bei folgenden Ereignissen des *Recordset*-Objekts: *Open*, *MoveNext*, *Move*, *MoveLast*, *MoveFirst*, *MovePrevious*, *Bookmark*, *AddNew*, *Delete*, *Requery* und *Resync*.	Datensatzzeiger soll auf einen anderen Datensatz bewegt werden.
MoveComplete	Nach dem abgeschlossenen *WillMove*-Ereignis.	Datensatzzeiger wurde auf einen anderen Datensatz bewegt.
WillChangeField	Bevor sich der Wert eines Feldes ändert.	Der Inhalt eines oder mehrerer Felder soll sich ändern.
FieldChangeComplete	Nach dem abgeschlossenen *WillChangeField*-Ereignis.	Der Inhalt eines oder mehrerer Felder hat sich geändert.

Ereignis	Wann tritt es auf?	Was zeigt es an?
WillChangeRecord	Bei folgenden Ereignissen des *Recordset*-Objekts: *Update*, *Delete*, *CancelUpdate*, *UpdateBatch* und *CancelBatch*.	Der Inhalt des aktuellen Datensatzes soll sich ändern (Ereignis folgt dem *WillChangeField*-Ereignis).
RecordChangeComplete	Nach dem abgeschlossenen *WillChangeRecord*-Ereignis.	Der Inhalt des aktuellen Datensatzes wurde geändert.
WillChangeRecordset	Bei folgenden Ereignissen des *Recordset*-Objekts: *Requery*, *Resync*, *Close*, *Open* und *Filter*.	Der Inhalt des gesamten Recordset-Objekts soll sich ändern (Ereignis geht dem *WillChangeRecord*-Ereignis voraus).
RecordsetChangeComplete	Nach dem abgeschlossenen *WillChangeRecordset*-Ereignis.	Der Inhalt des *Recordset*-Objekts wurde geändert.
InfoMessage	Wenn der OLE-Provider eine Verbindungsnachricht weitergegeben hat.	Eine Meldung, die für die angeforderte Verbindung von Bedeutung ist.

Tabelle 4.4: Die Ereignisse des ADO-Datensteuerelements (Fortsetzung)

4.5 Weitere gebundene Steuerelemente

Neben den »fest eingebauten« Steuerelementen bietet Visual Basic eine Reihe weiterer Steuerelemente mit besonderen Eigenschaften: *DataList* (ein Listenfeld), *DataCombo* (ein Kombinationsfeld), *DataGrid* (ein Gitternetz), *Chart* (ein gebundenes Steuerelement zum Darstellen von Diagrammen) und *DataRepeater* zum Darstellen von Datensätzen in einer scrollbaren Liste. Darüber hinaus bietet Visual Basic 6.0 von Anfang an ein weiteres Grid als Alternative zum *DataGrid*: Es heißt *HFlexGrid* und ist besonders gut zur Darstellung hierarchischer Datensatzgruppen, aber auch normaler Tabellen geeignet. Das normale FlexGrid-Steuerelement ist zwar auch ein gebundenes Steuerelement, doch kann es nur an ein DAO-Datensteuerelement gebunden werden und kommt damit für ADO nicht in Frage. Als normales Tabellenblatt, das unabhängig von einer Datenbank oder manuell gebunden wird, ist es bestens geeignet. Es sei an dieser Stelle noch einmal erwähnt, daß ein Steuerelement nicht gebunden sein muß, damit Sie den Inhalt eines Datenbankfeldes anzeigen und bearbeiten können. Die einge-

baute Bindung erleichtert die Programmierung, da Sie sich um einige Details nicht mehr zu kümmern brauchen.

Am Anfang mag die Vielfalt bei den gebundenen Steuerelementen etwas verwirrend sein. Mit zunehmender Erfahrung werden Sie jedoch die Stärken und Schwächen der einzelnen Steuerelemente abwägen und jene Variante verwenden, die für einen bestimmten Zweck (vermeintlich) am besten geeignet ist. Da alle diese Steuerelemente nicht fester Bestandteil der Werkzeugsammlung sind, müssen sie zunächst über den Menübefehl PROJEKT/KOMPONENTEN zu einem Projekt hinzugefügt werden.

Wenn Sie Ihr Visual Basic 6.0 über Visual Basic 5.0 installiert haben, werden Sie in der Komponentenliste eine Reihe weiterer Steuerelemente, wie z.B. Data Bound Grid Control, entdecken. Diese Steuerelemente können nicht im Zusammenhang mit ADO eingesetzt werden.

Die einzelnen gebundenen Steuerelemente werden in diesem Kapitel nur kurz vorgestellt, da sie in der Regel im Zusammenhang mit der Darstellung von Datensatzgruppen verwendet werden, die aus mehreren Tabellen stammen, diese Thematik aber erst in Kapitel 9 behandelt wird. Ihre Eigenschaften, Methoden und Ereignisse sind in der MSDN-Hilfe ausführlich beschrieben. Möchten Sie z.B. erreichen, daß bei der Bearbeitung der Tabelle *Fahrzeugdaten*, in der jedes Fahrzeug lediglich durch seine Modellnummer und nicht durch seinen Modellnamen vertreten ist, dennoch der Modellname angezeigt wird, müssen Sie entweder über das DataCombo-Steuerelement die Tabelle *Fahrzeugdaten* und die Tabelle *Modelldaten* über ihr gemeinsames Feld *ModellNr* verbinden oder im Zusammenhang mit der Datenumgebung mit einem hierarchischen *Command*-Objekt arbeiten, bei dem das *Command*-Objekt für die Tabelle *Fahrzeugdaten* ein Unter-*Command*-Objekt für die Tabelle *Modelldaten* enthält. Das hört sich zwar viel komplizierter an, als es in Wirklichkeit ist, setzt aber gewisse Grundkenntnisse über den Umgang mit den ADO-Objekten voraus.

4.5.1 Das DataGrid-Steuerelement

Das DataGrid-Steuerelement ist ein Gitternetz mit einer bemerkenswerten Eigenschaft. Als Gitternetz ist es in der Lage, eine komplette Datensatzgruppe anzuzeigen. Folglich besitzt es lediglich eine *DataSource*- und eine *DataMember*-Eigenschaft. Eine *DataField*-Eigenschaft wird nicht benötigt, da stets alle Felder der Datensatzgruppe angezeigt werden.

Das Schöne dabei ist, daß Sie lediglich die *DataSource*-Eigenschaft mit dem Namen eines ADO-Datensteuerelements (oder einer Datenumgebung) verbinden müssen. Nach dem Programmstart zeigt das DataGrid nicht nur die Feldnamen in der Überschriftenzeile an, sondern paßt die Spaltenbreite (weitestgehend) automatisch an die Größe der Felder an. Ja mehr noch, während das Programm läuft, können Sie Daten editieren oder die Spaltenbreite mit der Maus justieren. In dieser Beziehung ist das DataGrid sehr flexibel.

Das DataGrid aus Visual Basic 6.0 ist »programmkompatibel« mit dem DBGrid aus Version 5.0. Das bedeutet, daß es die gleichen Eigenschaften, Ereignisse und Methoden unterstützt. Mit einer Ausnahme: Einen »Unbound«-Modus gibt es beim neuen DataGrid nicht. Und es gibt noch einen wichtigen Unterschied. Während das alte DBGrid auf DAO basierte, setzt das neue DataGrid auf OLE DB und ADO auf.

Hinzufügen des DataGrid-Steuerelements

Da das DataGrid-Steuerelement nicht fester Bestandteil der Werkzeugsammlung ist, müssen Sie es zu jedem Projekt hinzufügen (oder sich eine Projektvorlage anlegen, in der es bereits vorhanden ist).

1. Führen Sie den Menübefehl PROJEKT/KOMPONENTEN aus.
2. Wählen Sie den Eintrag »Microsoft DataGrid Control 6.0 (OLEDB)« aus, und bestätigen Sie mit *OK*.

Wenn Sie sehen möchten, wie breit das DataGrid später bei der Ausführung sein wird, können Sie die Feldnamen für die Überschriften bereits zur Laufzeit abrufen. Klicken Sie dazu das DataGrid mit der rechten Maustaste an, und wählen Sie im Kontextmenü den Eintrag FELDER ABRUFEN. Voraussetzung ist aber, daß die *DataSource*-Eigenschaft bereits auf ein ADO-Datensteuerelement (oder eine Datenumgebung) verweist, die wiederum mit einer Datensatzgruppe verbunden ist.

Anwendungsbeispiel für das DataGrid-Steuerelement

Die wichtigste Anwendung für das DataGrid ist nicht so sehr das Betrachten einer einzelnen Datensatzgruppe (wenngleich sich dieses, ohne eine einzelne Programmzeile eingeben zu müssen, bewerkstelligen läßt), da dies genauso mit Hilfe einzelner Steuerelemente möglich ist, die etwa mit Hilfe des

Formularassistenten ebenfalls ohne Programmierung hinzugefügt werden können. Sehr viel wichtiger in der Praxis ist jene Situation, in der zwei Datensatzgruppen zusammen angezeigt werden und Datensatzgruppe 1 ein Feld enthält, dessen Inhalt auf einen Datensatz der Datensatzgruppe 2 verweist. Das klassische Beispiel ist eine Kundentabelle mit einem Feld *KundenNr*. Bewegt sich der Anwender in dieser Datensatzgruppe, werden in dem DataGrid alle Aufträge des aktuellen Kunden aus der Auftragstabelle aufgelistet. Bezogen auf unsere Fuhrpark-Datenbank entspräche dieses Szenario einem Formular, bei dem der Anwender in der Tabelle *Modelldaten* in den Daten der verschiedenen Modelle blättern kann und zu jedem Modell in einem DataGrid die tatsächlich vorhandenen Fahrzeuge aufgelistet werden. Möglich wird eine solche automatische Verknüpfung allerdings nur unter Zuhilfenahme von SQL, so daß Sie auf Kapitel 9 vertröstet werden.

Ein wenig realistischer (bezogen auf den tatsächlichen Nutzen) ist eine Erweiterung der Fuhrpark-Datenbank, die ebenfalls in Kapitel 9 vorgestellt wird. Hier wird die Datenbank um eine Tabelle mit entliehenen Fahrzeugen erweitert. In diesem Szenario wird es möglich sein, durch die Datensatzgruppe der Fahrzeugdaten zu blättern und zu jedem Fahrzeug die »Ausleihvorgänge« (also welcher Mitarbeiter das Fahrzeug in welchem Zeitraum ausgeliehen hat) sehen zu können.

Ein kleines Beispiel mit dem DataGrid-Steuerelement

Im folgenden wird ein kleines Beispiel für das DataGrid-Steuerelement vorgestellt. Es wird davon ausgegangen, daß Sie, wie es in Kapitel 3.9 Schritt für Schritt beschrieben wird, eine UDL-Datei für unsere *Fuhrpark.mdb* angelegt haben. Sollte dies nicht der Fall sein, holen Sie es bitte zunächst nach. Eine UDL ist zwar keine Voraussetzung für den Zugriff, da Sie die erforderliche Verbindungszeichenfolge jedesmal neu anlegen können. Sie werden jedoch nach dem zehnten oder zwölften Mal feststellen, daß man sich diese Schrittfolge doch eigentlich sparen kann. Und genau dafür ist eine solche UDL-Datei da, denn hier sind alle erforderlichen Einstellungen in einer Datei zusammengefaßt.

Schritt 1: Hinzufügen eines DataGrid-Steuerelements

Als erstes muß das DataGrid-Steuerelement zur Werkzeugsammlung hinzugefügt werden. Wie das geht, wird zu Beginn dieses Abschnitts beschrieben.

Schritt 2: Anordnen des DataGrid auf dem Formular

Ordnen Sie das DataGrid auf dem Formular an, wobei es vor allem auf die Breite ankommt, denn der Anwender sollte später nicht sowohl horizontal als auch vertikal scrollen müssen.

Schritt 3: Ordnen Sie das ADO-Datensteuerelement auf dem Formular an

Die Verbindung zwischen der anzuzeigenden Datensatzgruppe und der *DataSource*-Eigenschaft des DataGrid soll über ein ADO-Datensteuerelement hergestellt werden (Sie werden in Kürze erfahren, wie sich diese Verknüpfung auch programmgesteuert aufbauen läßt).

Schritt 4: Setzen der ConnectionString-Eigenschaft

Wählen Sie über die *ConnectionString*-Eigenschaft des ADO-Datensteuerelements die UDL-Datei *Fuhrpark.udl* aus.

Schritt 5: Setzen der RecordSource-Eigenschaft

Wählen Sie für die *RecordSource*-Eigenschaft des ADO-Datensteuerelements die Tabelle *Modelldaten* aus.

Schritt 6: Setzen der DataSource-Eigenschaft des DataGrids

Verbinden Sie das DataGrid mit dem ADO-Datensteuerelement, indem Sie die *DataSource*-Eigenschaft des DataGrid auf den Namen des ADO-Datensteuerelements setzen.

Schritt 7: Starten des Programms

Starten Sie das Programm. Der Inhalt der Tabelle *Modelldaten* wird jetzt in dem DataGrid angezeigt, wobei Sie den aktuellen Datensatz über die Schaltflächen verschieben können (meist bleibt das ADO-Datensteuerelement aber unsichtbar).

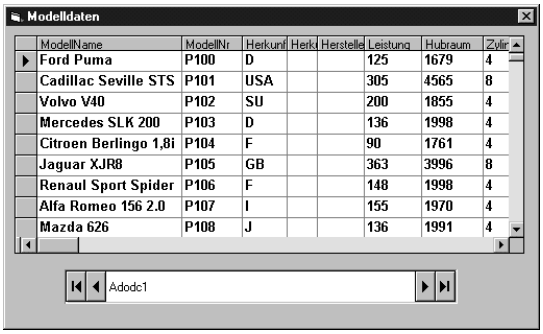


Bild 4.9: Das DataGrid zeigt den Inhalt der Tabelle »Modelldaten« an

Dies ist ein Tip, der ausnahmsweise einmal nichts mit der Programmierung zu tun hat. Klicken Sie beim DataGrid auf den kleinen Balken links vom linken Scrollpfeil, wird die Tabelle geteilt, und Sie können mehrere Ansichten gleichzeitig betrachten.

Programmgesteuerte Verknüpfung des DataGrid mit einer Datenquelle

Da sich die *DataSource*-Eigenschaft des DataGrid auch zur Laufzeit setzen läßt, ist es kein Problem, die anzuzeigende Datensatzgruppe erst während der Programmausführung auszuwählen:

```
Set dgrTest.DataSource = adoModelldaten
```

Besonders erwähnenswert ist der Umstand, daß sich das DataGrid (wie jedes gebundene Steuerelement) auch direkt an ein *Recordset*-Objekt binden läßt. Das macht das ADO-Datensteuerelement im Prinzip überflüssig:

```
Set Cn = New ADODB.Connection
Set Rs = New ADODB.Recordset
Cn.ConnectionString = "FILE NAME=C:\Eigene
Dateien\Fuhrpark.udl"
Cn.Open
Rs.Open ActiveConnection:=Cn, Source:="Modelldaten", _
   CursorType:=adOpenStatic
Set dgrTest.DataSource = Rs
```

Wenn Sie diese Befehle, z.B. in *Form_Load*, ausführen, wird der gleiche Effekt erzielt wie bei der Bindung über ein ADO-Datensteuerelement. Um die Befehlsfolge verstehen zu können, müssen Sie bereits die ADO-Objekte kennen, die in Kapitel 5 vorgestellt werden. Machen Sie sich also keine Gedanken, wenn das alles noch ein wenig kompliziert erscheint. Das kleine Beispiel soll lediglich demonstrieren, daß sich alle Einstellungen über die Benutzeroberfläche auch per Programm erledigen lassen.

Durchführen einer SQL-Abfrage mit dem DataGrid

Soll das DataGrid das Ergebnis einer SQL-Abfrage (also nur eine Teilmenge einer oder mehrerer Tabellen) anzeigen, muß das SQL-Kommando der *DataSource*-Eigenschaft des ADO-Datensteuerelements oder des *Recordset*-Objekts zugewiesen werden. Im folgenden wurde das letzte Beispiel so

modifiziert, daß nur die Wagen aus dem Herkunftsland »D« angezeigt werden:

```
Set Cn = New ADODB.Connection
Set Rs = New ADODB.Recordset
Cn.ConnectionString = "FILE NAME=C:\Eigene
Dateien\Fuhrpark.udl"
Cn.Open
Rs.Source = _
"Select * From Modelldaten Where HerkunftslandKürzel = 'D'"
Rs.Open ActiveConnection:=Cn, CursorType:=adOpenStatic
Set dgrTest.DataSource = Rs
```

Soll der gleiche Effekt ohne Programmierung erreicht werden, muß das SQL-Kommando bei der *RecordSource*-Eigenschaft im Eigenschaftsfenster des ADO-Datensteuerelements eingetragen werden.

Nach dem Zuweisen eines Wertes an die *RecordSource*-Eigenschaft kann der Aufruf der *Refresh*-Methode erforderlich sein:

```
adoTest.RecordSource = NeueAbfrage
adoTest.Refresh
```

Weitere Besonderheiten b eim DataGrid

Wird das DataGrid im Zusammenhang mit einer sogenannten Remote-Datenbank (also einer Datenbank, die über eine Netzwerkverbindung angesprochen wird, wie es z.B. beim Microsoft SQL-Server der Fall ist) aufgerufen, und sollte sich die Tabellenstruktur zwischenzeitlich geändert haben, zeigt das DataGrid nicht mehr die aktuellste Struktur an. Für diesen Fall muß mit der *Rebind*-Methode des DataGrid der Inhalt neu aufgebaut werden. Soll dabei auch das Layout der Tabelle neu aufgebaut werden (z.B. weil eine angezeigte Spalte in der neuen Tabellenstruktur nicht mehr existiert und daher auch nicht mehr angezeigt werden soll), muß zuvor die *ClearFields*-Methode aufgerufen werden.

Feststellen, welche Zellen selektiert wurden

Um festzustellen, wann eine Zelle vom Benutzer selektiert wird, gibt es beim DataGrid das *RowColChange*-Ereignis.

```
Private Sub grdData_RowColChange(LastRow As Variant, _
ByVal LastCol As Integer)
   ' Ausgabe der selektierten Zelle mit
   ' Zeilen- und Spaltennummer.
   Debug.Print grdData.Text; grdData.Row; grdData.Col
End Sub
```

4.5.2 Die CellText- und die CellValue-Eigenschaften

Für den Zugriff auf die Zellen einer einzelnen Spalte über die *Columns*-Eigenschaft spielt es eine Rolle, ob dazu die *CellText*- oder die *CellValue*-Eigenschaft verwendet wird. Jede Spalte mit einem numerischen Inhalt kann über die *NumberFormat*-Eigenschaft individuell formatiert werden. Diese Eigenschaft ändert allerdings nicht das tatsächliche Format der Daten. Während die *CellText*-Eigenschaft den formatierten String zurückgibt, liefert die *CellValue*-Eigenschaft den ursprünglichen Zellwert.

Der folgende Befehl legt für die Spalte *Zylinder* zunächst ein eigenes Zahlenformat fest:

```
grdData.Columns(5).NumberFormat = "00#"
```

Über die *CellText*-Eigenschaft erhält man dagegen den formatierten Wert des Feldes:

```
?grdData.Columns(5).CellText(grdData.Bookmark)
004
```

Über die *CellValue*-Eigenschaft dagegen den »wahren« Wert der Zelle:

```
?grdData.Columns(5).CellValue(grdData.Bookmark)
4
```

In beiden Fällen wird die *Bookmark*-Eigenschaft dazu benutzt, die aktuelle Spalte in dem Grid anzusprechen.

4.5.3 Das DataList- und das DataCombo-Steuerelement

Zwar sind auch das normale Listen- und Kombinationsfeld gebunden, doch sind deren Möglichkeiten relativ eingeschränkt. Bindet man ein einfaches Listenfeld an eine Datensatzgruppe, legt die Position des Datensatzzeigers lediglich den Wert der *ListIndex*-Eigenschaft fest. Das Auffüllen der Liste muß innerhalb des Programms geschehen. Für weiterreichende Möglichkeiten bietet Visual Basic das DataList- und das DataCombo-Steuerelement.

Möchten Sie erreichen, daß das DataList-Steuerelement automatisch mit dem Inhalt einer Spalte gefüllt wird, müssen Sie lediglich die Eigenschaft *RowSource* auf ein ADO-Datensteuerelement, eine Datenumgebung oder ein *Recordset*-Objekt und die Eigenschaft *ListField* auf den Namen der Spalte (also des Feldes) setzen. Doch beide Steuerelemente können noch etwas mehr. Ihr Haupteinsatzgebiet sind Situationen, in denen eine Datensatzgruppe bearbeitet wird, in der ein Feld als Fremdschlüssel auf einen Primärschlüssel in einer anderen Tabelle verweist. Anstelle des Fremdschlüssels (etwa eine Kundennummer) soll ein korrespondierender Wert in der Tabelle mit dem Primärschlüssel (etwa der Kundenname) angezeigt werden. Ein Beispiel wäre ein Dialogfeld, in dem der Inhalt einer Auftragstabelle angezeigt wird. Damit statt der wenig aussagekräftigen Kundennummer der Name des Kunden angezeigt wird, muß ein DataList- oder DataCombo-Steuerelement zum Einsatz kommen, das mit zwei verschiedenen Datensatzgruppen verbunden ist. Ein Beispiel für den Einsatz des DataList-Steuerelements bei der Erfassung neuer Datensätze für die Tabelle *Fahrzeugdaten* finden Sie in Kapitel 9. Hier sorgt das Steuerelement dafür, daß beim Erfassen eines neuen Datensatzes anstelle einer Liste aller bereits erfaßten Modellnummern eine Liste der korrespondierenden Modellnamen erscheint.

Neben der Fähigkeit, sich mit zwei verschiedenen *Recordset*-Objekten verbinden zu können, zeichnen sich sowohl das DataCombo- als auch das DataList-Steuerelement durch eine flexible Eingabetechnik aus, bei der der Benutzer einen Listeneintrag durch Eingabe der ersten Buchstaben auswählen kann.

4.5.4 Das Chart-Steuerelement

Das (neue) Chart-Steuerelement ist ein echtes Highlight unter den gebundenen OLE DB-Steuerelementen. Mit nur wenigen Zuweisungen läßt sich der (Zahlen-)Inhalt einer Tabelle oder einer SQL-Abfrage äußerst attraktiv in einem Diagramm darstellen, wobei eine vielseitige Auswahl an Diagrammtypen zur Verfügung stehen.

Hinzufügen des Chart-Steuerelements zur Werkzeugsammlung

Um das Chart-Steuerelement benutzen zu können, müssen Sie es zur Werkzeugsammlung hinzufügen:

1. Führen Sie den Menübefehl PROJEKT/KOMPONENTEN aus.
2. Wählen Sie den Eintrag »Microsoft Chart Control 6.0 (OLED6)«.

Sobald Sie das Chart-Steuerelement auf einem Formular anordnen, wird ein Diagramm angezeigt. Lassen Sie sich davon nicht irritieren, denn es sind nur »Zufallszahlen«. Es ist jedoch alles andere als kompliziert, echte Daten darzustellen. Da das Chart-Steuerelement über eine *DataSource*-Eigenschaft verfügt, können diese Daten entweder aus einer Datenquelle, etwa einem ADO-Datensteuerelement oder einem *Recordset*-Objekt, stammen oder direkt zugewiesen werden.

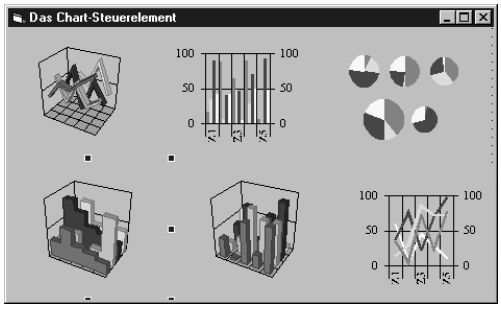


Bild 4.10: Das Chart-Steuerelement bietet eine interessante Auswahl an Diagrammtypen

Das folgende Beispiel soll Sie von der Leichtigkeit des Chart-Steuerelements überzeugen. Sie sollten zu diesem Zeitpunkt über die Datenbank *Fuhrpark.mdb* verfügen, in der die Tabelle *Fahrzeugdaten* die Daten der im Fuhrpark vorhandenen Fahrzeuge enthält. Gemäß dem in Kapitel 3.2 beschriebenen Aufbau gibt das Feld *Preis* der Tabelle den Anschaffungspreis eines Fahrzeugs an. Im folgenden wird gezeigt, wie Sie die Preise der vorhandenen Fahrzeuge in einem Balkendiagramm darstellen.

Schritt 1: Anordnen des Chart-Steuerelements auf dem Formular

Fügen Sie das Chart-Steuerelement, wie zu Beginn des Abschnitts beschrieben, zur Werkzeugsammlung hinzu, ordnen Sie es auf dem Formular an, und geben Sie ihm den Namen »dbChart«.

Schritt 2: Programmgesteuertes Herstellen einer Verbindung

Am einfachsten ist es, die *DataSource*-Eigenschaft bereits zur Entwurfszeit auf den Namen eines ADO-Datensteuerelements oder einer Datenumgebung (in diesem Fall muß die *DataMember*-Eigenschaft das *Recordset*-Objekt auswählen) zu setzen. Im folgenden wird gezeigt, wie sich der gleiche Effekt programmgesteuert erreichen läßt.

Deklarieren Sie zunächst die benötigten Variablen:

```
Private Cn As ADODB.Connection
Private Rs As ADODB.Recordset
```

Fügen Sie in *Form_Load* folgende Befehle ein:

```
Private Sub Form_Load()
    Set Cn = New ADODB.Connection
    With Cn
     .Provider = "Microsoft.Jet.OLEDB.3.51"
     .ConnectionString = _
        "Data Source=C:\Eigene Dateien\Fuhrpark.mdb"
     .Open
    End With
    Set Rs = New ADODB.Recordset
    With Rs
    .Source = "Select ModellNr, Preis From Fahrzeugdaten"
     .CursorType = adOpenStatic
     Set .ActiveConnection = Cn
     .Open
    End With
    With dbChart
        Set .DataSource = Rs
    End With
End Sub
```

Das (übliche) formelle Grundgerüst des Programms soll den Umstand nicht verdecken, daß die Prozedur nur zwei wichtige Befehle enthält:

```
.Source = "Select ModellNr, Preis From Fahrzeugdaten"
```

Hier wird die *Source*-Eigenschaft des *Recordset*-Objekts auf ein SQL-Kommando gesetzt, das die Auswahl der Datensätze durchführt (mehr zu den Feinheiten von SQL in Kapitel 7). Sie erhalten alle Datensätze der Tabelle *Fahrzeugdaten* zurück, allerdings nur die Felder *ModellNr* und Preis, da die übrigen Felder in dem Diagramm keine Rolle spielen sollen. Der zweite wichtige Befehl verbindet das *Recordset*-Objekt mit dem Chart-Steuerelement:

```
Set .DataSource = Rs
```

Anschließend werden die in dem *Recordset*-Objekt enthaltenen Daten in dem Diagramm dargestellt.

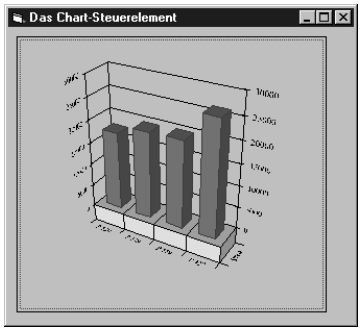


Bild 4.11: Das Chart-Steuerelement zeigt den Inhalt der Felder »Preise« der Tabelle »Fahrzeugdaten« an

Eine kleine Erweiterung

Im jetzigen Zustand werden absichtlich nur die Modellnummern, nicht aber die Modellnamen angezeigt. Möchten Sie die Modellnamen sehen, muß das SQL-Kommando erweitert werden:

```
.Source = "Select ModellName, Preis From Modelldaten INNER
JOIN  Fahrzeugdaten ON Fahrzeugdaten.ModellNr =
Modelldaten.ModellNr "
```

Jetzt sorgt eine JOIN-Operation dafür, daß der Modellname über das Feld *ModellNr* in der Tabelle *Modelldaten* nachgeschlagen wird.

Falls Ihnen die etwas gedrängte optische Darstellung nicht zusagt, besteht die Möglichkeit, daß der Modellname erst nach Anklicken des jeweiligen Balkens angezeigt wird. Dazu muß *PointSelected*-Ereignis des Chart-Steuerelements herangezogen werden:

```
Sub dbChart_PointSelected(Series As Integer, _
  DataPoint As Integer, MouseFlags As Integer, _
  Cancel As Integer)
    Dim sModellName As String
    With Rs
     If .State = 1 Then .Close
     .Source = _
     "Select ModellName From Modelldaten Where ModellNr =" _
     & Chr(39) & dbChart.DataGrid.RowLabel(DataPoint, 1) _
     & Chr(39)
     .CursorType = adOpenStatic
     Set .ActiveConnection = Cn
     .Open
     sModellName = Rs.Fields("ModellName").Value
     .Close
```

```
    End With
    MsgBox Prompt:=sModellName
End Sub
```

Der Ereignisprozedur wird die Nummer des angeklickten Balkens über den *DataPoint*-Parameter übergeben (falls mehrere Datenreihen dargestellt werden, muß auch der *Series*-Parameter abgefragt werden). Über die *RowLabel*-Eigenschaft des Unterobjekts *DataGrid* erhält man die Beschriftung des Balkens, die dem Inhalt des Feldes *ModellNr* entspricht. Die Modellnummer wiederum wird für ein SQL-Kommando verwendet, das aus der Tabelle *Modelldaten* den entsprechenden Modellnamen ausliest.

Ist es nicht schön, wie bei Visual Basic die Datenbankprogrammierung mit den übrigen Programmelementen zusammenspielt?

4.5.5 Das DataRepeater-Steuerelement

Das DataRepeater-Steuerelement ist ein besonderes Steuerelement, das Sie von Anfang an für Ihre Formulare berücksichtigen sollten, wenngleich es nicht auf Anhieb klar sein dürfte, was sich dahinter verbirgt. Sie wissen, daß ein Datensatz im allgemeinen aus mehreren Feldern besteht. Sie wissen auch, daß Sie den Inhalt eines Datensatzes entweder in mehreren unabhängigen Steuerelementen oder in einem Grid darstellen können. Beide Verfahren habe ihre Vor- und Nachteile. Unabhängige Steuerelemente bedeuten etwas mehr Arbeit, bieten aber ein individuelles Darstellungsformat. Ein Grid ist sehr einfach anzuwenden, dafür aber nicht sehr flexibel in der Darstellung. Das DataRepeater-Steuerelement versucht, beide Vorteile zu kombinieren. Sie können für die Darstellung der Feldinhalte beliebige Steuerelemente innerhalb der Ausgabefläche des DataRepeaters anordnen. Gleichzeitig können Sie, das ist der besondere Clou, die Ausgabefläche scrollen, so daß Sie sich, je nach Höhe der Ausgabefläche, mehrere Datensätze anschauen können.

Hinzufügen des DataRepeater-Steuerelements zur Werkzeugsammlung

Um das DataRepeater-Steuerelement (es liegt in Gestalt der Datei *MsDataRep.ocx* vor) benutzen zu können, müssen Sie es zur Werkzeugsammlung hinzufügen:

1. Führen Sie den Menübefehl PROJEKT/KOMPONENTEN aus.
2. Wählen Sie den Eintrag »Microsoft DataRepeater Control 6.0 (OLED6)«.

Das DataRepeater-Steuerelement in der Praxis

Leider gilt es vor dem Einsatz des DataRepeater-Steuerelements eine kleine Hürde zu nehmen. Sie können nicht einfach ein einzelnes Steuerelement in der Ausgabefläche des DataRepeater-Steuerelements anordnen. Sie müssen vielmehr das Steuerelement über die *RepeatedControlName*-Eigenschaft auswählen. Hier werden nur die bindungsfähigen Steuerelemente aufgeführt. Da Sie jedoch im allgemeinen nicht ein einzelnes Feld, sondern mehrere Felder oder den ganzen Datensatz anzeigen möchten, muß es ein spezielles Steuerelement sein, das für jedes darzustellende Datenbankfeld eine passende Eigenschaft besitzt. Und da es ein Steuerelement, das zufällig den Aufbau des Datensatzes besitzt, den Sie darstellen möchten, in der Werkzeugsammlung nicht geben dürfte, müssen Sie es in Form eines ActiveX-Steuerelements erst einmal erstellen. Das ist zwar nicht kompliziert, stellt aber einen zusätzlichen Arbeitsschritt dar.

Das Erstellen von ActiveX-Steuerelementen ist mit dem Ablaufmodell von Visual Basic 6.0 nicht möglich. Sie benötigen dazu mindestens die Einsteiger-Edition oder die kostenlos erhältliche Visual Basic 5.0 Control Creation Edition.

Das folgende Beispiel ist etwas umfangreicher. Es demonstriert den Einsatz des DataRepeater-Steuerelements zur Darstellung von Datensätzen der Tabelle *Fahrzeugdaten*. Es setzt allerdings ein spezielles ActiveX-Steuerelement voraus, das Sie zunächst erstellen müssen.

Schritt 1: Anlegen eines ActiveX-Steuerelement-Projekts

Starten Sie Visual Basic, und legen Sie ein neues ActiveX-Steuerelement-Projekt an. Das ActiveX-Steuerelement muß in eine OCX-Datei kompiliert werden, damit es vom DataRepeater benutzt werden kann.

Schritt 2: Umbenennen des UserControl-Objekts

Benennen Sie das *UserControl*-Objekt in »usrFahrzeugdaten« um und das Projekt in »FahrzeugControl«[1].

1 Der sogenannte »Programmatische Bezeichner« (kurz ProgID) setzt sich aus Projekt- und Controlname zusammen und darf nicht länger als 39 Zeichen sein.

Schritt 3: Anordnen von Steuerelementen

Ordnen Sie drei Textfelder (für die Felder *ModellNr*, *Farbe* und *Preis*) sowie ein DT-Picker-Steuerelement (siehe Kapitel 4.5.8 – für das Anschaffungsjahr) auf dem UserControl-Objekt an.

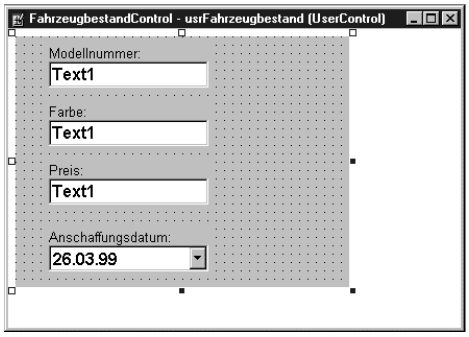


Bild 4.12: Das Benutzersteuerelement nach dem Anordnen der vier Steuerelemente

Schritt 4: Ändern von Eigenschaften

Um die Namenskonventionen einzuhalten, erhalten die vier Steuerelemente neue Namen:

Alter Name	Neuer Name
Text1	*txtModellNr*
Text2	*txtFarbe*
Text3	*txtPreis*
DTPicker1	*dtpAnschaffungsjahr*

Schritt 5: Hinzufügen von Bezeichnungsfeldern

Fügen Sie zu jedem Steuerelement ein Bezeichnungsfeld hinzu, dessen Aufgabe lediglich darin besteht, das Steuerelement zu bezeichnen. Die *Caption*-Eigenschaft jedes der Bezeichnungsfelder sollte daher dem Feldnamen entsprechen.

Schritt 6: Anpassen der Größe

Passen Sie die Größe des *UserControl*-Objekts so an, daß das *UserControl*-Objekt die Steuerelemente gerade umschließt (siehe Bild 4.12).

Schritt 7: Hinzufügen von Eigenschaften

Das Benutzersteuerelement ist damit fast fertig. Es fehlen noch Eigenschaften, über die später der Zugriff auf die Tabelle erfolgt. Fügen Sie in das *UserControl*-Modul folgende Befehle ein:

```
Public Property Get ModellNr() As String
    ModellNr = txtModellNr.Text
End Property

Property Let ModellNr (tmpWert As String)
   txtModellNr.Text = tmpWert
End Property

Public Property Get Farbe() As String
    Farbe = txtFarbe.Text
End Property

Property Let Farbe (tmpWert As String)
   txtFarbe.Text = tmpWert
End Property

Public Property Get Preis() As String
   Preis = txtPreis.Text
End Property

Property Let Preis (tmpWert As String)
    txtPreis.Text = tmpWert
End Property

Public Property Get Anschaffungsjahr() As String
    Anschaffungsjahr = dtpAnschaffungsjahr.Value
End Property

Property Let Anschaffungsjahr (tmpWert As String)
    dtpAnschaffungsjahr.Value = tmpWert
End Property
```

Da ein UserControl eine Klasse darstellt, werden auf diese Weise neue Eigenschaften hinzugefügt.

Schritt 8: Hinzufügen von PropertyChanged-Ereignissen

Damit eine Änderung einer Eigenschaft auch im Eigenschaftsdialogfeld des Benutzersteuerelements registriert wird, muß das *Change*-Ereignis der Steuerelemente Meldung machen. Fügen Sie in das Benutzersteuerelement folgende Befehle ein:

```
Private Sub txtModellNr_Change ()
    PropertyChanged "ModellNr"
End Sub

Private Sub txtFarbe_Change ()
    PropertyChanged "Farbe"
End Sub

Private Sub txtPreis_Change ()
    PropertyChanged "Preis"
End Sub

Private Sub dtpAnschaffungsjahr_Change ()
    PropertyChanged "Anschaffungsjahr"
End Sub
```

Schritt 9: Festlegen der Datenbindungen

Zum Schluß müssen die vier Eigenschaften »bindungsfähig« gemacht werden, damit sie später vom DataRepeater-Steuerelement benutzt werden können. Wer bereits ein ActiveX-Steuerelement programmiert hat, weiß, daß dies nicht gerade zu den herausragenden Eigenschaften von Visual Basic gehört. Gehen Sie dabei für alle vier Eigenschaftsprozeduren wie folgt vor:

- ✘ Setzen Sie die Textmarke in die erste der vier *Property*-Eigenschaftsprozeduren (das ist nicht zwingend notwendig, erleichtert die Angelegenheit aber ein wenig).
- ✘ Wählen Sie aus dem EXTRAS-Menü den Befehl PROZEDURATTRIBUTE.
- ✘ Stellen Sie sicher, daß in der Namensauswahlliste die Eigenschaft *ModellNr* selektiert ist.
- ✘ Klicken Sie auf die Schaltfläche *Weitere*.
- ✘ Selektieren Sie die Option *Datengebundene Eigenschaft* und *Zur Entwurfszeit in DataBindings-Auflistung anzeigen*.
- ✘ Wiederholen Sie die Schritte für die übrigen drei Eigenschaften.

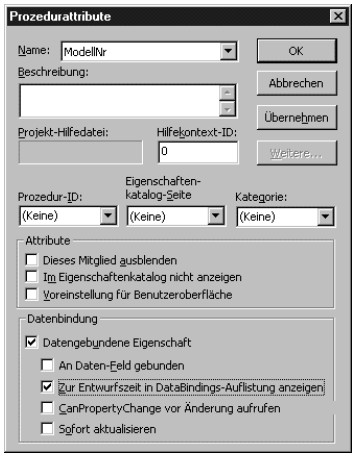


Bild 4.13: Die Eigenschaften müssen im Dialogfeld Prozedurattribute *bindungsfähig gemacht werden*

Schritt 10: OCX-Datei erstellen

Speichern Sie das Projekt ab, und erstellen Sie über den Menübefehl DATEI/FAHRZEUGCONTROL.OCX ERSTELLEN das ActiveX-Steuerelement. Dadurch wird es registriert, so daß es auf dem PC für alle künftigen Projekte automatisch zur Verfügung steht.

Schritt 11: ADO-Datensteuerelement auf dem Formular anordnen

Legen Sie ein neues Standard-Exe-Projekt an, ordnen Sie ein ADO-Datensteuerelement auf dem Formular an, und geben Sie ihm den Namen »adoDaten«.

Schritt 12: Binden des ADO-Datensteuerelements

Binden Sie das ADO-Datensteuerelement über seine *ConnectionString*-Eigenschaft an die Datenbank *Fuhrpark.mdb*, und wählen Sie über die *RecordSource*-Eigenschaft die Tabelle *Fahrzeugdaten* aus.

Schritt 13: DataRepeater-Steuerelement auf dem Formular anordnen

Fügen Sie ein DataRepeater-Steuerelement zur Werkzeugsammlung hinzu, ordnen Sie es auf dem Formular an, und geben Sie ihm den Namen *dtpFahrzeugdaten*.

Schritt 14: DataRepeater-Steuerelement mit dem ADO-Datensteuerelement verbinden

Weisen Sie der *DataSource*-Eigenschaft des DataRepeater-Steuerelements den Namen des ADO-Datensteuerelements zu.

Schritt 15: Verbinden des DataRepeater-Steuerelements mit dem ActiveX-Steuerelement

Damit der DataRepeater »weiß«, welches ActiveX-Steuerelement es darstellen soll, müssen Sie ihm dies über die *RepeatedControlName*-Eigenschaft mitteilen. Nach dem Öffnen der Auswahlliste erscheinen sämtliche registrierten ActiveX-Steuerelemente, was insofern ungünstig ist, als daß nur ein Teil davon in Frage kommt. Wählen Sie das Steuerelement mit dem (Doppel-)Namen *FahrzeugControl.usrFahrzeugdaten.ocx* aus. Es wird daraufhin im DataRepeater dargestellt.

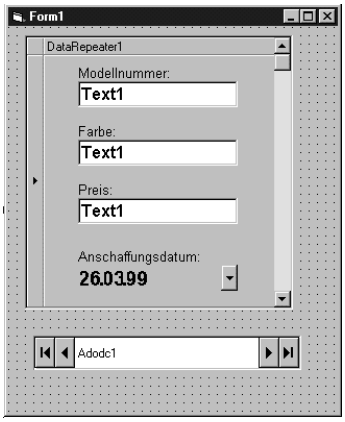


Bild 4.14: Das ActiveX-Steuerelement wird im DataRepeater dargestellt

Schritt 16: Verbinden des DataRepeater mit den Datenbankfeldern

Zum Schluß müssen Sie den einzelnen Eigenschaften des ActiveX-Steuerelements die passenden Datenbankfelder zuweisen. Wählen Sie dazu im Eigenschaftsdialogfeld des DataRepeater den Eintrag *Benutzerdefiniert*, öffnen Sie das Eigenschaftendialogfeld, und aktivieren Sie die Registerkarte *RepeaterBindings*. In der Auswahlliste *PropertyName* sollten nun alle Eigenschaften aufgelistet werden. Ordnen Sie jeder Eigenschaft nacheinander über die Auswahlliste *DataField* das passende Datenbankfeld zu, und klikken Sie auf *Hinzufügen*.

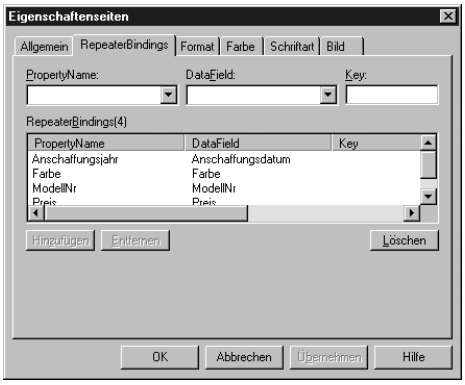


Bild 4.15: In der Registerkarte RepeaterBindings findet die Zuordnung zwischen den Eigenschaften des ActiveX-Steuerelements und den Datenbankfeldern statt

Schritt 17: Das Programm wird gestartet

Geschafft! Wenn Sie das Programm starten, sollte im DataRepeater der erste Datensatz angezeigt werden. Durch die Bildlaufleiste des DataRepeater (das ADO-Datensteuerelement bleibt passiv im Hintergrund) können Sie sich durch die Datensatzgruppe bewegen.

4.5.6 Das HFlexGrid-Steuerelement

Das HFlexGrid-Steuerelement ist eine direkte Alternative zum DataGrid. Es wird über seine *DataSource*-Eigenschaft mit einem ADO-Datensteuerelement, einer Datenumgebung oder programmgesteuert mit einem *Recordset*-Objekt verbunden. Neben einer Fülle von Einstellmöglichkeiten (die alle in der MSDN-Hilfe erklärt werden), bietet das HFlexGrid-Steuerelement eine herausragende Eigenschaft, der es auch seinen Namen verdankt. Es ist in der Lage, sogenannte hierarchische *Recordset*-Objekte darzustellen, die entweder aus dem SQL-Kommando *SHAPE* oder einer Datenumgebung mit *Command*-Objekten, die wiederum über Unter-*Command*-Objekte verfügen, resultiert. Auch bezüglich hierarchischer *Recordsets* werden Sie noch einmal auf Kapitel 9 vertröstet, da in diesem Kapitel praktische Beispiele auf der Grundlage der Fuhrpark-Datenbank vorgestellt werden.

Hinzufügen des HFlexGrid-Steuerelements zur Werkzeugsammlung

Um das HFlexGrid-Steuerelement benutzen zu können, müssen Sie es zur Werkzeugsammlung hinzufügen:

✘ Führen Sie den Menübefehl PROJEKT/KOMPONENTEN aus.

✘ Wählen Sie den Eintrag »Microsoft Hierarchical FlexGrid Control 6.0 (OLED6)«.

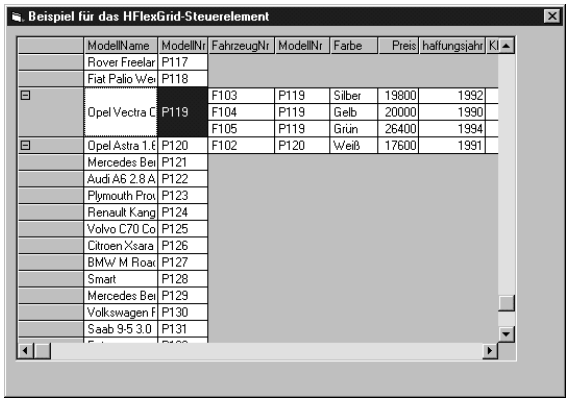


Bild 4.16: Das HFlexGrid-Steuerelement besitzt im Vergleich zum DataGrid eine Reihe zusätzlicher Möglichkeiten – wie z.B. die Darstellung von Hierarchien in einer Datensatzgruppe

4.5.7 Das FlexGrid-Steuerelement

Das FlexGrid-Steuerelement ist ein relativ einfaches, dafür aber auch recht schnelles Grid, das als Alternative zum DataGrid zur Verfügung steht. Auch wenn es über eine *DataSource*-Eigenschaft verfügt, kann es in der jetzigen Version 6.0 nicht an eine OLE DB-Datenquelle gebunden werden. Das ist nicht weiter tragisch, da für es diesen Zweck das HFlexGrid-Steuerelement gibt. Wenn Sie das einfache FlexGrid einsetzen möchten, müssen Sie den Datenzugriff über das alte Datensteuerelement (und damit über die DAO-Objekte) durchführen[1].

4.5.8 Die Steuerelemente MonthView und DatePicker

Zum Abschluß dieses Kapitels gibt es noch einen angenehmen Ausblick auf den neuen Komfort, auf den Visual-Basic-Programmierer bis zur Version 6.0 oft verzichten mußten. Das ist allerdings weniger der Verdienst von Visual Basic, sondern beruht auf dem Umstand, daß mit dieser Version eine Reihe neuer Windows-Standardsteuerelemente zur Verfügung gestellt wurden. (Standardsteuerelemente sind Steuerelemente, die Windows für Anwendungen bereithält, so daß die Anwendungen die Steuerelemente nicht

1 Sowohl FlexGrid als auch HFlexGrid wurden von der kleinen Softwarefirma VideoSoft eingebaut. Sollte es Updates zu beiden Steuerelementen geben, erfahren Sie es im Internet unter *www.videosoft.com*. Hier können Sie sich auch eine Probierversion des »großen Bruders« VSFlexGrid Pro 6.0 herunterladen.

selber zur Verfügung stellen müssen – insgesamt gibt es 14 Standardsteuerelemente.) Zu den Standardsteuerelementen gehören unter anderem:

- ✘ MonthView
- ✘ DataPicker

Beide Steuerelemente sind sich sehr ähnlich. Während MonthView stets im Stile eines Kalenderblattes einen kompletten Monat anzeigt, zeigt DatePicker zunächst nur ein einzelnes Datum an. Das (vermutlich identische) MonthView-Steuerelement wird erst beim Öffnen des Steuerelements angezeigt. Beiden Steuerelementen ist gemeinsam, daß sie über die Eigenschaften *DataField*, *DataFormat*, *DataMember* und *DataSource* verfügen und zur Auswahl eines Datums bestens geeignet sind.

Hinzufügen des MonthView- und des DatePicker-Steuerelements zur Werkzeugsammlung

Um sowohl das MonthView- als auch das DatePicker-Steuerelement benutzen zu können, müssen Sie es zur Werkzeugsammlung hinzufügen:

1. Führen Sie den Menübefehl PROJEKT/KOMPONENTEN aus.
2. Wählen Sie den Eintrag »Microsoft Windows Common Controls-2 6.0«.

Lassen Sie sich nicht durch den Umstand irritieren, daß gleich eine ganze Reihe von Steuerelementen in der Werkzeugsammlung erscheinen. MonthView und DatePicker gehören zu den Standardsteuerelementen, die wiederum ein Teil von Windows sind und bei Visual Basic in Gestalt zweier OCX-Dateien zur Verfügung gestellt werden.

Die folgende Übung geht davon aus, daß die Datenbank *Fuhrpark.mdb* bereits die Tabelle *EntlieheneFahrzeuge* enthält, die in Kapitel 9 vorgestellt wird. In dieser Tabelle sind neben der Fahrzeugnummer und einem Mitarbeiternamen auch das Entleihdatum und das Rückgabedatum enthalten. Beide Datenbankfelder sind natürlich ideale Kandidaten für das DatePicker- bzw. das MonthView-Steuerelement.

Schritt 1: Anordnen des ADO-Datensteuerelements

Ordnen Sie auf einem leeren Formular ein ADO-Datensteuerelement an, und geben Sie ihm den Namen »adoDaten«. Wählen Sie über die *ConnectionString*-Eigenschaft die Datenbank *Fuhrpark.mdb* und über die *RecordSource*-Eigenschaft die Tabelle *EntlieheneFahrzeuge* aus.

Schritt 2: Hinzufügen der Steuerelemente

Fügen Sie (gemäß Bild 4.17) ein Textfeld (*txtMitarbeitername*), ein DatePicker-Steuerelement (*dtpAusleihdatum*) und ein MonthView-Steuerelement (*mtvRückgabedatum*) hinzu. Setzen Sie die *DataSource*-Eigenschaft dieser Steuerelemente auf den Namen des ADO-Datensteuerelements. Wählen Sie über die *DateField*-Eigenschaft jeweils ein passendes Datenbankfeld aus:

Steuerelement	Datenbankfeld
txtMitarbeitername	Mitarbeitername
dtpAusleihdatum	Ausleihdatum
mtvRückgabedatum	Rückgabedatum

Schritt 3: Starten des Programms

Starten Sie das Programm. Sie können sich über das ADO-Datensteuerelement in der Datensatzgruppe bewegen, wobei in den beiden Datumssteuerelementen das Datum nicht nur angezeigt wird, sondern auch eingestellt werden kann.

Bild 4.17: DatePicker und MonthView in Aktion – achten Sie auf das Rückgabedatum!

4.6 Zusammenfassung

Die Aufgabe des ADO-Datensteuerelements ist es, die Verbindung zwischen einer Datenquelle und gebundenen Steuerelementen herzustellen. Die Datenquelle kann jede Datenbank sein, für die es einen OLE DB-Provider gibt, es kann aber auch eine Visual-Basic-Klasse sein. Das ADO-Datensteuerelement ist keine Alternative, sondern eher eine Ergänzung zu den ADO-Objekten und der Datenumgebung. Es basiert ebenfalls auf den ADO-Objekten *Connection* und *Recordset*.

Zwingend erforderlich ist das ADO-Datensteuerelement nicht, da Steuerelemente über ihre *DataSource*-Eigenschaft auch mit einer Datenumgebung (in diesem Fall wird die Datensatzgruppe über die *DataMember*-Eigenschaft ausgewählt) oder direkt mit einem *Recordset*-Objekt verbunden werden können. Das ADO-Datensteuerelement stellt eine gewisse Erleichterung (gerade beim Kennenlernen der ADO-Technik) dar, da sich mit seiner Hilfe sehr schnell die Verbindung zu einer Datenquelle herstellen läßt. Mit zunehmender Erfahrung werden Sie es vermutlich nur noch selten einsetzen.

4.7 Wie geht es weiter?

Mit dem ADO-Datensteuerelement haben Sie die erste und vermeintlich einfachste Variante für den Zugriff auf eine OLE DB-Datenquelle kennengelernt. Im nächsten Kapitel wird mit den ADO-Objekten die Grundlage für das ADO-Datensteuerelement vorgestellt. In Kapitel 6 lernen Sie, daß es mit der Datenumgebung eine Alternative gibt, die das ADO-Datensteuerelement in den meisten Fällen überflüssig machen kann. Das ADO-Datensteuerelement ist ein Relikt aus jenen Tagen, in denen es keine Alternativen gab. Das macht das ADO-Datensteuerelement nicht weniger wertvoll. Man muß es lediglich als eine von mehreren Möglichkeiten sehen, um von einem Visual-Basic-Programm an eine Datenbank »heranzukommen«.

4.8 Fragen

Frage 1:

Welche beiden Eigenschaften sorgen dafür, daß das ADO-Datensteuerelement mit einer Datensatzgruppe verbunden wird?

Frage 2:

Welche Werte kann die *DataSource*-Eigenschaft eines ADO-Datensteuerelements besitzen?

Frage 3:

Welche Eigenschaft legt bei einem gebundenen Steuerelement fest, welches Datenbankfeld angezeigt wird?

Frage 4:

Welche Rolle spielt die *DataFormat*-Eigenschaft bei einem gebundenen Datensteuerelement?

Frage 5:

Ist es möglich, und wenn ja auf welche Weise, das Ergebnis einer SQL-Abfrage direkt über das ADO-Datensteuerelement anzusprechen?

Frage 6:

Warum können nicht alle gebundenen Steuerelemente im Zusammenhang mit dem ADO-Datensteuerelement eingesetzt werden?

Die Antworten zu den Fragen finden Sie in Anhang D.

Kapitel 5

Die ADO-Objekte stellen sich vor

Die ADO-Objekte sind bei Visual Basic der Schlüssel zur Datenbank – pardon, zur Datenquelle. Zwar gibt es mit den DAO-Objekten noch einen weiteren Schlüssel, der beim Zugriff auf die Jet-Engine manchmal aufgrund fehlender Funktionalität in den ADOs 2.1 auch als »Ersatzschlüssel« fungieren kann, doch sind aus bereits mehrfach dargelegten Gründen die ADOs am wichtigsten. Mit dem ADO-Datensteuerelement (Kapitel 4) haben Sie bereits eine Variante kennengelernt, ADO-Funktionalität in einem Visual-Basic-Programm zu benutzen. Mit der Datenumgebung lernen Sie in Kapitel 6 eine weitere Variante kennen. In diesem Kapitel geht es um die ADO-Objekte selber, denn unabhängig davon, wie komfortabel der Datenzugriff dank Datensteuerelementen und Datenumgebungen auch von Fall zu Fall ist, ohne die Mitwirkung der ADO-Objekte geht nichts. Auf eine einfache Formel gebracht: Je besser Sie die ADOs kennen, desto weiter steht Ihnen das Tor zur Datenbankprogrammierung mit Visual Basic und VBA offen. Ganz einfach, und das sei bereits vorweg genommen, ist es allerdings auch nicht. Auch wenn es bei ADO nur elf Objekte gibt, ergeben sich in der Praxis gerade am Anfang viele Situationen, in denen man auf Ausprobieren, vage Vermutungen und andere Hilfsmittel angewiesen ist. Die MSDN-Hilfe erklärt zwar alle Objekte mit ihren Eigenschaften, Methoden und Ereignissen, doch bleibt dabei manche Frage offen. Der beste Weg, um ADO-Objekte kennenzulernen, sind daher einfache Beispiele, die Sie in diesem Kapitel und in Kapitel 8 (gewisse Überschneidungen lassen sich leider nicht vermeiden) kennenlernen werden.

Die Behandlung der ADO-Ereignisse fällt bereits in die Kategorie der fortgeschritteneren Programmierung. Sie sollten Ihr Augenmerk daher zunächst

auf das Objektmodell und die ADO-Objekte *Connection* und *Recordset* richten und die Ereignisse zu einem späteren Zeitpunkt nachholen.

Sie lernen in diesem Kapitel etwas zu folgenden Themen:

- ✘ Das ADO-Objektmodell in der Übersicht
- ✘ Die Verbindung zur Datenquelle über das *Connection*-Objekt herstellen
- ✘ Der Zugriff auf die Daten über das *Recordset*- und das *Fields*-Objekt
- ✘ Das *Command*- und das *Parameter*-Objekt
- ✘ Die Fehlerbehandlung über das *Errors*-Objekt
- ✘ Die Ereignisse der ADO-Objekte
- ✘ ADO und Microsoft Access 2000

In diesem Kapitel geht es in erster Linie um eine Vorstellung der verschiedenen ADO-Objekte. Praktische Beispiele finden Sie in Kapitel 8, in dem die Datensatzgruppen und das *Recordset*-Objekt im Vordergrund stehen.

5.1 Warum überhaupt Objekte?

Warum gibt es überhaupt so etwas wie die *Active Data Objects*, kurz ADOs? Warum gibt es in VBA keine »Datenbankbefehle«, oder warum wird der Zugriff nicht einfach über einen Satz von Funktionen durchgeführt? Auf diese mehr als berechtigte Frage gibt es gleich zwei Antworten. Erstens: Das »Objektprinzip« ist der Grundgedanke der Visual-Basic-Philosophie. Praktisch alles, was nicht Teil von VBA ist (und das betrifft u.a. die komplette Datenbankprogrammierung), muß über externe Objekte angesprochen werden. Zweitens: Das ADO-Prinzip bietet eine enorme Flexibilität. Mögliche Alternativen, wie etwa eine fiktive Funktionsbibliothek, pure VBA-Programmierung auf der Basis der Dateibefehle (siehe Kapitel 1), aber auch die alten DAO-Objekte, könnten da nicht mithalten. Wer sich bereits mit der ODBC-Programmierung beschäftigt hat, kann einen direkten Vergleich ziehen. Anders als bei ADO bzw. OLE DB kann die ODBC-Funktionalität entweder durch den direkten Aufruf von Funktionen (enthalten in der System-DLL *ODBC.DLL*) oder aber Visual-Basic-konform über Objekte (die RDOs oder die DAOs im ODBCDirect-Modus) durchgeführt werden. Auch wenn der Funktionsaufruf durchaus machbar und nicht besonders kompliziert ist (und von Datenbankprogrammierern daher auch praktiziert wird), ist der Weg über die Objekte sehr viel komfortabler. Man erreicht die gleiche Wirkung, meistens jedoch mit sehr viel weniger Aufwand. Da OLE DB

anders als ODBC auf dem *Component Object Model* (COM) und damit auf einer objektorientierten Schnittstelle basiert, kam die Bereitstellung einer Funktionsbibliothek (auch wenn es machbar wäre) gar nicht erst in Frage. Die ADOs sind daher der einzige und beste Weg, die Funktionalität von OLE DB zu nutzen.

Leider ist der offizielle Sprachgebrauch alles andere als einheitlich. Wer die Microsoft-Webseite nach dem Stichwort »ADO« durchsucht, findet hier an einigen Stellen den Begriff »ADO-API«. Hier hat irgend jemand nicht richtig nachgedacht oder versucht, den ODBC-Programmierern eine kleine Brücke zu bauen. Unter der ADO-API versteht man auch hier das ADO-Objektmodell. Es gibt keine ADO-Funktionen, sondern nur Objekte, Eigenschaften, Methoden und Ereignisse.

5.1.1 Ein Wort zur Versionsnummer

Die ADOs werden unabhängig von Visual Basic entwickelt, da sie allen Entwicklungswerkzeugen (z.B. Visual C++, Visual InterDev, Office/VBA und auch den Scriptsprachen) zur Verfügung stehen. Das bedeutet, daß, noch während die Version 6.0 von Visual Basic aktuell ist, aktuellere ADO-Versionen erscheinen können. Die aktuelle (Stand: 4/99) ADO-Version lautet 2.1 (sie wird u.a. durch den Internet Explorer 5.0 installiert)[1]. Wer dagegen noch mit Visual Basic 5.0 arbeitet und den Internet Explorer 4 installiert hat, wird unter Umständen lediglich über ADO-Version 1.5 verfügen. Da in diesem Buch nur die Grundfunktionalität von ADO beschrieben wird, spielt die Versionsnummer keine Rolle.

In diesem Buch geht es um ADO 2.0, da diese Version mit Visual Basic 6.0 ausgeliefert wurde.

Wer tiefer in die Datenbankprogrammierung mit ADO einsteigen will oder bereits an einem konkreten Projekt arbeitet, sollte auf neuere Versionen achten, da diese neue Funktionen (oder einfach nur weniger Fehler) enthalten können. Die stets aktuellste Version der ADOs kann im Rahmen des *Microsoft Data Access-Pakets* (kurz MSDAC) im Internet unter der Adresse

1 Im Zusammenhang mit der Vorabversion von Windows 2000 ist bereits eine Version 2.5 im Umlauf.

www.microsoft.com/data/ado heruntergeladen werden. Laden Sie dabei aber nur MSDAC (ruhig in der kleinsten Verpackungseinheit) herunter, nicht aber das Data Access SDK, da letzteres für Visual Basic keine Rolle spielt.

Für einige Programmierer mag es ungewohnt sein, mit einem Entwicklungswerkzeug nicht wie früher üblich stets die aktuellste Version zu erhalten, sondern sich diese erst aus dem Internet besorgen und dabei zu allem Überfluß noch auf die richtige Versionsnummer achten zu müssen. In Anbetracht der kurzen Entwicklungszyklen ist das bei den ADOs aber nicht anders machbar[1]. Dafür findet man auf der ADO-Webseite (leider nur in Englisch) nützliche Informationen und Probekapitel aus aktuellen Büchern. Aber, darauf soll noch einmal hingewiesen werden, zum Kennenlernen der ADOs müssen Sie nichts herunterladen, sondern können mit jener Version arbeiten, die Teil von Visual Basic 6.0 ist.

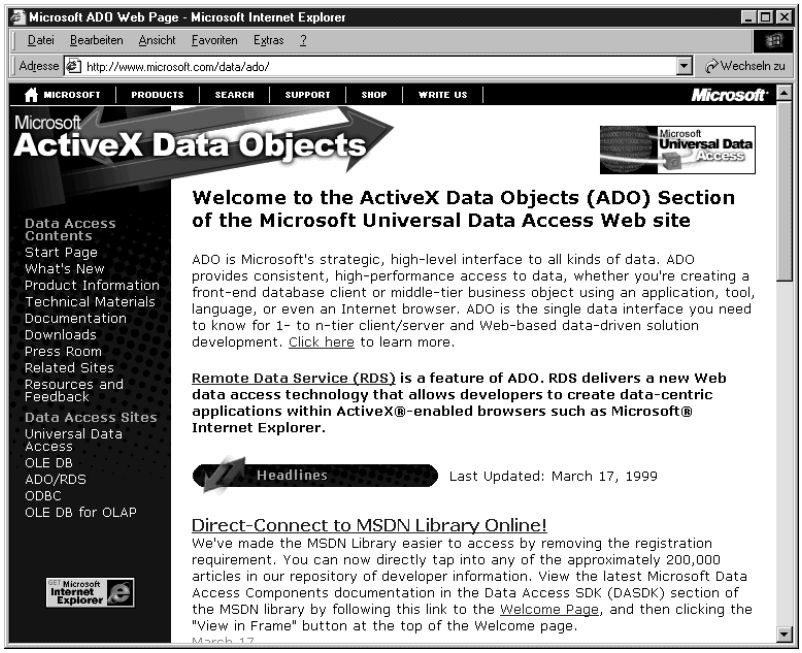


Bild 5.1: Die stets aktuellste ADO-Version gibt es auf der Microsoft-Webseite

1 Dafür sind sie auch kostenlos, wenngleich eine CD, die allen registrierten Anwendern zugeschickt wird, keine schlechte Idee wäre.

5.2 Das ADO-Objektmodell in der Übersicht

Bevor es an die einzelnen ADO-Objekte geht, sollen Sie zunächst einen Gesamtüberblick erhalten. Da sich hinter den ADOs mehrere Objekte verbergen, die zueinander in Beziehung stehen, spricht man von einem *Objektmodell*. Stellen Sie sich unter dem Begriff bitte nicht zuviel vor. Ein Objektmodell, das es z.B. auch für die verschiedenen Office-Anwendungen oder die Entwicklungsumgebung von Visual Basic gibt, beschreibt lediglich die Objekte mit ihren Eigenschaften, Methoden und Ereignissen. Der Begriff »Modell« rührt von dem Umstand her, daß zwischen den einzelnen Objekten eine Beziehung bestehen kann (aber nicht bestehen muß). So besitzt im ADO-Objektmodell das *Recordset*-Objekt eine *Fields*-Eigenschaft, über die ein Zugriff auf die einzelnen *Field*-Objekte (also die Felder eines Datensatzes) möglich ist.

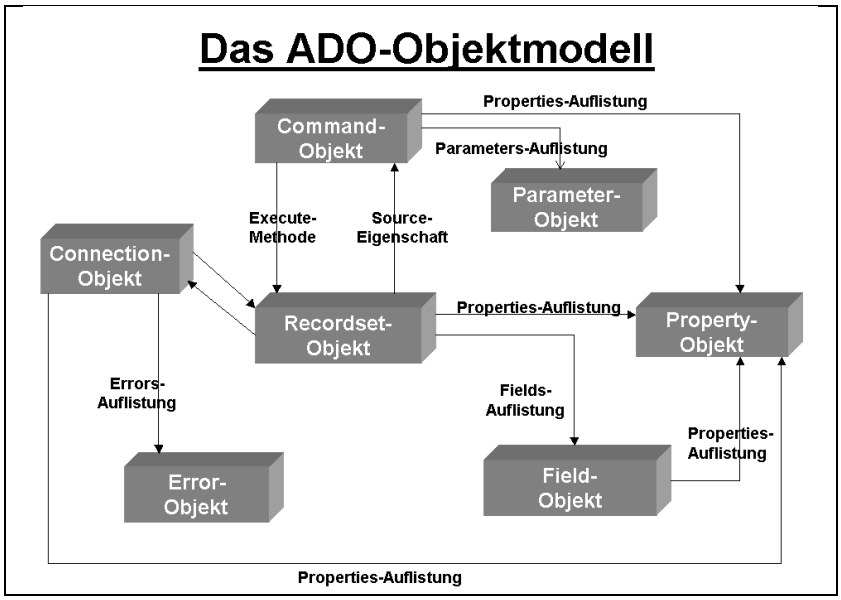
Wer bereits die DAOs kennt, wird am Anfang vermutlich ein wenig irritiert sein, da das ADO-Objektmodell nur ganze elf Objekte umfaßt (bei den DAOs sind es immerhin über 30). Das liegt daran, daß die ADOs von einem sehr allgemeinen Modell einer Datenquelle ausgehen, in dem z.B. die spezifischen Eigenschaften der Jet-Engine (wie etwa der Zugriff auf die Datenbankstruktur oder die Benutzerverwaltung) nicht enthalten sind. Das heißt aber nicht, daß diese Dinge nicht möglich sind. Zum einen gibt es bei Microsoft Access 2000 eine Erweiterungsschnittstelle mit dem Namen *ADOX*, die genau diese Funktionalität bietet[1]. Zum anderen hat niemand etwas dagegen, wenn Sie frei nach dem Motto »Das richtige Werkzeug für den richtigen Zweck« für diese Dinge auf die DAOs zurückgreifen.

In Bild 5.2 sehen Sie das ADO-Objektmodell mit den insgesamt sieben Objekten. Moment, sind es nicht elf? Im Prinzip ja, daß jedoch einige der Kästchen (Objekte) etwas anders dargestellt sind, hat einen besonderen Grund. Es handelt sich um Auflistungen, d.h. Objekte, deren Aufgabe darin besteht, andere Objekte zu »beinhalten«. Ein Beispiel für eine Auflistung ist das *Fields*-Objekt, das lediglich den Zugang zu allen *Field*-Objekten (also den Feldern eines Datensatzes) ermöglicht, aber keine eigene Funktionalität be-

1 Was heißt das nun wieder? Ganz einfach, unter dem offiziellen Namen »ADO Extensions for Data Definition Language and Security« stehen zusätzliche Objekte zur Verfügung, mit denen sich u.a. die Tabellenstruktur oder Benutzergruppen ansprechen lassen.

sitzt. Das *Fields*-Objekt zählt daher häufig nicht als ADO-Objekt, auch wenn es ein ADO-Objekt ist (alles klar?). Entsprechendes gilt für die Objekte *Errors*, *Properties* und *Parameters*, bei denen es sich ebenfalls um Auflistungen handelt. Zieht man die reinen Auflistungen ab, verbleiben nur noch sieben »wahre« ADO-Objekte. Übrigens wäre eine optische Markierung nicht notwendig gewesen, denn üblicherweise besitzen Auflistungsobjekte den gleichen Namen wie das Objekt, das sie beinhalten, nur daß der Name mit einem »s« endet (um die Mehrzahl anzudeuten).

Bild 5.2: Das ADO-Objektmodell im Gesamtüberblick



Die ADO-Objekte, um die es in diesem Kapitel geht, gehören zur ADODB-Bibliothek. Dieser Hinweis ist deswegen wichtig, weil es für die Jet-Engine unter dem Namen ADOX eine weitere ADO-Bibliothek gibt (mehr dazu in Kapitel 5.12)

Tabelle 5.1: Die ADO-Objekte in der Gesamtübersicht

ADO-Objekt	Bedeutung
Connection	Stellt die Verbindung zu einer Datenquelle her.
Recordset	Enthält die von einer SQL-Abfrage oder dem Öffnen einer Tabelle zurückgegebenen Datensätze.
Field	Steht für ein einzelnes Feld eines Datensatzes. Wird über das *Fields*-Objekt zur Verfügung gestellt.

ADO-Objekt	Bedeutung
Command	Steht für eine einzelne Datenbankabfrage. Ist in der Regel für die Ausführung von SQL-Aktionsabfragen (z.B. *UPDATE*) oder Abfragen mit Parametern zuständig, es kann aber auch für normale Abfragen oder Tabellen stehen. Die *Execute*-Methode des *Command*-Objekts gibt ein *Recordset*-Objekt zurück.
Parameter	Steht für einen einzelnen Parameter einer SQL-Abfrage mit Parametern. Wird über das *Parameters*-Objekt zur Verfügung gestellt.
Error	Steht für einen Fehler, der während der letzten Datenbankoperation auftrat. Wird über das *Errors*-Objekt zur Verfügung gestellt.
Property	Steht für ein einzelnes Attribut eines ADO-Objekts, das zu speziell ist, um es über eine reguläre Eigenschaft zur Verfügung zu stellen. So besitzt das *Connection*-Objekt beim OLE DB-Provider für die Jet-Engine 68 verschiedene Eigenschaften. Die verschiedenen *Property*-Objekte werden über die *Properties*-Eigenschaft zur Verfügung gestellt. Jedes *Property*-Objekt besitzt eine *Name*- und eine *Value*-Eigenschaft.

Tabelle 5.1: Die ADO-Objekte in der Gesamtübersicht (Fortsetzung)

In den folgenden Abschnitten werden zu den einzelnen ADO-Objekten auch deren wichtigsten Eigenschaften und Methoden vorgestellt (um die Ereignisse geht es in Kapitel 5.11). Um nicht jedesmal zwischen Eigenschaften und Methoden sprachlich unterscheiden zu müssen, werden beide auch unter dem Sammelbegriff »Mitglieder« zusammengefaßt. Dieser Begriff wird z.B. auch im Objektkatalog verwendet.

5.3 Die Idee der Objektvariablen

Der Zugriff auf die ADO-Objekte erfolgt nicht direkt, sondern über Objektvariablen. Eine Objektvariable ist eine Variable, die eine Referenz (also eine Adresse) auf ein Objekt im Arbeitsspeicher enthält. Der Zugriff auf die Objektvariable entspricht dem Zugriff auf das Objekt, da die Objektvariable der »Stellvertreter« des (im Arbeitsspeicher angelegten) Objekts ist. Um beispielsweise eine Verbindung zu einer Access-Datenbank über ein *Connection*-Objekt öffnen zu können, wird zunächst eine Variable vom Typ *ADODB.Connection* benötigt:

```
Private Cn As ADODB.Connection
```

Das Präfix »ADODB« wäre nicht zwingend notwendig und soll in diesem Zusammenhang lediglich Verwechslungen mit anderen *Connection*-Objekten vermeiden. Indem der Name der Objektbibliothek (und damit der Name des obersten »Objekts«) vorangestellt wird, wird das Objekt eindeutig ausgewählt. Damit existiert aber lediglich eine Variable und kein Objekt. Diese wird erst durch den *Set*-Befehl, z.B. in *Form_Load*, zum Leben erweckt:

```
Set Cn = New ADODB.Connection
```

Dieser Befehl legt zwar ein *Connection*-Objekt an, es besitzt aber noch keine Einstellungen. Diese muß es durch eine Reihe von Zuweisungen erhalten, wofür sich der *With*-Befehl als nützlich erweist:

```
With Cn
     .CursorLocation = adUseClient
     .Provider = "Microsoft.Jet.OLEDB.3.51"
     .ConnectionString = "C:\Eigene Dateien\Fuhrpark.mdb"
     .Open
End With
```

Achten Sie auf die vorletzte Zeile. Durch den Aufruf der *Open*-Methode wird das *Connection*-Objekt geöffnet. Ging alles gut, verfügen Sie über eine aktive Verbindung. Sie können die Variable *Cn* z.B. beim Öffnen eines *Recordset*-Objekts übergeben:

```
Set Rs = New ADODB.Recordset
Rs.Open ActiveConnection:=Cn
```

Am Ende werden Objektvariablen auf *Nothing* (den Wert Null für Objektvariablen) gesetzt, was aber meist nicht zwingend notwendig ist.

In einigen Beispielen wird eine Objektvariable in einem Schritt definiert und instanziiert:

```
Dim Cn As New ADODB.Connection
```

Dies ist zum einen kein guter Programmierstil und kann zum anderen eine Reihe subtiler Nebeneffekte nach sich ziehen. So sollten Sie daher nicht programmieren, sondern statt dessen Deklaration und Instanzierung auf zwei Befehle verteilen:

```
Dim Cn As ADODB.Connection
```

und später

```
Set Cn = New ADODB.Connection
```

Auch die *CreateObject*-Funktion (streng genommen ist es eine Methode) sollte für das Erstellen eines ADO-Objekts nicht verwendet werden, auch wenn es möglich ist, da VBA in diesem Fall einen Registry-Zugriff durchführen muß (um über die übergebene ProgID den CLSID-Wert und damit den »Aufenthaltsort« der COM-Komponenten zu lokalisieren), was eine Weile dauern kann.

5.4 Das Connection-Objekt

Mit diesem Abschnitt beginnend, lernen Sie die sieben ADO-Objekte endlich im Detail kennen. Für das Herstellen einer Verbindung zur Datenquelle ist das *Connection*-Objekt zuständig. Ein *Connection*-Objekt benötigt eine sogenannte Verbindungszeichenfolge (engl. »connection string«), die sich aus mindestens zwei Angaben zusammensetzt:

- ✘ Den Namen des OLE DB-Providers
- ✘ Den Namen der Datenquelle

Die Verbindungszeichenfolge wird entweder der *ConnectionString*-Eigenschaft zugewiesen oder beim Aufruf der *Open*-Methode als Zeichenkette übergeben. Auch für die Angabe des OLE DB-Providers gibt es Alternativen. Sie können seinen Namen entweder in die Verbindungszeichenfolge einfügen oder über die dafür zuständige Eigenschaft *Provider* festlegen. Alle diese Verfahren führen zum gleichen Ziel, zu einer geöffneten Verbindung. Das ist typisch für ADO. Ein und dieselbe Angelegenheit kann auf verschiedene Weisen geregelt werden. Tabelle 5.2 gibt eine Übersicht über die bei Visual Basic 6.0 zur Auswahl stehenden OLE DB-Provider und den Namen, der der *Provider*-Eigenschaft zugewiesen werden muß, damit der betreffende Provider ausgewählt wird. Diese Liste wird laufend erweitert. Der Vollständigkeit halber wird auch der OLE DB Provider für die Jet Engine 4.0 aus Microsoft Access 2000 aufgelistet, der allerdings nicht Teil von Visual Basic 6.0 ist.

OLE DB-Provider	Provider-Eigenschaft	OLE DB-Provider für ...
Microsoft Jet 3.51	*Microsoft.Jet.OLEDB.3.51*	... den Zugriff auf Datenbanken, die mit Access 95 oder Access 97 erstellt wurden.
Microsoft Jet 4.0	*Microsoft.Jet.OLEDB.4.0*	... den Zugriff auf Datenbanken, die mit Microsoft Access 2000 erstellt wurden.

Tabelle 5.2: Übersicht über die bei Visual Basic 6.0 zur Verfügung stehenden OLE DB-Provider

Tabelle 5.2: Übersicht über die bei Visual Basic 6.0 zur Verfügung stehenden OLE DB-Provider (Fortsetzung)

OLE DB-Provider	Provider-Eigenschaft	OLE DB-Provider für ...
Microsoft SQL Server	*SQLOLEDB*	... den Zugriff auf SQL-Server-6.5- und -7.0-Datenbanken.
Oracle	*MSDAORA*	... den Zugriff auf Oracle-Datenbanken, wobei nur statische Cursor unterstützt werden.
ODBC	*MSDASQL*	... den Zugriff auf Datenbanken, für die ein ODBC-Treiber vorhanden ist. Dies ist die Voreinstellung, d.h. wird kein Provider ausgewählt, wird dieser Provider verwendet.
Microsoft Index Server	*MSIDXS*	... den Nur-Lese-Zugriff auf jene Daten, die vom Microsoft Index Server indiziert worden sind. Das können z.B. die Inhalte aller HTML-Dokumente in einem Webverzeichnis sein.
Microsoft Active Directory Service	*ADSDSOObject*	... den Zugriff auf jene Objekte (wie z.B. Netzwerkverbindungen, Drucker, Benutzer usw.), die von einem Verzeichnisdienst verwaltet werden. Setzt aber Windows 2000 voraus.

Mit zunehmender Erfahrung werden Sie lernen, daß Ihnen ADO meistens mehrere Wege anbietet, eine Aufgabe zu erledigen. So können Sie eine Verbindung wahlweise beim Öffnen eines *Recordset*-Objekts angeben. Anstatt zuerst ein *Connection*-Objekt anzulegen, übergeben Sie die erforderliche Verbindungszeichenfolge beim Aufruf der *Open*-Methode:

```
Rs.Open ActiveConnection:= _
 "Provider=Microsoft.Jet.OLEDB.3.51; Data Source=C:\Eigene
Dateien\Fuhrpark.mdb", Source:="Personenfahrzeuge"
```

Na, wunderbar. Warum soll ich dann überhaupt erst ein *Connection*-Objekt anlegen, wenn es auch ohne geht? Nun, auch wenn die obige Variante etwas schneller ausgeführt werden kann, gibt es viele Situationen, in denen man ein explizites *Connection*-Objekt möchte. Auch im obigen Fall erhalten Sie ein *Connection*-Objekt, nur daß es »unsichtbar« ist und nur für das eine *Recordset*-Objekt zur Verfügung steht. Der Nachteil ist daher, daß das *Connection*-Objekt nicht von mehreren *Recordset*-Objekten gemeinsam genutzt werden kann, da es keiner Objektvariablen zugewiesen wurde. Außerdem glänzt der obige Befehl nicht gerade durch Übersichtlichkeit. Fazit, der einfachste Weg muß nicht immer der beste sein.

5.4.1 Herstellen einer Verbindung über das Öffnen eines Connection-Objekts

Die folgende Befehlsfolge öffnet ein *Connection*-Objekt für den Zugriff auf die Datenbank *Biblio.mdb*.

```
Const DBPfad = _
    "C:\Programme\Microsoft Visual Studio\VB98\Biblio.mdb"

Set Cn = New ADODB.Connection
With Cn
 .Provider = "Microsoft.Jet.OLEDB.3.51"
 .ConnectionString = "Data Source=" & DBPfad
 .Open
' Irgendwelche Befehle
 .Close
End With
```

Über die *Provider*-Eigenschaft wird der Name des OLE DB-Providers festgelegt (wird kein Name angegeben, geht Visual Basic von dem OLE DB-Provider für ODCB aus). Muß man ein Objekt stets über seine *Close*-Methode schließen? Im Prinzip ja, da es zu einem guten Programmierstil gehört. Bei Remote-Datenbanken, also Datenbanken, wie z.B. der SQL-Server, die von einem Server verwaltet werden, belasten zu viele offene Verbindungen den Arbeitsspeicher. Wenn Sie eine *Close*-Methode vergessen sollten, müssen Sie deswegen jedoch keinen Programmabsturz befürchten. Wird das Programm beendet oder verliert die Objektvariable ihre Gültigkeit, wird das Objekt (in der Regel) automatisch geschlossen.

5.4.2 Herstellen einer Verbindung beim Öffnen des Recordset-Objekts

ADO gilt als flexibel, was bedeutet, daß es stets mehrere Möglichkeiten gibt, eine Aufgabe zu erledigen. Das *Recordset*-Objekt macht das sehr schön deutlich, denn es kann geöffnet werden, ohne daß zuvor ein *Connection*-Objekt angelegt werden muß.

Die folgende Befehlsfolge öffnet ein *Recordset*-Objekt direkt.


```
Set Rs = New ADODB.Recordset
Rs.Open Source:="Authors", _
  ActiveConnection:= _
  "Provider=Microsoft.Jet.OLEDB.3.51; Data Source= " & _
  DBPfad
```

```
' Irgendwelche Befehle
Rs.Close
```

Der Umstand, daß hier kein *Connection*-Objekt im Spiel ist, darf nicht darüber hinweg täuschen, daß im Hintergrund dennoch eines angelegt werden mußte (es steht über die *ActiveConnection*-Eigenschaft zur Verfügung). Es ist daher im allgemeinen empfehlenswerter, die erste Variante und den vermeintlichen »Umweg« über ein *Connection*-Objekt zu wählen. Der wichtigste Vorteil ist, daß ein einmal geöffnetes *Connection*-Objekt von beliebig vielen *Recordset*- oder *Command*-Objekten benutzt werden kann.

5.4.3 Die wichtigsten Mitglieder des Connection-Objekts

Die beiden wichtigsten Eigenschaften beim *Connection*-Objekt sind *ConnectionString*, denn hier wird festgelegt, mit welcher Datenquelle eine Verbindung hergestellt werden soll, und *Provider*, die den zu verwendenden OLE DB-Provider festlegt.

Tabelle 5.3: Die wichtigsten Mitglieder des Connection-Objekts

Mitglied	Bedeutung
ConnectionString-Eigenschaft	Gibt die Verbindungszeichenfolge an, die für den Zugang zu einer OLE DB-Datenquelle verwendet wird.
Provider-Eigenschaft	Legt den OLE DB-Provider fest, über den der Zugriff durchgeführt wird.
Open-Methode	Öffnet das *Connection*-Objekt.
Close-Methode	Schließt das *Connection*-Objekt.
Mode-Eigenschaft	Bestimmt, in welchem Modus später Datensatzgruppen geöffnet werden (z.B. Exklusiv oder Nur-Lesen; diese Einstellungen werden bei DAO beim Anlegen des *Database*-Objekts, d.h. beim Öffnen der Datenbank getroffen).

Die wichtige *Mode*-Eigenschaft des *Connection*-Objekts spielt bei den Beispielen dieses Buches keine Rolle, da es sich stets um »Single-User«-Anwendungen handelt. Bei Datenbanken, die von mehreren Anwendern gleichzeitig benutzt werden können, sollten Sie sich mit dieser Eigenschaft aber ausführlich beschäftigen (z.B. MSDN-Hilfe), denn durch sie wird u.a. bestimmt, ob eine Datenbank exklusiv (*adModeShareExclusive*) geöffnet wird, so daß kein anderer Benutzer auf die Verbindung und damit auf die Datenbank zugreifen kann. Der Standardwert ist *adModeUnknown* (0), der besagt, daß

keine spezifischen Rechte existieren oder nicht bestimmt werden konnten. Es ist zu beachten, daß die *Mode*-Eigenschaft nur bei einem geschlossenen Connection-Objekt gesetzt werden kann, und daß eine getroffene Einschränkung (etwa Nur-Lesen) nicht durch ein über das *Connection*-Objekt geöffnetes *Recordset*-Objekt aufgehoben werden kann.

5.5 Das Recordset-Objekt

Ein geöffnetes *Connection*-Objekt ist nur die sprichwörtlich halbe Miete. Der Zugriff auf die Daten wird grundsätzlich über das *Recordset*-Objekt durchgeführt. Das *Recordset*-Objekt steht dabei für die Datensatzgruppe als Ganzes (also für alle Datensätze, die z.B. von einer Abfrage zurückgegeben wurden)[1]. Der Zugriff auf die Daten erfolgt über ein *Field*-Objekt, das von der *Fields*-Eigenschaft zurückgegeben wird.

Die folgenden Befehlszeilen öffnen zunächst eine Verbindung zur beliebten Datenbank *Biblio.mdb* und anschließend die Tabelle *Authors* über ein *Recordset*-Objekt.


```
Set Cn = New ADODB.Connection
With Cn
 .Provider = "Microsoft.Jet.OLEDB.3.51"
 .ConnectionString = "Data Source=" & DBPfad
 .Open
End With
Set Rs = New ADODB.Recordset
Rs.Open Source:="Authors", ActiveConnection:=Cn
' Irgendwelche Befehle
Rs.Close
Cn.Close
```

Haben Sie erst einmal ein geöffnetes *Recordset*-Objekt zur Verfügung, stehen die eigentlichen Daten über die *Fields*-Eigenschaft zur Verfügung. Diese Eigenschaft gibt eine Auflistung mit allen *Field*-Objekten des aktuellen Datensatzes zurück.

1 Der Begriff »Recordset« wäre mit Plattensammlung sicher nur unzureichend übersetzt, wenngleich es nicht ganz abwegig wäre.

Die folgende Befehlsfolge gibt den Inhalt aller Felder des aktuellen Datensatzes im Direktfenster aus.

```
Sub AlleFelderAusgeben(Datensatz As ADODB.Recordset)
    Dim F As ADODB.Field
    For Each F In Datensatz.Fields
        Debug.Print F.Value
    Next
End Sub
```

5.5.1 Besonderheiten der Open-Methode

Die *Open*-Methode des *Recordset*-Objekts führt den eigentlichen Datenbankzugriff aus, indem sie z.B. den Inhalt einer kompletten Tabelle oder das Ergebnis einer SQL-Abfrage in Gestalt einer Gruppe von Datensätzen in den Arbeitsspeicher einliest. Was für Anfänger reichlich verwirrend sein kann, ist der Umstand, daß es verschiedene Typen von Datensatzgruppen gibt. So gibt es Datensatzgruppen, die nicht geändert werden können. Bei anderen Datensatzgruppen liefert die *RecordCount*-Eigenschaft, die eigentlich die Anzahl der Datensätze zurückgeben sollte, lediglich den Wert –1. Die Tabelle 5.4 enthält eine Übersicht über die unterschiedlichen Datensatzgruppen. Welchen Typ die *Open*-Methode öffnet, wird über die Eigenschaft *CursorType* bestimmt (mehr zur Definition eines Cursors in Kapitel 8). Wird beim Öffnen eines *Recordset*-Objekts weder diese Eigenschaft noch die *LockType*-Eigenschaft gesetzt, was bei den Beispielen der letzten Abschnitte der Fall war, erhält man eine Datensatzgruppe vom Typ *ForwardOnly*, in der nur Vorwärtsbewegungen des Datensatzzeigers erlaubt sind, und die darüber hinaus vom Typ *ReadOnly* ist, d.h. keine Änderungen zuläßt.

Machen Sie sich über die verschiedenen Einstellungen im Moment noch keine Gedanken. Die tiefere Bedeutung des Cursortyps ist ein sehr vielschichtiges Thema, das beim Zugriff auf die Jet-Engine aber keine allzu große Rolle spielt. Merken Sie sich lediglich, daß bei einem »normalen« Zugriff auf eine Datensatzgruppe ein *Recordset*-Objekt meistens wie folgt geöffnet wird:

```
With Rs
     .ActiveConnection = Cn
     .CursorType = adOpenStatic
     .LockType = adLockOptimistic
     .Source = "Authors"
     .Open
End With
```

Sie erhalten damit ein *Recordset*-Objekt, dessen Datensätze Sie verändern können (darauf kommt es meistens an) und dessen *RecordCount*-Eigenschaft die Anzahl der Datensätze zurückgibt.

Datensatzgruppentyp (Cursor)	Parameter bei Open	Besonderheiten
Dynamisch	*adOpenDynamic*	Fügen andere Anwender (Prozesse) Datensätze zur Gruppe hinzu oder sie entfernen sie, spiegelt sich diese Änderung in der aktuellen Datensatzgruppe wieder.
Vorwärts (ForwardOnly)	*adOpenForwardOnly*	In dieser Gruppe ist nur eine Vorwärtsbewegung möglich (wird z.B. ein *MoveLast* ausgeführt, wird die Datensatzgruppe erneut gelesen). Der Preis für diese Einschränkung ist eine hohe Performance.
Keyset	*adOpenKeyset*	»Mittelding« zwischen statischem und dynamischem Cursor. Wie beim statischen Cursor kann sich die Gruppe der Mitglieder nicht ändern, wie beim dynamischen Cursor sind Updates an den Mitgliedern möglich.
Statisch	*adOpenStatic*	Die Datensätze können aktualisiert werden. Fügen aber andere Anwender (Prozesse) Datensätze hinzu, oder entfernen sie welche, wird dies in der Datensatzgruppe nicht berücksichtigt. Inwieweit dieser Typ unterstützt wird, hängt vom OLE DB-Provider ab.

Tabelle 5.4: Die verschiedenen Datensatzgruppentypen

Neben der *CursorType*-Eigenschaft gibt es eine zweite Eigenschaft, deren Wert eine wichtige Rolle für die geöffnete Datensatzgruppe spielt. Es ist die *LockType*-Eigenschaft, die festlegt, ob und auf welche Weise der Inhalt einer Datensatzgruppe vom Benutzer geändert werden kann. Die zur Auswahl stehenden Werte sind in Tabelle 5.5 zusammengefaßt.

LockType-Einstellung	Bedeutung
adLockBatchOptimistic	Diese Form der Sperre wird gewählt, wenn mehrere Änderungen auf einmal im Rahmen eines sogenannten Batch-Updates (hier befinden sich die Änderungswünsche auf einem »Stapel«, der abgearbeitet wird) durchgeführt werden. Spielt für »normale« Datenbankprogrammierung keine Rolle.

Tabelle 5.5: Die verschiedenen Einstellungen für die LockType-*Eigenschaft*

Tabelle 5.5: Die verschiedenen Einstellungen für die LockType-Eigenschaft (Fortsetzung)

LockType-Einstellung	Bedeutung
adLockOptimistic	Eine optimistische Sperre bedeutet, daß der Datensatz erst in dem Augenblick für andere gesperrt wird, wenn die *Update*-Methode ausgeführt wird.
adLockPessimistic	Eine pessimistische Sperre bedeutet, daß der Datensatz bereits in dem Augenblick für andere gesperrt wird, wenn die erste Änderung erfolgt.
adLockReadOnly	Die Datensatzgruppe kann nur gelesen, nicht aber geändert werden. Dies ist die Voreinstellung, wenn kein Wert für *LockType* festgelegt wird.

5.5.2 Kurzer Exkurs: die Idee der Auflistungen

Für alle Leser, die noch nicht hundertprozentig mit dem Prinzip der Auflistungen vertraut sind, hier eine kurze Wiederholung. *Fields* ist eine Eigenschaft, die eine Auflistung zurückgibt. Eine Auflistung ist ein Objekt, das (wie eine Feldvariable) eine Reihe von Unterobjekten enthalten kann. Im Falle des *Fields*-Objekts handelt es sich um eine Auflistung von *Field*-Objekten. Auf diese können Sie auf zwei unterschiedliche Weisen zugreifen:

1. über den Index (also eine Zahl zwischen 0 und der maximalen Anzahl an Elementen minus 1)

   ```
   Wert = Rs.Fields(0).Value
   ```

2. über den Namen

   ```
   Wert = Rs.Fields("Author").Value
   ```

Für welche Variante Sie sich entscheiden, hängt von den jeweiligen Umständen ab. Die erste Variante ist immer dann nützlich, wenn alle Felder in einer Schleife durchlaufen werden sollen und ein Schleifenzähler benötigt wird. Die zweite Variante stellt nicht nur sicher, daß das richtige Feld angesprochen wird (in einem Datensatz kann eine bestimmte Feldreihenfolge nicht garantiert werden), sondern macht das Programm sehr viel lesbarer.

Die von den DAO-Objekten bekannte Abkürzung mit dem !-Operator ist auch bei den ADOs erlaubt:

```
Wert = Rs!Author
```

Auch wenn diese Schreibweise aufgrund ihrer Kürze nicht nur verlockend, sondern auch völlig legal ist, empfehle ich, sie am Anfang nicht zu verwenden, da man schnell übersieht, daß auch bei diesem Ausdruck die Regeln des Objektmodells zur Anwendung kommen.

Über die *Fields*-Eigenschaft erfahren Sie nicht nur die Inhalte der Felder eines Datensatzes, sondern auch ihre Namen. Sie müssen nur anstelle der *Value*-Eigenschaft die *Name*-Eigenschaft verwenden:

```
For Each F In Rs.Fields
    Debug.Print F.Name
Next
```

5.5.3 Die Rolle des Datensatzzeigers

Es ist wichtig zu verstehen, daß es in einem *Recordset*-Objekt, also in einer Gruppe von Datensätzen, in der Regel einen »aktuellen« Datensatz gibt. Dieser aktuelle Datensatz ist jener Datensatz, auf den sich Methoden wie *Delete* (Löschen des aktuellen Datensatzes) oder *MoveNext* (mache den nächsten Datensatz zum aktuellen Datensatz) beziehen. Der aktuelle Datensatz wird durch den Datensatzzeiger festgelegt. Gibt es keinen aktuellen Datensatz, dann wurde der Datensatzzeiger entweder über den Beginn (die *BOF*-Eigenschaft des *Recordset*-Objekts ist in diesem Fall *True*) oder das Ende der Datensatzgruppe (*EOF*-Eigenschaft=*True*) bewegt, oder die Datensatzgruppe enthält keine Datensätze (in diesem Fall sind sowohl *BOF* als auch *EOF True*). Wird in diesem Fall eine Methode, wie etwa *MoveNext*, ausgeführt, ist ein Laufzeitfehler die Folge.

5.5.4 Eine Frage des Cursors

Ein *Recordset*-Objekt bildet nur den programmtechnischen Rahmen für eine Datensatzgruppe, verwaltet wird die Datensatzgruppe von dem sogenannten *Cursor*. Der Cursor sorgt dafür, daß es möglich ist, einzelne Datensätze über den Datensatzzeiger anzusteuern und diesen in der Datensatzgruppe vor- und zurückzubewegen. Gäbe es gar keinen Cursor, erhielte man bei einer SQL-Abfrage die Datensätze vielleicht als eine Aneinanderreihung von Zeichenketten zurück. Das Sortieren der Datensätze oder das Scrollen in der Datensatzgruppe wäre nicht möglich. Ein Recordset-Objekt unterstützt vier verschiedene Cursortypen: *Dynamisch*, *Keyset*, *Statisch* und *Vorwärts* (siehe Tabelle 5.4). Wird ein *Recordset*-Objekt geöffnet, muß ein Cursortyp festgelegt werden. Geschieht dies nicht, erhält das *Recordset*-Objekt den Cursor *Vorwärts*, was gewisse Einschränkungen bedeutet. Dies geschieht in erster Linie aus Performance-Gründen, denn je weniger Verwaltungsaufwand mit einem Cursor verbunden ist, desto schneller kann der Zugriff auf die Datensätze erfolgen. Der »teuerste« Cursor (bezogen auf die

Performance) ist der Cursortyp *Dynamisch*, da dieser den meisten Komfort bietet. Die zwei wichtigsten Fragen bei einem Cursortyp sind:

- ✘ Wie werden Änderungen an vorhandenen Datensätzen angezeigt?
- ✘ Wie werden Änderungen an der Datensatzgruppe angezeigt?

Bei den Cursortypen *Dynamisch* und *Keyset* werden Änderungen an einzelnen Datensätzen automatisch berücksichtigt. Die Cursortypen *Statisch* und *Vorwärts* fertigen eine Art »Momentaufnahme« der Daten an, Änderungen werden nur dann berücksichtigt, wenn eine *Resync*-Methode ausgeführt wurde. Werden von anderen Anwendern zur Datensatzgruppe (d.h. zu den Tabellen, die der Datensatzgruppe zugrundeliegen) Datensätze hinzugefügt oder entfernt, wird dies bei den Cursortypen *Keyset*, *Statisch* und *Vorwärts* nicht automatisch angezeigt. Hier muß eine *Requery*-Methode ausgeführt werden. Nur beim Cursortyp *Dynamisch* werden Änderungen am »Mitgliederbestand« automatisch berücksichtigt.

Abschließend sei zu dem Thema »Cursor, wie finde ich den richtigen?« erwähnt, daß diese Frage nur bei größeren Datenbankanwendungen eine Rolle spielt, wenn eine Abfrage unter Umständen weit über 10000 Datensätze zurückgeben kann und die Performance eine wichtige Rolle spielt. Bei typischen Access-Anwendungen dürften Sie mit einem *Keyset*-Cursor und dem Sperrtyp *optimistisch* am besten fahren.

5.5.5 Die wichtigsten Mitglieder des Recordset-Objekts

Das *Recordset*-Objekt verfügt über eine große Auswahl an Eigenschaften und Methoden. Sie werden aber sehr schnell feststellen, daß sie am Anfang nur wenige davon benötigen und diese praktisch selbsterklärend sind.

Tabelle 5.6: Die wichtigsten Mitglieder des Recordset-*Objekts*

Mitglied	Bedeutung
BOF-Eigenschaft	Ist *True*, wenn der Datensatzzeiger über den Anfang der Datensatzgruppe hinausbewegt wurde.
EOF-Eigenschaft	Ist *True*, wenn der Datensatzzeiger über das Ende der Datensatzgruppe hinausbewegt wurde.
MoveNext-Methode	Bewegt den Datensatzzeiger auf den nächsten Datensatz in der Datensatzgruppe.
MovePrevious-Methode	Bewegt den Datensatzzeiger auf den vorherigen Datensatz in der Datensatzgruppe.