Anhang A

Datenbank-programmierung von A bis Z

Kaum ein anderer Bereich der Visual-Basic-Programmierung dürfte so viele Abkürzungen, Synonyme und Begriffe zu bieten haben wie die Datenbankprogrammierung. Das Glossar in diesem Anhang soll Ihnen beim Einarbeiten in die Datenprogrammierung den Überblick erleichtern.

Übrigens, sollten Sie einen Begriff vermissen: Ein E-Mail an den Autor genügt (die E-Mail-Adresse lautet *peterm@activetraining.de*, bitte im Betreff nur »Datenbankbegriff gesucht:Begriff« angeben, wobei der Platzhalter »Begriff« für den gesuchten Begriff steht, da die Beantwortung automatisch erfolgen soll)[1].

1NF – 1. Normalform einer relationalen Datenbank. In dieser Form darf die Tabelle keine Felder besitzen, die nicht atomar sind, d.h. mehrere Werte beinhalten.

2NF – 2. Normalform einer relationalen Datenbank. In dieser Form darf die Tabelle keine Felder enthalten, die nicht vom Schlüssel (oder einem Teil des Schlüssels) abhängig sind.

3NF – 3. Normalform einer relationalen Datenbank. In dieser Form darf die Tabelle keine Nicht-Schlüssel-Felder enthalten, die den Wert eines anderen Wertes bestimmen, der ebenfalls kein Teil des Schlüssels ist.

1 Das erledigt ein kleines Visual-Basic-Programm, das den Outlook-Posteingang durchgeht, die Suchbegriffe analysiert, eine Datenbankabfrage in meinem »Glossar-Server« durchführt und den Antworttext als HTML zurückschickt. Dieses Visual-Basic-Programm gibt es aber leider noch nicht.

Abfrage – SQL-Kommando, das einen oder mehrere Datensätze zurückgibt. Eine Abfrage kann per Text an die Datenbank geschickt werden oder in der Datenbank enthalten sein.

Access-Datenbank – Anderer Name für eine mit Microsoft Access (oder allgemein über die DAOs) erstellte Datenbank. Die Verwaltung einer Access-Datenbank erfolgt über die Jet-Engine als DBMS.

AccessSQL – SQL-Dialekt, der von der Jet-Engine verwendet wird. Access SQL unterscheidet sich von Standard-SQL (ANSI 92-SQL) durch ein paar Kleinigkeiten, wie unterschiedliche Platzhalter, aber auch bei einigen SQL-Kommandos (z.B. UNION).

ADO – Active (X) Data Objects. Objektschnittstelle für den Zugriff auf OLE DB-Datenquellen. ADO wurde ursprünglich in der Version 1.0 mit dem Internet Explorer 4.0 eingeführt, in der Version 1.5 stark erweitert und in der Version 2.0 zum Bestandteil von Visual Basic 6.0. Die aktuelle ADO-Version lautet 2.1, die mit Internet Explorer 5.0 und SQL-Server 7.0 installiert wird. Office 2000 enthält sogar bereits das Service Pack 1 (SP1) von ADO 2.1 (eine Version 2.5, wie es manchmal zu lesen ist, wird es also doch nicht sein). ADO ist unabhängig von einer bestimmten Umgebung und kann auch von Visual Basic 5.0, Visual C++, Visual J++ oder etwa VBScript eingesetzt werden. Mittelfristig wird es die Datenbankschnittstellen DAO und RDO (und damit ODBC) ablösen.

ADOX – Mit Microsoft Access 2000 eingeführte Objektschnittstelle, die (als Ergänzung zu den ADOs) den Zugriff auf die Datendefinitions- und Datensicherheitsmerkmale der Jet-Engine ermöglicht. Die ADOX-Objekte werden in diesem Buch noch nicht behandelt[1].

Client/Server – Weit verbreitetes Konzept in der DV/IT-Welt, das das Zusammenspiel zweier Software-Komponenten unabhängig von ihrem physikalischen Standort beschreibt: Ein Client stellt (beliebige) Dienste zur Verfügung, die von einem Server benutzt werden. In der Datenbankwelt ist der Server das DBMS, der Client ein Programm, z.B. ein Visual-Basic-Programm.

Cursor – Datenstruktur, die die Verwaltung einer Datensatzgruppe mit einem Datensatzzeiger auf dem Client ermöglicht. Ohne Cursor ist z.B. kein Scrollen der Datensatzgruppe möglich. Welche Cursortypen zur Verfügung stehen, hängt von dem Datenbanktreiber ab. Die Auswahl eines Cursors richtet sich nach den Gesichtspunkten Performance und Komfort (beim Zu-

1 Sie sind normalerweise auch nicht das, auf was sich Datenbankneulinge sofort stürzen werden.

griff), die auf einen Nenner gebracht werden müssen (Datenbankprofis sagen daher, der beste Cursor ist kein Cursor, doch das läßt sich in der Praxis nur selten realisieren).

DAO – Data Access Objects. Aus Objekten bestehende Datenbankschnittstelle für den Zugriff auf die Jet-Engine und seit der Version 3.5 auch auf ODBC-Datenbanken (ODBCDirect-Modus). Die aktuelle DAO-Version lautet 3.51 bzw. 3.6 (weitere Versionen dürfte es vermutlich nicht geben, da ADO die bevorzugte Datenbankschnittstelle sein soll). Doch man soll bekanntlich nie nie sagen[1].

Datensteuerelement – Steuerelement, mit dem die direkte Anbindung an eine Datensatzgruppe möglich ist, und das daher auch ein Recordset-Objekt zur Verfügung stellt. Visual Basic 6.0 bietet drei Datensteuerelemente: Das »normale« Datensteuerelement, das auf DAO und der Jet-Engine basiert, das RDO-Steuerelement, das auf der ODBC-API basiert (nur Enterprise-Edition), und das neue ADO-Datensteuerelement, das auf OLE DB und ADO basiert. Die Verwendung des ADO-Datensteuerelements bietet im Vergleich zur Bindung an eine Datenumgebung nur wenige Vorteile. Außerdem können nicht alle Steuerelemente per ADO/OLE DB gebunden werden.

dBase – Von der Firma Ashton Tate in den frühen achtziger Jahren entwickelte Datenbank, die ursprünglich unter dem Betriebssystem CP/M auch auf Heimcomputern (C64 oder Apple II) lief[2]. In den späten achtziger Jahren wurde Ashton Tate von der Firma Borland (die inzwischen Inprise heißt) übernommen (aus dBase wurde Visual dBase, das an alte Erfolge jedoch nicht mehr anknüpfen konnte). Weniger als das DBMS spielt in der Datenbank das dBase- bzw. allgemein xBase-Format noch eine Rolle, das z.B. von Visual FoxPro verwendet wird. Auch die Jet-Engine ist in der Lage, das dBase-Format sowohl zu lesen als auch zu schreiben.

DBMS – DataBase Management System. Allgemeine Bezeichnung für eine Anwendung, mit der sich Datenbanken erstellen, administrieren und bearbeiten lassen.

DSN – Data Source Name. Wichtig für den Zugriff auf ODBC-Datenbanken, denn der DSN enthält die Verbindungsinformationen für den Zugriff auf die ODBC-Datenbank. DSN-Informationen werden entweder in der Re-

---

1 An anderer Stelle habe ich auch behauptet, daß Windows 98 der letzte auf dem »DOS-Kernel« basierende Vertreter der Windows-Familie sein wird. Später kam wieder einmal alles anders.

2 Der Firmenname Ashton Tate dürfte den meisten Lesern und Leserinnen vermutlich kein Begriff mehr sein. Ich habe in den achtziger Jahren eine Zeitlang in der Nähe des Firmensitzes in Culver City gewohnt, was mich allerdings dBase nicht sehr viel näher gebracht hat.

gistry oder in einer Datei (Erweiterung *.dsn*) gespeichert. Sie können auch vor dem Zugriff auf die Datenbank direkt übergeben werden, so daß kein DSN benötigt wird.

Dynaset – Spezieller Cursortyp der DAOs für die Jet-Engine. Ein Dynaset stellt eine Menge der Schlüssel aller Datensätze der Datensatzgruppe dar. Werden von anderen Benutzern Datensätze an die dem Dynaset zugrundeliegenden Tabellen angehängt oder aus diesen entfernt, werden diese Änderungen erst dann sichtbar, wenn der Dynaset neu geöffnet wird. Das ADO-Pendant ist der Keyset-Cursor.

Fremdschlüssel – Feld (Spalte) in einer Tabelle, die sich auf den Primärschlüssel einer anderen Tabelle bezieht. Ein Beispiel für einen Fremdschlüssel ist das Feld Kundennummer in einer Auftragstabelle, die als Fremdschlüssel auf den Primärschlüssel (d.h. auf das Kundennummernfeld) in der Tabelle mit den Kundendaten verweist.

Index – Sortierreihenfolge einer Tabelle, die in der Datenbank gespeichert wird, und die die Zugriffsgeschwindigkeit erhöhen soll. Indizes werden in der Regel beim Anlegen der Tabelle auf einzelne Felder (in der Regel den Primärschlüssel) gesetzt. Ein Nachteil von Indizes ist, daß z.B. beim Einfügen von Datensätzen die Änderung auch in der internen Indextabelle vorgenommen werden muß (zu viele Indexfelder können daher auch bremsen).

ISAM – Indexed Sequential Access Method. Inzwischen veraltetes Organisationsprinzip einer Datenbank, bei der alle Datenbankoperationen auf der Basis von Indextabellen durchgeführt werden.

Jet-Engine – DBMS von Microsoft, das fester Bestandteil von Microsoft Access (ab Version 2000 allerdings nur noch optional) und als optionales DBMS Teil von Visual Basic ist. Visual Basic 6.0 enthält die Version 3.51, mit Microsoft Access 2000 und den *Microsoft Data Access Components* (MDAC) 2.1 wird die Jet-Engine 4.0 ausgeliefert. Langfristig wird die Bedeutung der Jet-Engine durch die mit Office 2000 eingeführte Microsoft Desktop Engine (MDE) vermutlich spürbar nachlassen.

Locking – Verfahren, das den gleichzeitigen Zugriff mehrerer Benutzer (Prozesse) auf einen Datensatz regelt. Wird ein Datensatz editiert oder aktualisiert, wird er gesperrt (Locking), so daß andere Prozesse seinen Inhalt nicht ändern können. Locking ist performancekritisch, da es zum einen den Zugriff anderer Prozesse verlangsamt, zum anderen einen »Verwaltungsoverhead« nach sich zieht.

MDB-Datenbank – Microsoft DataBase. Dateierweiterung, die alle Access-Datenbanken aufweisen.

MSDE – Microsoft Desktop Engine. Die MSDE ist Bestandteil von Microsoft Access 2000 und damit »brandneu«. Es handelt sich um die Desktop-Version des Microsoft SQL-Servers 7.0 (die auch unter Windows 95/98 läuft) und stellt eine Alternative zur Jet-Engine dar. Der Zugriff erfolgt über ADO und den OLE DB-Provider für den SQL-Server 7.0.

Normalisierung – Vom »Datenbank-Papst« E. F. Codd 1970 aufgestellte (und Jahre später erweiterte) Regeln, deren Anwendung die Redundanz in Tabellen minimieren soll.

NULL – Spezieller Wert eines Feldes, der anzeigt, daß das Feld keinen Inhalt besitzt. Die Zuweisung eines *NULL*-Wertes an eine VBA-Variable, die nicht vom allgemeinen Typ *Variant* ist, führt zu einem Laufzeitfehler. Dafür gibt es in VBA die *IsNull*-Funktion, die das vorher abprüft.

ODBC – Open Database Connectivity. Von Microsoft Anfang der neunziger Jahre ins Leben gerufener Standard, der den Zugriff auf Datenbank-Management-Systeme verschiedener Hersteller vereinheitlichen soll. ODBC wird durch OLE DB und ADO in den kommenden Jahren an Bedeutung verlieren.

ODBCDirect – Spezieller Modus der DAOs, in dem ein direkter Zugriff (über die ODBC-API) auf ODBC-Datenbanken möglich ist. Der Vorteil gegenüber dem normalen Jet-Modus ist die höhere Performance und die bessere Anlehnung an das ODBC-Modell.

OLE DB – Von Microsoft entwickelter universeller Zugriffsmechanismus, der vollständig auf dem *Component Object Modell* (COM) basiert. OLE DB geht von allgemeinen Datenquellen aus und nicht ausschließlich von relationalen Datenbanken. Alles, was Daten enthält, kann (den passenden OLE DB-Treiber vorausgesetzt, der auch in Visual Basic programmiert werden kann) über OLE DB angesprochen werden. Der Zugriff auf OLE DB von Visual Basic aus erfolgt über ADO.

Oracle SQL-Server – Sehr leistungsfähiger Datenbankserver der Firma Oracle (*www.oracle.com*), der als Alternative zum Microsoft SQL-Server in Frage kommt. Der Zugriff auf eine Oracle-Datenbank kann wahlweise über ADO (OLE DB), DAO (ODBC), RDO, den direkten Aufruf von ODBC-API-Funktionen oder über die von Oracle *entwickelten Oracle Objects for OLE* (OO40) erfolgen. Der mit Visual Basic 6.0 ausgelieferte OLE DB-Provider für Oracle 7.x-Server stammt von der Firma *Intersolv* (*www.merrant.com*).

Primärschlüssel – Ein oder mehrere Felder (Spalten) in einer Tabelle, die einen Datensatz eindeutig identifizieren. Ein Beispiel für einen Primärschlüssel ist das Feld Kundennummer in der Kundenstammdatentabelle, auf

die die Datensätze einer Auftragstabelle mit ihren Kundennummernfeldern, die als Fremdschlüssel fungieren, verweisen.

RDBMS – Relationales DBMS mit dem sich relationale Datenbanken erstellen und bearbeiten lassen. Die meisten modernen DBMS sind auch RDBMS. Microsoft Access nimmt in dieser Beziehung eine Art »Zwitterposition« an, da es zwar das relationale Datenbankmodell unterstützt, aber aufgrund des Umstands, daß die Datenbanken ISAM-typisch in Dateien gehalten werden, ein wenig aus dem Rahmen fällt.

RDO – Remote Data Objects. Mit Visual Basic 5.0 eingeführte Objektschnittstelle für den Zugriff auf ODBC-Datenbanken über die ODBC-API. Die RDOs kombinieren Leistungsfähigkeit mit Performance. Ab Visual Basic 6.0 werden sie durch die ADOs abgelöst.

Recordlocking – Siehe Locking.

Recordset – Datensatzgruppe, d.h. eine Gruppe von Datensätzen, die z.B. aus einer Tabelle stammen oder von einer SQL-Abfrage zurückgegeben wurden (anders als bei DAO müssen bei ADO alle Datensätze aus einer Datenquelle stammen). Sowohl bei ADO als auch bei DAO werden Recordsets über ein *Recordset*-Objekt angesprochen.

Relationale Datenbanken – Organisationsprinzip von Datenbanken, bei denen die Daten (oft unter Anwendung der Normalisierungsregeln, wenngleich dies keine Voraussetzung ist) auf Tabellen verteilt sind, zwischen denen Beziehungen (Relationen) existieren. Auch wenn das relationale Datenbankmodell, das noch aus den siebziger Jahren stammt, für ca. 90% aller eingesetzten Datenbanken die Grundlage sein dürfte, dürfte es in den kommenden Jahren nach und nach durch Objektdatenbanken ersetzt werden (oder um es einmal ein wenig »prophetisch« zu formulieren: Im Jahre 2099 wird es mit an Sicherheit grenzender Wahrscheinlichkeit nicht mehr das dominierende Modell sein).

Relationen – Relationen legen Beziehungen zwischen zwei Tabellen fest, die über Felder definiert werden. Bei der Jet-Engine werden Relationen dazu benutzt, die referentielle Integrität sicherzustellen (Lösch- und Update-Verfolgung).

Remote-Datenbanken – Allgemeiner Name für Datenbanken, die nicht als Dateien vorliegen (wie z.B. Access-Datenbanken), sondern über einen Datenbanktreiber (OLE DB oder ODBC) angesprochen werden. Remote-Datenbanken befinden sich in der Regel auf einem anderen PC im Netzwerk, wenngleich dies keine Voraussetzung ist. Ein Beispiel für eine Remote-Datenbank ist der Microsoft SQL-Server.

Replikation – Replikation ist eine Eigenschaft eines DBMS, wie z.B. Microsoft Access, Microsoft SQL Server oder MSDE, und bedeutet, daß Teile der Datenbank mit einer anderen Datenbank synchronisiert werden.

Schlüssel – Auf der Grundlage von Schlüsseln werden Relationen zwischen zwei Tabellen hergestellt (z.B. über Primär- und Fremdschlüssel). Gleichzeitig werden auf Schlüssel Indizes gelegt, um die Zugriffsgeschwindigkeit zu steigern.

Sekundärindex – Index, bei dem es sich um den Primärindex (der Index, der auf dem Primärschlüssel liegt) handelt.

Snapshot – Spezieller Cursortyp der DAOs für die Jet-Engine. Ein Snapshot entspricht bezüglich seiner Möglichkeiten einem Dynaset, nur daß die Datensätze lediglich gelesen werden können. Außerdem werden beim Snapshot die Daten der Datensätze (und nicht nur die Schlüssel) in den Arbeitsspeicher geladen.

SQL – Allgemeine Datenbankabfrage- und definitionssprache, die 1974 in einem Forschungszentrum bei IBM von Donald Chamberlin entwickelt wurde und von praktisch jedem DBMS unterstützt wird. SQL ist standardisiert (ANSI 92), wird aber von verschiedenen DBMS teilweise nicht hundertprozentig befolgt (es gibt viele Erweiterungen). Die Jet-Engine verwendet mit AccessSQL einen Dialekt, der in den meisten wichtigen Bereichen mit ANSI 92 übereinstimmt und mit der Jet-Engine 4.0 praktisch »compliant« (übereinstimmend) sein soll.

SQL-Server – Produktname eines DBMS von Microsoft. Der SQL-Server wurde ursprünglich von der Firma Sybase (damals noch für das Betriebssystem OS/2) entwickelt und ab der Version 4.x von Microsoft alleine weiterentwickelt. Die aktuelle Version lautet 7.0.

Stored Procedure. – Gespeicherte Prozedur. Mehr oder weniger umfangreiche Datenbankoperationen, die in einer Microsoft SQL-Server- oder MSDE-Datenbank gespeichert werden. Stored Procedures, die in T SQL erstellt werden, werden sehr schnell ausgeführt.

Transaktion – Bilden den Rahmen, in dem eine Gruppe von Datenbankoperationen »sicher« ausgeführt werden kann. Stellt das Programm am Ende fest, daß ein Problem auftrat, wird die Transaktion rückgängig gemacht und die Datenbank dadurch in ihren ursprünglichen Zustand zurückversetzt.

T-SQL – Transact SQL. SQL-Dialekt, der vom Microsoft SQL-Server (und dem Sybase SQL-Server) sowie von der MSDE verwendet wird. T-SQL geht

weit über Standard-SQL hinaus und enthält z.B. Entscheidungsbefehle und Variablen.

UDA – Universal Data Access. Der Name der aktuellen (Stand: 4/99) Microsoft-Datenbankstrategie, die auf OLE DB basiert.

UDL – Universal Data Link. In UDL-Dateien werden die Verbindungsinformationen für eine OLE DB-Datenquelle gespeichert. UDL-Dateien werden durch Anklicken eines Ordnerfensters oder des Desktops mit der rechten Maustaste und Auswahl der Einträge NEU und DATA LINK erstellt. Sie sind allerdings optional, da die Verbindungsinformationen auch direkt (z.B. beim Öffnen eines *Recordset*-Objekts) angegeben werden können.

Visual Data Manager – Visual Basic-Add-In, das einen direkten Zugriff über alle von der Jet-Engine 3.51 unterstützten Datenbankformate erlaubt. Der Visual Data Manager unterstützt daher nicht OLE DB und ADO.

Anhang B

Die Active Data Objects (ADO)

Dieser Anhang gibt eine Kurzübersicht über die *Active (X) Data Objects*, die in Kapitel 5 ausführlicher vorgestellt werden[1]. Dieser Anhang soll in erster Linie als eine Orientierung dienen. Eine ausführliche Beschreibung aller ADO-Objekte mit ihren Eigenschaften, Methoden und Ereignissen finden Sie (wie immer) in der MSDN-Hilfe oder im Internet unter *http://msdn.microsoft.com*.

B.1 Das Connection-Objekt

Über das *Connection*-Objekt wird eine Verbindung zu einer Datenquelle (z.B. einer Access-Datenbank) hergestellt.

B.1.1 Die Eigenschaften des Connection-Objekts

Die wichtigsten Eigenschaften des *Connection*-Objekts sind *Connection-String* und *Provider*, durch die festgelegt wird, mit welcher Datenquelle die Verbindung hergestellt werden soll.

---

1 Ob es nun ActiveX oder nur Active heißen soll, dürfte selbst bei Microsoft nicht unstrittig sein. Ich finde, daß ActiveX zu stark an AkteX erinnert und man den Opponenten der Microsoft-Produktstrategie nicht zu leichte Angriffsflächen bieten sollte (Vorsicht, leichte Ironie – der Hinweis ist an dieser Stelle wichtig, da es in einer Fußnote keine Fußnote geben kann).

Tabelle B.1: Die Eigenschaften des Connection-Objekts

Eigenschaft	Bedeutung
Attributes	Gibt verschiedene und sehr spezielle Merkmale der Verbindung an, die das Verhalten bei Transaktionen betreffen, oder legt diese fest.
CommandTimeOut	Gibt die Dauer in Sekunden an, die auf die Ausführung eines Kommandos gewartet wird, oder legt diese fest. Die Voreinstellung beträgt 30 Sekunden.
ConnectionString	Enthält die Zeichenkette, über die die Verbindung hergestellt wurde.
ConnectionTimeOut	Gibt die Dauer in Sekunden an, die auf einen Verbindungsaufbau gewartet wird, oder legt diese fest. Die Voreinstellung beträgt 30 Sekunden.
CursorLocation	Gibt an, ob es sich um einen Client-Cursor (*adUseClient*) oder einen serverseitigen Cursor (*adUseServer*) handelt, oder legt dies fest. Die Einstellung *adUseNone* dient nur der Abwärtskompatibilität.
DefaultDatabase	Gibt die Standarddatenbank der Verbindung an (ODBC-Zugriff).
Errors	Ermöglicht einen Zugriff auf die *Errors*-Auflistung, die mögliche Fehler in Gestalt von *Error*-Objekten enthält.
IsolationLevel	Gibt die sogenannte Isolierungsebene für die Verbindung an oder legt sie fest. Die Isolierungsebene bezieht sich auf die Art und Weise, wie mehrere Transaktionen voneinander isoliert werden.
Mode	Setzt oder ermittelt die aktuellen Zugriffsberechtigungen für das Ändern von Daten in der Datenquelle.
Properties	Ermöglicht einen Zugriff auf die *Properties*-Auflistung, die spezifische Eigenschaften in Gestalt von *Property*-Objekten enthält.
Provider	Gibt den Namen des OLE DB-Providers an.
State	Gibt den aktuellen Zustand der Verbindung an (z.B. *adStateOpen* – 1–offen).
Version	Gibt die Versionsnummer der ADO-Objekte zurück (z.B. 2.0). Die Versionsnummer des Providers steht in der *Properties*-Auflistung zur Verfügung.

B.1.2 Die Methoden des Connection-Objekts

Die wichtigsten Methoden sind *Open* und *Close*, durch die die Verbindung geöffnet bzw. geschlossen wird. Um Laufzeitfehler zu vermeiden, sollte vor dem Aufruf von *Open* die *State*-Eigenschaft abgefragt werden.

Methode	**Bedeutung**
Execute	Führt die folgende Abfrage, die SQL-Anweisung oder die gespeicherte Prozedur aus. Der Rückgabewert kann ein *Recordset*-Objekt sein.
Open	Öffnet das *Connection*-Objekt.
OpenSchema	Öffnet ein sogenanntes Datenbankschema, das Strukturinformationen (z.B. Tabellenstruktur) über die Datenbank enthält.
RollbackTrans	Macht eine Transaktion rückgängig.

Tabelle B.2: Die Methoden des Connection*-Objekts*

Die Werte der *adState*-Konstanten können auch aufaddiert sein, so daß z.B. der Wert für *adStateClosed* (wenn man es genau machen will) über eine logische Verknüpfung ausgefiltert werden muß.

AdState	**Wert**
AdStateClosed	0
AdStateOpen	1
AdStateConnecting	2
AdStateExecuting	4
AdStateFetching	8

Tabelle B.3: Die Werte der adState*-Konstanten*

B.1.3 Die Ereignisse des *Connection*-Objekts

Neben dem *Recordset*-Objekt kann auch das *Connection*-Objekt auf Ereignisse reagieren, die aber nur relativ selten benötigt werden.

Tabelle B.4: Die Ereignisse des Connection-Objekts

Ereignis	Wird aufgerufen ...
BeginTransComplete	Nach einer *BeginTrans*-Operation.
CommitTransComplete	Nach einer *CommitTrans*-Operation.
ConnectComplete	Nachdem eine Verbindung hergestellt wurde.
Disconnect	Nachdem eine Verbindung beendet wurde.
ExecuteComplete	Nachdem die Ausführung eines Kommandos beendet wurde.
InfoMessage	Wenn der Provider nach dem Herstellen der Verbindung etwas mitteilen möchte.
RollbackTransComplete	Nach einer *RollbackTrans*-Operation.
WillConnect	Vor dem Beginn eines Verbindungsaufbaus.
WillExecute	Vor der Ausführung eines Kommandos.

B.2 Das *Recordset*-Objekt

Das *Recordset*-Objekt ist das zentrale ADO-Objekt, denn es stellt die Daten(schätze) zur Verfügung.

B.2.1 Die Eigenschaften des *Recordset*-Objekts

Keine der recht zahlreichen Eigenschaften muß vor dem Öffnen eines *Recordset*-Objekts gesetzt sein, da das *Recordset*-Objekt mit Voreinstellungen arbeitet.

Tabelle B.5: Die Eigenschaften des Recordset-Objekts

Eigenschaft	Bedeutung
AbsolutePage	Legt die Seite fest oder gibt diese an, auf der sich der aktuelle Datensatz befindet. Über die *PageSize*-Eigenschaft wird das *Recordset* in logische Seiten unterteilt. Muß vom Provider unterstützt werden.
AbsolutePosition	Legt die Position (Ordinalposition) des Datensatzzeigers fest bzw. gibt diesen zurück. Bei dieser Eigenschaft gibt es keine Garantie, stets die gleiche Nummer für einen Datensatz zu erhalten.

Tabelle B.5: Die Eigenschaften des Recordset-Objekts (Fortsetzung)

Eigenschaft	Bedeutung
ActiveCommand	Gibt das *Command*-Objekt an, das das *Recordset* erstellt hat.
ActiveConnection	Gibt das *Connection*-Objekt an, über das *Recordset* erstellt wurde.
BOF	Ist *True*, wenn sich der Datensatzzeiger vor dem ersten Datensatz der Datensatzgruppe befindet.
Bookmark	Legt ein Lesezeichen auf den aktuellen Datensatz an oder gibt es zurück. Über diese Eigenschaft sollte das Ansteuern eines bestimmten Datensatzes durchgeführt werden.
CacheSize	Legt bei serverseitigen Cursortypen fest, wie viele Datensätze im Zwischenspeicher gehalten werden. Bei clientseitigen Cursortypen werden alle Datensätze übertragen.
CursorLocation	Gibt an, ob es sich um einen Client-Cursor (*adUseClient*) oder einen serverseitigen Cursor (*adUseServer*) handelt, oder legt dies fest. Die Einstellung *adUseNone* dient nur der Abwärtskompatibilität.
CursorType	Legt den zu verwendenden Cursortyp fest oder gibt ihn an. Zur Auswahl stehen: *adOpenForwardOnly*, *adOpenKeyset*, *adOpenDynamic* und *adOpenStatic*.
DataMember	Weist dem Recordset eine Datensatzgruppe von jenem Objekt zu, das über die *DataSource*-Eigenschaft festgelegt wird.
DataSource	Legt das Objekt fest, das die Datensatzgruppe enthält, die im Recordset dargestellt werden soll.
EditMode	Gibt den Bearbeitungsstatus des aktuellen Datensatzes an. Zur Auswahl stehen: *adEditNone*, *adEditInProgress*, *adEditAdd* und *adEditDelete*.
EOF	Ist *True*, wenn sich der Datensatzzeiger nach dem letzten Datensatz der Datensatzgruppe befindet.
Fields	Ermöglicht einen Zugriff auf die *Fields*-Auflistung, die die Felder des Datensatzes umfaßt (Default).
Filter	Legt ein Kriterium fest, durch das nur bestimmte Datensätze im Recordset sichtbar sind.
Locktype	Gibt die Art der Zugriffssperre zurück oder legt sie fest. Zur Auswahl stehen *adLockReadOnly*, *adLockPessimistic*, *adLockOptimistic* und *adLockBatchOptimistic*.
MarshalOptions	Gibt an oder legt fest, auf welche Weise die Datensätze durch das sogenannte Marshalling (das Übertragen von Informationen über Prozeßgrenzen hinweg) bei einem clientseitigen Recordset zurückgegeben werden. Zur Auswahl stehen *adMarshalAll* und *adMArshalModifiedOnly*.

Tabelle B.5: Die Eigenschaften des Recordset-Objekts (Fortsetzung)

Eigenschaft	Bedeutung
MaxRecords	Gibt die maximale Anzahl an Datensätzen an oder legt die (bei einem geschlossenen Recordset) fest, die von einer Abfrage zurückgegeben werden sollen.
PageCount	Gibt die Anzahl der Datenseiten an.
PageSize	Legt fest, aus wie vielen Datensätzen eine Seite besteht. Die Voreinstellung ist 10.
Properties	Ermöglicht einen Zugriff auf die *Properties*-Auflistung, die spezifische Eigenschaften in Gestalt von *Property*-Objekten enthält.
RecordCount	Gibt die aktuelle Anzahl an Datensätzen im Recordset an. Muß vom Provider unterstützt werden und hängt vom Cursortyp ab (bei *ForwardOnly*-Cursorn geht es z.B. nicht). Gibt ansonsten –1 zurück.
Sort	Legt eine Sortierreihenfolge für das Recordset fest oder gibt diese zurück. Es handelt sich um einen Feldnamen, dem die Zusätze *ASCENDING* oder *DESCENDING* folgen.
Source	Gibt die Quelle des Recordsets an oder legt diese fest (z.B. eine Tabelle oder ein SQL-Kommando).
State	Gibt den aktuellen Zustand des Recordsets an (siehe Tabelle B.3).
Status	Gibt den Status des aktuellen Datensatzes in Bezug auf Stapelaktualisierungen oder andere Vorgänge an.
StayInSync	Legt in einem hierarchischen Recordset fest, ob die übergeordnete Zeile geändert wird, wenn die der Gruppe zugrundeliegenden untergeordneten Datensätze geändert werden.

B.2.2 Die Methoden des *Recordset*-Objekts

Das *Recordset*-Objekt besitzt ebenfalls eine Vielzahl von Methoden, von denen *Open*, *Find*, *Update* und die verschiedenen *Move*-Methoden die wichtigsten sind.

Tabelle B.6: Die Methoden des Recordset-Objekts

Methode	Bedeutung
AddNew	Fügt einen neuen Datensatz hinzu, wobei der neue Datensatz bis zum Aufruf der *Update*-Methode zum aktuellen Datensatz wird. Gebundene Steuerelemente werden dadurch automatisch geleert (Probleme gibt es, wenn das Recordset noch keine Datensätze enthält).
Cancel	Bricht die Ausführung eines anstehenden asynchronen Aufrufs der *Execute*- oder *Open*-Methode ab.

Tabelle B.6: Die Methoden des Recordset-Objekts (Fortsetzung)

Methode	Bedeutung
CancelBatch	Bricht eine anstehende Stapelaktualisierung (Batch-Update) ab.
CancelUpdate	Bricht die Änderung eines Datensatzes ab. Alle durchgeführten Änderungen vor dem Aufruf der *Update*-Methode werden verworfen.
Clone	Erstellt eine »Kopie« des Recordsets, das schreibgeschützt sein kann und mit einer eigenen Bookmark arbeiten kann.
Close	Schließt ein offenes Recordset. Dadurch werden die belegten Systemressourcen freigegeben.
CompareBookmarks	Vergleicht zwei einzelne Lesezeichen und gibt einen Wert für ihr relatives Verhältnis zurück.
Delete	Löscht den aktuellen Datensatz. Wird als Parameter *adAffectGroup* übergeben, werden alle Datensätze der über einen Filter ausgewählten Datensätze gelöscht.
Find	Positioniert den Datensatzzeiger auf den nächsten Datensatz, der dem angegebenen Kriterium (bestehend aus dem Vergleich mit genau einem Feld) entspricht.
GetRows	Ruft mehrere Datensätze aus dem Recordset ab und weist sie einer *Variant*-Variablen in Form eines zweidimensionalen Arrays zu, wobei der zweite Index die Datensätze, der erste die Felder auswählt.
GetString	Gibt das komplette Recordset als eine Zeichenfolge zurück.
Move	Bewegt den Datensatzzeiger um keine, eine oder mehrere Positionen. *Move 0* ist ein beliebter Trick, um den aktuellen Datensatz neu einzulesen.
MoveFirst	Bewegt den Datensatzzeiger auf den ersten Datensatz im Recordset.
MoveLast	Bewegt den Datensatzzeiger auf den letzten Datensatz im Recordset.
MoveNext	Bewegt den Datensatzzeiger auf den nächsten Datensatz im Recordset.
MovePrevious	Bewegt den Datensatzzeiger auf den vorhergehenden Datensatz im Recordset.
Open	Öffnet ein geschlossenes Recordset.
Requery	Führt die dem Recordset zugrundeliegende Abfrage neu aus.
Resync	Aktualisiert die Daten des Recordsets auf der Grundlage der zugrundeliegenden Datenbank.

Tabelle B.6: Die Methoden des Recordset-Objekts (Fortsetzung)

Methode	Bedeutung
Save	Speichert den Recordset in einer lokalen Datei (XML- oder ADTG-Format). Die Datei bleibt so lange geöffnet, wie auch das Recordset geöffnet ist.
Supports	Gibt an, ob der Recordset bestimmte Eigenschaften (wie z.B. AddNew) unterstützt.
Update	Speichert alle Änderungen, die am aktuellen Datensatz seit dem letzten Aufruf der *Update*-Methode durchgeführt wurden.
UpdateBatch	Speichert alle anstehenden Stapelaktualisierungen in der Datenbank.

B.2.3 Die Ereignisse des *Recordset*-Objekts

Das *Recordset*-Objekt verfügt über insgesamt elf und damit eine recht breite Auswahl an Ereignissen, denen aus diesem Grund mit Kapitel 13.8 ein eigener Abschnitt gewidmet wird.

B.3 Das *Field*-Objekt

Das *Field*-Objekt steht für ein eigenes Datenbankfeld, so daß alle Eigenschaften etwas mit dem Dateninhalt zu tun haben.

B.3.1 Die Eigenschaften des *Field*-Objekts

Die wichtigste Eigenschaft des *Field*-Objekts ist *Value*, die, da es eine *Default*-Eigenschaft ist, auch entfallen kann.

Tabelle B.7: Die Eigenschaften des Field-Objekts

Eigenschaft	Bedeutung
ActualSize	Gibt die tatsächliche Länge eines Feldes in Byte (*Long*-Wert) zurück.
Attributes	Gibt verschiedene und sehr spezielle Merkmale des Feldes an (z.B., daß das Feld NULL-Werte zuläßt).
DataFormat	Gibt das Format des Feldes an oder legt es fest.
DefinedSize	Gibt die definierte Größe des Feldes an.
Name	Legt den Feldnamen fest oder gibt ihn an.
NumericScale	Gibt die Anzahl an Nachkommastellen bei Feldern mit numerischem Inhalt an (*Byte*-Wert).

Eigenschaft	Bedeutung
OriginalValue	Gibt den Wert des Feldes an, bevor Änderungen vorgenommen wurden (zusammen mit *UnderlyingValue* und *Value* kann ein Feld gleichzeitig drei Werte besitzen).
Precision	Gibt die Genauigkeit (maximale Anzahl an Ziffern) bei Feldern mit numerischem Inhalt an.
Properties	Ermöglicht einen Zugriff auf die *Properties*-Auflistung, die spezifische Eigenschaften in Gestalt von *Property*-Objekten enthält.
Type	Gibt den Datentyp des Feldes an.
UnderlyingValue	Gibt den aktuellen Wert des Feldes in der Datenbank an.
Value	Gibt den aktuellen Wert des Feldes an (Default).

Tabelle B.7: Die Eigenschaften des Field*-Objekts (Fortsetzung)*

B.3.2 Die Methoden des *Field*-Objekts

Das *Field*-Objekt ist relativ »methodenarm«. Es besitzt lediglich zwei Methoden, mit deren Hilfe sich binäre Feldinhalte (etwa Bitmaps oder größere Memofelder) lesen bzw. schreiben lassen.

Methode	Bedeutung
AppendChunk	Fügt eine Bytefolge in ein Feld ein.
GetChunk	Gibt den Inhalt eines Feldes mit Binärdaten oder langen Memo-Werten teilweise oder vollständig zurück.

Tabelle B.8: Die Methoden des Field*-Objekts*

B.4 Das *Command*-Objekt

Über das *Command*-Objekt werden (SQL-)Abfragen und Stored Procedures (SQL-Server oder Oracle Server) ausgeführt Gegenüber der Ausführung eines SQL-Kommandos mit dem *Recordset*-Objekt bietet es den Vorteil, daß die Abfragen bereits vorverarbeitet vorliegen und dadurch schneller ausgeführt werden.

B.4.1 Die Eigenschaften des *Command*-Objekts

Die wichtigsten Eigenschaften des *Command*-Objekts sind *CommandText* (legt den Inhalt des Objekts fest, z.B. den Namen einer Stored Procedure) und *CommandType*, die grundsätzlich festgelegt werden sollte, da ADO so nicht alle Typen »durchprobieren« muß.

Tabelle B.9: Die Eigenschaften des Command-Objekts

Eigenschaft	Bedeutung
ActiveConnection	Gibt das *Connection*-Objekt an, über das das *Command*-Objekt erstellt wurde.
CommandText	Legt den Kommandotext fest oder gibt ihn an, der dem *Command*-Objekt zugrundeliegt.
CommandTimeOut	Gibt die Dauer in Sekunden an, die auf die Ausführung eines Kommandos gewartet wird, oder legt diese fest. Die Voreinstellung beträgt 30s.
CommandType	Legt den Typ des *Command*-Objekts fest oder gibt ihn an. Zur Auswahl stehen: *adCmdText*, *adCmdTable*, *adCmdTableDirect*, *adCmdStoredProc*, *adCmdUnknown*, *adCommandFile* und *adExecuteNoRecords*.
Name	Legt den Namen des *Command*-Objekts fest oder gibt ihn an.
Parameters	Gibt einen Verweis auf die *Parameters*-Auflistung zurück, in der alle *Parameter*-Objekte enthalten sind (Default).
Prepared	Legt fest oder gibt an, ob eine »kompilierte« Fassung des SQL-Kommandos vor der Ausführung gespeichert werden soll, damit diese beim nächsten Mal schneller ausgeführt werden kann.
Properties	Ermöglicht einen Zugriff auf die *Properties*-Auflistung, die spezifische Eigenschaften in Gestalt von *Property*-Objekten enthält.
State	Gibt den Status des Objekts an (offen oder geschlossen).

B.4.2 Die Methoden des *Command*-Objekts

Die wichtigste Methode des *Command*-Objekts ist *Execute*, denn sie führt das Kommando aus und gibt ein *Recordset*-Objekt zurück, das aber nur dann einen Inhalt besitzt, wenn die Abfrage Datensätze zurückgeben konnte. Es ist wichtig zu beachten, daß das zurückgegebene *Recordset*-Objekt immer vom Typ *ForwardOnly* ist. Über die *CreateParameter*-Methode werden neue *Parameter*-Objekte hinzufügt, was sich relativ elegant bewerkstelligen läßt:

```
With Cmd
    .Parameters.Append .CreateParameter(Name:="Test", _
      Type:=adBSTR)
End With
```

In diesem *With*-Befehl wird der *Parameters*-Auflistung ein *Parameter*-Objekt angehängt, das zuvor über *CreateParameter* angelegt wurde[1].

Methode	Bedeutung
Cancel	Bricht eine anstehende asynchron aufgerufene *Open*- oder *Execute*-Methode (*adAsyncExecute*) ab.
CreateParameter	Legt ein neues *Parameter*-Objekt an, das über die *Append*-Methode der *Parameters*-Auflistung dieser hinzugefügt werden kann.
Execute	Führt das über die *CommandText*-Eigenschaft festgelegte Kommando aus.

Tabelle B.10: Die Methoden des Command-*Objekts*

B.5 Das *Parameter*-Objekt

Das *Parameter*-Objekt ist dazu da, Parameter für *Command*-Objekte zur Verfügung zu stellen, die beim Aufruf einen oder mehrere Parameter erwarten. Das können Abfragen mit Parametern (Jet-Engine) oder sogenannte Stored Procedures (SQL-Server und Oracle-Server) sein.

B.5.1 Die Eigenschaften des *Parameter*-Objekts

Die wichtigste Eigenschaft des *Parameter*-Objekts ist *Value*, die den Wert des Parameters angibt.

Eigenschaft	Bedeutung
Attributes	Gibt verschiedene und sehr spezielle Merkmale eines Parameters an, die die Art der zulässigen Werte betreffen, oder legt diese fest.
Direction	Legt fest, ob es sich bei einem Parameter um einen Eingabe- oder Ausgabeparameter oder um beides handelt.
Name	Legt den Namen des Parameters fest oder gibt ihn an.
NumericScale	Gibt die Anzahl an Nachkommastellen bei Feldern mit numerischem Inhalt an (*Byte*-Wert). Bei *Parameter*-Objekten Schreib-/Lese-Eigenschaft.
Precision	Gibt die Genauigkeit (maximale Anzahl an Ziffern) bei Feldern mit numerischem Inhalt an. Bei *Parameter*-Objekten Schreib-/Lese-Eigenschaft.

Tabelle B.11: Die Eigenschaften des Parameter-*Objekts*

1 Manchmal ist auch VBA zu etwas zu gebrauchen.

Tabelle B.11: Die Eigenschaften des Parameter-Objekts (Fortsetzung)

Eigenschaft	Bedeutung
Properties	Ermöglicht einen Zugriff auf die *Properties*-Auflistung, die spezifische Eigenschaften in Gestalt von *Property*-Objekten enthält.
Size	Gibt die Größe des Wertes in Byte des Parameters an.
Value	Gibt den Wert des Parameters an oder legt ihn fest. Muß bei Datentypen mit variabler Länge (z.B. Strings) gesetzt werden (Default).

B.5.2 Die Methoden des *Parameter*-Objekts

Die einzige Methode des *Parameter*-Objekts ist dazu da, binäre Daten in den Parameter zu übertragen.

Tabelle B.12: Die Methoden des Parameter-Objekts

Methode	Bedeutung
AppendChunk	Füllt den Parameter mit binären Daten. Besitzt das *adFldLong*-Bit im *Attributes*-Feld den Wert *True*, wird dieses Verfahren auch unterstützt.

B.6 Das *Property*-Objekt

Das *Property*-Objekt ermöglicht den Zugriff auf jene (dynamische) Eigenschaften eines Objekts, die nicht über Eigenschaften angeboten werden. Je nach Objekt und Provider können dies bis zu 100 verschiedene Eigenschaften sein.

B.6.1 Die Eigenschaften des *Property*-Objekts

Die wichtigsten Eigenschaften eines *Property*-Objekts sind *Name* und *Value*. Erwähnenswert ist in diesem Zusammenhang auch die *Count*-Eigenschaft der *Properties*-Auflistung, denn sie gibt die Anzahl der verfügbaren Eigenschaften an. Es macht einen Unterschied, ob die *Count*-Eigenschaft vor oder nach dem Öffnen eines Objekts abgefragt wird.

Tabelle B.13: Die Eigenschaften des Property-Objekts

Eigenschaft	Bedeutung
Attributes	Gibt verschiedene Merkmale einer Eigenschaft (z.B. ob diese erforderlich ist oder gesetzt werden darf) an.
Name	Gibt den Namen einer Eigenschaft an.
Type	Gibt den Typ einer Eigenschaft an.
Value	Gibt den Wert einer Eigenschaft an (Default).

B.6.2 Die Methoden des *Property*-Objekts

Das *Property*-Objekt besitzt keine Methoden.

B.7 Das *Error*-Objekt

Die Aufgabe des *Error*-Objekts ist es, Fehler anzuzeigen, die durch Operationen von ADO-Objekten ausgelöst wurden und die keinen Laufzeitfehler auslösen können.

B.7.1 Die Eigenschaften des *Error*-Objekts

Die wichtigsten Eigenschaften des *Error*-Objekts sind *Description* und *Number*.

Eigenschaft	Bedeutung
Description	Enthält die Fehlermeldung (Default).
HelpContext	Gibt die Kontextnummer an, über die der Hilfetext aus der über die *HelpFile*-Eigenschaft festgelegten Hilfedatei zugeordnet wird.
HelpFile	Gibt den Pfad der Hilfedatei an (sofern vorhanden).
NativeError	Gibt die providerspezifische Fehlernummer (meistens negativ, da es sich um eine 32-Bit-Zahl mit gesetztem Bit Nr. 31 handelt) an.
Number	Gibt die Fehlernummer an.
Source	Gibt den Namen des Objekts oder der Anwendung an, die den Fehler ausgelöst hat.
SQLState	Gibt einen fünfstelligen Fehlercode des Providers an. Die Fehlercodes sind über den ANSI-SQL-Standard festgelegt.

Tabelle B.14: Die Eigenschaften des Error-*Objekts*

B.7.2 Die Methoden des *Error*-Objekts

Das *Error*-Objekt besitzt keine Methoden. Die *Clear*-Methode, um den Fehlerzustand zu löschen, gehört zum *Errors*-Objekt.

B.8 Die Ereignisse des *Recordset*-Objekts

Da das *Recordset*-Objekt über elf Ereignisse verfügt, denen eine am Anfang recht unübersichtliche Anzahl an Parametern übergeben wird, werden die Ereignisse in einem eigenen Abschnitt dargestellt.

B.8.1 Die Parameter der ADO-Ereignisse

Jede ADO-Ereignisprozedur wird mit einem Parameter aufgerufen, der über den aktuellen Status der Operation Auskunft gibt. Dieser Parameter heißt *adStatus*. Allerdings ist dieser Parameter nicht nur passiv. Man kann ihm (sozusagen als Rückmeldung an ADO) in den *Will*-Ereignisprozeduren einen anderen Wert geben und dadurch den weiteren Verlauf der Operation beeinflussen.

Der *adStatus*-Parameter

Beim Aufruf einer Ereignisprozedur kann *adStatus* jene Werte annehmen, die in Tabelle 13.15 aufgeführt sind.

Tabelle B.15: Die Werte von adStatus

Wert	Bedeutung
adStatusOK	Die Operation, die das Ereignis verursacht hat, wurde erfolgreich gestartet.
adStatusErrorsOccurred	Die Operation, die das Ereignis verursacht hat, wurde nicht erfolgreich gestartet, oder in einem *Will*-Ereignis wurde ein Abbruch der Operation angefordert. In diesem Fall erhält der *Error*-Parameter weitere Informationen.
adStatusCantDeny	Ein *Will*-Ereignis kann nicht den Abbruch der gestarteten Operation anfordern.

Die folgenden Werte kommen als Rückmeldung für *adStatus* in Frage:

Tabelle B.16: Die Werte für adStatus *bei den* Complete-*Ereignissen*

Wert	Bedeutung
adStatusUnwantedEvent	Es sollen keine weiteren Ereignisse ausgelöst werden (die Operation wird dennoch fortgesetzt).
adStatusCancel	Die begonnene Operation soll abgebrochen werden (Anforderung zum Abbruch oder engl. »request for cancellation«).

Da es anfangs nicht ganz leicht ist, die verschiedenen Situationen, in denen *adStatus* eine Rolle spielt, im Überblick zu behalten, fassen Tabelle B.17 und Tabelle B.18 die in Frage kommenden Konstellationen noch einmal zusammen. Aus den Tabellen geht hervor, wann *adStatus* welche Werte erhalten darf.

Ereignistyp	Mögliche Werte
Will-Ereignis	*adStatusOK*, *adStatusCantDeny*
Complete-Ereignis	*adStatusOK*, *adStatusErrorsOccurred*

Tabelle B.17: Mögliche Werte von adStatus *beim Aufruf einer Ereignisprozedur*

Ereignistyp	Mögliche Werte
Will-Ereignis	*adStatusOK*, *adStatusCancel*, *adStatusUnwantedEvent*
Complete-Ereignis	*adStatusOK*, *adStatusUnwantedEvent*

Tabelle B.18: Mögliche Werte von adStatus *beim Verlassen einer Ereignisprozedur*

Der *adReason*-Parameter

Einige ADO-Ereignisse teilen über den *adReason*-Parameter mit, warum sie ausgelöst wurden. In einigen Situationen wird ein und dasselbe Ereignis mehrfach nacheinander aufgerufen, wobei bei jedem Aufruf ein anderer Grund vorliegt und daher auch ein anderer Wert für *adReason* übergeben wird.

Das Auftreten des *WillChangeRecord*-Ereignisses kann naturgemäß mehrere Gründe haben. Durch Abfragen von *adReason* läßt sich nicht nur der Grund feststellen, es lassen sich auch Ereignisse »ausfiltern», so daß eine begonnene Operation nur dann abgebrochen wird, wenn eine bestimmte Aktion der Auslöser war.

Soll eine Ereignisprozedur nicht weiter aufgerufen werden, muß *adStatus* auf den Wert *adStatusUnwantedEvent* gesetzt werden. Dabei gilt es aber zu beachten, daß sich ein solches »Abschalten« stets nur auf jenes Ereignis bezieht, das über *adReason* gemeldet wird. Sollen alle weiteren Meldungen unterdrückt werden, muß *adStatus* gegebenenfalls mehrfach auf *adStatusUnwantedEvent* gesetzt werden.

Tabelle B.19: Die möglichen Werte für den Parameter adReason

Konstante	Wert
adRsnAddNew	1
adRsnDelete	2
adRsnUpdate	3
adRsnUndoUpdate	4
adRsnUndoAddNew	5
adRsnUndoDelete	6
adRsnRequery	7
adRsnResynch	8
adRsnClose	9
adRsnMove	10
adRsnFirstChange	11
adRsnMoveFirst	12
adRsnMoveNext	13
adRsnMovePrevious	14
adRsnMoveLast	15

Tabelle B.20: Die möglichen Werte für den Parameter adStatus

Konstante	Wert
adStatusOk	1
adStatusErrorsOccured	2
adStatusCantDeny	3
adStatusCancel	4
adStatusUnwantedEvent	5

Anhang C

Die Data Access Objects (DAOs)

Dieser Anhang gibt eine kurze Übersicht über die DAOs, die *Data Access Objects*. Die DAOs sind eine direkte Alternative zu den ADOs, wenn es um den Zugriff auf eine Access-Datenbank geht. Allerdings sind sie veraltet und werden von Microsoft (voraussichtlich) nicht weiterentwickelt. Als Alternative gibt es die *Active Data Objects* (ADOs). Doch da die ADOs sehr allgemein gehalten, die DAOs dagegen speziell auf die Jet-Engine zugeschnitten sind, sprechen zur Zeit (Stand: 4/99) eine Reihe von Gründen dafür, nach wie vor die DAOs zu verwenden. Da aber auch eine Reihe von Gründen gegen die Verwendung der DAOs sprechen und die Schicksalsfrage der Datenbankprogrammierer »DAO oder ADO?« bereits in der Einleitung behandelt wurde (wie Sie sehen, haben in diesem Buch die ADOs »gewonnen«), soll es in diesem Anhang lediglich um eine Übersicht über die DAOs kennen. Zum einen sollte ein erfahrener Visual-Basic-Datenbankprogrammierer alle wichtigen Konzepte und Zugriffsschnittstellen kennen. Zum anderen sollte es für jemanden, der die ADOs gut kennt, kein Problem sein, sich in die DAOs einzuarbeiten.

In diesem Anhang geht es um:

- ✘ Das DAO-Objektmodell
- ✘ Das *Database*-Objekt
- ✘ Das *Recordset*-Objekt
- ✘ Unterschiede zwischen *Dynaset*-, *Snapshot*- und *Table*-Recordsets

- *QueryDef*-Objekte
- Kleine Beispiele zu den DAOs
- Eine kurze Gegenüberstellung DAO zu ADO

C.1 Einbinden der DAOs in ein Projekt

Bei der Jet-Engine handelt es sich um eine Reihe von DLLs, von denen die Dateien *MSJET35.DLL* (ca. 1 Mbyte) und *DAO350.DLL* (569 Kbyte) die wichtigsten sind. Letztere enthält das Objektmodell und seine Komponenten. Damit Sie die Datenobjekte von Visual Basic ansprechen können, muß die Referenz »Microsoft DAO 3.51 Object Library« (oder einen Eintrag mit einer höheren Versionsnummer) über den Menübefehl PROJEKT/VERWEISE eingebunden sein.

C.2 Das DAO-Objektmodell

Auch die DAO-Objekte sind in einem Objektmodell angeordnet. Im Unterschied zu den ADOs ist das Objektmodell umfangreicher und speziell auf die Jet-Engine abgestimmt. Das beginnt bereits an der Spitze des Objektmodells mit dem *DBEngine*-Objekt, das für die Jet-Engine als Ganzes steht. Dieses Objekt besitzt eine bemerkenswerte Eigenschaft: Über die *DefaultType*-Eigenschaft legen Sie fest, ob Sie die DAOs im Jet-Modus (das ist die Voreinstellung) oder im *ODBCDirect*-Modus betreiben möchten. Im ODBCDirect-Modus nutzen Sie die DAO-Objekte lediglich, um direkt auf eine ODBC-Datenbank zuzugreifen. Außerdem stehen hier zusätzliche Objekte zur Verfügung. Die Jet-Engine spielt hier keine aktive Rolle. Der *ODBCDirect*-Modus ist für VBA-Programmierer von Interesse, die bereits ein Datenbankprogramm auf der Basis von DAO erstellt haben, dieses aber nun auf eine ODBC-Datenbank erweitern möchten, ohne den internen ODBC-Modus der Jet-Engine zu nutzen, der sehr viel langsamer ist als der direkte Zugriff über ODBCDirect. Der ODBCDirect-Modus ist in der MSDN-Hilfe sehr gut beschrieben und mit zahlreichen Beispielen unterlegt.

Erwähnenswert ist auch die *Version*-Eigenschaft, die die aktuelle Versionsnummer der Jet-Engine angibt:

```
?DBEngine.Version
3.51
```

Vom *DBEngine*-Objekt leiten sich die *Workspace*-Objekte ab, ebenfalls typisch für die Jet-Engine. Ein Workspace (zu deutsch »Arbeitsbereich«) faßt alle (Sicherheits-)Einstellungen eines einzelnen Anwenders, der auf die Jet-Engine zugreift, zusammen. Wenn Sie zwar die DAOs benutzen, bislang aber noch keinen Workspace angelegt haben, liegt dies daran, daß das Default-Workspace mit dem Benutzernamen *Admin* verwendet wird, wenn Sie kein *Workspace*-Objekt auswählen. Das Anlegen eines *Workspace*-Objekts ist daher nur dann erforderlich, wenn Sie die Standardeinstellungen für einen Zugriff nicht übernehmen möchten. Soll ein Anwender auf die Jet-Engine nur mit Einschränkungen zugreifen können, legen Sie ein *Workspace*-Objekt mit eingeschränkten Rechten an und führen alle Zugriffe für diesen speziellen Anwender mit diesem *Workspace*-Objekt durch. Alle *Workspace*-Objekte werden über die *Workspaces*-Auflistung des *DBEngine*-Objekts zur Verfügung gestellt:

```
?DBEngines.Workspaces.Count
1
```

Das erste Element in der Auflistung ist der sogenannte Default-Workspace:

```
Set Wo = DBEngines.Workspaces(0)
```

Innerhalb einer durch ein *Workspace*-Objekt definierten Sitzung lassen sich beliebig viele Datenbanken öffnen. Jede geöffnete Datenbank wird durch ein *Database*-Objekt in der *Databases*-Auflistung repräsentiert, die über ein *Workspace*-Objekt zur Verfügung gestellt wird:

```
Dim Wo As Workspace
Dim Db As Database
Set Wo = DBEngine.Workspaces(0)
Set Db = Wo.OpenDatabase("Biblio")
?Wo.Databases.Count
1
```

DAO-Objekt	Steht für ...
Database	Eine einzelne Datenbank in einer Workspace-Sitzung.
Errors	Enthält für jeden Teilfehler des zuletzt aufgetretenen Fehlers ein eigenes *Error*-Objekt, dessen *Number*- und *Description*-Eigenschaft den Fehlertyp angibt.
Field	Ein einzelnes Feld eines Datensatzes.
Index	Steht für einen einzelnen Index.
QueryDef	Eine in der Datenbank gespeicherte Abfrage.

Tabelle C.1: Die wichtigsten DAO-Objekte in der Übersicht

Tabelle C.1: Die wichtigsten DAO-Objekte in der Übersicht (Fortsetzung)

DAO-Objekt	Steht für ...
Recordset	Eine Datensatzgruppe.
Relation	Eine 1:1- oder 1:n-Beziehung zwischen einem Primär- und einem Fremdschlüssel zweier Tabellen.
TableDef	Eine Tabellendefinition (Schema), die in der Access-Datenbank enthalten ist und den Aufbau der einzelnen Tabellen beschreibt.
Workspace	Eine »Arbeitssitzung, die u.a. den Sicherheits- und Transaktionsrahmen für einen Datenbankzugriff festlegt.
Container	Alle Objekte, die zu einer Datenbank gehören, und die von der Jet-Engine in einem sogenannten Container zusammengefaßt werden. Standardmäßig sind die Objekte *Databases*, *Tables* und *Relationships* dabei. Durch Hinzufügen neuer Objekte kann die Jet-Engine um benutzerdefinierte Objekte erweitert werden[1].
Document	Alle *Container*-Objekte eines bestimmten Typs, z.B. *Tables*. Ein einzelnes *Document*-Objekt beschreibt ein einzelnes dieser in der Datenbank gespeicherten Objekte. Alle Document-Objekte gleichen Typs werden über *Documents* zusammengefaßt.
Group	Im Zusammenhang mit der Access-Sicherheit für die Daten einer Benutzergruppe. Sicherheitsfeatures können von Visual Basic nur genutzt, nicht aber implementiert werden.
User	Im Zusammenhang mit der Access-Sicherheit für die Daten eines Benutzers. Sicherheitsfeatures können von Visual Basic nur genutzt, nicht aber implementiert werden.

C.3 Das *Database*-Objekt

Das wichtigste Objekt ist das *Database*-Objekt, denn es steht für die Datenbank, auf die der Zugriff erfolgt. Über die *OpenDatabase*-Methode geben Sie entweder den Namen einer Access-Datenbank (oder einer ISAM-Datenbank) an oder legen direkt (über den Zusatz »ODBC;«) oder indirekt (durch Angabe eines DSN-Namens anstelle eines Datenbanknamens) fest, daß der Zugriff auf eine ODBC-Datenbank und damit über die Verbindungszeichenfolge eines *Data Source Name* (DSN) erfolgt.

1 Ich habe das allerdings noch nie ausprobiert. Es hört sich jedenfalls gut an.

Die folgende Befehlsfolge öffnet die Datenbank *Fuhrpark.mdb*:

```
Dim Db As Database
Set DB = DbEngine.Opendatabase("C:\Eigene
Dateien\Fuhrpark.mdb")
```

Da kein *Workspace*-Objekt explizit angegeben wurde, arbeitet die Jet-Engine nun mit dem Default-Workspace. Wird ein *Database*-Objekt nicht mehr benötigt, sollte es über seine *Close*-Methode geschlossen werden. Ein direktes Pendant zum *Database*-Objekt gibt es bei den ADOs nicht. Anstelle einer Datenbank gibt man hier eine Verbindung an, die über ein *Connection*-Objekt zur Verfügung gestellt wird.

C.4 Das *Recordset*-Objekt

Das *Recordset*-Objekt steht, wie bei den ADOs, für keinen, einen oder mehrere Datensätze. Es wird meistens über die *OpenRecordset*-Methode des *Database*-Objekts angelegt. Als Argument kann der Methode der Name einer Tabelle, ein SQL-Kommando oder der Name eines *QueryDef*-Objekts (also einer in der Access-Datenbank gespeicherten Abfrage) übergeben werden.

Die folgende Befehlsfolge öffnet ein *Recordset*-Objekt mit dem Inhalt der Tabelle *Personenfahrzeuge*.

```
Dim Db As DAO.Database
Dim Rs As DAO.Recordset
Set DB = DbEngine.OpenDatabase _
  ("C:\Eigene Dateien\Fuhrpark.mdb")
Set Rs = Db.OpenRecordset("Personenfahrzeuge")
' Irgendwelche Befehle
Rs.Close
Db.Close
```

Wie ein *Database*-Objekt sollte auch ein *Recordset*-Objekt am Ende über seine *Close*-Methode wieder geschlossen werden.

Wird bei *OpenRecordset* keiner der Typargumente *dbOpenDynamic* (nur bei ODBCDirect), *dbOpenDynaset*, *dbOpenForwardOnly*, *dbOpenSnapshot* oder *dbOpenTable* angegeben, wird ein Recordset vom Typ *Table* angelegt.

C.5 Unterschiede zwischen Dynaset-, Snapshot- und Table-Recordsets

Das *Recordset*-Objekt gibt es bei DAO in vier »Geschmacksrichtungen«: Dynaset, Snapshot, Table und ForwardOnly. Über die Unterschiede gibt Tabelle C.2 Auskunft.

Tabelle C.2: Die verschiedenen Recordset-Typen in der Übersicht

Recordset-Typ	Besonderheiten
Dynaset	Umfaßt die Schlüssel aller Mitglieder, was bedeutet, daß von anderen Benutzern hinzugefügte oder entfernte Datensätze erst nach einem Refresh sichtbar werden. Änderungen am Inhalt einzelner Datensätze durch andere Benutzer werden jedoch sofort sichtbar.
Snapshot	Entspricht einem Dynaset, nur daß ein Snapshot nicht verändert werden kann (Nur-Lese-Modus). Außerdem wird der Inhalt eines Snapshots komplett in den Arbeitsspeicher geladen und nicht nur die Schlüssel wie bei einem Dynaset.
Table	Entspricht 1:1 dem Inhalt einer Tabelle. Auf *Table*-Objekte läßt sich die schnelle, indexbasierende *Seek*-Methode anwenden, was bei den übrigen Recordset-Typen nicht geht. Dafür können Table-Objekte nicht aus SQL-Abfragen entstehen.
ForwardOnly	In der Datensatzgruppe ist nur eine Vorwärtsbewegung möglich. Ein *MoveFirst* oder *MovePrevious* führt zu einem Laufzeitfehler. Diese Einschränkung ist jedoch erwünscht, denn dieser Recordset-Typ bietet dafür eine sehr gute Performance.

C.6 Kleine Beispiele zu den DAOs

Auch den Umgang mit den DAOs lernt man (und Frau natürlich auch) am besten an kleinen Beispielen.

C.6.1 Durchlaufen einer Datensatzgruppe

Das Durchlaufen einer Datensatzgruppe geschieht (wie bei den ADOs) über die *MoveNext*-Methode und die EOF-Eigenschaft.


```
Dim Wo As Workspace
Dim Db As Database
Dim Rs As Recordset
Set Wo = DBEngine.Workspaces(0)
Set Db = Wo.OpenDatabase("Biblio")
```

```
Set Rs = Db.OpenRecordset _
    ("Publishers", dbOpenForwardOnly)
Do While Not Rs.EOF = True
    Debug.Print Rs.Fields("Name").Value
    Rs.MoveNext
Loop
Rs.Close
Db.Close
```

C.6.2 Die »geheimnisvolle« *RecordCount*-Eigenschaft

Die *RecordCount*-Eigenschaft besitzt bei *Dynaset*-Recordsets die Eigenheit, daß sie erst dann die Anzahl der Datensätze angibt, wenn der Datensatzzeiger einmal an das Ende der Datensatzgruppe bewegt wurde.

Im folgenden wird ein Recordset geöffnet, wobei die *RecordCount*-Eigenschaft lediglich den Wert 1 zurückgibt.

```
Const DBPfad = "C:\Eigene Dateien\Biblio.mdb"
Dim Rs As Recordset
Set DB = OpenDatabase(DBPfad)
Set Rs = DB.OpenRecordset _
  ("Select * From Authors", dbOpenDynaset)
MsgBox Prompt:=Rs.RecordCount & " Datensätze"
Rs.Close
```

Diesmal wird unmittelbar nach dem Öffnen die *MoveLast*-Methode ausgeführt, so daß *RecordCount* jetzt die tatsächliche Anzahl an Datensätzen angibt.

```
Const DBPfad = "C:\Eigene Dateien\Biblio.mdb"
Dim Rs As Recordset
Set DB = OpenDatabase(DBPfad)
Set Rs = DB.OpenRecordset _
    ("Select * From Authors", dbOpenDynaset)
Rs.MoveLast
MsgBox Prompt:=Rs.RecordCount & " Datensätze"
Rs.Close
```

Ein deutlicher Nachteil dieses Verfahrens ist, daß die *MoveLast*-Methode bei großen Datensatzgruppen recht lange dauern kann.

Daß scheinbar harmlose Zusätze Nebeneffekte besitzen können, macht der SQL-Zusatz »ORDER BY« deutlich, der beim Öffnen eines Recordsets eine bestimmte Sortierreihenfolge vorgibt und dadurch den Datensatzzeiger an das Ende der Datensatzgruppe bewegt.

```
Const DBPfad = "C:\Eigene Dateien\Biblio.mdb"
Dim DB As Database
Dim Rs As Recordset
Set DB = OpenDatabase("DbPfad)
Set Rs = DB.OpenRecordset _
   ("Select * From Authors order by Author", _
   dbOpenDynaset)
MsgBox Prompt:=Rs.RecordCount & " Datensätze"
Rs.Close
```

Ein *MoveLast* ist in diesem Fall nicht erforderlich.

C.6.3 Die Suche nach einem Datensatz

Für die Suche nach einem Datensatz in einem Recordset-Objekt gibt es bei DAO gleich vier Methoden: *FindFirst*, *FindNext*, *FindPrevious* und *FindLast*. Sie unterscheiden sich lediglich in der Frage, an welcher Position die Suche beginnt.

Die folgende Befehlsfolge öffnet die Tabelle *Publishers* (als Snapshot) in der Datenbank *Biblio.mdb* und gibt die Namen jener Verleger aus, die mit einem »S« beginnen.

```
Dim Wo As Workspace
Dim Db As Database
Dim Rs As Recordset
Set Wo = DBEngine.Workspaces(0)
Set Db = Wo.OpenDatabase("Biblio")
Set Rs = Db.OpenRecordset("Publishers", dbOpenSnapshot)
Rs.FindFirst "Name Like 'S*'"
Do While Not Rs.NoMatch = True
    Debug.Print Rs.Fields("Name").Value
    Rs.FindNext "Name Like 'S*'"
Loop
Rs.Close
Db.Close
```

Das Argument, das auf *FindFirst* folgt, entspricht der Bedingung einer *WHERE*-Klausel in einer SQL-Abfrage. Anders als bei der ADO-*Find*-Methode können hier mehrere Bedingungen kombiniert werden.

C.6.4 Editieren von Datenbankfeldern

Damit die Jet-Engine den Wert eines Datenbankfeldes ändert, muß das dahinterstehende *Recordset*-Objekt mit der *Edit*-Methode zunächst in den »Überarbeitenmodus« geschaltet werden (bei ADO ist das nicht erforderlich). Anschließend erhält das *Field*-Objekt über seine *Value*-Eigenschaft einen neuen Wert. Damit der Wert in die Datenbank übernommen wird, muß zum Schluß (wie bei ADO) die *Update*-Methode des *Recordset*-Objekts ausgeführt werden.

Die folgende Befehlsfolge öffnet die Tabelle *Authors* der Datenbank *Biblio.mdb* und gibt dem Feld *Year Born* einen neuen Wert (beachten Sie, daß der Feldname diesmal nicht in eckige Klammern gesetzt werden muß).

```
Dim Db As Database
Dim Rs As Recordset
Set Db = DBEngine.Workspaces(0).OpenDatabase _
   ("C:\Eigene Dateien\Biblio.mdb")
Set Rs = Db.OpenRecordset _
   (Name:="Authors", Type:=dbOpenTable)
With Rs
    .Edit
    .Fields("Year Born").Value = 2100
    .Update
    .Close
End With
Db.Close
```

Sollte ein anderes Programm zeitgleich auf den gleichen Datensatz zugreifen, kommt es zu einem Zugriffskonflikt, der sich durch einen Laufzeitfehler bemerkbar macht. Wann dieser auftritt, hängt von der Art der Sperre ab, die über die *LockEdits*-Eigenschaft des *Recordset*-Objekts eingestellt werden kann. Zur Auswahl stehen zwei Sperrmechanismen: die teilweise Sperre (engl. *optimistic locking, LockEdits=False*) und die vollständige Sperre (engl. *pessimistic locking, LockEdits=True*). Bei der *vollständigen Sperre* wird der aktuelle Datensatz gesperrt, sobald ein Prozeß für das *Recordset*-Objekt die *Edit*-Methode ausführt. Alle übrigen Prozesse können die *Edit*-Methode für diesen Datensatz (und die benachbarten Datensätze, denn die Jet-Engine sperrt stets einen 2 Kbyte großen Bereich) nicht mehr aufrufen und erhalten bei einem solchen Versuch den Laufzeitfehler 3260. Die Ausführung der *Update*-Methode gibt den Datensatz wieder frei. Diese Sperrmethode ist zwar sehr zuverlässig, hat aber den Nachteil, daß der Datensatz relativ lange für die anderen Benutzer gesperrt ist. Macht der Benutzer vor dem Ausführen der *Update*-Methode eine Mittagspause, kann in

dieser Zeit niemand an den gesperrten Datensätzen irgendwelche Änderungen vornehmen. Die *teilweise Sperre* ist etwas großzügiger. Hier wird die Sperre erst mit dem Aufruf der *Update*-Methode eingerichtet und anschließend wieder aufgehoben. Andere Benutzer (Prozesse) können in dieser Zeit ebenfalls die *Edit*-Methode aufrufen, um den Datensatz zu bearbeiten. Nach dem Motto »Wer zuerst kommt, ...« führt derjenige Prozeß die Änderungen in der Datenbank durch, der zuerst die *Update*-Methode ausführt. Würde in dieser (allerdings sehr kurzen) Zeitspanne ein anderer Benutzer versuchen, die *Edit*-Methode auszuführen, wäre ein Laufzeitfehler 3260 die Folge. Wenn ein Benutzer bei optimistischer Sperre die *Edit*-Methode ausgeführt hat und nun die *Update*-Methode ausführen möchte und ein anderer Prozeß in der Zwischenzeit bei pessimistischer Sperre ebenfalls die *Edit*-Methode ausgeführt und dabei bereits Feldinhalte geändert hat, stellt die Jet-Engine vor der Ausführung der *Update*-Methode des Benutzers mit optimistischer Sperre fest, daß die Daten nach der Ausführung der *Edit*-Methode geändert wurden, und gibt den Laufzeitfehler 3197 (»Daten wurden verändert; Operation angehalten«) aus.

Bis zur Version 3.51 (bzw. 3.6) sperrt die Jet-Engine stets einen 2 Kbyte großen Bereich um den betroffenen Datensatz herum, was bedeutet, daß auch Nachbardatensätze gesperrt werden. Mit Microsoft Access 2000 und der Jet-Engine 4.0 wird (endlich) eine datensatzweise Sperre geboten. Leider läßt sich diese nicht über die DAOs ändern, sondern lediglich nutzen. Dafür werden die ADO Extensions (ADOX) für die Jet-Engine benötigt.

C.6.5 Öffnen Datenbankabfrage (über ein *QueryDef*-Objekt)

QueryDef-Objekte sind dazu da, die in einer Access-Datenbank enthaltenen Abfragen auszuführen oder neue Abfragen anzulegen. Um mit einem bereits angelegten *QueryDef*-Objekt arbeiten, d.h. die in ihm enthaltene Abfrage ausführen, zu können, muß das Objekt lediglich geöffnet werden:

```
Dim Q As QueryDef
Dim Db As Database
Dim Rs As Recordset
Set Db = DBEngine.Workspaces(0) _
  .OpenDatabase("C:\Eigene Dateien\Biblio.mdb")
Set Q = Db.QueryDefs("[All Titles]")
Set Rs = Q.OpenRecordset(dbOpenSnapshot)
' Irgendwelche Befehle
```

```
Rs.Close
Db.Close
```

Das Ergebnis ist eine *Recordset*-Variable *Rs*, die alle Datensätze umfaßt, die von der Abfrage zurückgegeben wurden.

C.6.6 Anlegen eines *QueryDef*-Objekts

QueryDef-Objekte lassen sich programmgesteuert anlegen, auch wenn dies mit Microsoft Access oder dem Visual-Data-Manager-Add-In von Visual Basic sehr viel komfortabler geht.

Die folgende Befehlssequenz legt in der Datenbank *Biblio.mdb* über ein *QueryDef*-Objekt eine neue Abfrage an. Sollte diese bereits existieren, ist ein Laufzeitfehler die Folge, in dessen Verlauf die Abfrage zunächst über die *Delete*-Methode der *QueryDefs*-Auflistung gelöscht und anschließend das Anlegen der Abfrage über den *Resume*-Befehl wiederholt wird.

```
Private Sub QueryTest ()
    On Error GoTo cmdQueryDefAnlegen_Error
    Dim Q As QueryDef
    Dim Db As Database
    Dim Rs As Recordset
    Dim SQLString As String
    Set Db = DBEngine.Workspaces(0).OpenDatabase _
     ("C:\Eigene Dateien\Biblio.mdb")
    SQLString = _
      "SELECT * FROM Authors WHERE Author Like 'M*'"
    Set Q = Db.CreateQueryDef("AutorenMitM", SQLString)
    Set Rs = Q.OpenRecordset(dbOpenSnapshot)
    ' Irgendwelche Befehle
    Rs.Close
    Db.Close
    Exit Sub
cmdQueryDefAnlegen_Error:
    Select Case Err.Number
    Case 3012
     ' QueryDef-Objekt existiert bereits - dann löschen
        Db.QueryDefs.Delete "AutorenMitM"
        Resume
    Case Else
        MsgBox Prompt:=Err.Description & _
         " (" & Err.Number & ")", _
         Buttons:=vbExclamation, _
```

```
            Title:="Laufzeitfehler"
        End Select
End Sub
```

QueryDef-Objekte lassen sich aus einer Datenbank auch über den Visual Data Manager entfernen, indem Sie im Datenbankfenster die Abfrage löschen.

C.6.7 Aktionsabfragen

Aktionsabfragen sind SQL-Kommandos, die Veränderungen an der Datenbank vornehmen. Bei ADO werden Aktionsabfragen über ein *Command*-Objekt durchgeführt, bei DAO über die *Execute*-Methode eines *QueryDef*-Objekts.

Die folgenden Befehlsfolgen legen eine Aktionsabfrage an, die bei ihrer Ausführung in alle Felder der Tabelle *Authors*, die den Wert *NULL* besitzen, den Wert 9999 einträgt. Bevor Sie dieses Beispiel ausführen, sollten Sie eine Kopie der Datenbank *Biblio.mdb* anlegen, da sich die Änderungen nur indirekt (durch eine weitere Aktionsabfrage, die alle 9999-Felder wieder löscht) rückgängig machen lassen.

```
Dim Db As Database
Dim Q As QueryDef
Dim sSQLKommando As String
' Datenbank öffnen
Set Db = DBEngine.OpenDatabase _
   ("C:\Eigene Dateien\Biblio.mdb")
' SQL-String zusammenbauen
sSQLKommando = _
 "UPDATE Authors SET [Year Born] = 9999 " & _
 "WHERE [Year Born] = Null"
' QueryDef-Objekt anlegen
Set Q = Db.CreateQueryDef("NullFelderWeg")
With Q
 .SQL = sSQLKommando
' Aktionsabfrage ausführen
 .Execute
' Recordset-Objekt des Datensteuerelements neu aufbauen
 .Close
End With
```

C.6.8 Abfragen mit Parametern

Bei einer Abfrage mit Parametern werden ein oder mehrere Teile der (Aktions-)Abfrage durch Variablen ersetzt, denen vor der Ausführung der Abfrage Werte zugewiesen werden.

Die folgende Befehlsfolge führt ein weiteres Mal die beliebte Autorenabfrage durch, nur daß diesmal der erste Buchstabe des Autorennamens der Abfrage als Parameter übergeben wird.

```
Private Sub AutorenMitParameter ()
    On Error GoTo QueryDefParaErr
    Dim Q As QueryDef, Db As Database, Rs As Recordset
    Dim sSQLKommando As String, sSQLParameter As String
    sSQLParameter = "M*"
    Set Db = DBEngine.Workspaces(0).OpenDatabase _
       ("C:\Eigene Dateien\Biblio.mdb")
    sSQLKommando = _
    "PARAMETERS Param1 Text; SELECT * " & _
    "FROM Authors WHERE Author Like Param1"
    Set Q = Db.CreateQueryDef _
      ("AutorenMitParameter", sSQLKommando)
    With Q
     .Parameters("Param1") = sSQLParameter
     Set Rs = .OpenRecordset(dbOpenSnapshot)
    ' Irgendwelche Befehle
    Close
    End With
    Exit Sub
QueryDefParaErr:
    Select Case Err.Number
     Case 3012
     ' QueryDef-Objekt existiert bereits - dann löschen
        Db.QueryDefs.Delete "AutorenMitParamter"
        Resume
     Case Else
        MsgBox Prompt:=Err.Description _
         & " (" & Err.Number & ")", _
         Buttons:=vbExclamation, _
         Title:="Laufzeitfehler"
    End Select
End Sub
```

C.7 Eine Gegenüberstellung DAO zu ADO

In diesem Buch darf natürlich eine kurze Gegenüberstellung der ADOs mit den DAOs nicht fehlen. Anstelle wortreicher Formulierungen muß es aus Platzgründen aber bei einer kleinen Tabelle bleiben (ausführlichere Artikel finden Sie u.a. in der MSDN-Hilfe).

Tabelle C.3: Kurze Gegenüberstellung DAO- und ADO-Objekte

DAO-Objekt	ADO-Pendant	Unterschiede
Database	*Connection*	Das *Database*-Objekt ist kein echtes Pendant, sondern vielmehr jenes Objekt, das dem *Connection*-Objekt, das die Datenquelle auswählt, am nächsten kommt.
Workspace	–	Zum *Workspace* Objekt gibt es kein direktes Pendant.
TableDef	–	Zum *TableDef* Objekt gibt es kein direktes Pendant. Schemainformationen werden bei ADO entweder über die Methode *OpenSchema* oder über die *ADO Extensions* (ADOX) für die Jet Engine zur Verfügung gestellt, die es aber erst ab Microsoft Access 2000 gibt.
QueryDef	– (*Command*)	Zum Objekt *QueryDef* gibt es kein direktes Pendant. Abfragen und der Aufruf von Stored Procedures (etwa beim Microsoft SQL-Server) erfolgen über das *Command*-Objekt.

C.8 Versionsnummern bei DAOs

Die DAOs existieren in verschiedenen Versionen, die sich durch einzelne Leistungsmerkmale unterscheiden können. Die Versionsnummer der DAOs muß allerdings nicht mit der Versionsnummer der Jet-Engine übereinstimmen. So erfolgt der Zugriff auf die Jet-Engine 4.0 in Microsoft Access 2000 über DAO 3.51 bzw. 3.6. Denken Sie daran, daß es sich bei der Jet-Engine um ein DBMS, bei den DAOs nur um eine Objektschnittstelle handelt.

DAO-Version	Besonderheit
1.1	Wurde mit Visual Basic 3.0 eingeführt.
2.0	Wurde mit Microsoft Access 2.0 eingeführt
2.5	32-/16-Bit-Kompatibilitätsversion für den Zugriff auf 16-Bit-Jet-Engine von 32-Bit-Anwendungen.
3.0	32-Bit-Version. Wurde mit Microsoft Access 95 und Visual Basic 4.0 eingeführt.
3.5	Wurde mit Visual Basic 5.0 für den Zugriff auf die Jet-Engine 3.5 eingeführt, deren wichtigstes Merkmal der ODBCDirect-Modus ist.
3.51	Wurde mit Microsoft Access 97 eingeführt.
3.6	Wird mit Microsoft Access 2000 eingeführt und zeichnet sich u.a. durch eine Unicode-Unterstützung aus. Vermutlich ist dies die letzte DAO-Version. Die Jet-Engine, die bei Office 2000 in der Version 4.0 vorliegt, wird dagegen aller Voraussicht nach weiterentwickelt.

Tabelle C.4: Die verschiedenen DAO-Versionen in der Übersicht

C.9 Zusammenfassung

Auch wenn die ADOs für Visual Basic (ab Version 6.0) die wichtigste Datenbankschnittstelle darstellen und ihnen (bei Microsoft) die Zukunft gehört, stehen die mit Microsoft Access und Visual Basic 3.0 eingeführten *Data Access Objects* (DAOs) nach wie vor zur Verfügung. Wer sich für die Jet-Engine als DBMS entschieden hat, steht vor der nicht ganz einfachen Entscheidung: DAO oder ADO? Die DAOs sind veraltet, dafür aber 1:1 auf die Jet-Engine abgestimmt und für einen angehenden Datenbankprogrammierer, der bereits Microsoft Access kennt, ein wenig leichter verständlich[1]. Dafür werden die DAOs nicht von den modernen Steuerelementen und den mit Visual Basic 6.0 eingeführten Datenwerkzeugen, wie dem Datenumgebungs-Designer oder dem Reportgenerator, unterstützt. Weitere Argumente pro und contra ADO/DAO finden Sie in der Einleitung des Buches.

1 Ich habe beim Ausprobieren der Beispielprogramme für diesen Anhang einmal mehr festgestellt, daß die DAOs durchaus ihre Vorzüge haben. Doch eine Entscheidung gegen die ADOs läßt sich damit alleine sicher nicht begründen.

Anhang D

Antworten zu den Fragen

In diesem Anhang erhalten Sie Antworten auf Ihre Fragen. Nicht auf die des Lebens (dafür reicht der Platz leider nicht[1]), sondern auf jene Fragen, die Sie am Ende von Kapitel 1 bis 10 vorfinden (Kapitel 11 hat keine Fragen an Sie).

Die Antworten zu Kapitel 1

Antwort 1:

Eine Datenbank ist ein Ort, an dem Daten gespeichert werden, wobei die Daten üblicherweise in Tabellen, Datensätzen und Feldern organisiert sind.

Antwort 2:

DAO (Data Access Objects), RDO (Remote Data Objects) und ADO (Active Data Objects).

Diese Frage soll natürlich nicht suggerieren, es käme bei der Datenbankprogrammierung auf das Auswendiglernen von Abkürzungen an (die sich zudem eine einzige Firma ausgedacht hat), doch ein paar Kürzel muß man einfach kennen.

1 Abgesehen davon, daß sich der Autor ein wenig überfordert fühlen würde.

Antwort 3:

DAO basiert auf der Jet-Engine und damit auf einem bestimmten Datenbank-Management-System (DBMS). ADO basiert auf OLE DB, einer Programmierschnittstelle, und ist damit unabhängig von einem Datenbank-Management-System.

Antwort 4:

Auf welche Weise der physikalische Aufenthaltsort der Datenbank festgelegt wird, hängt vom gewählten OLE DB-Provider ab (es ist ein wenig komplizierter). Wurde der OLE DB-Provider für die Jet-Engine gewählt, legt die *ConnectionString*-Eigenschaft u.a. den Pfad der MDB-Datei fest. Wurde dagegen der OLE DB-Provider für ODBC gewählt, legt die *ConnectionString*-Eigenschaft den Namen eines Data Source Name (DSN) oder direkt die Verbindungsdaten (Name des Treibers, Name des Servers, Name der Datenbank usw.) fest.

Das ist das allgemeine Zugriffsprinzip für Datenbanken. Daß der Pfad einer Datenbank (-datei) direkt angegeben wird (wie bei Microsoft Access), ist eine Ausnahme.

Antwort 5:

Durch die Trennung von Datenbank und des Programms, das auf die Datenbank zugreift, ergibt sich eine vernünftige Arbeitsteilung. Die Datenbank befindet sich auf einem sehr leistungsfähigen PC (oder allgemein Computer) und wird von einer sehr leistungsfähigen Software (dem DBMS) verwaltet. Das Programm, das auf die Datenbank zugreift, muß dagegen wenig können, kann sich physikalisch »irgendwo« befinden und benötigt relativ wenig Ressourcen (geht es nur um das Anzeigen von Datensätzen, reicht sogar ein Uralt-PC aus (z.B. ein 386er mit 4 Mbyte RAM), auf dem noch nicht einmal Windows laufen muß. Wie es dagegen bei einer Access-Datenbank aussieht, die nicht in die Kategorie Client/Server-Datenbank fällt (zumindest bis zur Version 97), wissen Sie vermutlich bereits. Die neueste Hardware wird auch für jenes Programm benötigt, das »nur« die Daten anzeigen soll.

Antwort 6:

Da gibt es natürlich viele richtige Antworten: der Microsoft SQL Server, der Oracle SQL Server, Interbase von Inprise (früher Borland), SQL Anywhere von Sybase, Informix von Informix oder DB2 von IBM. Die Liste ließe sich noch fortsetzen, denn Client/Server-Datenbank-Management-Systeme sind in der Industrie sehr verbreitet. Allerdings kommen diese Systeme in der Regel nicht für den Privatanwender in Frage.

Die Antworten zu Kapitel 2

Antwort 1:

Eine relationale Datenbank ist eine Datenbank, bei der die Daten in (normalisierten) Tabellen enthalten sind, zwischen denen Beziehungen existieren (aber nicht müssen).

Antwort 2:

Die Aufteilung der Daten auf Tabellen ist die Voraussetzung dafür, daß sich die Regeln relationaler Datenbanken anwenden lassen, die wiederum die Datenintegrität sicherstellen und Datenbankabfragen per SQL ermöglichen. Ein unumstößliches Gesetz stellt dies allerdings nicht dar. Im Gegenteil, bei komplexen Datenbeständen (etwa dem Profil eines Bausparers mit allen seinen charakteristischen Daten) erweist sich das relationale Datenmodell als nicht optimal. Hier werden zunehmend objektorientierte Datenbanken (wie z.B. Poet) eingesetzt, bei denen sich die Beziehungen zwischen Objekten und vor allem die mit den Objekten verknüpften Daten sehr viel besser speichern lassen (hier ist die »zwangsweise«) Aufteilung in Tabellen eher hinderlich).

Antwort 3:

Leider keine. Wer eine Datenbank mit einem Software-Werkzeug und nicht mit Papier und Bleistift entwerfen möchte, muß auf eines der angebotenen Produkte, wie das nicht ganz preiswerte Erwin von LogicWorks zurückgreifen. Diese Werkzeuge, deren Handhabung auch erst einmal gelernt sein will, lohnen sich in der Regel erst, wenn es um den Entwurf größerer Datenbanken geht, die sich im täglichen Härtetest bewähren müssen. Für kleine Datenbanken oder bei Datenbanken für den Heim- oder Bürogebrauch kommen sie meistens nicht in Frage.

Antwort 4:

Der Schwachpunkt in der Tabelle ist hoffentlich leicht zu erkennen, denn das Feld *Autoren* ist nicht atomar. Die Tabelle befindet sich daher nicht in der 1. Normalform. Auch wenn es nicht »verboten« ist, mehrere Autorennamen in einem Feld zusammenzufassen, besitzt dies den Nachteil, daß sich die Tabelle nicht (oder nur relativ aufwendig) nach den einzelnen Autoren durchsuchen oder sortieren läßt. Um die Tabelle in die 1. Normalform zu bringen, müssen entweder für die Co-Autoren zusätzliche Felder eingeführt oder die Autoren in eine separate Tabelle ausgelagert werden. In diesem Fall muß die ursprüngliche Tabelle aber mit einem zusätzlichen Schlüsselfeld, etwa *BuchNr*, versehen werden.

Antwort 5:

Eine zu strenge Normalisierung, d.h. eine Aufteilung in zu kleine Tabellen, kann zu Performance-Nachteilen (je mehr Tabellen im Spiel sind, desto länger dauert eine Abfrage) und u.U. auch zu Informationsverlusten führen.

Antwort 6:

Die Lösung kennen Sie bereits. Sie führen zusätzliche Tabellen ein und stellen zwischen den Tabellen Beziehungen her. So wäre es sinnvoll, eine Tabelle mit dem Namen *Besuche* einzuführen, in der nur die Besuche der einzelnen Patienten vermerkt sind:

- ✘ BesuchNr
- ✘ PatientenNr
- ✘ Abrechnung
- ✘ Diagnose

Über den Fremdschlüssel *PatientenNr* verweist die Tabelle auf die Tabelle *Patienten*, wo *PatientenNr* als Primärschlüssel vorhanden ist. Die Tabelle *Patienten* besitzt nun den folgenden Aufbau:

- ✘ PatientenNr
- ✘ Name
- ✘ Adresse
- ✘ Telefon

Sie sehen, daß Informationen über eventuelle Besuche oder Diagnosen (also alle mehrfach vorkommenden Einträge) nicht mehr in der Tabelle enthalten sind.

Lösungen zu Kapitel 3

Antwort 1:

- ✘ Die Frage bezieht sich selbstverständlich auf »richtige« Datenbanken, denn sonst wäre auch Notepad (siehe Kapitel 1) eine korrekte Antwort: Microsoft Access, Visual Basic und der Visual Data Manager (letzterer wird in der Regel als Visual-Basic-Add-In aufgerufen, so daß die Trennlinie ein wenig unscharf ist). Visual Basic ist auch ohne den Visual Data Manager in der Lage, Datenbanken zu erstellen, allerdings mit weitaus weniger Komfort, da alle Anweisungen im Programmcode erfolgen müssen).

Antwort 2:

✘ Zu den eindeutigen Stärken gehört die extrem einfache Bedienung. Zu den eindeutigen Schwächen natürlich das Fehlen wichtiger Leistungsmerkmale (etwa das Herstellen von Relationen, das damit indirekt zusammenhängende Anzeigen von Beziehungsdiagrammen, der fehlende Komfort bei Datenbankabfragen usw.) Eine Stärke liegt allerdings im Verborgenen: Der Visual Data Manager liegt auch im Quellcode vor, so daß man die Chance erhält, praktisch alles über die Datenbankprogrammierung mit Visual Basic und der Jet-Engine (allerdings noch mit den alten DAO-Objekten) zu lernen. Bei älteren Versionen gab es sogar die Möglichkeit, sich Performance-Vergleiche bei verschiedenen Datenzugriffsmethoden anzeigen zu lassen.

Antwort 3:

✘ Ja, auch das geht (Sie können z.B. eine Microsoft SQL-Server-Datenbank anlegen), wenngleich es aufgrund des fehlenden Komforts nur eine theoretische Option ist.

Antwort 4:

✘ Leider nur begrenzt. Um etwa die Eigenschaften eines Feldes zu ändern, muß das Feld erst gelöscht und dann erneut eingefügt werden. Microsoft Access ist in dieser Beziehung sehr viel flexibler.

Antwort 5:

✘ Dazu muß in der Tabellenstruktur einer Tabelle ein Index hinzugefügt und die Option *Primary* angeklickt werden.

Antwort 6:

✘ Nein, noch nicht, denn dadurch wird in der Access-Datenbank noch keine Beziehung definiert. Das geht nur innerhalb von Microsoft Access, indem Sie im Menü EXTRAS den Befehl BEZIEHUNGEN ausführen, die Tabellen in das Beziehungsfenster einfügen und die Felder der beiden Tabellen verbinden, indem Sie das Feld mit dem Fremdschlüssel mit der Maus auf das Feld, das den Primärschlüssel darstellt, der anderen Tabelle ziehen.

Lösungen zu Kapitel 4

Antwort 1:

Die Eigenschaften *ConnectionString* und *RecordSource*.

Antwort 2:

Erst, nachdem die Bindung an ein Steuerelement erfolgt ist, steht die *DataSource*-Eigenschaft eines *Recordset*-Objekts für das Recordset, über das die Bindung hergestellt wird, zur Verfügung. Ihr muß daher über den *Set*-Befehl ein Wert zugewiesen werden.

Antwort 3:

Die *DataField*-Eigenschaft und indirekt auch die *DataSource*-Eigenschaft.

Antwort 4:

Es legt fest, in welchem Format ein aus einem Datenbankfeld eingelesener Wert in dem Steuerelement dargestellt wird. Hinter der *DataFormat*-Eigenschaft steht ein *DataFormat*-Objekt.

Antwort 5:

Das ist kein Problem, denn die Datensatzgruppe, die von der SQL-Abfrage zurückgegeben wurde, steht über die *Recordset*-Eigenschaft zur Verfügung.

Antwort 6:

Nicht alle Steuerelemente sind für OLE DB ausgelegt. Sie erkennen sie daran, daß keine *DataMember*-Eigenschaft zur Verfügung steht. Außerdem gibt es beim Versuch, eine Bindung mit dem ADO-Datensteuerelement durchzuführen, eine Fehlermeldung. Ein Beispiel ist das OLE-Steuerelement[1].

Lösungen zu Kapitel 5

Antwort 1:

Die ADO stellen die Brücke dar, über die ein Visual-Basic-Programm den Zugriff auf ein DBMS und damit auf die Datenbestände und Dienste durchführen kann, die von dem DBMS verwaltet bzw. angeboten werden. Bei ADO handelt es sich um eine Objektschnittstelle, die als Verweis in ein Projekt eingebunden werden muß.

Antwort 2:

Die richtige Antwort lautet: b.

---

1 Auch wenn die Namensähnlichkeit das Gegenteil suggeriert.

Antwort 3:

Weil das *Connection*-Objekt alle erforderlichen Verbindungsdaten (Name des OLE DB-Providers, Verbindungszeichenfolge usw.) enthält.

Antwort 4:

Den Namen der Datenquelle und den Namen eines OLE DB-Providers, wobei wenn letztere Angabe entfällt, automatisch der OLE DB-Provider für ODBC verwendet wird, mit dem sich u.a. auch Zugriffe auf Access-Datenbanken durchführen lassen.

Antwort 5:

Sowohl als auch. Sie kann sowohl der *ConnectionString*-Eigenschaft zugewiesen als auch beim Aufruf der *Open*-Methode des *Connection*- und des *Recordset*-Objekts übergeben werden. ADO ist in dieser Beziehung sehr flexibel.

Antwort 6:

Beides ist möglich. Das *Command*-Objekt bietet den Vorteil, daß sich in der Datenbank gespeicherte Abfragen ausführen und über die *Prepared*-Eigenschaft (sofern es der OLE DB-Provider unterstützt) Abfragen auch »vorkompilieren« lassen.

Lösungen zu Kapitel 6

Antwort 1:

Ein Designer ist dazu da, ein Modul eines Visual-Basic-Projekts, das auf einer Klasse basiert, zur Entwurfszeit zu bearbeiten.

Antwort 2:

Designer sind Teil des Projekts und stärker in die Entwicklungsumgebung integriert. Add-Ins sind ActiveX-DLLs, die lediglich in der Entwicklungsumgebung gestartet werden und auf deren Objekte zugreifen können.

Antwort 3:

Ein Objekt, das eines oder mehrere *Connection*-Objekte umfaßt, die jedes für sich eine beliebige Anzahl an *Command*-Objekten besitzen können, wobei für jedes *Command*-Objekt automatisch ein *Recordset*-Objekt angelegt wird. Datenumgebungen können über die Eigenschaften *DataSource* und *DataMember* mit Steuerelementen verbunden werden. Wird eine Datenumgebung über Steuerelemente gebunden, werden die betroffenen *Recordset*-Objekte nach Programmstart automatisch geöffnet.

Antwort 4:

Eine Datenumgebung ist eine Ergänzung, durch die man bestimmte Einstellungen nur einmal treffen muß und anschließend beliebig oft wiederverwenden kann. SQL-Profis können SQL-Kommandos erstellen und diese in Form einer Datenumgebung anderen Programmierern zur Verfügung stellen, die lediglich die *Command*-Objekte aufrufen.

Antwort 5:

In jedem Fall, denn auch der Zugriff auf Access-Datenbanken wird vereinfacht, indem sich z.B. Abfragen über *Command*-Objekte ausführen lassen.

Antwort 6:

Ist die Designerdatei weg, ist auch die Datenumgebung weg. In diesem Fall muß die Datenumgebung entweder neu aufgebaut oder die *Connection*-, *Command*- und *Recordset*-Objekte müssen im Programmcode angelegt werden. Verlieren Sie die Datenumgebung, verlieren Sie lediglich einen »Schlüssel« zur Datenbank, nicht aber die eigentlichen Daten.

Lösungen zu Kapitel 7

Antwort 1:

Die wichtigste Stärke von SQL ist die Standardisierung (wenngleich es natürlich kleinere Abweichungen geben kann, etwa bei AccessSQL), die Flexibilität und die dafür leichte Handhabbarkeit.

Antwort 2:

VBA weiß nichts von SQL (es gibt kein »Embedded SQL«), sondern kann ein SQL-Kommando nur als gewöhnlichen Textstring an eine externe Komponente, wie z.B. ein *Recordset*- oder *Command*-Objekt der ADODB-Bibliothek, weiterreichen.

Antwort 3:

```
SELECT * FROM Fahrzeugdaten WHERE AngeschafftAm > #" &
Format(DateAdd("yyyy", -1, Now), "dd-mm-yyyy") & "#"
```

Die Datumsangabe, die über die *DateAdd*-Funktion berechnet wird, muß bei der Jet-Engine in »#«-Zeichen eingerahmt werden (bei SQL-Server in einfache Hochkommata).

Antwort 4:

```
SELECT AVG(" & Year(Now) & " - Baujahr) FROM Fahrzeugdaten
```

Antwort 5:

```
UPDATE Fahrzeugdaten SET Preis = Preis * 0.9
```

Antwort 6:

Der Vergleich ergibt (bei 10000 Datensätzen) 14,2 Sekunden für die Schleifen-Variante und 1,6 Sekunden für die *UPDATE*-Variante und spricht damit eine deutliche Sprache (SQL-Kommandos sind meistens schneller als vergleichbare ADO-Konstruktionen). Als Benchmark wurde das folgende kleine Programm verwendet, das von einer bereits geöffneten Verbindung ausgeht und anstelle der Tabelle *Fahrzeugdaten* die (für diese Übung zuvor neu angelegte) Tabelle *DummyTabelle* verwendet, die lediglich aus zwei Feldern besteht:

```
Private Sub cmdBenchmark_Click()
    Dim Zeit As Single
    Dim n As Long
' Erst einmal 10000 Datensätze anlegen
    For n = 1 To 10000
        With Rs
         .AddNew
         .Fields("Feld1").Value = CStr(Int(Rnd * 100))
         .Fields("Feld2").Value = CStr(Int(Rnd * 100))
         .Update
        End With
    Next n
    Zeit = Timer
    Rs.MoveFirst
    Do While Rs.EOF = False
       Rs.Fields("Feld1").Value = _
          Rs.Fields("Feld1").Value * 2
       Rs.Update
       Rs.MoveNext
    Loop
    MsgBox "Zeit: " & Format(Timer - Zeit, "0.00s")
    Zeit = Timer
    Set Cmd = New ADODB.Command
    With Cmd
     .CommandType = adCmdText
     .CommandText = "UPDATE DummyTabelle SET " & _
        "Feld1 = Feld1 * 2"
     .ActiveConnection = Cn
     .Execute
```

```
    End With
    MsgBox "Zeit: " & Format(Timer - Zeit, "0.00s")
End Sub
```

Lösungen zu Kapitel 8

Antwort 1:

Über die *Source*-Eigenschaft des *Recordset*-Objekts.

Antwort 2:

Es fehlt ein Bezug auf das *Connection*-Objekt, das die Verbindung zur Datenquelle herstellt.

Antwort 3:

Damit die *Sort*-Eigenschaft beim Zugriff auf die Jet-Engine funktioniert, muß über die Eigenschaft *CursorLocation* ein clientseitiger Cursor (*adUseClient=2*) ausgewählt werden. Die Voreinstellung ist allerdings serverseitig (*adUseServer=3*), daher ist ein Laufzeitfehler 3251 die Folge.

Antwort 4:

Eine Möglichkeit ist es, daß sich der Datensatzzeiger bereits hinter den Datensätzen befindet, die das Kriterium erfüllen, so daß kein übereinstimmender Datensatz mehr gefunden werden kann.

Antwort 5:

Die *Find*-Methode lokalisiert einen Datensatz in einer beliebigen Datensatzgruppe. Die *Seek*-Methode basiert ausschließlich auf Indizes und funktioniert nur, wenn das zugrundeliegende *Command*-Objekt als »TableDirect« (*adcmdTableDirect=512*) geöffnet wurde.

Antwort 6:

Eine Möglichkeit ist es, die Tabelle als Recordset zu öffnen, dieses mit der *GetString*-Methode in eine Zeichenkette mit Trennzeichen umzuwandeln:

```
sTest = Rs.GetString(ColumnDelimeter:=":")
```

und die Zeichenkette entweder in einer Textdatei zu speichern, die später von Word geladen wird, oder per Automation an ein Word-Dokument zu übertragen.

Lösungen zu Kapitel 9

Antwort 1:

Eine wichtige Rolle, denn wann immer ein Steuerelement im Begriff ist, den Fokus zu verlieren, wird ein *Validate*-Ereignis ausgelöst. Damit lassen sich Eingabevalidierungen sehr gut durchführen.

Antwort 2:

Am einfachsten über die *DataFormat*-Eigenschaft. Die Systemsteuerung bestimmt dabei das angezeigte Währungsformat.

Antwort 3:

Das *DataCombo*-Steuerelement ist in der Lage, zwei Datensatzgruppen, die entsprechend über zwei ADO-Datensteuerelemente oder Datenumgebungen zur Verfügung gestellt werden müssen, zu verbinden, so daß über die Auswahl eines Listeneintrags ein Sekundärschlüssel ausgewählt wird, der ein »LookUp« in der zweiten Datensatzgruppe durchführt und somit einen Feldwert als Primärschlüssel holt, der anstelle des ursprünglichen Wertes angezeigt wird. Enthalten lediglich die Eigenschaften *RowSource* und *ListField* einen Wert, wird die Liste mit den über *ListField* festgelegten Feldinhalten gefüllt. Für diesen Fall wird keine zweite Datensatzgruppe benötigt. Beim einfachen Listenfeld gibt es diesen Komfort nicht. Hier muß die Liste durch das Programm gefüllt werden. Lediglich der oberste Wert (die *Text*-Eigenschaft) ist mit einem Datenbankfeld verbunden.

Antwort 4:

Das *HFlexGrid*-Steuerelement ist in der Lage, hierarchische Datensatzgruppen darzustellen, wobei sich die Hierarchie auch über mehrere Ebenen erstrecken kann.

Antwort 5:

Die *Find*-Methode kann nicht funktionieren, da sie nur ein Suchkriterium zuläßt und die restlichen ignoriert. Die *Filter*-Methode ist hier besser geeignet, wenn sie auch eine etwas andere Wirkung hat. Außerdem ist es problematisch bei Textvergleichen auf Übereinstimmung zu prüfen, da diese voraussetzt, daß der Suchbegriff exakt dem Feldinhalt entspricht.

Antwort 6:

Auch wenn Microsoft Visual Basic 6.0 als Jahr-2000-fähig mit »kleinen Unstimmigkeiten« (im Original »compliant with minor issues«) einstuft, hat jede Anwendung ein potentielles Jahr-2000-Problem, in der Jahreszahlen nur

zweistellig eingegeben werden, so daß das Programm das fehlende Jahrhundert nach eigenem Gutdünken ergänzen muß.

Grundsätzlich sollten Sie sich vor Beginn eines Projekts über den aktuellen Stand bei Visual Basic 6.0 im Internet unter der Adresse:

www.microsoft.com/technet/year2k/product/user_view22949EN.htm

informieren.

Lösungen zu Kapitel 10

Antwort 1:

Wenn ein Feld den Datentyp *Binary* (bzw. OLE-Objekt) besitzt, kann es beliebige Daten aufnehmen, die über die Methoden *AppendChunk* und *GetChunk* geschrieben bzw. gelesen werden.

Antwort 2:

Bei MP3-Dateien handelt es sich zwar um begehrte Sounddateien, doch für die Access-Datenbank sind es beliebige binäre Daten. Sie können daher mit dem *Open*-Befehl von VBA in ein *Byte*-Feld eingelesen und mit *AppendChunk* in ein Datenbankfeld eingetragen werden.

Antwort 3:

Microsoft Access fügt Bilder als OLE-Objekte ein, die aber nicht an die *Picture*-Eigenschaft eines Bildfeldes oder einer Anzeige gebunden werden können. Man muß daher über *GetChunk* das komplette Feld lesen und die Bitmap vom »OLE-Kopf« trennen und der *Picture*-Eigenschaft zuweisen.

Antwort 4:

An der fehlenden Installation von DCOM (Distributed COM), die vor der Installation der Anwendung bzw. des Microsoft Data Access Komponent-Kits (MDAC 2.1) installiert werden muß.

Antwort 5:

Am einfachsten geht dies über die *Save*-Methode des *Recordset*-Objekts. Eine andere Möglichkeit ist es, die Datensatzgruppe mit der *GetString*-Methode in eine Zeichenkette umzuwandeln, in einer Textdatei zu speichern und später mit der *Open*-Methode wieder einzulesen.

Antwort 6:

Am einfachsten geht dies natürlich über ein SQL-Kommando. Damit die Excel-Tabelle als »Datenbank« angesprochen werden kann, muß über ODBC (32bit) in der Systemsteuerung ein DSN angelegt werden, der z.B. *Torschützen* heißen kann. Das Visual-Basic-Programm kann nun wie folgt aussehen:

```
Set Cn = New ADODB.Connection
With Cn
    .Provider = "MSDASQL"
    .ConnectionString = "Data Source=Torschützen"
    .Open
End With
Set Rs = New ADODB.Recordset
    With Rs
     .ActiveConnection = Cn
     .CursorType = adOpenStatic
     .Source = _
       "SELECT Name FROM [Spieler$] WHERE Tore = " & _
       "(SELECT MAX(Tore) As MaxTore From [Spieler$] )"
     .Open
    End With
    MsgBox Prompt:="Der Spieler heißt: " & _
       Rs.Fields("Name").Value
    Cn.Close
End Sub
```

Die Angabe eines Providers ist nicht erforderlich, da ADO beim Weglassen von dem ODBC-Provider (*MSDASQL*) ausgeht.

Anhang E

Ressourcen für angehende Datenbankprogrammierer

Wer Spaß an der Datenbankprogrammierung mit Visual Basic und den ADO-Objekten gefunden hat oder sich in dieses interessante und vor allem auch sehr wichtige Thema aus anderen Gründen tiefer einarbeiten möchte (bzw. muß), benötigt zusätzliche Informationen. Damit sieht es, wie könnte es anders sein, sehr gut aus. Folgende Quellen kommen dabei in Frage:

- ✘ Weitere Bücher (etwa Einführungsliteratur zu SQL oder zur Programmierung des Microsoft SQL-Servers) und Zeitschriftenartikel (etwa die technisch sehr hochwertigen Artikel in der BasicPro)

- ✘ Die MSDN-Hilfe, die z.B. die ADO-Objekte (relativ) ausführlich beschreibt

- ✘ Verschiedene Webseiten im Internet (etwa die Microsoft-Webseite zu ADO)

- ✘ Verschiedene Newsgroups im Internet (etwa die ADO-Newsgroup zu Visual Basic und die Datenbank-Newsgroup der Zeitschrift BasicPro *www.basicworld.com*)

E.1 Bücher und Zeitschriften

Bedingt durch den Umstand, daß das Thema Datenbanken nicht nur in der Software-Entwicklung, sondern auch in der Ausbildung seit mehr als 30 Jahren zu den Standardthemen gehört, existieren zu diesem Thema eine kaum überschaubare Vielfalt an Büchern und eine Reihe von Fachzeitschrif-

ten, die sich ausschließlich der Anwendung und Programmierung von Datenbanksoftware widmen. Wer bereits einmal am Samstagvormittag in einer der großen Fachbuchhandlungen vor dem Regal mit den Datenbankbüchern gestanden hat, weiß vermutlich, daß es »das« Buch zum Thema Datenbanken nicht gibt. Die Angebotspalette läßt sich allerdings recht gut in verschiedene Kategorien einteilen:

- ✘ Lehrbücher zum Thema Datenbanken und Datenbankdesign.
- ✘ Lehrbücher und Nachschlagewerke zum Thema SQL.
- ✘ Anwendungsbücher für verschiedene Datenbanken (etwa Microsoft Access, Microsoft SQL Server oder Oracle SQL Server).
- ✘ Programmierbücher für eine bestimmte Datenbank (etwa Microsoft-Access-Programmierung).
- ✘ Programmierbücher für eine bestimmte Programmiersprache, in der es entweder teilweise oder ausschließlich um die Datenbankprogrammierung geht).

Wer richtig in die Datenbankprogrammierung einsteigen will, kommt leider mit einem Buch nicht aus, sondern benötigt im Prinzip mindestens ein Buch aus jeder Kategorie (die letzte Kategorie vielleicht ausgenommen, denn das Buch lesen Sie gerade). Bei den allgemeinen Datenbanklehrbüchern sind oft die klassischen Lehrbücher für Universitäten und Fachhochschulen empfehlenswert. Sie beziehen sich zwar nicht unbedingt auf eine bestimmte (oder aktuelle) Datenbank, stellen aber das notwendige Grundlagenwissen didaktisch sehr schön dar und sind in der Regel relativ preiswert. Hier zwei Tips von vielen Alternativen:

- ✘ Datenbanken und SQL, Edwin Schicker, 332 Seiten, Teubner Verlag (ISBN 3-519-02991-X)
- ✘ Access Databases Design&Programming, Steven Roman, 251 Seiten, O'Reilly (ISBN 1-56592-297-2)

Zum Thema SQL gibt es ebenfalls eine sehr große Auswahl an Literatur. Von der kleinen Referenz bis zur Enzyklopädie. Das Quasi-Standardwerk ist:

- ✘ A Guide To The SQL Standard, C.J.Date, 522 Seiten, Addison Wesley (ISBN 0-201-96426-0)

Auch wenn es nicht gerade durch leichte Lesbarkeit überzeugt, ist es eines der kompetentesten Bücher (Co-Autor Chris Date gehört, neben E. F. Codd, zu den »Vätern« des relationalen Datenbankmodells) zu diesem Thema[1].

Bei den Büchern zu den übrigen Kategorien hängt eine Auswahl in erster Linie von der Datenbank ab, die es zu programmieren gilt. Wer sich für Microsoft Access oder Oracle SQL entschieden hat, erhält bereits umfangreiche Dokumentation (bei Microsoft Access 97 die VBA-Hilfe und das Office 97-Programmierhandbuch, bei Microsoft Access 2000 die MSDN-Hilfe). Hier noch ein Tip für alle, die mit Visual Basic auf eine Oracle-Datenbank zugreifen möchten:

- Oracle Programming with Visual Basic, Nick Snowdown,, ca. 800 Seiten, Sybex Inc (ISBN 0-7821-2322-8)

Dies ist ein sehr wertvolles Buch und eines der wenigen, die sich speziell mit dem Thema Oracle-Datenbanken beschäftigen.

E.2 Microsoft-Webseiten

Microsoft unterhält im Internet eine der umfangreichsten Websites mit mehreren Hunderttausend Dokumenten, die rund um die Uhr abgerufen werden können. Eine spezielle Webseite für Datenbankprogrammierung mit Visual Basic gibt es zwar nicht, dafür aber zwei umfangreiche Seiten zu Visual Basic allgemein und zur Datenbankprogrammierung mit ADO:

- *msdn.microsoft.com/vbasic*
- *www.microsoft.com/data/ado*

Nicht vergessen werden soll natürlich die deutschsprachige MSDN-Seite, die stetig ausgebaut und inhaltlich wie optisch immer besser wird:

- *www.microsoft.com/germany/msdn*

1 Der »Kernighan&Ritchie« für SQL sozusagen.

E.3 Weitere Informationen im Internet

Wenn es das Internet nicht gäbe, dann hätten wir vermutlich alle mehr Zeit für andere Dinge. Wer einmal über eine der populären Suchmaschinen nach Stichworten wie Datenbanken, Database Design oder SQL forscht, stößt auf Tausende von Links, die alle mehr oder weniger interessante Informationen enthalten. Manchmal werden sogar komplette Lehrbücher veröffentlicht, wie es bei vorlesungsbegleitenden Materialien an Universitäten oft der Fall ist. Hier nur zwei Beispiele von vielen:

Eine grundlegende Einführung in die Theorie der Datenbanken findet man unter:

www.bib.informatik.th-darmstadt.de/TTT/AI.HTM

Eine Einführung in SQL, in der Regel als Begleitung zu Lehrveranstaltungen, findet man z.B. unter:

rzdspc77.informatik.uni-hamburg.de/Frauen/Admina/Beitraege /DB+WWW/sql-tutor/tuto2.htm

In diesem Zusammenhang sei auf die verschiedenen (Datenbank-)Bücher verwiesen, die bei einigen US-Verlagen auf deren Websitzes zum kostenlosen Lesen angeboten werden und die ihren (legalen) Weg auch auf CD-Sammlungen finden. So war das Buch »SQL in 21 Tagen« (Sams-Verlag) als HTML-Dokument auf der Monats-CD der Zeitschrift VBA-Magazin (*www.vba-magazin.de*) zu finden.

E.4 Newsgroups

Newsgroups dienen dem weltweiten Austausch von Programmierern und Anwendern. Auch Microsoft unterhält (auf dem Newsserver *msnews.microsoft.com*) eine Vielzahl öffentlicher Newsgroups, die aber praktisch ausnahmslos »unmoderiert« sind, d.h. ohne Mitwirkung von Microsoft-Mitarbeitern stattfinden. Die Diskussion in einer Newsgroup sieht so aus, daß man eine Frage stellt, ein Problem beschreibt oder einen Kommentar abgibt und hofft, daß andere Teilnehmer einem weiterhelfen können. Wer noch nie an einer Newsgroup teilgenommen hat, sollte sich über die »Spielregeln« im klaren sein:

- ✘ Für den Zugriff auf eine Newsgroup wird ein (beliebiger) Internet-Zugang und ein sogenannter Newsreader benötigt (ein Programm, das den In-

halt von Newsservern darstellen kann), wie er z.B. in Outlook Express enthalten ist.

- ✘ Newsgroups werden auf einem Newsserver gehalten, der eine feste Adresse besitzt. Die Adresse des Microsoft-Newsservers lautet: *msnews.microsoft.com*. Diese Adresse muß beim Einrichten eines Kontos bei Outlook Express angegeben werden.
- ✘ Die Teilnahme an Newsgroups ist kostenlos, freiwillig und im allgemeinen völlig ungefährlich (es sind keine Chats). Newsgroups sind kein Ersatz für den (fehlenden) Supportvertrag mit dem Hersteller. Auch ist der Hersteller nicht verpflichtet, Newsgroups zu betreiben bzw. deren Qualität zu überwachen. Es gibt daher keine Antwortgarantie.
- ✘ Wer etwas in Anspruch nimmt, sollte ab und zu auch etwas geben. Mit anderen Worten, wem bereits in einer Newsgroup geholfen wurde, der sollte auch bereit sein, anderen zu helfen.
- ✘ Der Umgangston in einer Newsgroup ist informell, d.h., es gibt meistens keine formale Anrede (die Nachricht richtet sich ja an alle und »Sehr geehrte Damen und Herren« wäre sicher nicht die passende Anrede), und es ist auch nicht üblich, sich für eine Hilfe zu bedanken (alleine, um das Nachrichtenaufkommen nicht unnötig zu steigern). Dennoch sollte man die Mithilfe anderer nicht als »kostenlose Serviceleistung« mißverstehen und seine Nachrichten entsprechend formulieren (der Ton macht die Musik). Newsgroups stellen ein soziales Gefüge dar, in dem einzelne Teilnehmer schnell in einer bestimmten Schublade landen. In jeder Newsgroup gibt es einen »harten Kern« häufig teilnehmender Mitglieder, die u.a. den Umgangston der Newsgroup beeinflussen. Hat man es sich mit diesen Teilnehmern verscherzt (was in den technischen Newsgroups aber nur selten vorkommen dürfte), muß man mit verbalen Attacken (das sogenannte »flaming«) rechnen. Aber wie gesagt, in den relativ anonymen und sprachlich sehr moderaten Microsoft-Newsgroups kommt dies sehr selten vor, so daß Neulinge nichts »befürchten« müssen.
- ✘ Wie in allen Newsgroups gilt auch in den Microsoft-Newsgroups strenge Etikette, die nicht von Microsoft, sondern von den Teilnehmern »überwacht« wird. So ist es absolut verpönt, für irgend etwas Kommerzielles zu werben (die eigene Homepage geht gerade noch) oder andere Teilnehmer aus irgendeinem Grund »anzumachen« (das »flaming« stellt eine gewisse Ausnahme dar, doch erfordert das Erkennen der feinen Trennlinie schon etwas Erfahrung). Auch themenfremde Fragen kommen nicht gut an (etwa eine Visual-Basic-Frage in einer ADO-Newsgroup) und werden meistens mit entsprechenden Belehrungen geahndet.

- ✘ Doch es gibt auch viele gute Seiten. In den meisten Microsoft-Newsgroups gibt es die MVPs. Diese »Most Valuable Players« sind freiwillige (und ehrenamtliche) Helfer mit sehr viel Erfahrung in dem speziellen Bereich. Sie kümmern sich um Fragen, auf die es zunächst keine Antworten gibt, sind stets hilfsbereit und können auch direkt angesprochen werden. MVPs werden von Microsoft-Mitarbeitern »ernannt«. Meistens sind es Newsgroup-Teilnehmer, die über einen längeren Zeitraum durch besondere »Auskunftsfreudigkeit«, Hilfsbereitschaft und Kompetenz aufgefallen sind. Es gibt sogar eine Webseite einiger Microsoft-MVPs: *www.mvps.org*.

Die folgenden Microsoft-Newsgroups gibt es zum Thema Datenbankprogrammierung:

- ✘ *microsoft.public.data.ado*
- ✘ *microsoft.public.vb.database.ado*
- ✘ *microsoft.public.vb.database*
- ✘ *microsoft.public.de.vb*

Auch die Fachzeitschrift BasicPro (*www.basicworld.com*) unterhält eine Reihe von Newsgroups (die Adresse des Newsservers lautet: *news.basicworld.com*), die direkt über die Webseite angesteuert werden können. Für Datenbankprogrammierer ist:

- ✘ *news.basicworld.com/basicworld.public.vb.datenbank*

interessant. Diese Newsgroup ist sehr aktiv, deutschsprachig (was auch ein Argument ist), und es gibt viele kompetente und hilfsbereite Teilnehmer. Sie scheint daher wohl eines der besten Foren zum Thema Datenbankprogrammierung zu sein.

E.4.1 Ein Tip zum Schluß

Namenskonventionen spielen vor allem bei der Datenbankprogrammierung eine Rolle. Eine aktuelle Übersicht der Microsoft-Namenskonventionen finden Sie u.a. unter *http://msdn.microsoft.com/library/devprods/vs6/vb/html/vbconobjectnamingconventions.htm*.

Stichwortverzeichnis