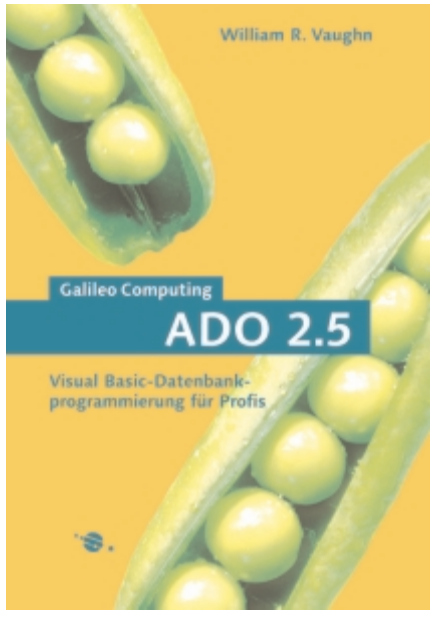
William R. Vaughn

ADO 2.5

VB-Datenbankprogrammierung für Profis



Galileo Press

Inhalt

Einleitung

Ich kann Ihnen gar nicht sagen, wie viele Leute gefragt haben, wann denn die nächste Auflage **des Hitchhiker's Guide to Visual Basic und SQL Server** geschrieben wird. Tatsächlich hat es seit dem Erscheinen des Buches so viele technologische Weiterentwicklungen gegeben, dass eine Aktualisierung des Wälzers dringend nötig ist. Seit Visual Basic 6.0 wurde ADO wenigstens acht Mal geändert, daher sind viele der Beispiele und Erläuterungen der 6. Auflage des **Guide** mittlerweile nicht mehr aktuell. Doch da Visual Basic 7.0 noch nicht fertig ist, ist es für eine Aktualisierung des **Guide** noch zu früh.

Als ich die Gelegenheit erhielt, ein Buch für APress zu schreiben, erkannte ich, dass dies eine perfekte Gelegenheit sein könnte, sich auf das Thema zu konzentrieren, bei dem es die einschneidendsten Änderungen gegeben hat – ADO. Ich könnte meine Leser mit den neuesten Features und Verfahren vertraut machen, ohne mich um das zukünftige Aussehen von Visual Basic kümmern zu müssen. Es gibt jedoch bereits eine Menge Bücher zur ADO-Programmierung. Was ich wollte, war ein Ansatz, der am ehesten auf Sie zugeschnitten ist, die Entwickler an vorderster Front. So entstand das vorliegende Buch **ADO 2.5**.

Einige von Ihnen werden bisher mit anderen Programmierschnittstellen gearbeitet haben, beispielsweise mit DAO (Data Access Objects) oder RDO (Remote Data Objects). Bei Gelegenheit werde ich Empfehlungen für den einfachsten Übergang von diesen Modellen geben, dieses Thema soll jedoch nicht erschöpfend erläutert werden. Einige von Ihnen arbeiten wahrscheinlich mit früheren Versionen von ADO und sind dabei, auf die neueste Version von ADO zu wechseln (gewollt oder ungewollt), und dieses Buch richtet sich an Sie.

Der Schwerpunkt des vorliegenden Buches liegt auf ActiveX Data Objects, Version 2.5 (ADO), es sollen jedoch die Erfahrungswerte einfließen, die von unzähligen Entwicklern, Trainern und Support-Experten gemacht und an mich weitergegeben wurden. Es sind genau diese Menschen, die den Weg für dieses Buch geebnet haben. Ich habe die letzten 14 Jahre mit Schreiben, Training und Codedokumentation verbracht, nicht aber mit dem Schreiben von Anwendungen oder mit der Anwendungsunterstützung, was ich in den ersten zwölf Jahren meiner Karriere tat. Aufgrund dessen muss ich stets eng mit Personen zusammenarbeiten, die genau dies tun. Wenn ich kann, helfe ich bei Problemen, die im wirklichen Leben auftreten, aber genauso häufig hilft mir die Entwicklergemeinde mit innovativen und erprobten Verfahren für die Verwendung der Tools. Diese Beispiele und »besten Vorgehensweisen« sind Methoden, die von Hunderten oder Tausenden der talentiertesten Entwickler in der ganzen Welt diskutiert, verbessert und eingesetzt werden. Und ja, viele dieser Ideen stammen von internationalen Entwick-

lern außerhalb der USA. Während einige der Ideen von Microsoft kommen, wodurch ich in gewissem Maße an die Geheimhaltung gebunden bin, stammen viele Ideen aus anderen Quellen, von Entwicklern, die sich mit dem begnügen müssen, was die Industrie bietet. Ich stehe in ständiger Kommunikation mit den Produktmanagern, Entwicklern und Produktsupportteams bei Microsoft, um auf der Grundlage Ihrer (und meiner) Anfragen die verfügbaren Tools und Schnittstellen zu verbessern.

ADO setzt da an, wo der **Guide** aufhört – beim Web. Es hat viele Diskussionen darüber gegeben, welche die beste Verwendung von ADO in ActiveX Server Pages (ASPs) sowie von Browserseiten in Visual Basic Script sei. Obwohl der Hauptschwerpunkt dieses Buches die neuen ADO 2.5-Features sind, werden an passender Stelle auch Technologien wie XML und MSXML besprochen. Auch wenn dieses Buch keine Referenz für den Programmierer ist, werden – wie im **Guide** – dennoch viele der einzelnen Objekte, Eigenschaften und Methoden angesprochen. Dem ganzen Webhype zum Trotz, bietet das vorliegende Buch eine umfassende Erläuterung zu Client/Server-, Middletier- und webbasierten ADO-Architekturen, zu Codeentwicklung und Bereitstellung sowie zu jedem dieser Themen eine spezifische Leistungsanalyse.

Das Thema Leistung zieht sich wie ein roter Faden durch das vorliegende Buch. Für mich bedeutet Leistung jedoch mehr als das Schreiben von Code mit kurzen Ausführungszeiten. Für mich bedeutet Leistung Code, der dem Entwickler die Möglichkeit zum effizienten Arbeiten gibt – das Schreiben von verschiedenen Lösungen in weniger Zeit und bei weniger Kosten, nicht jedoch unbedingt das Schreiben von mehr Codezeilen. Dies ist besonders wichtig für Entwickler, die im Team arbeiten. Sehr häufig wurden die Programme, an denen wir schreiben, ursprünglich von anderen Entwicklern entworfen und implementiert – oder wenigstens Teile dieser Programme. Dies bedeutet, dass Sie mit dem Design, den Methoden und der Dokumentation anderer Entwickler klar kommen müssen, vorausgesetzt natürlich, es ist eine Dokumentation vorhanden. Und andere wiederum müssen sich mit dem beschäftigen, was Sie schreiben. Wir sehen uns ständig der Herausforderung gegenüber, Code zu erstellen, der zum Zweck der Integration, für den Support oder das Debuggen an andere weitergegeben werden kann. Es gibt viele Methoden, um diesen Prozess zu erleichtern, leichter für Sie und leichter für diejenigen, die mit Ihrem Code arbeiten müssen.

Dieser Prozess erfordert jedoch etwas, das Sie nicht kaufen können, auch nicht bei Microsoft – Disziplin. Nein, ich spreche nicht von den Zeiten, in denen Oberlehrer mit gestärktem Kragen und Rohrstock durch die Schulzimmer schritten. Ich spreche davon, dass innerhalb der Unternehmen Standards erarbeitet und anschließend eingehalten werden müssen. Es bedeutet, dass Spezifikationen geschrieben

werden, die bei der Programmierung berücksichtigt werden. Es bedeutet auch, dass Sie nicht der Versuchung unterliegen, außerhalb dieser Spezifikationen zu programmieren, weil Sie eine viel bessere Lösung kennen als die Deppen, die diese Spezifikationen aufgestellt haben. Das mag der Fall sein, aber wenn Sie es nicht schaffen, Ihre Kollegen und Manager zu einer Änderung der Spezifikationen zu bewegen, sind Sie mit dieser Einstellung allenfalls kontraproduktiv. Wenn Sie sich bei der Codeerstellung nicht an die Spezifikationen halten, kann der innovativste Code weder mit dem Code anderer Entwickler noch in zukünftigen Projekten eingesetzt werden. Disziplin bedeutet auch, seine Arroganz unter Kontrolle zu halten. Das Motto »Take it or leave it« ist gegenüber dem Endkonsumenten keine sehr produktive Haltung, dennoch ist sie in vielen Köpfen zu finden. Wenn Sie nach Codezeile bezahlt werden, können Sie diese Anregungen natürlich ignorieren. Sie sollten jedoch nicht darauf hoffen, ein zweites Mal engagiert zu werden.

Wir werden uns darüber hinaus der Verringerung des COM-Overheads, der Reduzierung von Lastzeiten in Anwendungen sowie einigen Verfahren zur besseren Nutzung von LAN, WAN und Web sowie RAM, Festplattenspeicher und anderen Systemressourcen zuwenden. Wir werden versuchen, Programme zu schreiben, die von anderen Entwicklern verstanden und unterstützt werden können, ohne dass das Barbecue am Samstagnachmittag ausfallen muss. (Das wäre dann wohl ein gegrillter SQL Server).

1 Arbeiten mit ADO

Sie lesen dieses Buch nicht wegen der Witze – das hoffe ich wenigstens. Ich hoffe, Sie lesen dieses Buch, da es Anregungen zum Schreiben effizienter ADO-Datenzugriffsanwendungen enthält. Das erste Kapitel bietet einen Überblick über einige Grundlagen – die Dinge, die für den Einsatz von ADO Voraussetzung sind. Ich setze hierbei voraus, dass Sie in den meisten Fällen wissen, was Sie tun und lasse daher die unvermeidlichen Schritt-für-Schritt-Erklärungen der Bücher für Anfänger weg. Wenn Sie nicht wissen, was ADO ist, wie das zugehörige Objektmodell aussieht oder die einzelnen Objekte zueinander in Beziehung stehen, lesen Sie den Anhang am Ende dieses Buches.

1.1 Auswahl der »richtigen« ADO-Version

Eine Herausforderung, der wir uns zu stellen haben, ist die Frage, welche die »richtige« ADO-Version ist. Da es sehr viele Versionen gibt und diese z. T. sehr unterschiedlich arbeiten, ist eine solche Entscheidung häufig nicht einfach zu treffen. In einigen Fällen haben Sie keine Wahl – Sie müssen die Version verwenden, die der Kunde installiert hat oder gerne einsetzen möchte. Wenn Sie eine Wahl haben, muss auch bedacht werden, wie zukünftige Versionen die entwickelte Software ändern (beschädigen). Machen wir zum besseren Verständnis der riesigen Auswahlmöglichkeiten einen kleinen Ausflug in die ADO-Geschichte.

Seit im Sommer '98 Visual Basic 6.0 herauskam, wurde ADO verschiedene Male umgemodelt und nahm teilweise bizarre Formen an. Zum Lieferumfang von Visual Basic 6.0 gehörte eine frühe Version von ADO 2.0, ADO wurde geändert als SQL Server 7.0 veröffentlicht wurde, als der Internet Explorer 5.0 herauskam, als Office 2000 auf den Markt gebracht wurde, und danach wurden noch mindestens zwei Aktualisierungen vorgenommen. Während einige der Änderungen von weniger großer Bedeutung waren, waren andere dies sehr wohl, und Visual Basic (Visual Studio) wurde diesen Neuerungen nicht schnell genug gerecht.

Vor ADO 2.0 gab es die so genannte Microsoft ActiveX **Recordset** Library (ADOR). Zunächst sollte ADOR eine »Lightversion« von ADO darstellen – es war keine vollständige Installation nötig, daher hinterließ die Installation weniger Spuren, es war weniger RAM erforderlich und es wurden bessere Downloadzeiten erzielt. Es stellte sich jedoch heraus, dass sich der ganze Aufwand nicht lohnte, daher wurden mit ADO 2.0 die beiden Bibliotheken zusammengeführt. Deshalb sind weitere Verweise auf die ADOR-Bibliothek sinnlos.

Microsoft vertreibt ADO als MDCA-Paket (Microsoft Data Access Components), einem 6,35 MB großen, ausführbaren Paket (**mdac_typ.exe**), das ADO, ODBC und OLE DB-Treiber und -Provider enthält. Ja, Microsoft bündelt all diese Komponenten in einem Nimm-eins-und-bekomme-alles-Paket (ob Sie nun wollen oder nicht). Das Zusammenfassen und Installieren all dieser Einzelkomponenten zahlt sich im Hinblick auf die Stabilität jedoch aus.

Die verschiedenen MDAC-Bestandteile, nicht jedoch unbedingt alle, werden über viele, viele verschiedene Anwendungen installiert – Anwendungen von Microsoft und Drittanbietern. Wenn Sie Glück haben, führen diese Anwendungen zu einer tatsächlichen »Installation« von MDAC und nicht nur zu einem bloßen Kopieren der Dateien und der Hoffnung, dass schon alles gut gehen wird (wie dies bei einigen Anwendungen der Fall zu sein scheint). Ich werde Ihnen in einem späteren Abschnitt in diesem Kapitel zeigen, wie MDAC installiert wird.

Obwohl im Herbst '99 die »golden« bzw. »general release« von MDAC herauskam (2.1.2.4202.3 [GA]), enthalten weder Visual Basic SP3, das Visual Studio Plus Pack noch das Windows 2000 Readiness Kit diese Version von MDAC. Das Visual Basic-Team hat mir gesagt, dass Visual Basic 6.0 SP4 (voraussichtliches Erscheinungsdatum Sommer 2000) eine Aktualisierung von Visual Basic 6.0 mit sich bringen soll, anhand derer die Probleme behoben werden, die durch die Schnittstellen- und Funktionsänderungen in ADO 2.1 hervorgerufen werden.

Ich lungere häufig auf verschiedenen öffentlichen und internen Listenservern herum (beispielsweise **VBDATA-L@peach.ease.lsoft.com**), wo verschiedene Fragen zu ADO diskutiert werden. Eine häufige Frage lautet: »Wird bei der Installation von Internet Explorer 5.0 ADO 2.1 installiert?« In gewisser Weise. Über den Internet Explorer 5.0 wird ADO 2.1 installiert und verwendet, die Provider und Treiber des vollständigen MDAC-Setups werden jedoch **nicht** installiert. Daher führt das Installieren von Internet Explorer 5.0 nicht wirklich zu einer (vollständigen) Installation von ADO – Sie müssen eine Verteilung und Installation des MDAC-Setups mit Ihrem ADO 2.**x**-Clientprogramm vornehmen.

1.1.1 ADO 2.5 kommt auf den Markt

Nun, da ADO 2.5 auf den Markt gekommen ist, stellt diese ADO-Version einen integralen Bestandteil von Windows 2000 dar, kann jedoch für ältere Windows-Versionen weiterhin separat von der Website **http://microsoft.com/data** heruntergeladen werden. (Zufälligerweise ist diese Site auch der beste Ort, um sich über die neuesten Versionen von ADO und MDAC zu informieren.) Die Integration von ADO in Windows fand zu dem Zweck statt, dass sich ADO nicht länger mit jeder Mondphase ändert – es ist für die Entwickler schwieriger geworden, Änderungen an ADO vorzunehmen, selbst dann, wenn dies erforderlich ist. Es sieht so aus, als

ob ein ADO-Upgrade/Fix nur über eine Betriebssystemaufrüstung erfolgen kann. Dies bedeutet auch, dass die Zugpferde von Microsoft (Microsoft Office, der Internet Explorer, Visual Studio und andere) MDAC nicht so mir nichts, dir nichts ändern können – und das ist die gute Nachricht.

Warnung Versuchen Sie nicht, den Office 2000 Installer zum Entfernen von ADO einzusetzen. Dieses Vorgehen führt zu einem schwerwiegenden Fehler und dazu, dass Sie Windows 2000 neu installieren müssen.

Das eigentliche Problem all dieser Änderungen ist ein grundlegendes. Bei der Entwicklung von COM-Komponenten unterwerfen Sie sich als Entwickler einer Reihe von anerkannten Implementierungsstandards. Eine dieser Anforderungen lautet, dass zukünftige Versionen der Komponente alle vorherigen Versionen unterstützen sollten (zumindest die jeweils letzten Versionen). Obwohl also der Code, über den die veröffentlichten Schnittstellen implementiert werden, Änderungen unterliegen kann, werden die Schnittstellennamen, die Eigenschaften, Methoden, Ereignisse und die übergebenen Argumente **nicht** geändert.

Mit ADO 2.1 wurden diese Regeln gebrochen. ADO 2.0-Anwendungen stürzten ab oder zeigten ein sonderbares Verhalten, nachdem der Internet Explorer 5.0 oder Office 2000-Anwendungen installiert wurden, die eine Aktualisierung von ADO 2.0 auf ADO 2.1 durchführten. Noch schlimmer, ADO wurde in den folgenden Monaten weiter geändert. Selbst wenn sich die Schnittstellen nicht änderten, wurden Änderungen an MDAC vorgenommen. In einigen Fällen wurden durch »Verbesserungen« der ODBC-Treiber Sicherheitsbeschränkungen geändert – die dazu führten, dass für Benutzer plötzlich Sperren galten, die vorher problemlos mit Hilfe der Standardeinstellungen auf ihre Datenbanken zugreifen konnten. Obwohl ADO 2.1 und 2.5 ADO 2.0-Typbibliotheken enthalten, die älteren Anwendungen das Verwenden der alten, gewohnten Schnittstellen ermöglicht, führte das Zurückgeben von ADO 2.1-Recordsets (oder 2.5-Recordsets) anstelle des ADO 2.0-Recordsets durch einige überarbeitete (»verbesserte«) Ereignisbehandlungsroutinen zum Absturz der Anwendungen. Einige der schwerwiegendsten Fehler traten in Visual Basic selbst auf. Keines der Visual Database Tools kann mit ADO 2.1 oder höheren Versionen fehlerfrei verwendet werden. Dies führte zu einer Verstümmelung des Datenumgebungs-Designers. Zusätzlich wurden das ActiveX-Datensteuerelement, der Datenformular-Assistent und der Datenobjekt-Assistent beeinträchtigt.

Zum Zeitpunkt der Entstehung dieses Buches war noch nicht klar, was Microsoft und das MDAC-Team zur Lösung dieser Probleme unternehmen werden. Eine Umstellung auf die alte Version kommt sicherlich nicht in Frage. Ich habe ver-

sucht, das Visual Studio-Team dazu zu bewegen, eine Aufrüstung auf Visual Basic 6.0 vorzunehmen, um eine vollständige Unterstützung von ADO 2.5 zu gewährleisten – ob dieser Schritt durchgeführt wird, steht jedoch noch in den Sternen. Vielleicht werden all diese Probleme mit Visual Studio 6.0 SP4 gelöst. Mir wurde gesagt, dass mit dem Erscheinen von Visual Basic 7.0 abschließend sämtliche Schwierigkeiten beseitigt sein sollen. Bis dahin bin ich wahrscheinlich längst in Rente ...

1.2 Die neuen Features von ADO 2.5

Mit ADO 2.5 wird der eigentliche Anspruch von ADO fortgeführt – der Anspruch, **die** universelle Informationsschnittstelle bereitzustellen. Diese Version von ADO enthält nicht nur viele Bugfixes, sondern unterstützt auch zwei völlig neue Objekte, das **Stream**- und das **Record**-Objekt. Sie stoßen auf das **Record**-Objekt, sobald Sie dieses bei Verarbeitungszeit der Anweisungen versehentlich anstelle des **Recordset**-Objekts auswählen. Aber Sie werden es kennen lernen.

Anmerkung Die Beispiele in diesem Buch wurden mit Hilfe von ADO 2.5 auf einem Windows 2000- bzw. einem Windows 98-System geschrieben. Ich erwarte nicht, dass eine Kompilierung oder Ausführung mit älteren ADO-Versionen möglich ist.

1.2.1 »Record«- und »Stream«-Objekte

Mit ADO 2.5 wird das **Record**-Objekt eingeführt, das Dinge wie Verzeichnisse und Dateien in einem Dateisystem sowie Dokumente, Ordner und Nachrichten in E-Mail-Systemen repräsentieren und verwalten kann. Ein Record (Datensatz) kann auch eine Zeile in einem Recordset (einer Datensatzmenge) darstellen, sollte jedoch nicht mit dem **Recordset**-Objekt verwechselt werden, da dieses abweichende Methoden und Eigenschaften aufweist – darüber hinaus verhalten sich einige der gemeinsam genutzten Eigenschaften und Methoden unterschiedlich.

Das neue **Stream**-Objekt bietet die Mittel zum Lesen, Schreiben und Verwalten des binären Bytestreams oder Textes, aus dem sich eine Datei oder ein Nachrichtenstrom zusammensetzt. Stellen Sie sich einen Stream als eine speicherinterne Datei vor. Wie häufig mussten Sie einen Datenblock auf die Festplatte schreiben, obwohl Sie diesen eigentlich nur für eine Sekunde speichern oder lediglich an eine andere Komponente oder Schicht übergeben wollten?

1.2.2 URLs anstelle von Verbindungszeichenfolgen

In ADO 2.5 wird des Weiteren als Alternative zur Verwendung von Verbindungszeichenfolgen und Befehlstext die Verwendung von URLs (Uniform Resource Locators) zur Benennung von Datenspeicherobjekten eingeführt. URLs können sowohl mit den vorhandenen **Connection**- und **Recordset**-Objekten als auch mit den neuen **Record**- und **Stream**-Objekten eingesetzt werden.

ADO 2.5 unterstützt darüber hinaus OLE DB-Provider, die ihre eigenen URL-Schemata erkennen. Der OLE DB-Provider für die Internetveröffentlichung beispielsweise, der auf das Windows 2000-Dateisystem zugreift, erkennt das vorhandene HTTP-Schema.

1.2.3 Spezielle Felder für Dokumentquellenprovider

Mit ADO 2.5 wird eine spezielle Klasse von Providern vorgestellt, die so genannten »Dokumentquellenprovider«, die zum Verwalten von Verzeichnisordnern und Dokumenten eingesetzt werden können. Wird ein Dokument durch ein **Record**-Objekt repräsentiert, oder stellt ein **Recordset**-Objekt einen Dokumentenordner dar, werden diese Objekte durch den Quellenprovider mit einem eindeutigen Feldersatz versehen, der die Merkmale des Dokuments beschreibt. Diese Felder bilden einen Ressourcenrecord oder ein Ressourcenrecordset.

1.2.4 Neue Referenzthemen

Zur Unterstützung der **Record**- und **Stream**-Objekte wurden ADO 2.5 verschiedene neue Eigenschaften, Methoden und Ereignisse hinzugefügt. Das **Stream**-Objekt wird in Kapitel 9 besprochen. Diese neuen Eigenschaften unterstützen ADO bei der Definition von Dokumenten und Dateien. Die Methoden ermöglichen das programmatische Kopieren, Hinzufügen, Löschen und Bearbeiten von Zeichen, Dateien, Dokumenten oder anderen Datenbereichen von der Quelle zum Ziel.

1.3 Der ADO Component Checker

Sie haben also vor kurzem das neueste Visual Basic 6.0 SP3 installiert. Da Sie es erst letzte Woche erworben haben, gehen Sie davon aus, dass es die aktuellste ADO-Version umfasst, richtig? Tja, aber wissen Sie **mit Sicherheit**, welche MDAC-Version (also ADO-Version) auf Ihrem Entwicklungssystem, Ihrem Server oder dem Clientsystem installiert ist? Möchten Sie es gerne herausfinden? Nun, Yoda würde sagen: »Du wirst ... du wirst«. (Zufällig wird mit Visual Basic 6.0 SP3 eine **ältere** Version von ADO 2.1 installiert – nicht die endgültige GA-Version von ADO 2.1).

Sie haben Glück – unter dem Microsoft-Link Component Checker tool finden Sie alles zu MDAC (vielleicht sogar einiges, was Sie noch gar nicht kennen). Der Component Checker kann auf den folgenden Betriebssystemen ausgeführt werden: Microsoft Windows 98, Windows NT 4.0 und Windows 2000. Es werden ausschließlich 32-Bit- und 64-Bit-Plattformen unterstützt.

Der Component Checker ist ein Tool, das benutzerdefiniert angepasst werden kann und folgende Aufgaben ausführt:

- Ermitteln der aktuellen MDAC-Installation auf einem Computer.
- Erstellen einer Reihe von Berichten zu den Dateien, die im Zusammenhang mit der MDAC-Installation gefunden wurden.
- (Optional) Entfernen der aktuellen MDAC-Installation nach Auflistung der .dll-Konflikte und Ermitteln von Programmen, die auf eine vorhandene DLL verweisen.

Das Durchführen dieser Aufgaben ist nicht immer möglich. Nachdem eine ADO-Version installiert wurde, beispielsweise ADO 2.1, funktionieren ADO-abhängige Anwendungen nicht mehr ordnungsgemäß mit älteren ADO-Versionen (oder gar nicht mehr). Nach meinem Verständnis ist nach der Installation von ADO 2.5 eine Deinstallation **nicht mehr möglich** – ADO wird zu einem integralen Bestandteil des Betriebssystems.

Anmerkung Ich habe ADO 2.1 (GA) auf meinem neuen Visual Basic 6.0 SP3-System installiert und den Component Checker ausgeführt. Es tauchten mehr als ein Dutzend »Difugelties«[1] auf – eine Reihe fehlender Registrierungseinträge und ein ganzer Haufen Versionsfehler. Ich versuchte das Gleiche nach der Installation von ADO 2.5. Dito. Beängstigend. Sie arbeiten daran – sagen sie jedenfalls.

1.4 Installieren von ADO

Wenn Sie Visual Basic 6.0, Visual Studio 6.0, Office 2000 oder den Internet Explorer 5.0 verwenden, ist ADO bereits installiert. Auf dem System Ihres Kunden ist ADO jedoch möglicherweise nicht installiert, und die Chance, dass auf Ihrem und dem Kundensystem die gleiche Version ausgeführt wird, ist eher gering. Wenn Sie nicht kontrollieren, welche Version Ihr Kunde auf seinem System installiert hat, funktioniert Ihre ADO-Anwendung möglicherweise nicht besonders lange. Wenn der Kunde die Microsoft-Site für die Windows-Aktualisierung besucht und ein Patch oder eine neue Anwendung herunterlädt bzw. einfach nur

1. Difugelties: (althergebrachter Computerausdruck) Bug, Fehler, FUBAR (Fucked Up Beyond All Recognition/Repair).

eine Aktualisierung des Internet Explorers durchführt, kann dies zur Installation einer neueren Version von ADO führen. Daher können Sie nicht lediglich eine ältere ADO-Version (oder selbst eine aktuelle Version) auf dem Zielsystem beibehalten, selbst dann nicht, wenn Sie diese in einen gesperrten Bereich platzieren.

Gibt es Hoffnung für uns ADO- und COM-Benutzer? Nun ja, die Windows 2000-Technologie soll uns hier helfen. In der aktuellsten Version der MSDN-News (Januar/Februar 2000, **http://msdn.microsoft.com/voices/news/**) gibt es einen interessanten Artikel, der mich zu der Annahme bringt, dass über dieses Problem zumindest nachgedacht wurde. Der Artikel »The End of DLL Hell« lässt vermuten, dass die neue Technologie des Windows Installers und der Windows-Dateischutz (Windows File Protection) eine versehentliche (oder beabsichtigte) Beschädigung der DLLs verhindern. Nicht erwähnt wird, ob ADO als eine speziell geschützte System-DLL betrachtet wird. Zum jetzigen Zeitpunkt gibt es Hinweise darauf, dass dies nicht der Fall ist – dies muss vielleicht geändert werden.

Tipp Windows 9x-Benutzer: DCOM95 für Windows 95 muss **vor** der Installation von MDAC 2.0x oder 2.1x installiert werden. MDAC installiert Komponenten, die auf DLLs basieren, die von DCOM95 zur ordnungsgemäßen Registrierung installiert werden. DCOM95 ist unter Windows NT 4.0 nicht erforderlich. In einigen Fällen ist DCOM auf Windows 98-Computern nicht installiert. DCOM95 für Windows 95 steht unter **http://www.microsoft.com/com/tech/dcom.asp** für den Download zur Verfügung.

Gelegentlich möchten Sie die Installation von **mdac_typ.exe** selbst steuern. Hier ein Tipp. Probieren Sie zur Verwendung der wenig dokumentierten Funktion für eine unbeaufsichtigte Installation von MDAC auf einem Benutzercomputer diese Optionsflags aus:

```
Mdac_typ.exe /q /C:"Setup QN1"
```

Obwohl es sich nicht wirklich um eine unbeaufsichtigte Installation handelt, wird doch das gewünschte Ziel erreicht, und die Benutzer sehen, was vor sich geht (als ob es sie interessieren würde).

1.5 Bereitstellen von ADO

Das Bereitstellen von ADO auf dem Zielsystem stellt für den ADO- oder COM-Entwickler heutzutage eine sehr komplexe und problematische Aufgabe dar. Während die Analyse, der Entwurf und das Testen zu üblichen Schritten bei der Entwicklung geworden sind, wird die Problematik der Anwendungsbereitstellung häufig vernachlässigt.

Das Bereitstellen einer COM-Lösung umfasst im Allgemeinen viele Installationsaufgaben, u.a.:

- Verteilen von Dateien
- Erstellen von Registrierungseinträgen
- Erstellen von Desktopsymbolen
- Aktualisieren von Softwarekomponenten auf Clientrechnern

Der Paket- und Weitergabe-Assistent von Microsoft Visual Basic sowie viele Drittanbieterprodukte sind in der Lage, diesen Bereitstellungsprozess zu automatisieren. Dennoch nutzen viele dieser Tools nicht die Vorteile der neuesten Technologien zur Durchführung einer konzeptionell gesehen einfachen Aufgabe – der Softwareinstallation!

Diese Tools erfordern beispielsweise häufig, dass der Autor Installationsskripts zur Festlegung vordefinierter Pfade und Installationsoptionen erstellt. Das Entwickeln von Installationsskripts wird schwieriger, je komplizierter eine Lösung ist und je mehr sie von anderen Lösungen abhängt. Das Schreiben von Installationsskripts, bei denen drei Dateien kopiert werden, ist einfach, das Schreiben von Skripts für eine Lösung, bei der Dateien mit anderen Anwendungen gemeinsam genutzt werden, ist dagegen weniger einfach. Fügen Sie noch eine Logik für die Deinstallation hinzu, und die Skriptverwaltung wird unmöglich.

Installationsprogramme weisen ein weiteres Problem auf. Wie erstellen Sie eine Installationsprozedur, bei der sämtliche Unterschiede zwischen den einzelnen Clientrechnern berücksichtigt werden? Hierbei reichen die Varianten von den installierten Anwendungen über die aktivierten Betriebssystemfunktionen bis zu den Unterschieden, die während einer Installationsprozedur ermittelt werden können, und alles zusammen scheint eine unüberwindliche Aufgabe darzustellen. Wenn Skripts für Installationsprozeduren erstellt werden, muss sich der Autor vor der Installation die jeweilige Umgebung ansehen. Mit anderen Worten, Sie müssen wissen, wie das Zielsystem aussieht, was installiert ist, was nicht installiert ist und wie viele Ressourcen zur Verfügung stehen. Das Erhöhen der Informationsmenge, die von Installer und Betriebssystem gemeinsam genutzt wird, kann diese Aufgabe weniger kompliziert machen.

Das Erstellen von Installationsskripts ist schwierig, für die meisten Installationstools jedoch erforderlich. Der Visual Studio Installer vermeidet diesen Schritt und weitere der mit dem Erstellen robuster, effizienter Installationsprogramme zusammenhängende Schwierigkeiten. Bei gemeinsamer Verwendung mit der Windows Installer-Technologie scheint der Visual Studio Installer die beste Wahl für das Erstellen schneller, verwaltbarer, robuster Installationsanwendungen für Visual Studio-basierte Lösungen darzustellen.

1.5.1 Die Installer von Visual Studio und Windows

Um die in Zusammenhang mit Installationsprogrammen auftretenden Probleme zu lösen, hat Microsoft zur Bereitstellung von Anwendungen den Windows Installer eingeführt. Die Windows Installer-Technologie ist Bestandteil von Microsoft Windows 2000 und steht auch für die Betriebssysteme Microsoft Windows 95, Windows 98 und Windows NT 4.0 zur Verfügung. Eine automatische Installation findet immer dann statt, wenn Sie Office 2000 oder eine beliebige Anwendung installieren, die den Visual Studio Installer verwendet.

Der Microsoft Windows Installer ist, im Gegensatz zu den meisten anderen Entwicklungstools für Installationsprogramme, nicht skriptbasiert. Statt dessen dient der Windows Installer als Host für datengesteuerte Installationspakete. Der Windows Installer verwendet »Datenbündel«, so genannte Pakete, die Informationen zu den für eine Lösung erforderlichen Komponenten enthalten. Der Windows Installer verwendet die Paketinformationen zur Beantwortung der folgenden Fragen:

- Wie sind die Komponenten auf einem Rechner installiert?
- Welche Registrierungseinstellungen müssen erstellt oder bearbeitet werden?
- Wie lautet der Standardspeicherort der Dateien?
- Müssen Desktopsymbole erstellt werden?
- Sind weitere Informationen zur Installationszeit erforderlich?

All diese Informationen werden ermittelt, ohne dass der Entwickler auch nur eine Zeile Installationscode schreiben muss. Das interaktive, datengesteuerte Modell bietet gegenüber dem traditionellen, skriptbasierten Ansatz erhebliche Vorteile. Bei der datengesteuerten Installation werden viele der Schwierigkeiten umgangen, die bei einem skriptbasierten Installer auftreten, gleichzeitig wird die Arbeitslast für Entwickler und Betriebssupportmitarbeiter verringert.

Ich lernte diese Technologie auf der VBits im Herbst '99 kennen und war beeindruckt. Nachdem die Visual Basic-Anwendung mit dem Visual Studio Installer installiert worden war, wurden einige der MDAC-Dateien gelöscht, und die Anwendung wurde neu gestartet. Das System (in diesem Fall Windows 98) begann, die gelöschten Dateien neu zu installieren und den Fehler zu beheben. Cool.

Sie können den Visual Studio Installer aus dem Web herunterladen (**http://msdn.Microsoft.com/vstudio**) oder als Bestandteil des Visual Studio Plus Pack bzw. des Windows 2000 Readiness Kit (im Visual Studio Plus Pack enthalten) erwerben.

1.6 Weitere Hilfe zu ADO

Okay, vielleicht benötigen Sie weitere Informationen zu ADO und wissen nicht, wohin Sie sich wenden sollen. Ich würde Ihnen die Website von MSDN Online oder die MSDN-CDs empfehlen, die alle paar Monate erscheinen. Das Problem ist, dass die Informationen (sehr wahrscheinlich) zwar irgendwo stehen, sie aber auch zu finden, ist so eine Sache.

Zwei gute Referenzen zu ADO sind meine zwei Bücher **Hitchhiker's Guide to Visual Basic and SQL Server** (6th Edition, Microsoft Press, ISBN: 1-57231-848-1) und **ADO 2.0 Programmer's Reference** (Wrox Press, ISBN: 0-861001-83-5). In letzterem werden verschiedene Schlussfolgerungen über Cursor gezogen, die bestenfalls bizarr erscheinen mögen. Microsoft Press hat ebenfalls ein gutes, an ADO 2.5 orientiertes Buch von David Sceppa herausgebracht – **Programming ADO** (ISBN: 0-7356-0764-8), das ich Korrektur gelesen habe (und für das ich das Vorwort verfasst habe). Das Buch **Teach Yourself Microsoft SQL Server 7.0 in 21 Days** (Sams, ISBN: 0-672-31290-5) von Richard Waymire und Rick Sawtell stellt eine solide Referenz für SQL Server 7.0 dar. **Active Server Pages** (Wrox Press, ISBN:1-861000-72-3) ist eine Hilfe bei den webseitigen Entwicklungsthemen.

Es steht Ihnen auch eine Onlinehilfe zur Verfügung – sogar kostenlos. Mehrere Tausend Leute und ich treiben sich häufig auf einem Listenserver herum: **VBDATA@peach.ease.lsoft.com**. Hier werden haufenweise Fragen gestellt und beantwortet.

2 Einführung in ADO

Ich habe lange überlegt, ob ich dieses Kapitel schreiben soll, da ich Sie nicht mit Dingen langweilen möchte, die Sie längst kennen. Der Trainer in mir riet jedoch, vor dem Eintauchen in die verschiedenen Programmiertechniken eine solide Grundlage zu schaffen. Daher enthält dieses Kapitel einige theoretische Abschnitte, die ADO beschreiben und an geeigneter Stelle in die übergeordneten Konzepte eingliedern. Wenn Sie direkt zu den technischen Details übergehen wollen, nur zu. Der coole Teil beginnt im nächsten Kapitel. Wenn Sie aber noch ein paar Dinge zu ADO wissen möchten, lesen Sie das vorliegende Kapitel.

2.1 ADO und OLE DB

Wenn wir zum Datenzugriff über ADO kommen, möchten Sie wahrscheinlich ein bisschen mehr über die Dinge wissen, die sich im Hintergrund abspielen. Nun, wie Sie wissen (oder wissen sollten), ist ADO eine COM-Schnittstelle (Component Object Model) zu den OLE DB-Providern, und OLE DB erwartet einen Konsumentenzugriff wie über ADO. Daher ist ein **OLE DB-Konsument** formal gesehen ein Systembestandteil oder ein Anwendungscode, der eine OLE DB-Schnittstelle anspricht. Dies schließt auch die OLE DB-Komponenten selbst ein. Abbildung 2.1 zeigt die Interaktion zwischen ADO und OLE DB.

Ein Provider ist eine Softwarekomponete, die eine OLE DB-Schnittstelle offen legt. **OLE DB-Provider** können allgemein in zwei Klassen unterteilt werden, in Datenprovider und in Dienstkomponenten.

Ein **Datenprovider** ist ein OLE DB-Provider, der über Daten verfügt und diese in Tabellenform als Rowset (Zeilensatz) offen legt (wird später in diesem Kapitel definiert). Beispiele für Datenprovider sind relationale DBMSs (Data Base Management Systems), Speicher-Manager, Tabellenkalkulationsprogramme, ISAMs (Indexed Sequential Access Method) und E-Mail-Datenquellen.

Eine **Dienstkomponente** ist eine OLE DB-Komponente, die keine eigenen Daten besitzt, aber einige Dienste kapselt, indem Daten über OLE DB-Schnittstellen erzeugt und verwendet werden. Eine Dienstkomponente ist sowohl ein Konsument als auch ein Provider. Ein heterogener Abfrageprozessor ist beispielsweise eine Dienstkomponente – es müssen Daten von einer Quelle gezogen, neu strukturiert und an die anfordernde Komponente, den Konsumenten, weitergegeben werden. Angenommen, ein Konsument möchte die Tabellendaten zweier verschiedener Datenquellen zusammenführen. In seiner Rolle als Konsument ruft der Abfrageprozessor Zeilen von Rowsets ab, die in den jeweiligen Basistabellen erstellt wurden. In seiner Rolle als Provider erstellt der Abfrageprozessor ein Rowset aus diesen Zeilen und gibt diese an den Konsumenten zurück.

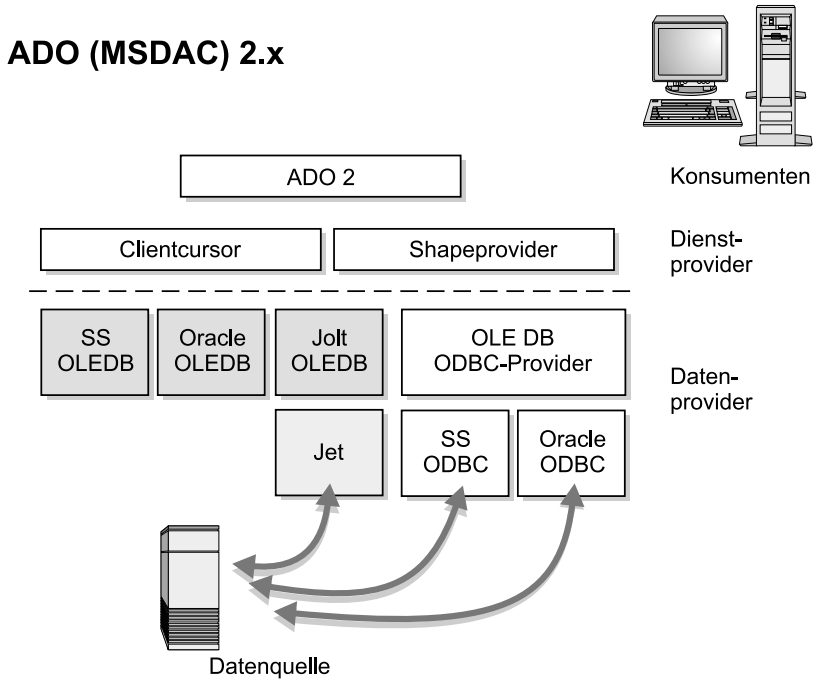


Abbildung 2.1 Verbindung zwischen ADO und OLE DB

Ein Datenbankverwaltungssystem (DBMS) ist eine Art Datenquelle, deren Aufgabe darin liegt, Informationen in anderer Form als ein OLE DB-Datenprovider zurückzugeben. In einigen Implementierungen ist ein DBMS in funktionale Bestandteile (Komponenten) gegliedert, die jeweils eine bestimmte Aufgabe übernehmen. Theoretisch besitzen DBMSs mit solchen Komponenten eine höhere Effizienz als traditionelle DBMSs, da der Konsument in der Regel nur einen Teil der Datenbankverwaltungsfunktionalität beansprucht und somit der Ressourcenoverhead geringer ist. OLE DB ermöglicht einfachen Tabellendatenprovidern das Implementieren einer dem Datenspeicher eigene Funktionalität: Auf unterster Ebene können nur die Schnittstellen verwendet werden, die Daten als Tabellen offen legen. Dies eröffnet neue Möglichkeiten bei der Entwicklung von Abfrageprozessorkomponenten, beispielsweise von SQL- oder geografischen Abfrageprozessoren. Diese können Tabelleninformationen beliebiger Provider verarbeiten, die ihre Daten über OLE DB offen legen. Zusätzlich können SQL-DBMSs ihre Funktionalität durch das Verwenden der OLE DB-Schnittstellen in einer strukturierteren Form bereitstellen.

Okay, soviel zur Theorie. Tatsache ist: ADO ist eine COM-basierte Programmierschnittstelle – wie DAO oder RDO. Anstelle von ODBC- (Open Database Connectivity) oder Jet-Treibern, die eine Verbindung zur Datenquelle herstellen, verwen-

det ADO OLE DB. Es zeigt sich, dass einer dieser OLE DB-Provider ODBC »spricht«, daher können Sie eine Verbindung zu einer beliebigen ODBC-Datenquelle herstellen (vorausgesetzt, Sie besitzen den richtigen ODBC-Treiber). Diese Treiber werden in der Terminologie von OLE DB »Datenprovider« genannt. Es gibt »systemeigene« OLE DB-Treiber für Jet-Datenbanken (die so genannten Jolt-Provider), mit denen sowohl auf Jet 3.x- (Access 9x) als auch auf Jet 4.0-Datenbanken (Office 2000) zugegriffen werden kann. Es gibt auch systemeigene Provider für SQL Server, Oracle und weitere, etwas obskure Datenquellen. In Abbildung 2.2 können Sie anhand des OLE DB-Dialogfeldes den zu verwendenden Provider auswählen. Dieses Dialogfeld erscheint, wenn Sie das Datenansichtfenster von Visual Basic oder ADODC (ADO Data Control) verwenden oder eine universelle ADO-Datenverbindung erstellen.

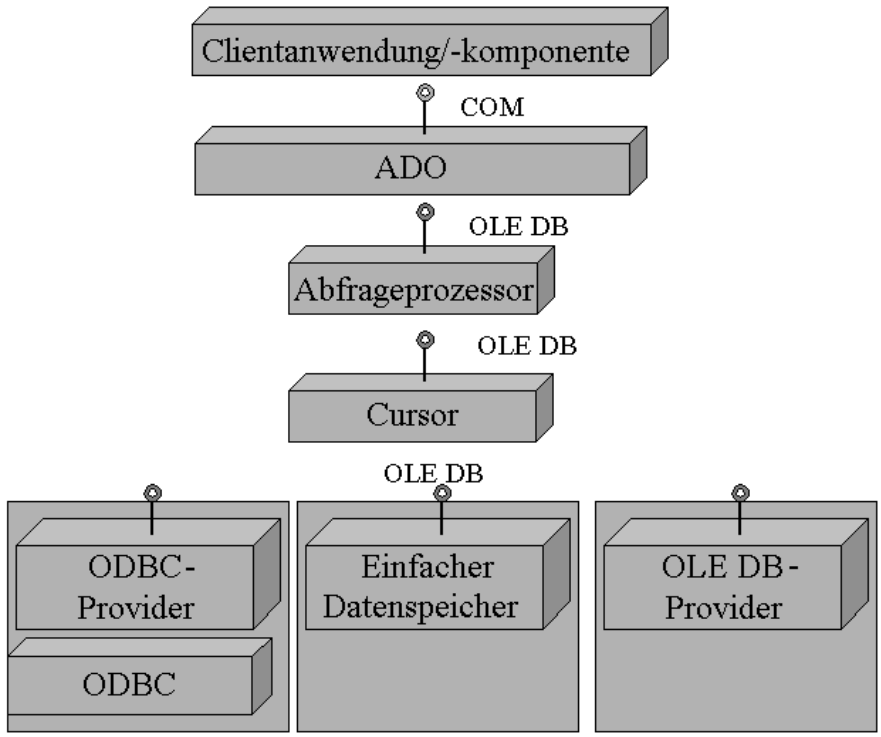


Abbildung 2.2 OLE DB-Provider, wie sie im Dialogfeld »Datenlinkeigenschaften« offen gelegt werden

ADO verwendet standardmäßig den OLE DB-Provider für ODBC – dies ist möglicherweise nicht die beste Wahl für Ihre Anwendung. Wenn Sie ein ADO-Programm codieren, gehen Sie bei der Auswahl des Providers unterschiedlich vor –

entweder übernehmen Sie die Standardeinstellung, oder Sie geben in den Eigenschaften des **Connection**-Objekts einen spezifischen Provider an. Nach der Verbindungsherstellung können Sie mit Hilfe anderer OLE DB-Provider Resultsets (Ergebnismengen), Cursor, Shapes (Formen, hierarchische Darstellungen von Daten und deren Beziehungen zu anderen Datensätzen) oder andere Datenstrukturen erstellen. Sie können mit Hilfe von ADO auch Daten von einer Schicht zu einer anderen verschieben, indem Sie Daten in das neue **Stream**-Objekt oder in einer Datei speichern. Im weiteren Verlauf des Buches wird all dies klarer werden ... oder zumindest hoffe ich das.

2.2 Das ADO-Objektmodell

Abbildung 2.3 zeigt das ADO 2.5-Objektmodell. Dieses Modell erscheint sehr viel einfacher als das DAO- oder das RDO-Modell – zumindest auf den ersten Blick. In den älteren DAO- und RDO-Technologien referenzierten Entwickler objektbasierte Eigenschaften und Methoden, um so die zugrunde liegenden relationalen Datenquellen zu bearbeiten. Bei ADO sind weit weniger objektbasierte Eigenschaften vorhanden, sehr viel weniger Objekte und nur vier Auflistungen. Sowohl das DAO- als auch das RDO-Modell erstickten förmlich an Auflistungen, die Speicher – und noch wichtiger – CPU-Ressourcen verbrauchten. Sie bildeten beim Erstellen und Zerstören von Objekten eine ständige Quelle von Speicherlecks.

Das ADO 2.5-Objektmodell weist nur fünf Basisobjekte auf – das **Connection**-, **Command**-, das **Recordset**-, das **Record**- und das **Stream**-Objekt. Diese können unabhängig voneinander erstellt werden. Dies bedeutet, dass Sie kein **Connection**- oder **Command**-Objekt erstellen müssen, um ein **Recordset**-Objekt erstellen zu können, und tatsächlich ist dieses Vorgehen häufig sogar sinnvoller. Beachten Sie, dass die **Connection**-, **Command**- und **Recordset**-Objekte durch die mit jedem Objekt verknüpften **Properties**-Auflistungen erweitert werden. Diese zusätzlichen Eigenschaften werden (in den meisten Fällen) durch den Provider generiert.

Wenn Sie eine Verbindung zu SQL Server herstellen, wird die **Properties**-Auflistung des **Connection**-Objekts mit Eigenschaften versehen, die speziell durch ausgewählte »systemeigene« oder ODBC-SQL Server-Provider offen gelegt werden. Diese offen gelegten Funktionen und Optionen sind SQL Server-spezifisch. Obwohl auch andere Provider Eigenschaften desselben Namens offen legen können, funktionieren diese vielleicht anders – wenn sie überhaupt implementiert sind. Ich rate meinen ADO-Programmierstudenten immer, einen Speicherauszug (ein Dump) der Eigenschaftenauflistung (**Properties**-Auflistung) des **Connection**-Objekts zu erstellen, bevor und nachdem eine Verbindung zum Datenprovider hergestellt wird bzw. wurde. Auf diese Weise wird ersichtlich, wie stark die Auswahl

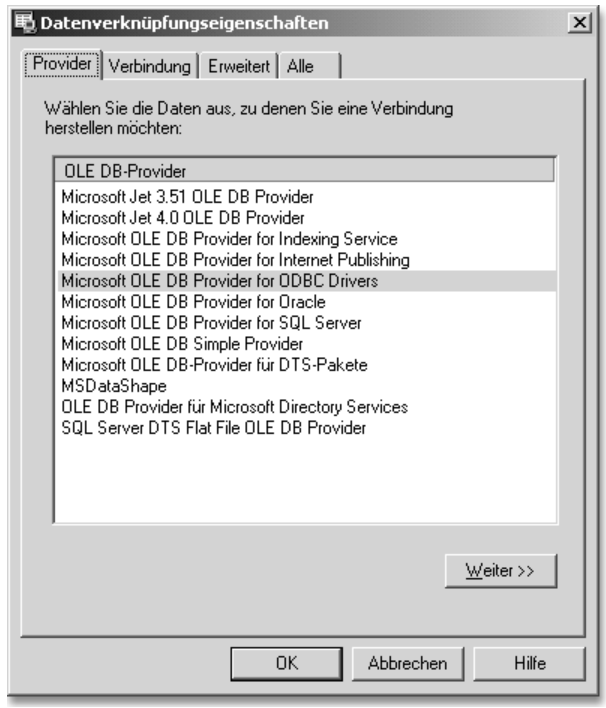


Abbildung 2.3 Das ADO 2.5-Objektmodell

des Providers sich auf die Eigenschaftenauflistung auswirkt und welche Dinge nicht von dieser Auswahl betroffen sind.

Tipp In Visual Basic werden Objekte erst instanziiert, wenn sie das erste Mal referenziert werden. Wenn Sie das ADO-**Connection**-Objekt im Code ansprechen (beispielsweise durch einen Dump der **Connection.Properties**-Auflistung), werden bestimmte Eigenschaften auf ihre Standardwerte gesetzt. Um sicherzustellen, dass eine Anwendung wie erwartet funktioniert, sollten Sie zunächst **Connection.ConnectionString** setzen, damit ADO weiß, welcher Datenprovider verwendet wird.

In der Tat werden diese Eigenschaften dokumentiert – fragt sich nur, **wo**. Nach einigem Suchen fand ich heraus, dass OLEDB.CHM im Suchpfad von MSDN nicht enthalten ist. Dies lässt darauf schließen, dass die Merkmale dieser Eigenschaften genau dort beschrieben werden. Ich würde einen Link auf diese Hilfedatei einfügen. Auf meinem System wurde die Datei zusammen mit anderen MSDN-CHM-Dateien geladen, aber es gibt keine Informationen darüber, wo im System die Datei gespeichert wird – suchen Sie einfach danach. Wenn Sie die Datei nicht finden

können, sollten Sie ein paar Seiten zurückblättern und die aktuellste ADO-Version unter **www.microsoft.com/data** herunterladen. In dieser Version ist die Hilfedatei enthalten.

Bevor Sie ein ADO-**Connection**-Objekt öffnen können, jedoch noch vor dem Einstellen der **ConnectionString**-Eigenschaft, enthält die **Connection.Properties**-Auflistung die folgenden Eigenschaften. Beachten Sie, dass die Mehrzahl dieser Eigenschaften keiner langen Erläuterungen bedarf, wenn Sie den OLE DB-Jargon einmal verstanden haben. Wenn Sie beispielsweise daran gewöhnt sind, eine **Default Database**-Eigenschaft in DAO oder RDO zu sehen, lautet diese in OLE DB **Initial Catalog**.

```
Password    =
Persist Security Info      =
User ID     =
Data Source   =
Window Handle          =
Location    =
Mode      =
Prompt     =  4
Connect Timeout        =    15
Extended Properties      =  DSN=LocalServer
Locale Identifier         =    1033
Initial Catalog =
OLE DB Services         =   -5
```

Nach dem Öffnen der Verbindung werden der Liste eine Menge weiterer Eigenschaften hinzugefügt (etwa drei Seiten, daher sollen diese nicht hier aufgeführt werden). Diese Eigenschaften sind etwas obskurer, aber wie gesagt, sie werden im OLE DB-SDK (Service Development Kit) dokumentiert. Das Anlegen eines Dumps der **Properties**-Auflistung werde ich zu einem späteren Zeitpunkt erläutern.

Das genaue Durchsehen der Ergebniseigenschaften nach dem Öffnen des **Connection**-Objekts kann sehr hilfreich sein. Ich denke, man könnte ein ganzes Buch über diese Eigenschaften und deren beste Verwendung schreiben. Die beste Informationsquelle für die Eigenschaften, die weder in diesem noch in den weiteren Kapiteln besprochen werden, stellt das OLE DB-SDK dar.

Berücksichtigen Sie, dass die Entwickler der OLE DB-Provider eine Menge Freiheit bei der Implementierung dieser Eigenschaften besitzen. Damit will ich sagen, dass OLE DB-Entwickler beim Erstellen eines Providers für den Zugriff auf eine Datenquelle viele der Eigenschaften nicht auf bestimmte Weise implementieren **müssen**. Während die meisten der gut definierten Eigenschaften einheitlich imple-

mentiert werden, wird den Entwicklern bei der Implementierung von weniger eindeutigen Operationen ein bisschen Trägheit nachgesehen. Freie Hand haben diese Entwickler auch bei der Implementierung von Features, die speziell für den entwickelten Datenprovider gelten.

Nachdem das ADO-**Connection**-Objekt geöffnet wurde, können Sie einen Dump der **Properties**-Auflistung erstellen, um anzuzeigen, welche zusätzlichen Lampen, Knöpfe und Befehlszeilenoptionen zur Steuerung des Datenproviders zur Verfügung stehen.

```
Dim cn As ADODB.Connection
Dim pr As Property

Private Sub Form_Load()
Set cn = New ADODB.Connection
cn.ConnectionString = "DSN=LocalServer"
For Each pr In cn.Properties
   Debug.Print pr.Name, " = ", pr.Value
Next pr
cn.Open "DSN=LocalServer", "SA", ""
For Each pr In cn.Properties
   Debug.Print pr.Name, " = ", pr.Value
Next pr
End Sub
```

3 Erstellen von ADO-Objekten

Obwohl viele von Ihnen wissen (oder zu wissen glauben), wie ADO- oder andere COM-Objekte erstellt werden, sollten wir kurz einige Grundlagen durchgehen. Ich kann dieses Thema nicht einfach übergehen, da ich zu viele seltsame Beispiele in Veröffentlichungen gesehen habe, deren technische Editoren es eigentlich besser wissen sollten. Ich spreche von der Einstellung der Verweise oder dem Schreiben von Code auf ASPs zu diesem Zweck und über einige Fehler, die viele Entwickler immer noch machen.

3.1 Instanziieren von ADO-Objekten

Okay, ich möchte zum technischen Teil des Programmierung kommen; bevor Sie die ADO-Objekte jedoch verwenden können, müssen die Objekte in Visual Basic instanziiert werden – aber das wussten Sie ja bereits. Wenn Sie in der Visual Basic-IDE (Integrated Development Environment) arbeiten, ist dieser Prozess kinderleicht. Alles, was Sie tun müssen, ist auf die ADO-Bibliothek zu verweisen – und dies ist vielleicht vor dem Starten einer Datenprojektvorlage oder vor dem Referenzieren der ADO-Datensteuerungskomponente bereits für Sie geschehen.

Wenn Sie in der Visual Basic-IDE arbeiten, können Sie eine geeignete ADO-Bibliothek über das Menü **Projekt/Verweise** auswählen. Verweisen Sie einfach auf die **Microsoft ActiveX Data Objects 2.x Library** (hierbei steht »x« für die verwendete ADO 2-Version) des in Abbildung 3.1 gezeigten Dialogfeldes. Nachdem auf die Bibliothek verwiesen wurde, untersucht Visual Basic die Typbibliotheken und füllt die Anwendungssymboltabelle. Dies bedeutet, dass Visual Basic bei der Codierung die Objekte, Eigenschaften, Methoden und (insbesondere) die aufgelisteten Argumente einfügt. Diese Funktion von Visual Basic führt zu einer erheblichen Leistungssteigerung – der Leistung bei der Codeerstellung. Mit dieser Funktion kann sichergestellt werden, dass Sie die Objektnamen richtig eingegeben haben und dass die Objekte zur Laufzeit vorhanden sind.

Tipp Wie sieht es mit der **Microsoft ActiveX Data Objects Recordset 2.1 Library** aus? Sollte dies nicht zu einer kleineren, schlankeren, leicht eingeschränkten Version von ADO 2.1 führen? Nun, es gab eine Zeit, da tat sie dies – diese Zeiten sind jedoch vorbei. Jetzt handelt es sich lediglich um eine Typbibliothek, die einen Teilsatz der ADO 2.1-Funktionalität offen legt, tatsächlich jedoch auf die gleiche MSADO15.DLL verweist, die von allen Versionen von ADO verwendet wird.

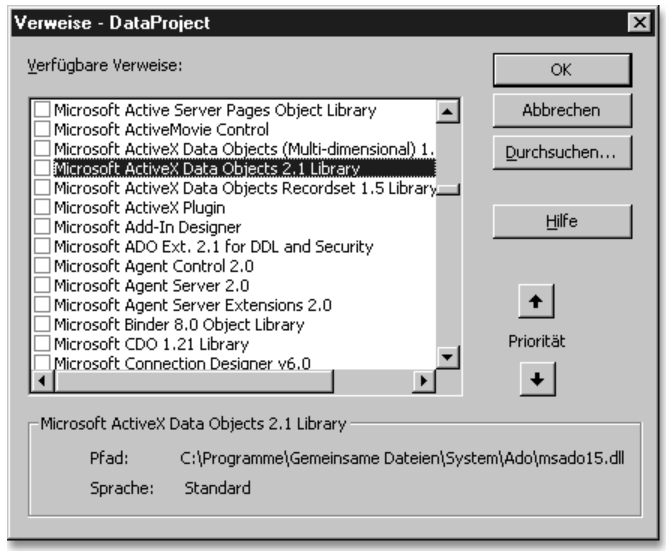


Abbildung 3.1 Das Dialogfeld »Projekt/Verweise« in der Visual Basic-IDE

Nun gut, wie also erstellen Sie diese Objekte im Code? In fast allen MSDN- und Dokumentationsbeispielen[1] wird die immer gleiche Syntax angegeben:

```
Dim cn as New ADODB.Connection          ' Dies ist die falsche Methode…
```

Wenn Sie diese Syntax verwenden, fügt Visual Basic jedem einzelnen Objektverweis einen messbaren Overhead hinzu. D. h., jedes Mal, wenn Visual Basic das Objekt **cn** anspricht, fügt der Compiler Folgendes hinzu:

```
If cn is Nothing then Set cn = New ADODB.Connection
```

Wie werden also ADO-Objekte (oder COM-Objekte) richtig deklariert und instanziiert? Unterteilen Sie die Operation einfach in zwei Schritte:

1. Deklarieren Sie die Variable nach Bedarf mit **Dim**, **Public** oder **Private**, aber verwenden Sie **nicht** den **New**-Operator.
2. Verwenden Sie eine separate **Set**-Anweisung, wenn (und nur dann, wenn!) Sie auf das neue Objekt im Code verweisen müssen. Verwenden Sie in diesem Fall den **New**-Operator, um eine neue Instanz des gewünschten Objekts zu erstellen.

1. Erwischt: Auch ich habe die Syntax *Dim xx as New yy* in früheren Versionen des Hitchhiker-Guide verwendet. Als ich jedoch bemerkte, wie sich diese Syntax auf die Leistung auswirkt, habe ich sie korrigiert. 'Tschuldigung.

Die richtige Syntax sieht folgendermaßen aus:

```
Dim cn as ADODB.Connection
...
Set cn = New ADODB.Connection
```

Wenn Sie dieser empfohlenen Codierungskonvention folgen, wird Ihre Anwendung ein wenig schneller ausgeführt, und darüber hinaus vermeiden Sie, dass von ADO bereitgestellte Features beeinträchtig werden, wenn Sie mit mehreren gespeicherten Prozeduren für Resultsets arbeiten. Hierzu später mehr.

Tipp Meine Studenten fragen mich oft, ob Sie den ADODB-Bezeichner verwenden müssen, wenn Sie **Dim**-Anweisungen programmieren. Ich sage ihnen dann, dass es ihre eigene Entscheidung ist. Gibt es jedoch auch nur eine winzige Chance, dass der Code zum Auslösen der DAO-Bibliothek (Data Access Object) führen könnte, sollte der Bezeichner zur Vermeidung von Namenskollisionen besser eingeschlossen werden. Wenn Sie das ADODB-Präfix verwenden, stellen Sie sicher, dass später von anderen hellen Köpfen vorgenommene Änderungen keine Schäden an Ihrer perfekten Codeversion hervorrufen.

3.2 Instanziieren von ADO-Komponenten in Visual Basic Script

Viele von Ihnen verwenden wahrscheinlich ADO über Visual Basic Script, dass unter IIS (Internet Information Service) oder mit dem lokalen Browser (weniger wahrscheinlich) ausgeführt wird. In diesem Fall verwenden Sie nicht die Visual Basic-IDE zum Erstellen Ihrer Anwendungen. Daher müssen Sie die ADO-Objektinstanziierung selbst programmieren:

```
Dim cn      ' Dies wird unser ADO-Connection-Objekt.
Set cn = Server.CreateObject("ADODB.Connection")
```

Wenn Sie zum Entwickeln von ASP- (Active Server Pages) oder HTM-Seiten (Hypertext Markup) Visual InterDev (VI) verwenden, versucht VI, Sie durch das Auffüllen mit Objekten zu unterstützen, die anhand der Anweisungsverarbeitung erkannt werden – aber diese Hilfe ist bei weitem nicht so umfassend wie die Unterstützung in der Visual Basic-IDE. Darum muss Ihrem Projekt eine Einschlussdatei hinzugefügt werden, mit der die Symboltabelleneinfügung für die Webentwicklung vervollständigt wird, wie im folgenden VBScript-Code gezeigt wird. Ohne **adovbs.inc** ist der ASP-Code nicht in der Lage, bestimmte Argumente und nicht aufgelistete Optionen zu referenzieren. Naja, so ganz stimmt das nicht. Ihr Code kann Dinge wie **adCmdText** als Option für die **Recordset Open**-Methode referenzieren, eine Kompilierung oder Ausführung ist jedoch nicht möglich.

```
<!--#include File="adovbs.inc"-->
<%
Dim rs, cn
set cn = server.CreateObject ("ADODB.Connection")
Set rs = Server.CreateObject("ADODB.Recordset")
Set stm = Server.CreateObject("ADODB.Stream")
  cn.ConnectionString =  "dsn=WorkServer;uid=TestASP;pwd=Secret"
  cn.CursorLocation  = aduseclient
  cn.Open
  if err then
    BuildErrorRecord 0, err, err.Description
  Else
```

Nachdem die ASP-ADO-Objekte erstellt sind, können diese in VBScript wie im »normalen« Visual Basic referenziert werden. Beachten Sie, dass Visual InterDev oder das Visual-Notepad (der Visual Basic-Skript-Editor) bei der Programmierung die Anfangsbuchstaben von Objekt- und Eigenschaftennamen nicht automatisch in Großbuchstaben umwandelt. Daher können Sie sich nicht (immer) sicher sein, dass der eingegebene Code auch wirklich richtig ist. Vielleicht in Visual Basic 7.0 …

ADO kann in einer beliebigen Schicht eingesetzt werden – sofern dies einen Sinn ergibt. Das ADO-**Command**-Objekt stellt z. B. nicht die beste Wahl für jede Anwendung dar.

Es gab hier bei uns heftige Diskussionen darüber, ob es effizient ist, bei der Arbeit mit MTS-Komponenten (Microsoft Transaction Server) oder ASP-Seiten ein **Command**-Objekt in der mittleren Schicht zu erstellen.

Sie sollten lediglich daran denken, dass ADO-Objekte (wie alle COM-Objekte) Zeit für die Instanziierung benötigen und zusätzliche Zeit, um wieder zerstört zu werden. Da diese Erstellungs- und Zerstörungsoperationen **jedes Mal** ausgeführt werden, wenn auf das Objekt verwiesen wird (nein, MTS unterstützt immer noch keine Poolkomponenten), sollten nach Möglichkeit keine **unnötigen** ADO-Objekte erstellt werden. Ja, Sie können die ODBC-API (Application Programming Interface) verwenden, um die Zeit zu reduzieren, die zum Starten und Zerstören der Komponenten benötigt wird. Wenn Sie daran denken, ADO mit Hilfe von OLE DB direkt zu umgehen, vergessen Sie es. Sie können OLE DB von Visual Basic aus nicht einsetzen – trotz der Leistungsvorteile gegenüber ADO. Wir werden dieses Thema in den Kapiteln zur Entwicklung in der mittleren Schicht noch ausführlicher behandeln.

4 Verbindungsherstellung

Wenn Sie mit ADO Daten abrufen möchten, müssen Sie entweder die Daten selbst erstellen oder eine Verbindung zu einer Datenquelle herstellen, um die Daten herunterzuladen. In einigen Fällen (wie wir in den verschiedenen Abschnitten zur Leistung und in Kapitel 8, »Übergeben von Resultsets zwischen Schichten« noch sehen werden) können Sie Daten nicht traditioneller Datenquellen abrufen, beispielsweise von ActiveX Server Pages (ASP) oder einem XML-Datenstream. Wenn Sie jedoch von einer traditionellen Datenbank Daten abrufen, beispielsweise von SQL Server, einer Access- (Jet) oder einer Oracle-Datenbank, müssen Sie eine Verbindung verwenden – zumindest am Anfang. Dies trifft auch auf die Unmenge von ODBC- und OLE DB-Datenquellen zu, die nach der Installation von MDAC 2.x (siehe Abbildung 4.1) verfügbar sind. Sie müssen jedoch **nicht unbedingt** ein eigenes ADO-**Connection**-Objekt erstellen – in vielen Fällen wird über ADO ein solches Objekt für Sie erstellt.

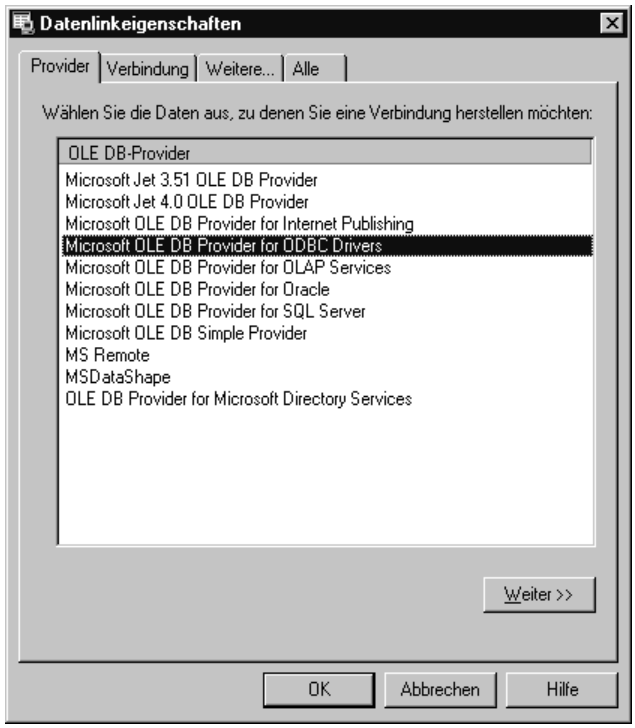


Abbildung 4.1 OLE DB-Provider

In diesem Kapitel wird erläutert, warum, wann und wie ADO-**Connection**-Objekte zur Verbindungsherstellung eingesetzt werden. Obwohl Ihnen diese Objekte im richtigen Leben nicht weiterhelfen, ermöglichen sie Ihnen doch einen Einblick in Ihre Daten. In Anbetracht der vielen Fragen, die in meinen Unterrichtsstunden zu diesem Thema gestellt werden, sowie der Zeit, die für dieses Thema aufgewendet werden muss, scheint die Verbindungsherstellung (mit einer Datenbank) für einige Leute ein etwas einschüchterndes Thema zu sein – und dies nicht nur für ADO-Anfänger. Beispielsweise scheint hinsichtlich **ConnectionString** und dessen ordnungsgemäßer (und effizienter) Verwendung ein wenig Verwirrung zu herrschen. Ich werde Ihnen im weiteren Verlauf dieses Kapitels ein paar Tipps und Tricks hierzu geben.

4.1 Das ADO-»Connection«-Objekt

Das ADO-**Connection**-Objekt ist (sprichwörtlich) Ihre Pipeline zu den Client/Server-Datenquellen. Vielleicht ist Ihnen nicht klar, dass ADO im Hintergrund häufig eines dieser Objekte erstellt, auch, wenn Sie dieses nicht ausdrücklich programmieren. Es gibt natürlich Ausnahmen (gibt es die nicht immer?), und diese werde ich in einer Minute besprechen, aber für den Moment soll es genügen, dass Sie an dem einen oder anderen Punkt üblicherweise ein **Connection**-Objekt erstellen müssen.

Ihr **Connection**-Objekt wird zum Erfassen von Eigenschaften eingesetzt, die an die zugrunde liegenden Daten- und Dienstprovider übergeben werden – beispielsweise ODBC, OLE DB oder Cursorprovider. Deshalb bietet die Dokumentation für die Provider (ODBC oder OLE DB) wichtige Grundlageninformationen zu den Vorgängen in diesen Parametern. Ich hoffe, Sie benötigen diese Informationen nicht.

Bei der Arbeit mit Webseiten wird häufig die Erstellung des **Connection**-Objekts übersprungen, statt dessen wird direkt zur **Recordset.Open**-Methode übergegangen. Das ist cool und geht schnell, erschwert aber die Fehlerbehandlung. Berücksichtigen Sie, dass die VBScript-Fehlerbehandlung fast genauso hoch entwickelt ist wie die in MBASIC 80 für CP/M[1]. Daher sollten Sie webbasierten Code mit etwas mehr Granularität ausstatten, um Fehler schon bei deren Entstehung aufzufangen. Weitere Informationen hierzu finden Sie in Kapitel 9.

1. CP/M ist das 8-Bit-Betriebssystem (8080/Z80), das ich unterstützt habe, als ich in den 80ern bei Digital Research gearbeitet habe. MBASIC-80 ist eine der frühen Versionen von Microsoft Basic.

Setzen Sie nie voraus, dass eine Verbindung geöffnet wird. Es gibt sehr viele Gründe dafür, warum eine Verbindung nicht aufgebaut wird (oder nicht aufrechterhalten werden kann). Lassen Sie uns einen kurzen Blick auf die Top Ten dieser Gründe werfen.

1. Die Datenquelle ist nicht vorhanden. Der Servername wurde falsch geschrieben oder der Server ist derzeit heruntergefahren. Dateiname oder Pfad sind falsch, oder das System, auf dem die Datei gespeichert ist, wurde heruntergefahren. Der Server wurde nachts von fanatischen Tierschützern gestohlen.
2. Das Netzwerk ist nicht in Betrieb. Hub, Router, Switch oder Domänencontroller sind nicht aktiv. Eines dieser kleinen Kästchen mit den coolen Blinklichtern wurde an jemanden ausgeliehen, der seine Weihnachtsdeko noch etwas aufpeppen wollte.
3. Das Netz ist da, der Server ist da, aber beide sind überlastet – Sie und noch irgendjemand haben alle verfügbaren Verbindungen belegt. Ihr Systemadministrator hat am Wert für die maximale Verbindungszahl herumgespielt – seine Sonderzulage ist wieder nicht genehmigt worden.
4. Sie haben nicht genügend Lizenzen, um den Server für die Verwendung durch weitere Benutzer aufzurüsten. Ihr Chef sagt, dass Sie erstens zu viel Geld für die Software ausgegeben haben und zweitens sowieso besser ein Freewaresystem eingesetzt hätten.
5. Die Verbindung wurde aufgebaut, aber Sie haben keine Berechtigung, auf die Datenbank zuzugreifen, die als anfänglicher Katalog oder Standarddatenbank angegeben ist. Sie haben gestern den Systemadministrator verärgert, weil Sie ihm vorgeworfen haben, am Wert für die maximale Verbindungszahl herumgespielt zu haben.
6. Der Server, zu dem Sie eine Verbindung herstellen möchten, versteht keine Named Pipes, nur Sockets. Sie haben vergessen, dass der Zielserver MSDE (Microsoft Data Engine) ist.
7. Der angegebene Name der Datenquelle in **ConnectionString** existiert nicht. Sie haben vergessen, diese auf jedem einzelnen System auf der Erde zu installieren, auf dem Ihre Anwendung ausgeführt wird.
8. Sie haben den falschen Provider für die Datenbank ausgewählt. Sie haben vergessen, dass der richtige Provider für eine Access 2000-Datenbank der Jolt 4.0-Provider ist. Sie haben nicht gewusst, dass der Systemadministrator alle Ihre Jet 3.5-Datenbanken in Jet 4.0-Datenbanken konvertiert hat – er ist immer noch sauer wegen der Geschichte mit der Verbindungszahl.

9. Sie haben für alle Ihre Anwendungen die Benutzer-ID des Systemadministrators verwendet – so wie es immer in allen Beispielen in allen Büchern steht – leider hat der **richtige** Systemadministrator das Kennwort geändert. Der Systemadministrator redet nicht mal mehr mit Ihnen – es gibt Anzeichen dafür, dass er nach Argentinien ausgewandert ist.
10. Die Benutzer haben STRG-ALT-ENTF gedrückt, bevor eine Zeitüberschreitung für Ihre Verbindung erfolgen konnte, da Sie die **ConnectionTimeout**-Eigenschaft auf 0 gesetzt haben. Sie hätten gerne noch länger gewartet, aber sie mussten zum Flughafen – irgendeine Reise in den Süden, um den Systemadministrator zu treffen oder so.

Ehrlich gesagt ist diese Liste nicht sehr vollständig. Die jeweilige Situation innerhalb der verschiedenen Unternehmen ist unterschiedlich und muss entsprechend gehandhabt werden. Obwohl die meisten dieser Fehler aufgefangen werden können, ist durch bessere Organisation, durch Teamwork und mehr Disziplin auf Seiten von Team, Management und – falls möglich – auch auf Seiten des Systemadministrators die Vermeidung einiger dieser Fehler im Vorfeld möglich. Dies mag ein bisschen viel verlangt erscheinen, aber wenn Sie eine stabile Anwendung erreichen möchten, müssen Sie etwas dafür tun, dass Ihre Anwendung funktionsfähig bleibt.

Ein **Connection**-Objekt repräsentiert eine eindeutige Sitzung mit einer Datenquelle. Im Falle eines Client/Server-Datenbanksystems kann es sich hierbei um eine tatsächliche Netzwerkverbindung zum Server handeln. Je nach vom Provider unterstützter Funktionalität stehen möglicherweise einige Auflistungen, Methoden oder Eigenschaften des **Connection**-Objekts nicht zur Verfügung. Wie können Sie feststellen, welche dieser Methoden, Eigenschaften usw. verfügbar sind und welche nicht? Prüfen Sie auch hierzu nach dem Öffnen des **Connection**-Objekts die **Property**-Auflistung.

Die Auflistungen, Methoden und Eigenschaften eines **Connection**-Objekts bieten Ihnen folgende Möglichkeiten:

- Konfigurieren der Verbindung vor deren Öffnung über die Eigenschaften **ConnectionString**, **ConnectionTimeout** und **Mode**. Nach dem Öffnen des **Connection**-Objekts können Sie die Eigenschaft **ConnectionString** untersuchen, um zu ermitteln, welche Optionen **tatsächlich** zum Verbindungsaufbau verwendet wurden.
- Setzen der **CursorLocation**-Eigenschaft, um den Clientcursorprovider auszulösen, der Stapelaktualisierungen unterstützt, oder um serverseitige Cursor mit Unterstützung schreibgeschützter Cursor zu verwenden.

- Festlegen der Standarddatenbank (Anfangskatalog) für die Datenbank über die **DefaultDatabase**-Eigenschaft.
- Festlegen eines OLE DB-Providers über die **Provider**-Eigenschaft. Vergessen Sie nicht, der Standardprovider ist der OLE DB-Provider für ODBC. Wenn Sie die **Property**-Auflistung einsehen, bevor die Providereigenschaft festgelegt wurde, ist diese auf **MSDASQL** eingestellt (der Name des OLE DB-Providers für ODBC).
- Verwalten von Transaktionen für die Verbindung, einschließlich verschachtelter Transaktionen, wenn diese vom Provider unterstützt werden. Hierzu werden die Methoden **IsolationLevel**, **BeginTrans**, **CommitTrans**, **RollbackTrans** sowie die Attributeigenschaften eingesetzt.
- Einrichten und anschließendes Trennen der physischen Verbindung zur Datenquelle mit den Methoden **Open** und **Close**. Beachten Sie, dass das **Recordset**-Objekt das **Connection**-Objekt auch eigenständig öffnen kann.
- Ausführen eines Befehls für das **Connection**-Objekt über die **Execute**-Methode.
- Ausführen eines benannten **Command**-Objekts für das **Connection**-Objekt – einschließlich der Parameterverwaltung sowie der Rückgabe eines Recordsets.
- Abrufen von Schemainformationen zur Datenbank über die **OpenSchema**-Methode.
- Prüfen der von der Datenquelle zurückgegebenen und in der **Errors**-Auflistung verzeichneten Fehler.
- Ermitteln der aktuellen ADO-Implementierungsversion mit der **Version**-Eigenschaft.

4.2 Verwenden des »Connection«-Objekts

Wie bereits gesagt, ist es nicht immer nötig, ein ADO-**Connection**-Objekt explizit zu deklarieren und zu öffnen – gelegentlich wird dieser Schritt von ADO übernommen. Die von Ihnen erstellten und mit der **Recordset Open**- oder der **Command Execute**-Methode verwendeten Verbindungszeichenfolgen werden allesamt unter Verwendung gleicher Elemente erstellt (also aufgepasst!).

4.2.1 Bereich des »Connection«-Objekts

Wenn Sie es nicht besser wüssten, würden Sie vielleicht denken, dass es cool sei, ein **Connection**-Objekt in einem Arbeitsschritt zu erstellen und es zur geeigneten Verwendung an einen weiteren Prozess zu übergeben. Dies stellt jedoch keine Option dar – und es war niemals eine. Es ist noch nie möglich gewesen, DB-Bibliotheks-, ODBC- oder OLE DB-Verbindungen zu erstellen und die Verbindungs-

handles oder -objekte an andere Prozesse zu übergeben – nicht einmal auf dem gleichen System. Es ist möglich, ein prozessübergreifendes Marshaling für **Recordset**-Objekte durchzuführen (selbst für **Command**-Objekte), für das **Connection**-Objekt ist dies jedoch nicht möglich.

Es gibt jedoch eine Alternative. Sie können mit Hilfe des MS REMOTE-Providers eine Remoteverbindung herstellen – es gelten jedoch einige Beschränkungen. Der MS REMOTE-Provider ermöglicht dem Client via HTTP, HTTPS oder DCOM das Erstellen eines Remoteproxys auf einem Remoteserver. Dieser Remoteproxyserver erstellt die tatsächliche Verbindung und führt die Abfragen für Sie aus. Dieser Vorgang ist vergleichbar damit, einen vertrauenswürdigen Agenten zu einer Auktion zu schicken. Der Agent berichtet Ihnen, welche Stücke zur Versteigerung angeboten werden, und Sie teilen dem Agenten Ihre Anweisungen mit. Funktioniert die Kommunikation, können Sie möglicherweise ein antikes Möbelstück erwerben – funktioniert die Kommunikation nicht, können Sie sich einen ausgestopften Elchkopf ins Wohnzimmer hängen. Aus Sicherheitsgründen sollte nach Möglichkeit die serverseitige Sicherheitsdatei MSDFMAP.INI verwendet werden.

Sie können eine Remoteverbindung beispielsweise auf die folgende Weise codieren:

```
Dim cn as adodb.connection
' Ich verwende hier nicht MSDFMAP.INI. Diese Vorgehensweise ist sehr unsicher,
folgen Sie daher nicht meinem Beispiel.
Set cn = New Connection
cn.open "Provider=MS REMOTE;Remote Server=http://MyServer;" _
   "Remote Provider=SQLOLEDB;Data Source=SS_Hoo; _
   "Initial Catalog=Pubs;User Id=sa;Password=;"
cn.execute "create table MyDBTable(id int)"
```

4.2.2 Setzen der »Provider«-Eigenschaft

Meistens ignoriere ich die **Provider**-Eigenschaft des **Connection**-Objekts – oder zumindest referenziere ich es nur dann, wenn ich eine Verbindungszeichenfolge debugge. Die **Provider**-Eigenschaft kann auf verschiedene Weise gesetzt werden, üblicherweise automatisch. Es stehen folgende Möglichkeiten zur Verfügung:

- **Keinesfalls** einen Provider benennen: Keine Referenzierung des Providers in der Verbindungszeichenfolge, kein Referenzieren der **Property**-Auflistung des **Connection**-Objekts im Code und (ganz besonders) kein Verweis auf die **Provider**-Eigenschaft selbst. In diesem Fall lautet die Standardeinstellung für die **Provider**-Eigenschaft **MSDASQL** – der OLE DB-Provider für ODBC.

- Erwähnen eines Providers in der Verbindungszeichenfolge. In diesem Fall wird die **Provider**-Eigenschaft auf die aktuelle **Version** des Providers eingestellt. Wenn die Verbindungszeichenfolge beispielsweise **Provider = SQLOLEDB** lautet, wird die **Provider**-Eigenschaft auf **SQLOLEDB.1** eingestellt.
- Festlegen der **Provider**-Eigenschaft direkt im Code

Das Ändern der **Provider**-Eigenschaft kann schwierig sein. Ich bin bereits in Situationen geraten, bei denen das Ändern dieser Einstellung nach der Providerfestlegung (durch Referenzierung der **Property**-Auflistung) durch Verwenden eines anderen **Provider**-Wertes zu Unstimmigkeiten bei ADO geführt hat.

4.2.3 Verbindungsherstellung zu systemeigenen OLE DB- und ODBC-Datenprovidern

Im Lieferumfang von MDAC 2.5 sind mindestens drei »traditionelle« relationale SQL-Datenspeicher enthalten – SQL Server, Oracle und Jet (.mdb)[2]. Bei den ersten MDAC-Versionen musste der OLE DB-Provider für ODBC-Daten eingesetzt werden, der den geeigneten ODBC-Treiber für den Zugriff auf diese Datenspeicher verwendete. Bei MDAC 2.0 gehen die Entwickler davon aus, dass die beste Lösung die Verwendung **systemeigener** OLE DB-Provider darstellt. Hierbei wird impliziert, dass der Zugriff auf die Daten schneller erfolgt und weniger Festplatten- und Arbeitsspeicher benötigt wird. Diese Denkweise geht (teilweise) darauf zurück, dass die systemeigenen Provider keine Übersetzung von ODBC auf die systemeigene Sprache der zugrunde liegenden Datenquelle erfordern. Der SQL Server-Provider ist beispielsweise für TDS geschrieben, der Oracle-Provider für OCI, der Jet-Provider ist auf die Microsoft Jet-Engine-APIs zugeschnitten.

Ich unterteile die Erläuterung in zwei Abschnitte. Zunächst soll die Verbindungsherstellung über den standardmäßigen OLE DB-Provider für ODBC, anschließend die Verbindungsherstellung über systemeigene OLE DB-Provider erläutert werden. Es gibt systemeigene Provider für SQL Server, Oracle und Jet 3.5 sowie Jet 4.0 (sowie für viele mehr).

Herstellen einer Verbindung mit einer ODBC-Verbindungszeichenfolge

Beginnen wir mit dem OLE DB-Provider für ODBC. Dieser verwendet eine Verbindungszeichenfolge, mit der Sie vielleicht schon vertraut sind. Eine der Herausforderungen, denen Sie sich als ADO-Programmierer gegenüber sehen (selbst als erfahrener Programmierer), ist, unabhängig vom ausgewählten Provider, das Erstel-

2. Die Liste der OLE DB-Provider meiner Visual Basic- und ADO 2.5-Version umfasst 12 Provider, einschließlich des OLE DB-Providers für ODBC. Es werden zwei Versionen des Jet-Providers bereitgestellt – einer für Jet 3.51, der andere für Jet 4.0.

len einer Verbindungszeichenfolge zum Übergeben der **ConnectionString**-Eigenschaft. Dies kann sehr einfach sein:

```
cn.ConnectionString = "DSN=MyServer"        'Setzen von ConnectionString
cn.Open , "Admin", "pw"
```

In diesem Fall übernimmt OLE DB die meisten Standardeinstellungen. Es wird der OLE DB-Provider für ODBC verwendet und der angegebene DSN (Data Source Name, Datenquellenname) geöffnet, indem die Windows-Registrierung angesprochen wird. Die Inhalte des DSN-Registrierungseintrags füllen die **Connection**-Eigenschaft für die erweiterten Eigenschaften. Die Ergebniszeichenfolge **ConnectionString** sieht (hinter den Kulissen) folgendermaßen aus:

```
Provider=MSDASQL.1;Password=pw;User ID=Admin;Connect Timeout=15;Extended
Properties="DSN=MyServer;Description=Local MSDE Server;
SERVER=(local);UID=Admin;PWD=pw;WSID=BETAV8;DATABASE=biblio";Locale
Identifier=1033
```

Nachdem die Verbindung geöffnet wurde, enthält **Connection.Properties** (erweiterte Eigenschaften) Folgendes:

```
DSN=MyServer;Description=Local MSDE
Server;SERVER=(local);UID=Admin;PWD=pw;WSID=BETAV8;DATABASE=biblio
```

Beachten Sie, wie über ADO die Benutzer-ID und das Kennwort in die Verbindungszeichenfolge eingefügt werden. Diese werden nicht über den DSN in der Registrierung gespeichert, sondern über die **Connection.Open**-Methode bereitgestellt. Sie können diese Werte in der Verbindungszeichenfolge bereitstellen und müssen sie in der **Open**-Methode nicht erneut angeben. Und nein, es gibt keine Möglichkeit, diese zu codieren, damit der Quellcode die Sicherheit des Kennworts gewährleistet.

Anmerkung Das in der **Open**-Methode verwendete Kennwort ist in den Zeichenfolgen klar erkennbar. Sie sollten bei der Übergabe dieser Objekte und Verbindungen daher vorsichtig sein.

Beachten Sie auch, dass die Verbindungszeichenfolge das Argument **Provider=MSDASQL.1** enthält. So wird OLE DB darüber informiert, dass Version 1 des OLE DB-Providers für ODBC verwendet werden soll. Das Entfernen der ».1« würde lediglich dazu führen, dass die aktuellste Providerversion dieses Namens verwendet wird. Zu diesem Zeitpunkt ist kein solcher Provider vorhanden – das heißt jedoch nicht, dass dies immer so bleibt. Beachten Sie, dass **WSID** (Name der Arbeitsstation) automatisch von ADO festgelegt wird. Sie können diesen Wert mit

einer eigenen Einstellung überschreiben. Ich verwende diese Eigenschaft in Client/Server-Anwendungen zur Identifizierung einzelner Clients. Wird Ihre Verbindung jedoch in der mittleren Schicht ausgeführt, ist das Ändern des **WSID**-Arguments sinnlos – Microsoft Transaction Server nimmt diese Einstellung vor, um ein Pooling der Verbindungszeichenfolgen zu ermöglichen.

Herstellen der Verbindung mit Hilfe von systemeigenen OLE DB-Providern

Wenn Sie jedoch nicht planen, über einen ODBC-DSN (Data Source Name) eine Verbindung zu SQL Server herzustellen (was Sie wahrscheinlich nicht möchten), müssen Sie eine komplexere Verbindungszeichenfolge erstellen – zumindest eine etwas kompliziertere. Berücksichtigen Sie, dass für jeden Provider von ADO Argumente an OLE DB übergeben werden müssen. Ja, viele dieser Argumente stimmen überein, aber es gibt einige entscheidende Unterschiede.

Denken Sie daran, wenn Sie die **Property**-Auflistung des **Connection**-Objekts im Code oder der IDE »anfassen«, dass hinter den Kulissen die **Connection**-Eigenschaft für den Providernamen eingestellt wird. Obwohl dies **nicht** aus der **Property**-Auflistung ersichtlich ist, wird dieser Wert auf ODBC (MSDASQL) gesetzt. Wenn Sie also zunächst **ConnectionString** festlegen, wird die Eigenschaft für den Providernamen auf den Angaben basieren, die unter **ConnectionString** gesetzt sind. Dies bedeutet, dass nach dem ersten Einsatz der **Property**-Auflistung – selbst bei Verwendung des Lokalfensters in der Visual Basic-IDE – eine Änderung des Providernamens nicht mehr möglich ist. Wenn Sie also einen Jet- oder einen systemeigenen Oracle-Provider öffnen möchten, erhalten Sie einen ODBC-Fehler.

Tipp Sie können die Zeichenfolge »(Local)« in der ODBC-Verbindungszeichenfolge verwenden, um auf den lokalen Server zu verweisen. Diese Methode funktioniert jedoch nicht für das Datenquellenargument in einer OLE DB-Verbindungszeichenfolge.

Dateibasierte Verbindungen

Beim Zugriff auf dateibasierte Datenquellen kann die Verbindungszeichenfolge unter Verwendung des Datenquellenarguments häufig einfach auf den Pfad und den Dateinamen der Datenbankdatei verweisen. Sie können jedoch mit Hilfe des Arguments für den Dateinamen auch auf eine dateibasierte UDL oder einen DSN verweisen:

```
cn.ConnectionString = "file name=c:\biblio.udl"
```

Bei der Arbeit mit ASP (Active Server Pages) möchten Sie vielleicht sicherstellen, dass eine vollständige Dateipfadangabe erfolgt, damit die .udl-Datei außerhalb der Website und damit in Reichweite des Browsers liegt (nicht gut). Platzieren Sie in diesem Fall die .udl-Datei außerhalb des Verzeichnisses **/inetpub/**. Das Verwenden von **Server.MapPath** führt nicht zum Erfolg, da ein Verzeichnis relativ zum virtuellen Verzeichnis zurückgegeben wird, und die gemeinsamen Dateien außerhalb des virtuellen Verzeichnisses gespeichert werden sollen.

Herstellen einer Verbindung zu Jet-Datenbanken (Access)

Sehen wir uns kurz die besonderen Probleme an, die bei der Verbindungsherstellung zum systemeigenen OLE DB-Provider für Jet-Datenbanken auftauchen. Sie versuchen, eine Jet-Datenbank zu öffnen, für die Benutzerkonten eingerichtet wurden. Sie wissen, dass Sie für den Provider ein Kennwort und eine Benutzer-ID angeben müssen, aber die Anwendung erhält die folgende Meldung: »-2147217843 Start der Anwendung nicht möglich. Die Datei mit Arbeitsgruppeninformationen fehlt oder wurde durch einen anderen Benutzer exklusiv geöffnet."[3] Sie sind verwirrt, sehen in der MSDN-Hilfe nach und finden folgende Fehlererklärung: »Zur Gewährleistung der referenziellen Integrität von durch die Microsoft Jet-Datenbankengine erstellten Datenbanken muss Ihre Anwendung die Microsoft Jet-Datenbankdatei **System.mdw** einlesen. Stellen Sie sicher, dass sich die Datei an dem im **SystemDB**-Wert angegebenen Standort des Windows-Registrierungsschlüssels **\HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Office\8.0\Access\Jet\3.5\Engines\Jet** befindet.«

Super. Microsoft erwartet, dass die Entwickler in der Registrierung herumsuchen, um sicherzustellen, dass die Datenbank mit einem Kennwort geöffnet werden kann. Und dann stellen Sie auch noch fest, dass der Registrierungseintrag in der Meldung nicht richtig ist. Sie haben Office 2000 installiert – die Fehlermeldung wurde nie aktualisiert. Seufz. Also suchen Sie in der Registrierung und finden den richtigen Eintrag (**HKEY_LOCAL_MACHINE\Software\Microsoft\Office\9.0\Access\Jet\4.0\Engines**), der jedoch auf **C:\PROGRA~1\MICROS~1\OFFICE\SYSTEM.MDW** eingestellt ist. Nochmals seufz – ein DOS-Kurzname.

Worüber sich ADO beschwert, ist die Jet-WIF-Datei (Workgroup Information File). Dies ist eine Datei, die die Jet-Engine liest, wenn die Datenbank zum ersten Mal geöffnet wird. Die WIF-Datei enthält Informationen zu den Benutzern einer Arbeitsgruppe, einschließlich der Benutzerkontennamen, Kennwörter und Gruppenmitgliedschaften. Sie müssen Jet mitteilen, wo sich diese Datei befindet, indem Sie vor dem Öffnen der Verbindung eine erweiterte Eigenschaft setzen.

3. Der Text der angezeigten Fehlermeldung kann je nach ADO-Version variieren. Dies gilt auch für alle folgenden Fehlermeldungen.

In Microsoft Access 95 und früheren Versionen werden Einstellungsinformationen zu den einzelnen Benutzern (angegeben im Dialogfeld **Optionen**) in der WIF-Datei gespeichert. In Microsoft Access 97 und späteren Versionen werden diese Informationen in der Windows-Registrierung gespeichert. In Microsoft Access, Version 2.0, lautet der Standardname der WIF-Datei **System.mda**. Gelegentlich wird diese Datei auch als Arbeitsgruppendatei oder Systemdatenbankdatei bezeichnet. In Microsoft Access 95 und späteren Versionen lautet der Standardname für diese Datei **System.mdw**.

Okay, all dies ist furchtbar interessant, aber wie öffnen Sie eine sichere .mdb-Datei? Nun, dank eines Entwicklers, der sich auch auf der Liste eines von mir abonnierten Dienstes befindet, habe ich herausgefunden, dass der Pfadname der Jet-Datei **System.mdw manuell** gesetzt werden muss. Sie können beispielsweise die Eigenschaft **Jet OLEDB:Systems Database** in der Verbindungszeichenfolge ansprechen:

```
"data source=C:\Databases\Secure40.mdb;" _
   & "Jet OLEDB:System Database=C:\Windows\System\System.MDW;"
```

Darüber hinaus müssen Sie die richtige Benutzer-ID und das Kennwort in den üblichen Eigenschaften für Benutzer-ID und Kennwort angeben, um eine Verbindung zu erhalten.

Tipp Sie müssen die Eigenschaft **Jet OLEDB:System Database** auf der Registerkarte **Alle** im Eigenschaftendialogfeld des Datenansichtfensters setzen, um eine Datenverknüpfung zu einer sicheren Jet-Datenbank zu erstellen. Wenn Sie dies tun, gibt die Oberfläche ein Dialogfeld aus, in dem Sie nach dem Administratorkennwort gefragt werden.

Bedenken Sie, dass das Öffnen eines Dialogfeldes nicht das ist, was bei der Verbindungsherstellung von MTS oder einer ASP-Seite aus passieren sollte. Erstens können Sie dieses Dialogfeld nicht sehen, und zweitens erscheint es, als sei die Site »eingefroren« – oder sie stürzt gleich ab.

4.2.4 Nachträgliches Öffnen von Verbindungszeichenfolgen

Nachdem das **Connection**-Objekt geöffnet wurde, kopiert ADO die providererstellte Verbindungszeichenfolge in die Eigenschaft **Connection.ConnectionString** zurück. Diese Zeichenfolge umfasst alle Argumente, die Sie in der ursprünglichen **ConnectionString** oder in der **Open**-Methode oder in beidem verwendet haben. Darüber hinaus sind auch alle Standardargumente enthalten. Es handelt sich hierbei um die Zeichenfolge, die zwischengespeichert und verglichen wird, wenn in der mittleren Schicht das Verbindungspooling eingesetzt wird.

Ist es für ADO nicht möglich zu entscheiden, wie mit einem Argument verfahren werden soll, kann es vorkommen, dass das entsprechende Argument in den Abschnitt der erweiterten Eigenschaften von **ConnectionString** eingefügt wird. Sie werden bemerken, dass die erstellte Zeichenfolge sowohl erweiterte Eigenschaften von OLE DB **als auch** von ODBC enthält und darüber hinaus noch »standardmäßige« Einstellungen, von denen nie die Rede war.

Nehmen Sie beispielsweise diese OLE DB-Verbindungszeichenfolge:

```
Provider=SQLOLEDB.1;Data Source=betav8
```

Obige Zeile führt zur Rückgabe der folgenden, nachträglich geöffneten Verbindungszeichenfolge:

```
Provider=SQLOLEDB.1;Password=pw;User ID=Admin;Data Source=betav8;Locale
Identifier=1033;Connect Timeout=20;Use Procedure for Prepare=1;Auto Trans-
late=True;Packet Size=4096;Workstation ID=BETAV8
```

Hieraus ergibt sich eine ODBC-Verbindungszeichenfolge ohne Abschnitt für die erweiterten Eigenschaften:

```
"Provider=MSDASQL;Driver={SQL Server};Server=Betav8;Database=pubs;uid=sa;pwd=;
```

Diese endet jedoch mit einem vollständig gefüllten Abschnitt für die erweiterten Eigenschaften, der die OLE DB-Äquivalente des ODBC-Arguments enthält.

```
Provider=MSDASQL.1;Password=pw;User ID=Admin;Connect Timeout=20;
Extended Properties="DRIVER=SQL Server;SERVER=Betav8 ;UID=Admin;PWD=pw;
WSID=BETAV8;DATABASE=pubs;Network=DBMSSOCN;
Address=betav8,1433";Locale Identifier=1033
```

Verwenden der Visual Database Tools zum Erstellen von Verbindungszeichenfolgen

Ehrlich gesagt, klaue (äh, leihe) ich zur Verbindungsherstellung zu anderen Datenquellen neben SQL Server mit Hilfe des OLE DB-Providers eine Verbindungszeichenfolge, die durch eines der Visual Database Tools oder ADODC (ADO Data Control) erstellt wird. Da die von ADO erzeugte Fehlermeldung eher dürftig bzw. einfach verwirrend ist, ist das Ermitteln der Verbindung, die den Fehler verursacht, häufig ziemlich frustrierend. Gehen wir den Prozess durch, der dazu führt, dass diese Verbindungzeichenfolgen von den Visual Database Tools für Sie erstellt werden.

Der Trick dabei ist, die IDE zum Erstellen der Verbindung zu verwenden und anschließend diese Verbindung zu stehlen. Hierzu stehen verschiedene Optionen

zur Verfügung. Ich tendiere dazu, für diese Übung ADODC zu verwenden, da es hierbei einfacher ist, die Verbindungszeichenfolge nach deren Erstellung zu extrahieren. Beginnen wir also mit dieser Methode.

1. Starten Sie mit Hilfe der Datenvorlage ein neues Visual Basic-Projekt.
2. Ziehen Sie ein ADODC-Steuerelement von der Toolbox auf **Form1** (oder ein beliebiges anderes Formular).
3. Klicken Sie mit der rechten Maustaste auf das ADODC-Objekt, aktivieren Sie auf der Registerkarte **Allgemein** die Einstellung **Verbindungszeichenfolge verwenden**, und klicken Sie auf **Erstellen**.
4. Wählen Sie im Dialogfeld **Datenlinkeigenschaften** aus der Providerliste den geeigneten OLE DB-Provider aus (siehe Abbildung 4.1).
5. Klicken Sie auf **Weiter**. Basierend auf dem gewählten Provider müssen verschiedene Angaben zum jeweiligen Provider gemacht werden.
6. Nachdem Sie die Eigenschaften angegeben haben (beispielsweise Datei- oder Servername, Benutzer-ID, Kennwort), klicken Sie auf **Verbindung testen** (Abbildung 4.2). Durch diesen Schritt wird sichergestellt, dass die erstellte Verbindungszeichenfolge auch tatsächlich (auf dem Entwicklungssystem) funktioniert. Kann die Verbindung nicht eingerichtet werden, ist ADO nicht in der Lage, die Standarddatenbankliste zu füllen. Wenn Sie eine Verbindung erstellen können, wird die Dropdownliste **Wählen Sie die Datenbank auf dem Server aus** mit Werten versehen.

Lassen Sie uns einen Moment innehalten, denn es müssen einige Dinge berücksichtigt werden. Erstens findet über das Dialogfeld **Datenlinkeigenschaften** (siehe Abbildung 4.2) tatsächlich eine **Verbindungsherstellung** statt, und hierzu ist häufig ein gültiger Benutzername sowie ein Kennwort erforderlich. Dies bedeutet, dass die Datenquelle (Netzwerk und Server) verfügbar sein muss, dass das Konto über die Berechtigungen für den Server oder die Datenquellendatei verfügen muss, die das Tool für eine Verbindungsherstellung benötigt. Zusätzlich müssen Sie eventuell die Netzwerkbibliothek zum Erreichen des Providers einstellen – wie bei der Verbindung zu MSDE (diese **erfordert** TCP/IP). Siehe hierzu auch den Abschnitt »Verwenden der SQL Server-Clientkonfiguration« weiter unten in diesem Kapitel.

Zweitens müssen Sie daran denken, dass die in die Visual Database Tools integrierten Fehlerbehandlungsroutinen ziemlich unausgereift sind. Es scheint beispielsweise nicht klar zu sein, wie mit überlasteten Verbindungen umgegangen werden soll – es werden zum Teil etwas sinnlose Fehlermeldungen zurückgegeben.

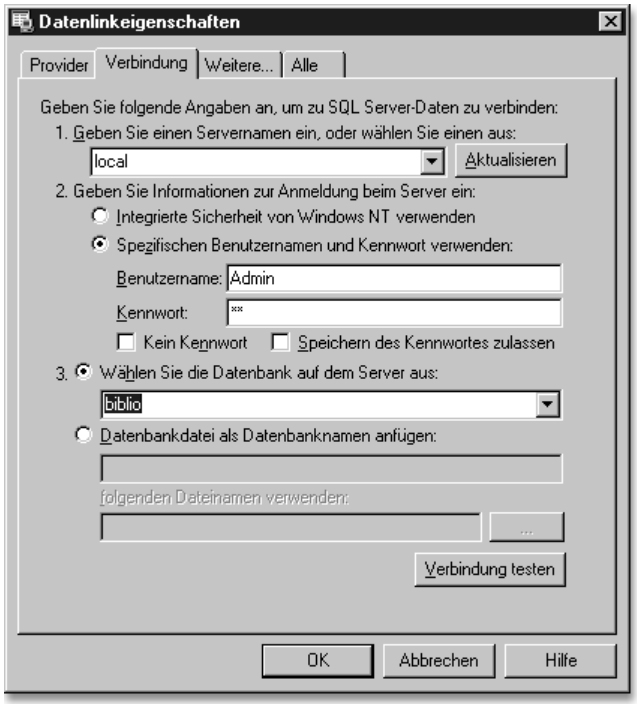


Abbildung 4.2 Verwenden des Dialogfeldes »Datenlinkeigenschaften« zum Erstellen einer Verbindungszeichenfolge

Da Visual Basic nicht alle Verbindungen wieder freigibt, die während der Experimentierphase verwendet werden, ist das Schließen und erneute Öffnen von Visual Basic einen Versuch wert. So werden Verbindungen freigegeben, die von Visual Basic selbst gehalten werden. Es reicht nicht aus, einfach die Anwendung zu stoppen.

Drittens werden der im Dialogfeld **Datenlinkeigenschaften** aufgeführte Benutzername und das Kennwort nicht in der Verbindungszeichenfolge gespeichert, es sei denn, Sie aktivieren das Kontrollkästchen **Speichern des Kennworts zulassen**. Andernfalls wird die Verbindungszeichenfolge dazu verwendet, die Verbindung einzurichten und die Liste der gültigen Datenbanken abzurufen, und wird anschließend verworfen.

Abschließend sollten Sie bedenken, dass eine erfolgreiche Verbindungsherstellung auf dem Entwicklungssystem mit eben diesen Eigenschaften noch keine Garantie dafür ist, dass auch bei der Bereitstellung Ihrer Anwendung, Komponente oder Webseite eine Verbindungsherstellung möglich ist.

Als Nächstes soll die Erstellung einer Verbindungszeichenfolge mit Hilfe der Visual Database Tools erläutert werden.

1. Klicken Sie auf **OK**, wenn Sie sich davon überzeugt haben, dass die Verbindung geöffnet wird. So kehren Sie zum gefüllten ADODC-Steuerelement zurück. Dieses Steuerelement sollte dem in Abbildung 4.3 ähneln.
2. Nun liegt eine funktionierende Verbindungszeichenfolge vor. Markieren Sie einfach die Zeichenfolge unter **Verbindungszeichenfolge verwenden**, und fügen Sie diese in Ihren Code ein.
3. Löschen Sie das ADODC-Objekt aus dem Formular.

Abbildung 4.3 Die ADODC-Eigenschaftenseite nach Erstellung der Verbindungszeichenfolge

Die soeben erstellte Verbindungszeichenfolge sieht folgendermaßen aus:

```
Provider=SQLOLEDB.1;Persist Security Info=False;User ID=Admin;Initial
Catalog=biblio;Data Source=(local)
```

Hierbei handelt es sich jedoch um eine einfache Verbindungszeichenfolge. Wenn Sie mit der gleichen Methode eine Verbindungszeichenfolge für eine Jet-Datenquelle erstellen, würde diese nach dem Extrahieren aus ADODC zunächst so aussehen:

```
Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\Program Files\Microsoft
Visual Studio\VB98\Nwind.mdb;Persist Security Info=False
```

Hm, sieht immer noch ziemlich einfach aus. Wann wird es also hässlich? Nun, wenn Sie nicht ADODC, sondern das Datenansichtfenster und den Datenumgebungs-Designer zum Erstellen der Verbindungszeichenfolge einsetzen, möchten Sie die **DataEnvironment1.Connection**-Objekteigenschaft der Verbindungsquelle (**ConnectionSource**) vielleicht so extrahieren, wie sie auf der Eigenschaftenseite der Visual Basic-IDE offen gelegt wird. Wenn Sie diese Zeichenfolge erfassen,

nachdem Sie auf die gleiche Jet-Datenbank verwiesen haben, erhalten Sie eine vollständig andere Zeichenfolge:

```
Provider=Microsoft.Jet.OLEDB.4.0;Password="";User ID=Admin;Data
Source=C:\Program Files\Microsoft Visual Studio\VB98\Nwind.mdb;Mode=Share
Deny None;Extended Properties="";Locale Identifier=1033;Jet OLEDB:System
database="";Jet OLEDB:Registry Path="";Jet OLEDB:Database Password="";Jet
OLEDB:Engine Type=4;Jet OLEDB:Database Locking Mode=0;Jet OLEDB:Global Par-
tial Bulk Ops=2;Jet OLEDB:Global Bulk Transactions=1;Jet OLEDB:New Database
Password="";Jet OLEDB:Create System Database=False;Jet OLEDB:Encrypt Data-
base=False;Jet OLEDB:Don't Copy Locale on Compact=False;Jet OLEDB:Compact
Without Replica Repair=False;Jet OLEDB:SFP=False
```

Beachten Sie die Vielzahl von doppelten Anführungszeichen. Diese werden zum Angeben leerer Argumente verwendet, z. B. **Password=""**. Wenn Sie diese Notierung in einer Visual Basic-Zeichenfolge übernehmen, erhalten Sie mit Sicherheit eine Menge Fehler. Der Grund hierfür ist, dass in Visual Basic zwei doppelte Anführungszeichen als einfaches doppeltes Anführungszeichen interpretiert werden. Alles klar? Egal, wenn Sie diese Verbindungszeichenfolge verwenden möchten, müssen Sie diese durchgehen und **alle** doppelten Anführungszeichen entfernen. Was, wenn richtige, in Anführungszeichen gesetzte Parameter übergeben werden? Nun, dies sollte kein Problem sein, da die Erstellungsroutine für die Verbindungszeichenfolge im Datenumgebungs-Designer nicht versucht, leere, in Anführungszeichen stehende Zeichenfolgen für Argumente mit Parametern zu übergeben. Ein Bug? Ich denke, ja.

Sie sehen also, warum ich die ADODC-Variante bevorzuge. Es ist leichter, die Verbindungszeichenfolge zu stehlen (ähm, zu leihen).

Datenquellennamen und Microsoft-Datenverknüpfungen

In früheren Tagen von ODBC wurde das Sammeln und Speichern von Informationen zu Verbindungszeichenfolgen über registrierte DSNs (Data Source Names) erledigt. Sie können in ADO-Verbindungszeichenfolgen weiterhin auf diese DSNs verweisen – ich habe dies in einem der vorangegangenen Beispiele getan. Das Problem bei diesem Ansatz besteht darin, dass Sie zwar ohne Weiteres in der Lage sind, einen DSN auf Ihrem Entwicklungssystem zu erstellen, das Installieren eines solchen auf dem Zielsystem jedoch ein ziemlicher PIA[4] sein kann. Es gibt verschiedene Alternativen zur Verwendung registrierter DSNs:

4. PIA (Pain in the Ass, technischer Fachausdruck).

- Verwenden Sie einen »dateibasierten« ODBC-DSN, der die gleichen DSN-Informationen enthält (im .ini-Format), jedoch nicht in der Registrierung, sondern in einer Datei gespeichert wird. Vorteile: leichte Bereitstellung und Installation – kopieren Sie einfach die Datei. Probleme: langsamer erster Zugriff, leichte Zerstörung oder Änderung durch Benutzer. Interne Benutzer-IDs werden neugierigen Blicken ausgesetzt.
- Verwenden Sie eine »DSN-lose« ODBC-Verbindung. Bei diesem Ansatz wird eine Hartcodierung aller erforderlichen Parameter in der Verbindungszeichenfolge vorgenommen. Sie geben den Server, Treiber und alle weiteren ODBC-Parameter an. Vorteile: leicht zu codieren (wenn man es einmal verstanden hat) und schnell – sehr schnell. Probleme: Sie haben den Servernamen in der Anwendung oder Komponente hartcodiert – der Servername ändert sich jedoch nicht sehr häufig (oder doch?). Im Grunde handelt es sich hierbei um den Ansatz, den wir bei der Übernahme einer ADODC-generierten Verbindungszeichenfolge in unser Projekt verwendet haben.

```
cn.ConnectionString = "Provider=SQLOLEDB.1;Persist Security Info=False;" _
   & "User ID=Admin;Initial Catalog=biblio;Data Source=(local)"
cn.Open , , "pw"
```

Tipp Teilen Sie Zeichenfolgen nicht mit Hilfe des Verknüpfungsoperators auf, wie ich dies im obigen Beispiel getan habe. Visual Basic stoppt so jedes Mal, um eine Zeichenfolge neu zu erstellen. Dieser Ansatz wird verwendet, um den Code zu Dokumentationszwecken etwas lesbarer zu machen, die Verarbeitung in Visual Basic erfolgt dadurch jedoch nicht unbedingt schneller.

- Verwenden Sie eine OLE DB-Microsoft-Datenverknüpfung. In diesem Fall können Sie eine Datei mit einer .udl-Erweiterung erstellen, die wie ein dateibasierter DSN an einem beliebigen Standort gespeichert werden kann. Die Datei enthält jedoch lediglich eine (Unicode-) Verbindungszeichenfolge (und nichts weiter). Diese werden in Windows 9x oder Windows NT 4.0 bzw. früheren Versionen mit Hilfe des Dateiexplorers erstellt (**Datei · Neu · Microsoft Datenlink**). Unter Windows 2000 erstellen Sie lediglich eine Textdatei und versehen diese mit einer .udl-Erweiterung. Durch das Doppelklicken auf diese neue Datei wird das so eben besprochene Datenlink-Dialogfeld geöffnet. Die Verbindungszeichenfolge wird in der .udl-Datei gespeichert, die .udl-Datei wird über einen Verweis auf Pfad und Name (Dateinamenargument in einer Verbindungszeichenfolge) geöffnet.

```
cn.ConnectionString = "File Name=c:\biblio.udl"
cn.Open
```

ADO hat sich mit Version 2.5 ein wenig verbessert, da nun alle Verweise auf DSNs und UDLs zwischengespeichert werden – auch, wenn Sie aus Dateien stammen. Darum wird auch für das Öffnen dateibasierter (und einiger registrierungsbasierter) Datenquellen beim **ersten** Mal etwas mehr Zeit benötigt.

Tipp Der Befehlsaufruf **Datei · Neu · Microsoft Datenlink** ist unter Windows 2000 nicht vorhanden. Sie können jedoch eine Datei erstellen und diese mit der .udl-Erweiterung versehen.

4.2.5 Codieren minimalistischer Verbindungszeichenfolgen

Ich sage häufig zu meinen Töchtern (George und Fred), dass weniger oft mehr ist. Für ADO-Verbindungszeichenfolgen gilt dies ebenso. Verbindungszeichenfolgen können sehr kurz sein.

```
Cn.Open "DSN=Fred"
```

Dieses Beispiel setzt voraus, dass Ihr Programm alle Standardwerte akzeptiert. In diesem Fall setzen Sie den ODBC-Provider voraus, und als DSN wird eine vertrauenswürdige Verbindung vorausgesetzt, daher benötigen Sie weder Benutzername noch Kennwort, und der Systemadministrator muss die richtige Standarddatenbank zugewiesen haben.

OLE DB-Zeichenfolgen können auch kurz sein, Sie müssen jedoch **erst** den Provider angeben, um Verwirrung bei der Zeichenfolgenanalyse zu vermeiden.

```
Cn.Open "Provider=SQLOLEDB;Data Source=MyServer;"
```

Wieder muss gewährleistet sein, dass die Standardeinstellungen akzeptiert werden. Nachfolgend einige kurze Beispielverbindungszeichenfolgen für verschiedene Datenquellen:

- Systemeigener OLE DB-Provider für SQL Server (sqloledb.dll)

  ```
  Provider=Sqloledb;Data Source=ServerXX;Initial Catalog=pubs;User
  Id=sa;Password=;
  ```

- Jet / Access 97, Version 3.51 (msjtor35.dll)

  ```
  Provider=Microsoft.Jet.OLEDB.3.51;Data Source=c:\northwind.mdb
  ```

- Jet / Access 97, Version 4.0 (msjetoledb40.dll)

  ```
  Provider=Microsoft.Jet.OLEDB.4.0;Data Source=c:\northwind.mdb
  ```

- Oracle (msdaora.dll)

  ```
  Provider=Msdaora;Data Source=Serverxx.world; User ID=scott;Password=ti-
  ger"
  ```

- Zur Verwendung einer alten ODBC-Treiberversion verwenden Sie folgende Syntax:

  ```
  "Provider=MSDASQL;Driver={SQL Server};Server=MyServer;Data-
  base=pubs;uid=sa;pwd=;"
  ```

- Für ISAM, beispielsweise Excel:

  ```
  "Provider=MSDASQL;Driver={Microsoft Excel Driver (*.xls)}; DBQ=D:\inet-
  pub\wwwroot\testXL\pubs.xls;"
  ```

- MSDataShape (msadds.dll)

  ```
  "Provider=MSDataShape;Data Provider=SQL Oledb;Data Source=ServerXX;
  Initial Catalog=pubs;User Id=sa;".
  ```

- MS Remote (msdarem.dll) – Sie können dreischichtige, parametrisierte Hierarchien verwenden, indem Sie MSDataShape und MS Remote miteinander kombinieren.

  ```
  "Provider=MSDataShape;Data Provider=MS Remote;Remote
  Provider=sqloledb;Remote Server=http://your_IIS_server;Data
  Source=ServerXX;Initial Catalog=pubs;User Id=sa;"
  ```

Erstellen einer neuen Jet-Datenbank

In einigen Fällen möchten Sie vielleicht eine Speicherung in einer Microsoft Jet-Datenbank vornehmen. Mit ADO kann dies umgesetzt werden, es muss jedoch auf eine andere Objektbibliothek verwiesen werden. Wählen Sie die Option **Microsoft ADO Ext. 2.1 (or 2.5) for DDL and Security**, und verwenden Sie die folgende Syntax:

```
Dim cat as ADOX.Catalog
Set Cat = New ADOX.Catalog
Cat.Create "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=c:\MyNewDB.mdb"
```

Konvertieren von ODBC- in OLE DB-Verbindungszeichenfolgen

Als die OLE DB-Experten mit der Definition der Argumente begannen, die durch den Verbindungszeichenfolgenparser erkannt werden sollen, sollte der Konvertierungsvorgang nicht **zu einfach** sein, daher wurden mehrere der Argumente neu definiert – in Tabelle 4.1 werden einige aufgelistet.

ODBC	OLE DB
UID=;PWD=;	User ID=, Password=
Database=	Initial Catalog=
Trusted_Connection=Yes	Integrated Security=SSPI;
Server	Data Source

Tabelle 4.1 Verbindungszeichenfolgenargumente für ODBC- und OLE DB-Provider

Verbindungszeichenfolgenargumente für den SQL Server-Provider (SQLOLEDB)

Obwohl in der Visual Studio-Dokumentation die anfänglichen Eigenschaften für den SQLOLEDB-Provider aufgeführt werden, sind einige der Eigenschaftenbeschreibungen falsch. Es ist wichtig, dass nun eine SQL Server 7.0-Verbindungszeichenfolge erstellt werden kann, die alle Parameter enthält, die beim Erstellen eines registrierten DSNs angegeben werden können. In früheren Versionen von ODBC war dies unmöglich – viele der Argumente waren für die registrierungsbasierten DSNs reserviert. In Tabelle 4.2 werden die SQL Server-Providereigenschaften und deren Zweck beschrieben.

Eigenschaft	Beschreibung
"Data Source"	Der SQL Server, zu dem eine Verbindung hergestellt werden soll.
"Initial Catalog"	Die standardmäßige SQL Server-Datenbank.
"Integrated Security"	Eine Zeichenfolge, die den Namen des Authentifizierungsdienstes enthält. Dieser kann auf »SSPI« (Secured Support Provider Interface) oder auf »« gesetzt werden, um die integrierte Windows NT-Sicherheit zu verwenden.
"Locale Identifier"	SQLOLEDB validiert die Ländereinstellungs-ID und gibt einen Fehler zurück, wenn diese nicht unterstützt wird oder nicht auf dem Clientcomputer installiert ist.
"Password"	Das Kennwort, das einer SQL Server-Anmeldung zugeordnet ist. Diese Eigenschaft wird verwendet, wenn der SQL Server-Authentifizierungsmodus für den autorisierten Zugriff auf eine SQL Server-Datenbank ausgewählt wurde.
"Persist Security Info"	SQLOLEDB speichert die Authentifizierungswerte, falls angefordert, einschließlich eines Kennwortimages. Es wird keine Verschlüsselung bereitgestellt.

Tabelle 4.2 Verbindungszeichenfolgenargumente für den SQL Server-Provider (SQLOLEDB)

Eigenschaft	Beschreibung
"Prompt"	SQLOLEDB unterstützt alle Eingabeaufforderungsmodi für die Datenquelleninitialisierung. SQLOLEDB verwendet den Wert 4 (keine Eingabeaufforderung) als Standardeinstellung für die Eigenschaft. Die vier möglichen Werte sind: (1): Benutzer immer zur Eingabe von Initialisierungsinformationen auffordern. (2): Benutzer nur auffordern, wenn weitere Informationen erforderlich sind. (3): Benutzer nur auffordern, wenn weitere Informationen erforderlich sind. Dem Benutzer nicht ermöglichen, optionale Informationen einzugeben. (4): Keine Eingabeaufforderung an den Benutzer. Zurückgeben eines auffangbaren Fehlers an die Anwendung, wenn Benutzername und Kennwort nicht validiert werden können.
"User ID"	Ein SQL Server-Anmeldekonto. Diese Eigenschaft wird verwendet, wenn der SQL Server-Authentifizierungsmodus für den autorisierten Zugriff auf eine SQL Server-Datenbank ausgewählt wurde.
"Window Handle"	Ein Fensterhandle von der aufrufenden Anwendung. Ein gültiges Fensterhandle ist für das Initialisierungs-Dialogfeld erforderlich, das angezeigt wird, wenn die Eingabeaufforderung nach Initialisierungseigenschaften erlaubt ist.
"Connect Timeout"	SQLOLEDB gibt einen Initialisierungsfehler zurück, wenn eine Verbindung zum SQL Server nicht innerhalb der angegebenen Zeit (in Sekunden) aufgebaut werden kann. Der minimale Wert liegt etwa bei zehn Sekunden.

Tabelle 4.2 Verbindungszeichenfolgenargumente für den SQL Server-Provider (SQLOLEDB)

Die in Tabelle 4.3 gezeigten Eigenschaften sind erweiterte Eigenschaften, die durch den systemeigenen OLE DB-SQL Server-Provider (SQLOLEDB) definiert werden. Diese werden in der Auflistung der Verbindungseigenschaften offen gelegt und sind les- und schreibbar.

Erweiterte Eigenschaft	Beschreibung
"Application Name"	Der Name der Clientanwendung.
"Auto Translate"	Falls diese Eigenschaft den Wert **True** aufweist, führt SQLOLEDB beim Abrufen oder Senden von Zeichenfolgen von oder an SQL Server eine OEM/ANSI-Zeichenkonvertierung durch. Bei Einstellung des Wertes **False** wird keine Konvertierung vorgenommen.
"Current Languge" (Beachten Sie die falsche Schreibweise.)	Ein SQL Server-Sprachenname. Gibt die Sprache an, die für die Systemmeldungsauswahl und die Formatierung verwendet wird. Die Sprache muss auf dem SQL-Server installiert sein, ansonsten schlägt die Initialisierung fehl.

Tabelle 4.3 Erweiterte ODBC-Eigenschaftsargumente für SQL Server 7.0

Erweiterte Eigenschaft	Beschreibung
"Network Address"	Die über die Eigenschaft angegebene Netzwerkadresse von SQL Server. Diese wird für IPX- und NetBIOS-Protokolle verwendet. Siehe auch Artikel Q175472 in der MSDN-Hilfe. Es kommt auch vor, dass der Servername, gefolgt von einer Anschlussadresse, für TCP/IP (BetaV9.1433) verwendet wird.
"Network Library"	Der Name der Netzwerkbibliothek (DLL), die zur Kommunikation mit SQL Server eingesetzt wird. Der Name sollte weder den Pfad noch die .dll-Dateierweiterung aufweisen. Die Standardeinstellung wird bei der SQL Server-Clientkonfiguration bereitgestellt.
"Packet Size"	Die Größe eines Netzwerkpakets in Byte. Der Wert für die Paketgröße muss zwischen 512 und 32.797 liegen. Der Standardwert lautet 4.096.
"Use Procedure for Prepare"	Definiert die Verwendung von temporär gespeicherten SQL Server-Prozeduren: 0: Bei der Vorbereitung eines Befehls wird keine temporär gespeicherte Prozedur erstellt. 1: Standardwert. Bei der Vorbereitung eines Befehls wird eine temporär gespeicherte Prozedur erstellt. Die temporär gespeicherten Prozeduren werden gelöscht, wenn die Sitzung wieder freigegeben wird. 2: Bei der Vorbereitung eines Befehls wird eine temporär gespeicherte Prozedur erstellt. Die Prozedur wird gelöscht, wenn der Befehl nicht vorbereitet oder wenn ein neuer Befehl für das Befehlsobjekt angegeben wird bzw., wenn alle Anwendungsverweise auf den Befehl freigegeben werden.
"Workstation ID"	Eine Zeichenfolge zur Identifizierung der Arbeitsstation.
"Initial File Name"	Diese Eigenschaft ist für eine zukünftige Verwendung mit SQL Server 7.0-Datenbankdateien vorgesehen.

Tabelle 4.3 Erweiterte ODBC-Eigenschaftsargumente für SQL Server 7.0

Verwenden der SQL Server-Clientkonfiguration

In Fällen, in denen der Server nicht weiß, wie eine Named Pipe geöffnet oder erstellt werden soll oder Sie einen Server im Web referenzieren müssen, rufen Sie die SQL Server-Clientkonfiguration auf (siehe Abbildung 4.4), um ODBC anzuweisen, einen anderen Netzwerktreiber für den Zugriff auf den speziellen Server zu verwenden. Dies ist eine Anforderung für den Zugriff auf MSDE-Versionen (Microsoft Database Engine) von SQL Server 7.0, die unter Windows 9x ausgeführt werden. Das Betriebssystem weiß in diesem Fall nicht, wie mit Named Pipes umzugehen ist (der Standardnetzwerkschnittstelle). Ehrlich gesagt ist das TCP/IP-Protokoll schneller als Named Pipes, leichter zu installieren und zuverlässiger. Sie können jedoch keine domänenverwaltete Sicherheit ohne Named Pipes einrichten.

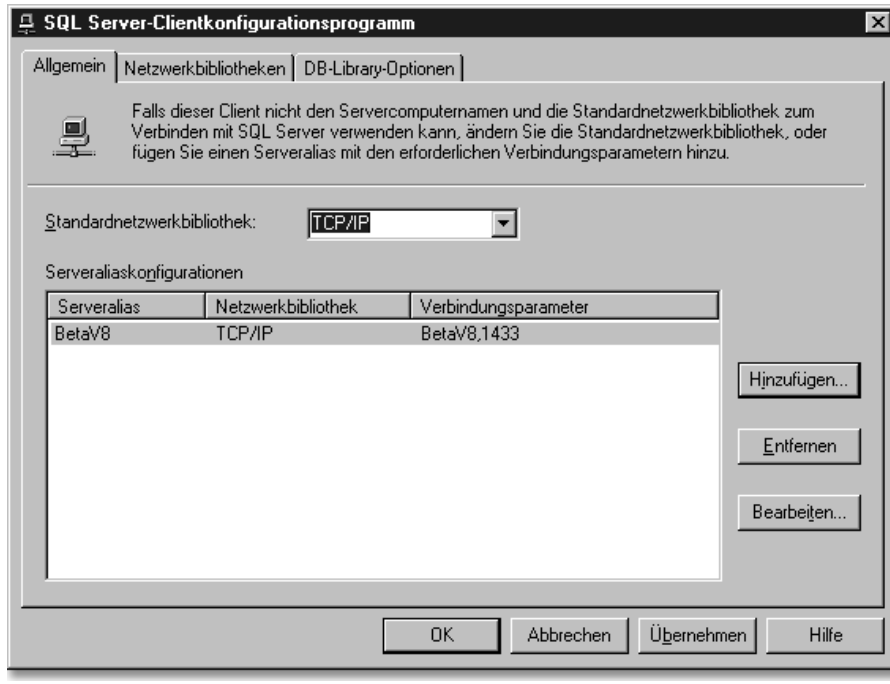


Abbildung 4.4 Verwenden der SQL Server 7.0-Clientkonfiguration zur Auswahl eines alternativen Protokolls

Führen Sie zum Starten dieses Dienstprogramms einfach **Start · Microsoft SQL Server · Clientkonfiguration** aus. Geben Sie den Namen des betreffenden Servers ein, und wählen Sie die geeignete Netzwerkschnittstelle aus der Optionenliste auf der linken Seite aus.

Es gibt eine weitere Möglichkeit, das Verwenden von TCP/IP (oder anderer Protokolle) anstelle des Standardwertes (Named Pipes) zu erzwingen: Fügen Sie Ihrer Verbindungszeichenfolge ein Netzwerkargument (network=) hinzu. Durch die folgende Verbindungszeichenfolge wird beispielsweise die Verwendung von TCP/IP erzwungen:

```
Network=dbmssocn;
```

4.2.6 Setzen der »CursorLocation«-Eigenschaft

Standardmäßig wird durch ADO **kein** Cursor zur Handhabung der Rowsets[5] erstellt, die das Ergebnis von Abfragen darstellen, daher spielt die **CursorLocation**-Eigenschaft keine besonders große Rolle.

5. Ein Rowset ist eine Menge von Zeilen, die die in der SELECT-Anweisung angegebenen Spalten enthält und die Mitgliedschaftsanforderungen der WHERE-Klausel einer Abfrage erfüllt.

ADO setzt die **CursorLocation**-Eigenschaft standardmäßig auf serverseitige Cursor (**adUseServer**). Dies bedeutet, dass bei Verwendung der Bibliotheksfunktionen clientseitiger Cursor zunächst ein Wechsel auf clientseitige Cursor (**adUseClient**) erfolgen muss. In den Kapiteln zum Recordset wird die Auswirkung dieser Auswahl näher erläutert. Es stellt sich heraus, dass Sie nach Bedarf zwischen den **CursorLocation**-Einstellungen wechseln können.

Tipp Wenn Sie Probleme mit schwerfälligen Abfragen haben, die an den Server gesendet wurden (belegt durch den SQL Server Profiler oder die Ablaufverfolgungsroutinen Ihres Datenproviders), versuchen Sie, die **CursorLocation**-Eigenschaft zu ändern. Die Datenprovider weisen, basierend auf dem Ort der Cursorerstellung, unterschiedliche Ansätze beim Datenzugriff auf, selbst in Fällen, in denen cursorlose Resultsets verwendet werden. Die Probleme mit der **Prepared**-Eigenschaft beispielsweise, die in Kapitel 5 besprochen werden, können gemildert werden, indem die **CursorLocation**-Eigenschaft auf **adUseClient** geändert wird.

Verwenden client- oder serverseitiger Cursor

Ich bin schon häufig gefragt worden, welche denn die »besseren« Cursor seien, die client- oder die serverseitigen Cursor. Ich sage üblicherweise: »Keine der beiden.« Wenn Sie Cursor verwenden **müssen**, muss berücksichtigt werden, wie (und wo) Ressourcen am besten eingesetzt werden. Serverseitige Cursor erzeugen das Rowset an einem temporären Speicherort auf dem Server, daher verbrauchen diese Cursor serverseitige CPU- und Festplattenressourcen. Sie müssen das Rowset noch auf den Client übertragen, daher besteht von diesem Gesichtspunkt aus kein großer Unterschied zur clientseitigen Implementierung. Clientseitige Cursor weisen ADO an, die Schlüsselgruppe oder das statische Rowset im Client-RAM oder im Festplattenspeicher zu erstellen. In der mittleren Schicht kann es sich bei diesen Ressourcen möglicherweise um die gleichen Ressourcen handeln, die vom lokalen Server verwendet werden.

Können signifikante Unterschiede in der Leistung ermittelt werden, wenn ein Cursortyp den anderen überlagert? Ich bezweifle dies. Es ist sehr viel Arbeit in die clientseitige Cursortechnologie gesteckt worden – und darauf ist das MDAC-Team sehr stolz. Ein Zeugnis hiervon ist der (relativ) neue Shapeprovider. Viele Microsoft-Entwickler sind davon überzeugt, dass die clientseitigen Cursor auf jeden Fall vorzuziehen sind, dass sie die meisten Funktionen und die höchste Leistung bieten. Ich lasse Sie selbst entscheiden. Sehen wir uns eine Reihe von Methoden an, die nur mit serverseitigen Cursorn implementiert werden können, und andere, die nur mit clientseitigen Cursorn funktionieren.

Anmerkung Ich habe in einer Reihe von Büchern einen Hinweis auf die Verwendung der CREATE CURSOR TSQL-Syntax gefunden. Hiermit sind keine »richtigen« serverseitigen Cursor gemeint. Der Grund hierfür ist, dass die TSQL-Cursor durch den ADO-Provider generiert werden – einschließlich serverseitiger Cursor, falls erforderlich. Obwohl Sie mit dieser Methode eigene serverseitige Cursor codieren und verwalten können, kann diese Technik viel Ärger hervorrufen.

4.2.7 Setzen der Eigenschaften für die Zeitüberschreitung

ADO legt sowohl **ConnectionTimeout**- als auch **CommandTimeout**-Eigenschaften offen. Dies ist toll, aber haben Sie jemals bemerkt, dass einige dieser Zeitüberschreitungseinstellungen nicht richtig zu funktionieren scheinen? Vielleicht deswegen, weil niemand sie richtig versteht.

- **ConnectionTimeout** startet, wenn eine LAN-Verbindung mit dem Datenbankserver aufgebaut wird und endet, wenn der Datenquellenprovider eine Verbindung erstellt. Dies bedeutet, dass die Netzwerkkomponente **nicht** Teil der Zeitmessung ist. Aufgrund dieser Tatsache erfolgt eine Zeitüberschreitung auch über die Einstellung für die Netzwerkzeitüberschreitung, wenn Ihr Ehepartner mal wieder mit dem Staubsauger das Netzwerkkabel gekappt hat. Leider gibt es allerdings keine solche Einstellung. Die Netzwerkkartentreiber handhaben Zeitüberschreitungen auf eigene Art – es gibt keine Möglichkeit zu steuern, wie lange eine Karte auf ein Paket wartet, das niemals ankommt. Das MDAC-Team möchte einen OLE DB-verwalteten Thread für die Zeitüberschreitung von Operationen hinzufügen, der auf den Zeitüberschreitungseinstellungen basiert.
- **CommandTimeout** startet, wenn der Datenbankserver den Befehl akzeptiert hat, und endet, wenn der Datenbankserver den ersten Eintrag zurückgibt. Sind Server oder Netzwerk beschäftigt (oder antworten gar nicht), kann mit dieser Einstellung die Steuerung nicht zurückerlangt werden.

Beide Eigenschafteneinstellungen hängen von der Netzwerkzeitüberschreitung ab – keine der ADO-Zeitüberschreitungseigenschaften nimmt Rücksicht darauf, wie viel Zeit ADO und die Provider der niedrigen Ebene zum Verbindungsaufbau zur LAN-Hardware benötigen. Da die NIC-Treiber (Network Interface Card) der niedrigen Ebene einen synchronen Netzwerk-API-Aufruf ausführen, und da dieser Aufruf nicht zurückgegeben wird bis die Zeitspanne für die Netzwerkzeitüberschreitung abgelaufen ist, wird der ADO-Zeitüberschreitungscode so lange blockiert, bis das Netzwerk antwortet – falls es antwortet.

Falls das Netzwerk arbeitet, können die meisten Verbindungen in weniger als fünf Sekunden hergestellt werden. Das Einstellen der **ConnectionTimeout**-Eigenschaft

auf einen Wert, der viel höher als 15 Sekunden liegt, ist unsinnig – und führt dazu, dass beim Client (beim Benutzer) eine Zeitüberschreitung auftritt. Das heißt, falls Ihre Clientanwendung den Benutzer zu lange auf eine Operation warten lässt, verliert dieser vielleicht die Geduld und drückt STRG+ALT+ENTF.

Anmerkung Für eine Verbindung erstellte **Command**- und **Recordset**-Objekte erben die **CommandTimeout**-Eigenschaft des **Connection**-Objekts. Während dieser Wert in einem **Command**-Objekt zurückgesetzt werden kann, ist dies in einem Recordset nicht möglich – die **Connection**-Einstellung (oder **Command**) muss übernommen werden.

4.2.8 Aufforderung zur Eingabe von Benutzer-ID und Kennwort

Nicht sämtliche ADO-Verbindungen werden durch menschliche Wesen initiiert. Nein, damit will ich nicht sagen, dass einige von Ihnen zur Gattung der Steine, Tiere, Pflanzen oder Außerirdischen gehören – auch, wenn dies vom Verhalten her gelegentlich zutreffen würde. (Reden wir gar nicht erst von dem Kartoffelkopf, mit dem ich letzte Woche zu tun hatte.) Was ich sagen möchte ist, dass einige Verbindungen über Komponenten gestartet werden, die keine Ahnung davon haben, welcher »Benutzer« die Operation anfordert. Eine Komponente der mittleren Schicht beispielsweise stellt lediglich eine Verbindung zur Datenquelle her und fordert Daten an. Diese Anmeldung erfolgt nicht als »Fred« oder anhand einer anderen spezifischen Benutzer-ID, die an einen Benutzer gekoppelt ist. Sofern nicht die Systemadministratoranmeldung verwendet wird (was eine böse Sache ist), verwendet die Komponente eine private Benutzer-ID und ein Kennwort, das speziell für diese und ähnliche Komponenten erstellt worden ist.

Einige von Ihnen möchten vielleicht dennoch eine Benutzer-ID und ein Kennwort abfragen und dieses an die ADO-Verbindung übergeben, um Zugriff auf das Datenbankverwaltungssystem zu erhalten (DBMS). Meiner Meinung nach ist dieser Ansatz nicht länger erforderlich und stellt in den meisten Fällen keine gute Lösung dar. Das Problem, das hier gelöst werden soll, ist die Benutzervalidierung – sollten Sie dem aktuellen (menschlichen) Benutzer Zugriff auf die Datenbank gewähren. Es gibt für dieses Problem verschiedene Ansätze – nachfolgend drei Vorschläge:

- Verwenden Sie die Windows-Netzwerksicherheit zur Validierung der Benutzer. Dies setzt voraus, dass der angemeldete Benutzer und der Benutzer, der das geschützte Programm (mit Benutzervalidierung) verwendet, ein und dieselbe Person sind. Auf diese Weise können sie die Windows-Anmeldenamen als Benutzer-ID verwenden, wenn eine Verbindung geöffnet wird. Gleichzeitig wird diese Benutzer-ID jedoch durch das Netzwerkprotokoll gegenüber dem Server offen gelegt – was bei TCP/IP nicht der Fall ist. Dieser Ansatz setzt ebenfalls vo-

raus, dass die Benutzer entweder in Gruppen organisiert sind, die der Server erkennt, oder dass sich die Benutzer als Einzelpersonen anmelden. Die administrative Seite dieses Ansatzes kann relativ kompliziert sein.

- Beschränken Sie den Zugriff auf die Datenbank so, dass die Endbenutzer die DBMS-Tabellen, Sichtenregeln, Trigger, Abfragen usw. weder anzeigen noch bearbeiten können. In diesem Fall kann nur über bestimmte (vielleicht geheime) Kombinationen aus Benutzer-ID und Kennwort auf ausgewählte Prozeduren zur Durchführung von Abfragen und Aktualisierungen zugegriffen werden. Dies ist mein bevorzugter Ansatz, der jedoch immer noch einen gewissen Validierungsmechanismus erfordert.
- Richten Sie eine zweite Sicherheitsschicht ein. Dieser Ansatz wird häufig benötigt, wenn die Benutzer sich mit spezifischen Rollen anmelden. Nachdem sich ein Benutzer anhand einer anwendungsverwalteten ID angemeldet hat, meldet sich die Anwendung selbst, oder eine zugrunde liegende Datenzugriffskomponente, mit einer geheimen Benutzer-ID sowie einem Kennwort an. Auf diese Weise erhalten Sie bei der Benutzerverwaltung mehr Steuerungsmöglichkeiten. Sie können beispielsweise den Datenbankzugriff eines Benutzers nach Tageszeit, Anzahl der Zugriffsversuche, Anzahl der derzeit angemeldeten Benutzer, Jobfunktion oder der Anzahl der Beine einschränken.

Eingabedialogfelder

Unabhängig von der gewählten Art der Benutzervalidierung möchten Sie gültige Kombinationen aus Benutzer-IDs und Kennwort selbst erfassen – diese Aufgabe sollte nicht über den Datenprovider erfolgen. Es soll also **nicht** über ODBC oder OLE DB ein Dialogfeld angezeigt werden (wie dargestellt in Abbildung 4.5), in dem Daten zu Benutzer-ID und Kennwort (sowie Datenbank, Datenquelle oder Server) abgefragt und an die **Connection Open**-Methode übergeben werden. Wird ein solches Dialogfeld geöffnet, erhält der Benutzer die Gelegenheit, eine Kombination aus Benutzer-ID und Kennwort zu erraten oder eine andere Datenbank auszuwählen – und dann so lange weiter zu raten, bis er richtig liegt. Nicht so gut.

ADO steuert dieses Verhalten über die **Prompt-**Eigenschaft des **Connection**-Objekts. Glücklicherweise ist das Standardverhalten keine Eingabeaufforderung (**no prompt**), und ADO wird angewiesen, einen auffangbaren Fehler zu erzeugen. Die Entwickler der Visual Database Tools dagegen waren jedoch aus Gründen, denen ich nicht zustimme, der Meinung, dass die Standardeinstellung **prompt always** lauten sollte, d.h., es erfolgt stets eine Eingabeaufforderung. Wenn Sie also den Datenumgebungs-Designer zum Öffnen einer Verbindung einsetzen, muss die **Prompt**-Eigenschaft außer Kraft gesetzt werden.

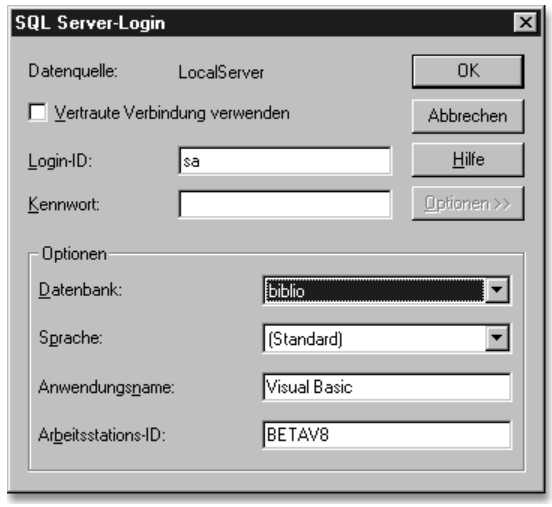


Abbildung 4.5 Die Ebene der Eingabeaufforderung sollte so eingerichtet werden, dass derartige Dialogfelder nicht angezeigt werden.

Dies können Sie mit Hilfe des Eigenschaftenfensters von Visual Basic und durch Auswahl des **DataEnvironment Connection**-Objekts erreichen. Stellen Sie sich lediglich darauf ein, dass Sie sich hierbei etwa sechs Mal bei der Datenbank anmelden müssen. Beachten Sie, dass der Datenumgebungs-Designer sowohl das Laufzeit- als auch das Entwurfszeitverhalten von Eingabeaufforderungen unterstützt.

Vertrauenswürdige Verbindungen

Sowohl die ODBC- als auch die OLE DB-Provider unterstützen die domänenverwaltete Sicherheit. Dies bedeutet, dass der Windows-Anmeldename sowie das Kennwort erfasst und als Aliasbenutzer-ID und -kennwort an den Datenprovider übergeben werden können. Dies setzt voraus, dass der Windows-Anmeldename des Benutzers verwendet werden kann, da es sich um eine Person handelt, die für den Datenbankzugriff über Ihre Anwendung als vertrauenswürdig eingestuft wurde. Dieser Ansatz wird in ausschließlichen TCP/IP-Konfigurationen wie MSDE bzw. dort, wo es sich bei dem »Benutzer« tatsächlich um eine Komponente der mittleren Schicht handelt, nicht unterstützt, da diese Komponenten nicht am Netzwerk angemeldet sind.

In ODBC können Sie der Verbindungszeichenfolge die Zeile **Trusted_Connection=Yes** hinzufügen und so angeben, dass die NT-Authentifizierung (domänenverwaltet oder integriert) verwendet werden soll. Alle weiteren Benutzer-ID- und Kennworteinträge in der Verbindungszeichenfolge und alle weiteren Einträge, die anhand der **Open**-Anweisung übergeben werden, werden ignoriert. Das Standardverhalten für einen ODBC-Provider lautet **Trusted_Connection=No.** Diese

Einstellung ermöglicht die Authentifizierung im gemischten Modus, d.h. es wird – je nach Situation – entweder die SQL Server- **oder** die NT-Authentifizierung verwendet.

Das Verwenden der integrierten Sicherheit (NT-Domänenauthentifizierung) mit ADO ist in der Regel sicherer als die Verwendung der SQL Server-Authentifizierung. Dies liegt daran, dass bei der clientgesteuerten Sicherheit (bei der Menschen involviert sind) Entwickler Benutzer-IDs und Kennwörter (gegen allen Rat) letztlich oft auf Festplatte speichern, entweder im Code oder in einer UDL- oder DSN-Datei. Jeder, der diese Dateien durchsieht, könnte das Kennwort erraten. Unglücklicherweise führt uns die Art und Weise, mit der unsere Tools und Dienste entwickelt werden, häufig in genau diese Falle. Es ist beispielsweise unmöglich, die integrierte Sicherheit zu verwenden, wenn ADO prozessintern mit IIS eingesetzt wird, da IIS unter dem lokalen Systemkonto ausgeführt wird und keine Autorisierung für das Netzwerk besitzt. Wenn Sie Ihre Anwendungen prozessextern ausführen, kann die integrierte Sicherheit genutzt werden, die Leistung der IIS-Implementierung ist so jedoch in vielen Szenarien unzureichend. Es gibt immer Probleme bei der Einrichtung von Laufzeitvariablen, die Benutzer-IDs und Kennwörter enthalten, ich möchte jedoch wetten, dass sich viele Entwickler nicht an die empfohlenen Vorgehensweisen halten.

OLE DB unterstützt ein ähnliches Sicherheitsschema, verwendet jedoch ein anderes Argument in der Verbindungszeichenfolge – **Integrated Security=SSPI** entspricht der Zeile **Trusted_Connection=Yes** von ODBC.

Ich habe eine Anwendung geschrieben, um die von ADO zurückgegebenen Fehler zu testen und zu protokollieren, die bei der Durchführung verschiedener Operationen auftreten, einschließlich der Verwaltung von Benutzer-IDs und Kennwörtern[6]. Ich begann mit einfachen Problemen hinsichtlich der Verbindungsberechtigungen und fand etwas Erstaunliches heraus. Nachdem die Anwendung mit Hilfe von ODBC-DSN und gültiger Kombination aus Benutzer-ID und Kennwort (nicht notwendigerweise **SA**) eine Verbindung hergestellt hatte, brauchte ich bei der erneuten Anmeldung an dieselbe Datenquelle keine Benutzer-ID und kein Kennwort mehr anzugeben. Wenn ich eine falsche Benutzer-ID oder ein falsches Kennwort eingab, schlug die Verbindungsherstellung über ADO fehl, wenn ich jedoch einfach die **Open**-Methode ohne Benutzer-ID und Kennwort verwendete, konnte die Verbindung wieder aufgebaut werden. Es stellte sich heraus, dass anstelle der fehlenden Benutzer-ID und des Kennwortes einfach mein Windows-Anmeldename verwendet wurde.

6. Diese Anwendung finden Sie auf der Begleit-CD-ROM zu diesem Buch. Suchen Sie einfach nach **ConnErrTrap.exe**.

Verwenden der Systemadministratoranmeldung als Benutzer-ID

Viele der auftretenden Sicherheitsprobleme werden verdeckt, wenn Sie **SA** (Systemadministrator) als Benutzer-ID für den Zugriff auf SQL Server verwenden. Da das Systemadministratorkonto Rechte für den Zugriff auf sämtliche Serverressourcen besitzt, ist mit diesem Konto praktisch alles möglich – einschließlich des Löschens der gesamten Datenbank auf einen Schlag. Das Preisgeben des Systemadministratorkennwortes stellt eine schwerwiegende Sicherheitsverletzung dar. Ohne dieses Kennwort könnten »gewöhnliche« Benutzer leicht daran gehindert werden, größeren Schaden anzurichten. Wenn Sie also schreiben, sprechen, unterrichten oder codieren, sollten Sie immer daran denken und darauf hinweisen, dass das Systemadministratorkennwort nicht als Benutzer-ID verwendet werden sollte.

4.2.9 Auswahl der Standarddatenbank

Wenn Sie eine Verbindung zu einem Datenbankverwaltungssystem herstellen, das mehrere Datenbanken unterstützt, beispielsweise SQL Server, wird Ihre Benutzer-ID mit einer der Datenbanken des Systems verknüpft. Wenn Ihr Systemadministrator auf Draht ist, wird die Standarddatenbank auf die Arbeitsdatenbank geändert, die Sie am wahrscheinlichsten verwenden und für die Sie die nötigen Zugriffsberechtigungen besitzen. Falls nicht, kann die Standarddatenbank leicht die Masterdatenbank sein – die Systemdatenbank, für die Sie keine Rechte besitzen (sollten). Da Sie sich nicht mit der Benutzer-ID **SA** angemeldet haben, sollte die Standarddatenbank für die Verbindung nicht eingestellt werden müssen. Denken Sie daran, falls nicht anders angegeben, werden Ihre SELECT-Anweisungen und Abfragen für die Standarddatenbank durchgeführt.

Tipp Wenn Sie den gefürchteten **ConnectionWrite("GetOverLapped-Result")-**Fehler erhalten, sollten Sie versuchen, die Netzwerkprotokolle zu ändern. Alles mit Ausnahme von Named Pipes scheint zu funktionieren, aber **multiprotocol** und TCP/IP scheinen in dieser Situation am besten geeignet zu sein – besonders, wenn sowohl Client als auch Server zur Verwendung von TCP/IP-Sockets konfiguriert wurden. In den meisten Fällen stellt TCP/IP im Hinblick auf Geschwindigkeit, Stabilität und Verwaltungsaufwand die beste Wahl dar.

Zur Auswahl einer neuen Standarddatenbank vor dem Öffnen der Verbindung fügen Sie der Verbindungszeichenfolge einfach ein weiteres Argument hinzu. Zum Ändern der Standarddatenbank in »Fred« fügen Sie einer ODBC-Verbindungszeichenfolge beispielsweise **Database=Fred** und einer systemeigenen OLE DB-Verbindungszeichenfolge **Initial Catalog=Fred** hinzu.

Nach dem Verbindungsaufbau kann die Standarddatenbank geändert werden, hierbei ist jedoch Vorsicht geboten. Sie können ADO nicht erlauben, während dieses Prozesses ein weiteres **Connection**-Objekt zu erzeugen. In dieser Hinsicht scheint ADO 2.5 bessere Manieren aufzuweisen als ADO 2.1. Wenn Sie mit Hilfe des TSQL-Befehls **Use <db>** versuchen, die Standarddatenbank von SQL Server zu ändern, und Sie verwenden die **Execute**-Methode des Command-Objekts zur Abfrageausführung, erstellt ADO zur Abfrageausführung **vielleicht** eine weitere Verbindung – insbesondere jedoch dann, wenn die aktuelle Verbindung durch eine andere Operation belegt wird.

Es ist sicherer, ADO mitzuteilen, dass Sie diese Änderung vornehmen – setzen Sie lediglich die **Current Catalog**-Eigenschaft (ODBC- oder OLE DB-Verbindungen) auf die neue Datenbank. ADO sendet eine **USE <db>-**Abfrage an die aktuelle Verbindung, jedoch nur dann, wenn derzeit keine noch ausstehenden Ergebnisse vorhanden sind. Anschließend wird für die ausgeführten Abfragen die neue Standarddatenbank verwendet. Wenn die weiteren Datenbankstatuseinstellungen nicht von Bedeutung sind, könnten Sie natürlich auch einfach die Verbindung schließen und erneut öffnen. Da es sich wahrscheinlich um eine im Pool befindliche Verbindung handelt, stellt dies keine erhebliche Leistungseinbuße dar.

Der folgende Code veranschaulicht eine typische **Open**-Methode, bei der anschließend ein Dump der Standarddatenbank (aktueller Katalog) erstellt wird, um zu zeigen, dass diese im Verlauf der Verbindungsöffnung durch den Server geändert wurde. Im Anschluss daran wird eine erneute Änderung mit Hilfe der **Property**-Auflistung des **Connection**-Objekts vorgenommen.

```
cn.Open "dsn=LocalServer;", "admin", "pw"    ' Anmeldung an der durch den
Systemadministrator definierten Standarddatenbank
Debug.Print cn.Properties("Current Catalog")    ' Ermitteln der Standard-
datenbank
cn.Properties("Current Catalog") = "adoclass"    ' Änderung der Standard-
datenbank in beliebigen Wert…
rs.Open "select ID from justinadoclass", cn    ' Jetzt kann eine Objektrefe-
renzierung in der neuen Datenbank erfolgen
```

Tipp Wenn Sie bei der Verbindungsherstellung mit einem OLE DB-Provider für ODBC (Standard) einen **Create File**-Fehler erhalten, kann dies auch einfach nur heißen, dass ADO den angegebenen Server nicht finden konnte. Sie haben doch einen angegeben, oder? Und der Server ist auch betriebsbereit, oder?

Verwalten des Verbindungsstatus

Das »Standard«-Attribut einer Datenbank ist eine der Statuseigenschaften für die Datenbankverbindung, die auf dem Server und (in geringem Umfang) in der **Property**-Auflistung des **Connection**-Objekts verwaltet wird. Diese Eigenschafteneinstellungen werden solange gespeichert, wie die Netzwerkverbindung zum Server aufrechterhalten wird. Erstellt ADO eine weitere Verbindung für Sie, werden dieser Verbindung keine Eigenschaften vererbt, die nach der Verbindungsöffnung vorgenommen werden. Dies rührt daher, dass die Verbindung aus der bereits erstellten **ConnectString**-Eigenschaft erstellt wird – selbst dann, wenn Sie die **Command**-Objekteigenschaften, z. B. **Current Catalog**, geändert haben. Findet jedoch über Microsoft Transaction Server ein Verbindungspooling für die erstellte und geschlossene Verbindung statt, erben andere Benutzer den auf dem Server verwalteten Status (da die physische Datenbankverbindung nie geschlossen wird).

Dieses Verhalten kann auch auf einem normalen Client/Server-System auftreten, da das Verbindungspooling nicht auf Microsoft Transaction Server beschränkt ist – es wird praktisch **überall** implementiert, es sei denn, sie deaktivieren es. Sie sollten die Auswirkungen einer Statusänderung auf dem Server besonders in den Situationen einschätzen können, in denen die Verbindung durch einen Poolingmechanismus wieder verwendet wird – auch bei Ihrem eigenen. Die Standarddatenbank ist nicht die einzige Eigenschaft, die auf Ihrem System ein Chaos anrichten kann. Bei jeder Verwendung der SET-Anweisung in TSQL wird der Datenbankstatus geändert. Da ADO jedoch nichts von dieser Änderung weiß, kann es sich dieser Änderung nicht entziehen – selbst dann nicht, wenn es hierfür konzipiert wäre (ist es aber nicht). Wenn Sie beispielsweise SET NOCOUNT ON **außerhalb einer gespeicherten Prozedur** einsetzen, wird die Änderung in der Verbindung gespeichert, bis diese geschlossen wird. Verwenden Sie eine SET-Anweisung in einer gespeicherten Prozedur, wird der Status nach Beendigung der gespeicherten Prozedur zurückgesetzt. Wenn Sie eine temporäre Tabelle oder einen temporären Cursor erstellen, werden diese ebenfalls als Teil des Datenbankstatus beibehalten. Jede Anwendung oder Komponente, die diese Verbindung erbt, erhält sämtliche Statusänderungen und weiß dabei nicht, welche Änderungen am ursprünglichen Status vorgenommen wurden.

Was geschieht also, wenn Ihr Entwicklungsteam beginnt, am Status herumzubasteln? Nun, da die Verbindung nicht mit anderen Komponenten oder Anwendungen gemeinsam genutzt wird, die abweichende Verbindungszeichenfolgen verwenden, kann dieses Problem in Grenzen gehalten werden. Wenn jedoch Verbindungen gemeinsam von unterschiedlichen Komponenten verwendet werden sollen (ein wünschenswertes Ziel), ist weitaus größere Vorsicht geboten. Die Symptome einer Statusbeschädigung sind Komponenten, die meistens funktio-

nieren, jedoch in unregelmäßigen Abständen ausfallen – beispielsweise nach bestimmten Operationen, die gerade durchgeführt wurden – nicht jedoch immer. Hässlich.

4.2.10 Verwenden des Datenansichtfensters zum Erstellen von Verbindungen

Das Datenansichtfenster der Visual Basic 6.0-IDE wurde zum Erfassen von Verbindungszeichenfolgen für ADO-Provider sowie zum Offenlegen des Datenbankschemas konzipiert, das angezeigt und bearbeitet werden soll. Nachdem eine Datenverknüpfung zu einem der mit zusätzlichen Funktionen ausgestatteten ADO-Provider hergestellt wurde, sehen Sie (wenigstens) eine Liste mit Tabellen und (vielleicht) Datenbankdiagramme, Sichten und gespeicherte Prozeduren, wie dargestellt in Abbildung 4.6.

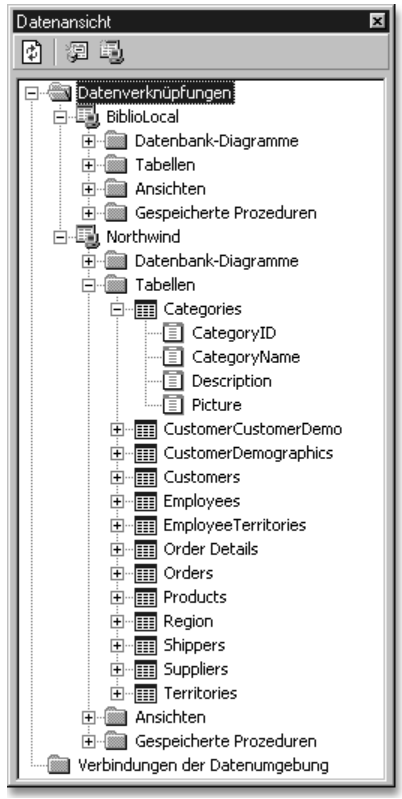


Abbildung 4.6 Das Datenansichtfenster von Visual Basic 6.0

Nicht alle Provider sind in der Lage, mit dem Datenansichtfenster zu arbeiten – oder wenigstens nicht vollständig. Ist eine Zusammenarbeit möglich, sollten alle Funktionen des Datenansichtfensters zur Verfügung stehen. Einzelne Tabellen können geöffnet werden, um Daten oder Schemata zu untersuchen und zu ändern – unter Verwendung einer interaktiven, grafischen Oberfläche. Wenn Sie Änderungen an der Tabelle vornehmen, wird die entsprechende SQL Server-Abfrage über das Datenansichtfenster an das Backend gesendet. Dies heißt, dass Sie Änderungen an »Livedatenbanken« vornehmen können – ob dies Sinn ergibt oder nicht. Darüber hinaus steht Ihnen die Möglichkeit offen, das SQL-Befehlsskript zu speichern, das anschließend an Ihren Systemadministrator weitergeleitet wird (wenn dieser noch mit Ihnen spricht, obwohl Sie das letzte Mal die Datenbank ohne seine Erlaubnis geändert haben).

An weiteren Informationen zum Datenansichtfenster interessiert? Mein Buch **The Hitchhiker's Guide to Visual Basic and SQL Server** enthält einen ausführlichen Abschnitt zu diesen Tools.

IMHO # IMHO = In My Humble Opinion (Meiner bescheidenen Meinung nach)Visual Basic versucht, das Verbinden mit einer ADO-Datenquelle so einfach wie möglich zu machen. Schade nur, dass die Codeentwickler nur das Nötigste miteinander besprochen haben. Meiner Meinung nach gibt es einen zu großen Unterschied zwischen dem, was implementiert wurde, und dem, was hätte implementiert werden können/sollen.

Nun denn, Sie verfügen jetzt über eine eingerichtete Datenverknüpfung. Beim nächsten Start von Visual Basic 6.0 wird diese Datenverknüpfung im Datenansichtfenster zur Verwendung angezeigt (wie dargestellt in Abbildung 4.7). Haben Sie erwartet, dass die Microsoft-Datenlinks hier angezeigt werden? Ich ja. Stimmt aber nicht. Die hier erstellten Datenverknüpfungen werden in der Registrierung gespeichert – die Microsoft-Datenverknüpfungen nicht.

Das Verwenden der Datenverknüpfungen zur Schemaverwaltung ist cool, aber Sie können auch die Verbindungszeichenfolge aus der Datenverknüpfung zur Verwendung im eigenen Code extrahieren. Unglücklicherweise ist dies nicht so einfach wie die Verwendung von ADODC. Sie können versuchen, auf die Eigenschaftenseite einer ausgewählten Datenverknüpfung zu wechseln, aber das angezeigte Dialogfeld **zeigt** lediglich die Verbindungszeichenfolge – das Kopieren in die Zwischenablage ist nicht möglich. Sie könnten natürlich zu Papier und Feder greifen und die wichtigsten Abschnitte niederschreiben, aber das wäre vielleicht etwas umständlich.

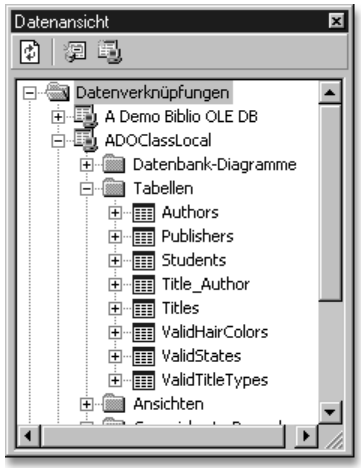


Abbildung 4.7 Das Datenansichtfenster von Visual Basic 6.0 mit den (soeben) durch VB6 generierten Datenverknüpfungen

Wie also kommen Sie an die bereits getestete Verbindungszeichenfolge? Nun, mit dem Datenumgebungs-Designer. Die Datenverknüpfungen werden (programmiertechnisch gesehen) erst dann sichtbar, wenn Sie eines der Elemente (Tabelle, Sicht, gespeicherte Prozedur, Datenbankdiagramm) über das Fenster des Datenumgebungs-Designers ziehen. (Abbildung 4.8 zeigt eine gespeicherte Prozedur, die aus dem Datenansichtfenster gezogen wurde.)

Abbildung 4.8 Der Datenumgebungs-Designer mit einer gespeicherten Prozedur, die mit der Drag&Drop-Methode aus dem Datenansichtfenster verschoben wurde

Versuchen Sie dies:

1. Starten Sie anhand der Datenvorlage ein neues Visual Basic-Projekt. Auf diese Weise wird Ihrem Projekt ein ADO 2.0-Verweis, ein Datenumgebungs-Designer und ein Datenberichts-Designer hinzugefügt. Sie können den Datenberichts-Designer entfernen, falls dieser nicht benötigt wird.

2. Richten Sie eine Datenverknüpfung zu Ihrer bevorzugten Datenquelle ein (Abbildung 4.9). Ich stelle in diesem Fall eine Verbindung zur Biblio-Testdatenbank auf meinem lokalen (MSDE) SQL-Server her. Klicken Sie einfach auf das Symbol zum Hinzufügen einer neuen Datenverknüpfung, und geben Sie in den Dialogfeldern (wie zuvor besprochen) die benötigten Informationen an. Ja, das Datenansichtfenster verwendet die gleichen OLE DB-Dialogfelder. (Siehe Abbildung 4.1.)

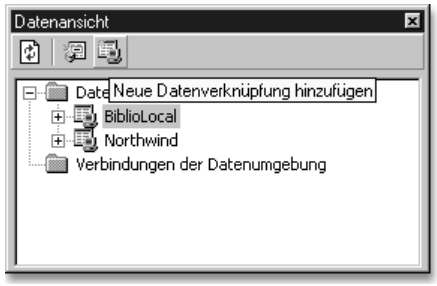


Abbildung 4.9 Verwenden des Datenansichtfensters zum Erstellen einer neuen Datenverknüpfung

3. Wenn Sie alle OLE DB-Providereigenschaften angegeben haben, klicken Sie auf OK und benennen die neue Datenverknüpfung. Verwenden Sie einen intuitiven Namen. »Datenlink1« wäre nicht aussagekräftig genug.
4. Klicken Sie auf das Tabellensymbol unter der neuen Datenverknüpfung, um die sichtbaren Tabellen zu erweitern. Sind keine Tabellen sichtbar, ist möglicherweise eine Zusammenarbeit zwischen Provider und Datenansichtfenster nicht möglich, oder Sie haben keine Berechtigung, Tabellen niedriger Ebene anzuzeigen. Die Tabellen werden jedoch häufig auch dann angezeigt, wenn Sie eigentlich keine Berechtigung für diesen Zugriff haben. Auf die Zugriffsverletzungen aufgrund fehlender Berechtigungen werden Sie erst zur Laufzeit aufmerksam gemacht. Wenn Sie nicht der Besitzer der Tabellen sind, sind Sie nicht in der Lage, diese mit Hilfe der Visual Database Tools zu ändern.
5. Doppelklicken Sie im Projektfenster auf den Datenumgebungs-Designer, um ein neues Datenumgebungs-Designerfenster zu öffnen.
6. Wählen Sie eine der Tabellen aus, und ziehen Sie sie auf das Fenster des Datenumgebungs-Designers. Zu diesem Zeitpunkt erstellen der Datenumgebungs-Designer und ADO basierend auf der Verbindungszeichenfolge für die Datenverknüpfung ein neues **Connection**-Objekt (**Verbindung2**) sowie ein ADO-**Command**-Objekt, das als eine Methode eines **DataEnvironment**-Objekts gekapselt ist.

7. Klicken Sie zum Extrahieren der Verbindungszeichenfolge für **Verbindung2** einfach auf das zugehörige Symbol im Datenumgebungs-Designer, drücken Sie F4, um die Eigenschaftenseite von Visual Basic zu öffnen, und kopieren Sie die Eigenschaft **ConnectionSource** in die Zwischenablage – und anschließend in Ihren Anwendungscode.
8. Denken Sie daran, jegliche mehrfach aufgeführten doppelten Anführungszeichen zu entfernen, bevor Sie die Zeichenfolge in Visual Basic einsetzen.

Verwenden des Datenumgebungs-Designers

Nachdem Sie nun einen Datenumgebungs-Designer in Ihrem Visual Basic-Projekt eingerichtet haben, wie kann dieser am besten in einem Visual Basic-Programm eingesetzt werden? Unglücklicherweise gibt es eine Reihe von Dingen, die den effizienten Einsatz des Datenumgebungs-Designers erschweren. Sicher, Sie können mit Hilfe des Designers Anwendungen erstellen, aber es gibt einfach zu viele Berichte über mysteriöse Probleme und enttäuschte Manager. Aufgrund dieser negativen Berichte ist der Datenumgebungs-Designer auch von meiner Liste der »empfohlenen Vorgehensweisen« verschwunden. Verstehen Sie mich nicht falsch – es gibt einige gute Features des Datenumgebungs-Designers, die für einen Ausgleich sorgen, aber es gibt auch eine Reihe von Problemen, die die Verwendung des Designers zu einer Herausforderung machen, hat man erst einmal den ersten Schritt getan. Sehen Sie die Skispitzen dort drüben aus dem Schnee schauen? Das ist Jim. Er hat versucht, den Datenumgebungs-Designer jenseits des **double-diamond**-Bereichs[7] der Datenumgebung einzusetzen.

Ein Verständnis der Funktionsweise des Datenumgebungs-Designers kann Sie auf den Boden der Tatsachen zurückholen. Erstens sollten Sie daran denken, dass der Datenumgebungs-Designer (wie viele andere Tools auch) eine einfache Möglichkeit zum Offenlegen von ADO gegenüber dem Entwickler darstellt, nur, dass der Datenumgebungs-Designer dieses Ziel mit einer anderen Laufzeitengine erreicht. Der Visual Basic-Designer (im Gegensatz zu einem Assistenten) erstellt also keinen Quellcode, an dem Sie herumbasteln können. Die DSR-Datei enthält binäre Anweisungen, mit denen beschrieben wird, wie die **DataEnvironment**-Laufzeitengine das **Connection**-, das **Command**- und die hierarchischen **Shape**-Objekte erstellen soll. Dies bedeutet, dass Sie nicht in der Lage sind, **DataEnvironment**-Objekteigenschaften im Code zu ändern – zumindest nicht in größerem Umfang. Ich habe verschiedenen Leute gesehen, die versucht haben, die Verbindungszeichenfolge oder die **Command**-Eigenschaften zu ändern, nur um festzustellen, dass das System verrückt gespielt hat.

7. Anm. d. Übers.: Im Amerikanischen wird mit »double-diamond« auch eine Skipiste mit hohem Schwierigkeitsgrad bezeichnet.

4.2.11 Verwenden von Datenquellensteuerelementen

Nach dem Feedback, das ich im Verlaufe der Jahre erhalten habe, wenden sich immer mehr Entwickler von den Datenquellensteuerelementen wie ADO (ADODC), DAO (Data) und RDO-Datensteuerelementen (RDC) ab. Es ist nicht so, dass kein Platz mehr für diese Steuerelemente wäre, aber als damit begonnen wurde, Anwendungen zur Handhabung immer kniffligerer Situationen zu entwickeln, kam man zu dem Schluss, dass diese Steuerelemente im Weg sind. Da diese Steuerelemente versuchen, viele Operationen »für uns« im Hintergrund auszuführen, werden gleichzeitig die Steuerungsmöglichkeiten des Entwicklers eingeschränkt. Es wird ohnehin schon sehr viel Code geschrieben – da ist das »Herumschreiben« um die Fehler und Features dieser intelligenten Steuerelemente nicht auch noch nötig.

4.2.12 Ausführungsmethoden für Verbindungsabfragen

So, jetzt sind wir soweit, einige Abfragen für das **Connection**-Objekt auszuführen. Nein, Sie brauchen kein **Command**-Objekt zu erstellen, aber wenn durch die Abfrage ein Rowset zurückgegeben wird, ist das Erstellen eines Recordsets erforderlich. Nähere Informationen hierzu finden Sie im Kapitel 6, »Recordsetstrategien«.

Angenommen, Sie haben die Verbindung zur Ausführung einer Abfrage bzw. Aktionsabfrage geöffnet. Hierzu stehen nach dem Verbindungsaufbau verschiedene Möglichkeiten zur Verfügung.

- Verwenden Sie die **Execute**-Methode des **Connection**-Objekts zum Ausführen einer Abfragezeichenfolge. Auf diese Weise werden Zeilen in ein Recordset zurückgegeben, dass zuvor erstellt wurde.
- Verwenden Sie die **Execute**-Methode des **Connection**-Objekts zum Ausführen einer Aktionsabfrage, bei der keine Zeilen zurückgegeben werden.
- Führen Sie ein benanntes ADO-**Command**-Objekt für das **Connection**-Objekt aus. In diesem Fall werden die **Command**-Argumente und ein zuvor erstelltes Recordset übergeben.
- Führen Sie eine gespeicherte Prozedur aus, indem Sie diese einfach als Methode des **Connection**-Objekts verwenden. ADO akzeptiert das gespeicherte Prozedurargument als Methodenargument und akzeptiert hierbei ein benanntes **Recordset**-Objekt als letztes Argument. (Cool.) Wir diskutieren den Einsatz (und die Auswirkung) dieser Methode in Kapitel 6, »Recordsetstrategien«.

Da wir das **Command**-Objekt bisher noch nicht besprochen haben, werden wir auch die Erläuterung der **Command**-Objektausführung verschieben, aber sehen wir uns die Fähigkeit des **Connection**-Objekts an, SQL-Anweisungen und Verwaltungsabfragen auszuführen oder die Zeilen einer Tabelle an ein Recordset auszugeben.

Die grundlegende Syntax der **Execute**-Methode des **Connection**-Objekts erfordert Folgendes:

- **Command text** – Bei diesem Argument kann es sich um eine Zeichenfolge handeln, die eine SQL-Anweisung, einen Tabellennamen, den Namen einer gespeicherten Prozedur, eine URL oder Text enthält, der für den Datenprovider Sinn ergibt.
- Optional können Sie eine **Long**-Variable bereitstellen, um die Anzahl der Zeilen abzurufen, die von der Abfrage betroffen sind – vorausgesetzt, der Provider gibt ein Resultset zurück.
- Ebenfalls optional ist die Übergabe einiger weniger Parameter, die ADO dabei unterstützen, die Ausführungsart der Abfrage festzulegen. Grundsätzlich wird mit diesen Parametern das Befehlstextargument beschrieben und ADO wird darüber informiert, wie die Abfrage verarbeitet werden soll. Diese Parameter werden in Kapitel 5, »ADO-Befehlsstrategien« besprochen.

Eine typische **Execution**-Methodenauslösung könnte beispielsweise so aussehen:

```
Set rs = cn.Execute("select City, State from publishers",,adCmdText)
```

Oder für Aktionsabfragen:

```
cn.Execute("truncate table framis",, adExecuteNoRecords)
```

Da ADO das **Recordset**-Objekt für Sie erstellt, handelt es sich immer um ein schreibgeschütztes, vorwärts gerichtetes, cursorloses Resultset.

Tipp Wenn Sie ein **Recordset**-Objekt mit mehr Funktionalität benötigen, erstellen Sie ein **Recordset**-Objekt mit den gewünschten Eigenschafteneinstellungen und verwenden Sie die **Open**-Methode des **Recordset**-Objets zum Ausführen der Abfrage sowie zum Zurückgeben des gewünschten Cursortyps. Sie können mit der **Command**-Methode auch eigene **Recordset**-Objekte verwenden – dieses Thema wird ausführlich in Kapitel 5, »ADO-Befehlsstrategien«, besprochen.

4.2.13 Verbindungspooling

In ODBC wurde vor einiger Zeit das Verbindungspooling implementiert, und OLE DB folgte diesem Beispiel zur Zeit von ADO 2.1. Diese Funktion wurde implementiert, um den Durchsatz von Microsoft Transaction Server und IIS-verwalteten Servern zu erhöhen. Hierbei werden die durch Komponenten der mittleren Schicht und Webseiten erstellten Verbindungen nicht geschlossen, sondern durch das Verbindungspooling als wieder verwendbar markiert. Versucht eine andere

Anwendung, eine Verbindung mit einer Verbindungszeichenfolge zu öffnen, die mit einer bereits geöffneten Verbindung übereinstimmt (exakt übereinstimmt – Byte für Byte), wird das Verbindungshandle einfach an den neuen »Benutzer« übergeben. Das Verbindungspooling ist standardmäßig aktiviert. Wenn Sie nicht mit dem Verbindungspooling arbeiten möchten, ändern Sie den **CPTimeout**-Wert in der Registrierung auf 0 (**HKEY_LOCAL_MACHINE\Software\ODBC\ODBCINST.INI\SQL Server**) – oder entfernen Sie den Eintrag.

ADO verwendet standardmäßig das **Sitzungspooling** von OLE DB, um über einen Pool Verbindungen zur Datenbank zu verfügen. In einigen Fällen ziehen Sie vielleicht das ODBC-Verbindungspooling dem OLE DB-Sitzungspooling vor.

Zur Aktivierung des OLE DB-Sitzungspoolings in einer Desktopanwendung müssen Sie sicherstellen, dass in der Visual Basic-Anwendung auf globaler Ebene ein Verweis auf ein **Connection**-Objekt erhalten bleibt. Mit anderen Worten, Microsoft erwartet, dass für alle Anwendungen, die über einen Pool verfügen möchten, immer eine Verbindung offen gehalten wird. Durch das Offenhalten einer Verbindung wird ein Verweis auf die **IDataInitialize**-Schnittstelle unterhalten – die OLE DB-Dienstkomponente, in der das Sitzungspooling stattfindet. In MTS und ASP dagegen ist dies nicht erforderlich. Diese Umgebungen gehen Hand in Hand mit dem ODBC-Verbindungspooling und dem OLE DB-Sitzungspooling, ohne dass Sie ständig durch brennende Reifen springen müssen (wenigstens die meiste Zeit). Microsoft empfiehlt die Verwendung des Sitzungspoolings, sofern keine Bugs oder andere Probleme vorliegen, die Sie hiervon abhalten. Weitere Informationen zum »freien Sitzungspooling« finden Sie in der MSDN-Dokumentation und in der entsprechenden Website – es stehen verschiedene Artikel zu diesem Thema bereit.

Umschalten auf das Verbindungspooling in ODBC

Zur Aktivierung des ODBC-Verbindungspoolings von einer Visual Basic/ADO-Anwendung aus, erwartet Microsoft von Ihnen, dass Sie durch ein paar brennende Reifen springen – mit an den Füßen festgebundenen Steinen. Hier die zwei erforderlichen Schritte:

1. Wenn Sie über den ODBC-Administrator 3.5 oder höher verfügen, öffnen Sie über die Systemsteuerung den ODBC-Datenquellen-Administrator, wie dargestellt in Abbildung 4.10. Wählen Sie die Registerkarte **Verbindungs-Pooling**. Suchen Sie in der Liste nach dem verwendeten Treiber und doppelklicken Sie auf diesen. Aktivieren Sie die Option **Verbindungen zu diesem Treiber zusammenlegen**, und geben Sie im darunter angezeigten Feld einen Zeitüberschreitungswert ein.

Abbildung 4.10 Verwenden des ODBC-Datenquellen-Administrators zur Auswahl der Verbindungspoolingoptionen

2. Fügen Sie in Ihrer Anwendung **SQLSetEnvAttr** einen ODBC-API-Funktionsaufruf mit den geeigneten Optionen zur Aktivierung des ODBC-Verbindungspoolings für den Prozess hinzu. Ja, ich sagte ODBC-API-Funktionsaufruf. Seufz. Diese Funktion sollte nur einmal pro Prozess aufgerufen werden und muss vor dem Ausführen von ADO-Code aufgerufen werden. Wenn Sie nicht mit der ODBC-API vertraut sind, können Sie alles darüber im **Guide** lesen und/oder sich das folgende Beispiel ansehen.

Hinweis Dieser Code wurde einem Artikel der Microsoft-Knowledgebase (Q237844) entnommen. Ich habe den Code in eine Klasse und auf die CD gepackt, dort finden Sie ihn also auch. Beachten Sie außerdem, dass dieser Code nicht aufgerufen werden muss, wenn Sie Ihre Verbindungen innerhalb von MTS oder ASP öffnen – diese Komponenten kümmern sich selbst darum.

Dieser Code führt dazu, dass **SQLSetEnvAttr** die ODBC-API verwendet.
Viel Glück!

```
Dim rc As Integer
Const SQL_ATTR_CONNECTION_POOLING = 201
Const SQL_CP_ONE_PER_DRIVER = 1
Const SQL_IS_INTEGER = -6
```

```
Const SQL_CP_OFF = 0
Private Declare Function SQLSetEnvAttr Lib "odbc32.dll" ( _
                ByVal EnvironmentHandle As Long, _
                ByVal EnvAttribute As Long, _
                ByVal ValuePtr As Long, _
                ByVal StringLength As Long) As Integer
Public Function TurnOffPooling() As Integer
     TurnOffPooling = SQLSetEnvAttr(0&, _
             SQL_ATTR_CONNECTION_POOLING, _
             SQL_CP_OFF, _
             SQL_IS_INTEGER)
End Function
Public Function TurnOnPooling() As Integer
   TurnOnPooling = SQLSetEnvAttr(0&, _
              SQL_ATTR_CONNECTION_POOLING, _
              SQL_CP_ONE_PER_DRIVER, _
              SQL_IS_INTEGER)
   If rc <> 0 Then
      Debug.Print "SQLSetEnvAttr Error " & rc
   End If
End Function
```

Wenn Sie wirklich **SQLSetEnvAttr** verwenden möchten, lesen Sie auf jeden Fall Seite 943 der **ODBC 3.0 Programmer's Reference, Volume 2** (Microsoft Press), in der die Details des API-Aufrufs und dessen Auswirkungen auf Ihr System und alle anderen besprochen wird. Sie sind offiziell gewarnt worden.

Ändern der Verbindungspoolingoptionen in OLE DB

Für OLE DB kann das Verbindungspooling dauerhaft deaktiviert werden. (Ich empfehle Ihnen, nicht an der Registrierung herumzuspielen, denn wie eine vor dem Spiegel an sich selbst durchgeführte Gehirnoperation, birgt auch dieses Vorgehen gewisse Risiken. In dem Moment, in dem Sie glauben, dass alles perfekt ist, stürzt Ihre Tochter herein und stößt bei der Makeup-Überprüfung an Ihren Ellbogen.) In diesem Fall müssen Sie die Provider-GUID ermitteln (suchen Sie in der Registrierung nach dem Namen). Der Providereintrag für SQLOLEDB befindet sich beispielsweise unter **HKEY_LOCAL_MACHINE\Software\CLASSES\CLSID**{0C7FF16C-38E3-11d0-97AB-00C04FC2AD98}. Merken Sie sich unbedingt die GUID – nur so können Sie Ihre computerbesessenen Freunde in der Kneipe beeindrucken. Nachdem Sie die GUID ermittelt haben, können Sie (unbesorgt?) den zugehörigen **OLEDB_SERVICES-Schlüssel** unter Verwendung der Werte in Tabelle 4.4 ändern.

Dienst	Wert
Alle Dienste (Standardeinstellung)	0xffffffff
Alle Dienste mit Ausnahme von Pooling und AutoEnlistment	0xfffffffc
Alle Dienste mit Ausnahme von Client Cursor	0xfffffffb
Alle Dienste mit Ausnahme von Pooling, AutoEnlistment und Cursor	0xfffffff0
Keine Dienste	0x00000000

Tabelle 4.4 Registrierungseinstellungen für OLEDB_SERVICES

Ich weiß nicht, wie Sie sich beim Ändern der Registrierung fühlen, mir ist jedenfalls nicht wohl dabei. Es wird so nicht nur Ihr Code beeinflusst, sondern der aller anderen Entwickler auch. Eine empfohlene Vorgehensweise lautet daher, die Verbindungszeichenfolge einfach so zu ergänzen, dass das **OLE DB Service**-Argument auf eine der in Tabelle 4.5 gezeigten Optionen gesetzt wird.

Dienst	Wert
Alle Dienste (Standardeinstellung)	`"OLE DB Services = -1;"`
Alle Dienste mit Ausnahme von Pooling und AutoEnlistment	`"OLE DB Services = -4;"`
Alle Dienste mit Ausnahme von Client Cursor	`"OLE DB Services = -5;"`
Alle Dienste mit Ausnahme von Pooling, AutoEnlistment und Cursor	`"OLE DB Services = -7;"`
Keine Dienste	`"OLE DB Services = 0;"`

Tabelle 4.5 OLE DB Services-Argumente für die Verbindungszeichenfolge

Nähere Informationen zum Verbindungs- und Sitzungspooling finden Sie in der MSDN-Dokumentation. Die folgenden Artikel aus der Knowledgebase könnten ebenfalls von Interesse sein: Q189410, Q237844, Q169470 und einer der besten: Q176056 »INFO: ADO/ASP Scalability FAQ«. Siehe **http://www.msdn.microsoft.com**. Nein, nicht alle diese Artikel sind auf der/den MSDN-CD(s) enthalten.

Warnung Wenn ADO es für nötig hält, wird zur Ausführung einer Abfrage möglicherweise ein neues **Connection**-Objekt erstellt. Hierbei erstellt ADO unbemerkt einen neuen Transaktionsbereich und eine Datenbankverbindung, die nach Beendigung der Operation wieder geschlossen wird.

Ablaufverfolgung für verwaiste Verbindungen beim Verbindungspooling

Ich habe bereits viele Ratschläge dazu gehört, wie man beim Verbindungspooling herausfindet, ob die Verbindungen erneut gebraucht, missbraucht oder gar nicht gebraucht und sich selbst überlassen werden. Ich denke, die folgende Liste, die ich in meiner Mailbox gefunden habe, ist wahrscheinlich eine der besten zu diesem Thema:

1. Verwenden Sie den SQL Server 7.0 Profiler (dargestellt in Abbildung 4.11) um zu überprüfen, ob Ihre Verbindungen stets erstellt, aber nie erneut verwendet werden. Sie sollten in der Lage sein, dies mit Hilfe der Spalten **Verbindungs-ID** und **SPID** zu prüfen.

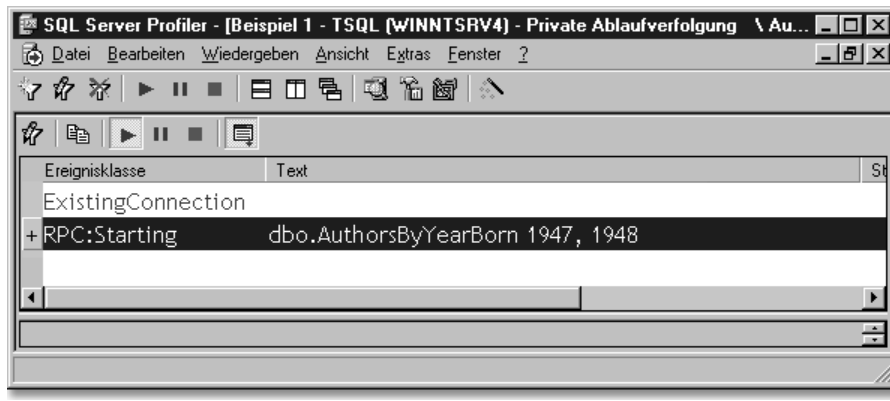


Abbildung 4.11 Verwenden des SQL Server Profiler zum Verfolgen von TSQL-Abfragen

2. Prüfen Sie, ob für diese Verbindungen nach 60 Sekunden eine Zeitüberschreitung erfolgt, oder ob sie aktiv bleiben, aber nie wieder verwendet werden. Werden die Verbindungen nicht wieder verwendet, weisen jedoch nach 60 Sekunden eine Zeitüberschreitung auf, kann es sein, dass durch das Sitzungspooling keine ordnungsgemäße Wiederverwendung der Verbindungen stattfindet.
3. Versuchen Sie als Nächstes zu prüfen, ob alle Verbindungen nur einmalig verwendet werden oder ob nur einige ungenutzt bleiben (und ob deren Anzahl steigt). Mehren sich diese Verbindungen und findet keine Zeitüberschreitung statt, haben wir es wahrscheinlich mit einer nicht ordnungsgemäß veröffentlichten Schnittstelle zu tun, daher werden die Verbindungen nicht in den Pool zurückgegeben.

4. Bleiben nur einige Verbindungen ungenutzt, steigt aber deren Anzahl, schauen Sie nach, ob die Verbindungen an eine bestimmte Aktivität gebunden werden können. Verwenden Sie **dbcc inputbuffer(spid)** im Query Analyzer zur Anzeige der letzten Aktivität für die betreffende Verbindung.
5. Prüfen Sie, ob Transaktionsoperationen durchgeführt wurden. Sehen Sie nach, ob einige Seiten **@transaction=required**-Tags aufweisen und andere nicht.
6. Prüfen Sie, ob die Authentifizierung der ASP-Seiten **Anonymous**, **Basic** oder **NTLM** lautet. Die Einstellungen **Basic** und **NTLM** können dazu führen, dass Verbindungen auf Pro-Benutzer-Basis erstellt werden, wodurch die Vorteile des Sitzungspoolings nicht genutzt werden. In einem System mit mehreren Prozeduren kann dies zu einer schnellen Anhäufung von Verbindungen führen.
7. Fügen Sie Ihrer Verbindungszeichenfolge den Wert **Use Procedure for Prepare=0** hinzu, und prüfen Sie, ob sich die Situation bessert. Auf diese Weise wird der Provider daran gehindert, während der Abfrageausführung temporär gespeicherte Prozeduren zu erstellen.
8. Zum Schluss steht natürlich noch ein Möglichkeit zur Verfügung – versuchen Sie, das Sitzungspooling zu deaktivieren.

Das Sitzungspooling kann auf verschiedene Weisen deaktiviert werden. Aber die obige Liste scheint Sinn zu ergeben. Berücksichtigen Sie, dass mit ADO 2.1 oder späteren Service Packs von ADO 2.1 alle ODBC-Treiber standardmäßig in das Sitzungspooling einbezogen werden, wenn der OLE DB-Provider für ODBC verwendet wird (MSDASQL). (Wenn Sie einen ODBC-Treiber von ADO verwenden, verwenden Sie MSDASQL.) Innerhalb einer Benutzeranwendung bedeutet dies, dass bei mindestens einer geöffneten Verbindung alle folgenden Verbindungen **automatisch** im Pool platziert werden, selbst dann, wenn das ODBC-Verbindungspooling im ODBC-Administrator für diesen Treiber **deaktiviert** wurde. Werden Verbindungen im Pool platziert, bleiben Sie auch nach dem Schließen für hartcodierte 60 Sekunden geöffnet.

Sie können zur Deaktivierung des Poolings Folgendes ausprobieren:

- Stellen Sie sicher , dass das ODBC-Verbindungspooling für den ODBC-Treiber deaktiviert wurde. Sie können dies im ODBC-Administrator auf der Registerkarte **Verbindungs-Pooling** (siehe Abbildung 4.10) prüfen – neben dem Treibernamen sollte der Eintrag **<nicht zusammengelegt>** angezeigt werden. Ist dies nicht der Fall, doppelklicken Sie im Listenfeld auf den betreffenden Treiber (den verwendeten Treiber) und deaktivieren Sie ihn.
- Fügen Sie allen Verbindungszeichenfolgen den Wert **OLE DB Services =2** hinzu, um das Sitzungspooling für ADO zu deaktivieren. Weitere Informationen zu den Einstellungen für **OLE DB Services** in ADO finden Sie im MSDN-White-

paper »Pooling in the Microsoft Data Access Components«. Eine typische Verbindungszeichenfolge könnte so aussehen:

```
conn.open "DSN=MyDB2DataSource;UID=Me;PWD=My;OLE DB Services=-2;"
```

Die Regel, dass eine dauerhafte Verbindung pro Satz eindeutiger Benutzeranmeldeinformationen vorhanden sein muss, gilt **nur**, wenn Ihre Anwendung **nicht** unter IIS (4.0) oder MTS (2.0) ausgeführt wird. IIS **und** MTS verwalten den Pool (wenigstens so lange, bis eine Zeitüberschreitung für die Verbindung eintritt oder diese nicht länger wieder verwendet wird). Außerhalb dieser zwei Umgebungen wird der Pool (an OLE DB-Ressourcen) aufgelöst, wenn die letzte Verbindung in Ihrer Anwendung, die den eindeutigen Satz Benutzeranmeldeinformationen zur Erstellung des Pools enthält, geschlossen wird. In MTS und IIS wird ein dauerhafter Pool eingerichtet.

Das Speichern von Verbindungen kann ein PIA darstellen – besonders dann, wenn Sie Ihre Anwendung entwerfen und nicht wissen, dass dies wichtig ist. Denken Sie jedoch daran, dass beim ODBC-Verbindungspooling Verbindungen durcheinander gebracht werden, unabhängig von Authentifizierung und zugrunde liegendem Quelltreiber/Datenspeicher. Wenn Ihre Anwendung eine Verbindung anfordert, wird sie während der Suche nach einer passenden Verbindungszeichenfolge im Pool blockiert. Das Sitzungspooling dagegen wurde so optimiert, dass eine solche Blockierung nicht auftritt. Der einzige Pool, der nach einer geöffneten Verbindung durchsucht wird, ist der Pool, der mit der Benutzerauthentifizierung übereinstimmt (falls vorhanden). Beim Sitzungspooling treten weniger Sperrprobleme für einen vorhandenen Pool auf, d.h. Sie sollten Ihre Verbindung sehr viel schneller zurückerhalten. OLE DB verwaltet (je nach Anzahl der Prozessoren im Computer) ebenfalls Pool**sätze,** um freie Verbindungen so schnell wie möglich aufzufinden.

4.2.14 Asynchrones Öffnen einer Verbindung

In einer traditionellen Client/Server-Anwendung öffne ich das **Connection**-Objekt oft (üblicherweise) asynchron. Ja, ich werte dies als empfohlene Vorgehensweise, da die einem Benutzer zur Verfügung stehende Zeit besser genutzt wird. In meinen Anwendungen werden die Formulare schneller gezeichnet und nein, ich aktiviere die Steuerelemente auf dem Formular nicht, die Zugriff auf die Verbindung benötigen – nicht, bevor die Verbindung geöffnet ist.

Der Visual Basic-Code zur asynchronen Verbindungsherstellung ist einfach – fügen Sie der **Open**-Methode lediglich die **adAsyncConnect**-Option hinzu.

```
cn.Open "dsn=localserver", "admin", "pw", adAsyncConnect
```

Anmerkung Nicht alle Provider unterstützen asynchrone Operationen, und es sind keine zusätzlichen Eigenschaften im **Connection**-Objekt vorhanden, mit denen geprüft werden kann, ob asynchrone Verbindungen erstellt oder asynchrone Abfragen ausgeführt werden können.

Bei der asynchronen Verbindungsherstellung beginnt ADO mit dem Prozess des Verbindungsaufbaus, während der Clientthread andere Aufgaben erledigt, beispielsweise das Abschließen von **Form_Load**. Während der Verbindungsherstellung wird die **State**-Eigenschaft des **Connection**-Objekts auf **adStateConnecting** gesetzt.

Nachdem die Verbindungsherstellung über ADO abgeschlossen wurde, erfolgreich oder nicht, löst ADO das **ConnectComplete**-Ereignis aus und deaktiviert das **adStateConnecting**-Bit in der **State**-Eigenschaft des **Connection**-Objekts. Gibt die Ereignisbehandlungsroutine ein **Error**-Objekt zurück, dass auf **Nothing** gesetzt wurde, war die Verbindungsherstellung erfolgreich. Ist ein Fehler aufgetreten, sind nähere Informationen in der **Error**-Auflistung des **Connection**-Objekts enthalten. Zusammen mit einem Zeiger auf das **Error**-Objekt erhalten Sie (ohne Aufpreis) ein **Status**-Objekt, das ebenfalls einen Bericht darüber bereitstellt, ob die Operation erfolgreich war oder nicht. Eine Prüfung des Status nach **adStatusOK** erledigt dies.

Muss also eine Ereignisbehandlungsroutine erstellt werden, um zu prüfen, ob die Verbindungsoperation abgeschlossen wurde? Nein, Sie können einfach die **State**-Eigenschaft prüfen und warten, bis das **adStateOpen**-Bit aktiviert wird (und das **adStateConnecting**-Bit deaktiviert wird).

4.2.15 Prüfen, ob die Verbindung noch steht

Möchten Sie wissen, ob die Verbindung noch geöffnet ist? Nun, es gibt keine Eigenschaften für das **Connection**-Objekt, die sich ändern, wenn die Verbindung verloren geht. Und Sie können auch nicht einfach nur die **State**-Eigenschaft prüfen, da ADO keine eigene Abfrage oder Prüfung durchführt. Eventuell haben Sie das Bedürfnis, diese Prüfung selbst durchzuführen. Versuchen Sie, eine Abfrage geringen Umfangs zu senden, beispielsweise SELECT 1. Diese Abfrage erfordert etwa 3 Pakete und ist relativ effizient. Funktioniert die Abfrage, ist die Verbindung weiterhin geöffnet. Nicht, dass die Verbindung auch dann offen ist, wenn Sie sie benötigen, aber wenigstens findet keine Zeitüberschreitung für weitere »n« Sekunden statt.

4.3 Verwalten von Transaktionen

ADO kann eine Transaktionsverwaltung für Sie durchführen – diese Verwaltung findet auf Verbindungsebene statt, wobei vorausgesetzt wird, dass der Datenprovider Transaktionen unterstützt. Unterstützt Ihr Provider Transaktionen, ist die Eigenschaft **Connection.Properties(»Transaction DLL«)** vorhanden. Unterstützt der Provider keine Transaktionen, lösen Sie beim Verweisen auf diese Eigenschaft einen auffangbaren Fehler aus. Ich denke, dies ist ein untrügliches Zeichen dafür, dass keine Unterstützung gegeben ist.

Wir nehmen also an, dass Ihr Provider Transaktionen unterstützt. Sie können neue Transaktionen mit der **BeginTrans**-Methode des **Connection**-Objekts starten, eine Transaktion mit **CommitTrans** übergeben und mit **RollbackTrans** Transaktionen rückgängig machen. Nach Aufrufen der **BeginTrans**-Methode werden Änderungen nicht länger sofort durch den Provider übernommen, sondern erst dann, wenn **CommitTrans** oder **RollbackTrans** zum Beenden der Transaktion aufgerufen werden.

Rufen Sie sich den Geltungsbereich von Transaktionen ins Gedächtnis. Wenn Sie für eine Verbindung mehrere Abfragen ausführen, werden sämtliche Operationen im Geltungsbereich einer Transaktion ausgeführt. Öffnen Sie (oder ADO) eine zusätzliche Verbindung, bleibt der zugehörige Transaktionsbereich von anderen Transaktionen für weitere Verbindungen unberührt.

Bei Providern, die verschachtelte Transaktionen unterstützen, wird durch das Aufrufen der **BeginTrans**-Methode innerhalb einer geöffneten Transaktion eine neue, geschachtelte Transaktion gestartet. Der Rückgabewert gibt den Grad der Verschachtelung an. Der Wert 1 gibt an, dass eine Transaktion der obersten Ebene geöffnet wurde (d.h., die Transaktion ist nicht innerhalb anderer Transaktionen verschachtelt), der Wert 2 deutet darauf hin, dass Sie eine Transaktion zweiter Ebene geöffnet haben (eine Transaktion, die sich innerhalb einer Transaktion erster Ebene befindet) usw. Das Aufrufen von **CommitTrans** oder **RollbackTrans** wirkt sich nur auf die zuletzt geöffnete Transaktion aus; Sie müssen die aktuelle Transaktion schließen oder rückgängig machen, bevor Sie eine Transaktion höherer Ebene auflösen können.

In Abhängigkeit der **Attributes**-Eigenschaft des **Connection**-Objekts wird durch die Methoden **CommitTrans** oder **RollbackTrans** möglicherweise automatisch eine neue Transaktion gestartet. Ist die **Attributes**-Eigenschaft auf **adXactCommitRetaining** gesetzt, startet der Provider nach einem **CommitTrans**-Aufruf automatisch eine neue Transaktion. Ist die **Attributes**-Eigenschaft hingegen auf **adXactAbortRetaining** gesetzt, startet der Provider nach einem **RollbackTrans**-Aufruf automatisch eine neue Transaktion.

4.3.1 Ereignisse für das »Connection«-Objekt

ADO 2.0 implementiert einen umfassenden Ereignissatz. Dies brachte ADO im Konkurrenzkampf mit RDO, mit dem der erste (wenn auch eingeschränkte) Ereignissatz für den Datenzugriff implementiert worden war, wieder nach vorn. Unglücklicherweise wurden diese Ereignisbehandlungsroutinen in ADO 2.1 neu definiert – das ist die schlechte Nachricht, wenn Sie bisher die ADO 2.0-Ereignisse verwendet haben. Die gute Nachricht ist, dass in ADO 2.5 keine erneute Überarbeitung stattgefunden hat. Grundsätzlich geben die neuen Ereignisbehandlungsroutinen ein Recordset im ADO 2.1-Format zurück (und das gleiche Format wurde auch in ADO 2.5 verwendet). Dieses zerschoss die Ereignisbehandlungsroutinen in vorhandenen ADO 2.0-Anwendungen, die ein Recordset im 2.0-Format erwarteten. Zur Beseitigung dieses Problems müssen Sie vorhandene ADO 2.0-Ereignisbehandlungsroutinen umcodieren, nachdem Sie Ihren Projektverweis auf ADO 2.1 oder 2.5 eingestellt haben. Diese Aufgabe sollte nicht mehr als ein Ausschneiden und Einfügen umfassen.

Tipp Beim Debuggen von Ereignisbehandlungsroutinen sollten Sie daran denken, dass Visual Basic dazu neigt, Ereignisse zu »verlieren«. Dies bedeutet, dass gelegentlich Ereignisse ausgelöst werden, wenn der Code an einem Haltepunkt oder Dialogfeld gestoppt wird (beispielsweise **MsgBox**). Falls diese Situation eintreten sollte, können die Ereignisse verworfen werden. Ein Bug? Ein Bug.

In einigen Fällen stellt das Ausschneiden und Einfügen der Ereignisbehandlungsroutinen in die neuen Musteranweisungen jedoch keine Option dar. Angenommen, Ihr Code verwendet ADODC, den Datenformular-Assistent oder eines der Visual Database Tools wie etwa den Datenumgebungs-Designer, und Sie setzen Ihren Projektverweis auf ADO 2.1 oder 2.5. In diesem Fall müssen drastischere Korrekturmaßnahmen getroffen werden, da sämtliche dieser Tools speziell für die (alleinige) Unterstützung von ADO 2.0-**Recordset**-Objekten geschrieben wurden. Zur Beseitigung dieses Problems müssen Sie Ihren Quellcode bearbeiten. Hierbei muss an jeder Stelle, an der Ihre Ereignisbehandlungsroutine (die beispielsweise durch den Datenumgebungs-Designer generiert wird) auf den zugehörigen **prototype** der Ereignisbehandlungsroutine trifft, der Code – ausdrücklich – auf das ADO 2.0-Recordset verweisen. Beachten Sie z. B., wie ich in der folgenden **prototype**-Anweisung für die Ereignisbehandlungsroutine das **ADODB.Recordset** durch das **ADODB.Recordset20** ersetzt habe:

```
Private Sub Connection1_WillExecute(Source As String, _
   CursorType As ADODB.CursorTypeEnum, _
   LockType As ADODB.LockTypeEnum, Options As Long, _
```

```
    adStatus As ADODB.EventStatusEnum, _
    ByVal pCommand As ADODB.Command, ByVal pRecordset As ADODB.Recordset20, _
    ByVal pConnection As ADODB.Connection)
End Sub
```

Die ADO 2.1- und 2.5-Typbibliotheken umfassen sowohl Definitionen für die älteren als auch für die neueren **Recordset**-Objekte. Denken Sie jedoch daran, dass die neueren Recordset 2.1-Eigenschaften, -Methoden und -Optionen in der 2.0-Version des Recordsets nicht zur Verfügung stehen.

Wenn Sie Ereignisbehandlungsroutinen für ein ADO-Objekt offen legen möchten, müssen Sie die Objekte mit dem Operanden **WithEvents** deklarieren. Der folgende Code beispielsweise weist Visual Basic an, sämtliche ADODB-Verbindungsereignisse für die weiteren Objekte und Ereignisse im Codefenster einzuschließen.

Gehen wir ein Codebeispiel durch, mit dem die Verwendung verschiedener dieser Ereignisse veranschaulicht wird:

```
Option Explicit
Dim WithEvents cn As ADODB.Connection
Dim er As ADODB.Error
Dim strMsg As String
```

Dieser erste Abschnitt richtet das **cn**-Objekt (**Connection**-Objekt) mit Hilfe der **WithEvents**-Syntax zur Offenlegung der Ereignisbehandlungsroutinen ein.

```
Private Sub Form_Load()
    Set cn = New Connection
    cn.Open "dsn=localserver", "admin", "pw", adAsyncConnect
End Sub
```

Das **Form_Load**-Ereignis richtet das **Connection**-Objekt ein und weist ADO an, mit dem Öffnen der Verbindung zu beginnen. Wir gelangen zu **End Sub**, d.h. das anfängliche Formular wird gezeichnet. Die **Command**-Schaltfläche ist weiterhin deaktiviert – wenn wir diese also später nicht aktivieren, ist eine Ausführung der Abfrage unmöglich.

```
Private Sub cn_ConnectComplete(ByVal pError As ADODB.Error, _
    adStatus As ADODB.EventStatusEnum, ByVal pConnection As ADODB.Connection)
    If adStatus = adStatusOK Then
        cmdTestPrint.Enabled = True
        MsgBox strMsg, vbInformation, "Connect Complete"
    Else
        MsgBox "Could not connect." & pError.Description
    End If
```

```
    strMsg = ""
End Sub
```

Die Verbindungsherstellung wird beendet, und das Ereignis **ConnectComplete** wird ausgelöst. Wir prüfen, ob die Verbindung erfolgreich hergestellt wurde. Ist dies der Fall **(adStatus = adStatusOK),** aktivieren wir das **Command**-Schaltflächensteuerelement. Andernfalls zeigen wir ein **MsgBox**-Dialogfeld an und unternehmen nichts.

```
Private Sub cmdTestPrint_Click()
    cn.Execute "Execute TestPrint 'Dies ist eine Testmeldung'" , , adCmdText
End Sub
```

Jetzt kann die gespeicherte Prozedur getestet werden. In diesem Fall werden verschiedene TSQL-**Print**-Anweisungen ausgeführt. Sie landen in der **Errors**-Auflistung des **Connection**-Objekts. Wir hätten diese Anweisung auch asynchron ausführen können, aber hierzu bestand keine Notwendigkeit.

```
Private Sub cn_ExecuteComplete(ByVal RecordsAffected As Long, ByVal pError As
ADODB.Error, adStatus As ADODB.EventStatusEnum, ByVal pCommand As ADODB.Com-
mand, ByVal pRecordset As ADODB.Recordset, ByVal pConnection As ADODB.Connec-
tion)
    DumpErrors
    MsgBox strMsg, vbInformation, "Execute Complete"
    strMsg = ""
End Sub
```

Nach Beendigung der Abfrage wird das **ExecuteComplete**-Ereignis ausgelöst, wie für **alle** Operationen dieser Verbindung (**pConnection**). Beachten Sie, dass hier ein **Command**-Objekt offen gelegt wird, obwohl es nicht ausdrücklich erstellt wurde. Außerdem ist ein Zeiger auf ein Recordset vorhanden, dessen **State**-Eigenschaft den Wert **adStateExecuting** aufweist.

```
Private Sub cn_InfoMessage(ByVal pError As ADODB.Error,  _
    adStatus As ADODB.EventStatusEnum, ByVal pConnection As ADODB.Connection)
    strMsg = "Info message: " & pError.Description
    DumpErrors
End Sub
```

Beim Öffnen des **Connection**-Objekts wird das **InfoMessage**-Ereignis ausgelöst, um uns mitzuteilen, dass die Standarddatenbank auf **Biblio** geändert wurde – das geschah jedoch zweimal. Warum? BHOM[8]. An diesem Punkt sollte keine weitere Verbindung geöffnet werden.

8. BHOM: ein alter Ausdruck. »Beats the hell out of me« oder »weiß nicht«.

Der folgende Code erstellt ein Dump der **Errors**-Auflistung in einer einzelnen Zeichenfolge, die zum Debuggen der Operation eingesetzt werden kann – keine Zeichenfolge, die einem Benutzer angezeigt werden könnte.

```
Sub DumpErrors()
   For Each er In cn.Errors
      strMsg = strMsg & "Errors:" & er.Description & vbCrLf
   Next er
End Sub
```

4.3.2 Sind Ereignisse in der mittleren Schicht sinnvoll?

Einige Entwickler haben den Sinn und den praktischen Nutzen hinterfragt, den das Implementieren asynchroner Operationen in der mittleren Schicht mit sich bringt. Sicher, es ist möglich. Die Ereignisbehandlungsroutinen für Komponenten der mittleren Schicht können Ihre Anwendung via DCOM benachrichtigen, wenn Sie möchten, aber dieser Ansatz impliziert eine Statusverwaltung in der mittleren Schicht – keineswegs eine empfohlene Vorgehensweise. Es ist also möglich, Sie sollten es aber nicht versuchen. Es macht Ihre Anwendungen komplizierter als nötig und öffnet unregelmäßigen Abstürzen die Tür.

4.3.3 Das »InfoMessage«-Ereignis

Einige Provider, beispielsweise SQL Server, geben so genannte Informationsmeldungen zurück, wenn bestimmte Operationen auf dem Server stattfinden. Wenn sich z. B. die aktuelle Datenbank ändert, informiert SQL Server Sie über diese Operation, indem eine Nachricht an den Provider gesendet wird. Der Provider verwirft die Nachricht, sofern keine **InfoMessage**-Ereignisbehandlungsroutine vorhanden ist. In den meisten Fällen sind diese Nachrichten unwichtig.

4.3.4 Fehlerbehandlung

Herauszufinden, was bei der Verbindungsherstellung schiefgegangen sein könnte, kann eine große Herausforderung darstellen. Die meisten Fehler scheinen unter drei Fehlernummern zu fallen, mit jeweils abweichenden Beschreibungen. Ja, dies bedeutet, dass Sie die Zeichenfolgen nach wichtigen Informationen durchsuchen und entsprechend reagieren müssen. Die empfohlenen Vorgehensweisen diktieren, dass Sie diese Meldungen nicht einfach an die Endbenutzer weiterleiten. Von den Endbenutzern darf am wenigstens erwartet werden, dass sie in der Lage sind, das Problem zu lösen.

- **Error.Number – 2147467259 (0x80004005)** wird bei verbindungsbezogenen Fehlern ausgelöst. Hierzu zählen falsche Protokolle, fehlende (oder falsch geschriebene) Server(namen), Netzwerkberechtigungsprobleme – praktisch alles, was ADO daran hindert, eine Verbindung zum Remoteserver herzustellen. Die Beschreibung enthält häufig (aber nicht immer) nützliche Informationen zur Fehlerursache.
- **Error.Number – 2147217843 (0x80040E4D)** wird bei anmeldungsbezogenen Fehlern ausgelöst. Sie haben vielleicht die richtige Benutzer-ID, aber den falschen Server angegeben und erhalten diesen Fehler, wenn es sich bei dem genannten Server um einen Arbeitsserver handelt.
- **Error.Number – 2147221020 (0x800401E4)** wird ausgelöst, wenn die Syntax der Verbindungszeichenfolge Fehler aufweist. Dieser Fehler ist eher selten und sollte nach dem Debuggen einer Anwendung nicht mehr auftreten. Wird der Fehler bei einer kompilierten Anwendung ausgegeben, sind DSN oder UDL wahrscheinlich fehlerhaft.

Tipp Lassen Sie nicht den Benutzer Ihr Programm debuggen. Fangen Sie Fehler auf, und ermitteln Sie die Fehlerursachen.

Die zurückgegebene Fehlerbeschreibung (**Error.Description**) weist als Präfix den Quellstack des Fehlers auf. Jede Schicht, die eine Fehlerbehandlung durchführt, fügt der Beschreibungszeichenfolge einen zusätzlichen, in Klammern eingeschlossenen Ausdruck hinzu. Daher kann die Fehlermeldung folgendermaßen aussehen:

```
[Microsoft][ODBC SQL Server Driver][SQL Server]Login failed for user 'Fred'.
```

Tipp Wenn Sie **ConnectionWrite(GetOverlappedResult)**-Fehler erhalten, sollten Sie auf TCP/IP wechseln (weg von den Named Pipes).

5 ADO-Befehlsstrategien

Wenn wir zu Kapitel 6 kommen, in dem es vor allem um Recordsets geht, werden wir viele verschiedene Methoden zur Abfrageausführung besprechen – von denen viele die Verwendung des **Command**-Objekts erfordern. Der einzig wichtige Fall, in dem das **Command**-Objekt erforderlich ist, ist der, wenn Parameter erfasst werden müssen, die von gespeicherten Prozeduren **zurückgegeben** werden. ADO ist sehr leistungsstark, wenn es um die Handhabung gespeicherter Prozeduren geht. Wenn Sie keine OUTPUT-Parameter zurückgeben und sich keine Gedanken über den Rückgabestatus von gespeicherten Prozeduren machen müssen, dann muss kein **Command**-Objekt erstellt werden.

Es gibt jedoch viele Vorteile bei der Verwendung von **Command**-Objekten. Im vorliegenden Kapitel werden wir untersuchen, wie das **Command**-Objekt der neuen SQL Server- und MDAC-Technologie zur effizienteren Ausführung sämtlicher Arten von Abfragen verhilft. Dies bedeutet, dass bei der Ausführung von Aktionsabfragen oder Abfragen, bei denen Rowsets zurückgegeben werden, das Einrichten eines **Command**-Objekts zur besseren Verwaltung der Abfrage und der zugehörigen Parameter beitragen kann. Nach der Erstellung eines **Command**-Objekts müssen Sie sich nicht länger um das Setzen von einfachen Anführungszeichen um Zeichenfolgen oder darum kümmern, wie eingebettete Apostrophe gehandhabt werden. Es ist also nicht erforderlich, alle irischen Nachnamen aus Ihrer Datenbank zu entfernen – beispielsweise O'Malley oder O'Brien. In diesem Kapitel wird außerdem erläutert, wie Sie ermitteln können, was bei einer Abfrage geschieht. Ein großer Teil dieses Kapitels befasst sich mit den SQL Server Profiler-Protokollen, mit denen genau aufgezeigt wird, zu welch unnatürlichen Handlungen SQL Server genötigt wird.

Bei der Ausführung einer SQL-Abfrage stellt das ADO-**Command**-Objekt häufig das geeignetste Objekt dar. Wie wir sehen werden, ist es aber nicht **immer** die beste Wahl. Glücklicherweise gibt es – aufgrund der Flexibilität von ADO – Alternativen, zu denen wir noch in Kapitel 6 kommen, wenn das **Recordset**-Objekt besprochen wird. Eine Sache, die Sie vielleicht noch nicht wissen (da »unterdokumentiert«): Alle ADO-**Command**-Objekte erscheinen als Methoden der verknüpften **Connection**-Objekte. Diese innovative Technik (naja, sie haben sie von RDO geklaut) ermöglicht Ihnen das Codieren des **Command**-Objekts nach Name, gefolgt von den zugehörigen Parametern, gefolgt vom Recordset mit dem Rowset. Cool. Mit diesem Thema werden wir uns später beschäftigen.

5.1 Das Innenleben des »Command«-Objekts

Der größte Vorteil des **Command**-Objekts ist seine Leistung. Es macht nicht nur die Abfrageausführung effizienter, es bringt auch **Sie** dazu, effizienter zu arbeiten. Bei Verwendung des **Command**-Objekts kann der Zeitaufwand für das Programmieren, Debuggen, Testen und Bereitstellen komplexer, parameterbasierter Abfragen erheblich reduziert werden – das Ausführen von gespeicherten Prozeduren mit Hilfe von Abfragen eingeschlossen. Beim Zugriff über SQL Server 7.0 wurden beispielsweise die ODBC- und OLE DB-Provider so verbessert, dass auf die neue gespeicherte Systemprozedur **sp_executesql** zugegriffen werden kann. Zu diesem Thema steht in der Onlinedokumentation von SQL Server[1] eine Menge Material zur Verfügung, das an dieser Stelle nur kurz zusammengefasst wird.

Im Grunde wird das **Command**-Objekt über das **CommandType**-Argument gesteuert, das ADO anweist, wie Ihre Abfrage an den Datenprovider übertragen werden soll. Angenommen, Sie möchten eine parameterbasierte Ad-hoc-Abfrage[2] ausführen:

```
Select author, au_id, year_born from authors where year_born = ?
```

Sie haben dort einen Parametermarker (?) platziert, wo ADO den Parameter einfügen soll, daher kann die Abfrage von ADO ausgeführt werden. Für diese Abfrage erstellt ADO eine SQL-Anweisung, die folgendermaßen aussieht:

```
sp_executesql N'Select author, au_id, year_born from authors where year_born
= @P1', N'@P1 int', 1947
```[3]

Die gespeicherte Systemprozedur **sp_executesql** wurde erstmals in Microsoft SQL Server 7.0 vorgestellt. Die MDAC-Entwickler möchten, dass wir diese Prozedur anstelle der EXECUTE-Anweisung zur Ausführung einer Abfragezeichenfolge verwenden. Die Unterstützung der Parametersubstitution macht die Prozedur **sp_executesql** nicht nur vielseitiger als EXECUTE, sondern auch effizienter, da Ausführungspläne generiert werden, die von SQL Server wieder verwendet werden können.

1. Onlinedokumentation: Der Satz Hilfethemen sowie die Beispiele, die zum Lieferumfang von SQL Server gehören – anstelle einer gedruckten Dokumentation. Ein Großteil der Informationen (wenn nicht sogar alle) stehen auch über ein MSDN-Abonnement oder online zur Verfügung.
2. Eine Ad-hoc-Abfrage ist einfach eine hartcodierte SQL-Abfrage oder -Aktion. Das Verwenden dieser Abfragen ist keine gute Idee, wenn Sie statt dessen eine gespeicherte Prozedur einsetzen können, aber viele Entwickler verlassen sich auf diese Abfragen, zumindest anfänglich.
3. Beachten Sie das »N«-Präfix im generierten Code (N'Select au...). Dieses N (in Großbuchstaben) gibt an, dass die folgende Zeichenfolge im Unicode-Format vorliegt. Unicode-Daten werden mit 2 Byte pro Zeichen gespeichert, im Gegensatz zu Zeichendaten, bei denen 1 Byte pro Zeichen gespeichert wird. Weitere Informationen finden Sie über den Index der SQL Server-Onlinedokumentation unter dem Stichwort »Konstanten«.

ADO und der SQL Server-Datenprovider haben eine weitere (proprietäre) Schnittstelle zur Handhabung serverseitiger Cursor implementiert. Diese werden als gespeicherte **sp_cursor**-Prozeduren auf Systemebene implementiert, die verschiedene weitere Datenoperationen öffnen, schließen und ausführen. Bei Verwendung der standardmäßigen serverseitigen **CursorLocation**-Einstellung werden Sie feststellen, dass viele Abfragen mit Hilfe dieser gespeicherten Prozeduren ausgeführt werden.

Tipp Bei der Arbeit mit SQL Server ist es wichtig, dass Sie den Profiler (oder die Ablaufverfolgung von SQL Server 6.5) aktivieren und aktiviert lassen, während Sie Ihren Code bearbeiten. Auf diese Weise erhalten Sie genaue Informationen darüber, was an SQL Server gesendet wird. Deaktivieren Sie den Profiler, wenn Sie mit der Codeleistung zufrieden sind.

5.1.1 Substituieren von Parameterwerten

Die gespeicherten Prozeduren **sp_executesql** und **sp_cursor** unterstützen die Substitution von Parameterwerten für beliebige in der Transact-SQL-Zeichenfolge spezifizierte Parameter – im Gegensatz zur (veralteten) TSQL-EXECUTE-Anweisung. Die durch **sp_executesql** generierten Transact-SQL-Zeichenfolgen sind den ursprünglichen SQL-Abfragen ähnlicher als die über die EXECUTE-Anweisung generierten Zeichenfolgen, daher stehen die Chancen besser, dass der Abfrageoptimierer von SQL Server übereinstimmende Transact-SQL-Anweisungen von **sp_executesql** aus den Ausführungsplänen zuvor ausgeführter Anweisungen ermittelt. Auf diese Weise müssen sehr viel weniger neue Ausführungspläne für die Ausführung der einzelnen Parameterabfragen kompiliert werden. Das ist gut.

Bei Verwendung der TSQL-EXECUTE-Anweisung müssen sämtliche Parameterwerte in Zeichen oder Unicode-Zeichen konvertiert und in die Transact-SQL-Zeichenfolge integriert werden, wie im folgenden Codebeispiel gezeigt wird:

```
DECLARE @IntVariable INT
DECLARE @SQLString NVARCHAR(500)
/* Erstellen und Ausführen einer Zeichenfolge mit einem Parameterwert. */
SET @IntVariable = 35
SET @SQLString = N'SELECT * FROM pubs.dbo.employee WHERE job_lvl = ' +
CAST(@IntVariable AS NVARCHAR(10))
EXEC(@SQLString)
/* Erstellen und Ausführen einer Zeichenfolge mit einem zweiten Parameterwert. */
SET @IntVariable = 201
SET @SQLString = N'SELECT * FROM pubs.dbo.employee WHERE job_lvl = ' +
```

```
CAST(@IntVariable AS NVARCHAR(10))
EXEC(@SQLString)
```

Wird die Anweisung wiederholt ausgeführt, muss bei jeder Ausführung eine komplett neue Transact-SQL-Zeichenfolge erstellt werden, selbst dann, wenn sich die bereitgestellten Parameterwerte nur geringfügig unterscheiden. Dies führt in verschiedener Hinsicht zu einem zusätzlichen Overhead:

- Die Fähigkeit des Abfrageoptimierers von SQL Server, die neue Transact-SQL-Zeichenfolge mit einem vorhandenen Ausführungsplan abzugleichen, wird durch beständig geänderte Parameterwerte im Text der Zeichenfolge behindert, besonders bei komplexen Transact-SQL-Anweisungen.
- Bei jeder Ausführung muss die gesamte Zeichenfolge neu erstellt werden.
- Es muss für die Parameterwerte (sofern es sich nicht um Zeichenwerte oder Unicode-Werte handelt) bei jeder Ausführung ein Typecast in ein Zeichen- oder Unicode-Format vorgenommen werden.

Im Gegensatz dazu unterstützt **sp_executesql** die von der Transact-SQL-Zeichenfolge unabhängige Festlegung von Parameterwerten:

```
DECLARE @IntVariable INT
DECLARE @SQLString NVARCHAR(500)
DECLARE @ParmDefinition NVARCHAR(500)
/* Einmaliges Erstellen der SQL-Zeichenfolge. */
SET @SQLString =N'SELECT * FROM pubs.dbo.employee WHERE job_lvl = @level'
/* Einmaliges Angeben des Parameterformats. */
SET @ParmDefinition = N'@level tinyint'
/* Ausführen der Zeichenfolge mit dem ersten Parameterwert. */
SET @IntVariable = 35
EXECUTE sp_executesql @SQLString, @ParmDefinition,@level = @IntVariable
/* Ausführen der gleichen Zeichenfolge mit dem zweiten Parameterwert. */
SET @IntVariable = 32
EXECUTE sp_executesql @SQLString, @ParmDefinition,@level = @IntVariable
```

Durch dieses **sp_executesql**-Beispiel wird nicht nur das gleiche Ziel erreicht wie durch das zuvor dargestellte TSQL-EXECUTE-Beispiel, darüber hinaus bietet diese Methode folgende Vorteile:

- Da der tatsächliche Text der Transact-SQL-Anweisung sich zwischen den Ausführungen nicht ändert, sollte der Abfrageoptimierer in der Lage sein, die Transact-SQL-Anweisung der zweiten Ausführung mit dem Ausführungsplan der ersten Ausführung abzugleichen. Die zweite Anweisung muss daher von SQL Server nicht kompiliert werden.

- Die Transact-SQL-Zeichenfolge wird nur einmalig erstellt.
- Der **integer**-Parameter wird im systemeigenen Format angegeben.
- Eine Konvertierung in Unicode ist nicht erforderlich.

Anmerkung Die Objektnamen in der Anweisung müssen vollqualifiziert sein, damit SQL Server den Ausführungsplan wieder verwenden kann.

5.1.2 Wiederverwenden von Ausführungsplänen

In früheren Versionen von SQL Server bestand die einzige Möglichkeit zur Wiederverwendung von Ausführungsplänen darin, die Transact-SQL-Anweisungen als gespeicherte Prozedur zu definieren und die gespeicherte Prozedur anschließend über eine Anwendung auszuführen. Die Prozedur **sp_executesql** kann anstelle von gespeicherten Prozeduren eingesetzt werden, wenn eine Transact-SQL-Anweisung mehrfach ausgeführt wird – besonders dann, wenn lediglich die an die Transact-SQL-Anweisung übergebenen Parameterwerte anders lauten. Da die Transact-SQL-Anweisungen selbst konstant sind und sich nur die Parameterwerte ändern, kann der Abfrageoptimierer von SQL Server den Ausführungsplan der ersten Ausführung wahrscheinlich wieder verwenden. Vorhandene, in SQL Server 7.0 portierte ODBC-Anwendungen profitieren von den Leistungsvorteilen, ohne dass eine Umarbeitung erforderlich ist. Weitere Informationen finden Sie in der SQL Server-Onlinedokumentation unter dem Stichwort »Using Statement Parameters«.

Der Microsoft OLE DB-Provider für SQL Server verwendet **sp_executesql** auch zur direkten Ausführung von Anweisungen ohne feste Parameter. Anwendungen, die OLE DB oder ADO einsetzen, nutzen die Vorteile von **sp_executesql**, ohne dass sie umgeschrieben werden müssen.

Für die Ausführung von Ad-hoc-Abfragen, die keine Parameter bzw. lediglich Verweise auf parameterlose gespeicherte Prozeduren aufweisen, setzt das ADO-**Command**-Objekt die Prozedur **sp_executesql** nicht ein.

5.2 Erstellen von »Command«-Objekten

Das Erstellen eines **Command**-Objekts erfordert etwas Zeit – sowohl CPU- als auch Entwicklungszeit. Betrachten Sie jedoch die Alternativen. In früheren Tagen war das Erstellen einer Abfrage wesentlich schwieriger als heute, da ADO zur Verfügung steht. Okay, keine »Ich habe noch Lochkartenmaschinen ohne Bänder bedient ...« -Geschichten, aber der Code zum Erstellen des **Command**-Objekts ist sehr viel einfacher zu generieren, zu verstehen und zu unterstützen als dies mit der

ODBC-API oder gar RDO (Remote Data Objects) je zu hoffen gewagt werden konnte. Mittlerweile gibt es sogar Assistenten für diese Aufgaben, kann es da noch einfacher werden?[4]

Tipp Wenn Sie ein **Command**-Objekt erstellen, sollte dies nur **einmal** geschehen. Erstellen Sie so viele Objekte wie Sie möchten, aber erstellen Sie keine Objekte, um anschließend – je nach Anforderung – deren Eigenschaften zu ändern (die Parameter natürlich ausgeschlossen). In früheren Versionen führte ADO zusätzliche Serverabfragen aus, um zu ermitteln, wie eine Abfrage ausgeführt werden soll. Meine Tests haben gezeigt, dass dies in ADO 2.5 nicht länger der Fall ist – jedenfalls nicht immer. Die gesamte Setupphase scheint vollständig auf dem Client durchgeführt zu werden. Derartiger Overhead muss nicht mehr als einmal toleriert werden.

Wie also erstellen Sie ein **Command**-Objekt? Ganz einfach:

1. Deklarieren Sie Ihre **Command**-Objekte in einem Bereich, der für alle Routinen sichtbar ist, die auf die Objekte zugreifen müssen. Erstellen Sie daher bei der Entwicklung von Client/Server-Anwendungen das **Command**-Objekt auf Modul- oder Formularebene. In der mittleren Schicht (Microsoft Transaction Server/COM+) muss bei der Erstellung eines **Command**-Objekts in der Methodenprozedur so verfahren werden. Der Grund hierfür ist, dass **Command**-Objekte apartmentübergreifend von verschiedenen Elementen gemeinsam genutzt werden können, und das Verwenden von globalen oder Klassenbereichsvariablen in Microsoft Transaction Server funktioniert einfach nicht besonders gut – um es dezent auszudrücken.
2. Benennen Sie Ihr **Command**-Objekt so, dass es als Methode des **Connection**-Objekts ausgeführt werden kann. Obwohl dieser Schritt optional ist, hilft Ihnen dies zu einem späteren Zeitpunkt. Verwenden Sie zur Benennung des **Command**-Objekts eine Zeichenfolge.
3. Stellen Sie die **CommandText**-Eigenschaft auf die SQL-Anweisung in der Abfrage, auf eine gespeicherte Prozedur, Tabelle oder Sicht ein. ADO liegt mit seinen Vorschlägen leider meistens falsch.
4. Setzen Sie die **CommandType**-Eigenschaft so, dass die Art und Weise widergespiegelt wird, mit der die **CommandText**-Zeichenfolge behandelt wird. Stan-

4. Ich habe gemeinsam mit den Developer Days-Leuten (jährlich stattfindende Microsoft-Entwicklerkonferenz) einen Assistenten optimiert, der auf der diesjährigen Konferenz übergeben wurde. Dieser Assistent generiert »korrekten« Quellcode für das Erstellen von **Command**-Objekten und der **Properties**-Auflistung, ohne dass Sie einen Finger rühren müssen – fast keinen jedenfalls. Den Assistenten finden Sie auf der Begleit-CD.

dardmäßig setzt ADO diese Eigenschaft auf **adCmdUnknown**, lassen Sie den Wert also besser nicht von ADO festlegen.

5. Stellen Sie die **ActiveConnection**-Eigenschaft so ein, dass auf das geeignete ADO-**Connection**-Objekt verwiesen wird. Sie können die **Parameters.Refresh**-Methode bzw. die Abfrage vorher nicht ausführen. Nach dem Setzen der Eigenschaft wird der benannte Befehl zu einer Methode der angegebenen Verbindung.
6. Erwartet die Abfrage einen oder mehrere Parameter, müssen Sie entscheiden, ob Sie die **Refresh**-Methode zum Erstellen der **Parameters-**Auflistung durch ADO verwenden oder ob Sie diese selbst erstellen, indem Sie **Parameters.Append** oder **Command.CreateParameter** einsetzen.

5.2.1 Festlegen der ADO-Befehlseigenschaften

Das Festlegen der ADO-Eigenschaften hat viel damit zu tun, die Erwartungen von ADO zu kennen. Wenn Sie eine Zahl angeben müssen, sollten Sie sicherstellen, dass sich diese im gültigen Bereich befindet. Vermeiden Sie figurative Konstanten – verwenden Sie statt dessen durch die Typbibliothek definierte Konstanten. So ist der Code besser lesbar und später einfacher zu verwalten. Wenn Sie eine Eigenschaft einer **TextBox** oder eines anderen Steuerelements festlegen, müssen Sie daran denken, die richtige Eigenschaft explizit zu referenzieren. Verlassen Sie sich nicht darauf, dass die Standardeigenschaften funktionieren – manchmal tun sie dies, oft genug aber auch nicht. Wenn Sie beispielsweise einen Befehlsparameter festlegen, sollten Sie folgendermaßen vorgehen:

```
Cmd(eParm.NameWanted) = txtNameWanted.Text
```

Dieser Code beruht auf der Definition eines **Enum**- und eines **TextBox**-Steuerelements. Das Auslassen des **.Text**-Eigenschaftenbezeichners kann, sagen wir, ungeahnte Folgen haben. Sagen Sie bloß nicht, ich hätte Sie nicht gewarnt.

Die »Name«-Eigenschaft

Wenn Sie die (coole) Technik »**Command**-Objekt als **Connection**-Methode« zum Ausführen des **Command**-Objekts verwenden möchten, **müssen** Sie dieses benennen. Dies stellt in jedem Fall eine gute Idee dar. Bei der Ausführung von gespeicherten Prozeduren **muss** der Name des Objekts mit der gespeicherten Prozedur übereinstimmen. Andernfalls können Sie Ihr **Command**-Objekt auch nach exotischen Pflanzen benennen, wenn Ihnen danach ist. Denken Sie nur daran, den Namen frühzeitig festzulegen – vor dem Einstellen der **ActiveConnection**-Eigenschaft. Und vergessen Sie nicht, eine **String**-Konstante oder Variable zum Benennen des Befehls zu verwenden.

Tipp Wenn Sie statt einer **String**-Konstante oder einer deklarierten Variablen einen nicht angeführten Wert übergeben, setzt Visual Basic voraus, dass es sich um den Namen einer **Variant**-Variablen handelt, die zur Laufzeit einen Wert erhält, sofern Sie nicht **Option Explicit** aktiviert haben (was Sie tun sollten). In diesem Fall erhalten Sie zur Kompilierungszeit eine Warnung, dass die Variable nicht definiert ist.

Die »CommandText«-Eigenschaft

Diese Eigenschaft teilt ADO und dem Datenprovider mit, was getan werden muss. Es kann die Frage sein, die Sie stellen möchten, es kann sich einfach um den Namen einer Tabelle, einer gespeicherten Prozedur oder sogar um eine URL handeln, an der sich die gesuchten Daten befinden.

Üblicherweise ist **CommandText** eine SQL-Anweisung, z. B. eine SELECT-Anweisung, aber es kann sich ebenso um einen anderen Typ Befehlsanweisung handeln, der durch den Provider erkannt wird, etwa ein gespeicherter Prozeduraufruf. Denken Sie daran, SQL in dem SQL-Dialekt zu programmieren, den der Datenprovider versteht. Wenn Sie also eine Verbindung zu Oracle herstellen, können (sollten) Sie Oracle-SQL-Erweiterungen verwenden, genauso wie Sie TSQL-Erweiterungen einsetzen, wenn Sie SQL Server-Abfragen durchführen.

Je nach **CommandType**-Eigenschafteneinstellung verändert ADO möglicherweise die **CommandText**-Eigenschaft. Sie können die **CommandText**-Eigenschaft jederzeit lesen, um den tatsächlichen Befehlstext anzuzeigen, den ADO während der Ausführung verwendet.

Sie können auch die **CommandText**-Eigenschaft verwenden, um eine relative URL einzustellen oder zurückzugeben, die eine Ressource bezeichnet, beispielsweise eine Datei oder ein Verzeichnis. Die Ressource wird relativ zu einem Standort angegeben, der explizit durch eine absolute URL oder implizit durch ein geöffnetes **Connection**-Objekt spezifiziert wird.

»CommandText« und Skalierbarkeit Aber warten Sie. Da eine Menge der Probleme im Hinblick auf die Skalierbarkeit durch die **CommandText**-Eigenschaft verursacht wird, sollten wir uns kurz diesem Thema zuwenden. Vergessen Sie nicht, dass die in der **CommandText**-Eigenschaft angegebene Abfrage lediglich eine Anforderung von Diensten des Datenproviders darstellt. Es liegt in der Verantwortlichkeit des Providers, die physischen E/A-Operationen zum Ausführen dieser Anforderung durchzuführen, unabhängig davon, wie falsch sie erscheinen mögen.

Angenommen, Sie rufen bei der Pizzeria »La Toskana« in Siegen an und bestellen tausend Dönerpizzen und zweimal Pizza vegetarisch (Sie möchten den Sport-

freunden Siegen und deren Fans eine Freude machen). Der Manager würde Sie sicher zurückrufen, um den Rahmen Ihrer Kreditkarte, Ihre Vertrauenswürdigkeit und Ihren geistigen Zustand zu prüfen, bevor er die Bestellung bearbeitet. Ein ADO-Datenprovider ruft nicht zurück und fragt, ob Sie ihn auf den Arm nehmen wollen, wenn Sie tausend oder zehntausend oder zehn Millionen Zeilen einer Datenbank anfordern. Er beginnt einfach damit, die Daten abzurufen und sie an Sie weiterzuleiten. Bei deren Empfang fängt ADO pflichtbewusst an, die Daten im RAM zwischenzuspeichern und anschließend auf die Festplatte zu spoolen – bis RAM und Festplatte überlaufen und die Daten hinten aus dem Computer fallen. Nein, ADO hat keine **Hab' Nachsicht mit mir, ich bin etwas dumm**-Eigenschaft – es wird vorausgesetzt, dass Sie wissen, was Sie tun.

Intelligente Abfragenerstellung oder Erstellen intelligenter Abfragen

Nach der Verbindungsherstellung muss die Anfrage an die Datenbankengine übermittelt werden. Dies bedeutet, dass Sie eine Abfrage senden müssen – üblicherweise eine SELECT-Anweisung zur Rückgabe von Zeilen oder eine Aktionsabfrage zur Datenänderung. Die Gesamtleistung kann durch unzureichend entworfene Abfragen stärker beeinflusst werden als durch alle weiteren Leistungsfaktoren zusammen. Mit anderen Worten, wenn Sie die Datenbankengine anweisen, eine Aufgabe auszuführen, die fünf, fünfzig oder fünfzigtausend Sekunden dauert, kann nichts auf der Welt die Clientseite der Abfrage dazu bringen, dass die Zeilen auch nur eine Sekunde eher angezeigt werden. Darüber hinaus kann es bei unzulänglich entworfenen Parallelzugriffen zu Dauersperren kommen, die Ihre Anwendung für immer daran hindern, auch nur eine Zeile zurückzugeben.

Es gibt vielfältige Informationen und Hinweise zum Entwerfen effizienter Abfragen, die im Wesentlichen auf die folgenden Richtlinien reduziert werden können:

- Rufen Sie **nur** die Spalten[5] (Felder) ab, die Sie benötigen, nicht mehr. Verwenden Sie daher niemals SELECT *, auch dann nicht, wenn Sie alle (derzeit definierten) Spalten abrufen möchten. Über SELECT * werden möglicherweise Spalten abgerufen, deren Abruf sehr aufwendig ist bzw. die für die auszuführende Aufgabe irrelevant sind. Darüber hinaus kann bei SELECT * die Reihenfolge der Spaltenrückgabe **nicht** garantiert werden. Wenn also ein ambitionierter Systemadministrator die Tabellenspalten nach dem Alphabet sortiert oder einfach eine neue Spalte in die Tabelle einfügt, kann SELECT * eine Anwendung töten (technischer Fachausdruck).

5. Okay, ich sollte für relationale Datenbanken den Begriff »Zeilen und Spalten« verwenden. Die ISAM-Welt verwendet die Begriffe »Datensätze und Felder«. Die Microsoft-Leute, die ADO geschrieben haben, mögen offensichtlich die ISAM-Begriffe, daher Datensätze und Felder. Seufz.

Ein Aspekt ist die Leistung des **Entwicklers**. Wie effizient arbeitet der Programmierer an einer Anwendung, wie viele Fehler entstehen, und wie viele Missverständnisse treten bei der Kommunikation mit anderen Entwicklern auf? SELECT * scheint dieses Problem anzugehen, indem der Server angewiesen wird, einfach alle Spalten eines Resultsets zurückzugeben. Wenn der Anwendung jedoch kein Code zur automatischen Anpassung an Änderungen im zugrunde liegenden Schema hinzugefügt wird, wird die Produktivität der Entwickler nicht gesteigert. Im Gegenteil – Sie bürden denjenigen mehr Arbeit auf, die nach einer Schemaänderung die Fehlerursachen ermitteln müssen.

- Rufen Sie **nur** die Zeilen (Datensätze) ab, die Sie benötigen, nicht mehr. Skalierbare Anwendungen rufen in einem Arbeitsschritt eine ausreichende Zeilenmenge ab. Der Begriff »ausreichend« wird hierbei über Ihren Entwurf definiert, da ebenfalls Kosten entstehen, wenn der Server für einen weiteren Datenabruf erneut kontaktiert werden muss. Mit Ihrem Code muss die nötige Balance zwischen erneuten Abrufvorgängen und dem Anzeigen nicht benötigter Zeilen gefunden werden. Das Abrufen zu vieler Zeilen erhöht außerdem die Menge der Sperren, die durch die Datenbank ausgegeben werden. Dies kann die Skalierbarkeit einer Anwendung angreifen und das Risiko von Deadlocks erhöhen. Verwechseln Sie interaktive, durch den Menschen gesteuerte Anwendungen nicht mit Berichtsanwendungen, die häufig mit weitaus größeren Zeilenmengen arbeiten müssen.
- Entwerfen Sie nur Cursor in einer Anwendung, wenn dies unbedingt erforderlich ist. Bei der Erstellung skalierbarer Anwendungen mit Hilfe von komplexen Abfragen und gespeicherten Prozeduren werden Sie feststellen, dass ADO keine leistungsfähigen Cursor für generierte Rowsets erstellen kann. Die **cursorlosen** Resultsets (Standardverhalten von ADO) haben sich als schneller erstell- und abrufbar erwiesen. Beachten Sie, dass es bei der Arbeit mit ausgefeilten Tabellenbeziehungen in den wenigstens Fällen genügt, einer Basistabelle eine Zeile hinzuzufügen. In vielen Fällen müssen zunächst einer Fremdschlüsseltabelle Zeilen hinzugefügt werden. Dies impliziert, dass einfache Cursoraktualisierungen nicht funktionieren und Sie sich auf gespeicherte Prozeduren oder clientintensivere, transaktionsgesteuerte Operationen verlassen müssen.
- Ziehen Sie für den Datenabruf das Verwenden von **Return Status**-, OUTPUT- oder INPUT-OUTPUT-Parametern anstelle von Recordsets (Cursorn) in Erwägung. Diese sind erheblich schneller als über ADO erstellte Recordsets, die eine einzelne Datenzeile zurückgeben.
- Wenn Sie einen Cursor erstellen **müssen**, erstellen Sie einen rollbaren Cursor **nur** dann, wenn dies absolut erforderlich ist. Rollbare Cursor wirken sich dramatisch auf die Leistung aus, da ADO zusätzlichen Code zum Erstellen von Cur-

sorschlüsselgruppen oder statischen Rowsets im Speicher ausführen muss. Während der Overhead Ihre Anwendung dazu zwingt, einen beträchtlichen einmaligen Zeitverlust bei der Recordseterstellung hinzunehmen, verschlimmert sich die Situation bei Verwendung dynamischer Cursor insofern weiter, da ADO gezwungen ist, beim Durchlaufen der Seiten die Datenbank wiederholt abzufragen.

- Wenn Sie pessimistische Sperren verwenden, sollten Sie im Hinblick auf die Größe des abgerufenen Rowsets vorsichtig sein, da alle Zeilen in einem Rowset (und vielleicht die Seiten, auf denen diese gespeichert werden) so lange gesperrt bleiben, wie der Cursor geöffnet ist – nicht lediglich während der Bearbeitung einer Zeile. Verwenden Sie pessimistische Sperren nur dann, wenn Sie die Nebenwirkungen der Sperren gründlich untersucht haben.
- Lassen Sie beim anfänglichen Abrufen von Zeilen vom Server nicht den **Benutzer** entscheiden, wann (und ob) das gesamte Rowset abgerufen wird. Vermeiden Sie es, die erste Zeile abzurufen und anzuzeigen und dem Benutzer die Möglichkeit zu geben, per Knopfdruck den nächsten Satz Zeilen anzuzeigen. Erarbeiten Sie Strategien, bei denen alle Zeilen in einem Arbeitsschritt abgerufen werden. Ziehen Sie beispielsweise eine Trennung des Recordsets oder das Verwenden der Methoden **GetRows**[6] oder **GetString** in Betracht. Das Verzögern beim Auffüllen führt auch zu einer Verzögerung bei der Aufhebung von Zeilensperren. Obwohl sich dies nicht negativ auf **Ihre** Anwendung auswirkt, blockieren Sie doch andere Anwendungen, die auf dieselben Datenseiten zugreifen möchten. Als allgemeine Regel zur Erreichung höherer Skalierbarkeit sollten Sperren auf Daten, die dem Benutzer angezeigt werden, vermieden werden.
- Führen Sie keine hartcodierten Abfragen aus, wenn Sie eine gespeicherte Prozedur ausführen können. Durch das Vorkompilieren des Abfragecodes in einer gespeicherten Prozedur können Wartezeiten aufgrund von Servervalidierung, Kompilierung und Erstellung eines Ausführungsplans vermieden werden.
- Setzen Sie bei der Ausführung von Ad-hoc-Abfragen (die immer ausgeführt werden, wenn Sie die **CommandText**-Eigenschaft des **Command**-Objekts auf eine Zeichenfolge einstellen, die nicht den Namen einer gespeicherten Prozedur enthält) den Wert der **Prepared**-Eigenschaft nicht auf **True**. Diese Eigenschaft ist meiner Meinung nach nicht funktionsfähig.
- Berücksichtigen Sie die Auswirkungen Ihrer Anwendung auf den Server und andere Benutzer – beschränken Sie sich nicht darauf, mit einer Abfrage lediglich **Ihre** Clientanwendung oder Komponente schneller zu machen. Gelegentlich

6. **GetRows** ist vielleicht keine so gute Idee. Diese Methode erzeugt beim Erstellen der Ausgabestruktur für das **Variant**-Array relativ viel Overhead.

können Sie Clientoperationen durchführen, die zwar zu einer starken Beschleunigung der lokalen Anwendung, gleichzeitig jedoch zu Sperren für andere Benutzer führen oder in anderer Form die Skalierbarkeit beeinträchtigen. Skalierbarkeit und Leistung gehen nicht immer Hand in Hand, besonders dann nicht, wenn viele Benutzer beteiligt sind.

- Überwachen Sie die Auswirkung Ihrer Anwendung auf den Server. Verwenden Sie den SQL Profiler zum Anzeigen der Operationen niedriger Ebene, deren Ausführung über den Code beim Server angefordert wird (siehe Abbildung 4.11). Versuchen Sie, erneute Serverabfragen und die Menge der zurückgegebenen (sinnvollen) Daten auszugleichen. Der Profiler und andere Diagnosetools, die später näher betrachtet werden sollen, zeigen klar, welche Auswirkung Ihre Programme auf das System und auf einander haben – wenn Sie wissen, wie die Dumps interpretiert werden müssen.

Einige dieser Strategien tragen zur Verkürzung der Clientantwortzeiten bei, andere machen das gesamte System schneller. Daher unterstützen einige dieser Vorschläge Sie bei der Erstellung von Anwendungen, durch die die Systemressourcen effektiver genutzt werden – Ressourcen, um die alle Clients konkurrieren. So werden alle Ihre Anwendungen schneller, weisen kürzere Antwortzeiten auf und unterliegen einem weniger großen Risiko, durch das Warten auf Ressourcen gesperrt zu werden.

Leistung – Öffnen eines »Command«-Objekts Wie ich bereits zu einem früheren Zeitpunkt erwähnt habe, kann bei mehrfacher Abfragenausführung durch das Erstellen eines **Command**-Objekts zur Abfrageverwaltung Ausführungs- und Programmierzeit gespart werden – besonders dann, wenn die Abfrage Parameter erfordert. In der mittleren Schicht und auf Webseiten ist es jedoch nicht unüblich, einelementige Abfragen auszuführen und den Vorgang zu beenden – die Erstellung eines **Command**-Objekts zur Leistungsverbesserung ist daher nicht erforderlich. Tatsache ist, dass sich möglicherweise sogar Vorteile hinsichtlich der Leistung ergeben, wenn Sie **kein Command**-Objekt im Code erstellen – insbesondere dann, wenn Sie mit Visual Basic Script auf ASP-Seiten (ActiveX Server Pages) programmieren, wobei jede Visual Basic-Zeile einzeln interpretiert wird. Wenn die Ausführung für SQL Server erfolgt, werden Abfragen, die einen zuvor zwischengespeicherten Ausführungsplan wieder verwenden können, nicht erneut kompiliert.

Berücksichtigen Sie außerdem, wie die **Command**-Objekte Sie bei der Parameterverwaltung unterstützen. Gelegentlich ist es erforderlich, Rückgabewerte oder OUTPUT-Parameter zu erfassen und Sie haben einfach keine andere Wahl. Und da die meisten Abfragen parametergesteuert sind, kann durch das Verwenden von **Command**-Objekten zur einfachen Parameterverwaltung außerdem die Anzahl der ausgeführten Visual Basic-Codezeilen verringert werden.

Die »CommandType«-Eigenschaft

Wenn Sie ADO nicht mitteilen, **wie** Ihr Befehl interpretiert und verarbeitet werden soll, übernimmt diese Entscheidung die ADO-eigene Logik. Es ist besser (viel besser), die **CommandType**-Eigenschaft zu setzen, bevor durch ADO ein Vorschlag dazu gemacht wird, wie die Abfrage interpretiert werden sollte. Auf diese Weise wird bei der internen Verarbeitung (beträchtliche) Zeit eingespart. Eine weitere Leistungsverbesserung kann durch Verwendung der **adExecuteNoRecords**-Option mit **adCmdText** oder **adCmdStoredProc** erzielt werden. Auf diese Weise wird ADO mitgeteilt, dass durch die Abfrage kein Rowset zurückgegeben wird, und daher die Erstellung eines Cursors nicht erforderlich ist.

Der Dokumentation kann entnommen werden, dass bei Festlegung des Wertes **adCmdUnknown** (Standardwert) für die **CommandType**-Eigenschaft eine Leistungsbeeinträchtigung eintreten kann, da ADO Aufrufe an den Provider durchführen muss, um zu ermitteln, ob es sich bei der **CommandType**-Eigenschaft um eine SQL-Anweisung, eine gespeicherte Prozedur oder einen Tabellennamen handelt. Der SQL Profiler gibt keinen Hinweis auf diese DDL-Anforderungen, bei einigen Providern kann dies jedoch erforderlich sein. Stimmt die **CommandType**-Eigenschaft nicht mit dem Befehlstyp in der **CommandText**-Eigenschaft überein, tritt beim Aufrufen der **Execute**-Methode ein Fehler auf. In Tabelle 5.1 werden die gültigen Einstellungen für die **CommandType**-Eigenschaft aufgeführt.

CommandType	Beschreibung
`adCmdUnspecified`	Befehl gibt Abfragetyp nicht an.
`adCmdText`	Wertet Befehlstext als SQL-Befehl oder SQL-**Call**-Anweisung[1] aus.
`adCmdTable`	Wertet Befehlstext als Tabellennamen aus. ADO führt bei Verwendung dieser Option SELECT * FROM <table> aus. Sie sollten genau wissen, wie sich diese Methode auf die Skalierbarkeit auswirkt.
`adCmdStoredProc`	Wertet Befehlstext als den Namen einer gespeicherten Prozedur aus.
`adCmdUnknown`	Standardwert. Gibt an, dass der Befehlstyp der **CommandText**-Eigenschaft nicht bekannt ist. Auf diese Weise wird die Entscheidung von ADO getroffen, bzw. ADO weiß auch anschließend nicht, um welchen Befehlstyp es sich handelt.
`adCmdFile`	Wertet Befehlstext als Dateiname eines dateibasierten Recordsets aus.
`adCmdTableDirect`	Wertet Befehlstext als Name einer Tabelle aus, bei der sämtliche Spalten zurückgegeben werden. Diese Option ist nur für Jet 4.0-Datenbanken und -Provider verfügbar.

Tabelle 5.1 Gültige Einstellungen für die »CommandType«-Eigenschaft

1. Es ist nicht länger erforderlich, gespeicherte Prozeduren anhand der in RDO eingesetzten **Call**-Syntax aufzurufen.

Voreinstellen von Ausführungsoptionen Bei der Erstellung eines **Command**-Objekts kann es sinnvoll sein, eine oder mehrere der Optionen im Vorfeld zu definieren, die sich darauf auswirken, wie ADO die Abfrage ausführt. Sie könnten beispielsweise eine asynchrone Ausführung anfordern, indem Sie **adAsyncExecute** oder eine Option verwenden, mit der angegeben wird, dass über den Befehl keine Zeilen zurückgegeben werden, z. B. **adExecuteNoRecords**. Unglücklicherweise lässt ADO Sie diese Optionen nicht beim Festlegen des Befehlstyps setzen. Laut Dokumentation ist dies möglich, Sie erhalten bei diesem Versuch jedoch einen 3001-Laufzeitfehler. Sie müssen diese Optionen an die **Execute**-Methode übergeben, bis dies behoben ist.

Die »ActiveConnection«-Eigenschaft

Wenn Sie Ihr **Command**-Objekt für einen bestimmten Datenprovider ausführen möchten (und die meisten von Ihnen möchten genau das), müssen Sie die **ActiveConnection**-Eigenschaft so einstellen, dass Sie auf ein **Connection**-Objekt an der Datenquelle verweist. Der Versuch, einen Befehl auszuführen oder selbst die **Parameters.Refresh**-Methode ohne gültige **ActiveConnection**-Eigenschafteneinstellung zu verwenden, führt – wie Sie sich denken können – zu einem auffangbaren Fehler.

> **Tipp** Nachdem Sie die **ActiveConnection**-Eigenschaft gesetzt haben, wird die **Name**-Eigenschaft des **Command**-Objekts eingefroren. Wenn Sie den Befehl nach Name referenzieren möchten, legen Sie die **Name**-Eigenschaft vor der **ActiveConnection**-Eigenschaft fest.

Vor dem Öffnen der Verbindung enthält die ActiveConnection-Eigenschaft eine »Definition« der Verbindung, und diese Definition ist eine einfache Verbindungszeichenfolge. Dies bedeutet, dass Sie die **ActiveConnection**-Eigenschaft mit Hilfe der gleichen Zeichenfolge festlegen können, die Sie für die **ConnectionString**-Eigenschaft des **Connection**-Objekts verwenden. Nachdem die Verbindung geöffnet wurde, enthält **ActiveConnection** einen gültigen Verweis auf das **Connection**-Objekt. Wenn Sie einen Befehl für eine Reihe von Verbindungen ausführen möchten, können Sie die ActiveConnection-Eigenschaft auf **Nothing** festlegen und später für andere **Connection**-Objekte einstellen.

Sie müssen die **ActiveConnection**-Eigenschaft festlegen, bevor Sie versuchen, die Methode **Parameters.Refresh** zu verwenden. Wenn Sie zum Erstellen der **Parameters**-Auflistung die **Parameters.Refresh**-Methode einsetzen und die **ActiveConnection**-Eigenschaft auf **Nothing** einstellen, löscht ADO die **Parameters**-Auflistung (aus irgendwelchen Gründen). Das Ändern der **ActiveConnection**-Eigen-

schaft wirkt sich jedoch nicht auf die **Parameters**-Auflistung aus, wenn Sie die Erstellung im Code vornehmen. Durch das Schließen des **Connection**-Objekts wird die **ActiveConnection**-Eigenschaft für jedes mit der Verbindung verknüpfte **Command**-Objekt auf **Nothing** gesetzt (wie gemein). Das Festlegen der ActiveConnection-Eigenschaft auf ein geschlossenes **Connection**-Objekt erzeugt einen Fehler (puuhh).

Die (so genannte) »Prepared«-Eigenschaft

Theoretisch wurde die **Prepared**-Eigenschaft entworfen, um durch eine Vorkompilierung von Ad-hoc-Abfragen den Arbeitsaufwand auf dem Server zu verringern, da nachfolgende Ausführungen anstelle einer erneuten Kompilierung für jede Abfrageausführung temporär gespeicherte Prozeduren verwenden. Dies ist jedoch bei der ADO-Implementierung **nicht** der Fall – lesen Sie weiter.

Da vor einigen Jahren ODBC eingeführt wurde, hat es verschiedene Verbesserungen an SQL Server gegeben – es können nun (im Cache) vorhandene, bereits kompilierte Abfragepläne wieder verwendet werden. Wenn Sie also eine Abfrage von (oder über) ADO ausführen, erstellt SQL Server Abfragepläne, speichert diese im Prozedurcache und führt sie aus. Ist die Abfrage abgeschlossen, markiert SQL Server den Abfrageplan für das mögliche Verwerfen, behält ihn aber so lange wie möglich im Cache bei. Muss eine identische (oder nahezu identische) Abfrage verarbeitet werden, was bei Systemen mit mehreren Clients durchaus möglich ist, setzt SQL Server den zwischengespeicherten Plan einfach erneut ein. Durch diese Technik wird nicht nur eine Menge Zeit gespart, sondern gleichzeitig auch die Skalierbarkeit erheblich verbessert. Tatsächlich wird SQL Server mit steigender Benutzerzahl **schneller** – vorausgesetzt, diese Benutzer führen mit gleichen Abfragesätzen ähnliche Aufgaben aus.

ADO und die zugehörigen ODBC- und OLE DB-Datenprovider kennen diese Strategie, und in den meisten Fällen nutzen Sie diese Funktion durch Einsatz von **sp_executesql**. Dies führt jedoch im Hinblick auf die **Prepared**-Eigenschaft zu einer Zwickmühle. Diese Eigenschaft besteht darauf, temporär gespeicherte Prozeduren zu erstellen, der Datenprovider besteht darauf **sp_executesql** zu verwenden. Das Ergebnis? Chaos. Dieses Thema wird weiter unten in diesem Kapitel im Rahmen der Befehlsausführung näher besprochen.

Meine Empfehlung für die **Prepared**-Eigenschaft: vergessen Sie sie – wenigstens für SQL Server. Bei anderen Providern sollten Sie eine Ablaufverfolgung einrichten, um einen genauen Überblick darüber zu behalten, was vorgeht – welche Operationen beim Server angefordert werden.

Die »CommandTimeout«-Eigenschaft

Die **CommandTimeout**-Eigenschaft gibt an, wie lange (in Sekunden) auf die Ausführung eines Befehls gewartet werden soll, bevor der Vorgang abgebrochen und ein Fehler erzeugt wird. Wie Sie wissen, startet **CommandTimeout**, wenn der Datenbankserver den Befehl akzeptiert hat, und endet, wenn der Datenbankserver den ersten Eintrag zurückgibt. Sind Server oder Netzwerk beschäftigt (oder antworten gar nicht), kann mit dieser Einstellung die Steuerung nicht zurückerlangt werden.

Es ist wichtig, diesen Wert auf einer realitätsnahen Basis festzulegen. Wenn Sie (nach wiederholtem Testen unter Spitzenbelastung) wissen, dass die Ausführung einer Abfrage längere Zeit in Anspruch nimmt, sollten Sie einen entsprechenden Wert für **CommandTimeout** einstellen – und einen gewissen Puffer mit einplanen. Denken Sie daran, dass der Server oder das Netzwerk zu einer Verzögerung beitragen können, wenn eine Lastenänderung eintritt oder die Datenbank weiteren Speicherplatz oder andere Ressourcen zuweisen muss. Der 30 Sekunden-Standardwert ist möglicherweise nicht ausreichend. Setzen Sie den Wert jedoch niemals auf 0, denn so wird die Zeitüberschreitung deaktiviert. Sie möchten doch nicht, dass Ihre Anwendung erstarrt, weil Sie auf eine Abfrage wartet, die nie zu Ende geht.

Zweimal diese Woche schon musste ich Benutzern bei der Behebung von Zeitüberschreitungsfehlern helfen. In beiden Fällen stellte sich heraus, dass eine Erhöhung des Wertes für die Zeitüberschreitung das Problem nicht löste. Zeitüberschreitungen werden durch die Unfähigkeit des Datenproviders verursacht, die angeforderte Operation in der vorgegebenen Zeit (in Sekunden) abzuschließen. Obwohl es zahlreiche Ursachen für einen nicht rechtzeitigen Abschluss einer Operation gibt, ist die häufigste Ursache jedoch eine Sperre. Hierbei richtet eine Anwendung eine Sperre für eine Seite, Zeile oder Tabelle ein, sodass andere Anwendungen (oder selbst Teile derselben Anwendung) nicht auf die Daten zugreifen können. Sämtliche Komponenten, die auf die Daten zugreifen möchten, müssen warten, bis die Sperren aufgehoben werden. In einem optimierten System dauert dieser Vorgang im Schnitt einige Sekunden. Hebt die sperrende Anwendung die eingerichteten Sperren nicht wieder auf (weil sie vielleicht darauf wartet, dass eine andere Sperre aufgehoben wird), werden die Anwendungen, die ebenfalls auf den Datenzugriff warten, für immer blockiert. Bevor Sie jetzt jedoch bei der Ermittlung des Schuldigen in die Ferne schweifen, sollten Sie sich einfach eingestehen, dass es meistens Ihre Anwendung ist, die die Sperre eingerichtet hat. Wenn Sie beispielsweise ein aktualisierbares Recordset öffnen und versuchen, die Datenbank mit Hilfe einer UPDATE-Aktionsabfrage zu aktualisieren, werden Sie verstehen, was ich meine.

In RDO (und bei der ODBC-API) stand eine weitere Option zur Verfügung, die bei der Konvertierung in ADO verloren ging – erneuter Versuch bei Zeitüberschreitung. Wenn Sie hierbei nach einer Zeitüberschreitung für einen Befehl weiterhin warten wollten, konnten Sie einfach ein Flag an die Ereignisbehandlungsroutine zurückgeben und weiter warten. Diese Funktion ist in ADO nicht implementiert. Warum nicht? Keine Ahnung.

Anmerkung Die ADO-Einstellungen für die Zeitüberschreitung sind unabhängig von der Zeitüberschreitung des Netzwerks. Da die NIC-Treiber (Network Interface Card) der niedrigen Ebene einen synchronen Netzwerk-API-Aufruf ausführen, und da dieser Aufruf nicht zurückgegeben wird bis die Zeitspanne für die Netzwerkzeitüberschreitung abgelaufen ist, wird der ADO-Zeitüberschreitungscode blockiert.

5.2.2 Handhabung parameterbasierter Abfragen

Die meisten der ausgeführten Abfragen erfordern mindestens einen Parameter zur Steuerung des erstellten Rowsets. Diese Parameter werden üblicherweise auf die WHERE-Klausel einer Abfrage angewendet, können jedoch auch in anderer Form eingesetzt werden. Wenn Sie eine parameterbasierte Abfrage erstellen, müssen Sie die Abfrageparameter beschreiben und die Laufzeitwerte bereitstellen, aber es ist **nicht** erforderlich, ein **Command**-Objekt zu verwenden – nicht, sofern Sie es mit dem Rückgabestatus einer gespeicherten Prozedur oder mit Ausgabeparametern zu tun haben. Sie können zum Übergeben der Eingabeparameter andere Methoden einsetzen, und diese werden im Folgenden beschrieben. Grundsätzlich stehen bei der Erstellung parameterbasierter Abfragen verschiedene Ansätze zur Verfügung:

- Erstellen eines **Command**-Objekts im Code, durch das eine leere ADO-**Parameters**-Auflistung offen gelegt wird. Bei diesem Ansatz kann die **Parameters**-Auflistung unter Verwendung der **Refresh**-Methode oder durch das Erstellen einzelner Parameter generiert werden.
- Erstellen einer SQL-Anweisung, bei der die Parameter in der Abfragezeichenfolge enthalten sind. Bei diesem Ansatz können Sie eine **sp_executesql**-Abfrage erstellen, statt diese von ADO erstellen zu lassen.
- Über die Visual Database Tools (einschließlich des Datenumgebungs-Designers), können ebenfalls parameterbasierte SQL-Anweisungen erstellt und als auf dem Datenumgebungs-Designer basierende **Command**-Objekte offen gelegt werden. Dieses Vorgehen wird umfassend im **Hitchhiker's Guide to Visual Basic and SQL Server** erläutert. In diesem Buch konzentrieren wir uns auf die ADO-Programmierung.

- Verwenden Sie das im Lieferumfang dieses Buches enthaltene Visual Basic-Add-In[7], mit dem der Code zum Erstellen von **Command**-Objekten generiert wird.

Parameterverwaltung durch ADO-»Command«-Objekte

Wenn Sie zur Verwaltung Ihrer Parameter die ADO-**Parameters**-Auflistung verwenden, liegt es in der Verantwortlichkeit von ADO, diese Parameter im richtigen Format und an der richtigen Stelle in die Abfrage einzufügen. ADO ist auch verantwortlich für die Handhabung von zusätzlichen, »umgebenden« Anführungszeichen. Wenn beispielsweise der Parameter Zeichenfolgen durch einfache Anführungszeichen kennzeichnet (was meistens der Fall ist), werden diese über ADO automatisch »verdoppelt« (ein einfaches Anführungszeichen wird durch zwei einfache Anführungszeichen ersetzt). Auf diese Weise geht die Abfrage nicht sofort flöten, wenn Sie das erste Mal mit einem irischen Nachnamen zu tun haben, beispielsweise O'Mally oder O'Brien.

Bei der Arbeit mit **Command**-Objekten liegt es, sofern Sie nicht die **Refresh**-Methode verwenden, in Ihrer Verantwortlichkeit, die Parameter richtig zu beschreiben. Dies bedeutet, dass Sie beim Erstellen der **Parameters**-Auflistung die Parameter nacheinander erstellen müssen, und zwar in der Reihenfolge, in der sie der Datenprovider erwartet. Nein, ADO und die zugehörigen Datenprovider unterstützen keine »benannten« Parameter, daher **müssen** diese in der richtigen Reihenfolge angegeben werden. Zu wissen, wie Datentyp, Größe, Genauigkeit, Skalierung und Schuhgröße des Parameters angegeben werden müssen, liegt ebenfalls in Ihrer Verantwortung. Wenn Sie etwas falsch machen, wird ein Fehler erzeugt. Wenn Sie eine falsche Reihenfolge verwenden, wer weiß, was passiert …

Es können verschiedene Techniken angeführt werden, mit denen parameterbasierte Abfragen effizienter und leichter zu erstellen sind. Letzten Endes werden wahrscheinlich all Ihre Produktionsanwendungen auf parameterbasierten Abfragen beruhen, da so nicht nur die Leistung (sowohl des Systems als auch der Entwickler) verbessert, sondern auch die Entwicklung komponentenbasierter Entwürfe vereinfacht wird.

Tipp Nein, Sie **müssen** kein **Command**-Objekt erstellen, um eine parameterbasierte Abfrage zu erstellen. Wenn Sie jedoch an den Stellen, an denen es sinnvoll ist, die Vorteile von **sp_executesql** nicht nutzen, könnten Sie von der Leistung Ihrer Abfrage enttäuscht sein.

7. Dieses Visual Basic-Add-In wird MSDN-Abonnenten auch zur Verfügung gestellt – dies sollte bis zur Veröffentlichung dieses Buches jedenfalls so sein.

Erstellen der »Parameters«-Auflistung

In der **Parameters**-Auflistung des ADO-**Command**-Objekts ist für jeden Geschmack etwas dabei: **gazinta**- (Eingabe), **gazouta**- (Ausgabe) und **gazinta-gazouta**-Parameter (Eingabe-Ausgabe, bidirektional). Denken Sie daran, Eingabeparameter können auf Ad-hoc-Abfragen und auf gespeicherte Prozeduren angewendet werden. Der Trick ist, herauszufinden, wie und wann die **Parameters**-Auflistung des **Command**-Objekts im Code erstellt werden muss. Hierbei gibt es (wie gesagt) zwei Ansätze:

- Verwenden Sie die **Command.Parameters.Refresh**-Methode, um basierend auf dem angegebenen Befehlstext über ADO und die zugehörigen Provider die **Parameters**-Auflistung erstellen zu lassen.
- Eigenes Erstellen der **Command.Parameters**-Auflistung – Parameter für Parameter, basierend auf Ihrem Verständnis der Parameterdefinition.

Beide Methoden weisen Vorteile und Nachteile im Hinblick auf die Entwickler- und Anwendungsleistung auf. Meiner Meinung nach sollten beide Methoden nach Möglichkeit in der mittleren Schicht eingesetzt werden. Warum? Nun, Sie müssen berücksichtigen, dass die Zeit für die Ausführung von zusätzlichem Code zur Erstellung des **Command**-Objekts und der geeigneten **Parameter**-Objekte (einzeln und nacheinander) eigentlich vergeudet wird. Der Code muss immer dann erneut ausgeführt werden, wenn auf die ASP verwiesen oder die MTS-Komponente ausgeführt wird. Ja, das **Command**-Objekt trägt zu einer einfacheren Codierung bei. Wenn Sie nach einer einfachen Lösung suchen, die eine gewisse Skalierbarkeit aufweist, dann lesen Sie weiter.

Verwenden der »Refresh«-Methode Die **Command.Parameters.Refresh**-Methode scheint sämtliche Arbeit für Sie zu erledigen – die **Parameters**-Auflistung des **Command**-Objekts wird in einer einzelnen Codezeile erstellt. Das ist gut und schlecht. Es ist gut, da Sie sich nicht darum zu kümmern brauchen, wie ADO die **Parameter**-Objekte erstellt (in den meisten Fällen klappt alles). Schlecht ist, dass ADO und die Provider eine erneute Serveranfrage ausführen, um die Vorgehensweise zu ermitteln, und dies kann Kosten verursachen (wie bereits ausgeführt wurde). Da es sich hierbei um eine einmalige Leistungsbeeinträchtigung der Anwendung im frühen Stadium handeln kann, sind die Auswirkungen vielleicht nicht besonders groß.

Denken Sie daran, die **ActiveConnection**-Eigenschaft festzulegen, **bevor** Sie versuchen, die **Refresh**-Methode zu verwenden – ADO benötigt eine gültige Verbindung zur Datenbank, um die Parameter zu generieren.

Tipp Tatsächlich müssen Sie nicht einmal die **Refresh**-Methode verwenden, wenn Sie nicht möchten. Das Verwenden dieser Methode kann sogar dazu führen, dass ADO eine **weitere zusätzliche** Serveranfrage ausführt. Wenn Sie versuchen, das erste Mal eine Eigenschaft einer nicht initialisierten **Command.Parameters**-Auflistung zu lesen, wird die Parameterauflistung über ADO erstellt – so als hätten Sie die **Refresh**-Methode ausgeführt.

Nach Ausführen der **Parameters.Refresh**-Methode müssen einige der **Parameter**-Objekte vielleicht noch überarbeitet werden. Wenn Ihre gespeicherte Prozedur beispielsweise OUTPUT-Parameter erwartet, weist der Provider ADO häufig an, die **Parameter.Direction**-Eigenschaft auf **adParamInputOutput** statt auf **adParamOutput** zu setzen. Dies ist keine Katastrophe, da Sie beim Aufrufen der gespeicherten Prozedur einfach leere, Null- oder Standardwerte bereitstellen können.

Ein (gewisser) Vorteil bei Verwendung der **Parameters.Refresh**-Methode besteht darin, dass sich bei einer Änderung von Anzahl, Position oder Datentyp der Parameter Ihre Anwendung automatisch an die neuen Parameteranforderungen anpassen kann. Wenn es sich hierbei jedoch um wesentliche Änderungen handelt, kommt es leicht vor, dass über den Code falsche Parameterwerte übergeben werden. Ihr Code referenziert **Parameter**-Objekte nach Position – **nicht** nach Name. Angenommen, der Datentyp von Parameter vier ändert sich von **SmallInteger** in **Integer** – keine große Sache. Wenn Sie jedoch eine Änderung von **VarChar** in **VarBinary** vornehmen, sieht das schon ganz anders aus.

Erstellen der »Parameters«-Auflistung im Code Die zweite Vorgehensweise weist ebenfalls Vor- und Nachteile auf. Zur Erstellung der **Parameters**-Auflistung im Code müssen Sie die von ADO und der aufgerufenen Prozedur erwartete Erstellungsweise des **Parameter**-Objekts kennen. Dies impliziert des Weiteren, dass Sie ADO gut genug kennen, um eine Codierung des **Parameter**-Objekts selbst vornehmen zu können **und** dass Sie wissen, dass die Definitionen der Parameter **keinen** Änderungen unterliegen.

Bereits die Auswahl des geeigneten Datentyps für jedes Objekt kann schwierig sein – und zur Erstellung der **Parameter**-Objekte gehört häufig noch mehr, was die Sache kompliziert macht. Aus diesem Grund lasse ich diese Aufgabe häufig durch den Datenumgebungs-Designer oder durch die **Parameters.Refresh**-Methode erledigen. Obwohl der Datenumgebungs-Designer bei der Erstellung der Parameterauflistung die gleichen Fehler begeht wie ADO, muss der Entwickler bei dieser Methode jedoch weniger raten. Zur Ausnutzung der vordefinierten **Parameters**-Auflistung erstelle ich mit dem Datenumgebungs-Designer ein **DataEnvironment**-Objekt und kopiere die generierten Einstellungen für jede **Parameter**-Eigenschaft in meinen Code – besonders die einzelnen Datentypen für die Paramterobjekte.

Nachteile der »Refresh«-Methode Betrachten wir nun, wie ADO bei Verwendung der **Refresh**-Methode vorgeht – besonders in Zusammenhang mit der so genannten **Prepared**-Eigenschaft. In der Dokumentation werden einige wichtige Punkte nicht erwähnt. Im folgenden Beispiel wurde die **Refresh**-Methode zum Erstellen der **Parameters**-Auflistung eingesetzt:

```
With cmd
   .Name = "GetTitles"
   .Prepared = False
   .CommandType = adCmdText
   .ActiveConnection = cn
   .CommandText = "Select title from titles " _
       & "where title like ? " _
       & "and year_published between ? and ?"
   .Parameters.Refresh     ' Erstellen der Parameters-Auflistung durch ADO
zulassen
```

Gemäß SQL Server 7.0 Profiler wird bei Ausführung der **Refresh**-Methode der Provider (in diesem Fall SQL Server) angewiesen, die folgenden zwei Abfragen auszuführen (zwei zusätzliche Anforderungen):

```
SET FMTONLY ON select  title, year_published, year_published from  titles SET
FMTONLY OFF
declare @P1 int
set @P1=NULL
sp_prepare @P1 output, N'@P1 varchar(255),@P2 smallint,@P3 smallint',
N'Select title from titles where title like @P1 and year_published between @P2
and @P3', 1 select @P1
```

Bei Ausführung der Anwendung geschieht Folgendes:

```
cmd(eParms.TitleWanted) = "Hitch%"
cmd(eParms.YearHigh) = 1950
cmd(eParms.YearLow) = 1999
Set rs = New Recordset
rs.Open cmd
```

Der Profiler informiert Sie darüber, dass der SQL Server-Datenprovider die temporär gespeicherte Prozedur **sp_unprepare** (1) zerstört. ADO und der SQL Server-Provider fahren anschließend damit fort, unter Verwendung von **sp_executesql** die Abfrage auszuführen:

```
sp_unprepare
sp_executesql N'Select title from titles where title like @P1 and
year_published between @P2 and @P3', N'@P1 varchar(255),@P2 smallint,@P3
smallint', 'Hitch%', 1999, 1950
```

Jede nachfolgende Ausführung des **Command**-Objekts, unabhängig von der Syntax, führt zu einem Aufruf von **sp_executesql** mit den neuen Parametern:

```
sp_executesql N'Select title from titles where title like @P1 and
year_published between @P2 and @P3', N'@P1 varchar(255),@P2 smallint,@P3
smallint', 'Any%', 1999, 1950
```

Okay, dies ist nicht weiter schlimm (nur eine unnötige Zerstörung und Neuerstellung), was geschieht jedoch, wenn **Prepare** auf **True** gesetzt wird? Dieses Vorgehen scheint ADO und den SQL Server-Datenprovider zu verwirren. Genau wie im vorherigen Fall (in dem **Prepare** auf **False** gesetzt wurde), werden die DDL-Abfragen bei Verwendung der **Refresh**-Methode zur Erstellung der temporär gespeicherten Prozedur (**sp_prepare**) verwendet. Eigentlich hat sich nichts geändert. Wenn Sie jedoch die Vorgehensweise »**Command**-Objekt als **Connection**-Methode« einsetzen, verwendet ADO die vorhandene temporär gespeicherte Prozedur, zerstört diese jedoch bei **jeder nachfolgenden** Ausführung und erstellt sie anschließend neu.

Erstellen der »Parameters«-Auflistung im Code

Was geschieht, wenn Sie die Parameter selbst im Code erstellen – ohne die **Refresh**-Methode zu verwenden? Nun, wenn Sie **Prepared=True** einstellen, geht ADO wie zuvor beschrieben vor, statt jedoch die temporär gespeicherte Prozedur bei Ausführung der **Refresh**-Methode anhand von **sp_prepare** zu erstellen, wird diese nun bei erstmaliger Ausführung des Befehls ausgeführt. Darüber hinaus verwendet ADO die temporär gespeicherte Prozedur wieder, bis die Technik »**Command**-Objekt als **Connection**-Methode« eingesetzt wird, und ADO zu seinen alten schlechten Gewohnheiten zurückkehrt.

Wenn Sie jedoch die **Parameters**-Auflistung im Code erstellen und die **Prepared**-Eigenschaft **nicht** auf **True** setzen (der Standardwert lautet **False**), weiß ADO, wie vorzugehen ist – zur Abfrageausführung werden **sp_executesql**-Anweisungen erstellt. Wenigstens das weiß es. Keine zusätzlichen Abfragen zur Einrichtung der temporär gespeicherten Prozeduren, nur um diese wieder zu zerstören. Es werden lediglich die Abfragen ausgeführt.

```
sp_executesql N'Select title from titles where title like @P1 and
year_published between @P2 and @P3', N'@P1 varchar(20),@P2 int,@P3 int',
'Hitch%', 1999, 1950
```

```
sp_executesql N'Select title from titles where title like @P1 and
year_published between @P2 and @P3', N'@P1 varchar(20),@P2 int,@P3 int',
'Any%', 1999, 1950
sp_executesql N'Select title from titles where title like @P1 and
year_published between @P2 and @P3', N'@P1 varchar(20),@P2 int,@P3 int',
'Hitch%', 1940, 1999
```

Das SQL Server-Team kennt diesen Bug der **Prepared**-Eigenschaft, und obwohl dieses Problem mit SQL Server 7.0 Service Pack 2 (bisher) noch nicht vollständig gelöst wurde, ist eine Besserung jedoch in Sicht. Nach dem Installieren von SP2 tritt das fehlerhafte Verhalten nur dann auf, wenn Sie die Vorgehensweise »**Command**-Objekt als **Connection**-Methode« verwenden. Obwohl ich diese Methode aufgrund ihrer Einfachheit und Flexibilität schätze, würde ich sie bis zur vollständigen Beseitigung des Bugs nur mit Vorsicht einsetzen.

Dieser Bug verhindert, dass die Verwendung der **Refresh**-Methode zu den empfohlenen Vorgehensweisen gezählt werden kann, besonders in Verbindung mit der Einstellung **True** für die **Prepared**-Eigenschaft. Wenn Sie die **Parameters**-Auflistung im Code erstellen, begeht ADO diese Fehler nicht.

Wie können also (bis zur Beseitigung des Bugs durch Microsoft) diese Probleme verhindert werden?

- Setzen Sie die **Prepared**-Eigenschaft nicht auf **True**. So wird die Sache nur schlimmer.
- Erstellen Sie die **Parameters**-Auflistung für Ad-hoc-Abfragen im Code.
- Verwenden Sie für **adCmdtext**-Befehle nicht die **Refresh**-Methode. Auf diese Weise verwendet ADO für Abfragen die **sp_execute**-Strategie. Da SQL Server hinsichtlich dieses Aspekts anscheinend optimiert wurde, sollte sich dies nicht negativ auf die Leistung auswirken, im Gegenteil.
- Seien Sie bei Verwendung des **Command**-Objekts als **Connection**-Methode vorsichtig.
- Achten Sie auf den Profiler, um zu ermitteln, ob durch die Abfragen zusätzliche **sp_unprepare**- und **sp_prepare**-Operationen generiert werden.
- Ziehen Sie die im Kapitel über Recordsets erläuterte Technik der Verwendung von gespeicherten Prozeduren als **Connection**-Objekt in Betracht, oder verwenden Sie Strategien, bei denen keine **Command**-Objekt eingesetzt wird.

Die »Parameter.Direction«-Eigenschaft Über die **Direction**-Eigenschaft des **Parameter**-Objekts wird ADO mitgeteilt, wie und wann Daten für diesen Parameter zu erwarten sind. Der Standardwert für diese Eigenschaft lautet **adParamInput**, aber wie in Abbildung 5.1 dargestellt wird, stehen verschiedene Alternativen zur Verfügung:

- **adParamReturnValue**: Falls deklariert, ist der Rückgabewertparameter der erste der **Parameters**-Auflistung. Dieser wird zum Empfang der Rückgabestatusganzzahl der gespeicherten Prozedur eingesetzt.
- **adParamInput** (Standardeinstellung): Der Parameter wird der Abfrage zur Laufzeit übergeben.
- **adParamInputOutput**: Der Parameter wird der Abfrage zur Laufzeit übergeben, derselbe Parameter wird zum Empfang eines Wertes von der gespeicherten Prozedur verwendet.
- **adParamOutput**: Der Parameter wird zum Empfang eines Wertes von der gespeicherten Prozedur verwendet.
- **adParamUnknown**: Wenn ADO oder Sie nicht ermitteln können, wie ein Parameter gehandhabt werden soll, wird dieser **unknown**-Wert eingesetzt.

Abbildung 5.1 Auflistungskonstanten der »Direction«-Eigenschaft

Das erste **Parameter**-Objekt in der Parameterauflistung ist ein besonderes. Es kann entweder den Rückgabestatus einer gespeicherten Prozedur oder den ersten Parameter enthalten. Nur das erste **Parameter**-Objekt in der **Properties**-Auflistung kann auf **adParamReturnValue** gesetzt werden. Diese Position ist gespeicherten Prozeduren vorbehalten, die einen **integer**-Statuswert zurückgeben (ja, es ist immer ein **integer**-Wert). Wenn Sie keine gespeicherte Prozedur, sondern eine einfache Parameterabfrage ausführen, kann das erste Argument als **adParamInput** definiert werden. Bei gespeicherten Prozeduren kann der Rückgabestatus ignoriert werden, indem die erste (Ordinalzahl 0) **Direction**-Eigenschaft des **Parameter**-Objekts auf eine andere Richtungskonstante eingestellt wird.

Warnung Im Lokal-Fenster von Visual Basic wird die **Parameters**-Auflistung verweisbasiert angezeigt. **cmd.Parameters(0)** wird daher als **Parameters(1)** aufgeführt. Großartig.

Die »Parameter.Type«-Eigenschaft Beim Arbeiten in der Visual Basic-IDE kann Ihnen über die Funktion zur Anweisungsvervollständigung eine Liste der wählbaren gültigen Datentypen angezeigt werden. Der Trick besteht darin, den

richtigen auszuwählen. Die anderen Trainer, Supportmitarbeiter und ich erhalten ziemlich häufig Fragen zur **Type**-Eigenschaft des **Parameter**-Objekts, mit dem der zugehörige Datentyp beschrieben wird. Abbildung 5.2 zeigt, wie diese Abfrage der VB-IDE erfolgt, es steht jedoch eine Vielzahl von Datentypen zur Auswahl. Die einfachste Methode zur Ermittlung des richtigen Datentyps ist die Verwendung von ADO – zumindest anfänglich. Verwenden Sie hierbei die **Refresh**-Methode zum Auffüllen der **Parameters**-Auflistung, und untersuchen Sie die **Type**-Eigenschafteneinstellungen für jeden Parameter. Prüfen Sie hierbei auch die Einstellungen für **Size**, **Precision** und **NumericScale**. Es gibt eine Methode, die noch einfacher ist und keinen Code erfordert. Verwenden Sie das Datenansichtsfenster zum Aufrufen einer Tabelle, die ein Feld enthält, auf deren Datentyp ADO zugreifen soll. Klicken Sie mit der rechten Maustaste auf das betreffende Feld, und wie durch Magie erscheint die richtige Auflistungskonstante für den ADO-Datentyp.

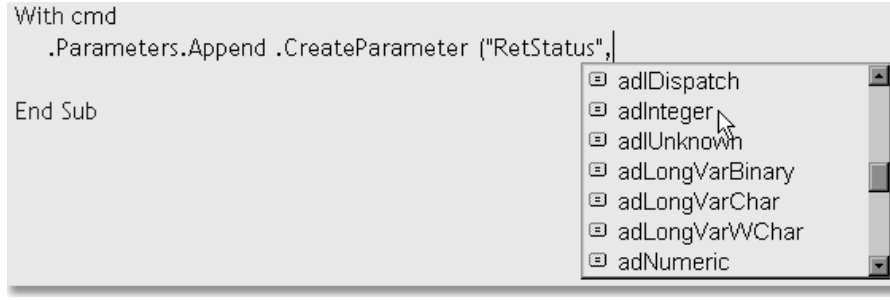


Abbildung 5.2 VB-IDE mit einer Liste möglicher ADO-Datentypen

In Tabelle 5.2 werden sämtliche SQL Server-Datentypen sowie einige Datentypen aufgeführt, die besondere Aufmerksamkeit erfordern. Beachten Sie den Datentyp **datetime** – yip, dieser muss als **adDBTimeStamp**[8] übergeben werden. Der Code zum Generieren dieser Tabelle ist auf der Begleit-CD enthalten und kann für Ihren eigenen Datenprovider ausgeführt werden.

SQL Server-Datentyp	ADO-»Parameter Type«-Eigenschaft	Genauigkeit (Precision)	Größe (Size)
(RÜCKGABEWERT)	adInteger	10	0
Varchar	adVarChar		(Ihre Wahl)
Char	adVarChar		(Ihre Wahl)
Int	adInteger	10	0

Tabelle 5.2 SQL Server-Datentyp und geeignete Auflistungskonstanten für den ADO-Datentyp

8. Der Grund für die Verwendung des **TimeStamp**-Datentyps anstelle eines anderen (**adDateTime** o. ä.) ist so unbedeutend, dass er hier nicht erwähnt werden soll.

SQL Server-Datentyp	ADO-»Parameter Type«-Eigenschaft	Genauigkeit (Precision)	Größe (Size)
Smallint	adSmallInt	5	0
Tinyint	adUnsignedTinyInt	3	0
Datetime	adDBTimeStamp		0
Smalldatetime	adDBTimeStamp		0
Bit	adBoolean		0
Text	adVarChar		2.147.483.647
Image	adVarBinary		2.147.483.647
Binary	adVarBinary		1
Varbinary	adVarBinary		1
Decimal	adNumeric	18	0
Smallmoney	adCurrency	10	0
Money	adCurrency	19	0
Numeric	adNumeric	18	0
Real	adSingle	7	0
Float	adDouble	15	0
Nchar	adVarWChar		1
Ntext	adVarWChar		1.073.741.823
Nvarchar	adVarWChar		1

Tabelle 5.2 SQL Server-Datentyp und geeignete Auflistungskonstanten für den ADO-Datentyp

Die »Parameter.Size«-Eigenschaft Verwenden Sie die **Size**-Eigenschaft zur Ermittlung der maximalen Größe von Werten, die aus der **Value**-Eigenschaft eines **Parameter**-Objekts gelesen oder in dieses geschrieben werden. Überschreitet die Parametergröße diesen Höchstwert, wird ein auffangbarer Fehler erzeugt.

Ja, einige Datentypen erfordern keine Einstellung der **Size**-Eigenschaft. Sie **müssen** die Eigenschaft für die **Char**- und **VarChar**-Felder setzen, da hier die Größe variieren kann. ADO und der Datenprovider gaben die in Tabelle 5.2 gezeigten standardmäßigen **Size**-Eigenschaften zurück (mit Ausnahme für **Char** und **VarChar**). Beachten Sie, dass die (maximale) Größe für **NText** (Unicode-Text) die Hälfte des äquivalenten ANSI-Textdatentyps beträgt.

Die Eigenschaften »Parameter.Precision« und »Parameter.NumericScale« Die **Precision**- und **NumericScale**-Eigenschaften werden zur Beschreibung der Zahlen eingesetzt, die Dezimalpunkte oder variierende Genauigkeitsgrade aufweisen. Für Bruchzahlen sollte mit Hilfe der **NumericScale**-Eigenschaft festgelegt werden, wie viele Stellen nach dem Dezimalkomma zur Darstellung der Werte verwendet werden. Die **Precision**-Eigenschaft wird dazu eingesetzt, die maximale Anzahl der Stellen anzugeben, die für eine Zahl gespeichert werden – Ganzzahl oder Fließkomma. Diese Eigenschaften müssen im Code gesetzt werden, da Sie durch die **CreateParameter**-Methode nicht unterstützt werden.

Erstellen der »Parameter«-Objekte mit »CreateParameter« Sie können einzelne **Parameter**-Objekte erstellen und diese anhand der **Append**-Methode der **Parameters**-Auflistung hinzufügen.

```
Dim Pr as Parameter
Set Pr = New Parameter
With Pr
   .Name = "P1"
   .Type = adVarChar
   .Direction = adParmInput
   .Size = 30
   .Value = "Fred graduates in 2000"
End With
Cmd.Append Pr
```

Wenn Sie nach Zeile berechnen, ist dieser der beste Ansatz – es ist jedoch sehr viel einfacher, all dies mit einer einzigen Codezeile auszudrücken:

```
.Parameters.Append .CreateParameter("P1", adVarChar, adParamInput, 30, _
"Fred graduates in 2000")
```

Durch die **CreateParameter**-Methode wird das gesamte **Parameter**-Objekt in einem einzigen Schritt erstellt. Hierbei werden zwar die Eigenschaften **NumericScale** oder **Precision nicht** eingeschlossen, die anderen wichtigen Eigenschaften jedoch schon – einschließlich **Value**. Da diese Methode Sie an alle wichtigen Eigenschaften erinnert, indem jede einzelne bei der Eingabe in der Visual Basic-IDE abgefragt wird, kann diese Technik als empfohlene Vorgehensweise eingestuft werden.

Entwerfen eigener Parameterabfragen

Nein, Sie müssen zum Ausführen einer Parameterabfrage oder zur Ausführung von Abfragen ohne Parameter nicht unbedingt die **Parameters**-Auflistung verwenden. Sie können beispielsweise eine Abfrage in einer Zeichenfolge erstellen und diese

der **Source**-Eigenschaft eines Recordsets oder der **CommandText**-Eigenschaft eines **Command**-Objekts übergeben. Sie können darüber hinaus eine beliebige gespeicherte Prozedur als Methode des **Connection**-Objekts ausführen. Die Parameter der gespeicherten Prozedur werden einfach als Methodenargumente[9] übergeben. Verfügt die Abfrage über Parameter, können Sie die Parameter in der WHERE-Klausel miteinander verketten. Gibt die gespeicherte Prozedur ein Recordset zurück, übergeben Sie dieses als letztes Argument, wie nachfolgend gezeigt wird:

```
Set Rs = New Recordset
Cn.MySP "My parm1", 2, "O'Malley", RS
```

In diesem Beispiel wird ein Parameterabfrage ohne (offensichtliche) Verwendung des **Command**-Objekts ausgeführt – die **Source**-Eigenschaft des **Recordset**-Objekts wird lediglich auf eine erstellte SELECT-Anweisung eingestellt. Tatsächlich wird im Hintergrund von ADO ein **Command**-Objekt zur Verwaltung der Abfrage erstellt.

```
Set rs = New Recordset
With rs
   SQL = "Select title, Year_Published from titles " _
      & "where title like '" & txtParm1 & "'" _
      & "and year_published between " & txtParm2 & " and " & txtParm3
   .ActiveConnection = cn
   .Source = SQL
   .CursorType = adOpenStatic
   .LockType = adLockOptimistic
   .Open Options:=adCmdText
End With
```

Der Datenprovider wird, je nach Erstellungsort des Cursors, sehr unterschiedlich verwaltet. Wird vor dem Öffnen einer Verbindung ein clientseitiger Cursor ausgewählt, lauten die zwei einzigen SQL-Anweisungen, die zum Server gesendet werden, folgendermaßen:

```
SET NO_BROWSETABLE ON
Select title, Year_Published from titles where title like 'Hitch%'and
year_published between 1900 and 1999
```

9. Das Verfahren der gespeicherten Prozedur als **Connection**-Methode wird in Kapitel 6 besprochen.

Benötigt der clientseitige Cursor Zeilen, ruft ADO diese einfach aus den lokal zwischengespeicherten Daten ab, die über die einzige SELECT-Anweisung geladen werden.

Werden jedoch serverseitige Cursor verwendet, weist ADO den Datenprovider an, einen serververwalteten Cursor zu erstellen. Werden Zeilen benötigt, führt der Provider eine weitere Abfrage zum Zeilenabruf vom Cursor aus.

```
sp_cursoropen @P1 output, N'Select title, Year_Published from titles where
title like ''Hitch%''and year_published between 1900 and 1999', @P3 output,
@P4 output, @P5 output select @P1, @P3, @P4, @P5
sp_cursorfetch 531488860, 16, 1, 2048
sp_cursorfetch 531488860, 16, 1, 1
sp_cursorclose 531488860
```

Beachten Sie, dass der Provider Abfragen zur Ausführung von gespeicherten Prozeduren übermittelt, die möglicherweise in der Onlinedokumentation nicht dokumentiert werden – dies rührt daher, dass keine Dokumentation vorhanden ist. Die mit **sp_cursor…** beginnenden gespeicherten Prozeduren werden durch die Routinen installiert, mit denen die ODBC- und OLE DB-Datenzugriffsprovider eingerichtet werden. Deren Quelle ist verschlüsselt, daher können Sie nicht sehen, welche magischen Dinge der Provider hinter den Kulissen vollführt, um Ihren Cursor zu implementieren. Wenn ich die Erlaubnis dazu erhalte, werde ich diese neuen gespeicherten Prozeduren dokumentieren und die Ergebnisse veröffentlichen.

Harte Brocken bei der Parametrisierung Während Konstanten leicht in die WHERE-Klausel einer SQL-Abfrage eingefügt werden können, ist das Parametrisieren anderer Bestandteile einer Abfrage nicht immer so einfach. Wenn Sie nicht die zuvor genannte Verkettungstechnik verwenden, erfordert das Ändern von Tabellen, Feldern oder Verknüpfungssyntax die Erstellung eines besonderen Typs Abfrage, die bei jeder Ausführung kompiliert wird. Dies ist jedoch ein TSQL-Thema für ein weiteres Buch.

Festlegen von Parameterwerten

Gelegentlich möchten Sie vielleicht dem einen oder anderen Parameter einer Abfrage eine Konstante oder eine Variable übergeben. Die **Value**-Eigenschaft dient der Datenübergabe von und zur Prozedur – in Abhängigkeit von der Einstellung für die **Direction**-Eigenschaft. Bevor also das **Command**-Objekt ausgeführt wird, müssen Sie die **Value**-Eigenschaft für alle Eingabeparameter setzen, deren Standardwert **nicht** in der gespeicherten Prozedur definiert ist. Dies bedeutet, dass Sie einen Laufzeitparameterwert für alle Ad-hoc-Abfragenparameter angeben müssen.

Tipp: Ich werde noch erläutern, wie Standardwerte in Eigenschaften verwendet und gesetzt werden. Siehe hierzu den Abschnitt »Verwenden der **Execute**-Methode für ein **Command**-Objekt« weiter unten in diesem Kapitel.

Beachten Sie, dass bei der ersten Erstellung des **Command**-Objekts eine anfängliche **Value**-Eigenschafteneinstellung gesetzt werden kann. Dieser Wert ist jedoch bei der Erstellung des **Command**-Objekts nicht schreibgeschützt, daher handelt es sich nicht um einen richtigen Standardwert. Ist der Wert eingestellt, verwendet ADO diesen Wert, wenn vor der Befehlsausführung kein anderer spezifischer Wert angegeben wird.

Beim Schreiben von gespeicherten Prozeduren können Sie ebenfalls Standardwerte angeben. Im folgenden Beispiel wird z.B. die Codierung einer gespeicherten SQL Server-Prozedur veranschaulicht, deren erstes Argument eine Standardeinstellung aufweist. Wenn also der gespeicherten Prozedur kein Parameter übergeben wird, wird der Standardwert verwendet.

```
Create Procedure DefaultValues
   (@InputParm VarChar(20) = 'Fred', @InputParm2 integer)
As
   SELECT @InputParm Stuff
   return @InputParm2
```

Die **Value**-Eigenschaft des **Command**-Objekts kann nach Name referenziert werden. ADO benennt die **Parameter**-Objekte für Sie als Param1, Param2 usw. Bedenken Sie jedoch, dass die Referenzierung eines Auflistungsobjekts nach Name mehr Zeit beansprucht als eine Referenzierung nach Ordnungszahl, wie als Nächstes gezeigt wird. Sie können bei der (empfohlenen) Erstellung eigener **Parameter**-Auflistungen die **Parameter**-Objekte ebenfalls benennen.

```
Set cmd = New Command
With cmd
   .ActiveConnection = cn
   .CommandText = "DefaultValues"
   .CommandType = adCmdStoredProc
   .Parameters.Refresh
   Set rs = .Execute
   Debug.Print .Parameters(0)  ' Rückgabestatus
   Debug.Print .Parameters(1)  ' Eingabeparameter (Standardwert?)
End With
```

Was geschieht, wenn der Wert falsch ist? ADO verfügt über eine recht brutale Methode zur Handhabung von Verstößen gegen die Feldeinschränkungen. Okay, ich bin vielleicht etwas empfindlich, aber wenn Sie eine 22-Byte-Zeichen-

folge einem Feld oder Parameter zuordnen, das Sie als **VarChar(20)** beschrieben haben, gibt ADO einen auffangbaren Fehler aus. Ich hatte erwartet, dass ADO die Zeichenfolge einfach abschneidet, aber ich denke, dies ist besser – Sie werden darüber informiert, dass eine Feldeinschränkung verletzt wurde. Es bedeutet auch, dass Sie Ihre Routinen zur Parametervalidierung überarbeiten müssen. Das gleiche passiert, wenn Sie ein ungültiges Datum, beispielsweise den 30. Februar 2000, oder einen ungültigen **integer**-Wert übergeben (einen zu kleinen oder zu großen Wert, oder einen Wert mit Dezimalkomponente).

Nein, Sie brauchen keine Werte für Parameter von gespeicherten Prozeduren anzugeben, für die Standardwerte definiert wurden. Dies bedeutet, dass Sie vielleicht keine Parameter angeben müssen. Wenn Sie keinen Wert darauf legen, Rückgabestatus oder Ausgabeparameter zu erfassen, brauchen Sie überhaupt kein **Command**-Objekt zu codieren – führen Sie die gespeicherte Prozedur lediglich unabhängig vom **Connection**-Objekt aus:

```
Cn.spAllDefaultValues
```

Die **Parameters.Item(n) Value**-Eigenschaft ist die Standardeigenschaft für das **Command**-Objekt. Zur Referenzierung einzelner Parameterwerte können Sie daher folgendermaßen vorgehen:

```
MyCommand(3) = "George has a baby"        ' Notes-Parameter
```

Anstelle von:

```
MyCommand.Parameters.Item(3).Value = "George has a baby"    ' Notes-Parameter
```

Wir werden die effiziente Referenzierung von COM-Objekten später noch besprechen, für den Moment soll die Erwähnung genügen, dass über Zeichenfolgen referenzierte Parameter zwar einfacher zu lesen sind, jedoch im Vergleich zu über Zahlen referenzierte Parameter nahezu doppelt soviel Zeit in Anspruch nehmen. Dies bedeutet, dass **keine** der beiden Methoden zu empfehlen ist, wenn das Hauptziel in einer guten Laufzeitleistung besteht:

```
MyCommand("Notes") = "George has a baby"    ' Notes-Parameter
MyCommand!Notes = "George has a baby"    ' Notes-Parameter
```

Wie also erreichen Sie gute Lesbarkeit für den Menschen? Ein Ansatz ist die Verwendung von Parameteraufzählungen. Wenn Sie also folgende **Enum** definieren:

```
Enum eParms
   ReturnStatus
   PName
   PGrade
   Notes
End Enum
```

können Sie auf einzelne **Value**-Eigenschaften ausgewählter **Parameter**-Objekte verweisen, wie nachfolgend gezeigt wird:

```
MyCommand(eParms.Notes) = "It's a boy!"  ' Ja, ADO kümmert sich um die zusätzlichen Apostrophe
```

Das Verweisen auf **Field**- und **Parameter**-Objekt durch **Enum** oder Ordnungszahlen (mit Kommentaren) ist eine weitere empfohlene Vorgehensweise, da so Lesbarkeit und Leistung verbessert werden.

Wenn Sie die erforderlichen Parameterwerte nicht angeben, wird dies von ADO durch einen auffangbaren Fehler angezeigt. Ich habe beispielsweise einen der drei erforderlichen Parameter für eine Ad-hoc-Abfrage angegeben und den folgenden Fehler (**Err.Description**) erhalten:

```
[Microsoft][ODBC SQL Server Driver][SQL Server]Prepared statement '(@P1 varchar(255),@P2 smallint,@P3 smallint)Select title from ti' expects parameter @P2, which was not supplied.
```

Meine Abfrage sah folgendermaßen aus:

```
"Select title from titles where title like ? and year_published between ? and ?"
```

Jetzt verstehen Sie vielleicht, warum einige Entwickler wahrscheinlich genauso verwirrt wären wie ADO. Nun, da ich für die Abfrage **Prepared = True** eingestellt hatte, war diese **Prepared**-Anweisungssyntax nicht ganz unerwartet. Um etwas Verwirrung zu stiften, kürzte ich die Fehlerbehandlungsroutine (im Datenprovider) der Abfrage etwas (schnitt sie kurz nach der **From**-Klausel ab). Zur Vorbereitung der Abfrage erstellte ADO eine temporär gespeicherte Prozedur für die Abfrageausführung – die SQL für die Abfrage spiegelt sich in der Fehlermeldung wider. Ergibt das einen Sinn? Ich hoffe doch.

Abrufen von Parameterwerten

Der Rückgabestatus einer gespeicherten Prozedur sowie die Parameter, die als Eingabe- (**adParamOutput**) oder Eingabe/Ausgabe-Parameter (**adParamInputOutput**) markiert sind, stehen zur Verfügung, jedoch erst **nachdem** sie vom Datenprovider gesendet wurden. Dies bedeutet, dass sie erst sichtbar sind, wenn das Recordset EOF (End of File) erreicht, es sei denn, das Protokoll der unteren Ebene gibt diese früher zurück (dies ist üblicherweise nicht der Fall). Denken Sie daran – der Rückgabestatus ist immer der erste (0) Parameter in der **Parameters**-Auflistung – wenn Sie diesen abrufen.

```
MyRetStatus = cmd(0)          ' Parameter für den Rückgabestatus
```

Handhabung von OUTPUT-Parametern ADO scheint nicht zu wissen, wie gute OUTPUT-Parameterabfragen erstellt werden. Da ADO nicht wirklich raten kann, wie die Parameter im Code verwendet werden[10], geht ADO davon aus, dass es sich bei einfachen OUTPUT-Parametern tatsächlich um INPUT-OUTPUT-Parameter handelt. Aufgrund dieser Tatsache müssen Sie für jeden OUTPUT-Parameter die **Direction**-Eigenschaft in **adParamOutput** ändern. Da Sie Ihre **Parameters**-Auflistung mit Hilfe von Code neu erstellen, sollte dies kein Problem darstellen. Sie könnten natürlich auch die gespeicherte Prozedur umschreiben, um anstelle von OUTPUT-Parametern SELECT-Rowsets zu übergeben ...

Sie werden feststellen, dass leicht Fehler entstehen, da die Dokumentation für das Einrichten der **Parameters**-Auflistung etwas zu wünschen übrig lässt. Nicht nur dies, stellen Sie sich vor, der Datenbankadministrator ändert die betreffende Prozedur, um einem anderen Entwickler die Arbeit zu erleichtern. Wenn Sie den Datentyp für ein bestimmtes Prozedurargument hartcodiert haben und dieses sich in der Datenbank ändert, müssen Sie eine Neucodierung, -kompilierung und eine erneute Bereitstellung vornehmen[11]. Vergessen Sie nicht, dem Systemadministrator die Reifen zu zerstechen, bevor Sie vom Parkplatz fahren.

Parameterbasierte Ad-hoc-Abfragen

Sie müssen weder gespeicherte Prozeduren noch ADO-**Command**-Objekte erstellen, um parameterbasierte Abfragen auszuführen. Stattdessen können Sie eine SQL-Abfrage erstellen, die Fragezeichen (?) als Parametermarken enthält. Diese können von ADO für Sie verarbeitet werden – wenn Sie die Parameter im Code richtig beschreiben. Auf diese Weise sparen Sie Programmierzeit.

Obwohl ADO eine **prototype-Parameters**-Auflistung für Sie erstellen kann, müssen Sie zur Vervollständigung des Prozesses Code hinzufügen. Wenn Sie beispielsweise die Parameter im Code selbst verwalten, müssen Sie weiterhin sicherstellen, dass keiner der zeichenfolgenbasierten Parameter einfache Anführungszeichen enthält (das zusätzliche SQL-Anführungszeichen). Sie müssen außerdem die **ActiveConnection**-Eigenschaft einstellen, bevor ADO die **Parameters**-Auflistung für Sie erstellen kann – ja, Sie haben es bereits geahnt, ADO muss die Datenbank abfragen, um die **prototype-Parameters**-Auflistung zu erstellen. Nachdem Sie die **ActiveConnection**-Eigenschaft gesetzt haben, müssen Sie für jedes **Parameter**-Objekt in der **Parameters**-Auflistung erforderliche Werte für **Type** (Datentyp), **Direction** (der Standardwert lautet **adParamUnknown**), **Precision**, **NumericScale** und **Attributes** festlegen.

10. Sofern Sie nicht für einen kleinen Aufpreis das Addin zum Lesen der Programmierergedanken erworben haben.
11. Wenn Ihre Datenbank in regelmäßigen Abständen neu definiert wird, ist die **Parameters.Refresh**-Methode vorzuziehen.

Automatisches Generieren von Code für die »Parameters«-Auflistung

Sie können zur einfacheren Erzeugung von geeignetem Code für parameterbasierte, gespeicherte Prozeduren ein neues Visual Basic-Addin einsetzen. Dieser Code verbreitete sich nach den Developer Days 1999 in der Entwicklergemeinde. Obwohl durch dieses Addin Ihr Code nicht automatisch an sich ändernde Prozedurparameter angepasst wird, ist es (fast) nicht mehr erforderlich, die Erstellungsweise der **Parameters**-Auflistung im Code vorherzusagen. Durch einfache Angabe der Informationen, die zur Verbindungsherstellung mit der Datenbank erforderlich sind, sowie durch Identifikation der zu referenzierenden gespeicherten Prozedur kann das Addin den benötigten Code zum Erstellen der **Parameters-**Auflistung des **Command**-Objekts für Sie schreiben. Ja, Sie müssen weiterhin sicherstellen, dass die OUTPUT-Parameter im Code richtig codiert sind, aber der Rest funktioniert automatisch. Dieses Addin steht ebenfalls auf der Begleit-CD zur Verfügung.

Tipp Bei der Arbeit mit Active Server Pages (und Visual Basic Script) müssen der zusätzliche Aufwand, der beim Kompilieren des Quellcodes zum Erstellen der **Parameters**-Auflistung entsteht, und die Kosten für das Verwenden der **Refresh**-Methode gegeneinander abgewogen werden. Es kann sich herausstellen, dass die zusätzlichen Serveranforderungen weniger Kosten verursachen als das Kompilieren und Ausführen von vierzig Codezeilen zum Erstellen einer umfangreichen **Parameters**-Auflistung.

5.2.3 Codieren von »Command«-Objekten für gespeicherte Oracle-Prozeduren

Ich glaube, im Zusammenhang mit der Verwendung von **Command**-Objekten zur Ausführung von gespeicherten SQL Server-Prozeduren habe ich jetzt alles gesagt. Wir haben Eingabe-, Ausgabe, Eingabe/Ausgabe-Parameter sowie Parameter für den Rückgabestatus besprochen. Im Kapitel zu den Recordsets werden die Resultsets beschrieben, die durch gespeicherte Prozeduren generiert werden. Mit anderen Worten, dem ist nichts mehr hinzuzufügen – wenigstens nicht hier!

Es gibt jedoch eine Reihe von Problemen, die beim Aufrufen von gespeicherten Oracle-Prozeduren auftreten. Es gibt verschiedene Knowledgebase-Artikel zur Verbindungsherstellung zu Oracle-Datenbanken, und der folgende Abschnitt ist eine Aktualisierung von einem der nützlichsten dieser Artikel.

Zunächst müssen Sie sich die aktuellsten Oracle-ODBC- oder OLE DB-Treiber besorgen. In diesem Bereich hat sich eine Menge getan, daher sollten Sie die Verbesserungen und Bugfixes nutzen, die die aktuellsten Versionen bieten. Ohne diese Treiber ist es unmöglich, Recordsets von gespeicherten Oracle-Prozeduren abzurufen.

Oracle-Entwickler wissen, dass Sie Visual Basic oder Visual Studio nicht ohne Weiteres mit einem Oracle-System einsetzen können, es sind weiterhin die proprietären Treiber erforderlich. Diese werden separat auf dem Client installiert. Da ich kein Oracle-Experte bin, werde ich diese Installation hier nicht erläutern.

Mit der Veröffentlichung des Microsoft ODBC-Treibers für Oracle 2.0 und höher können nun Resultsets von gespeicherten Oracle-Prozeduren abgerufen werden. Durch das Erstellen von gespeicherten Oracle-Prozeduren, die Parameter vom Typ TABLE zurückgeben, können Sie Zeilen- und Spaltendaten zurückgeben, die als Resultset bearbeitet und angezeigt werden können. Der Knowledgebase-Artikel Q174679 verwendet das Beispiel in der Hilfedatei für den Microsoft ODBC-Treiber für Oracle v2.0 und zeigt, wie dieses Beispiel in Visual Basic umgesetzt wird.

Anmerkung Die anhand von gespeicherten Oracle-Prozeduren über den Microsoft ODBC-Treiber für Oracle 2.0 und 2.5 erstellten Resultsets sind READ ONLY und STATIC. Zum Abrufen eines Resultsets muss ein Oracle-Paket erstellt werden.

5.3 Verwalten von »Command«-Objekten

Wie Sie wissen, umfasst die Erstellung und Ausführung von **Command**-Objekten vier Phasen:

1. Erstellen des **Command**-Objekts und Erstellen der **Parameters**-Auflistung.
2. Setzen der **Value**-Eigenschaft für jeden Eingabeparameter.
3. Ausführen der Abfrage.
4. Verarbeiten der Ergebnisse.

Phase 1 sollte nur **einmal** durchlaufen werden. Die verbleibenden Phasen können auf Bedarfsbasis später in der Anwendung ausgeführt werden. Wenn Sie eine Client/Server-Anwendung überarbeiten, in der die erste Phase wiederholt durchlaufen wird, gehen Sie erneut zur Quelle und codieren Sie den Initialisierungsprozess als eine einmalige Ereignisprozedur. Dies ist ebenfalls eine empfohlene Vorgehensweise. Unglücklicherweise muss das **Command**-Objekt im Code für die mittlere Schicht (und ASP) jedes Mal erstellt werden. Denken Sie jedoch daran, dass **Command**-Objekte **nicht** erforderlich sind, es sei denn, Sie führen Abfragen aus, die OUTPUT- oder Rückgabestatuswerte der Abfrage zurückgeben.

5.3.1 Ausführen und erneutes Ausführen von »Command«-Objekten

Nach Erstellung des **Command**-Objekts auf dem Client ist der aufwendige Teil erledigt. In Kapitel 4 wurden verschiedene Möglichkeiten zur Abfrageausführung besprochen, daher ist all dies nicht neu für Sie. Die **Command**-Abfrage kann auf verschiedene Weise ausgeführt werden:

- Verwenden der **Execute**-Methode für das erstellte **Command**-Objekt.
- Verwenden des **Command**-Objekts als Methode des **Connection**-Objekts.
- Referenzieren des Befehls mit der **Execute**-Methode für das **Connection**-Objekt.
- Referenzieren des Befehls mit der **Open-**Methode für das **Recordset**-Objekt. Diese Vorgehensweise wird in Kapitel 6 besprochen, dessen Schwerpunkt die Erstellung von Recordsets darstellt.

Sehen wir uns diese Techniken im Einzelnen an. Einer der Hauptunterschiede besteht in der Art und Weise, in der das Recordset erstellt wird (wird dieses vorzeitig oder unter Verwendung des Standardwertes erstellt?) und in der Art, mit der die Parameter an das **Command**-Objekt übergeben werden (müssen diese Parameter für Parameter angegeben werden oder können sie als Methodenargumente übergeben werden?).

Verwenden der »Execute«-Methode für ein »Command«-Objekt

Eine der einfachsten und unflexibelsten Methoden stellt das einfache »Ausführen« des **Command**-Objekts dar. Bei diesem Ansatz wird ADO angewiesen, das **Command**-Objekt wie derzeit gefüllt auszuführen und (optional) ein neues **Recordset**-Objekt zum Empfangen des Rowsets zu erstellen. Die Verwendung der **Execute**-Methode für ein **Command**-Objekts ähnelt grundsätzlich der Verwendung für ein **Connection**-Objekt (wie in Kapitel 4 besprochen).

Das folgende Codebeispiel veranschaulicht die Verwendung der **Execute**-Methode. (Wir werden dieses **Command**-Objekt für die verbleibenden Beispiele zur Befehlsausführung verwenden.) Der grundlegende Unterschied besteht darin, dass ADO bei Verwendung der **Execute**-Methode ein »jungfräuliches« Recordset erstellt, dass zur Handhabung des Rowsets auf **ReadOnly/ForwardOnly** gesetzt wird. Wenn Sie die Vorgehensweise »**Command**-Objekt als **Connection**-Methode« verwenden, können Sie im Voraus ein eigenes Recordset erstellen. Obwohl mit dem standardmäßig über die **Execute**-Methode erstellten Recordsets Zeilen effizient abgerufen werden können, dürfen Sie bei dieser Vorgehensweise keine Aktualisierbarkeit erwarten.

```
Set cmd = New Command
With cmd
   .Name = "GetTitles"
   .ActiveConnection = cn
   .CommandText = "Select title from titles " _
      & "where title like ? " _
      & "and year_published between ? and ?"
   .Parameters.Append CreateParameter("TitleWanted", adVarChar, adParamInput, 20)
   .Parameters.Append CreateParameter("YearLow", adInteger, adParamInput,, 1940)
   .Parameters.Append CreateParameter("YearHigh", adInteger, adParamInput)
   .Prepared = False
End With
```

Handhabung von Standardparameterwerten Beachten Sie, dass im vorgenannten Beispiel der zweite Parameter in der **Parameters**-Auflistung auf den Wert (1940) eingestellt wurde. Hierdurch wird ADO angewiesen, den Parameter nach der Verwendung des **Command**-Objekts auf diesen Wert zurückzusetzen. Nach der Ausführung eines **Command**-Objekts setzt ADO sämtliche Parameterwerte (**Parameter.Value**) auf die anfänglichen Einstellungen zurück. Im Falle des ersten und dritten Parameters wird der Wert auf »leer« zurückgesetzt.

Jetzt, nachdem **Command** eingerichtet wurde, können wir eine Ausführung vornehmen. Die folgende Zeile könnte weiterhelfen.

```
cmd.Execute
```

Dieser Code ignoriert den Wert für die betroffenen Zeilen – es wird für diese Abfrage sowieso nichts zurückgegeben – und akzeptiert die vorhandenen Parameterwerteinstellungen. Uups, da diese nicht gesetzt waren, erhalten wir einen auffangbaren Fehler. Wir sollten zunächst die Parameterwerte setzen:

```
cmd(0) = "Hitch%"
cmd(1) = 1940
cmd(2) = 1990
cmd.Execute
```

Jetzt funktioniert die **Execute**-Methode prima. Nach Befehlsausführung werden die Parameter auf ihren anfänglichen Status zurückgesetzt – leer, 1940 und leer.

Übergeben von Argumenten mit »Variant«-Arrays Eine weitere Möglichkeit bei der Übergabe von Eingabeparametern stellt die Verwendung von **Variant**-Arrays dar. Dieser Ansatz ist ziemlich cool. Und noch besser, wenn Sie eines der Elemente nicht angeben, übermittelt ADO diesen Parameter nicht – es wird davon ausgegangen, dass der Provider den Standardwert einfügt. Ist kein Stan-

dardwert gesetzt, wird entweder in der aufgerufenen gespeicherten Prozedur oder in der anfänglichen **Parameters**-Auflistung über ADO ein auffangbarer Fehler ausgegeben: »-2147217900 Falsche Syntax im Bereich des Schlüsselwortes 'DEFAULT'.«

```
vParms = Array("Hitch%", 1940, 1990)
cmd.Execute lRA, vParms
```

Lassen Sie uns eine andere Variante ausprobieren:

```
cmd.Execute lRA, Array("Hitch%", 1940, 1990)
```

Bei dieser Vorgehensweise wird die Erstellung eines separaten **Variant**-Arrays umgangen und einfach die neue Visual Basic-Funktion **Array** verwendet. In diesem Fall lassen wir die Verwendung der vorhandenen Einstellung für den zweiten Parameter (**cmd(1)**) zu, da dieser einen Standardwert aufweist. Bei dieser Variante können jedoch die OUTPUT-Parameter nicht aus dem Array ausgelesen werden.

Wenn Sie eine Aktionsabfrage ausführen, ist es gut zu wissen, wie viele Zeilen betroffen sind. Kann Ihnen der Provider diese Information zur Verfügung stellen, wird der Wert als erstes Argument in der **Long**-Variable übergeben, **RecordsetsAffected**. Die Anzahl der Records im Rowset wird nicht zurückgegeben. Diese wird an die Routinen in der **IRA**-Variable übergeben.

Einstellen zusätzlicher Optionen zur Ausführungszeit Bei der Definition des **Command**-Objekts wurde die **CommandType**-Eigenschaft angegeben, daher sollte diese nicht erneut angegeben werden müssen, wenn ein **Command**-Objekt ausgeführt wird, möglich ist es jedoch. Es gibt weitere interessante Optionen, die bei Verwendung der **Execute**-Methode zur Verfügung stehen. Hierzu zählen Optionen, die ADO u.a. mitteilen, dass Sie keine Zeilen erwarten, dass eine asynchrone Ausführung erfolgen soll usw., wie gezeigt in Tabelle 5.3.

Konstante	Beschreibung
adAsyncExecute	Gibt an, dass der Befehl asynchron ausgeführt werden sollte. Auf diese Weise erfolgt eine sofortige Rückgabe der Steuerung an die Anwendung oder Komponente (sobald die ersten **CacheSize**-Zeilen abgerufen wurden), um Ihren Thread für andere Aufgaben freizugeben. Es macht nicht viel Sinn, diese Option für die mittlere Schicht einzusetzen, kann bei Client/Server-Systemen jedoch zu einer merklichen Leistungssteigerung führen.

Tabelle 5.3 »CommandType«-Optionen

Konstante	Beschreibung
adAsyncFetch	Weist ADO an, die nach dem anfänglichen Abrufen der in CacheSize festgelegten Zeilenmenge verbleibenden Zeilen asynchron abzurufen. Ist diese Option eingestellt, setzt ADO den Zeilenabruf fort, sodass die Auffüllung des Rowsets schneller erfolgt. Dies ist wichtig für das Gesamtsystem und die Anwendung, da Zeilen nach dem anfänglichen Cache schneller verfügbar sind. Der asynchrone Abruf wird in Kapitel 6 besprochen.
adAsyncFetchNonBlocking	Verhindert, dass der Hauptthread während des Abrufs blockiert wird. Wurde die angeforderte Zeile nicht abgerufen, findet eine automatische Verschiebung von der aktuellen Zeile zum Ende des Resultsets statt (EOF, End of File).
adExecuteNoRecords	Gibt an, dass der Befehlstext einen Befehl oder eine gespeicherte Prozedur bezeichnet, mit dem/der **keine** Zeilen zurückgegeben werden (beispielsweise ein Befehl, durch den lediglich Daten eingefügt werden). Werden Zeilen abgerufen, werden sie verworfen und nicht zurückgegeben. Diese Option wird stets mit den **Command**Type-Optionen von adCmdText oder adCmdStoredProc kombiniert. Diese Option hindert ADO daran, ein Recordset-Objekt zu erstellen, wenn dies nicht erforderlich ist.

Tabelle 5.3 »CommandType«-Optionen

Durch Verwenden des richtigen Optionensatzes unterstützen Sie nicht nur ADO dabei, eine möglichst effiziente Abfrageausführung vorzunehmen, sondern sorgen auch dafür, dass der Zeilenabruf möglichst effizient durchgeführt wird. Sie müssen möglicherweise verschiedene Optionen »zusammenfügen«, wie das folgende Codebeispiel zeigt:

```
cn.Execute("MyUpdateSP", , adCmdStoredProcedure + adExecuteNoRecords)
```

Nach (erfolgreichem oder erfolglosem) Ausführen des **Command**-Objekts wird ein **ExecuteComplete**-Ereignis ausgelöst.

Ausführen eines »Command«-Objekts als »Connection«-Methode

Da wir auch über die **Entwicklerleistung** sprechen, sollte die Vorgehensweise »**Command**-Objekt als **Connection**-Methode« erläutert werden. Bei dieser Technik wird ein benanntes **Command**-Objekt als Methode des verknüpften ADO-**Connection**-Objekts eingesetzt. Hierbei kann nach Benennung eines **Command**-Objekts (durch Setzen der zugehörigen **Name**-Eigenschaft) und Einstellen von **ActiveConnection** auf ein **Connection**-Objekt zur Ausführung die folgende Syntax verwendet werden:

```
Private Sub cmdRunQuery_Click()
   cnMyConnection.GetAuthors txtYearWanted.Text, rs
       ' Parameter,  Recordset
   ProcessResults(rs)
End Sub
```

In diesem Fall wird das **Command**-Objekt namens »GetAuthors« ausgeführt. Beachten Sie, dass die Eingabeparameter als Argumente der Methode übergeben werden. Das letzte Argument wird als Verweis auf ein instanziiertes **Recordset**-Objekt übergeben. Dieser Verweis enthält das Rowset, wenn ADO die Abfrageausführung beendet hat.

Tipp Aus Gründen der Funktionsweise von COM[12] **müssen** Sie bei Einsatz dieser Technik alle Objektquellenparameter vollständig angeben. Dies bedeutet, dass Sie nicht einfach **txtYearWanted** übergeben können, dass sich auf die standardmäßige **Text**-Eigenschaft des referenzierten Steuerelements **TextBox** beziehen sollte. Stattdessen müssen Sie **txtYearWanted.Text** übergeben. Dieses Vorgehen verhindert merkwürdige Verhaltensweisen, beispielsweise die Übergabe seltsamer Parameter an den Datenprovider.

Die Technik »**Command-**Objekt als **Connection**-Methode« ist darüber hinaus sehr einfach zu codieren und bietet exzellente Leistungsergebnisse. In diesem Fall arbeiten Sie mit einem zuvor erstellen **Recordset**-Objekt, das mit korrektem **CursorType**, **LockType** und weiteren sinnvollen Eigenschaften konfiguriert wurde. Im Gegensatz zur **Execute**-Methode des **Command**-Objekts (diese wird als nächste besprochen), wird die Erstellung des **Recordset**-Objekts beschrieben.[12]

Warnung Es gibt einen bisher nicht behobenen Bug, der sich bei Verwendung dieser Technik negativ auswirkt. Weitere Informationen finden Sie im Abschnitt »Nachteile der **Refresh**-Methode« weiter oben in diesem Kapitel.

Verwenden der »Execute«-Methode des »Command«-Objekts

Die **Execute**-Methode des **Command**-Objekts ist im Vergleich zu anderen Techniken weit weniger flexibel. Darüber hinaus hat diese Vorgehensweise einige, naja, interessante Nebeneffekte, derer Sie sich vielleicht nicht bewusst sind. Es gibt zwei Formen der **Execute**-Methode:

12. Alle Parameter werden in einer DISPARAMS-Struktur reiner **variant**-Werte gespeichert, Steuerelementverweise sind jedoch gültige **variant**-Werte, daher übergibt Visual Basic einfach diese – und ADO erstickt.

- Für ein **Connection**-Objekt: In diesem Fall geben Sie ein **Source**-Argument an, das Angeben eines **Command**-Objekts ist jedoch – im Vergleich zur **Open**-Methode des **Recordset**-Objekts – nicht möglich (dieses wird in Kapitel 6 besprochen). Sie müssen die SQL-Abfrage als SQL-Anweisung, Tabellenname, URL oder Name einer gespeicherten Prozedur übergeben. Sie müssen im **Options**-Argument außerdem einen geeigneten **CommandType** übergeben – oder ADO diese Entscheidung überlassen. Sie sollten jedoch immer den **CommandType** angeben, um ADO daran zu hindern, die Art der übermittelten Abfrage zu »raten«.
- Für ein **Command**-Objekt: In diesem Fall ist das Quellargument, wie alle weiteren zur Abfrageausführung benötigten Argumente, im **Command**-Objekt enthalten. Hierzu zählt auch **CommandType**. Erneut erstelllt ADO das Recordset für Sie, daher müssen die standardmäßigen Recordseteigenschaften festgelegt werden – schreibgeschützt (RO, Read-Only), vorwärts (FO, Forward-Only).

In beiden Fällen muss zur Rückgabe eines Rowsets das Recordset erfasst werden, dass von der **Execute**-Methode zurückgegeben wird. Beachten Sie, dass die **Execute**-Methode immer ein Recordset mit Standardeinstellungen (RO, FO) für Sie erstellt. Beispiel:

```
Set rs = cmd.Execute
```

Wenn Sie kein Recordset benötigen, verwenden Sie alternativ diese Syntax:

```
cmd.Execute
```

Hierbei gibt es einen interessanten Nebeneffekt. In einigen Fällen, und besonders bei der Erstellung eines Recordsets mit Hilfe der **Execute**-Methode eines **Connection**-Objekts, öffnet ADO eine **zusätzliche** Verbindung zur Abfrageausführung, die nach deren Ausführung wieder geschlossen wird. Dies gilt besonders für das erneute Verwenden der **Execute**-Methode vor dem Auffüllen des ursprünglichen Recordsets.

Tatsächlich wird dieses Verhalten durch die Reihenfolge der Operationen hervorgerufen. Wenn Sie beispielsweise **cmd.ActiveConnection** auf **cn** setzen und die **cmd.Execute**-Methode gefolgt von **cn.Execute** einsetzen, öffnet ADO keine neue Verbindung. Wenn Sie jedoch die **cn.Execute**-Methode gefolgt von der **cmd.Execute**-Methode ausführen, erstellt ADO zur Befehlsausführung eine weitere Verbindung.

5.3.2 Ablaufverfolgung für Server und Datenprovider

Wenn Sie zur Abfrageausführung ADO einsetzen, erfüllt der Server pflichtbewusst die Anweisungen von ADO und Datenprovider. Wie effizient diese Anweisungen jedoch ausgeführt werden, und wie sich die Anweisungen auf die Skalierbarkeit

auswirken, ist ein wichtiger Punkt. Die Werte für **CursorLocation** (clientseitiger oder serverseitiger Cursor), **CursorType**, **LockType** und selbst die verwendete Syntax können sich darauf auswirken, welche Art Abfrage an das Backend gesendet wird. Da nur SQL Server genügend Informationen zur Ablaufverfolgung bereitstellt, um tatsächlich ermitteln zu können, was an den Server übermittelt wird, werde ich zur Veranschaulichung eine Reihe von Konfigurationen verwenden, damit Sie den geeignetsten Abfragetyp für Ihre Anwendung oder Komponente auswählen können. Sie sollten zumindest verstehen, welche Faktoren Einfluss auf die an SQL Server gesendeten Befehle nehmen.

Anmerkung Ja, Sie können auch ein Dump der ODBC-Protokolle erstellen, um sich ein grobes Bild davon zu machen, welche Anweisungen von der oberen ODBC-Schicht an den Treiber übermittelt werden. Es ist jedoch nicht wirklich eine Möglichkeit, zu zeigen, wie der Datentreiber den ODBC-Aufruf implementiert. Diese Aufgabe kann über den SQL Server Profiler (oder SQLTrace) ausgeführt werden. Hierbei kann die Funktionsweise des OLE DB-Providers »hinter den Kulissen« nicht aufgezeigt werden.

Wir werden die verschiedenen Möglichkeiten einzeln durchgehen. Zunächst sehen wir uns serverseitige Cursor mit den standardmäßigen Recordseteinstellungen an, RO/FO (Read-Only, Forward-Only). Diese werden in Tabelle 5.4 aufgeführt. Tabelle 5.5 listet die serverseitigen Cursor mit Schlüsselgruppen- bzw. optimistischen Cursorn auf. Beachten Sie den auffallenden Unterschied bei der Anzahl der zusätzlichen Serverabrufe und der Komplexität der Abfragen, die der Server verarbeiten muss, um im Grunde das gleiche **Command**-Objekt (abgesehen von einer abweichenden Syntax zur Unterstützung des jeweiligen Cursortyps) auszuführen.

Befehlssyntax	Erzeugte Abfrage
`Cn.GetTitlesByYear`	GetTitlesByYear 'Hitch%', 1900, 1999
`rs.Open cmd`	**sp_executesql** N'Select Top 50 title, ISBN, Year_Published from titles where title like @P1 and year_published between @P2 and @P3', N'@P1 varchar(20),@P2 int,@P3 int', 'Hitch%', 1900, 1999
`rs.Open strSQL`	Select Top 50 title, ISBN, Year_Published from titles where title like 'Hitch%' and year_published between 1900 and 1999
`Cn.GetTitles`	Sp_executesql N'Select Top 50 title, ISBN, ...
`Set rs = cmd.Execute…`[13]	Sp_executesql N'Select Top 50 title, ISBN, …

Tabelle 5.4 Serverseitige RO/FO-Cursor

13. Das Verwenden der Command.Execute-Methode ist nur sinnvoll mit FO/RO, da das Recordset bei jeder Auslösung durch ADO erstellt wird.

Befehlssyntax	Erzeugte Abfrage
Cn.GetTitlesByYear	declare @P1 int declare @P3 int declare @P4 int declare @P5 int set @P1=NULL set @P3=102401 set @P4=311300 set @P5=NULL exec sp_cursoropen @P1 output, N'Select Top 50 title, ISBN, Year_Published from titles where title like @P1 and year_published between @P2 and @P3', @P3 output, @P4 output, @P5 output, N'@P1 varchar(20),@P2 int,@P3 int', 'Hitch%', 1900, 1999 select @P1, @P3, @P4, @P5
rs.Open cmd	Sp_cursoropen … sp_cursorfetch … (mehrfach)
rs.Open strSQL	Sp_cursoropen…sp_cursorfetch (mehrfach)
Cn.GetTitles [14]	sp_cursoropen … (mehrfach) sp_cursorfetch … (mehrfach)

Tabelle 5.5 Serverseitige Schlüsselgruppen- bzw. optimistische Cursor

Clientseitige Cursor sind weitaus unflexibler. In diesem Fall wird nur der Cursortyp **Static** unterstützt, und Aktualisierbarkeit ist nur gegeben, wenn Sie als **LockType** die Einstellung **adLockBatchOptimistic** wählen. Nein, ich erwarte nicht, dass Sie diese Tabellen ganz genau studieren. Ich erwarte, dass Sie diese Tests selbst durchführen. Der Quellcode zur Einrichtung dieser Tests befindet sich auf der Begleit-CD. Sie finden ihn im Verzeichnis **\sample application\Command objects**. In Tabelle 5.6 werden clientseitige statische/optimistische Cursor gezeigt, Tabelle 5.7 enthält clientseitige, optimistische statische bzw. Stapelaktualisierungscursor.

Befehlssyntax	Erzeugte Abfrage
rs.Open cmd	sp_executesql…
cmd.Execute	sp_executesql …
Cn.GetTitlesByYear	GetTitlesByYear 'Hitch%', 1900, 1999
Cn.GetTitles	sp_executesql…
rs.Open strSQL	Select Top 50 title, ISBN, Year_Published from …

Tabelle 5.6 Clientseitige statische/optimistische Cursor

Befehlssyntax	Erzeugte Abfrage
rs.Open cmd	sp_executesql…
cmd.Execute	sp_executesql …

Tabelle 5.7 Clientseitige, optimistische statische bzw. Stapelaktualisierungscursor

14.Es gibt kontroverse Meinungen zu dieser Technik. Die ADO-Leute arbeiten noch daran.

Befehlssyntax	Erzeugte Abfrage
Cn.GetTitlesByYear	GetTitlesByYear 'Hitch%', 1900, 1999
Cn.GetTitles	sp_executesql…
rs.Open strSQL	Select Top 50 title, ISBN, Year_Published from …

Tabelle 5.7 Clientseitige, optimistische statische bzw. Stapelaktualisierungscursor

Wie Sie sehen können, gibt es verschiedene Ansätze für den SQL Server-Datenprovider, wenn im Namen von ADO Abfragen ausgeführt werden. ADO scheint jedoch ziemlich vorhersehbar zu sein, unabhängig von der Cursoreinstellung. Wenn Sie clientseitige Cursor verwenden und mit Hilfe der **Recordset Open**-Methode eine Ad-hoc-Abfrage oder eine gespeicherte Prozedur ausführen, nimmt ADO einfach die SQL und »gibt sie durch«, d.h., es wird nicht der Versuch unternommen, eine der **sp_cursor**-Funktionen des Systems zu nutzen oder gar die **sp_executesql**-Funktion zu verwenden. Dies ist nicht unbedingt falsch. SQL Server weiß, wie Abfragepläne für Ad-hoc-Abfragen gespeichert werden, und die Abfragepläne für SQL Server sind bereits kompiliert.

Das Aufrufen gespeicherter Prozeduren scheint weitaus effizienter zu sein als die Verwendung von Ad-hoc-Abfragen oder **Command**-Objekten. Wenn Sie bei FO/RO bleiben, übergibt ADO lediglich die Auslösung der gespeicherten Prozedur an den Server und verwaltet hierbei sogar die Parameter. Wenn Sie einen aktualisierbaren Cursor anfordern, versieht ADO den gespeicherten Prozeduraufruf mit **sp_cursoropen**, um das Scrolling (rollbare Cursor) und Aktualisierbarkeit handhaben zu können.

5.3.3 Ermitteln des Befehlsstatus

Die **State**-Eigenschaft des **Command**-Objekts ist erst dann wirklich interessant, wenn Sie eine Abfrage asynchron ausführen. Wenn Sie also ein **Command**-Objekt mit Hilfe einer der zuvor besprochenen Methoden ausführen, kann über die **State**-Eigenschaft angezeigt werden, ob die Abfrage ausgeführt wurde oder nicht. Solange ADO damit beschäftigt ist, das **Command**-Objekt auszuführen, wird das Ausführungsbit (**adStateExecuting**) der **State**-Eigenschaft (4) aktiviert und das Bit **adStateOpen** ist deaktiviert. Ja, Sie können mit Hilfe des **adStateExecuting**-Bits darauf warten, dass eine asynchrone Operation abgeschlossen wird. Ich würde in diesem Fall jedoch das **ExecuteComplete**-Ereignis vorziehen. Der Einsatz eines Ereignisses beansprucht im Vergleich zur Abfrage weniger CPU-Ressourcen, besonders im Vergleich zur Schleifenabfrage. Diese ähnelt einem kleinen Kind, das auf dem Weg zur Großmutter auf dem Rücksitz Ihres Autos herumzappelt und (alle zwei Minuten) fragt: »Sind wir schon da, Papi?«

5.3.4 Die »Cancel«-Methode

Sie haben also die Ausführung Ihres **Command**-Objekts mit der Option **adAsyncExecute** gestartet und so die Steuerung Ihres Threads zurückgegeben, damit der Benutzer in der Wartezeit mit einem Fortschrittsbalken oder Ages of Empire II unterhalten werden kann. Wenn der Benutzer sich dafür entscheidet, dass er nicht länger warten möchte, können Sie die Abfrageverarbeitung mit Hilfe der **Cancel**-Methode abbrechen – Sie können es zumindest versuchen. Gelegentlich ähnelt dies dem Versuch, einen Zug 15 Meter vor einem Bahnübergang zu stoppen.

> **Anmerkung** Die **Cancel**-Methode kann auch auf das **Connection**-Objekt angewendet werden, um eine asynchrone Öffnungsoperation zu stoppen, aber dies ist weniger oft erforderlich – die meisten Verbindungen werden aufgebaut, bevor der Benutzer auch nur zwinkern kann, und wenn die Verbindungsherstellung länger dauert, liegt dies üblicherweise an einer Netzwerkverzögerung, die sowieso nicht verhindert werden kann.

Sie erhalten einen auffangbaren Fehler, wenn die Operation nicht unterbrochen werden kann, vergleichbar mit dem Versuch, eine Operation abzubrechen, die nicht asynchron gestartet wurde.

Beachten Sie, dass es möglicherweise keine sichere Methode ist, eine derzeit ausgeführte Operation abzubrechen – besonders dann nicht, wenn Änderungen an der Datenbank vorgenommen werden. SQL Server und andere Provider machen bereits vorgenommene Änderungen nicht (unbedingt) wieder rückgängig, sofern im Vorfeld keine Transaktion gestartet wurde.

5.3.5 Plan B für ADO-»Command«-Objekte

Plan B für benannte ADO-**Command**-Objekte umfasst gespeicherte Prozeduren. Dies bedeutet Folgendes: Wenn Sie kein **Command**-Objekt erstellen und neben dem **Connection**-Objekt auf ein benanntes Objekt verweisen (wie hier):

```
cnOLEDB.TestQuery txtParm, rs
```

geht ADO davon aus, dass es sich bei **TestQuery** um eine gespeicherte Prozedur handelt. Ist dies nicht der Fall, erhalten Sie einen Fehler mit einer Beschwerde, dass die gespeicherte Prozedur **TestQuery** nicht gefunden werden konnte. Das gleiche geschieht, wenn Sie **adCmdText** als **Command.CommandType**-Eigenschaft verwenden und einen einzelnen Objektnamen übermitteln.

Dies bedeutet, dass Sie eine beliebige gespeicherte Prozedur (Parameter usw.) mit Hilfe der folgenden Syntax ausführen (vorausgesetzt, **GetTitles** ist eine gespeicherte Prozedur):

```
Cn.GetTitles "1980", rsMyRecordset
```

Das Problem bei der Verwendung von gespeicherten Prozeduren als **Connection**-Methode besteht darin, dass die OUTPUT-Parameter nicht gehandhabt werden können. Wenn also die gespeicherte Prozedur über OUTPUT-Parameter verfügt, werden diese bei Einsatz dieser Technik nicht an Sie zurückgegeben – es wurde kein **Command**-Objekt zur Behandlung dieser erstellt. Sie können die gespeicherte Prozedur ausführen, aber Sie müssen Platzhalter für jeden OUTPUT-Parameter übergeben, da ADO die Richtung der Ausgabeparameter (**OUTPUT Parameter.Direction**) standardmäßig auf INPUT/OUTPUT (**adParamInputOutput**) setzt. Nach Ausführung der Abfrage steht kein **Command**-Objekt zum Abrufen der Rückgabeparameter zur Verfügung.

5.3.6 Tipps und Warnungen zum »Command«-Objekt

Bei der Arbeit mit dem **Command**-Objekt zur Entwicklung von Beispiel- und Testcode stieß ich auf verschiedene Tipps. Eine Zusammenfassung dieser Tipps und Vorgehensweisen führe ich nachfolgend auf. Ich habe hier auch einige Vorschläge mit einbezogen, die von meinen Studenten und den Personen stammen, mit denen ich per E-Mail kommuniziere.

- Wenn Sie versuchen, eine gespeicherte Prozedur auszuführen, und die beschriebenen Parameter stimmen nicht mit den Erwartungen von SQL Server überein, kann alles Mögliche passieren, einschließlich der Anzeige des folgenden Fehlers: »-2147418113 Unerwarteter Fehler«.
- Wenn Sie die **CommandType**-Eigenschaft auf **adCmdStoredProc** setzen und statt einer gespeicherten Prozedur eine Tabelle referenzieren, wird der folgende auffangbare Fehler erzeugt: »-2147217900 – Die Anforderung der Prozedur 'Publishers' schlug fehl, da 'Publishers' ein Tabellenobjekt ist«. Es wird klar, dass ADO Sie beobachtet. Dem Profiler konnte nicht entnommen werden, dass zusätzliche Serverabfragen ausgeführt wurden, obwohl diese tatsächlich stattgefunden haben. Diese Informationen wurden jedoch möglicherweise zwischengespeichert.

Wenn das an die **Command**-Objektparameter übergebene Argument nicht den Datentypanforderungen entspricht (**string**, **number**, **integer**, **date**, **currency** usw.), wird durch den Code Fehler 3421 ausgelöst: »Die Anwendung verwendet für die aktuelle Operation einen Wert vom falschen Typ«. Wer schreibt eigentlich diese Fehlermeldungen?

- Wird der **CommandType**-Wert **adCmdUnknown** beibehalten und enthält die Abfrage eine SELECT-Anweisung, wird der Befehlstyp über ADO ermittelt und auf **adCmdText** gesetzt.
- Lautet der **CommandType**-Wert **adCmdTable**, führt jeder Eintrag im Befehlstext, bei dem es sich nicht um einen Tabellennamen handelt, zum folgenden Syntaxfehler: »-2147217900 Falsche Syntax im Bereich des Schlüsselwortes

'select'«. Der ODBC-Provider dagegen spürt keine falschen Tabellennamen auf. Ich habe mit Hilfe von **adCmdTable** und dem Befehlstext »Fred« ein **Command**-Objekt erstellt, und als ADO »SELECT * FROM FRED« übermittelte, erfolgte keine Beschwerde durch ADO. Der OLE DB-Provider dagegen gab den folgenden Fehler zurück: »-2147217865 Ungültiger Objektname 'fred'«. Interessant.

Anmerkung Vom OLE DB-Provider zurückgegebene Fehlermeldungen werden ohne Angabe zur fehlererzeugenden Schicht zurückgegeben. Die ODBC-Meldungen werden folgendermaßen angezeigt:

```
[Microsoft][ODBC SQL Server Driver][SQL Server] Ungültiger Objektname
'fred'.
```

5.3.7 Ereignisbehandlung für »Connection«-Objekte

Dies ist einfach – für das **Command**-Objekt werden keine Ereignisse offen gelegt. Sie müssen zur Behandlung von Ereignissen im Zusammenhang mit dem **Command**-Objekt das **Connection**- oder **Recordset**-Objekt heranziehen. Wenn Sie ein **Command**-Objekt asynchron ausführen, können Sie das **WillExecute**- oder **ExecuteComplete**-Ereignis für das verknüpfte **Connection**-Objekt dazu verwenden, die Ausführungsbeendigung zu ermitteln oder die **State**-Eigenschaft des **Command**-Objekts abzufragen.

5.4 Überlegungen hinsichtlich der Leistung: Sinnvoller Einsatz von »Command«-Objekten

Eines der häufigsten Leistungsprobleme mit **Command**-Objekten besteht in der Instanziierung (Erstellung), Ausführung und Zerstörung der Objekte mit einem einzigen **Command_Click**-Ereignis, über eine Komponente der mittleren Schicht oder auf einer Webseite. Dies ist ein Problem, da in diesen Fällen die Kosten für die Objekteinrichtung nicht nur einmalig, sondern bei jeder Befehlsausführung anfallen. Da Sie nicht erwarten können, eine Anwendung oder Komponente so zu skalieren, dass **Command**-Objekte praktisch herumliegen und auf potenzielle Abnehmer warten, müssen diese bei Bedarf erstellt werden. Obwohl es sicherlich Fälle gibt, in denen ein **Command**-Objekt erstellt werden muss, möchten Sie im Allgemeinen Ressourcen nur dann zur Objektinstanziierung und -vorbereitung einsetzen, wenn dies erforderlich ist. In der mittleren Schicht (Microsoft Transaction Server oder ASP) haben Sie hierbei allerdings oft keine Wahl. Daher sollten Sie in solchen Situationen Ihre Abfragestrategie genau überdenken und ermitteln, ob es tatsächlich nötig ist, ein **Command**-Objekt immer dann zu erstellen/zu verwenden/zu zerstören, wenn die Methode oder die ASP-Seite referenziert werden – insbesondere dann, wenn günstigere Alternativen verfügbar sind. Obwohl mit

Hilfe von **Command**-Objekten Prozedurparameter einfacher verwaltet werden können, wiegt der zusätzliche Overhead bei deren Erstellung die bessere Leistung möglicherweise nicht auf.

Wir gingen immer davon aus, dass der fundamentale Zweck des **Command**-Objekts in der Unterstützung von ADO bei der Erstellung temporär gespeicherter Prozeduren zum schnelleren Ausführen von Abfragen liegt. Es hat sich jedoch gezeigt, dass ADO den Zweck dieses Ansatzes der Erstellung und Ausführung temporär gespeicherter Prozeduren dahingehend geändert hat, SQL Server und andere Provider bei der effizienteren Handhabung wiederholter Ad-hoc-Abfrageauslösungen zu unterstützen.

Die **Execute**-Methode des **Command**-Objekts erstellt darüber hinaus das **Recordset**-Objekt für Sie. Daher können Sie **LockType**, **CursorType**, **CacheSize**, **MaxRecords** oder weitere interessante (aber leistungsbeeinträchtigende) Recordseteigenschaften nicht im Voraus festlegen – Sie müssen sich mit den Standardeinstellungen zufrieden geben. Das standardmäßige Recordset ist ein schreibgeschütztes (RO), vorwärts gerichtetes (FO), sehr effizientes Resultset mit **CacheSize=1** (cursorlos), dessen Verhalten einer Hochleistungsanwendung nahe kommt. Wenn Sie jedoch das **Options**-Argument für die **Execute**-Methode codieren, können Sie ADO mitteilen, dass mit der Abfrage keine Zeilen zurückgegeben werden (wie gleich gezeigt wird), und dies führt zu einer verbesserten Leistung.

```
Set rs = cmd.Execute(Options:=adExecuteNoRecords)
```

Jegliche Informationen, die Sie ADO hinsichtlich der Verarbeitung des Inhalts der **CommandText**-Eigenschaft mitteilen können, tragen zu einer Leistungssteigerung bei. Wie bei der **Open**-Methode des **Recordset**-Objekts, das in Kapitel 6 behandelt wird, müssen Sie ADO mitteilen, ob es sich um den Namen einer gespeicherten Prozedur, um eine Tabelle, eine Datei oder um eine wahlfreie SQL handelt. Nein, **adCmdTableDirect** ist für die meisten Provider nicht zulässig. Der standardmäßige Wert für **CommandType** lautet **adCmdUnknown** – bei dieser Einstellung »rät« ADO, und dies beansprucht Zeit.

In vielen Fällen kann die Leistung durch das Vermeiden von zusätzlichen Serverabfragen durch den Client gesteigert werden. Wenn Sie beispielsweise eine Reihe zueinander in Beziehung stehender Operationen in einer einzelnen Abfrage zusammenfassen, können Sie den System- und Backendoverhead verringern. Es unterstützen jedoch nicht alle Provider Mehrfachoperationen – SQL Server bietet diese Unterstützung, Jet/Access nicht. Ich verwende diese Vorgehensweise, um Masseneinfügevorgänge oder eine Reihe von in einer Transaktion gebundenen Befehle auszuführen. Sie müssen hierbei mit den komplexeren Resultsets umgehen können, die bei Einsatz dieser Technik von ADO erzeugt werden. Aber das ist nicht schwer.

Das Kapitel fehlt in dieser PDF-Datei.

Sorry, aber wir wollen ja auch das eine oder andere Buch verkaufen :-)

9 Webbasierte Lösungen

Für viele von Ihnen ist die Webentwicklung möglicherweise etwas völlig Neues oder eine Fertigkeit, die Sie noch erlernen möchten. Im Gegensatz zu Al Gore, der das Internet wohl vor einiger Zeit erfunden hat und alles darüber weiß, gehöre ich zu der Gruppe von Leuten, die den Umgang mit dem Web auf die harte Tour erlernen mussten – über das Trial-and-Error-Prinzip und eine Menge einschlägiger Literatur. Dieses Kapitel setzt kaum Vorkenntnisse voraus. Es beginnt mit der grundlegenden Webentwicklungsarchitektur und erläutert die zugehörige Terminologie sowie die andersartige Funktionsweise von ADO in einer ASP (ActiveX Server Page).

Microsoft gibt für neue Webentwicklungstools eine Menge Geld aus – fast so viel wie in letzter Zeit für Rechtsanwälte. Wir werden Visual Studio 7.0 noch bestaunen (irgendwann) und wie es Code der mittleren Schicht nutzt, der in einer Visual Basic-ähnlichen Sprache geschrieben ist[1], die HTML-, XML- und XSL-Skripts wiederverwertet. Nach der Lektüre dieses Kapitels werden Sie idealerweise wissen, wie ADO und diese neuen, aus dem Boden sprießenden Tools optimal genutzt werden können – zumindest stehen Sie den innovativen Möglichkeiten nicht mehr mit Scheuklappen gegenüber.

9.1 Webentwicklung 101

Diejenigen unter Ihnen, die mit dem Web bereits gearbeitet haben und mit den Grundlagen vertraut sind, können mit dem nächsten Abschnitt fortfahren. Ich werde versuchen, die Webentwicklung denjenigen zu erklären, die sich gerade erst langsam an dieses neue Prinzip gewöhnen.

Im Grunde vollzieht sich die Webentwicklung in zwei Formen. Erstens beschäftigen sich Programme wie Front Page und andere mit der **Darstellung** von Informationen. Mit diesen werden Hintergrund, Schaltflächen und alle übrigen visuellen Aspekte der Webseite erstellt. Diese Art der Codierung wird in Form von HTM- und anderen Dateien auf einem Webserver gespeichert, der in der Regel über Internet Information Server (IIS) verwaltet wird. Auch wenn es für ein »normales« Visual Basic-Win32-Programm Möglichkeiten für den Zugriff auf IIS-basierte Anwendungen gibt, beschränken wir uns (zumindest vorläufig) auf VBScript-Anwendungen (Visual Basic Script). Active Server Pages und andere HTML-Seiten, die Sie erstellen, können in Kombination von HTML mit einer der Skriptsprachen codiert

1. Nein, die Visual Studio-Version von Visual Basic wird nicht mehr diejenige sein, in der Sie heute Ihren Code verfassen. Ähnlich zwar, jedoch unterschiedlich genug, um einen Moment innezuhalten.

werden. Ich persönlich bin ein Verfechter von VBScript, die Browsergemeinschaft unterstützt es jedoch nicht generell. Unter IIS wird es jedoch immer unterstützt, daher wird mein webbasierter Code unter IIS ausgeführt.

Die zweite Form der Entwicklung, die auf den meisten Hochleistungssystemen anzutreffen ist, führt Code auf dem zentralen Server aus – Internet Information Server (IIS). Interessanterweise handelt es sich nicht unbedingt um binär kompilierten Code (obwohl über Webklassen die Möglichkeit besteht), sondern um interpretierte Visual Basic- oder Java-**Skripts**. Dies ist nicht einmal »kompilierter« Visual Basic-"P«-Code, sondern unverarbeiteter Quellcode, der in ASPs (ActiveX Server Pages) gespeichert, zur Laufzeit geladen, interpretiert und ausgeführt wird.

VBScript ist eher mit MBASIC 80 als mit Visual Basic vergleichbar – es ist erheblich weniger kompliziert. Mit Hilfe von VBScript können Sie beispielsweise keine ADO- oder andere COM-Objekte im Voraus erstellen, wie dies in »regulärem« Visual Basic-Code möglich ist. Auch wenn eine ASP-Seite auf COM-Objekte, sogar auf ADO-COM-Objekte, verweisen kann, müssen diese in Visual Basic oder einer anderen Sprache erstellt werden, die einen binären Compiler unterstützt (sodass DLLs erzeugt werden können). Um von einem VBScript-Programm auf ADO oder ein anderes COM-Objekt zugreifen zu können, müssen Sie die **CreateObject**-Methode verwenden, um diese Objekte während der Laufzeit zu erstellen. Außerdem arbeitet VBScript ausschließlich mit **Variant**-Variablen (die wiederum Zeichenfolgen, Zahlen, Arrays, Objekte usw. enthalten können). Das bedeutet, dass sämtliche VBScript-COM-Verweise »spät« gebunden werden.

Zudem gibt es eine neue Reihe von Einschränkungen, und viele dieser Einschränkungen bringen eine Vielzahl unserer Bemühungen zur Leistungsoptimierung zum Erliegen:

- Visual InterDev hilft beim Ausfüllen der VBScript-Methodenargumente, jedoch nur bei den vollständig aufgelisteten (und das sind beileibe nicht alle). Das bedeutet, dass Sie eine Datei mit sämtlichen Visual Basic-ADO-Konstanten angeben müssen. Nein, bitte kommen Sie nicht auf die Verwendung von Werten anstelle der Konstanten zurück. Sie können auch einen Verweis auf ADO in der Datei **Global.ASA** ablegen. Auf diese Weise kann Visual InterDev die ADO-Anweisungen während der Eingabe automatisch vervollständigen. ADO wird mit einer Datei geliefert, die in die ASP-Seite einbezogen werden kann: **\Program Files\Common Files\System\Ado\Adovbs.inc**
- VBScript unterstützt ausschließlich den Datentyp **Variant** – die AS-Klausel in der **Dim**-Anweisung erzeugt einen Syntaxfehler. Mit Hilfe von **Dim** können Sie daher Variablen und Arrays, nicht jedoch deren Datentyp deklarieren. Wenn

Sie nach guter Programmiermanier vorgehen, enthält Ihre Seite eine **Option Explicit**-Anweisung, die Sie zum Deklarieren aller Variablen zwingt und mit deren Hilfe Sie nicht deklarierte Variablen auffinden können.

- VBScript unterstützt nicht alle Funktionen der **Dim**-, **Public**- und **Private**-Anweisungen, die zum Deklarieren von Variablen verwendet werden. Sie können weiterhin Variablen anderer Datentypen einsetzen – Sie haben nach wie vor die Möglichkeit, **Date**-Werte, **String**-Werte und **Double**-Werte zu erstellen, diese müssen jedoch nach den für den **Variant**-Datentyp gültigen Regeln erstellt werden.
- Datentypen sind nicht das Einzige, was fehlt. VBScript lässt außerdem mehrere Anweisungen vermissen. Beispielsweise fehlen alle Anweisungen im Zusammenhang mit Datei-E/A, wie **Open**, **Close**, **Read** und **Input**. Das **Scripting.FileSystemObject**-Modell ersetzt diese grundlegenden E/A-Funktionen.
- Andere Anweisungen, die nicht verfügbar sind, sind **Write**, **GoSub**, **On GoSub**, **On GoTo**, **On Error** (und alle **Resume**-Operatoren) sowie **DoEvents**. Das bedeutet, dass Sie Ihren Ansatz zur Fehlerbehandlung neu überdenken müssen.[2]

9.1.1 Fehlerbehandlung in VBScript

Eine der größten Einschränkungen des VBScript-Codes ist die Fehlerbehandlungsroutine. Es gibt nämlich keine. Sie können zwar die Fehlerbehandlung aktivieren, jedoch nur, um Fehler verursachenden Code zu überspringen und den Fehler zu ignorieren – **On Error Resume Next**. Die meisten Webseiten, die ich gesehen habe, setzen diese Technik ein. Das bedeutet, dass Sie Code zur Verarbeitung der Fehler hinzufügen müssen, und zwar nach **jeder** Zeile des VBScript-Codes, die möglicherweise einen Fehler verursachen könnte.

IIS5 (unter Windows 2000) implementiert jedoch eine neue Technologie, die dieses Problem abschwächt, indem sie Ihnen die Konfiguration einer ASP-Ausnahmeseite ermöglicht. Diese Seite wird jedes Mal aufgerufen, wenn das Skript auf einen nicht verarbeiteten Fehler trifft. Sie können also basierend auf diesem Code eine »zentrale Fehlerbehandlungsroutine« implementieren.

Die standardmäßige Ausnahmeseite, die von IIS angegeben wird (**%WinDir%\Help\IisHelp\Common\500-100.asp**), kann an die besonderen Fehlerbedingungen Ihrer Site angepasst werden. Wenn Sie eine bessere Benutzeroberfläche, eine bessere Protokollierung oder einfach eine bessere Fehlerverwaltung benötigen, können Sie IIS konfigurieren und festlegen, zu welcher Seite in einer Website, einem Verzeichnis oder einem Seitenbereich gewechselt werden soll.

2. Weitere Informationen zu den Besonderheiten der Codierung in der Visual Basic Scripting Edition finden Sie unter http://msdn.microsoft.com/isapi/msdnlib.idc?theURL=/library/partbook/egvb6/programmingwithvbscript.htm.

Diese Seite besitzt Zugriff auf das **ASPError**-Objekt, dessen Eigenschaften Ihnen alle erforderlichen Informationen über den Fehler mitteilen. Handelt es sich beispielsweise um einen Skriptsyntaxfehler oder um einen Laufzeitfehler? Welche Seite hat den Fehler verursacht, und welche Zeile und welche Beschreibung wurde angegeben?

Die Verwendung dieser Ausnahmeseiten kann auch Probleme im Zusammenhang mit unzulänglicher Fehlerbehandlung und mangelhaften Debugginginformationen abschwächen, wie im weiteren Verlauf des Kapitels beschrieben wird. Die Standardfehlerseite zeigt zwar (aus irgendeinem obskuren Grund) keine Fehlerbeschreibung an, Sie können sie jedoch bearbeiten und in Zeile 65 Folgendes einfügen:

```
<h3><%= objASPError.Description %></h3>
```

Durch diese Änderung wird die Beschreibung des Fehlers oberhalb der übrigen Informationen bereitgestellt.

Sie werden außerdem feststellen, dass es nicht einfach ist, Fehlermeldungen zurückzusenden, wenn in den ASP-Seiten irgendetwas schief läuft. Sie möchten nicht, dass IIS oder VBScript eine Fehlerseite anzeigt, sondern Sie möchten die Fehler abfangen und sie verarbeiten. Wenn Sie jedoch eine XML-Antwort zurücksenden möchten, können Sie darin keine **Response.Write**-Nachrichten zu Informationszwecken einbetten. Wenn die ASP also XML-formatierte Daten zurücksenden soll und ein Fehler auftritt, müssen Sie diese Fehler selbst lösen (auf der ASP) oder eine Reihe anderer Techniken anwenden, die im weiteren Verlauf beschrieben werden.

ASP-Code ist eine merkwürdige Mischung aus ausführbarem Skriptquellcode (Visual Basic oder Java) und HTML-Befehlen (Hypertext Markup Language). Darauf bin ich zunächst hereingefallen, weil ich das Konzept der Umschaltzeichen (Shift) nicht begriffen hatte, die den Compiler anweisen, etwas Text zu kompilieren und den Rest an den Browser weiterzuleiten, um irgendeine Funktion zu erfüllen. Daher werden Sie manchmal von Skriptcode durchsetztes HTML mit speziellen Tags vorfinden, die die Umschaltung zum kompilierten Code kennzeichnen. Die meisten meiner ASP-Seiten, die XML zurückgeben, versuchen jedoch nicht, HTML zum Browser zurückzusenden, das nicht dem XML-Format entspricht – sofern kein Problem vorliegt.

Tipp Die zum Umschalten in den VBScript-Code und wieder zurück verwendeten Tags auf einer ASP-Seite lauten **<%** und **%>**. Alles innerhalb dieser Anfangs- und Endetags wird als Skriptcode betrachtet, der kompiliert und beim Laden der Seite ausgeführt wird. Sie können beliebig oft in den Code und wieder zurück wechseln. Alles **außerhalb** der Codeklammern wird als unformatierter Text oder als HTML-Befehle an den Browser gesendet.

9.1.2 Der HTML-Interpreter des Browsers

Der Browser verfügt über einen eigenen Interpreter, der HTML-Befehle z. B. zum Erstellen von Tabellen, zum Zeichnen von Linien oder Rahmen oder einfach zum Verschieben des Textcursors in die nächste Zeile erkennt – genau wie die 3270[3]-Systeme. Naja, fast. Die heutigen Browser unterstützen auch 3D und Vektorrendering, Drag&Drop und ein Ereignissystem, das das 3270-System nie hatte. Ja, es ist möglich, den Browser zum Interpretieren und Ausführen von Skriptcode zu veranlassen, beachten Sie jedoch, dass nur der Internet Explorer (IE) VBScript unterstützt. Alle Browser unterstützen JavaScript, daher finden Sie eine Menge Codegeneratoren für diese neue Einheitssprache.

Da alle Browser HTML unterstützen, bietet dies eine universelle Möglichkeit der Kommunikation mit dem Browser von einer ASP-Seite aus, die VBScript auf dem Server ausführt. Spätestens im Frühjahr 2001 werden wir bewundern dürfen, wie Visual Basic 7.0 eine Möglichkeit zum Erstellen von **binären** Visual Basic-Programmdateien und zum Extrahieren von HTML (und XML) bietet, um Inhalte an den Client zurückzugeben.

Der Browser erwartet, dass Sie so genannte **Tags** angeben, die bestimmte Bereiche der Webseite beschreiben. Diese Tags werden von spitzen Klammern (< und >) umschlossen. Ein Tag zum Erfassen einer Benutzereingabe in einem Textfeld sieht beispielsweise folgendermaßen aus:

```
Geben Sie den gesuchten Namen ein: <INPUT TYPE= "TEXT" NAME="txtName" VALUE=
"Default" >
```

Dieses Tag beginnt mit dem **<**-Zeichen, gefolgt vom Tagnamen INPUT. Das Tag endet mit dem **>**-Zeichen. Innerhalb der spitzen Klammern befinden sich **Attribute**, das Pendant zu Eigenschaften. In diesem Fall werden die Attribute TYPE, NAME und VALUE angegeben, um dem Browser mitzuteilen, wie dieser INPUT-Bereich zu verarbeiten ist und wie von einer anderen Stelle auf dieser Seite oder auf an-

3. 3270: Als ich in den Siebzigern anfing, mit Mainframes zu arbeiten, wurde das 3270-System als »intelligentes« Terminal zur Anzeige von Daten verwendet, die von einem Mainframe ausgegeben wurden. Es verfügte über eine rudimentäre Datenanzeige und eine Bearbeitungssprache, die Ähnlichkeit mit wirklich elementarem HTML besaß.

deren geladenen Seiten darauf verwiesen werden kann. Die Zeichenfolge »Geben Sie den gesuchten Namen ein:« wird auf der Webseite vor dem INPUT-Textfeld angezeigt. Alles außerhalb der HTML-Tags wird auf der Seite als Text angezeigt.

9.1.3 Ausführen von ActiveX Server Pages

Es gibt viele Möglichkeiten, den Code auf der ASP-Seite auszuführen. Eine häufig eingesetzte Methode besteht darin, von einer normalen Webseite darauf zu verweisen. Dies könnte über ein URL-Tag verwirklicht werden, das auf die ASP-Datei auf dem Server zeigt. Zu demselben Zweck können Sie auch das FORM-Tag und das **action**-Attribut einsetzen (wie im folgenden Codebeispiel dargestellt). Das FORM-Tag benennt eine bestimmte ASP-Datei auf dem Webserver, so, als würden Sie auf eine Remotedatei verweisen. In diesem Fall habe ich die **Post**-Technik eingesetzt, um ein Argument (den gesuchten Namen) an die ASP weiterzuleiten. Diese Technik verhindert, dass an die ASP gesendete Parameter in der URL-Adresse angezeigt werden, die die ASP aufruft.

Der hier gezeigte Code ist eine einfache (sehr einfache) Webseite, die zum Abfragen einiger Parameter von einem Browser und zum Starten einer ASP verwendet wird. Den gesamten Quellcode finden Sie in **PostExample1.HTML** im Beispielweb (ADOBP) auf der Begleit-CD.

```
<html>
   <meta NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
   <FONT SIZE="4" FACE="ARIAL, HELVETICA">
   <B>ADO Examples and Best Practices Example 1</B></FONT><BR>
    Passing arguments between an HTML form and an ASP<BR>
    using the Post method<BR>
   <body>
```

Diese nächste Zeile ist das FORM-Tag, das auf die Seite **SimpleASP.ASP** in einem bestimmten Verzeichnis auf dem Webserver verweist. Die Funktionsweise dieses Tags wird im weiteren Verlauf dieses Kapitels beschrieben.

```
   <FORM action=..\asp\SimpleASP.asp method=post>
```

Die nächsten Zeilen zeigen zwei INPUT-Felder zum Erfassen von Tastatureingaben des Benutzers an. (Im Grunde genommen handelt es sich um browserbasierte **TextBox**-Steuerelemente.) Das NAME-Attribut für jedes Steuerelement wird verwendet, um die Steuerelemente für die ASP zu identifizieren. Beachten Sie, dass das VALUE-Attribut einen Anfangswert (Standardwert) enthält, der gesendet wird, wenn der Benutzer keine Eingabe im Steuerelement vornimmt. Die Attribute die-

ser Steuerelemente werden als Mitglieder der FORM-Eigenschaft des **Request**-Objekts angegeben, sodass sie über ihren Namen auf der ASP referenziert werden können.

```
   Name: <INPUT NAME="txtName" VALUE="Vau%" > <BR><BR>
   Max Rows:<INPUT  NAME="txtMaxRows" VALUE=50
      style="HEIGHT: 25px; WIDTH: 31px" maxlength=2
      dataformatas=Numeric><BR><BR>
   <INPUT type="submit" value="Submit Query">
</FORM>
</body>
</html>
```

Etwas später in diesem Kapitel wird der serverseitige ASP-Code erläutert. In dem Code werden die Argumente extrahiert, die vom oben gezeigten Webseitencode abgefragt wurden, indem die FORM-Eigenschaft des **Request**-Objekts untersucht wird. Der serverseitige Code wird in Kürze behandelt.

Die Webseite, die durch den oben beschriebenen HTML-Code erstellt wird, sehen Sie in Abbildung 9.1.

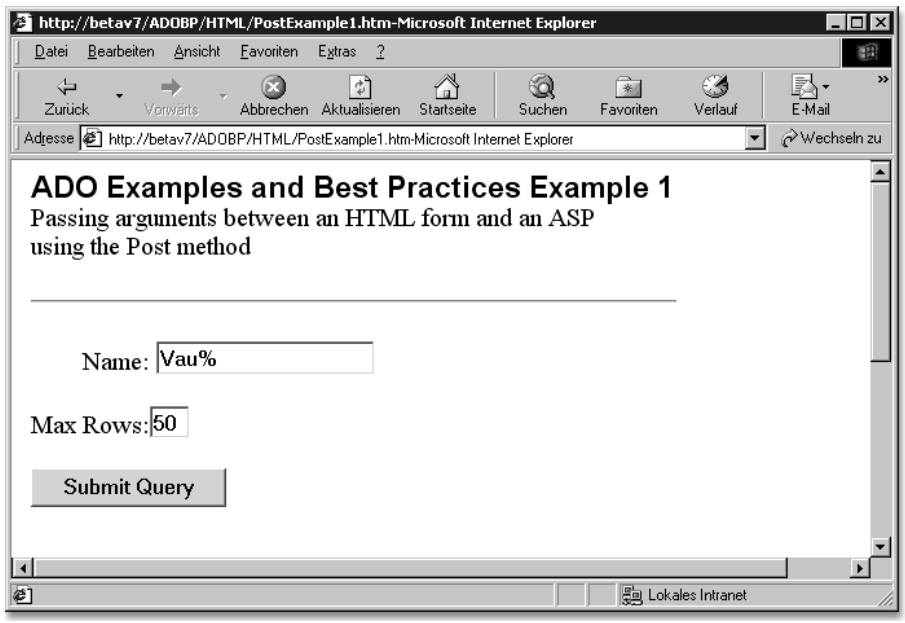


Abbildung 9.1 Abfragen von Parametern von einer Webseite und Aufrufen einer ASP

9.1.4 Angeben von Parametern

Da die auf der ASP ausgeführten Abfragen normalerweise Parameter benötigen, müssen Sie über INPUT-Tags (oder ein anderes HTML-Tag) Parameter abrufen oder die Parameter mit der URL-Adresse verketten, die zum Starten der Webseite verwendet wird. Wenn Sie beispielsweise eine Reihe von Autoren aus der **Biblio**-Testdatenbank auswählen möchten, deren Nachnamen mit einer bestimmten Zeichenfolge beginnen, könnten Sie den Benutzer über den vorgenannten Code zur Eingabe auffordern. Beachten Sie das INPUT-Tag, dessen Typ auf **submit** gesetzt ist (siehe unten). Hierdurch wird die Befehlsschaltfläche erstellt, die die ASP startet, wenn der Benutzer darauf klickt.

```
<INPUT type="submit" value="Submit Query" >
```

Zu diesem Zeitpunkt wird die aktuelle Seite wieder an die ursprüngliche Stelle im Stapel nach unten verschoben. Der Benutzer kann dorthin zurückwechseln, in der Regel generiert die ASP jedoch die nächste Seite, die dem Benutzer angezeigt wird. Wenn der ASP-Code allerdings zuviel Zeit in Anspruch nimmt, wird der Benutzer wahrscheinlich eine andere Adresse aufsuchen (aufgeben) oder erneut auf die **Submit**-Schaltfläche klicken. Durch erneutes Klicken startet IIS eine weitere Kopie der ASP und startet den Prozess erneut, wodurch die Situation sich nur noch verschlimmert. Anders als unter Visual Basic gibt es in diesem Fall eigentlich keine Möglichkeit, das zu verhindern.

Das FORM-Tag im vorherigen Code (unten noch einmal aufgeführt) weist den Browser an, ein **Form**-Objekt zu erstellen. Wenn auf die **Submit**-Schaltfläche geklickt wird, leitet der Browser das **Form**-Objekt an die ASP weiter, die im **Form Action**-Attribut genannt wird. Das FORM-Tag gibt außerdem die Methode an, die zum Weiterleiten eines beliebigen abgerufenen INPUT-Wertes an die Zielseite verwendet werden soll.

```
<FORM action=..\asp\SimpleASP.asp method=post>
```

In diesem Fall haben wir festgelegt, dass der Browser die **Post**-Technik zum Weitergeben der Argumente in der Hintergrundstruktur an die ASP einsetzen soll. Auf diese Weise werden die Argumente nicht in der URL-Adresse (Uniform Resource Locator) aufgeführt. Somit kann niemand die URL-Adresse im Browser bearbeiten, um die Parameter zu ändern (was möglich wäre, wenn Sie das **Method**-Attribut auf **Get** setzen).

Anmerkung Eine URL-Adresse zeigt einfach auf den »virtuellen« Dateinamen auf dem Remote-IIS-Server. Hierbei kann es sich um eine HTM-, ASP- oder eine andere vom Browser erkannte Seite handeln.

Wenn Sie das **Method**-Attribut des **Form**-Objekts auf **Get** (und nicht auf **Post**) einstellen, um Parameter von der Webseite an die ASP zu senden, fügt der Browser die Argumente, durch Fragezeichen (?) getrennt, an das Ende der URL-Adresse an. Das Ende jedes Parameters ist, wie nachfolgend dargestellt, durch ein Prozentzeichen (%) gekennzeichnet (allerdings wird es nicht immer verwendet).

```
http://www.betav.testsite.getdata.asp/?txtName=Vau%
```

Wenn Sie das **Method**-Attribut des **Form**-Objekts auf **Get** einstellen, kann die ASP über die **QueryString**-Auflistung auf einzelne Argumente verweisen:

```
Dim txtName
txtName = Form.QueryString("txtName")
```

9.1.5 Visual Basic im Vergleich zur VBScript-Entwicklung

Das ASP- oder Internetmuster verdeutlicht einen grundlegenden Unterschied zwischen dem Schreiben interaktiver Client/Server-Anwendungen in Visual Basic und dem Schreiben webbasierter Anwendungen. In Visual Basic haben wir Darstellungssteuerelemente erstellt, die im Prinzip statisch sind. Das heißt, sie werden durch Code gefüllt, der auf dem Client ausgeführt wird. Im VBScript-ASP-Programmiermodell führt der Client eine Datenanforderung aus und trägt den Inhalt des Resultsets anschließend in diese statischen Anzeigesteuerelemente ein. Im Allgemeinen erstellen Visual Basic-Client/Server-Entwickler nicht für jede Abfrage ein neues Formular mit aus Daten generierten Steuerelementen (was durchaus möglich wäre).

Bei der webbasierten Entwicklung mit ASP-Seiten hat es den Anschein, dass die neuen Seiten (oder zumindest Bereiche der Seiten) bei jeder Abfrage ganz neu aufgebaut werden. Wie jedoch im weiteren Verlauf dieses Kapitels erläutert wird, kann man denselben IIS-basierten serverseitigen ASP-Code von herkömmlichen Visual Basic-Programmen aus aufrufen – sogar auf ziemlich einfache Weise.

9.2 Verwenden von XML zum Zurückgeben von Daten aus Webseiten

In diesem Abschnitt untersuchen wir Schritt für Schritt ein Beispiel-Webprojekt, das mit einer einfachen Webseite beginnt, die zum Abrufen der erforderlichen Abfrageparameter eingesetzt wird. Dazu verwenden wir die soeben besprochene Webseite und gehen eine ASP-Seite durch, die die Daten, basierend auf den angegebenen Parametern, in XML zurückgibt. Die zweite Anwendung (die im Abschnitt »Verwalten von Recordsets mit Hilfe von XML-**Stream**-Objekten« des Ka-

pitels besprochen wird) verwendet eine mit Visual Basic-Client/Server vergleichbare Anwendung zum Starten von ASP-Seiten, zum Aktualisieren von Zeilen und zum Verarbeiten der Fehler.

Im vorgenannten Beispiel wurde ausgeführt, wie über die **Submit**-Schaltfläche eine ASP-Seite gestartet wird, die basierend auf den Eingabeparametern eine Abfrage ausführt. Dieser ASP-Code gibt ein XML-Recordset zurück, das formatiert und über eine XSL-Vorlage in HTML konvertiert wird. Auf diese Weise wird eine Tabelle erstellt und mit den Recordsetdaten auf dem Browser des Benutzers gefüllt. In diesem Fall handelt es sich bei der generierten »Tabelle« um eine Reihe von Rahmen, Überschriften und Inhalten, nicht um eine Datenbanktabelle.

9.2.1 Die serverseitige Active Server Page

Der ASP-Code übernimmt in Ihrer Anwendung den größten Teil der Arbeit. Er öffnet eine SQL Server-Verbindung, führt eine Abfrage aus, gibt ein Recordset zurück und erstellt das XML, das an den Browser zurückgesendet werden soll. Er formatiert das XML außerdem mit Hilfe eines XLS-Skripts in Form einer HTML-Tabelle. Die XSL-Formatvorlage ist so aufgebaut, dass sie sich an jede XML-Struktur anpasst, sodass sie sich zur Verwendung für andere »generische« XML-Anwendungen anbietet.

Der ASP-Code selbst wird als Datei mit eingerichteten Berechtigungen und Eigenschaften auf dem IIS-Server gespeichert, sodass IIS ihn als ausführbare Seite erkennt. Ohne diese Einstellungen kann IIS nicht viel mit der Seite anfangen, sondern beschwert sich vielmehr, dass Sie nicht über die zum Ausführen der ASP erforderlichen Berechtigungen verfügen. Eine schnelle Möglichkeit, die richtigen Einstellungen vorzunehmen, besteht darin, die ASP-Seiten in einem Verzeichnis mit anderen ASP-Seiten abzulegen, die über Ihr Visual InterDev-System erstellt wurden. Jawohl, ich habe zum Entwickeln dieser Beispiele Visual InterDev eingesetzt – und ich stehe dazu!

Tipp Bei der Arbeit mit Visual InterDev werden Sie feststellen, dass viele Methodenargumente automatisch während der Eingabe eingefügt werden. Das ist superpraktisch. Wenn Sie jedoch nicht aufgelistete Optionen einbringen möchten, müssen Sie sicherstellen, dass die Anwendung die ADO-Konstanten erkennen kann, indem Sie im oberen Bereich der ASP folgende Zeile hinzufügen:

```
<!-- #INCLUDE FILE="../include/ADOVBS.INC" -->
```

Anmerkung Ohne diese Zeile habe ich ständig einen ADO-Fehler (0x800A0BB9) ohne weitere Erläuterung erhalten. Ich vermute, dass die Optionseinstellung **adCmdText** der **Open**-Methode den Interpreter abgewürgt hat. Sobald ich dann **Option Explicit** in den ASP-Code eingefügt habe, teilte mir der Interpreter mit, dass er **adCmdText** nicht verstehe. Stellen Sie daher sicher, dass die **Include**-Datei im Projekt enthalten ist.

Sehen wir uns den serverseitigen ASP-Code genauer an. Der erste Abschnitt richtet die Variablen ein, die im ASP-Code verwendet werden. Beachten Sie, dass wir in diesem Fall nicht die Möglichkeit besitzen, die Datentypen zu deklarieren – alle Variablen entsprechen dem Typ **Variant**. Durch die Anweisung **On Error Resume Next** wird vermieden, dass der VBScript-Interpreter einfach eine Fehlerseite an den Client (Browser) zurückgibt; ich kennzeichne diese Anweisung jedoch während der Entwicklungsphase als Kommentar, damit ich auf Syntaxfehler aufmerksam gemacht werde. Sobald Sie **On Error Resume Next** aktivieren, werden alle Fehler »ignoriert«, und Sie müssen nach jeder Operation, die möglicherweise einen Fehler verursachen könnte, Code für die Fehlerprüfung einfügen. Danach sollte der Code Ihnen sehr bekannt vorkommen. Wir erstellen eine ganz normale Verbindungszeichenfolge (**ConnectionString**), die später in der ASP an die **Recordset Open**-Methode weitergeleitet wird.

Beachten Sie, dass **ConnectionString** (sConn) den Benutzernamen, das Kennwort, den Servernamen und die Datenbank in der ASP hardcodiert. Natürlich hätten wir diese auch von der HTM-Seite, die die ASP startet, einlesen und mit Hilfe der Technik **Request.Form(»variable name«)** in den Visual Basic-Code einfügen können, jedoch wird auf diese Weise das Verbindungspooling deaktiviert, sofern nicht jeder haargenau dasselbe eingibt. Sie müssen sich jedoch über die Hartcodierung dieser Werte keine Gedanken machen, da man sie später sehr einfach wieder ändern kann. Sobald die Seite geändert und neu gespeichert wurde, verwenden alle nachfolgenden Verweise sofort die neuen Einstellungen.

In sicheren Produktionssystemen sollten Sie den Benutzernamen und das Kennwort allerdings nicht mehr hartcodiert belassen, vor allem deshalb, weil wahrscheinlich noch viele andere an der Seite arbeiten und jeder von ihnen das Kennwort einsehen kann. Eine durchführbare Lösung hierfür ist, wie bereits erwähnt, die Verwendung einer UDL-Datei, die die Verbindungszeichenfolge enthält. Für die UDL können Sie Lese-/Schreibzugriff an die Administratoren und nur Lesezugriff an Benutzer erteilen, in deren Kontext die Datei verwendet wird (normalerweise die lokalen Benutzer **System** sowie **IUSR_machinename** und **IWAM_machinename** von IIS).

Sehen wir uns den ASP-Code genauer an. Beachten Sie, dass wir IIS als Erstes mitteilen, dass die Seite VBScript als Skriptsprache einsetzt. Nach diesem Hinweis sieht der ASP-Code eigentlich jeder anderen Visual Basic-Anwendung ziemlich ähnlich.

```
<%@ Language=VBScript%>
<%
Option Explicit
Dim sConn, sSql, rs, Query, MaxRows, stm
On error resume next
sConn = "Provider=SQLOLEDB.1;Password=xyzzy;User ID=ADOASP;" _
   & "Initial Catalog=biblio;Data Source=(local)"
```

Der folgende Code extrahiert die Parameter nach Namen aus der **Request.Form**-Eigenschaftenauflistung. Diese benannten Eigenschaften entsprechen den benannten INPUT-Steuerelementen, die auf der aufrufenden clientseitigen Webseite durch Tags gekennzeichnet werden. Jedes Eingabeargument wird ausgewertet und auf einen Standardwert gesetzt, wenn es als ungültig befunden wird. Die Eingabeüberprüfung könnte auch auf dem Client durchgeführt werden, wenn sich jedoch die Geschäftsregeln ändern, müssten Sie auch den clientseitigen Code synchronisieren. Das ist jedoch nicht annähernd so schwierig wie bei der Arbeit mit Win32-Client/Server-Anwendungen. Sie müssen dazu lediglich das Webseiten-HTML so ändern, dass die INPUT-Werte im Voraus überprüft werden.

```
MaxRows=Request.Form("txtMaxRows")
if MaxRows > "99" or MaxRows < "1" then MaxRows = "50"
Query = Request.Form("txtName")
if Query ="" then Query = "A%"
```

Der folgende Code erstellt eine SELECT-Anweisung zum Zurückgeben des gewünschten Rowsets. Nein, ich empfehle keinesfalls die Verkettung mit **&** zum Zusammensetzen der Zeichenfolge, wir müssen sie jedoch an dieser Stelle verwenden, damit meine Editoren die Zeichenfolge nicht zerhacken (sie würde sich über die gedruckte Seite hinaus erstrecken). Als Nächstes erstellen wir ein ADODB-**Recordset**-Objekt auf die harte Tour – durch Verwendung der **CreateObject**-Methode. Das Recordset öffnen wir über die SQL-Zeichenfolge und die Verbindungszeichenfolge (**ConnectionString**), die wir vorher erstellt haben. Beachten Sie, dass wir in diesem Fall kein separates **Connection**-Objekt erstellen – ADO hingegen hinter den Kulissen schon.

Tipp Die ASP-Methode **Connection.Open** wird viele Male in schneller Folge ausgeführt – einmal von jedem Client, der auf die ASP verweist. Daher ist es wichtig, dass das Verbindungspooling funktioniert. (Dies wurde im Kapitel über Verbindungen erläutert).

Um die Verwaltung von Verbindungsfehlern zu vereinfachen, könnten Sie die Erstellung eines separaten **Connection**-Objekts in Betracht ziehen, das Sie anschließend zum Öffnen der Datenbankverbindung verwenden. Auf diese Weise können Sie Fehler unabhängig vom Öffnen des Recordsets auffangen. Sobald die Verbindung geöffnet ist, können Sie das geöffnete **Connection**-Objekt an die **Recordset Open**-Methode weiterleiten.

```
sSQL="Select Top " & MaxRows & " Author, Year_Born " _
   & " from authors where author like '" & query & "' "
set rs = CreateObject("ADODB.Recordset")
rs.Open sSQL, sConn,,,adCmdText
```

IMHO In vielen (na gut, in fast allen) Codebeispielen, die ich gefunden habe, wurde die **CommandType**-Option bei Verwendung der **Open**-Methode nicht eingestellt. Dies ist schlicht auf Faulheit zurückzuführen – für die Ausführgeschwindigkeit einer Abfrage kann diese Einstellung jedoch von entscheidender Bedeutung sein.

In der nächsten Zeile wird getestet, ob die **Open**-Methode funktioniert hat, indem das Visual Basic-**Err**-Objekt auf 0 geprüft wird. Liegen Fehler vor, wird einfach über die **Response.Write**-Methode eine Nachricht an den Browser zurückgegeben. Durch diese Logik wird auch jeder Versuch zum Zurückgeben des XML vermieden.

```
if err then
   Response.Write "Error occurred:" & err.description & "<BR>"
   Err.Clear
else
   if rs.EOF then
      Response.Write "No authors found using " & Query & " filter.<BR>"
  Else
```

Tipp Stellen Sie beim Zurücksenden von Nachrichten an den Browser sicher, dass am Ende jeder Zeile das
-Tag hinzugefügt wurde. Im Gegensatz zur **Print**-Funktion von Visual Basic wird am Ende der Zeichenfolge nicht automatisch eine CRLF-Sequenz hinzugefügt.

Jetzt ist es an der Zeit, das XML zu erstellen, das an den Client zurückgesendet wird. In diesem Beispiel speichern wir das Recordset in einer im Arbeitsspeicher befindlichen Struktur, die erstmals in ADO 2.5 implementiert wurde – im »Stream«-Objekt. In diesem Fall wird der Stream als auf dem Recordset basierende XML-Struktur gespeichert. Anschließend »schreiben« wir den Stream (mit dem Recordset in XML) als Antwort auf den Clientbrowser. Dieser wird gefolgt von einer XSL-Formatvorlagenseite. Sowohl der Stream als auch die XSL-Datei werden über die **Response.Write**-Technik zurückgesendet, beachten Sie jedoch, dass wir **Response.ContentType** auf **text/XML** setzen, sodass der Browser weiß, wie die von der ASP zurückgegebenen Daten zu verarbeiten sind. Das **Content-Type**-Attribut wird als **type/subtype** formatiert, wobei **type** die allgemeine Inhaltskategorie und **subtype** den bestimmten Inhaltstyp bezeichnet. Beispielsweise sind **text/HTML**, **image/GIF**, **image/JPEG**, **text/plain** und (wie in unserem Beispiel) **text/XML** typische **ContentType**-Einstellungen. Sehen wir uns den Code an, der all diese Wunder vollbringen soll.

Warnung Wenn Sie **ContentType** auf **text/XML** einstellen und sich später entschließen, einfachen Text (oder HTML) zu schreiben, zeigt der Browser diesen nicht an.

```
Set stm = Server.CreateObject("ADODB.Stream")
   rs.Save stm,adPersistXML
   Response.ContentType = "text/xml"
   Response.Write "<?xml:stylesheet type=""text/xsl""
href=""recordsetxml.xsl""?>" & vbCrLf
   Response.Write stm.ReadText
   stm.close
   Set stm = Nothing
   end if
end if
%>
```

Anmerkung Der letzte Schritt (wie oben gezeigt) knüpft eine kleine XML-Vorlagendatei (XSL) zum Formatieren der XML-Daten an. Sie ist auf der CD enthalten, sodass Sie sie sich bei Gelegenheit einmal ansehen können. Dieser weitere Einheitsansatz (OSFA-Ansatz, One Size Fits All) wurde von Andrew Brust entwickelt[4]. Das heißt, dieses Vorlagenskript kann XML zerlegen und aus der Struktur eine browserbasierte Tabelle erstellen, solange sie nicht hierarchisch aufgebaut ist.

Die Nicht-XML-**Response.Write**-Meldungen werden nur dann zurückgesendet, wenn während des **Open**-Vorgangs (**Connection** oder **Recordset**) etwas schief läuft, und wir wechseln **nicht** zum Inhaltstyp **XML**. Wenn Sie im Klartext lesbare Meldungen in ein XML-Dokument einstreuen, erhalten Sie seltsame Meldungen über ungültige Header. Tatsächlich beschädigen die zusätzlichen **Response.Write**-Zeichenfolgen das sorgfältig formatierte XML, das im nächsten Schritt generiert wird.

IMHO[5] Meiner Ansicht nach sollte die Fehlerbehandlung die höchste Priorität bei der Webseitenentwicklung erhalten. Wir sind alle schon auf »professionelle« Sites gestoßen, die nichts sagende Meldungen ausspucken, wenn etwas danebengeht. Mit der Anzahl der Aufgaben, die Ihre Anwendung erfüllt, wächst die Wahrscheinlichkeit, dass eine ASP-Seite fehlschlägt. Wenn der Server keine Verbindungen mehr unterstützen kann, der Arbeitsspeicher ausgelastet ist oder das System von anderen Vorgängen blockiert wird, müssen Sie schon mehr als die dämliche (Standard-) Meldung bieten, die vom oben gezeigten Code generiert würde.

Die nächsten Codezeilen schließen das VBScript-Tag und weisen den VBScript-Interpreter an, die Datei ADOVBS.INC einzubeziehen. Ich nehme an, dass die Leistung gesteigert werden könnte, wenn nur die spezifischen Konstanten, die in dieser (ziemlich großen) Datei enthalten sind, berücksichtigt werden könnten – vor allem, wenn Sie bedenken, dass die ASP bei jeder Ausführung aus dem Quellcode **interpretiert** wird. Je mehr Zeilen zu kompilieren sind, desto mehr verschlechtert sich die Leistung.

4. Ich habe einiges an Referenzmaterial durchgesehen, um Beispiele für die ASP-Codierung zu finden. Dazu gehörten unter anderem das Buch *Professional Active Server Pages*, herausgegeben von Wrox Press (Homer, Enfield u.a.), und verschiedene Artikel, die im *Visual Basic Programmer Journal* veröffentlicht wurden, einschließlich eines großartigen Artikels namens »ADO Enhances the Web« von Andrew Brust und William Wen (Progressive Consulting). Ich habe außerdem sehr viel Unterstützung von meinen Freunden und Kollegen bei Microsoft erhalten, beispielsweise John Thorson, Carmen Sarro (und viele andere), und von meinem technischen Editor Eduardo.
5. In My Humble Opinion (Meiner bescheidenen Meinung nach)

```
<!-- #INCLUDE FILE="../include/ADOVBS.INC" -->
```

IMHO Ich habe Visual InterDev zum Schreiben und Testen dieser Beispiele eingesetzt. Im Vergleich mit Visual Basic und seinen interaktiven Debuggingmöglichkeiten, der sequenziellen Ausführung einzelner Anweisungen und den Supportfenstern (z.B. Lokalfenster) schneidet Visual InterDev ziemlich dürftig ab. Ich fand es sehr schwierig herauszufinden, was mit meinem Code passierte und warum er funktionierte – oder auch nicht. Die von ADO und IIS zurückgegebenen Fehlermeldungen waren allenfalls kryptisch.

9.3 Verwalten von Recordsets mit Hilfe von XML-»Stream«-Objekten

Nach Abschluss des vorhergehenden XML-Beispiels habe ich begonnen, mir einige Fragen zu stellen, und fand ein paar Antworten, die mir ein ganz neues Bild von der Verwendung von XML eröffnet haben. Auch wenn Sie XML mittlerweile vielleicht schon satt haben – der Zeitaufwand, der für die Untersuchung des nächsten Beispiels nötig ist, lohnt sich bestimmt.

Im vorherigen Beispiel wurde eine ASP verwendet, die ein parametergesteuertes Recordset generiert hat, das als formatierte XML-HTML-Tabelle zurückgegeben wurde. Die ASP-Seite setzte als Client eine browserbasierte Webseite voraus, die den **Name**-Parameter und die maximale Anzahl weiterzuleitender Zeilen abrief. Im Gegensatz dazu zeigt Ihnen das nächste Beispiel, wie Visual Basic als Frontend zum Ausführen des ASP-Codes verwendet wird (kein besonders bahnbrechendes Konzept) – in diesem Fall werden wir jedoch ein aktualisierbares Recordset erstellen, Änderungen daran vornehmen und das nicht verbundene Recordset an die ASP zurücksenden, um es über die **UpdateBatch**-Methode in der Datenbank abzulegen. Das Knifflige daran (falls Ihnen der Schwierigkeitsgrad noch zu niedrig erscheint) liegt in der Fehlerverwaltung. Diese Fehler werden (zumindest theoretisch) in jeder Phase generiert: beim Öffnen einer Verbindung, beim Ausführen der ursprünglichen Abfrage, beim Erstellen von XML, beim Senden und Empfangen von XML, beim Zurücksenden der Änderungen an IIS und beim Abrufen der Ergebnisse von **UpdateBatch**. Die Architektur dieses neuen Beispiels wird in Abbildung 9.2 dargestellt.

Dieser Ansatz ähnelt in vielerlei Hinsicht anderen Architekturen, die bereits erörtert wurden. Er verwertet Code, der auf einer anderen Plattform ausgeführt wird – in diesem Fall IIS. Im Gegensatz zu kompilierten gespeicherten Prozeduren unter SQL Server, die (nur) in TSQL geschrieben werden, wird dieser Code jedoch zur Laufzeit interpretiert, was die Ausführung im Vergleich zu kompilierten gespeicherten Prozeduren erheblich verlangsamt.

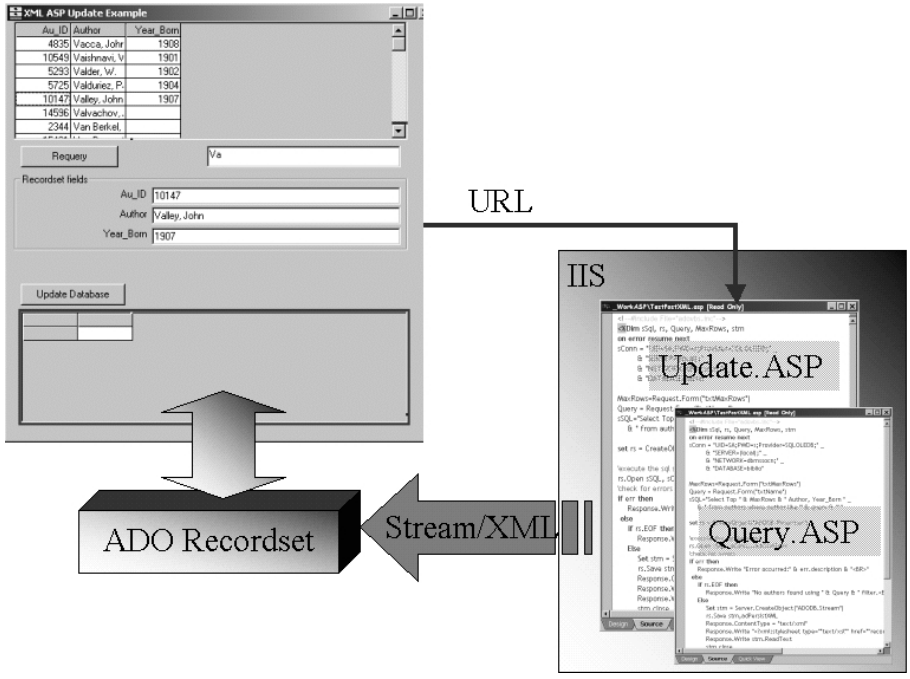


Abbildung 9.2 Zurückgeben von XML aus einer ASP über einen URL-Verweis zum Weiterleiten eines Recordsets

Auf der anderen Seite ist VBScript viel flexibler als TSQL, sodass Vorteile entstehen, die den Mangel an Geschwindigkeit eventuell (nur eventuell) wieder gutmachen. Einer dieser Unterschiede ist die Möglichkeit des ASP-Codes, binären Code auszuführen, der in Visual Basic, C oder einer beliebigen anderen Sprache entwickelt wurde. Dieser binäre Code kann außerdem dieselben gespeicherten Prozeduren ausführen, die Sie die ganze Zeit verwenden – genau wie der ASP-VBScript-Code.

Es gibt jedoch noch einen weiteren Aspekt. Während gespeicherte Prozeduren mit dem Client (oder der Komponente) über das RPC-Protokoll (Remote Procedure Call) über TDS (Tabular Data Stream) kommunizieren, sendet der IIS-ASP-Code Strukturen im HTTP-Format (und erwartet diese auch zu empfangen). Das hat zur Folge, dass viel mehr Pakete übertragen werden. Dieser Vergleich hinkt jedoch ein wenig, wie Edward (mein technischer Editor) mir aufgezeigt hat. Die Quintessenz ist, dass dieses neu entstehende Konzept völlig anders, jedoch in vielerlei Hinsicht viel flexibler ist als diejenigen, die wir bisher über Jahrzehnte eingesetzt haben und einsetzen mussten.

9.3.1 Untersuchen des Codebeispiels von Visual Basic zu ASP

Sehen wir uns die drei Codemodule nacheinander an. Wir untersuchen zunächst die Visual Basic-Clientanwendung, in der der Abfragefilter abgerufen wird, der als Parameter an diejenige WHERE-Klausel der ASP weitergegeben wird, welche die Abfrage für uns ausführen soll.

Im Grunde genommen funktioniert das Visual Basic-Clientprogramm folgendermaßen. Wir starten die Anwendung und geben unter Verwendung des aktuellen Wertes im Textfeld für Abfragekriterien (**txtQueryParm**) eine erste Abfrage aus. Zum Ausführen der Abfrage verwenden wir einfach die **Recordset Open**-Methode zum Ansprechen und Ausführen der Abfrage-ASP (**recordsetxml.asp**), wobei das Abfrageargument in der URL angegeben wird. Die ASP gibt XML zurück, das von ADO direkt in ein Recordset auf dem Client konvertiert wird (diese Funktion wurde in ADO 2.1 implementiert).

An dieser Stelle wird es erstmals knifflig. ADO erwartet ein Recordset, das von der ASP zurückgegeben wird, keine Fehlermeldung. Da es sich jedoch nicht um eine Webseite handelt, können wir nicht einfach einen **Response.Write**-Vorgang für die ASP ausführen, um dem Benutzer zu melden, wo der Fehler liegt. Um also die Fehlermeldung und eine Beschreibung zurückzugeben, erstellt die ASP ein neues Recordset und fügt ihm eine Zeile mit dem Fehler und entsprechenden Details hinzu. Cool? Wenn die Anwendung ein Klickereignis auf dem **MSHFlexGrid**-Steuerelement feststellt, wird eine Reihe von **TextBox**-Steuerelementen gefüllt, die zur Überprüfung eventueller Änderungen verwendet werden.

Sobald der Benutzer auf die Befehlsschaltfläche **Update** klickt, verwenden wir eine andere Technik, um das Recordset an die ASP zurückzusenden, die den **UpdateBatch**-Vorgang durchführt. Dieser Code verwendet das **MSXML**-Objekt zum Senden und Empfangen von XML von der ASP – eine neue Funktion von ADO 2.5. Dieses Objekt bietet nicht nur eine Möglichkeit zum Weiterleiten unseres Recordsets (das in einem Stream mit dem XML-Äquivalent des Recordsets gespeichert ist), sondern auch ein Verfahren zum Abrufen der Antwort von der ASP. Das kam mir sehr gelegen, als ich Fehlerinformationen in Form von in XML eingebetteten Kommentaren zurückgeben musste, da das **MSXML**-Objekt sowohl **ResponseText** als auch **ResponseXML** offen legt.

Wenn Fehler auftreten (und das ist tatsächlich gelegentlich der Fall), wird dem Benutzer die Option bereitgestellt, den Vorgang zu wiederholen oder seine Änderungen einfach durchzusetzen. Im Quellcode finden Sie ein Grundgerüst von Vorgehensweisen für die Fehler, die eintreten können. Wir stellen die **Update Criteria**-Eigenschaft so ein, dass die vorgenommenen Änderungen durchgesetzt werden oder die Abfrage einfach erneut gesendet wird.

Da der Quellcode für dieses Programm übrigens auf der CD enthalten ist, werde ich Sie an dieser Stelle nicht mit öden Details langweilen – ich zeige Ihnen nur die Highlights, die an diesem Ansatz einzigartig sind. Sie werden viele Auflistungen und andere geläufige Techniken aus vorherigen Passagen wiedererkennen, daher wäre es müßig, sie noch einmal durchzugehen.

Tipp Ich fand es einfacher, den Muster-ASP-Code in der Visual Basic 6.0-IDE zu entwickeln. Sobald er in Visual Basic funktionierte, war die Übertragung des Codes auf die ASP ein Kinderspiel.

9.3.2 Der clientseitige Visual Basic-Code

Zunächst erstellen wir eine Konstante, die auf den IIS-Server und den Webstamm zeigt. Sehen Sie erst gar nicht nach, dieser Server ist nicht im WWW. Das Textfeld **txtField** wird gefüllt, wenn wir ein Klickereignis auf dem **MSHFlexGrid**-Steuerelement erhalten.

```
Const WebHost As String = "http://betav7/testxml/myhtml/"
```

Der folgende Code wird ausgeführt, wenn der Benutzer die Kriterien ändert und auf die Befehlsschaltfläche **Requery** klickt. Außerdem wird er beim ersten Anwendungsstart ausgeführt. Er ruft die ASP auf und leitet die aktuellen Kriterien mit der URL-Adresse weiter. Der **Recordset Open**-Methode wird entweder ein Rowset aus der ASP oder ein neues Recordset mit dem Fehlertyp (Verbindung oder Abfrage), der Fehlernummer und einer Beschreibung angegeben. Lautet das erste Feld **EType**, wissen wir, dass sich im XML-Stream ein Fehler eingeschlichen hat. Liegt jedoch kein Fehler vor, so gibt es ein Rowset. In diesem Fall füllen wir die Tabelle einfach mit den Recordsetdaten.

```
Private Sub cmdRequery_Click()
On Error GoTo cmdEH
If rs.State = adStateOpen Then rs.Close
rs.Open WebHost & "recordsetxml.asp?Query=" & txtQueryParm
If InStr(rs.Fields(0).Name, "EType") Then
   ' Die Verbindung oder Abfrage hat nicht funktioniert.
   Die XML enthält die Fehlermeldungen... nicht die Zeilen
   Select Case rs.Fields("EType")          ' Fehlertyp
       Case enuError.Connection
          MsgBox "The active server page could not connect to the server. " _
          & vbCrLf & rs.Fields("Description"), _
          vbCritical, "Error Connecting"    ' Beschreibungsfeld
      Case enuError.OpenRecordset
          MsgBox "The active server page could not execute the query. " _
```

```
            & vbCrLf & rs.Fields("Description"), vbCritical, _
            "Error Opening"    ' Beschreibungsfeld
        Case Else
            'Bitte?
            MsgBox "Unrecognized error type. Programming error", vbCritical
    End Select
 Else
Set MSHFlexGrid1.Recordset = rs
End If
Quit:
    Exit Sub
```

Der nächste Codeabschnitt wird ausgeführt, wenn der Benutzer auf die Befehlsschaltfläche **Update** klickt. Wir erstellen ein **MSXML**-Objekt, das ein **Stream**-Objekt mit dem Recordset (in XML) an die ASP weiterleitet, auf der die **UpdateBatch**-Methode angewendet wird.

```
Private Sub cmdUpdate_Click()
Dim strMsg As String
Dim stm As ADODB.Stream
' Microsoft XML Version 2.0
Dim xml As MSXML.XMLHTTPRequest
Dim rsLocal As Recordset
Dim fld As Field
Dim stmXML As Stream

On Error GoTo cmdUpdateEH

Set xml = New MSXML.XMLHTTPRequest
Set rsLocal = New Recordset
rsLocal.CursorLocation = adUseClient
rsLocal.Properties("Update Criteria") = adCriteriaAllCols
RetryUpdate:
Set stm = New ADODB.Stream
rs.Save stm, adPersistXML          ' Globalen RS in Stream speichern
'
' Zugreifen auf die Seite, die eine UpdateBatch-Operation ausführen kann
'
xml.Open "POST", WebHost & "UpdateXMLRS.ASP", False
xml.send stm.ReadText
```

Die ASP versucht, die Datenbank zu aktualisieren, jedoch können Sie laut Murphy einen XML-Stream mit dem »aktualisierten« Recordset (mit ausstehenden Zeilen) zurückerwarten, anhand dessen Sie feststellen können, welche Zeilen nicht aktualisiert wurden.

Sobald das Recordset eintrifft (denken Sie daran, dass es nur zurückkommt, wenn Konflikte aufgetreten sind), können wir die **Filter**-Eigenschaft zum Aufheben der Auswahl nicht betroffener Zeilen verwenden. Beachten Sie, dass im Recordset noch immer alle Zeilen enthalten sind; wenn Sie also dieses gefilterte Recordset an die Tabelle weitergeben, werden alle Zeilen angezeigt. Aus diesem Grund habe ich eine andere Routine gewählt, um die Tabelle manuell zu füllen. Achten Sie darauf, dass die Werte für Fehlernummer und Beschreibung von der Aktualisierungs-ASP über das Kommentartag in XML zurückgegeben werden. Anhand des Aktualisierungs-ASP-Codes können wir feststellen, wie dies geschieht.

```
If xml.responseText <> "" Then            ' Testen auf Fehlertext oder Rowset
   rsLocal.Open xml.responseXML           ' Rowset öffnen
   rsLocal.Filter = adFilterPendingRecords
   If rsLocal.RecordCount > 0 Then
      DumpRsToGrid rsLocal
   If InStr(xml.responseText, "RSUpdate error") Then    ' Fehler beheben…
```

9.3.3 Der serverseitige ASP-Abfragecode

Anhand dieses Codes werden einige neue Techniken vorgestellt. Natürlich habe ich den Code aus dem ersten XML-Beispiel in diesem Kapitel wiederverwertet, aber in diesem Fall können wir keine Fehlerinformationen über HTML zurücksenden – wir können nur in Form eines Recordsets antworten. Der Grund hierfür liegt darin, dass wir die ADO-**Recordset Open**-Methode zum Abrufen der Zeilen verwenden. Wenn bei der Verbindung oder der Abfrage etwas schief läuft, müssen wir dies dem Client mitteilen. Wir könnten zwar auch einfach gar nichts an den Client zurückgeben und ihn die ADO-Funktion **RatMalWoran'sLiegt** ausführen lassen, diese hat jedoch auf meinem System nicht mehr funktioniert, seit ich Age of Empires installiert habe. Kurzum, den Teil des Codes, den wir bereits besprochen haben, lasse ich diesmal aus.

Genau wie bei früheren ASP-Seiten, erstellen wir die ADO-Objekte durch Ausführen von **Server.CreateObject**. Um die Verarbeitung von Verbindungsfehlern zu vereinfachen, verwenden wir ein separates **Connection**-Objekt. Außerdem erstellen wir ein **Recordset**- und ein **Stream**-Objekt, um die Daten von einer Schicht an die nächste weiterzugeben.

```
' Objekte initialisieren:
On error resume next
set cn = server.CreateObject ("ADODB.Connection")
Set rs = Server.CreateObject("ADODB.Recordset")
Set stm = Server.CreateObject("ADODB.Stream")
```

Durch den nächsten Code wird eine Verbindungszeichenfolge (**ConnectionString**) erstellt und, wenn die Verbindung geöffnet ist, ein Recordset geöffnet, das auf den Parametern basiert, die am Ende der URL-Adresse verkettet sind. Wir haben keine Möglichkeit, ein HTML-**Form**-Objekt in Visual Basic zu erstellen, daher müssen wir das **Form Method**-Attribut auf **Get** setzen. Außerdem stellen wir sicher, dass das Recordset mit optimistischen Stapelsperren erstellt wird, damit ADO es im Schreib-/Lesemodus an den Client zurückgeben kann. Denken Sie daran, dass der Standardmodus RO (schreibgeschützt) lautet.

```
cn.ConnectionString =  "file name=c:\biblio.udl"
cn.CursorLocation  = aduseclient
cn.Open
if err then
   BuildErrorRecord 0, err, err.Description
else
   Parm = Request.QueryString("Query")
   rs.Open "select Au_ID, Author, Year_Born " _
   & " from authors where author like '" & Parm & "%'"  Order By Author", _
      cn, adOpenKeyset, adLockBatchOptimistic, adCmdText
   if err then
      BuildErrorRecord 1, err, err.Description    'type 1 = OpenRecordset
   end if
End if
```

Na also, das Recordset kann jetzt zurückgesendet werden. Der nächste Codeabschnitt speichert das Recordset im XML-Format im lokalen **Stream**-Objekt. Anschließend werden die Informationen zum Unicodeformat an die sendende »Seite« (unsere Visual Basic-Anwendung) zurückgeschrieben. Stimmt, das sind eine ganze Menge Bits. Beachten Sie, dass wir die **Recordset ActiveConnection**-Eigenschaft diesmal nicht auf **Nothing** gesetzt haben, wie beim Zurücksenden von **Recordset**-Objekten in anderen Entwürfen für die mittlere Schicht. Warum? Fragen Sie nicht, aber es funktioniert.

```
rs.Save stm, adPersistXML
Response.ContentType = "text/xml"
Response.Write stm.ReadText
```

Wenn beim Öffnen der Verbindung oder beim Ausführen der Abfrage ein Fehler auftritt, erstellen wir ein neues Recordset, um die Fehlerinformationen zu speichern. Die folgende Routine wird ausgeführt, wenn Fehler vorliegen, die an die Visual Basic-Schicht zurückgemeldet werden müssen. Wir erstellen ein unformatiertes **Recordset**-Objekt mit drei Feldern für den Fehlertyp (Verbindung oder Abfrage), die Fehlernummer und die Fehlerbeschreibung. Denken Sie beim Erstellen von ADO-**Recordset Field**-Objekten daran, dass ADO den gesamten Wert ignoriert, wenn das Feld nicht groß genug ist, um den angegebenen Wert aufzunehmen. Wenn Sie beispielsweise ein Zeichenfolgenfeld für 100 Byte erstellen und dessen **Value**-Eigenschaft auf eine 101-Byte-Zeichenfolge einzustellen versuchen, ignoriert ADO den gesamten Vorgang, ohne auch nur einen Ton von sich zu geben. Ich hätte eigentlich erwartet, dass es die Zeichenfolge nach 100 Byte abschneidet, aber nein, es ignoriert einfach die gesamte Zeichenfolge. Fies!

```
Sub BuildErrorRecord (intType, intErr, strDescription)
   set rs = server.CreateObject ("adodb.Recordset")
   With rs
      .Fields.Append "EType", adInteger
      .Fields.Append "Error", adInteger
      .Fields.Append "Description",adVarChar,255
      .open          ' Erstellen eines neuen Recordsets nur für Fehler
      .addnew             ' Hinzufügen einer neuen Zeile
      .fields("EType") = intType
      .fields("Error") = intErr
      .fields("Description") = strDescription
      .UpdateBatch
   End With
End Sub
```

9.3.4 Der serverseitige ASP-Aktualisierungscode

Die letzte unserer drei Routinen ist für eine Aufgabe zuständig – für das Ablegen der am Client vorgenommenen Änderungen in der Datenbank. Wie bei den vorherigen ASP-Seiten verwenden wir **On Error Resume Next**, um die VBScript-Fehlerbehandlungsroutinen zu deaktivieren.

IMHO Leider fehlt ADO ein wichtiges Element. Ich wollte zu oft Fehler- oder benutzerdefinierte Statusinformationen mit dem Recordset zurückgeben. Es sollte eigentlich eine einfachere Möglichkeit geben, zusätzliche Eigenschaften für das Recordset zu definieren, die zusammen mit dem Rowset durch Marshaling übertragen werden.

```
<!--#include File="adovbs.inc"-->
<%
Dim rs, stm, srtoption, ErrorMessage
on error resume next

Set rs = Server.CreateObject ("ADODB.Recordset")
Set stm = Server.CreateObject("ADODB.Stream")
```

Diese ASP-Seite ist ziemlich einfach. Wir brauchen lediglich ein **Recordset**-Objekt und einen Stream. Das **Recordset**-Objekt wird aus dem Stream neu erstellt, der an die ASP weitergegeben wird. Beachten Sie, dass die **Open**-Methode das **Request**-Objekt als **Source**-Argument nutzt. Anschließend verknüpfen wir die Verbindungszeichenfolge erneut und weisen ADO an, die Änderungen in der Datenbank abzulegen.

```
with rs
   .CursorLocation = adUseClient
   .Open Request
   .ActiveConnection = "dsn=LocalServer;uid=TestASP;pwd=Secret"
   .UpdateBatch
end with
```

Wenn alles gut geht, wird die ASP beendet und die Steuerung an die Visual Basic-Anwendung zurückgegeben, die die ASP aufgerufen hat – nichts wird zurückgesendet. Ich denke, wir könnten einige XML-Kommentare mit den betroffenen Datensätzen oder ähnlichem Feedback ausgeben.

Wenn jedoch ein Fehler auftritt, müssen wir eine XML-Zeichenfolge erstellen, die die Fehlerinformationen und das Recordset enthält – dieses umfasst nun neue Statusinformationen für diejenigen Zeilen, die aus irgendeinem Grunde nicht aktualisiert wurden. Praktisch wird hierzu einfach ein Kommentar in den XML-Stream eingefügt, der über die **MSXML ResponseText**-Eigenschaft zurückgegeben wird.

```
if err then
       ErrorMessage =  "RSUpdate error: " & err.Description _
       & "[" & err.Number & "]"
       rs.Save stm,adPersistXML  ' Siehe unten stehenden Tipp
   ' Einstellen des Inhaltstyps auf xml, Angeben der XSL für die Formatierung,
   ' Weiterleiten des XML-Streamtexts an die aufrufende Komponente:
       Response.ContentType = "text/xml"
       ' Wird als Kommentar behandelt
       Response.Write "<!-- " & ErrorMessage &  " -->"
       Response.Write stm.ReadText
end if
```

Tipp In Windows 2000 (also in ADO 2.5 und höher) können Sie direkt in ein **Response**-Objekt speichern, anstatt zunächst in einen Stream speichern zu müssen. Daher:

```
rs.Save Response, adPersistXML
```

Wie dieses Beispiel zeigt, gibt es viele innovative Techniken zum Weiterreichen von Daten von einer Schicht zur nächsten – zwischen Client und Server, zwischen Webclient und IIS und zwischen Komponenten der mittleren Schicht und den übrigen Schichten.

Ich schätze, dass einige dieser Techniken an Stabilität und Benutzerfreundlichkeit gewinnen werden, während Visual Basic 7.0 immer näher rückt – und dann dürfen wir alle noch einmal von vorne anfangen.

10 Die Daten in Form bringen

Nein, dieses Kapitel beschäftigt sich nicht damit, wie Sie die fülligen Zeilen aus Ihren Cursorn schlanker machen – was vielleicht keine schlechte Idee wäre. Es soll die Verwendung der ADO-**Shape**-Syntax zur Verwaltung hierarchischer Daten besprochen werden. Aber was sind eigentlich »hierarchische« Daten? Die meisten von uns arbeiten sehr häufig mit hierarchischen Daten, ohne es zu wissen. Wenn Sie beispielsweise mit einer Kundentabelle arbeiten, die mit einer Rechnungstabelle verknüpft ist, die wiederum eine verknüpfte Artikeltabelle aufweist, handelt es sich um eine Hierarchie. Der Unterschied zwischen einer normalen Abfrage und einer hierarchischen Abfrage besteht darin, dass die Beziehungen in einer Hierarchie verankert sind. Das heißt, das Schema der Beziehungen ist Bestandteil der hierarchischen Struktur. Daher »weiß« das Tool zur Datenreferenzierung, dass 0 bis **n** Aufträge für jeden Kunden vorliegen können, und dass die Kunden-ID in der Auftragstabelle auf eine gültige Zeile in der Kundentabelle verweisen sollte.

Seit einiger Zeit haben die FoxPro- und ADO-Teams an einem Mechanismus gearbeitet, der für die Erstellung und Verwaltung von Datenhierarchien eingesetzt werden kann. Das Ziel war, die Verwaltung der zugrunde liegenden hierarchischen Tabellen zu erleichtern. Mit anderen Worten: Die neue ADO-Technologie, die hier beschrieben werden soll, wird sowohl dazu verwendet, Daten abzurufen als auch Zeilen hinzuzufügen, zu ändern und zu löschen, ohne dass man hierzu durch brennende Reifen springen müsste. Das Ergebnis dieser Arbeit ist ein neuer OLE DB-Provider, der in ADO integriert ist und dazu konzipiert wurde, insbesondere diese häufig eingesetzten Datenstrukturen zu handhaben. Der neue OLE DB-Provider – der Shapeprovider – scheint die Handhabung verschiedener üblicher Datenzugriffsprobleme zu erleichtern. Wie wir jedoch sehen werden, wurde der Provider für eine bestimmte Zielgruppe entwickelt, zu der Sie möglicherweise nicht gehören.

10.1 Der Shapeprovider

Wie bereits erläutert wurde, besteht das OLE DB-Konzept in der Handhabung beliebiger Daten – von flachen, nicht strukturierten Daten über relationale Daten bis hin zu komplexen objektorientierten Datenquellen und allem, was dazwischen liegt. Der Shapeprovider ist eines der neuen Features von ADO 2.1, mit dem die OLE DB-Funktionalität erweitert wird. Wie wir später sehen werden, wird der Shapeprovider ähnlich verwendet wie eine Cursorbibliothek oder andere OLE DB-Provider. Wie bei einem Cursorprovider benötigen Sie weiterhin einen Datenprovider zum Abrufen der Zeilen – d.h., der Shapeprovider kann praktisch mit einer beliebigen Datenquelle eingesetzt werden. Grundsätzlich verwaltet der

Shapeprovider hierarchische Daten für Datenquellen, die hierzu nicht in der Lage sind, bzw. für Entwickler, die nicht wissen, wie mit Hilfe von Datenprovidern hierarchische Daten zurückgegeben werden. Zeilen werden für den Client abgerufen, in bezogenen Gruppen angeordnet und für Anzeigesteuerelemente (beispielsweise **MSHFlexGrid**) oder Berichtsengines (z. B. Data Report von Visual Basic 6) offen gelegt. Der Shapeprovider unterstützt, wie ein Cursorprovider, nach Möglichkeit die Aktualisierbarkeit. Darüber hinaus werden verschiedene Aggregatfunktionen zum Zusammenfassen der Daten oder zum einfachen Bereitstellen von berechneten Spalten unterstützt.

Wer verwendet hier bei Microsoft den Shapeprovider? Nun, der Shapeprovider wird von einer Reihe interner Anwendungen verwendet. Wie später noch erläutert werden soll, verwenden diese Anwendungen relativ kleine Datenbanken (Datenbanken, die auf einer Diskette gespeichert werden können), und viele verwenden den Shapeprovider zusammen mit Webanwendungen. Einige Entwickler betrachten den Shapeprovider als nützlich, wenn der Benutzer über Daten mit hierarchischen Beziehungen verfügt und der systemeigene Datenspeicher keine Hierarchien unterstützt. Dies trifft auf die meisten relationalen Datenspeicher zu.

Hierarchien sind auch dort hilfreich, wo Verknüpfungen (Joins) verwendet werden. Während durch diese Verknüpfungen die zueinander in Beziehung stehenden Zeilen zurückgegeben werden, wird diese Beziehung nicht in die zurückgegebene Struktur integriert. Die Entwickler haben schon länger nach einer Möglichkeit zur Programmierung hierarchischer Objektmodelle zum Offenlegen der Verknüpfungsbeziehungen gesucht, damit es nicht mehr erforderlich ist, flachgeklopfte, denormalisierte Sätze verknüpfter Datensätze aufzubröseln. Der Shapeprovider stellt diese Funktion bereit, indem die normalisierten Daten auf dem Client beibehalten werden. Darüber hinaus bietet der Shapeprovider ein konsistentes Programmmodell für beliebige Backends. Die Rowsetfähigkeiten in der **Shape**-Syntax stellen des Weiteren ein leistungsfähiges Tool zum Erstellen temporärer Tabellen und Hierarchien bereit, bei denen kein Verweis auf einen dauerhaften Datenspeicher erforderlich ist. Wenn Sie wissen möchten, wo **Shape** überall eingesetzt werden kann, sprechen Sie mit den Access- und Data Page-Leuten von Office 2000 – sie haben von **Shape** umfangreichen Gebrauch gemacht.

Anmerkung Als ich mit der Untersuchung des Shapeproviders begann, stellte ich fest, dass es zwar viele Artikel zu diesem Thema gab, keiner jedoch die Aspekte zu behandeln schien, die mich interessierten. In den meisten Beispielen werden ungebundene Abfragen für winzige Datenbanken ausgeführt, beispielsweise SELECT * FROM Authors in der Beispieldatenbank **Pubs**. Ich habe einen interessanten Artikel auf der MSDN-CD gefunden (»Shape Up Your Data Using ADO 2.0 Hierarchical Recordsets in ASP«), bei dem Parameterabfragen erstellt wurden, aber in diesem Artikel wird über eine webbasierte Anwendung gesprochen, und ich befasse mich üblicherweise hauptsächlich mit Client/Server-Visual Basic-Anwendungen.

In diesem Kapitel sollen vor allem das Client/Server-Modell und die Entwickler umfangreicher Datenbanken Berücksichtigung finden. Bei näherer Betrachtung der **Shape**-Funktionalität wurde mir klar, dass der Provider für kleine, tabellenbasierte Datenbanken gedacht ist – besonders dann, wenn Sie das Standardverhalten verwenden. Zu viele der funktionellen Voraussetzungen (und Standardeinstellungen) des Shapeproviders machen diesen für größenveränderliche Datenbanken oder Datenbanken, die eigenständig komplexe hierarchische Resultsets erstellen, unbrauchbar. SQL Server beispielsweise kann ohne weiteres anhand eines einzigen Serveraufrufs ein Recordset erstellen, dass auf einer einzelnen oder mehreren übergeordneten Zeilen und sämtlichen verknüpften untergeordneten Zeilen basiert – und dies bis zur gewünschten Verschachtelungstiefe. Es sind jedoch nicht alle OLE DB- oder ODBC-Provider in der Lage, derartig umfassende Funktionalität bereitzustellen, und einfachere Provider bieten keine Funktionen zur Rückgabe dieser Hierarchien. Dennoch ist der Shapeprovider mehr als nur eine Möglichkeit zur Rückgabe von Überordnungs/Unterordnungs-Hierarchien. Der Shapeprovider kann zum Erstellen, Berechnen und Verwalten komplexer Aggregatfunktionen eingesetzt werden, wie sie bei der Berichtserstellung benötigt werden.

Bei der Arbeit mit dem Visual Studio-Marketing-Team von Microsoft habe ich den Shapeprovider verschiedene Male vorgestellt. Nach etwa einer Minute des Klickens und Ziehens konnte ich eine drei Tabellen umfassende Hierarchie aufweisen, die der in Abbildung 10.1 ähnelte. Dies ist relativ einfach. Ziehen Sie einfach die drei Tabellen aus dem Datenansichtfenster über das Fenster des Datenumgebungs-Designers, und setzen Sie die Datenverknüpfungseigenschaften (auf der Eigenschaftenseite), um einen Querverweis auf die Felder **au_id** und **title_id** zu erstellen. Ich zog das übergeordnete Objekt (in diesem Fall der **authors**-Befehl) über ein Formular und wählte zum Anzeigen der Zeilen das **MSHFlexGrid**-Steuerelement aus. Nein, ich werde Sie nicht dazu nötigen, dieses Verfahren auch anzuwenden. Ich werde Ihnen statt dessen eine einfachere Methode zeigen, die einige

Codezeilen erfordert, mit der Sie jedoch weitaus bessere Leistungsergebnisse erzielen und Ihre Ergebnisse besser steuern können. Wenn Sie das Demo auch sehen möchten, dann fangen Sie mich auf der nächsten VBits ab.

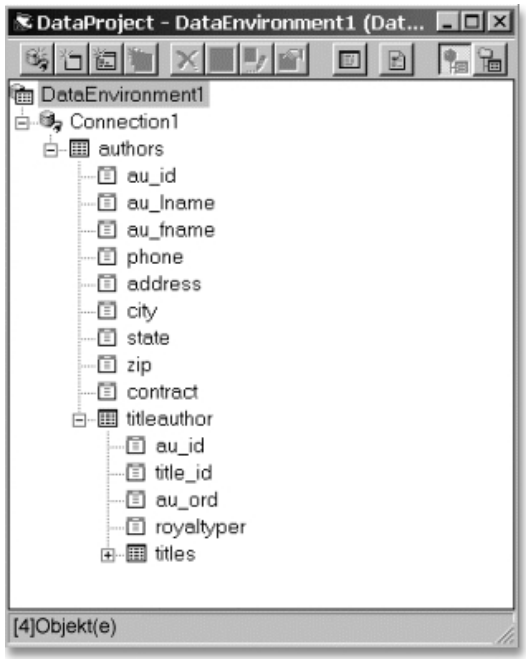


Abbildung 10.1 Durch den Datenumgebungs-Designer erstellte Hierarchie

10.2 Erstellen einer einfachen »Shape«-Anweisung

Beginnen wir mit den Grundlagen, indem wir eine Visual Basic-Anwendung erstellen, die den Shapeprovider für eine einfache Aufgabe einsetzt. Wir verwenden für diese Übung die Visual Basic 6.0 Enterprise Edition – ich kann nicht garantieren, dass diese Übung auch problemlos mit der Professional Edition durchgeführt werden kann. Ich habe des Weiteren das SP3 und die neue ADO 2.5-Bibliothek (MDAC) installiert. Ich referenziere jedoch weiterhin die 2.0-Bibliothek – auf diese Weise werden einige Probleme mit dem Datenumgebungs-Designer vermieden, die durch die neuere ADO-Version verursacht werden. Ich habe zum Testen meinen lokalen SQL Server 7.0 und die Datenbank **Pubs** eingesetzt. Darüber bin ich froh, denn die Ausführung einiger der Beispielanwendungen hätte den ganzen Nachmittag gedauert, wenn ich viele Daten abgefragt hätte (mehr als 20 Zeilen). Beginnen Sie zunächst mit einem kleinen Tabellensatz – wenn Sie sich etwas eingearbeitet haben, können Sie nach und nach größere Datenspeicher verwenden.

Anmerkung Denken Sie daran, dass das Standardverhalten des Shapeproviders die Verwendung von zwei (oder mehr) SELECT *-Abfragen (ohne WHERE-Klausel) vorsieht, genau wie die meisten Beispiele in der MSDN-Dokumentation. Das heißt, die Visual Basic-Tools generieren eine Abfrage ohne festes Ende als »übergeordneten« Befehl und eine zusätzliche Abfrage ohne festes Ende für jeden »untergeordneten« Befehl. Diese werden in einem einzigen Stapel an SQL Server übermittelt. Wenn Sie also keine andere Codierung vornehmen, gibt der Shapeprovider den **gesamten Inhalt** von **n** Tabellen an den Client (oder den Server der mittleren Schicht) zurück – oder er versucht es wenigstens.

Wie erstellen Sie also eine einfache **Shape**-Anweisung? Nun, Sie können hierzu den Datenumgebungs-Designer einsetzen oder eine Anweisung aus meinem unten gezeigten Code kopieren. Ich werde auch nicht auf die Syntax der **Shape**-Anweisung eingehen – hierzu gibt es ein halbes Dutzend MSDN-Artikel. Siehe hierzu die allgemeinen **Shape**-Befehle in der MSDN-Dokumentation für Anfänger. Offen gesagt, dort wird mehr gesagt, als Sie zum Ausführen der hier gezeigten Beispiele wissen müssen.

Richten Sie zunächst einige allgemeine Deklarationsvariablen ein:

```
Dim cnn As ADODB.Connection
Dim rst As ADODB.Recordset
Dim rstTitleAuthor As ADODB.Recordset
```

Als Nächstes öffnen Sie eine Verbindung zu Ihrem bevorzugten SQL Server – einem, auf dem die **Pubs**-Datenbank geladen ist. Beachten Sie, dass Sie ein ADO-**Connection**-Objekt erstellen und die zugehörige **Provider**-Eigenschaft vor dem Öffnen auf **MSDataShape** setzen müssen. Sie können entweder den standardmäßigen OLE DB-Datenprovider für ODBC oder einen OLE DB-Provider für SQL Server verwenden. Dies bedeutet, dass Ihre Verbindungszeichenfolge aussehen sollte wie immer. Meine Verbindungszeichenfolge verweist auf einen DSN, Sie können aber auch eine DNS-lose Verbindung verwenden.

```
Set cn = New Connection
cn.Provider = "MSDataShape"
cn.Open "Data Provider=sqloledb;data source=(local);initial catalog=pubs;",
"admin", "pw"
```

Nachdem die Verbindung hergestellt wurde, können Sie ein neues Recordset erstellen und die **StayInSync**-Eigenschaft auf **False** setzen, um das Generieren zusätzlicher Abfragen zu verhindern. Hierzu komme ich später noch.

```
Set rs = New Recordset
rs.StayInSync = False
```

Im nächsten Schritt müssen Sie den Shapeprovider unter Verwendung der **Shape**-Syntax mit einer ordnungsgemäß erstellten Abfrage versehen. Wir übergeben diese Abfrage mit Hilfe der **Recordset Source**-Eigenschaft an den Provider, genau wie bei jeder anderen Abfrage. Berücksichtigen Sie jedoch, dass der Shapeprovider nicht viel mit einem Abfrageprozessor gemeinsam hat, halten Sie sich also etwas zurück. Dies heißt nicht, dass Sie keine komplexen Abfragen übermitteln können; diese werden vom Shapeprovider zur Verarbeitung einfach an das Backend weitergeleitet. Es bedeutet, dass einige der Tricks, die im Kapitel zu den Recordsets besprochen wurden, hier nicht anwendbar sind.

```
rs.Source = "SHAPE {select au_id,au_lname, au_fname, _
            Address, city, state, zip from authors} " _
            & " APPEND ({select * from titleauthor}" _
            & " RELATE au_id TO au_id) AS chapTitleAuthor"
rs.Open , cn, Options:=adCmdText
```

Im ersten in Klammern ({}) eingeschlossenen Ausdruck nach der **Shape**-Anweisung müssen Sie eine SQL SELECT-Anweisung (oder eine gespeicherte Prozedur) verwenden, über die das übergeordnete Resultset (Rowset) zurückgegeben wird. In unserem Beispiel fordern wir sämtliche Zeilen der **authors**-Tabelle an.

Anmerkung Natürlich ist das Auswählen der gesamten **authors**-Tabelle nicht besonders clever. Niemand, der einigermaßen bei Trost ist, würde sämtliche Spalten und alle Zeilen einer Produktionstabelle anfordern – es sei denn, die Tabelle weist so wenige Zeilen wie die Beispieldatenbank **Pubs** auf. Wir werden dieses Problem später lösen.

Nach Erstellung des übergeordneten Resultsets gibt der Shapeprovider dieses wie ein normales Recordset an die Anwendung zurück. Der Shapeprovider hängt jedoch ein zusätzliches **Field**-Objekt an das Ende des Recordsets. Wenn also Ihre Tabelle über acht Spalten verfügt, werden für jede Zeile neun **Field**-Objekte zurückgegeben. Dieser zusätzliche (letzte) **Field.Value** enthält ein weiteres **Recordset** – mit den Zeilen der untergeordneten Zeilen, die mit der übergeordneten Zeile verknüpft sind. Auf diese Weise enthält jede übergeordnete Zeile nicht nur die eigenen Daten, sondern auch sämtliche Daten der untergeordneten Elemente, die diesen Elementen untergeordneten Elemente, usw. Abbildung 10.2 zeigt, dass jede von der **authors**-Tabelle zurückgegebene Zeile ein zusätzliches Feld enthält, mit dem auf die Zeile(n) in der **titleauthor**-Tabelle verwiesen wird, die dem jeweiligen Autor zugeordnet sind.

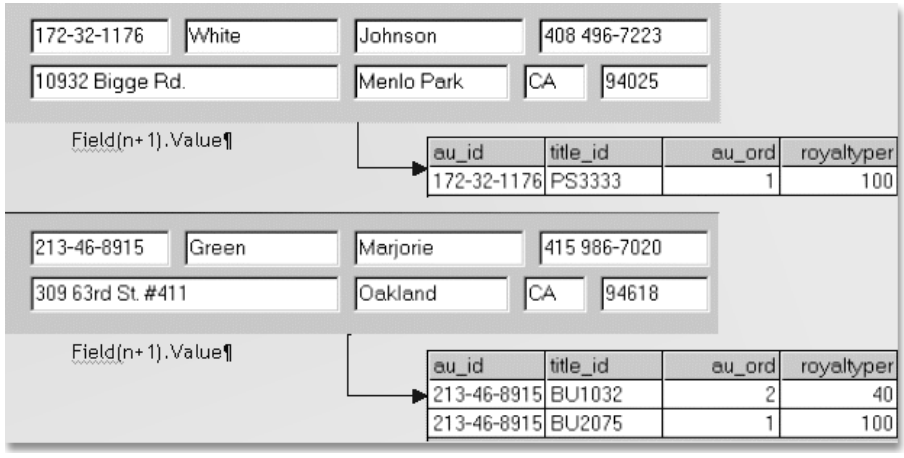


Abbildung 10.2 Übergeordnete Zeilen in der »author«-Tabelle mit den verknüpften untergeordneten Zeilen in der »titleauthor«-Tabelle

Die nächste Zeile in der SQL-Abfrage, die an den Shapeprovider übergeben wird, sollte eine APPEND-Anweisung enthalten, mit der dem Shapeprovider mitgeteilt wird, welche Zeilen der untergeordneten Tabelle (im Beispiel die **titleauthor**-Tabelle) für jede übergeordnete Zeile der Autorentabelle (**authors**) abgerufen werden sollten. Hierzu fügen wir die **APPEND ({select * from titleauthor}** in die Abfrage ein. Ja, Sie können mehrere APPEND-Anweisungen in die **Shape**-Abfrage einbauen, aber begnügen wir uns fürs Erste mit einer Unterordnungsbeziehung. Hier die gleiche SQL-Anweisung **Shape** mit markierter APPEND-Klausel.

```
rs.Source = "SHAPE {select au_id,au_lname, au_fname, _
Address, city, state, zip from authors} " _
            & " APPEND ({select * from titleauthor}" _
            & " RELATE au_id TO au_id) AS chapTitleAuthor"
```

Die letzte Zeile in der SQL-Abfrage ist eine RELATE-Anweisung, mit der dem Shapeprovider mitgeteilt wird, wie die zwei Rowsets verknüpft werden müssen.

Anmerkung In Abbildung 10.2 wird veranschaulicht, wie die Beziehungen zwischen primären und Fremdschlüsseln in der **authors**- und **titleauthors**-Tabelle mit der Spalte **au_id** verknüpft sind.

Im vorliegenden Fall muss das Feld **au_id** (Autoren-ID) der zwei Tabellen übereinstimmen. Dies bedeutet, dass der Shapeprovider für jede Zeile in der Autorentabelle sämtliche Zeilen der **titleauthor**-Tabelle abruft, die den gleichen **au_id**-Wert aufweisen, und aus diesen Zeilen ein untergeordnetes Recordset erstellt. Wir benennen diese Beziehung mit der AS-Klausel in der **Shape**-Anweisung. Dieser

Name wird dem letzten (zusätzlichen) **Field**-Objekt des übergeordneten Recordsets zugeordnet, damit dieses nach Name referenziert werden kann.

Tipp Ja, es ist der Shapeprovider, der für den Abgleich dieser Zeilen sorgt – nicht SQL Server, Jet oder ein anderes der Backends, die Sie üblicherweise einsetzen. Dies bedeutet, dass der Shapeprovider die Zeilen in den Speicher laden muss (in den RAM des Clientsystems).

Nachdem das Recordset zurückgegeben wurde, können Sie die Datensätze wie gewohnt durchlaufen. Vergessen Sie nicht, dass das Auffüllen des Recordsets umgehend abgeschlossen werden muss. Auf diese Weise werden eingerichtete Sperren aufgehoben, und auch andere Benutzer können die Daten verwenden – besonders, da eine Abfrage ohne festes Ende zum Erstellen des übergeordneten Resultsets eingesetzt wurde.

Im Rahmen des Beispiels habe ich ein **TextBox**-Steuerelementarray mit den Werten des übergeordneten Recordsets gefüllt, indem ich die **TextBox**-Steuerelemente an ein ADO-Datensteuerelement (ADC) gebunden habe, mit dem Aktualisierbarkeit bereitgestellt und die Navigation im übergeordneten Recordset ermöglicht wird. Die **TextBox**-Steuerelemente und ADC wurden zur Entwurfszeit nicht initialisiert – die erforderlichen Eigenschaften werden allesamt im Code festgelegt.

```
    '  Binden des Stammrecordsets an die Textfelder
For i = 0 To rst.Fields.Count - 2
      txtFields(i).DataField = rst.Fields(i).Name
Next I
```

Wir stellen die ADC-**Recordset**-Eigenschaft auf das durch den Shapeprovider zurückgegebene übergeordnete Recordset ein. Auf diese Weise werden die **TextBox**-Steuerelemente gefüllt, und es ist möglich, im SELECT * FROM Authors-Rowset einen Bildlauf nach oben und unten durchzuführen.

```
Set Adodc1.Recordset = rst
```

Wir referenzieren in jeder Zeile das untergeordnete Rowset, indem wird das Recordset aus dem zusätzlichen **Field**-Objekt extrahieren, das sich am Ende der Felder befindet, die für jede übergeordnete Zeile zurückgegeben werden. Beachten Sie, dass dieses **Field**-Objekt in der RELATE-Klausel der **Shape**-Anweisung benannt wird.

```
Set rstTitleAuthor = rst.Fields("chapTitleAuthor").Value
```

Anmerkung Das Referenzieren eines **Field**-Objekts nach Name stellt die langsamste Methode dar. Diese Technik kann bis zu sechsmal langsamer sein als das Referenzieren des **Field**-Objekts nach Nummer. In einer Hochleistungsanwendung sollte eine Referenzierung nach Ordnungszahl erfolgen, beispielsweise `rst.Fields(rst.Fields.Count-1).Value`. Diese kleine Änderung bewirkt, unter Berücksichtigung der Anzahl der **Field**-Objekte, einen erheblichen Unterschied in der Gesamtleistung.

Für das vorliegende Beispiel fügte ich dem Formular zur Anzeige der untergeordneten Zeilen ein Microsoft Hierarchical FlexGrid-Steuerelement (**MSHFlexGrid**) hinzu. Dies ist einfach – Sie müssen das untergeordnete Recordset lediglich der **DataSource**-Eigenschaft zuordnen. Nach der Zuordnung wird die Tabelle mit dem aktuellen Satz untergeordneter Zeilen gefüllt.

```
Set MSHFlexGrid1.DataSource = rstTitleAuthor
```

Könnten wir nicht mit Hilfe des **MSHFlexGrid**-Steuerelements sowohl die übergeordneten als auch die untergeordneten Zeilen anzeigen? Sicher. Diese Technik werde ich später erläutern. Aber eines nach dem anderen.

Bei Verwendung von ADC zum Wechseln von einer Zeile zur anderen müssen wir das **MSHFlexGrid**-Steuerelement über das anzuzeigende Recordset informieren. Dies wird über das **MoveComplete**-Ereignis erreicht, wie nachfolgend gezeigt wird.

```
Private Sub Adodc1_MoveComplete(ByVal adReason As ADODB.EventReasonEnum, ByVal
pError As ADODB.Error, adStatus As ADODB.EventStatusEnum, ByVal pRecordset As
ADODB.Recordset20)       ' Beachten Sie, dass wir statt einem Recordset ein
Recordset20 übergeben.
' Verwenden Sie hier die schnellere Technik.
      Set rsTitleAuthor = rs(rs.Fields.Count - 1).Value
      Set MSHFlexGrid1.DataSource = rsTitleAuthor
End Sub
```

Wir erhalten ein Formular (wie dargestellt in Abbildung 10.3), das im oberen Bereich in verschiedenen gebundenen **TextBox**-Steuerelementen eine einzelne übergeordnete Zeile darstellt, deren untergeordnete Zeilen in einem **MSHFlexGrid**-Steuerelement im unteren Bereich angezeigt werden.

Anmerkung Nachdem Sie der ADODC-Ereignisbehandlungsroutine diesen Code hinzugefügt haben, kann Ihre Anwendung nicht mehr kompiliert werden. Visual Basic erzeugt den Fehler »Prozedurdeklaration stimmt nicht mit Prozedur oder Ereignis gleichen Namens überein.« Dies ist ein Bug, aber zu erwarten, da wir den ADO 2.5- (oder 2.1-) MDAC-Stack verwenden. Denken Sie daran, dass das **Recordset**-Objekt von Version 2.0 auf 2.1 geändert wurde, daher tritt dieses Problem bei neueren Versionen immer auf. Wir müssen zur Umgehung dieses Problems die ADODC-Ereignisbehandlungsroutine dazu zwingen, die ADO 2.0-Version des **Recordset**-Objekts zu akzeptieren. Beachten Sie, wie dies durch das Umcodieren der **MoveComplete**-Ereignisbehandlungsroutine erreicht wird, die nun **Recordset20** anstelle von **Recordset** erwartet. Falls dieser Bug mit Visual Basic 6.0 SP4 behoben wird, sollte dies nicht länger ein Problem darstellen.

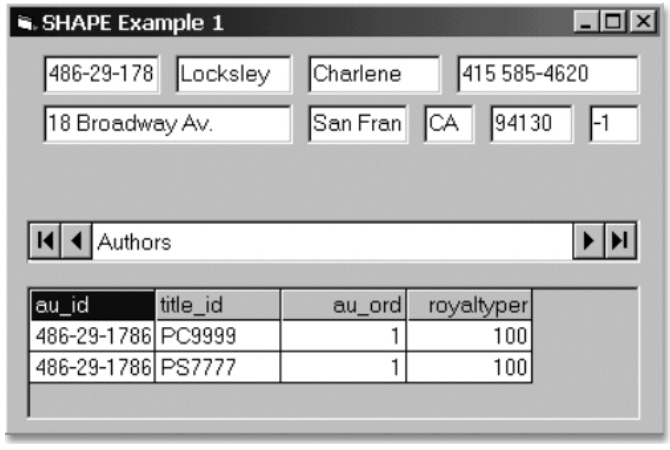


Abbildung 10.3 Verwenden gebundener »TextBox«-Steuerelemente mit »MSHFlexGrid«

10.3 Warum nicht den Datenumgebungs-Designer verwenden?

Wir hätten mit dem **MSHFlexGrid**-Steuerelement auch die übergeordneten **und** alle untergeordneten Zeilen anzeigen können. Ich habe diesen Ansatz aus mehreren Gründen nicht gewählt.

- Die Tabelle ist im Hinblick auf deren Einrichtung im Code ein PIA[1], und da ich den Datenumgebungs-Designer nicht zum Erstellen der Überordnungs/Unterordnungs-Beziehungen verwendet habe (was ich hätte tun können), müsste ich die Tabelleneigenschaften überarbeiten, um eine ordnungsgemäße Einrichtung

1. PIA = Pain in the Ass, technischer Fachausdruck.

zu gewährleisten. Die Tabelleneigenschaften können zur Laufzeit sowieso kaum geändert werden. Es gibt beispielsweise keine (offensichtliche) Möglichkeit zum Ändern der Zellenbreite (Tabellenspalten in einer Zeile) zur Laufzeit (oder Entwurfszeit).

- Die Hilfethemen für die Tabelle implizieren die Verwendung des Datenumgebungs-Designers und zeigen nicht wirklich auf, wie eine interaktive Einrichtung über die Eigenschaftenseiten der Tabelle vorgenommen wird.

Anmerkung Ich habe mit Fokus auf dem **MSHFlexGrid**-Steuerelement F1 gedrückt – in der Hoffnung, dass Hilfethemen zu diesem Steuerelement angezeigt werden. Dies war nicht der Fall. MSDN beschwerte sich folgendermaßen: »Die kompilierte Hilfedatei (.chm) enthält keine Kontext-IDs.« Ich wette, dies liegt daran, dass ich die aktuellste Version (Juli 1999) von MSDN verwende. Seufz. Es gibt eine Fülle von Hilfethemen zu **MSHFlexGrid** – führen Sie einfach unter diesem Stichwort einen Suchlauf in der MSDN-Dokumentation durch. Ich habe 166 Einträge gefunden, in denen **MSHFlexGrid** vorkam.

- Falls Sie diesen nicht anders instruieren, verwendet der Datenumgebungs-Designer nicht die richtigen (standardmäßigen) Eigenschaften für die ADO-Objekte **Connection** oder **Recordset**. Dies bedeutet, dass Sie den Designer daran erinnern müssen, den Benutzer während des Öffnens nicht zur Eingabe von Benutzer-ID und Kennwort aufzufordern.
- Falls nicht anders codiert, startet der Datenumgebungs-Designer sich selbst vor Ausführung von **Form_Load**. Dies bedeutet, dass bei Auftreten eines Fehlers keine entsprechende Behandlungsroutine vorhanden ist. Das Deaktivieren des Datenumgebungs-Designers ist keine einfache Aufgabe, wenn die gebundenen Steuerelemente zur Entwurfszeit automatisch synchronisiert werden sollen.
- Der Datenumgebungs-Designer wartet nicht auf die Übergabe von Parametern an die auszuführende Abfrage. Da (fast) ausschließlich parameterbasierte Abfragen ausgeführt werden sollen, funktioniert dies schlichtweg nicht.
- Selbst wenn wir das **WillExecute**-Ereignis auffangen und die Parameter für die erste Ausführung übergeben, müssen sämtliche gebundene Steuerelemente (wie z. B. **TextBox**-Steuerelemente) manuell erneut an das neu erstellte Recordset des Datenumgebungs-Designers gebunden werden (im Code). Obwohl dies mit dem Steuerelement **MSHFlexGrid** problemlos umsetzbar ist (es handelt sich um ein komplexes gebundenes Steuerelement), wirft dies doch bei vielen meiner Entwürfe Probleme auf.

Der Datenumgebungs-Designer-Ansatz wirkt verlockend. Mit diesem können aufsehenerregende Demos erstellt werden, und Sie erhalten eine Komponente, die sämtliche der übergeordneten Zeilen und jeweils direkt darunter die untergeordneten Zeilen anzeigt und ohne Code auskommt. Abbildung 10.4 zeigt, wie das **MSHFlexGrid**-Steuerelement nach dem Auffüllen über eine Datenumgebungs-Designer-basierte Abfrage aussieht. Beachten Sie, dass die einzelnen Zeilen bereits erweitert dargestellt werden. Mit Hilfe der **CollapseAll**-Methode können die untergeordneten Zeilen ausgeblendet werden. Durch Klicken auf das Pluszeichen (+) werden die untergeordneten Zeilen erweitert.

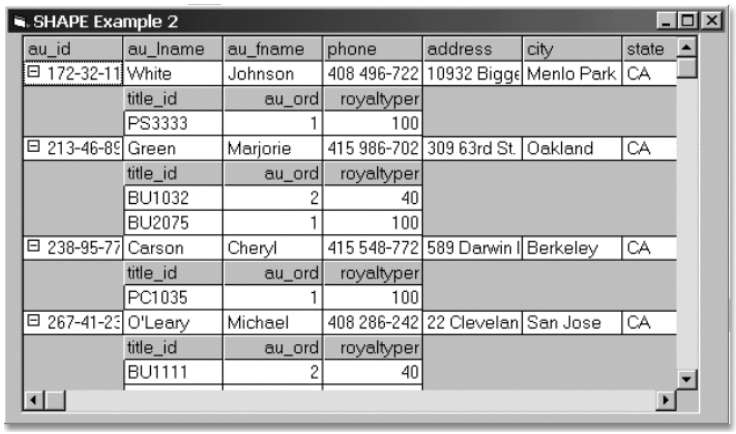


Abbildung 10.4 Ein »MSHFlexGrid«-Steuerelement, das über eine Datenumgebungs-Designer-basierte Abfrage aufgefüllt wurde

Zugegeben, dies ist ziemlich cool. Und die Einrichtung per Drag&Drop-Technik dauert auch nicht besonders lange. Die Auswirkungen auf Ihr System sind jedoch grundlegend. Um die Flexibilität, Leistung und Skalierbarkeit zu erzielen, die Sie sicherlich erwarten, benötigen Sie einen codebasierten Entwurf.

Tipp Wenn Sie darauf bestehen, den Datenumgebungs-Designer zum Erstellen der Beziehungen sowie zum Binden des **MSHFlexGrid**-Steuerelements einzusetzen, sollten Sie sicherstellen, dass die Eigenschaften des Datenumgebungs-Designer-**Connection**-Objekts mit Hilfe der zugehörigen Visual Basic-basierten Eigenschaftenseite zurückgesetzt werden, wie dargestellt in Abbildung 10.5. Setzen Sie **RunPromptBehavior** und **DesingPromptBehavior** auf den Wert **4 – adPromptNever**. Dies ermöglicht Ihrem Code die Erfassung von Fehlern zur Anmelde-ID, anstatt über ODBC (oder OLE DB) Dialogfelder zur Anmelde-ID anzuzeigen. Diese Dialogfelder geben dem Benutzer die Möglichkeit, Anmelde-ID und Kennwort solange zu raten, bis sie richtig liegen – etwas, dass ich nach Möglichkeit vermeide.

Abbildung 10.5 Die (Visual Basic-) Eigenschaftenseite des »Connection«-Objekts für den Datenumgebungs-Designer

10.4 Analysieren der Auswirkungen

Okay, halten wir uns nicht länger mit dem Datenumgebungs-Designer und **MSHFlexGrid** auf. Die Stärken und Schwächen des Datenumgebungs-Designers werden in Kapitel 12 im Rahmen der Visual Database Tools noch genauer erläutert.

Bevor wir fortfahren, sollten wir prüfen, welche Auswirkungen das Standardprogramm (das Demo) auf den Server ausübt. Anschließend vergleichen wir diese Auswirkungen mit den Beulen, die das erste Beispiel hinterlassen hat. Dann untersuchen wir, was wir zum Verarzten dieser Beulen tun können.

Denken Sie daran, dass der Datenumgebungs-Designer in der anfänglichen Abfrage nicht eingesetzt wurde – es wurde die gleiche Abfrage ohne festes Ende zum Zurückgeben der Zeilen eingesetzt: zwei SELECT * FROM <table>-Abfragen, wie sie durch den Datenumgebungs-Designer generiert werden. Mit Hilfe des SQL Server 7.0 Profilers können wir ermitteln, welche Auswirkungen diese Abfragen auf SQL Server haben. Der Profiler zeigt die unverarbeiteten Transact SQL-Abfragen (TSQL) und weitere Operationen (beispielsweise das Öffnen von Verbindungen) an, aber dies wissen Sie ja bereits.

Anmerkung Ich habe bei der Entwicklung der datenquellenbasierten Beispiele (unter Verwendung von ADC und dem Datenumgebungs-Designer) festgestellt, dass der Server gelegentlich anfing, ohne ersichtlichen Grund Verbindungsfehler zurückzugeben. Der Profiler teilte mir mit, dass ADO, ADC oder Visual Basic es beim Drücken der **Stopp**-Schaltfläche in der Visual Basic-IDE oder selbst nach dem normalen Beenden des Codes nicht für nötig befunden hatten, die Verbindung zu schließen und wieder freizugeben. Ich versuchte vergeblich, Code zum expliziten Schließen des **Recordset**- und **Connection**-Objekts hinzuzufügen. Ich musste Visual Basic vollständig beenden, um die **Connection**-Objekte zu schließen. Leute, dies ist das gleiche Problem, über das ich in meinem ersten Buch über VBSQL berichtet habe – vor sieben Jahren.

Sehen wir uns die Profiler-Ausgabe für (Teile) des ausgeführten Beispiels an.

```
sp_prepare @P1 output, NULL, N'select * from titleauthor', 1 select @P1
select au_id,au_lname, au_fname, Address, city, state, zip from
authors;select * from titleauthor
```

Der Profiler informiert uns darüber, dass ADO SQL Server angewiesen hat, einen Abfrageplan für die **titleauthor**-Abfrage »vorzubereiten« und anschließend die zwei SELECT-Abfragen ohne festes Ende für **authors** und **titleauthors** auszuführen. Das ist alles. Der Shapeprovider akzeptiert sämtliche Zeilen beider SELECT-Abfragen und gibt die Daten zur Verarbeitung an die Anwendung zurück. ADO führt keine weiteren Serveranfragen aus. Dies bedeutet, dass der Shapeprovider diese Daten als **statisch** ansieht – es wird beim Durchlaufen der Zeilen nicht der Versuch unternommen, aktualisierte Daten zu erhalten.

Anmerkung Die mit Hilfe des Drag&Drop-Verfahrens über den Datenumgebungs-Designer generierten Abfragen sind mit denen identisch, die über den Code für das hartcodierte Beispiel 1 (siehe oben) generiert werden.

Da beide Tabellen recht klein sind (die Autorentabelle enthält etwa 22 Zeilen, die Tabelle mit den Autorentiteln weist ca. 25 Zeilen auf), werden lediglich um die 50 Zeilen abgerufen – keine große Sache. Was geschieht jedoch, wenn 16.000 Autoren und 50.000 Zeilen mit Autorentiteln vorhanden wären? Aus ersichtlichen Gründen würde der Ansatz nicht funktionieren, oder zumindest wäre er nicht besonders schnell. Da es sich um Abfragen ohne festes Ende handelt, würde sich des Weiteren die Beeinträchtigung in der Skalierbarkeit deutlich niederschlagen. Zusätzlich muss berücksichtigt werden, das es sich um die einfachste Form der Strukturierung handelt – es sind lediglich zwei Elemente vorhanden, ein

übergeordnetes und ein untergeordnetes Objekt. Was geschieht jedoch, wenn zwei oder mehr untergeordnete Objekte vorhanden sind? Die Anzahl der über das Netzwerk an den Client zurückgesendeten Zeilen würde innerhalb kürzester Zeit erschreckend ansteigen.

10.5 Auf der Suche nach einer besseren Lösung

Mit dieser Vorstellung im Hinterkopf machte ich mich auf die Suche nach einer akzeptableren Lösung, bei der nur die tatsächlich benötigten Zeilen abgerufen werden. Gleichzeitig sollte der Shapeprovider weiterhin seine Arbeit tun können – er sollte weiterhin die 1:n-Beziehungsverwaltung im Speicher übernehmen und Aktualisierbarkeit in den (wenigen) Fällen bieten, in denen eine Aktualisierung per Cursor erfolgen sollte.

Klar ist, dass die Anzahl der abgerufenen Zeilen durch die anfängliche »übergeordnete« Abfrage eingeschränkt werden muss. Nachdem dies erledigt ist, kann damit begonnen werden, die untergeordneten Zeilen »nach Bedarf« abzurufen – oder können wir das nicht? Nun, es ist möglich, aber Sie müssen eine andere Funktion des Shapeproviders nutzen, die als nächste erläutert werden soll – das Verwenden eines Indexes zum Herstellen einer Querverbindung zwischen über- und untergeordnetem Element.

Das Beschränken der anfänglichen Abfrage erscheint recht einfach – fügen Sie einfach eine WHERE-Klausel ein. Wir möchten (in den meisten Fällen) jedoch keine Hartcodierung der WHERE-Klausel vornehmen, deshalb habe ich versucht, ein ADO-**Command**-Objekt zum Generieren der **Shape**-Recordsets zu erstellen. Vergessen Sie es, geht nicht. Die gute alte Technik der »Verkettung im Vorübergehen« funktionierte jedoch. Durch das Beschränken der Autorenabfrage auf die Rückgabe eines einzelnen Autors zwang ich ADO dazu, immer dann einen neuen Abruf auszuführen, wenn ein anderer Autor angezeigt werden sollte. Sie könnten auch eine Abfrage einrichten, mit der nur ausgewählte Gruppen übergeordneter Zeilen zurückgegeben werden – beispielsweise nur die kalifornischen Autoren. Wir werden noch über die Verwendung gespeicherter Prozeduren zur Handhabung dieses Problems sprechen.

Selbst wenn jedoch nur eine einzelne übergeordnete Zeile zurückgegeben wird, werden über die untergeordnete Abfrage weiterhin alle Zeilen der **titleauthors**-Tabelle zurückgegeben, nicht lediglich die Übereinstimmungen mit der übergeordneten Tabelle. Ich dachte mir, dass der Shapeprovider über eine Möglichkeit zum Abfragen der untergeordneten Zeilen nach Bedarf verfügen **muss**, und das stimmt. Wenn Sie die **Shape**-Anweisung so umformulieren, dass ein Parameter

der übergeordneten Abfrage an die untergeordnete Abfrage übergeben wird, ändern ADO und der Shapeprovider ihre Strategie und rufen die untergeordneten Zeilen nach Bedarf ab. So sieht die parameterbasierte Syntax aus:

```
rst.Source = "Shape  {select au_id, au_lname, au_fname, city, state, zip" _
             & " From authors where state ='" & txtStateWanted & "'} " _
               & " APPEND ({select * from titleauthor where au_id = ?}" _
               & " RELATE au_id TO Parameter 0) AS chapTitleAuthor"
```

Anmerkung Der Ansatz, bei dem alle Zeilen auf einmal abgerufen werden, kann in Situationen erforderlich sein, in denen Sie sich nicht auf die Dauerhaftigkeit der Verbindung verlassen können, beispielsweise bei Webseiten oder Komponenten der mittleren Schicht. Dies bedeutet nicht, dass Sie auf Abfragen ohne festgelegtes Ende zurückgreifen müssen. Es gibt weitere Möglichkeiten zum Schreiben von Abfragen, mit denen Daten nach Bedarf abgerufen werden.

Ich erfasse den gewünschten Bundesstaat in einem **TextBox**-Steuerelement (siehe Beispiel 3[2]) und füge diesen in die Abfrage ein, sobald der Benutzer auf die Schaltfläche zum Suchen klickt. Beachten Sie, dass durch den Code die **au_id** in der übergeordneten Abfrage mit einem **?** in der untergeordneten Abfrage in der RELATE-Klausel verknüpft wird.

Können Sie anstelle der SELECT-Anweisung(en) in den über- und untergeordneten Abfragen einfach eine gespeicherte Prozedur ausführen? Sicher. In Beispiel 4 finden Sie ein funktionierende Umsetzung dieses Ansatzes.

```
rst.Source = "Shape {execute GetAuthorsByState '" & txtStateWanted & "'} " _
             & " APPEND ({select * from titleauthor where au_id = ?}" _
             & " RELATE au_id TO Parameter 0) AS chapTitleAuthor"
```

10.6 Die Auswirkungen des Bedarfsansatzes

Sehen wir uns erneut an, was der Profiler zu den parameterbasierten Abfragen zu melden hat. Ich habe hier etwas Interessantes entdeckt, das Sie wahrscheinlich genauso überraschen wird wie mich.

Die gesendeten TSQL-Anweisungen scheinen sehr sinnvoll zu sein – zumindest auf den ersten Blick.

Als Erstes fordert ADO SQL Server zur Rückgabe von Metadaten zur **titleauthor**-Abfrage auf.

2. Sämtliche der genannten Beispiele befinden sich auf der Begleit-CD. Die Beispiele aus Kapitel 10 befinden sich unter **..\Sample Applications\Hierarchy and Shape\Shape**.

```
SET FMTONLY ON
select * from titleauthor
SET FMTONLY OFF
```

Als Nächstes führt ADO mit Hilfe der im Code (über das **TextBox**-Steuerelement auf dem Formular) bereitgestellten verketteten Parameter die neue parameterbasierte Abfrage aus. Auf diese Weise wird ein anfängliches Rowset zurückgegeben, das lediglich die kalifornischen Autoren enthält.

```
select au_id,au_lname, au_fname, phone, Address, city,
state, zip from authors where state ='CA'
```

Jetzt führt ADO eine Serveranfrage aus, um die Metadaten für die schlüsselbasierte Abfrage der **titleauthor**-Tabelle zu ermitteln. Bis jetzt sind es also drei zusätzliche Serveranfragen.

```
SET FMTONLY ON
select  au_id from titleauthor
SET FMTONLY OFF
```

An diesem Punkt erstellt ADO eine temporär gespeicherte Prozedur, die dem Zeilenabruf aus der **titleauthor**-Tabelle dient. Diese Abfrage wird ausgelöst, wenn eine spezielle Zeile der **authors**-Tabelle ausgewählt wird. Da eine ganze Reihe von Autorenzeilen ausgewählt wurde, können wir damit rechnen, dass diese Abfrage des öfteren ausgeführt wird – ein Mal für jede Zeile im übergeordneten Recordset. Beachten Sie, dass die abschließende SELECT-Anweisung in diesem Stapel lediglich den Parameter **@P1** der Anweisung **sp_prepare** zurückgibt.

```
sp_prepare @P1 output, N'@P1 varchar(11)', N'
select * from titleauthor where au_id = @P1', 1
select @P1
```

Der letzte durch den Profiler zurückgegebene TSQL-Code ist (endlich!) der Abruf für unsere erste übergeordnete Zeile. Das heißt, der Shapeprovider weiß, dass die **au_id** für die erste Zeile **172-32-1176** lautet, daher wird der Server nach Titeln abgefragt, die mit dieser Autoren-ID übereinstimmen.

```
sp_execute 1, '172-32-1176'    17:02:13.987
```

Nun, wir sind noch nicht fertig. An diesem Punkt haben wir lediglich zur ersten Zeile des übergeordneten Recordsets gewechselt und den ersten Satz untergeordneter Zeilen (mit Autorentiteln) abgerufen. Beim Abrufen zusätzlicher Zeilen würden wir erwarten, dass ADO und der Shapeprovider die temporär gespeicherte Prozedur #1 (wie oben erstellt) zum Abrufen zusätzlicher Zeilen aus der **titleauthor**-Tabelle wieder verwenden. Traurigerweise ist dies nicht der Fall. Wie aus

der Profiler-Ausgabe ersichtlich wird, weist der Shapeprovider SQL Server an, die zuvor verwendete temporär gespeicherte Prozedur zu löschen (**unprepare**) und eine identische Prozedur zu erstellen. Seufz. Anschließend wird die neue temporär gespeicherte Prozedur zum Zurückgeben des nächsten Satzes untergeordneter Zeilen verwendet.

```
sp_unprepare 1
sp_prepare @P1 output, N'@P1 varchar(11)', N'
select * from titleauthor where au_id = @P1', 1
select @P1   17:21:01.367
sp_execute 2, '213-46-8915'
```

Dies war eine Überraschung für mich. Ich hatte erwartet, dass der Shapeprovider in der Lage ist, wenigstens die Arbeit zu nutzen, die bereits in das Erstellen der Abfragen für untergeordnete Zeilen gesteckt wurde. Wie Sie sehen, weist der Bedarfsansatz eigene Probleme auf. Es werden mehr zusätzliche Serveranfragen pro Zeile durchgeführt als beim Ansatz, der alle Zeilen abruft und der jetzt offen gesagt wieder etwas mehr an Attraktivität gewinnt. Der Ansatz »nach Bedarf« ruft zwar die aktuellen Daten für jede untergeordnete Zeile ab, hängt jedoch von einem statischen Satz übergeordneter Zeilen zum Ausführen der sekundären Abfrage ab.

Wie Sie sehen, kann das Hinzufügen zusätzlicher untergeordneter Elemente zu einer Hierarchie recht interessant sein.

Abfragen der Form Kunden-Aufträge-Artikel wären etwa ein übliches Beispiel.

Des Weiteren wird ersichtlich, dass eine zusätzliche Komplexität in der Hierarchie zu einem exponentiellen Overheadanstieg führt.

Das Kapitel fehlt in dieser PDF-Datei.

Sorry, aber wir wollen ja auch das eine oder andere Buch verkaufen :-)

Index

B

C

D

E

F

G

N

O

P

R

S

T

U

V

W

X

Z