ASP-Programmierung mit ADO

Jörg Krause

ASP-

Programmierung mit ADO

ADDISON-WESLEY

An imprint of Pearson Education

München • Bosten • San Francisco • Harlow, England
Don Mills, Ontario • Sydney • Mexico City
Madrid • Amsterdam

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Ein Titeldatensatz für diese Publikation
ist bei Der Deutschen Bibliothek erhältlich.

Die Informationen in diesem Produkt werden ohne Rücksicht auf einen eventuellen Patentschutz veröffentlicht. Warennamen werden ohne Gewährleistung der freien Verwendbarkeit benutzt. Bei der Zusammenstellung von Texten und Abbildungen wurde mit größter Sorgfalt vorgegangen. Trotzdem können Fehler nicht vollständig ausgeschlossen werden. Verlag, Herausgeber und Autoren können für fehlerhafte Angaben und deren Folgen weder eine juristische Verantwortung noch irgendeine Haftung übernehmen.
Für Verbesserungsvorschläge und Hinweise auf Fehler sind Verlag und Herausgeber dankbar.

Alle Rechte vorbehalten, auch die der fotomechanischen Wiedergabe und der Speicherung in elektronischen Medien. Die gewerbliche Nutzung der in diesem Produkt gezeigten Modelle und Arbeiten ist nicht zulässig.

Fast alle Hardware- und Softwarebezeichnungen, die in diesem Buch erwähnt werden, sind gleichzeitig auch eingetragene Warenzeichen oder sollten als solche betrachtet werden.

Umwelthinweis:
Dieses Buch wurde auf chlorfrei gebleichtem Papier gedruckt.
Die Einschrumpffolie – zum Schutz vor Verschmutzung – ist aus
umweltfreundlichem und recyclingfähigem PE-Material.

10 9 8 7 6 5 4 3 2 1

04 03 02 01

ISBN 3-8273-1655-3

© 2001 by Addison-Wesley Verlag,
ein Imprint der Pearson Education Deutschland GmbH,
Martin-Kollar-Straße 10–12, D-81829 München/Germany
Alle Rechte vorbehalten
Einbandgestaltung: atelier für gestaltung, niesner & huber, Wuppertal
Lektorat: Sylvia Hasselbach, shasselbach@pearson.de
Herstellung: Anna Plenk, aplenk@pearson.de
Satz: reemers publishing services gmbh, Krefeld
Druck und Verarbeitung: Freiburger Graphische Betriebe, Freiburg
Printed in Germany

Schnellübersicht

Inhaltsverzeichnis

Vorwort

ADO – Active Data Objects – sind das wichtigste Handwerkzeug des ASP-Programmierers beim Umgang mit Datenbanken. In der Praxis habe ich oft beobachtet, wie ADO eingesetzt wird. Neben der oberflächlichen Nutzung einfacher Objekte wird es mit einer Mischung aus SQL-Abfragen und VBScript-Code kombiniert. Dabei entstehen – leider – haarsträubende Lösungen. Tatsächlich ist ADO viel leistungsfähiger und reichhaltiger, als viele Programmierer vermuten. Nahezu jedes Datenbankproblem lässt sich mit einer passenden Methode oder Eigenschaft elegant lösen. Oft führt dies zu einer besseren Performance – auch im Webspace.

Die Kenntnis der ADO-Objekte ist dennoch nicht unproblematisch. Die Komplexität ist enorm und nicht jede Methode hat einen konsequenten und konsistenten Stil in der Wahl der Syntax. Viele sind gar so komplex, dass sie nur sehr selten anzutreffen sind – trotz enormer Funktionalität, wie `OpenSchema` zum Beispiel.

Das vorliegende Buch behandelt diese Objekte, erklärt alle Parameter und zeigt bei schwierigeren Anwendungen passende Beispiele. Diese Skripte unterscheiden sich grundlegend von denen in der Online-Dokumentation. Dort wird versucht, in einem Skript alles zu zeigen, was an tollen Möglichkeiten existiert. Das ist aber eher verwirrend. Die Skripte in diesem Buch sind einfacher und zeigen überwiegend genau eine Funktion. Sie können dann damit spielen, bis das Skript entsprechend Ihrer Vorstellung funktioniert.

Die evolutionäre Weiterentwicklung von ADO heißt ADO.NET. Die Migration zum neuen Objektmodell der .Net-Offensive wird nur langsam und für neue Projekte vollzogen werden. Die vermutlich andauernde Koexistenz von ADO 2.6 und ADO.NET legt die Verarbeitung in einem Buch also nahe. Da ADO.NET mit ASP.NET zum Einsatz kommt, wird auch dazu ein Kapitel angeboten – vor allem in Ermangelung anderer Literatur. Sie halten somit ein Buch mit Investitionsschutz in den Händen – ADO.NET wird erst Mitte 2001 in der finalen Version verfügbar sein. Lernen Sie also ADO genau kennen und lesen Sie, was Ihnen die Zukunft als Softwareentwickler bringen wird.

Jörg Krause
Berlin, im Dezember 2000

Teil I
ADO 2.6

1 Über das Buch

In diesem Kapitel wird die Zielgruppe definiert und die grundlegenden Voraussetzungen zum Ausführen der Beispiel werden vorgestellt.

1.1 Zielgruppe

An wen sich dieses Buch wendet

Dieses Buch wendet sich an Programmierer, die bereits Erfahrungen mit VBScript und ASP haben und in Zukunft größere und komplexere Projekte bearbeiten müssen. Die Grundlagen von ASP und VBScript werden deshalb nicht behandelt. Beschrieben wird der Zugriff auf Daten, per ADO, ADO.NET und XML:

- ADO – die ActiveX Data Objects für den Zugriff auf Datenbanken, unter anderem mit den Spezialthemen:
  - Data Shaping-Sprache
  - ADOX
  - ADO MD
- ADO.NET – die neueste Generation aus Visual Studio 7
- XML – Datenspeicherung in der Extensible MarkUp Language

Diese Objektsammlungen sind komplexe Werkzeuge für den Programmierer. Deren Darstellung in der Online-Hilfe lässt sich getrost als konfus und sehr einseitig beschreiben. Dieses Buch versucht eine bessere Darstellung, damit die Zielgruppe auch einen wirklichen Nutzen daraus zieht – Programmierer, die ein Projekt schnell und gezielt umsetzen müssen.

Was Sie nicht finden

Was nicht reingepasst hat

Weniger Berücksichtigung fanden in diesem kompakten Buch Techniken, die veraltet sind (wie ODBC), sich so proprietär verhalten, dass der universelle Einsatz nicht gegeben ist (wie RDS) oder die für sich so komplex sind, dass sie ein eigenes Buchprojekt erfordern, wie beispielsweise C#.

1.2 Aufbau des Buchs

Wie Sie was finden

Das Buch unterscheidet sich in seinem Aufbau ein wenig von anderen Fachbüchern. Statt der Trennung in einen erklärenden Teil und eine Referenz wurde dem Charakter einer Referenz mehr Aufmerksamkeit geschenkt. Der Praktiker wird dies schnell bemerken, denn alle Befehle werden systema-

tisch erläutert. Im Gegensatz zu einer Referenz ist die Betrachtung so ausführlich, dass ein Zugriff auf weitere Quellen nicht mehr notwendig ist. Das Prinzip könnte man also als »One-Stop-Solution« beschreiben.

1.2.1 Struktur

Eine strenge Struktur hilft bei der Suche

Eine sehr strenge Struktur unterstützt diese Idee. Die Objekte werden systematisch beschrieben, zuerst die Methoden, dann die Eigenschaften. Beispiele erläutern jedes wichtige Element. Manchmal stehen Beispiele am Ende eines Abschnitts, wenn mehrere Methoden oder Eigenschaften gleichzeitig zum Einsatz kommen.

Drei Teile

Das Buch ist in drei Teile gegliedert, die sich jeweils einem grundlegenden Thema widmen:

- Teil I – ADO 2.6
- Teil II – ADO.NET
- Teil III – Konstanten und Referenzen

Die Reihenfolge ist nicht willkürlich gewählt. ADO bildet die grundlegende Technik zum Zugriff auf Datenbanken. ADO.NET stellt dagegen die revolutionäre Weiterentwicklung dar. Diese ist Anfang 2001 aber noch im Betastadium und sicher erst auf mittelfristige Sicht allgemein präsent. Betrachten Sie diese Kapitel deshalb als Investitionsschutz für dieses Buch.

Teil III listet alle wichtigen Konstanten auf, die intern durch numerische Werte repräsentiert werden. Hier finden Sie auch einen Index, der alle Methoden und Funktionen alphabetisch aufführt.

1.2.2 Symbole

Das Buch nutzt intensiv die Marginalspalten, um beim schnellen Durchblättern die Übersicht zu erleichtern. Dazu gehören auch einige Symbole, die Sie kennen sollten:

Dieses Symbol bezeichnet eine Position, an der die Syntax eines Befehls erläutert wird.

Dieses Symbol kennzeichnet Absätze mit Zusatzinformationen, die ein wenig aus dem aktuellen Thema herausführen.

1.2.3 Schreibweisen

Wenn Überschriften einen Befehl enthalten, erkennen Sie dies am grauen Hintergrund:

Wie Sie was im Text erkennen

GetString

GetString

Soweit es sich um Abschnitte mit reinen Syntaxbeschreibungen handelt, wird der Name auch in der Marginalspalte wiederholt. Dies dient vor allem zum schnelleren Auffinden auf der Seite.

Beispielcodes sind im Text ebenfalls grau hinterlegt. So finden Sie sofort die wertvollen Beispiele:

```
Set objConn = Server.CreateObject("ADODB.Connection")
```

Im Text sind einzelne Befehle mit einer nicht proportionalen Schrift hervorgehoben worden:

»Besonders interessant ist der Befehl `GetString` für die Erstellung von Tabellen.«

Wird dagegen auf einen Parameter Bezug genommen, der im Skript durch einen »echten« Wert ausgetauscht werden muss, wird dieser kursiv geschrieben:

»Setzen Sie hier für *name* den Namen und für *url* eine Adresse ein.«

Syntaxbeschreibung

Die Syntaxbeschreibungen haben im gesamten Buch einen einheitlichen Aufbau:

```
typ Rückgabe = Objekt.Methode(typ Parameter [, typ Option])
```

- `typ` bezeichnet den Datentyp, den das jeweilige Element hat oder den Typ des Rückgabewerts einer Funktion.
- *`Parameter`* wird durch einen konkreten Wert mit dem angegebenen Datentyp ersetzt. Das können Konstanten, Zeichenketten oder Zahlenwerte sein. Im Text wird der Parameter *kursiv* geschrieben.
- `[`*`Optionen`*`]` stehen in eckigen Klammern. Diese Angaben können entfallen.
- Die `Eigenschaft` oder `Methode`, um die es eigentlich geht, ist fett geschrieben.

Hier ein Beispiel für ein reales Syntaxdiagramm:

```
[Set object RS = ] objConnection.Open(string Connection
                                      [, string User]
                                      [, string Password]
                                      [, integer Options])
```

Die mehrzeilige Schreibweise soll nur die Lesbarkeit im Buch verbessern. Beim Kodieren schreiben Sie alles in eine Zeile. Das Diagramm hat nun folgende Bedeutung:

Der Rückgabewert `[Set object RS = ]` ist optional. Wenn er geschrieben wird, sieht das so aus:

```
Set objRS =
```

`object` weist also lediglich auf den Datentyp hin, *RS* ist ein Platzhalter, den Sie durch einen selbst gewählten Variablennamen ersetzen. Meist ist diese Schreibweise irritierend, dann habe ich den Datentyp durch einen Präfix gekennzeichnet, der der späteren Schreibweise eher entspricht: `objRS`.

objConnection ist kursiv und steht für eine Variable, auf die die Methode angewendet werden kann.

`Open` ist fett geschrieben – um diese Methode geht es hier.

Connection ist eine Zeichenkettenvariable (Typ `string`). Sie können natürlich auch die Zeichenkette direkt angeben.

User und *Password* sind auch Zeichenkettenvariablen oder Zeichenketten, die Angabe ist aber optional. Entfallen sie, wird auch das Komma nicht geschrieben (das ist nicht bei allen Befehlen der Fall!).

Options ist vom Typ `integer`, also eine Ganzzahl.

Und so könnte der Befehl in der Praxis angewendet werden:

```
Set objMyRS = objConn.Execute("SELECT * FROM artikel")
```

Möglich ist aber auch diese Version:

```
intRecords = 0
strQuery = "UPDATE artikel SET brutto = netto * 1.16"
objConn.Execute(strQuery, intRecords, _
                adCmdText And adExecuteNoRecords)
```

Die durch das _-Zeichen getrennte Zeile wurde nur für den Druck umgebrochen, schreiben Sie alles besser in eine Zeile.

1.3 Voraussetzungen

Welche Technik wird vorausgesetzt?

Die Beispiele in diesem Buch entstanden auf einer Entwicklungsumgebung, die auch beim Leser vorausgesetzt wird:

- Für ADO: Windows 2000 (Server oder Professional) mit Microsoft SQL Server 7 oder 2000 als Datenbankmanagementsystem. Sie können auch mit NT 4 arbeiten, müssen aber ADO 2.5 nachträglich installieren, da

NT 4 nur mit ADO 2.0 geliefert wird. Einige Beispiele zu ADO MD setzen Access 2000 voraus.

- Für ADO.NET: Visual Studio 7 (zumindest in der Preview-Version, mit der dieses Buch entstand). Das .net-SDK alleine reicht leider nicht aus.

Sie sollten mindestens über einen Server und eine Workstation verfügen, die miteinander per TCP/IP verbunden sind und beide Zugriff auf das Internet haben. Wenn Sie über »weniger« verfügen, werden einige Beispiele nicht funktionieren. Dies gilt in ganz besonderem Maße für den Ersatz des SQL Servers durch Access 2000.

Für die Entwicklung ist außerdem dringend eine gute Entwicklungsumgebung anzuraten. Für dieses Buch wurde mit Visual InterDev (VID) gearbeitet. Sie können aber auch einen anderen Editor verwenden, beispielsweise HomeSite 4.5. Durch die gute Integration der Datenumgebung ist VID aber klar im Vorteil, auch wenn einige Eigenheiten des Editors sicher nicht jeden Programmierer glücklich machen werden.

Voraussetzung ist natürlich auch ein installierter und laufender Webserver – konkret getestet wurden alle Beispiele mit dem IIS 5, der mit Windows 2000 geliefert wird. Und nicht zuletzt sei darauf verwiesen, dass der »einzig wahre« Browser der Internet Explorer ist. Ich habe mir nicht die Mühe gemacht, Designstudien mit Netscape zu betreiben ;-).

1.4 Betriebsumgebung der Skripte

Im Buch finden Sie viele Skripte, die aus Platzgründen nur in Ausschnitten wiedergegeben werden. Alle Skripte greifen auf eine Basisbibliothek zurück, die ich zu diesem Zweck entworfen habe. Sie finden diese hier ausführlich vorgestellt. Später wird darauf nur in Ausnahmefällen eingegangen.

1.4.1 Die Bibliothek open.inc.asp

Die Bibliothek OPEN.INC.ASP wird von allen Listings im Teil I verwendet. Sie enthält folgende Funktionen und Prozeduren:

- `open()`. *Funktion*. Öffnet eine Verbindung zu SQL Server mit dem OLEDB Provider SQLOLEDB. Im Erfolgsfall wird `TRUE` zurückgegeben.
- `open_shape()`. *Funktion*. Öffnet eine Verbindung zu SQL Server mit dem OLEDB Provider MSDataShape. Im Erfolgsfall wird `TRUE` zurückgegeben.
- `echo`. Eine Prozedur, die Informationen zum Browser sendet. Die übergebene Zeichenkette wird nach Variablennamen durchsucht, die mit $ anfangen. Anschließend werden die Namen dann durch die Werte

ersetzt. Dies kennen Sie vielleicht von Perl oder PHP. Diese Prozedur spart Ihnen eine Menge Schreibarbeit.

- `show_table`. Diese Prozedur gibt ein `RecordSet`-Objekt vollständig als HTML-Tabelle aus. Feldnamen werden als Überschrift der Spalten verwendet. Dies dient vor allem zum Debuggen von Skripten.
- `show_form`. Diese Prozedur liest ein `RecordSet`-Objekt und erzeugt ein zu den Feldern passendes Formular. Außerdem wird der Wert der Felder aus gleichnamigen Formularfeldern gelesen. Damit lassen sich einfach Skripte debuggen, die Daten schreiben.
- `form_to_array`. Diese Prozedur füllt ein Array mit den Werten aus einem Formular, das mit `show_form` erzeugt wurde. Dieses Array kann mit `AddNew` verwendet werden.
- `open_query`. Eine Prozedur, die eine SQL-Anweisung ausführt und im Fehlerfall detaillierte Fehlerinformationen anzeigt.
- `encode()`. Eine Funktion, die HTML-Code dekodiert und Zeilenumbrüche in <br> umwandelt.

1.4.2 Listing

Eine Bibliothek zum bequemen Umgang mit den Skripten

Im Folgenden finden Sie das Listing der Bibliothek. Damit lassen sich einige Skripte besser nachvollziehen.

```
<% on error resume next %>
<!-- For ADO 2.6  -->
<!-- #include file="adovbs.inc" -->
<!-- For ADOX 2.5 -->
<!-- METADATA TYPE="TypeLib"
            NAME="Microsoft ADO Ext. 2.5 for DDL and Security"
            UUID="{00000600-0000-0010-8000-00AA006D2EA4}"
            Version="2.5" -->
<% on error goto 0 %>
<%
' Globale Variablen
dim objConn
dim objRS
dim ASP_SELF

ASP_SELF = Request.Servervariables("SCRIPT_NAME")

' Funktion öffnet Verbindung zu DB
function open()
   dim strProvider, strDataSrc, strCatalog, strUser
   dim strPassword, strConnection
   dim strQuery
```

```
   ' Passen Sie diese Daten an, wenn Sie eine andere
   ' Umgebung haben
   strProvider = "SQLOLEDB"  ' Provider
   strDataSrc  = "WWW"       ' Name des Servers oder IP-Nummer
   strCatalog  = "Northwind" ' Name der Datenbank
   strUser     = "sa"        ' Nutzername des SQL Servers
   strPassword = ""          ' Kennwort
   ' Ende anpassbarer Bereich
   strConnection = "Provider=" & strProvider
                   & "; Data Source= " & strDataSrc
                   & "; Initital Catalog=" & strCatalog
                   & "; User Id=" & strUser
                   & "; Password=" & strPassword
   on error resume next
   Set objConn = Server.CreateObject("ADODB.Connection")
   objConn.Open strConnection
   if Err.Number > 0 then
      with Response
         .write "<div style=""color:red""><b>Fehler:</b> "
         .write Err.Description
         .write "</div>"
      end with
      open = FALSE
   else
      open = TRUE
   end if
   on error goto 0
   strQuery = "USE " & strCatalog
   objConn.Execute strQuery
end function

' Funktion oeffnet eine Data Shape-Verbindung
function open_shape()
   dim strProvider, strDataSrc, strCatalog, strUser
   dim strPassword, strConnection
   dim strQuery
   ' Passen Sie diese Daten an, wenn Sie eine andere
   ' Umgebung haben
   strProvider = "SQLOLEDB"  ' Provider
   strDataSrc  = "WWW"       ' Name des Servers oder IP-Nummer
   strCatalog  = "Northwind" ' Name der Datenbank mit
   strUser     = "sa"        ' Nutzername des SQL Servers
   strPassword = ""          ' Kennwort
   ' Ende anpassbarer Bereich
   strConnection = "Data Provider=" & strProvider
```

```
                    & "; Data Source= " & strDataSrc
                    & "; Initital Catalog=" & strCatalog
                    & "; User Id=" & strUser
                    & "; Password=" & strPassword
   on error resume next
   Set objConn = Server.CreateObject("ADODB.Connection")
   objConn.Provider = "MSDataShape"
   objConn.Open strConnection
   if Err.Number > 0 then
      with Response
         .write "<div style=""color:red""><b>Fehler:</b> "
         .write Err.Description
         .write "</div>"
      end with
      open_shape = FALSE
   else
      open_shape = TRUE
   end if
   on error goto 0
   strQuery = "USE " & strCatalog
   objConn.Execute strQuery
end function

' Prozedur zur einfachen Ausgabe von Zeichenketten
' Ersetzt mit $-gekennzeichnete Variablen durch ihren Wert
sub echo(message)
   dim pattern, myRegex, repla
   if len(Message) > 0 then
      message = replace(message, chr(34),
                        chr(34) & " & chr(34) & " & chr(34))
      pattern = "(\$)(\w+)"
      set myRegex = new RegExp
      myRegex.Pattern = pattern
      myRegex.IgnoreCase = TRUE
      myRegex.Global = TRUE
      repla = """
            & myRegex.Replace(message, """ & $2 & """) & """
      Response.write eval(repla)
   end if
end sub

' Zeigt ein RS-Objekt komplett als Tabelle an
sub show_table(objRecordSet)
   dim fname
   echo "<table border=0 cellpadding=1>"
```

```
   echo "<tr bgcolor=#eeeeee>"
   if objRecordSet.State = 1 then
      for each fname in objRecordSet.Fields
         echo "<th>" & fname.name & "</th>"
      next
      echo "</tr>"
      while not objRecordSet.EOF
         echo "<tr>"
         for each fname in objRecordSet.Fields
            echo "<td valign=top nowrap>"
                  & fname.value & "</td>"
         next
        echo "</tr>"
        objRecordSet.MoveNext
      wend
      echo "</table>"
   end if
end sub

' Zeigt ein Formular an, dessen Felder den Feldern eines Datensatzes
entsprechen
sub show_form(objRecordSet)
   dim fname
   const suffix = "_RSF"
   if objRecordSet.State = 1 then
      echo "<table border=0 cellpadding=1>"
      for each fname in objRecordSet.Fields
        echo "<tr>"
        echo "<td>" & fname.Name & "</td>"
        echo "<td>"
        echo "<input type=text "
        echo " size=" & cint(fname.DefinedSize)
        echo " maxlength=" & fname.DefinedSize
        echo " name=""" & fname.Name & suffix & """
        echo " value=""" & Request.Form(fname.Name) & """
        echo ">"
        echo "</td>"
        echo "</tr>"
      next
      echo "</tr>"
      echo "</table>"
   end if
end sub

' Zeigt eine Liste von Feldnamen aus einem RecordSet-Objekt an
```

```
sub show_headers(RS)
   dim f
   for each f in RS.Fields
      echo f.name & ", "
   next
   echo "<p>"
end sub

' Liest Formularfelder, die mit der Prozedur show_form()
' erzeugt wurden, in ein Array ein, das mit AddNew
' verarbeitet werden kann
sub form_to_array(byref arrFields, byref arrValues)
   dim i
   const suffix = "_RSF"
   i = 0
   for each formfield in Request.Form
    if instr(formfield, suffix) > 0 then
     redim preserve arrFields(i)
     redim preserve arrValues(i)
     arrFields(i) = left(formfield,
                         len(formfield) - len(suffix))
     arrValues(i) = Request.Form(formfield)
     i = i + 1
    end if
   next
end sub

' Führt eine SQL-Abfrage aus und ändert das Datnsatzobjekt
' im Original (ByRef)
sub open_query(byref objRecordSet, strQuery, objConnection,
intLockType, intCursorType, intOptions)
   dim errorfield
   on error resume next
   objRecordSet.Open strQuery, objConnection,
                     intLockType, intCursorType, intOptions
   if Err.Number <> 0 then
      for each errorfield in objConnection.Errors
         echo "<b>Fehler</b>: " & errorfield.Description
               & " (" & hex(errorfield.Number) & ")<br>"
        echo "<b>Quelle</b>: " & errorfield.Source & "<br>"
        echo "<b>String</b>: <code>" & strQuery & "</code><p>"
      next
   end if
   on error goto 0
end sub
```

```
' Stellt HTML-Code direkt dar, Zeilenumbrüche werden zu <br>
function encode(html)
   dim i, c, dc
   for i = 1 to len(html)
      c = mid(html, i, 1)
      if c = chr(13) then
        dc = dc & "<br>"
      else
        dc = dc & Server.HTMLEncode(c)
      end if
   next
   encode = dc
end function

%>
```

Listing 1.1: open.inc.asp: Die Basisbibliothek für die Skripte im Buch

1.5 Die Website zum Buch

Alle Skripte finden Sie im Internet zum Download!

Alle Skripte im Buch und weiterführende Informationen finden Sie auf der Website zum Buch unter der folgenden Adresse:

- *http://www.asp.comzept.de/dotnet*

Kennzeichnung der Skripte

Alle Skripte im Buch, die Sie aus dem Web herunterladen können, sind unterstrichen: open.inc.asp. Unter diesem Namen finden Sie Skripte auch wieder, sortiert nach Kapiteln.

Vielleicht ist auch die Website zum »klassischen« ASP für Sie interessant:

- *http://www.asp.comzept.de*

Alle Skripte, die im Buch abgedruckt sind, können Sie über die Website anhand des Namens und der Listing-Nummer finden. Zur besseren Kennzeichnung sind Namen im Buch unterstrichen. Die Sortierung auf der Website erfolgt nach Kapiteln. Zum Austausch mit anderen Lesern finden Sie dort auch ein Forum und eine Mailingliste.

2 Grundlagen ADO 2.6

Dieses Buch widmet sich in wesentlichen Teilen ADO 2.6 und den Unterschieden und Neuigkeiten in ADO.NET. Dabei stehen der Einsatz mit Visual Basic.NET und die Anwendung zum Datenbankzugriff unter ASP.NET im Vordergrund.

2.1 Struktur und Aufbau

Dieser Abschnitt klärt, was ADO eigentlich ist, wie es entstand und welchen Platz es in der Microsoft-Welt einnimmt.

2.1.1 Was ist ADO?

Active Data Objects

ADO steht für »Active Data Objects«. Oft wird auch die ausgeschriebene Version als »ActiveX Data Objects« bezeichnet, was grundsätzlich nicht falsch ist. Tatsächlich ist ADO ein Satz von ActiveX-Steuerelementen, die einen einfachen programmiertechnischen Zugriff auf die elementare Datenbankzugriffsebene bieten.

OLEDB

Diese Ebene wird OLEDB genannt. Sie ist als Satz von Schnittstellen ausgeführt, der zu jeder Datenquelle einen speziellen Treiber (den so genannten Provider) bereitstellt und die Übertragung der Daten auf einem einheitlichen Weg ermöglicht. OLEDB basiert direkt auf dem Windows-API (*Application Programming Interface*), das für Sprachen wie C++ entwickelt wurde. ADO setzt diese Schnittstelle auf das Niveau von ActiveX/COM-Objekten um, damit sie einem größeren Spektrum von Sprachen zur Verfügung stehen – auch Skriptsprachen und damit auch der Webserverprogrammierung.

Der Unterschied zwischen ActiveX und COM mag Skriptprogrammierern nicht geläufig sein, oft werden die Begriffe auch miteinander verwechselt. ActiveX ist eine plattformübergreifende Technologie für Komponenten, dagegen ist COM auf die Windows-Plattform beschränkt. Beide bauen jedoch auf der COM-Architektur auf, welche die grundlegenden Techniken liefert.

Eine ähnliche Verwechslungsgefahr besteht bei ODBC und OLEDB. ODBC ist eine universelle Schnittstelle für den Datenbankzugriff. ODBC ist älter und weniger universell als OLEDB und kann durch dieses vollständig ersetzt werden. Auf ODBC gehe ich deshalb nicht mehr ein.

ADO bietet damit einen standardisierten Weg, jede Art von Daten zu behandeln und Datenbankmanagementsysteme (DBMS) aller Art zu steuern. Auch wenn die hier gezeigten Beispiele auf den SQL Server 2000 zugeschnitten sind, heißt dies nicht, dass ADO darauf beschränkt wäre. So ist Access 2000 genauso eine geeignete Datenquelle wie Oracle oder sogar MySQL. Allerdings kann die Schnittstelle nur teilweise fehlende Funktionen in den DBMS ersetzen. Um alle Möglichkeiten zu zeigen und zu nutzen, ist ein vollwertiges DBMS zu nutzen – und dies ist in der Microsoft-Welt nur der SQL Server.

DNA und die Anwendung von ADO im Zusammenhang mit ASP

Im Zusammenhang mit der Anwendungsentwicklung im Internet wird der Begriff DNA (*Distributed interNet Applications*) fallen. ADO ist die wichtigste Technologie bei der Entwicklung datenbankgestützter Webseiten. Der Einsatz wird ausschließlich zusammen mit ASP oder ASP.NET erfolgen. ASP dient hier aber nur als »Mittel zum Zweck«, entsprechende Kenntnisse werden also beim Leser vorausgesetzt. Informationen über ADO, die in ASP keine Rolle spielen, wie z. B. Ereignisse, werden nicht betrachtet.

Die Struktur des Teils I – *ADO* spiegelt die Struktur der Objekte wider. Ausgehend vom Objektmodell werden in Kapitel 3 alle ADO-Objekte vorgestellt:

- `Connection` – das Objekt zur Herstellung und Kontrolle einer Verbindung
- `RecordSet`, `Record` – zwei Objekte zur Behandlung von Daten
- `Command` – das Objekt zur Steuerung von Abfragen
- `Field`, `Property` – Objekte für den Zugriff auf Datenstrukturen
- `Error` – zur Fehlerbehandlung

Entwicklung von ADO

Geschichte

ADO wurde mit der Version 1.5 eingeführt. Die Vorgängerversion 1.0 kam kaum zum Einsatz. ADO 1.5 war Bestandteil des Option Packs von Windows NT 4, von Visual InterDev und vieler Programmiersprachen von Microsoft mit der Versionsnummer 5.

ADO 2.0 wurde mit Visual Studio 6.0 eingeführt und damit mit den Programmiersprachen der Versionsnummer 6. Mit Office 2000 und den ersten Betaversionen von Windows 2000 erfolgte die Einführung von Version 2.1.

Mit dem Erscheinen von Windows 2000 wurde ADO erneut überarbeitet, es steht nun mit der Version 2.6 zur Verfügung. ADO selbst kann auch direkt von folgender Adresse bezogen werden:

- *http://www.microsoft.com/data*

2.1.2 Architektur

Es ist interessant zu sehen, wie sich OLEDB und ADO in die Gesamtarchitektur einer Applikationsumgebung einfügen. So wird auch klar, dass ADO vor allem für den Skriptprogrammierer als »Vereinfachungsschicht« dient – ein offensichtlich erfolgreicher Weg.

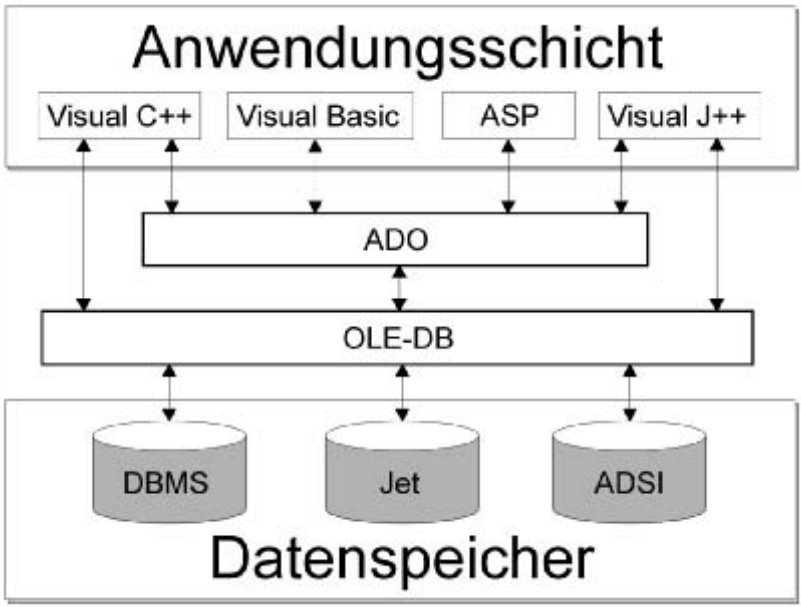


Abbildung 2.1: So ordnet sich ADO zwischen OLEDB und der Applikationsschicht ein

OLEDB ist für kritische Applikationen, die in C++ geschrieben werden, eher geeignet. Dagegen sind die ohnehin weniger effizienten Programme in VBScript deutlich schneller und einfacher zu entwickeln, wenn ADO zum Einsatz kommt. Mit OLEDB werden Sie nicht in Berührung kommen – abgesehen von der Angabe des passenden Datenproviders.

OLEDB-Provider und -Consumer

Für die erfolgreiche Nutzung der ADO-Funktionen ist eine kurze Bezugnahme auf OLEDB notwendig. Diese Schicht stellt die Verbindung zu den Datenquellen her. Microsoft führte dafür auf Grund der Vielfalt von Möglichkeiten den Begriff »Provider« ein. Entsprechend wird der Empfänger der Daten als »Consumer« bezeichnet, dieses Wort ist in diesem Zusammenhang aber seltener anzutreffen. Die Idee dahinter ist nicht neu – sie wurde bereits mit ODBC eingeführt. Im Idealfall können Sie so programmieren, dass ein Austausch des Providers keine Änderungen am Code erfordert. Das setzt aber voraus, dass keine Befehle verwendet werden, die von speziellen Merkmalen eines Providers abhängen. Unter Umständen lohnt die Anschaffung eines teuren DBMS nur dann, wenn Sie die besonderen Funktionen auch nutzen können. Damit ist die Universalität aber nicht mehr vollkommen gegeben. Trotzdem erleichtert OLEDB den Zugriff auf die Datenquelle und es kommt auch dann zum Einsatz, wenn ein Wechsel des DBMS nicht vorgesehen ist.

Standardprovider

Microsoft liefert einige Standardprovider, die Sie kennen sollten:

- *Jet OLEDB 4.0* – der Provider für Microsoft Access
- *Indexing Service* – zum Zugriff auf den Index Server

- *ODBC* – erlaubt den Zugriff auf ODBC-Quellen
- *SQL Server* – für den SQL Server
- *MS Data Shape* – für hierarchische Datenstrukturen
- *Microsoft Directory Services* – für die neuen Verzeichnisdienste des Active Directory

Dies ist nur eine kleine Auswahl. Andere Provider, wie Oracle, verlangen nach einem installierten Client des entsprechenden Anbieters.

Universal Data Access

UDA

Im Zusammenhang mit OLEDB ist bei Microsoft auch von UDA (*Universal Data Access*) die Rede. Dahinter steckt die Idee, jede beliebige Datenquelle ansprechen zu können. Das ist nichts völlig Neues – eher eine Marketingidee als ein Technologiemodell. Grundsätzlich wird ein ähnlicher Ansatz verfolgt wie früher mit ODBC. Nur dehnt UDA dies auf alle Datenquellen aus – nicht nur auf solche aus relationalen Datenbanken. Andere Datenquellen können Textdateien, Verzeichnisdienste, XML-Dateien sein. Realisiert wird dies durch einen passenden Provider, der der OLEDB-Schnittstelle die Daten in einer definierten Form zur Verfügung stellt – nichts Neues also.

Eigene Provider

Ginge es um wirklich universellen Zugriff, wäre die Verfügbarkeit geeigneter Provider tatsächlich ein wesentliches Merkmal. Nun kann aber nicht erwartet werden, dass für jede erdenkliche Datenquelle schon Provider existieren. Es ist deshalb vergleichsweise einfach, eigene Provider zu schreiben. Dazu eignet sich neben Visual C++ sogar Visual Basic.

2.1.3 Existierende Technologien

Ein paar Begriffe wurden schon eingeführt, vor allem solche, die miteinander in Zusammenhang stehen. Wenn Sie sich zusätzliche Literatur zum Thema ADO beschaffen, finden Sie weitere Abkürzungen (vorzugsweise DBAs – *Drei Buchstaben Abkürzungen*). Weil bei Microsoft manches anders ist als anderswo in der Programmierwelt, gibt es aber auch längere Abkürzungen. Die folgenden Begriffe sollten Sie in diesem Zusammenhang auch kennen:

Wichtige Technologien im Zusammenhang mit ADO

- *DBLib*: Dies ist eine Bibliothek, auf der der SQL Server aufbaut und die eine Programmierschnittstelle für C-Programme zur Verfügung stellt.
- *ODBC*: Diese Abkürzung steht für Open Database Connectivity, der erste Schritt auf dem langen Weg zu UDA. ODBC ist eine datenbankunabhängige Methode des Zugriffs auf Daten, die von Microsoft entwickelt wurde und sich inzwischen als herstellerunabhängiger Standard etabliert hat.
- *DAO*: Die »Data Access Objects« bilden das Datenzugriffsmodell, das mit Access und anderen ISAM-Datenbanken eingeführt wurde und auch für Visual Basic verfügbar ist. DAO setzt auf ODBC auf und kann

universell verwendet werden, ist dann aber relativ langsam. Nur der Zugriff auf Jet-Datenbanken ist optimiert (dazu gehört praktisch nur Access).

- *RDO*: Die »Remote Data Objects« sind der Nachfolger von DAO und erlauben zusätzlich den Zugriff auf entfernte Datenquellen, z.B. über das Internet. Die Nachteile von DAO wurden nicht beseitigt.
- *ODBCDirect*: Diese Version kombiniert DAO und RDO und erlaubt den Zugriff auf alle ODBC-Quellen, ohne auf Jet-Datenbanken angewiesen zu sein.
- *JDBC*: »Java Database Connectivity« ist eine herstellerunabhängige Programmierschnittstelle für Java-Applikationen.

Der Weg zu OLEDB

Alle diese Technologien verursachen in der Praxis Probleme. So ist die Programmierung mit der DBLib oder ODBC (auf Programmierebene) für den Datenbankzugriff sehr kompliziert. Skriptsprachen bieten erst gar keine Möglichkeit der Nutzung. DAO und RDO dagegen bauen auf ODBC auf und legen damit eine weitere Schicht zwischen Datenbank und Programmierschnittstelle. Dies funktioniert zwar nun auch in Skriptsprachen, ist aber sehr langsam. Alle Schnittstellen haben ein streng hierarchisches Modell und damit einen großen Overhead. Es gab also gute Gründe, diese verschiedenen Versionen zu vereinen und dabei die Vorteile zu erhalten, die Nachteile aber zu reduzieren – dies genau ist mit OLEDB gelungen.

2.1.4 OLEDB

OLEDB wurde bereits mehrfach als Nachfolger von ODBC genannt. Die Gründe, die im letzten Abschnitt angeführt wurden, mögen zwar einleuchtend sein, aber warum auf ODBC verzichten? Es ist in Windows und anderen Systemen verfügbar, es funktioniert und es ist relativ einfach. Dass Microsoft die Ablösung wünscht, wird durch bestimmte Merkmale deutlich. So wurden in Windows 2000 alle administrativen Systemfunktionen in die Management Konsole überführt – mit Ausnahme von ODBC. Das wäre natürlich für den Softwareentwickler kein Grund, auf ODBC zu verzichten.

Flexible Datenspeicherung

Ein Grund ist die Form der Datenspeicherung. Heute werden Daten nicht mehr ausschließlich in relationalen Datenbanken gehalten, sondern auch in XML-Dateien, in Verzeichnisdiensten oder hierarchischen Dateisystemen. OLEDB kann auf alle diese Quellen zugreifen.

COM-basiert

Außerdem setzt OLEDB konsequent auf COM (*Component Object Model*). COM bietet eine universelle Basis für die Anwendungsentwicklung. Es genügt, eine Applikation für COM zu schreiben, sie kann dann überall in der Windows-Welt verwendet werden. Mit DCOM (Distributed COM) dehnt sich die Applikation dann auch über das Netzwerk aus.

Um den Umstieg zu erleichtern, existieren auch zwischen ODBC und OLEDB Beziehungen. Denn es gibt heute noch mehr ODBC-Treiber als OLEDB-Provider. ODBC wird deshalb als weiterer Provider angeboten

(siehe Abbildung 2.2). Wann immer es einen nativen Provider gibt, sollten Sie natürlich ODBC umgehen – es kann nur langsamer werden, – denn es wird eine weitere Schicht zwischengeschoben.

3.1 *Connection* ab Seite 47

Wie Sie OLEDB in ADO verwenden, wird im Abschnitt über das Verbindungsobjekt `Connection` erläutert – denn nur dort werden Sie damit konfrontiert.

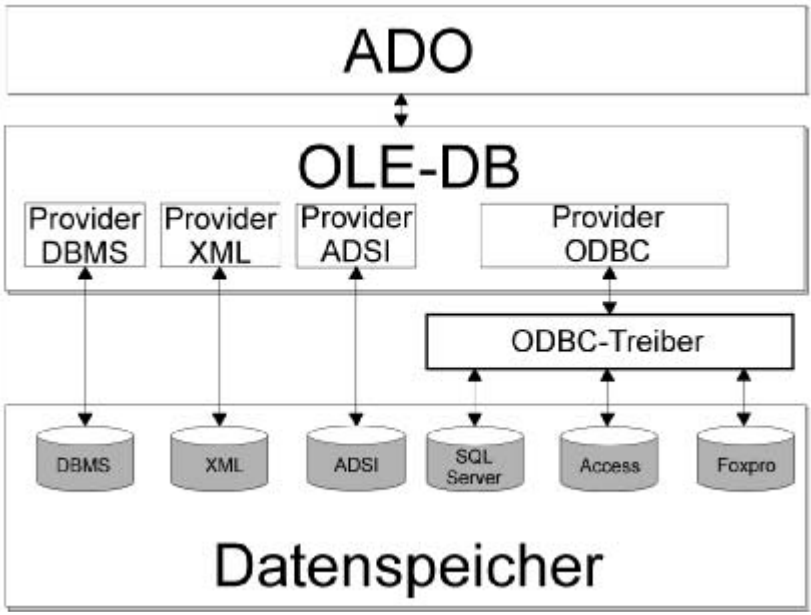


Abbildung 2.2: So ordnen sich ODBC und das ODBC-Treibermodell in die OLEDB-Welt ein

2.1.5 ADO-Funktionen

Unterstützung und Einschränkungen in ASP

ADO 2.6 bietet verschiedene grundlegende Eigenschaften. Nicht alle stehen für den ASP-Programmierer zur Verfügung. Dies müssen Sie bei der Anwendungsentwicklung berücksichtigen und – vor allem – beim Lesen der Dokumentation. Denn dort wird selten auf die speziellen sprachlichen Belange Rücksicht genommen.

- *Ereignisse*: ADO unterstützt Ereignisse, dabei werden bestimmte Funktionen beim Eintreten eines Zustands ausgelöst. Das funktioniert mit ASP nicht.
- *Visual C++*- und *Visual J++*-Erweiterungen: Diese dienen der direkten Unterstützung von ADO in Visual C++-Programmen und für Java.
- *Hierarchische Zeiger* und *Data Shaping*: Diese spezielle Eigenschaft erlaubt die Darstellungen hierarchischer Daten. Das ist notwendig, um auch andere Datenquellen außer relationalen Datenbank abbilden zu können, z.B. ein Verzeichnissystem. Diese Funktionen sind in ASP verfügbar.

- *Kundenspezifische Modifikation der Datenschnittstellen:* Damit können Sicherheitsprobleme beseitigt werden, die ADO früher hatte.
- *Erzeugen von Datensätzen*: ADO kann nun Datensätze selbst, ohne Zugriff auf die Datenbank, erstellen. Für die Übertragung von Daten zwischen den Schichten einer Mehrschichtapplikation kann so auf die Speicherung in der Datenbank verzichtet werden.
- *Persistente Datensätze*: Dadurch wird die Speicherung von Datensätzen in einer Datei möglich. Damit können Daten später weiter verarbeitet werden, auch wenn die Verbindung zur Datenbank bereits verloren ging. Außerdem können komplette Datensätze in HTML, XML oder andere Formate überführt werden.
- *Indizierung und Filterung*. ADO führt Indizierungs- und Filtervorgänge in Datensätzen ohne Zugriff auf die Datenbank aus. Damit kann teilweise auf komplexe SQL-Anweisungen verzichtet werden.
- *Unterstützung des Visual Studio Analyzer*. Der Visual Studio Analyzer hilft bei der Leistungsanalyse von Programmen. Es werden nun auch Optimierungen in ADO-basierten Programmen durchgeführt.
- *Konfliktauflösung für Client-Zeiger*. Wenn Datensätze komplett zum Client übertragen und dort verändert werden, zugleich aber Zugriffe von anderen Clients erfolgen, kann es beim Rückschreiben der Daten Konflikte geben. ADO löst diese nun mit speziellen Synchronisationsmethoden auf.
- *Neues Sicherheitsmodell*: Bei der Verwendung von ADO in Clientapplikationen können Anwendungen für ein im Internet Explorer entworfenes Sicherheitsmodell erstellt werden. Auf dem Server (und damit für ASP) hat das keine Bedeutung.

Anfänger fragen sich oft, ob die Unterstützung von Ereignissen mit neuen Versionen auch in ASP möglich sein wird. Die Antwort ist in der Arbeitsweise des Internets und des bestimmenden Protokolls HTTP zu finden. Aktionen im Browser verlangen eine Übertragung der Daten zum Server, also das Auslösen eines Links oder das Absenden eines Formulars. Dies ist kein Ereignis im Sinne der ADO-Spezifikation. Andere Verbindungen gibt es nicht – abgesehen von den ActiveX-Erweiterungen und RDO, die in der Praxis schon früher gescheitert sind. Sie müssen also, wenn Sie ADO kennen und nun ASP programmieren, einige Überlegungen anstellen, um die Ausführung der Software den Umgebungsbedingungen anzupassen. Es geht – fast immer – auch ohne Ereignisse.

2.1.6 Übersicht der ADO-Objekte

ADO 2.6 selbst hat auch ein reichhaltiges Objektmodell. Es gibt fünf Objekte und mehrere Kollektionen; die Objekte stehen dabei miteinander in einer direkten Beziehung. Kollektionen sind Sammlungen von Daten oder weitere Objekten, die über Schlüssel angesprochen werden können, ähnlich einem Index.

Objekte

Der elementare Zugriff auf diese Objekte erfolgt in ASP mit folgendem Code:

```
Set objMyConnection = CreateObject("ADODB.Object")
```

Dabei steht *Object* für eines der folgenden Objekte:

3.1 Connection ab Seite 47

- `Connection`. Dieses Objekt dient zum Verbinden mit einer Datenquelle. Hiermit wird der passende OLEDB-Provider angegeben. Es gibt aber auch ein implizites `Connection`-Objekt, das erstellt wird, wenn Sie ein `RecordSet`-, `Record`- oder `Command`-Objekt nutzen, ohne auf eine bestehende Verbindung zuzugreifen.

3.4 Command ab Seite 122

- `Command`. Dieses Objekt dient der Anwendung von Befehlen auf einen Datenspeicher. Eine häufige Nutzung ist der Start einer gespeicherten Prozedur im SQL Server. Entsteht bei einem solchen Befehl ein `RecordSet`- oder `Record`-Objekt, wird dieses implizit erzeugt. Öfter werden jedoch Befehle abgesetzt, die keine Daten zurückgeben – dann gibt es keine Alternative zum `Command`-Objekt.

3.2 RecordSet ab Seite 69

- `RecordSet`. Daten aus einer Datenquelle lesen – das ist der Hauptanwendungsfall für das `RecordSet`-Objekt. Wenn Sie wissen, dass Daten zurückgegeben werden, sollten Sie das Objekt explizit erzeugen. Die Darstellung der einzelnen Zeilen eines `RecordSet`-Objekts erfolgen durch das `Record`-Objekt.

3.3 Record ab Seite 113

- `Record`. Zum Zugriff auf eine einzelne Datenzeile dient das `Record`-Objekt. Dieses Objekt ist neu seit ADO 2.5. Es wird für Datenquellen verwendet, die in jeder Zeile eine andere Struktur der Daten haben, z. B. ein Verzeichnissystem oder ein E-Mail-Ordner.

3.7 Stream ab Seite 141

- `Stream`. Dieses Objekt ist ebenfalls neu seit ADO 2.5 und ergänzt das `Record`-Objekt um die Fähigkeit, mit unstrukturierten Daten umgehen zu können. Eine Anwendung ist der mögliche Zugriff auf die Daten eines E-Mail-Servers – damit können Sie möglicherweise auf CDO verzichten.

Kollektionen

Neben den Objekten gibt es noch Kollektionen. Auf Kollektionen greifen Sie zu, indem Sie die Elemente sukzessive durchlaufen, wie es folgender ASP-Code andeutet:

```
For Each objElement In colCollection
   ' … Aktion mit objElement ausführen
Next
```

Eine andere Variante besteht im Zugriff auf die Anzahl der Elemente der Kollektion:

```
For i = 0 To colCollection.Count
   ' … Aktion mit objElement ausführen
Next
```

Jede Kollektion besitzt eine Eigenschaft `Count`, die die Anzahl der Elemente repräsentiert.

ADO 2.6 kennt folgende Kollektionen:

- `Fields`. Diese Kollektion enthält Informationen über die Felder eines `RecordSet`- oder `Record`-Objekts. **4.1 Fields ab Seite 159**
- `Properties`. Mit dieser Kollektion werden Eigenschaften der `Connection`-, `RecordSet`, `Record`- und `Command`-Objekte verwaltet, die dort nicht schon statisch definiert wurden. Das sind in der Regel Eigenschaften, die auf einen spezifischen Provider zugeschnitten sind. **4.2 Properties ab Seite 164**
- `Parameters`. Diese Kollektion findet ausschließlich zusammen mit dem `Command`-Objekt Anwendung und enthält die Parameter, die zusammen mit dem Kommando übergeben werden. Da die Anzahl der Parameter variabel ist, wird eine Kollektion verwendet. **4.4 Parameters ab Seite 167**
- `Errors`. Diese Kollektion enthält Angaben über die Fehler, die beim Verarbeiten von Daten entstehen. **3.8 Error ab Seite 153**

2.1.7 Konstanten

ADO verfügt über eine Datei mit Konstanten, ADOVBS.INC (für VBScript) bzw. ADOJAVAS.INC (für JScript). Sie finden diese Datei im folgenden Verzeichnis:

```
C:\Programme\Gemeinsame Dateien\System\ado
```

Konstanten einbinden

Sie müssen diese Konstanten in jedem Skript zur Verfügung stellen. Dazu wird die SSI-Anweisung `#INCLUDE` verwendet: **adovbs.inc**

```
<!-- #INCLUDE FILE="adovbs.inc" -->
```

Das kann ausgesprochen lästig sein, wenn Sie Dutzende von Dateien erstellen müssen. In diesem Fall sollten Sie die folgende Zeile in die Datei GLOBAL.ASA schreiben: **»typelib«**

```
<!-- METADATA TYPE="typelib" FILE="C:\Programme\Gemeinsame
                           Dateien\System\ado\msado15.dll" -->
```

Lassen Sie sich nicht durch den Dateinamen MSADO15 irritieren, das ist die Version 2.6. Sie können diesen Zugriff natürlich auch im ASP-Skript selbst platzieren.

2.2 Datenzugriff

Jedes Experiment mit ADO beginnt mit einer Verbindung zur Datenquelle. Über die Einrichtung der Datenquelle und die entsprechenden Verbindungszeichenfolgen sollten Sie also vorher Bescheid wissen. Abschnitt 3.1 Connection ab Seite 47 zeigt den Umgang mit dem `Connection`-Objekt, das die hier beschriebenen Verbindungszeichenfolgen akzeptiert.

2.2.1 Verbindungen

Verbindung zur Datenquelle

Unabhängig davon, ob Sie explizit (mit `Connection`) oder implizit (mit allen anderen Objekten) eine Verbindung aufbauen, gelten die nachfolgend beschriebenen Prinzipien.

- *Verbindungszeichenfolge*. Sie geben eine Zeichenkette an, die alle nötigen Angaben über den zu verwendenden Provider enthält.
- *Datenlinkdatei*. Diese Datei mit der Endung UDL enthält die zur Verbindungsaufnahme nötigen Daten.
- *DSN (Data Source Name)*. ODBC-Quellen werden typischerweise durch DSN-Dateien angesprochen. Sie können dies aber auch umgehen. Die Einrichtung der verschiedenen DSN-Arten erfolgt im ODBC-Manager.

Alle Varianten werden nachfolgend beschrieben, auch die für ODBC. Das Thema ODBC ist aber damit abgeschlossen.

2.2.2 Verbindungszeichenfolgen

Aufbau der Verbindungszeichenfolgen

Die enge Verwandtschaft zwischen OLEDB und ODBC spiegelt sich auch im Design der Verbindungszeichenfolgen wider. Der OLEDB-Provider wird durch den Schlüssel `Provider=` angesprochen; ohne diese Angabe wird implizit die DSN-lose Version von ODBC verwendet. Da oft beides funktioniert, werden Sie eine falsche Angabe nicht sofort bemerken.

Die folgende Aufstellung zeigt alle typischen Verbindungszeichenfolgen. Variable Angaben, die Sie entsprechend Ihren Bedingungen anpassen müssen, sind kursiv dargestellt.

OLEDB

Access

```
Provider=Microsoft.Jet.OLEDB.4.0; Data Source=database.mdb
```

- PROVIDER: bezeichnet den OLEDB-Provider

- DATA SOURCE: Name der Access-Datenbank

SQL Server

```
Provider=SQLOLEDB; Data Source=server; Database=datenbank;
UID=nutzername; PWD=kennwort
```

- PROVIDER: bezeichnet den OLEDB-Provider
- DATA SOURCE: Name des SQL Servers
- DATABASE: Name der Datenbank
- UID: Username
- PWD: Kennwort des Users

Index Server

```
Provider=MSIDXS; Data Source=katalogname
```

- PROVIDER: bezeichnet den OLEDB-Provider
- DATA SOURCE: Name des Index-Katalogs

ODBC

Access

```
Driver={Microsoft Access Treiber (*.mdb)}; DBQ=datenbank.mdb
```

- DRIVER: Name des Treibers
- DBQ: Name der Access-Datenbank

SQL Server

```
Driver={SQL Server}; Data Source=server; Database=datenbank;
UID=nutzername; PWD=kennwort
```

- DRIVER: bezeichnet den Treiber des SQL Servers
- DATA SOURCE: Name des SQL Servers
- DATABASE: Name der Datenbank
- UID: Username
- PWD: Kennwort des Users

Index Server

Einen ODBC-Treiber für den Index Server gibt es nicht. Hier können Sie nur die native OLEDB-Schnittstelle verwenden.

2.2.4 DSN ab Seite 42

Alternativ zur direkten Angabe eines ODBC-Treibers können Sie auch eine DSN einrichten. Das setzt aber voraus, dass Sie Zugriff auf die Systemsteuerung des Servercomputers haben.

2.2.3 Datenlinkdatei

Die UDL-Datei von Hand erstellen

Eine Datenlinkdatei erstellen Sie folgendermaßen:

- Legen Sie mit dem Editor eine Datei mit der Erweiterung UDL an.

- Schreiben Sie in die erste Zeile `[oledb]`.
- Schreiben Sie in die zweite Zeile die oben bereits gezeigte Verbindungszeichenfolge für OLEDB.

Wenn Sie die so erzeugte Datei mit einem Doppelklick öffnen, gelangen Sie in das Programm DATENVERKNÜPFUNGSEIGENSCHAFTEN. Hier können Sie bequem alle Parameter der Verbindungszeichenfolge einstellen. Leider existiert kein direkter Weg zu diesem Programm.

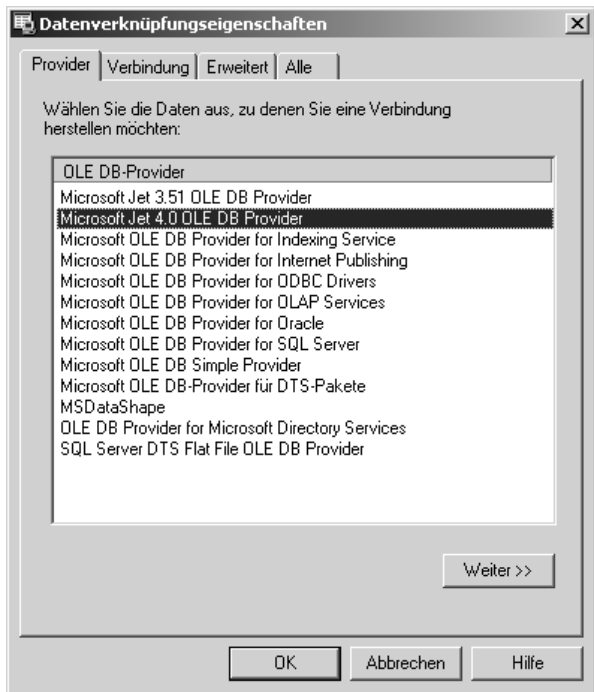
Um mit dieser Datei zu arbeiten, sieht die Verbindungszeichenfolge so aus:

```
File Name=dateiname.udl
```

Das Programm Datenverknüpfungseigenschaften

Auf der ersten Registerkarte PROVIDER wählen Sie die Datenquelle aus, mit der Sie arbeiten möchten. Alle weiteren Einstellungen hängen vom Provider ab. Normalerweise müssen Sie die Datenbankdatei, den Nutzernamen und ein Kennwort angeben.

Abbildung 2.3: Einrichtung einer Datenlinkdatei



2.2.4 DSN

Data Source Name

DSN steht für *Data Source Name*. Es gibt drei Arten von DSN, auf die zugegriffen werden kann:

- *User-DSN*. Nutzerspezifische Datenquellen. Wird nur verwendet, um »private« Datenquellen zu erzeugen. Auf diese Quelle kann nur der lokale Nutzer des Computers zugreifen; die Datenquelle muss auch auf diesem Computer laufen.
- *System-DSN*. Eine Datenquelle, die dem Computer zugeordnet ist. Auf diese Quellen können Personen zugreifen, die Zugriff auf den Computer haben. Auch laufende Dienste wie der Webserver können auf diese Datenquellen zugreifen.
- *File-DSN*. Eine nutzerspezifische Datenquelle, auf die mehrere Personen zugreifen können. Sie kann irgendwo im Netzwerk liegen und alle, die über gleichartige Treiber verfügen, können darauf zugreifen. Diese DSN speichert die Parameter in Textdateien.

Welcher DSN-Typ ist der richtige?

Wenn Sie nicht genau wissen, welche Variante die richtige ist, geben Sie eine System-DSN an. Diese Form ist am einfachsten zu verwalten, der Webserver erhält problemlos Zugriff. Die Einrichtung nehmen Sie in der Systemsteuerung vor.

Wählen Sie dazu das ODBC-Symbol und dann die Registerkarte SYSTEM DSN aus. Dann klicken Sie auf die Schaltfläche HINZUFÜGEN..., um eine neue Datenquelle anzulegen. Nach der Auswahl des Treibers sind alle weiteren Dialoge vom verwendeten DBMS abhängig. So beschränkt sich der Access-Treiber auf ein Dialogfenster, während der Treiber des SQL Servers einen Assistenten startet.

Angabe der DSN

Mit DSN arbeiten

Die DSN geben Sie in der nachfolgend beschriebenen Form an. Zuerst eine System- oder Benutzer-DSN:

```
DSN=Name
```

Bei einer Datei-DSN verweisen Sie auf eine Datei, welche die Angaben enthält:

```
FileDSN=Dateiname.dsn
```

2.3 Objektmodell

Jede Vorstellung von ADO wäre unvollständig, wenn das Objektmodell nicht gezeigt würde. Dieses Modell zeigt, wie die Objekte und Kollektionen zusammenhängen und welche Teile sich aus anderen ableiten lassen.

2.3.1 Darstellungsformen

Verschiedene Darstellungsformen

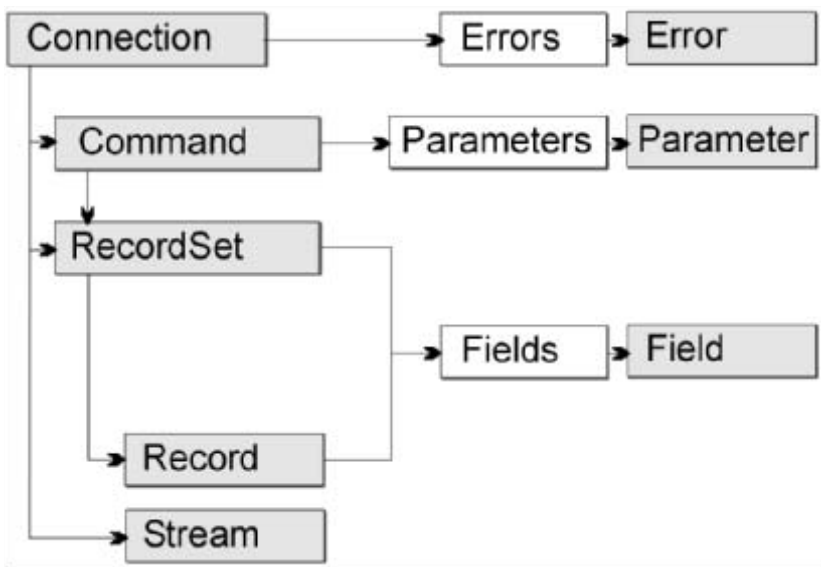
Die Darstellung des Objektmodells ist keinesfalls einheitlich. Die von Microsoft selbst propagierte Form leidet unter dem Handbuch-Syndrom – Voll-

ständigkeit wird hier zu Lasten der Lesbarkeit bevorzugt. In der Praxis können Sie damit wenig anfangen. Andere Modelle zeigen alle Abhängigkeiten durch vielfach verschachtelte Graphen an. Hier werden zwar auch seltener benötigte Zusammenhänge klar, die Auflösung ist aber nicht einfach und für einen schnellen Blick ist es denkbar ungeeignet. Ich habe mich hier für das gesplittete Modell entschieden. Direkte Zusammenhänge werden in einer einfachen Baumstruktur dargestellt und was nicht direkt hineinpasst, wandert in ein eigenes Diagramm. Am Ende stehen statt eines großen viele kleine Bäume. Das ist durchaus auch für Anfänger durchschaubar und eignet sich eher zum Nachschlagen.

2.3.2 Die Objektmodelle

Das Objektmodell enthält sowohl einfache Objekte als auch Kollektionen, die aus einer Sammlung von Objekten bestehen. Kollektionen sind in den folgenden Darstellung weiß hinterlegt, alle Objekte dagegen grau. In der ersten Darstellungen finden Sie alle direkt ansprechbaren Objekte und deren Abhängigkeiten.

Abbildung 2.4: Basismodell ADO



Wie ist dieses Modell zu lesen? Den Ausgangspunkt bildet ein Verbindungs-Objekt: `Connection`. Davon direkt abgeleitet wird die `Errors`-Kollektion, die Fehler enthält. Jeder einzelne Fehler wiederum wird in einem `Error`-Objekt gehalten. Auf dem Weg zu den Daten gibt es drei Möglichkeiten: `Command`, `RecordSet` und `Stream`. Von `RecordSet` können Sie `Record` ableiten, beide bestehen aus einer `Fields`-Kollektion, die wiederum `Field`-Objekte enthält. Sie können also z. B. `Field` nie direkt verwenden.

Fast alle Objekte können viele zusätzliche Eigenschaften enthalten. Diese werden in einer `Properties`-Kollektion gespeichert und heißen `Property`.

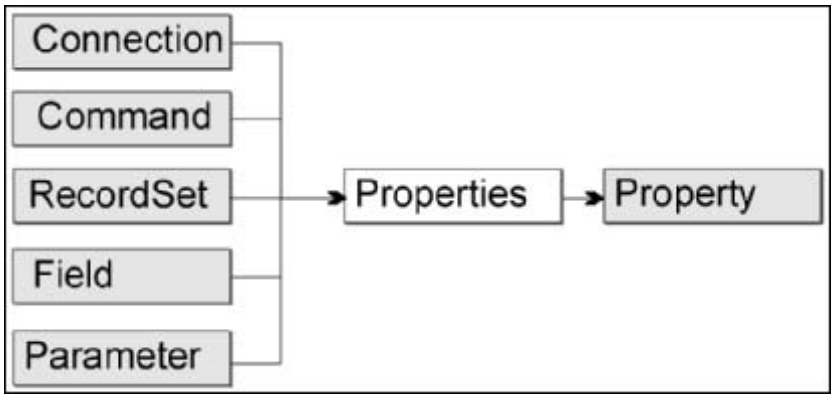


Abbildung 2.5: Die Properties-Kollektion im Objektmodell

Verwendungshinweise

Wenn Sie Skripte entwickeln, werden Sie sich intensiv mit den Methoden und Eigenschaften der Objekte auseinander setzen. Dabei fällt auf, dass oft die explizite Verwendung eines Objekts nicht notwendig ist. So können Sie auf `Connection` verzichten und die Verbindungszeichenfolge direkt an die `Open`-Methoden der untergeordneten Objekte übergeben. Das funktioniert – elegant ist es dennoch nicht.

Denn Sie verbergen die Struktur des Skripts, erschweren die Wartung und Pflege und werden früher oder später Probleme mit der verringerten Flexibilität bekommen. Es bringt auch keinen Leistungsgewinn – ADO wird das Verbindungsobjekt dann implizit anlegen und dennoch verwalten.

Das Datensatzobjekt lässt sich folgendermaßen verwenden:

Beispiel

```
strConn = "Provider=SQLOLEB; Initial Catalog=NorthWind"
set objRS = Server.CreateObject("ADODB.RecordSet")
objRS.Open "SELECT * FROM Customers", strConn
```

Diese Konstruktion öffnet ein `Connection`-Objekt implizit und verwendet dabei die Zeichenfolge *strConn*. Das ist unproblematisch, solange nur ein solches Objekt aktiv ist. Öffnen Sie aber mehrere Datensätze, werden diese auch mehrere implizite Verbindungsobjekte erzeugen und damit konkurrierende Verbindungen zur Datenbank aufmachen. Solche Verbindungen sind aber eine wertvolle Ressource. Es kann schnell zu Leistungseinbrüchen kommen. Wenn Sie dagegen ein Verbindungsobjekt erzeugen, können Sie es mit der damit eröffneten Verbindung immer wieder verwenden. Besser ist also folgende Schreibweise:

```
strConn = "Provider=SQLOLEB; Initial Catalog=NorthWind"
set objConn = Server.CreateObject("ADODB.Connection")
```

```
set objRS = Server.CreateObject("ADODB.RecordSet")
objRS.Open "SELECT * FROM Customers", objConn
```

Tipps für Anfänger

»Faulheit« ist also auch hier nicht angebracht, auch wenn einige Konstrukte dazu verlocken mögen. Versuchen Sie, sauber und übersichtlich zu programmieren. Gerade Anfänger sind schnell begeistert von ADO und verstricken sich dann auf dem Weg zu größeren Applikationen in ein Dickicht von Objekten und scheinbar »raffinierten« Codes. Tatsächlich aber verlieren sie früher oder später die Übersicht und geben dann entnervt auf. Ein paar Zeilen mehr sind deshalb immer der bessere Weg. Leistungsvorteile bringt enger Code nur sehr selten und wenn, dann nicht in einer signifikanten Größenordnung.

3 ADO 2.6 im Detail

Dieser Abschnitt beschreibt alle ADO-Objekte, deren Methoden und Eigenschaften *vollständig*. Sie benötigen keine zusätzlichen Informationen aus anderen Referenzen oder der Online-Hilfe.

3.1 Connection

Das `Connection`-Objekt erzeugen Sie mit dem folgenden Befehl:

```
Set objConnection = CreateObject("ADODB.Connection")
```

Die Objektvariable *objConnection* wird in allen folgenden Syntaxdiagrammen dieses Abschnitts vorausgesetzt.

3.1.1 Einführung

Das Objekt `Connection` (Verbindungsobjekt) verbindet den Consumer mit dem Provider. Einige Methoden der Objekte `Record`, `RecordSet` und `Command` können zwar diese Verbindung auch implizit öffnen, normalerweise wird aber der explizite Aufruf des Objekts genutzt:

3.2 RecordSet ab Seite 69
3.3 Record ab Seite 113
3.4 Command ab Seite 122

```
Set objConnection = CreateObject("ADODB.Connection")
```

Jetzt steht das `Connection`-Objekt in der Objektvariablen *objConnection* zur Verfügung. Üblicherweise ist der erste Befehl die Eröffnung der Verbindung:

Open ab Seite 54

```
objConnection.Open "Verbindungszeichenfolge", Parameter
```

Verbindungspooling

Erhalt der Verbindung

Konform mit der verbindungslosen Natur des HTTP-Protokolls werden am Ende eines ASP-Skripts alle Variablen gelöscht, alle Objekte zerstört. Jede neue Seite ist wieder völlig neu. Verbindungen zwischen Seiten, die den Nutzer wiedererkennen, müssen aufwändig programmiert werden oder nutzen versteckte Algorithmen, z. B. Cookies.

In anderen Umgebungen wird dieses Verfahren auch als Persistenz bezeichnet; man öffnet dann eine »persistente« Verbindung.

Der Pool

Für das Verbindungsobjekt heißt das, dass die Verbindung bei jedem Aufruf einer Seite immer wieder geschlossen und geöffnet wird. Leider ist dieser Vorgang zeitaufwändig. Das Verbindungspooling bietet dafür eine Lösung. Die OLEDB-Schicht speichert alle geöffneten Verbindungen. Greift ein weiterer Prozess mit identischen Daten darauf zu, wird die im »Pool« gespeicherte Verbindung genutzt. Verbindungen verbleiben dort eine gewisse Zeit, danach werden sie automatisch gelöscht. Änderungen an der Datenquelle spiegelt das Objekt im Pool natürlich nicht wider – Sie sollten Maßnahmen zum Abfangen von Fehlern vorsehen, wenn mit Verbindungspooling gearbeitet wird.

Verbindung wiedererkennen

Das Wiedererkennen der Verbindung erfolgt durch Vergleich der Parameter – nur wenn diese exakt übereinstimmen, wird das Objekt aus dem Pool verwendet. Das ist auch aus Sicherheitsgründen notwendig. Wechselt der Benutzer (bei sonst identischen Zugriffsinformationen), muss ein neues Verbindungsobjekt erstellt werden.

Zeitersparnis

Die tatsächliche Zeitersparnis ist nicht signifikant, sie liegt unter 10%. Das mag erstaunen – wozu der ganze Aufwand bei einem so geringen Effekt? Bei einer großen Site, die am Limit des technisch Möglichen arbeitet, kann es aber schon sinnvoll sein. Dabei werden Sie selten eine Verbindung innerhalb eines Skripts mehrfach öffnen und schließen. Greifen aber in Spitzenzeiten Hunderte von Nutzern gleichzeitig zu, ist die Auswirkung des Verbindungspooling schon zu spüren.

Open ab Seite 54

4.1 Fields ab Seite 159

Die Einstellung erfolgt mit der Methode `Open` oder der `Properties`-Kollektion, wenn Sie OLEDB verwenden. Normalerweise ist das Verbindungspooling eingeschaltet. Ausschalten können Sie es folgendermaßen:

```
objConnection.Properties("OLE_DB_Services") = -2
```

Bei ODBC finden Sie die entsprechenden Optionen im ODBC-Manager.

3.1.2 Übersicht

Methoden

- `BeginTrans`, Seite 49
- `Cancel`, Seite 50
- `Close`, Seite 50
- `CommitTrans`, Seite 51
- `Execute`, Seite 51
- `Open`, Seite 54
- `OpenSchema`, Seite 56
- `RollbackTrans`, Seite 61

Eigenschaften

3.1.3 Methoden

BeginTrans

BeginTrans

Die Methode BeginTrans startet eine neue Transaktion.

```
Long = objConnection.BeginTrans
```

Als Transaktion bezeichnet man die Zusammenfassung verschiedener Prozesse zu einer Einheit. Die Befehle werden nur dann ausgeführt, wenn alle Teile der Transaktion erfolgreich abgeschlossen werden konnten. Misslingt ein Teil, werden auch alle anderen Aktionen rückgängig gemacht.

Der Rückgabewert der Methode bezeichnet die Stufe bei verschachtelten Transaktionen. Die folgende Abbildung zeigt das Prinzip:

Die Transaktion wird am Ende mit der Methode CommitTrans bestätigt. Traten Fehler auf, wird sie alternativ mit RollbackTrans zurückabgewickelt. Wird im Beispiel die Transaktion A am Ende nicht bestätigt, werden alle Datenbankbefehle dazwischen unwirksam, auch die der Transaktionen B und C. Wird dagegen die Transaktion C nicht bestätigt, bleiben A und B gültig.

Wenn der Parameter adXactCommitRetaining gesetzt ist, wird bei einer verschachtelten Transaktion nach jedem CommitTrans automatisch eine neue Transaktion gestartet, der erneute Aufruf von BeginTrans ist nicht notwendig.

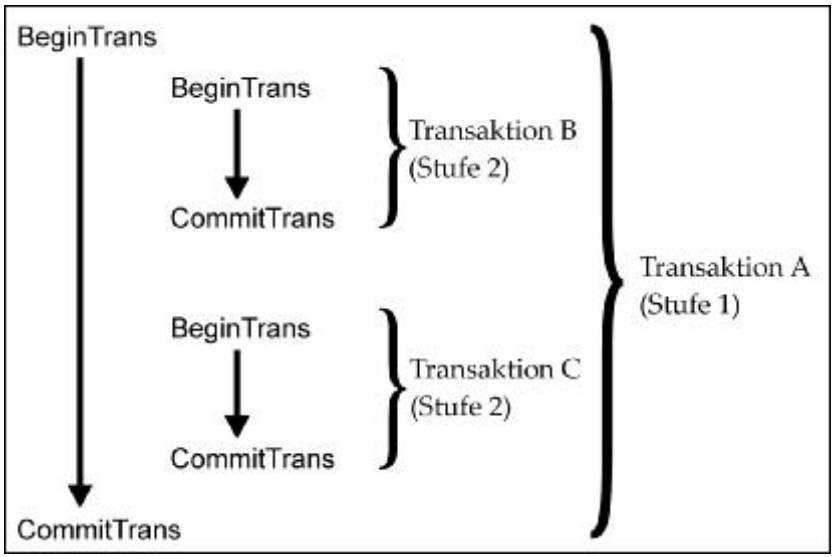


Abbildung 3.1: Prinzip verschachtelter Transaktionen

Nicht alle Datenquellen unterstützten Transaktionen. Wenn das DBMS Transaktionen nicht kennt, wird ein Laufzeitfehler erzeugt.

- `CommitTrans`, Seite 51
- `RollbackTrans`, Seite 61

Cancel

Cancel

Mit dieser Methode wird der Versuch einer Verbindungsaufnahme unterbrochen oder eine laufende `Execute`- oder `Open`-Methode abgebrochen.

```
objConnection.Cancel
```

Die Methode erzeugt beim Aufruf einen Laufzeitfehler, wenn `Execute` oder `Open` nicht mit dem Parameter `adRunAsync` aufgerufen wurden.

- `Execute`, Seite 51
- `Open`, Seite 54

Close

Close

Diese Methode schließt die Verbindung und alle damit verbundenen oder davon abhängigen Objekte. Das Objekt wird aber nicht aus dem Speicher entfernt.

```
objConnection.Close
```

Die Methode schließt auch alle Datensätze, die von der Verbindung abhängig sind. Abhängige `Command`-Objekte bleiben persistent, die erneute Nutzung führt aber zu einem entkoppelten Datensatz, da der Parameter `ActiveConnection` gelöscht wird.

Änderungen an Datensätzen, die nicht zurückgeschrieben wurden, gehen verloren. Offene Transaktionen werden zurückabgewickelt und die Transaktion selbst erzeugt einen Laufzeitfehler. Das Verbindungsobjekt wird am Ende des Skripts automatisch geschlossen. In diesem Fall wird kein Laufzeitfehler erzeugt, sondern die Transaktion bestätigt.

Um das Objekt zu entfernen, gehen Sie folgendermaßen vor:

```
objConnection.Close
Set objConnection = Nothing
```

Der Aufruf der Methode mit einem bereits geschlossenen Objekt führt zu einem Laufzeitfehler. Dies kann folgendermaßen verhindert werden:

```
If objConnection.State = adStateOpen Then
   objConn.Close
End If
Set objConnection = Nothing
```


- `Open`, Seite 54
- `Command`-Objekt, ab Seite 122

CommitTrans

Mit dieser Methode wird eine Transaktion explizit bestätigt. Alle Aktionen innerhalb des Transaktionsblocks sind nun gültig und werden in der Datenbank permanent gemacht.

objConnection`.CommitTrans`

Dieser Befehl wirkt automatisch nur zurück bis zum letzten Aufruf von `BeginTrans`. So sind verschachtelte Transaktionen möglich.

Nicht alle Datenquellen unterstützten Transaktionen. Wenn das DBMS Transaktionen nicht kennt, wird ein Laufzeitfehler erzeugt.

- `BeginTrans`, Seite 49
- `RollbackTrans`, Seite 61

Execute

`Execute` führt eine Abfrage an die Datenbank aus und erzeugt, falls Daten zurückgegeben werden, ein `RecordSet`-Objekt.

```
Set objRS = objConnection.Execute(string Command
                                  [,long affected]
                                  [,long Options])
```

```
objConnection.Execute(string Command
                      [,long affected]
                      [,long Options])
```

Parameter

Name	Typ	Option	Beschreibung	Standard
`Command`	String	-	Enthält eine SQL-Anweisung, einen Tabellennamen, den Namen einer gespeicherten Prozedur oder spezielle Befehle für den gewählten Provider.	
`affected`	Long	✓	Variable, in der der Provider die Anzahl der bearbeiteten Zeilen ablegt. Gilt nur für Anweisungen, die Änderungen an der Datenbank ergeben (z.B. `UPDATE`), nicht aber bei `SELECT`.	
`Options`	Long	✓	Parameter, der angibt, wie der Text in `Command` zu verarbeiten ist (siehe nächste Tabelle).	-1

Die Optionen geben an, wie der Wert in *`Command`* zu interpretieren ist. Die Angabe ist optional, denn `Execute` wird versuchen, dies selbst zu erkennen. Wenn Sie den Inhalt des Kommandos aber kennen, kann die Angabe den Vorgang beschleunigen.

Option (Konstante)	Bedeutung des Inhalts des Parameters Command
`adCmdText`	SQL-Anweisung
`adCmdTable`	Name einer Tabelle
`adCmdTableDirect`	Name einer Tabelle
`adCmdStoredProc`	Name einer gespeicherten Prozedur
`adCmdFile`	Ein gespeicherter Datensatz
`adCmdUnknown`	Nicht bekannt (Standardwert)
`adSyncExecute`	Asynchrone Ausführung
`adAsyncFetch`	Asynchrones Fetching
`adAsyncFetchNonBlocking`	Asynchrones Fetching ohne Blockbetrieb
`adExecuteNoRecords`	Das Kommando gibt keine Datensätze zurück

Sie können die Optionen kombinieren, indem eine binäre Addition ausgeführt wird:

```
objConnection.Execute strQuery, intRecords,
                      adCmdText And adAsyncExecute
```

`adExecuteNoRecords` unterdrückt die Erzeugung eines Datensatzes, führt den Befehl aber dennoch aus.

Der zurückgegebene Datensatz ist immer ein Standarddatensatz, der nur einen einfachen Vorwärts-Zeiger enthält und nur gelesen werden kann. Wenn Sie andere Eigenschaften benötigen, müssen Sie explizit mit dem `RecordSet`-Objekt arbeiten. Daraus resultieren auch andere Effekte. Wenn Sie zuvor ein `RecordSet`-Objekt erstellen, dessen Eigenschaften festlegen und dann diesem `RecordSet`-Objekt das Resultat der Abfrage übergeben, werden die Eigenschaften ignoriert. Das von `Execute` erzeugte Objekt ist völlig neu, es überschreibt das zuvor definierte `RecordSet`-Objekt. Wenn Sie also mit `RecordSet` arbeiten, sollten Sie dessen Methode `Open` verwenden.

Besser geeignet ist `Execute`, um Aktionen auf der Datenbank auszuführen, die keinen Datensatz zurückgeben, wie z.B. `USE`, `DELETE` oder `UPDATE`. Verwenden Sie dann unbedingt die Konstante `adExecuteNoRecords`. ADO versucht sonst, zuerst die Aktion auszuführen und dann festzustellen, ob etwas zurückgegeben wurde. Danach wird entschieden, ob ein `RecordSet`-Objekt erzeugt wird oder nicht. Diese Aktion kostet Zeit. Mit der Option nehmen Sie die Entscheidung vorweg und Ihr Skript läuft schneller.

Beispiele

Das folgende Beispiel ändert Datensätze in einer Tabelle und gibt die Anzahl der ausgeführten Änderungen aus. Ein `RecordSet`-Objekt wird nicht erzeugt:

```
strQuery = "UPDATE Products SET UnitPrice = UnitPrice * 1.10"
objConn.Execute strQuery, intRecs, adCmdText
echo "Es wurden " & intRecs & " Preise geändert"
```

Listing 3.1: Connection.Execute.1.asp: Daten ändern

Das folgende Beispiel gibt die gesamte Tabelle zurück:

```
strQuery = "SELECT * FROM Products"
set objRS = objConn.Execute(strQuery, , adCmdText)
do while not objRS.EOF
   echo objRS("ProductName")
   echo " kostet "
   echo objRS("UnitPrice") & " $<br>"
   objRS.MoveNext
loop
```

Listing 3.2: Connection.Execute.2.asp: Tabelle ausgeben

Beachten Sie die Setzung der Kommas zur Trennung der Parameter! Wenn Sie hier trotzdem eine Variable für den Parameter *affected* einsetzen, wird -1 zurückgegeben.

- CommandType-Property, Seite 127 (Abschnitt 3.4 *Command)*
- RecordSet, Seite 69

Open

Open

Open öffnet eine Verbindung. Dies ist die wichtigste Methode des Connection-Objekts.

```
objConnection.Open(string Connection
                   [,string User]
                   [,string Pass]
                   [,integer Options])
```

Parameter

Name	Typ	Option	Beschreibung	Default
Connection	String	-	Eine Verbindungszeichenfolge	
User	String	✓	Nutzername bei geschützten Verbindungen. Wenn der Name auch in der Verbindungszeichenfolge angegeben wurde, überschreibt der Parameter *User* die Verbindungszeichenfolge.	
Pass	String	✓	Kennwort des Nutzers	
Options	Long	✓	Verbindungsoptionen (siehe nächste Tabelle)	-1

Die Verbindungszeichenfolgen wurden bereits in Abschnitt 2.2.2 Verbindungszeichenfolgen, ab Seite 40 erläutert.

Die einzige derzeit mögliche Option finden Sie in der folgenden Tabelle:

Option (Konstante)	Bedeutung des Inhalts des Parameters Command
adAsyncConnect	Eröffnet eine asynchrone Verbindung

Die asynchrone Verbindung erlaubt es ADO, das Skript fortzusetzen, auch wenn die Antwort der Datenbank noch nicht erfolgt ist. Dadurch laufen Skripte, die vom Ergebnis der Aktion unabhängige Ausgaben erzeugen, subjektiv schneller ab. Wenn Sie die Daten aber anschließend benötigen, müssen Sie darauf auch warten. Sie können die Eigenschaft State verwenden, um später auf die Verbindung zu warten, wenn es längere Zeit dauert.

In der Literatur wird oft die Verwendung von `ConnectComplete` empfohlen. Dies ist ein Ereignis und wird unter ASP nicht unterstützt.

Beispiele

Das folgende Beispiel öffnet eine Verbindung zu einer ungeschützten Access 2000-Datenbank mit dem Namen ARTIKEL.MDB:

```
strConn = "Provider=Microsoft.Jet.OLEDB.4.0"
strDatabase = "artikel.mdb"
objConnection.Open(strConn & "; Data Source=" & strDatabase)
```

Das nächste Beispiel erlaubt den Zugriff auf eine SQL Server-Datenbank:

```
function open()
   dim strProvider, strDataSrc, strCatalog, strUser,
   dim strPassword, strConnection
   dim strQuery
   strProvider = "SQLOLEDB"' Provider
   strDataSrc  = "WWW"' Name des Servers
   strCatalog  = "Northwind"' Name der Datenbank
   strUser     = "sa"' Nutzername des SQL Servers
   strPassword = ""' Kennwort
   strConnection = "Provider=" & strProvider & ";
                   Data Source= " & strDataSrc & ";
                   Initital Catalog=" & strCatalog & ";
                   User Id=" & strUser & ";
                   Password=" & strPassword
   on error resume next
   Set objConn = Server.CreateObject("ADODB.Connection")
   objConn.Open strConnection
   if Err.Number > 0 then
      with Response
        .write "<div style=""color:red""><b>Fehler:</b> "
        .write Err.Description
        .write "</div>"
      end with
      open = FALSE
   else
      open = TRUE
   end if
   on error goto 0
   strQuery = "USE " & strCatalog
   objConn.Execute strQuery
end function
```

Listing 3.3: Die Standardfunktion open() aus der Datei open.inc.asp

Einfacher ist die Angabe einer ODBC-Quelle, da die Daten in der entsprechenden DSN stehen:

```
objConnection.Open("DSN=artikel")
```


- 2.2.2 Verbindungszeichenfolgen, Seite 40
- 2.2.4 DSN, Seite 42
- Verbindungspooling, Seite 47

OpenSchema

OpenSchema

Diese Methode stellt das Verbindungsschema dar. Die Angaben sind weitgehend vom Provider abhängig. Es ist keineswegs sicher, dass alle hier gezeigten Schemas auch funktionieren.

```
Set objRS = objConnection.OpenSchema(enum Query
                                     [, variant Restriction]
                                     [, variant SchemaID])
```

Die Parameter können Sie der folgenden Tabelle entnehmen:

Parameter

Name	Typ	Option	Beschreibung	Default
Query	String	-	Art der Abfrage des Schemas	
Restriction	String Array	✓	Array aus Abfragewerten für das Schema	
SchemaID	String	✓	GUID für eine providerspezifische Abfrage außerhalb OLEDB	

Zum Thema *Schema* ist zuerst eine Begriffsdefinition sinnvoll:

- Als *Katalog* wird allgemein eine Sammlung von Daten und zusätzlichen Definitionen bezeichnet, die *Schemas*. In Microsoft Access oder SQL Server ist der Katalog die Datenbank. Die Datenbank enthält neben den Daten weitere Informationen, u.a. zur Struktur.
- Als *Schema* wird eine Sammlung von Datenbankobjekten bezeichnet. Diese Objekte können nutzerabhängig sein und enthalten beispielsweise Angaben zum verwendeten Zeichensatz u.v.m. Microsoft Access kennt keine unterschiedlichen Schemas, so erscheint die gesamte Datenbank als ein Schema.

Anhang A.2 Schematas ab Seite 423 enthält die gesamte Liste an zulässigen Schemas und der entsprechenden Werte für den SQL Server und Access (Jet).

Beispiel

Das folgende Beispiel zeigt alle Tabellennamen einer Datenbank an:

```
Set objSchema = objConn.OpenSchema(adSchemaTables)
While Not objSchema.EOF
   echo "Tabelle: " & objSchema("TABLE_NAME") & "<br>"
   objSchema.MoveNext
Wend
```

Listing 3.4: Connection.OpenSchema.asp: Tabelleninformationen anzeigen

`OpenSchema` ist allgemein die einzige Methode, detaillierte Angaben über die Datenbank zu erhalten.

Die ersten Schritte mit `OpenSchema` sind erfahrungsgemäß nicht einfach. Das folgende Skript erlaubt den Zugriff auf alle Schemas und ist eine gute Spielwiese für eigene Experimente.

```
<% option explicit %>
<% Server.ScriptTimeOut = 600 %>
<% Response.Buffer = FALSE %>
<%
dim strQuery, i, k, schema, position, sfilter, afilter
dim objSchema, colSchema, intSchema, strSchema, strX, strS
set colSchema = Server.CreateObject("Scripting.Dictionary")
colSchema.Add "adSchemaAsserts", 0
colSchema.Add "adSchemaCatalogs", 1
colSchema.Add "adSchemaCharacterSets", 2
colSchema.Add "adSchemaCollations", 3
colSchema.Add "adSchemaColumns", 4
colSchema.Add "adSchemaCheckConstraints", 5
colSchema.Add "adSchemaConstraintColumnUsage", 6
colSchema.Add "adSchemaConstraintTableUsage", 7
colSchema.Add "adSchemaKeyColumnUsage", 8
colSchema.Add "adSchemaReferentialContraints", 9
colSchema.Add "adSchemaTableConstraints", 10
colSchema.Add "adSchemaColumnsDomainUsage", 11
colSchema.Add "adSchemaIndexes", 12
colSchema.Add "adSchemaColumnPrivileges", 13
colSchema.Add "adSchemaTablePrivileges", 14
colSchema.Add "adSchemaUsagePrivileges", 15
colSchema.Add "adSchemaProcedures", 16
colSchema.Add "adSchemaSchemata", 17
colSchema.Add "adSchemaSQLLanguages", 18
colSchema.Add "adSchemaStatistics", 19
colSchema.Add "adSchemaTables", 20
```

```
colSchema.Add "adSchemaTranslations", 21
colSchema.Add "adSchemaProviderTypes", 22
colSchema.Add "adSchemaViews", 23
colSchema.Add "adSchemaViewColumnUsage", 24
colSchema.Add "adSchemaViewTableUsage", 25
colSchema.Add "adSchemaProcedureParameters", 26
colSchema.Add "adSchemaForeignKeys", 27
colSchema.Add "adSchemaPrimaryKeys", 28
colSchema.Add "adSchemaProcedureColumns", 29
%>
<html>
<head>
   <title>Connection.OpenSchema</title>
</head>
<body>
<h1>Connection</h1>
<h2>OpenSchema</h2>
<div class=text>
Dieses Skript zeigt alle Schemas an.
</div>
<%
schema = Request.Form("schema") & Request.QueryString("schema")
if len(schema) = 0 then schema = 0
%>
<form method="post" action="<% = ASP_SELF %>">
Schema:
<select name="schema" size="1">
   <%
   for each strX in colSchema
      echo "<option value=""" & colSchema(strX) & """"
     if cint(colSchema(strX)) = cint(schema) then
        echo " selected "
         strSchema = strX
     end if
     echo ">$strX</option>"
   next
   %>
</select>
<input type="Submit" name="Submit" value="Anzeigen...">
</form>
<div class=text>
<%
' Datenverbindung herstellen
if open() and len(schema) > 0 then
  on error resume next
```

```
intSchema = cint(schema)
echo "<h4>Angezeigt wird Schema Nr.
      $intSchema ($strSchema)</h4>"
echo "<div class=text>Klicken Sie auf einen Text in der
      Tabelle, um diese Spalte mit diesem Bezeichner als
      Filter zu setzen. "
echo "Klicken Sie auf den Kopf der Spalte, um das Filter
      aufzuheben.</div><br>"
position = Request.QueryString("position")
if position > 0 then              ' Filtern?
   for i = 1 to cint(position)    ' Aufbau des Filters
    if position = 1 then
      afilter = ""
     exit for
    else
      afilter = afilter & "Empty, "
    end if
  next
  sfilter = Request.QueryString("sfilter")
  afilter = "Array(" & afilter & chr(34)
                     & sfilter & chr(34) & ")"
  position = position + 1
  echo "<div class=text><b>Aktives Filter:</b>
        <code>$sfilter</code> in Spalte <b>$position</b>.
        </div><br>"
  Set objSchema = objConn.OpenSchema(intSchema,
                                     eval(afilter))
else ' Nicht filtern
   Set objSchema = objConn.OpenSchema(intSchema)
end if
if Err.Number <> 0 then
   echo "<div style=""color:red; font-weight:bold"">
         Fehler: "
  echo Err.Description & " (" & Err.Number & ")"
  echo "</div><br>"
  Err.Clear
else
   echo "<table border=1>"
   echo "<tr>"
   for each strX in objSchema.Fields
     echo "<th class=head>"
     echo "<a href=$ASP_SELF?schema=$schema&position=0>"
     echo strX.Name
     echo "</a></th>"
   next
```

```
    echo "</tr>"
    while not objSchema.EOF
     echo "<tr>"
      ' Positionszaehler zur Steuerung des Filters der
      ' Methode OpenSchema
      position = 0
      ' Durchlaufen der Felder in horizontaler Richtung
      for each strX in objSchema.Fields
         ' Feldauswahl
         strS = objSchema(strX.Name)
         ' Null-Werte unterdruecken
         if isnull(strS) then strS = ""
         echo "<td nowrap class=text> "
         sfilter = ""
         ' Zeichenweise Ausgabe inkl. Sonderwerte
         for i = 1 to len(strS)
            if asc(mid(strS,i,1)) < 15 then
                sfilter = sfilter
                & " chr(" & asc(mid(strS,i,1)) & ") "
            else
                sfilter = sfilter & mid(strS,i,1)
            end if
         next
         echo "<a href=$ASP_SELF?
                schema=$schema
                &position=$position
                &sfilter=$sfilter>
                $sfilter
                </a>"
         echo "</td>"
         position = position + 1
      next
     echo "</tr>"
     objSchema.MoveNext
    wend
    echo "</table>"
  end if
end if
%>
</div>
</body>
</html>
```

Listing 3.5: Schema.ListSchemas.asp: Komfortabler Zugriff auf alle Schemas

Das Skript lässt die Auswahl eines Schemas zu und zeigt alle Daten an. Jeder Wert in jeder Spalte kann dann als Filter gesetzt werden, um die verschachtelten Werte aufzulösen.

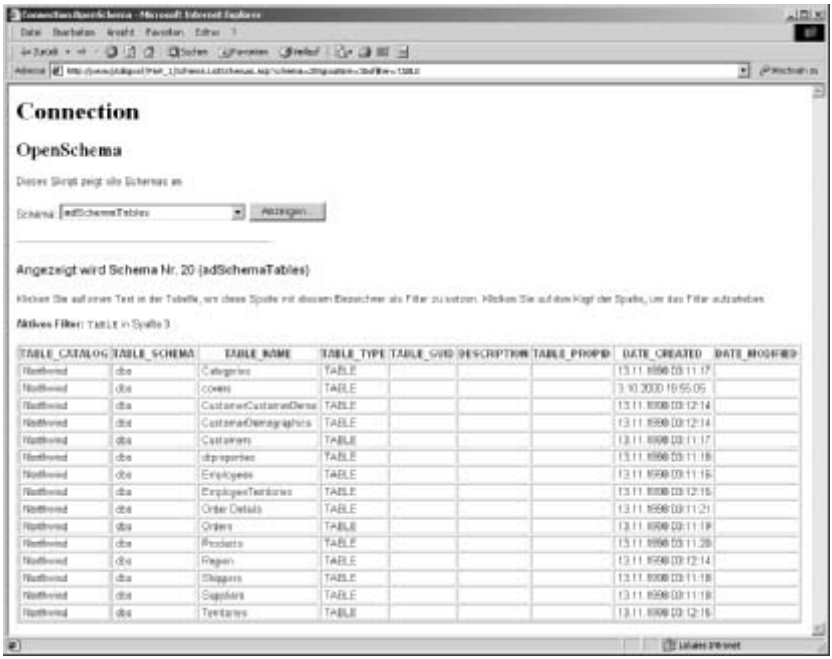


Abbildung 3.2: Anzeige von Tabelleninformationen mit dem Skript aus Listing 3.4 und einem Filter "Table" auf Spalte 4

RollbackTrans

RollbackTrans

Mit dieser Methode werden die innerhalb eines Transaktionsblocks vorgenommenen Transaktionen wieder rückgängig gemacht.

```
objConnection.RollbackTrans
```

Alle Änderungen an der Datenbank, die seit dem letzten `BeginTrans` ausgeführt wurden, werden rückgängig gemacht. Bei verschachtelten Transaktionen muss jeder vorhergehend gestartete Block abgeschlossen werden, entweder mit `CommitTrans` oder mit `RollbackTrans`.

- `BeginTrans`, Seite 49
- `CommitTrans`, Seite 51
- `Attributes`, Seite 62

3.1.4 Eigenschaften

Wenn bei den nachfolgend beschriebenen Eigenschaften zwei Syntaxdiagramme stehen, bei denen die Eigenschaft einmal auf der linken und einmal auf der rechten Seite der Zuweisung steht, so ist diese Eigenschaft schreib- und lesbar.

Attributes

Attributes

Diese Eigenschaft dient der Überwachung und Steuerung von Transaktionen.

```
long lngTrans = objConnection.Attributes
objConnection.Attribute = long lngTrans
```

Die zulässigen Attribute können Sie der folgenden Tabelle entnehmen:

Parameter

Attribut	Option
adXactCommitRetaining	Wird diese Option gesetzt, startet nach jedem CommitTrans automatisch eine neue Transaktion.
adXactAbortRetaining	Wird diese Option gesetzt, startet nach jedem RollbackTrans automatisch eine neue Transaktion.

Die beiden Optionen können durch binäre Addition kombiniert werden :

```
objConnection.Attributes = adXactCommitRetaing
                         And adXactAbortRetaining
```


- BeginTrans, Seite 49
- CommitTrans, Seite 51
- RollbackTrans, Seite 61

CommandTimeout

Command Timeout

Diese Eigenschaft bestimmt, wie lange auf die Ausführung eines Kommandos gewartet wird, bevor das Skript mit einem Laufzeitfehler abbricht.

```
long lngTime = objConnection.CommandTimeout
objConnection.CommandTimeout = long lngTime
```

Der Standardwert beträgt 30 Sekunden. Die Einstellung *lngTime* erfolgt in Sekunden. Setzen Sie den Wert auf 0, um unendlich zu warten.

Die Eigenschaft registriert die Zeit, die vergeht, bis der Provider Daten von der Datenbank erhält. Wenn Sie eine komplexe Anfrage starten, wird dies erwartungsgemäß funktionieren. Wenn Sie eine einfache Anfrage haben, die

aber sehr viele Datensätze zurückgibt, wird das nicht unbedingt erwartungsgemäß funktionieren. So könnte es sein, dass die Datenbank 250 000 Datensätze an einen clientseitigen Zeiger liefert. Wenn der erste Datensatz innerhalb der von `CommandTime` spezifizierten Zeit den Provider erreicht, gilt die Abfrage als erfolgreich und der Wert wird ignoriert. Wenn die Übertragung dann mehrere Minuten dauert, mag Ihnen das erscheinen, als ob das Skript steht und die Eigenschaft nicht funktioniert, denn nach der Timeout-Zeit erfolgt kein Abbruch. Dennoch ist das Verhalten korrekt. Falls Ihre Applikation derart langsam ist, sollten Sie einen Hinweis anzeigen und den Nutzer auf die Dauer vorbereiten. Setzen Sie in ASP `<% Response.Buffer = FALSE %>`, um die Pufferung auszuschalten und Zwischenergebnisse abzusenden. Achten Sie dabei darauf, keine HTML-Tabellen zu verwenden.

Die Eigenschaft `CommandTimeout` der `Properties`-Kollektion erbt diesen Wert nicht.

▶ `Fields`, Seite 159

ConnectionString

ConnectionString

Mit dieser Eigenschaft kann die Verbindungszeichenfolge eingestellt und ausgelesen werden.

```
string strConn = objConnection.ConnectionString
objConnection.ConnectionString = string strConn
```

ADO selbst interpretiert nur vier Elemente der Verbindungszeichenfolge. Alle anderen Werte werden direkt an den Provider weitergereicht.

Parameter

Attribut	Option
`Provider=`	Name des Providers
`File Name=`	Dateiname einer Datenlinkdatei (UDL-Datei), in der die Verbindungszeichenfolge steht
`Remote Provider=`	Provider bei einer RDS-Verbindung
`Remote Server=`	Server bei einer RDS-Verbindung

Eine ausführliche Diskussion der Verbindungszeichenfolgen finden Sie in den folgenden Abschnitten:

▶ 2.2.1 Verbindungen, Seite 40

▶ 2.2.2 *V*erbindungszeichenfolgen, Seite 40

▶ 2.2.3 Datenlinkdatei, Seite 41

▶ `Open`, Seite 54

ConnectionTimeOut

Connection TimeOut

Diese Eigenschaft bestimmt, wie lange auf die Öffnung einer Verbindung gewartet wird, bevor das Skript mit einem Laufzeitfehler abbricht.

```
long lngTime = objConnection.ConnectionTimeout
objConnection.ConnectionTimeout = long lngTime
```

Der Standardwert beträgt 15 Sekunden. Die Einstellung *lngTime* erfolgt in Sekunden. Wird der Wert auf 0 gesetzt, wird unbegrenzt gewartet.

Nachdem die Verbindung etabliert wurde, kann der Wert nicht mehr geändert werden.

CursorLocation

CursorLocation

Der interne Datensatzzeiger wird auch als Cursor (Zeiger) bezeichnet. Diese Eigenschaft bestimmt, wo der Zeiger aufgebaut wird.

```
integer intCursor = objConnection.CursorLocation
objConnection.CursorLocation = integer intCursor
```

Die Parameter können Sie der folgenden Tabelle entnehmen:

Parameter

Attribut	Beschreibung
adUseClient	Der Zeiger wird im Client erzeugt.
adUseClientBatch	Stapelzeiger im Client; dieser Wert ist nur aus Kompatibilitätsgründen vorhanden und sollte nicht eingesetzt werden.
adUseServer	Serverseitiger Zeiger
adUseNone	Zeigt an, dass keine Datensatzzeiger verwendet werden. Dieser Wert ist nur aus Kompatibilitätsgründen vorhanden und sollte nicht eingesetzt werden.

Wenn Sie Datensätze abkoppeln und dann damit weiterarbeiten, obwohl die Verbindung zur Datenbank unterbrochen wurde, müssen Clientzeiger verwendet werden.

Die Einstellung dieser Eigenschaft muss *vor* dem Öffnen der Verbindung erfolgen. Danach haben Änderungen keinen Effekt.

Mehr Informationen zu Zeigern finden Sie in Abschnitt 5.2 Datenbankzeiger ab Seite 197.

DefaultDatabase

DefaultDatabase

Diese Eigenschaft wählt eine Datenbank als Standarddatenbank aus. Die Anwendung ist nur sinnvoll, wenn der Provider keine Standarddatenbank kennt bzw. diese nicht ausgewählt wurde oder mehrere Datenbanken existieren und ein Wechsel erfolgen muss.

```
objConnection.DefaultDatabase = string strDB
string strDB = objConnection.DefaultDatabase
```

Unabhängig davon kann die Datenbank natürlich immer durch die entsprechende SQL-Syntax ausgewählt werden. Eine Alternative wäre die Anweisung `USE` :

```
objConnection.Execute("USE database")
```

▶ `Execute`, Seite 51

IsolationLevel

IsolationLevel

Diese Eigenschaft bestimmt das Ausmaß der Auswirkungen von Transaktionen, die von anderen Prozessen in der Datenbank ausgeführt werden.

```
objConnection.IsolationLevel = long lngLevel
long lngLevel = objConnection.IsolationLevel
```

Die Einstellung hat nur einen Effekt, nachdem mit `BeginTrans` eine Transaktion gestartet wurde. Die Parameter können Sie der folgenden Tabelle entnehmen:

Parameter

Attribut	Beschreibung
`adXactUnSpecified`	Der Provider hat ein anderes Isolationsniveau, aber der konkrete Wert ist nicht bekannt
`adXactChaos`	Ein höheres Transaktionsniveau hat die Kontrolle über die Datensätze übernommen. Änderungen anderer Nutzer können nicht überschrieben werden.
`adXactBrowse`	Dieser Parameter erlaubt es, unbestätigte Änderungen anderer Transaktionen zu sehen. Die Arbeit mit diesen Daten ist kritisch, denn Sie können nicht wissen, ob die Transaktion am Ende bestätigt oder verworfen wird.
`adXactCursorStability`	Dies ist der Standardwert. Sie können Änderungen anderer Transaktionen erst sehen, wenn diese bestätigt wurden.

Attribut	Beschreibung
`adXactRepeatableRead`	Dieser Wert bestimmt, dass Sie Änderungen erst sehen, wenn sie die Verbindung resynchronisiert haben. Neue Datensätze, die andere Nutzer hinzugefügt haben, erscheinen nach `Requery`.
`adXactIsolated`	Transaktionen sind vollkommen voneinander isoliert. Änderungen werden durch spätere Transaktionen überschrieben. Verschachtelte Transaktionen können zum »DeadLock« führen, Leistungseinbußen drohen.

Die Isolation von Transaktionen ist von Bedeutung, wenn mehrere Benutzer zur gleichen Zeit zugreifen und Daten verändern, die auch andere Benutzer einer Transaktion verwenden. Die reine Datensatzsperre (LockType) wird nicht immer ausreichen, da die Phase der Transaktion weiter reichen kann. Der einfachste Weg ist eine vollständige Isolation, was zwar einfach und sicher ist, oft aber zu drastischen Leistungseinbrüchen führt. Im Extremfall sperren sich Datensätze gegenseitig – der berüchtigte *Deadlock* tritt auf.

Typische Situationen

Wie das Problem entsteht, zeigt das folgende Beispiel: Ein Nutzer A beginnt eine Transaktion, er liest Datensätze ein und verändert diese. Dann liest er weitere Datensätze. Ein Nutzer B liest die veränderten Datensätze und beginnt ebenfalls eine Transaktion. Nun wird bei A ein Laufzeitfehler generiert und die Transaktion wird zurückabgewickelt. Die zuvor an B ausgelieferten Datensätze werden ungültig. B verfügt aber über eine lokale Kopie der Daten und auch über einen gültigen Transaktionsverlauf. Wenn er nun seinerseits die Daten zurückschreibt, zerstört er die Transaktionssteuerung von Nutzer A. Der Vorgang von B wird als Dirty Read, »schmutziges Lesen«, bezeichnet. Zugleich führt das Rückschreiben von B zu einem Nonrepeatable Read, einem »nicht wiederholbaren Lesen«. Werden Löschvorgänge ausgeführt, entstehen Phantomzeilen (Phantom Rows).

SQL Server

Der SQL Server verhindert standardmäßig Dirty Read, erlaubt aber Nonrepeatable Read und Phantom Rows. Das stellt einen guten Kompromiss zwischen Sicherheit und Leistung dar.

- `BeginTrans`, Seite 49
- `CommitTrans`, Seite 51
- `RollbackTrans`, Seite 61
- `Attributes`, Seite 62

Mode

Mode

Ermittelt oder setzt die Rechte zum Ändern von Daten.

```
objConnection.Mode = long lngMode
long lngMode = objConnection.Mode
```

Parameter

lngMode kann einen der folgenden Werte annehmen:

Konstante	Beschreibung
adModeUnknown	Unbestimmt (Standard)
adModeRead	Nur Leserecht
adModeWrite	Nur Schreibrecht
adModeReadWrite	Schreib- und Leserecht
adModeShareDenyRead	Verhindert, dass andere eine Verbindung zum Lesen öffnen können.
adModeShareDenyWrite	Verhindert, dass andere eine Verbindung zum Schreiben öffnen können.
adModeShareExclusive	Verhindert, dass andere eine Verbindung zum Schreiben oder Lesen öffnen können.
adModeShareDenyNone	Verhindert, dass andere eine Verbindung öffnen können.

Diese Eigenschaft ist nur im Zusammenhang mit MS Access sinnvoll. Access kann keine konkurrierenden Verbindungen verarbeiten. Sie können entsprechende Fehlermeldungen vermeiden, indem Sie die Verbindung exklusiv herstellen. Da Skripte oft nur Sekundenbruchteile laufen, ist das meist unkritisch. Für den professionellen Einsatz auf hochfrequentierten Sites ist Access auch denkbar ungeeignet, sodass dies nicht nachteilig erscheint. Auf SQL Server hat diese Eigenschaft keinen Einfluss – hier können Sie mehrere konkurriende Verbindungen aufbauen und Sperren auf Satzebene organisieren.

Solange die Verbindung geschlossen ist, kann der Wert gelesen und geschrieben werden. Nach dem Öffnen der Verbindung ist nur noch lesender Zugriff möglich.

Provider

Diese Eigenschaft bestimmt, welcher Provider genutzt wird.

Provider

```
objConnection.Provider = string strProvider
string strProvider = objConnection.Provider
```

Parameter

Die Parameter können Sie der folgenden Tabelle entnehmen:

Konstante	Beschreibung
MSDASQL	Provider für ODBC (Standardwert)
MSIDX	Index Server

Konstante	Beschreibung
ADSDSOObject	Active Directory Services
Microsoft.Jet.OLEDB.4.0	Microsoft Jet-Datenbanken (z.B. Access)
SQLOLEDB	SQL Server
MSDAORA	Oracle
MSDataShape	Provider für hierarchische Datensätze

Diese Eigenschaft wird auch durch `ConnectionString` gesetzt.

Solange die Verbindung geschlossen ist, kann der Wert gelesen und geschrieben werden. Nachdem die Verbindung geöffnet wurde, ist nur noch lesender Zugriff möglich.

- 2.2 Datenzugriff, Seite 40
- `Open`, Seite 54
- `ConnectionString`, Seite 63

State

State

Diese Eigenschaft gibt den Status einer Verbindung an. Sie kann nur gelesen werden.

```
integer intStatus = objConnection.State
```

Die Parameter können Sie der folgenden Tabelle entnehmen:

Parameter

Konstante	Beschreibung
adStateClosed	Die Verbindung ist geschlossen.
adStateOpen	Die Verbindung ist offen.

Da viele Methoden Laufzeitfehler erzeugen, wenn sie auf eine geschlossene Verbindung angewendet werden, sollten Sie den Status mit folgendem Code überprüfen:

```
if objConn.State = adStateOpen then
   strQuery = "SELECT * FROM Products"
   objConn.Execute(strQuery)
   echo "Verbindung war ge&ouml;ffnet, Abfrage konnte
         ausgef&uuml;hrt werden."
else
   echo "<b>Fehler:</b> Keine Verbindung."
end if
```

Listing 3.6: Connection.State.asp: Testen der Verbindung

- `Open`, Seite 54

Version

Diese Eigenschaft gibt die verwendete ADO-Version zurück. Derzeit ist das der Wert »2.6«.

Version

```
string strVersion = objConnection.Version
```

Sie können diese Funktion verwenden, um auf fremden Servern die Version festzustellen und Skripte zu aktivieren, die auf verschiedene Versionen abgestimmt sind. Damit lässt sich verhindern, dass Nutzer der Site mit Fehlermeldungen konfrontiert werden.

3.1.5 Kollektionen

Kollektionen werden in Kapitel 4 Kollektionen ab Seite 152 ausführlicher betrachtet. Zwei Kollektionen können von `Connection` abgeleitet werden:

- `Properties`. Diese Kollektion enthält `Property`-Objekte mit Eigenschaftsinformationen. Siehe Abschnitt 4.2 Properties ab Seite 164.
- `Errors`. Diese Kollektion enthält eines oder mehrere `Error`-Objekte mit Fehlerinformationen. Siehe Abschnitt 4.3 Errors ab Seite 165.

3.2 RecordSet

Das Datensatzobjekt enthält Informationen über den aktuell gelesenen Datensatz.

3.2.1 Einführung

Sie können dieses Objekt wie folgt instanziieren:

```
Set objRS = Server.CreateObject("ADODB.RecordSet")
```

Es wird auch implizit durch die Methode `Execute` der Objekte `Connection` und `Command` instanziiert, allerdings mit funktionalen Einschränkungen. Der volle Funktionsumfang steht nur zur Verfügung, wenn `RecordSet` direkt erzeugt wird. Ergänzend sollten Sie sich auch das Objekt `Record` ansehen, Abschnitt 3.3 ab Seite 113.

3.2.2 Übersicht

Die folgende Übersicht zeigt alle Methoden und Eigenschaften des Objekts auf einen Blick.

Methoden

Eigenschaften

3.2.3 Methoden

AddNew

AddNew

Fügt einem oder mehreren Feldern Werte hinzu. Das Datensatzobjekt erhält dabei einen neuen Datensatz.

objRS.AddNew [array *Felder*], [array *Werte*]

Felder können über den Feldnamen oder ihre Ordnungsnummer (mit 0 beginnend) angesprochen werden.

Das folgende Beispiel liest eine Tabelle der NorthWind-Datenbank und erlaubt das Hinzufügen eines weiteren Datensatzes über ein Formular. Es wird aus Platzgründen nur ausschnittsweise wiedergegeben. Das vollständige Listing finden Sie auf der Website zum Buch.

Erfahrungsgemäß erscheint einem der Umgang mit den Feldnamen als relativ lästig, zumal oft viele Tabellen angesprochen werden. Das folgende Listing ist äußerst kompakt, da es einige universelle Funktionen verwendet:

```
set objRS = Server.CreateObject("ADODB.RecordSet")
objRS.Open "SELECT * FROM Customers", objConn, adOpenDynamic,
            adLockOptimistic
echo "<form action=""" & ASP_SELF & """ method=post>"
show_form(objRS)
echo "<input type=submit value=Absenden>"
echo "</form>"
call form_to_array(arrFields, arrValues)
if objRS.Supports(adAddNew) then
   objRS.AddNew arrFields, arrValues
end if
show_table(objRS)
```

Listing 3.7: RecordSet.AddNew.asp: Hinzufügen von Feldern mit universellen Prozeduren

Die drei verwendeten Prozeduren sind in der Datei OPEN.INC.ASP zu finden. *show_table* zeigt eine Tabelle an und wurde bereits mehrfach verwendet. *show_form* liest alle Feldnamen einer Tabelle und erzeugt dazu die passenden Felder:

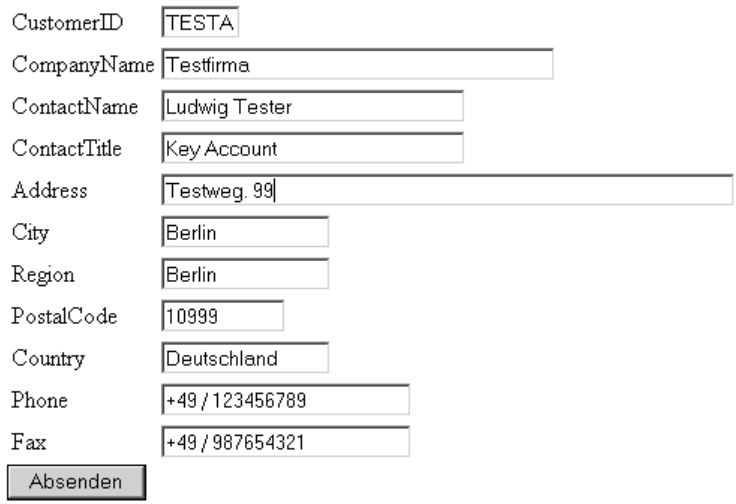
```
sub show_form(objRecordSet)
   dim fname
   const suffix = "_RSF"
   echo "<table border=0 cellpadding=1>"
   for each fname in objRecordSet.Fields
      echo "<tr>"
      echo "<td>" & fname.Name & "</td>"
```

```
      echo "<td>"
      echo "<input type=text "
      echo " size=" & cint(fname.DefinedSize / 2)
      echo " maxlength=" & fname.DefinedSize
      echo " name=""" & fname.Name & suffix & """
      echo " value=""" & Request.Form(fname.Name) & """
      echo ">"
      echo "</td>"
      echo "</tr>"
   next
   echo "</tr>"
   echo "</table>"
end sub
```

Listing 3.8: show_form aus der Datei open.inc.asp

Wenn Sie beschreibende Feldnamen für Ihre Tabellen verwenden, können Sie sich beim Erstellen der Formulare mit dieser Prozedur eine Menge Zeit sparen.

Abbildung 3.3: Das Formular wird in Abhängigkeit von Feldnamen und Feldgrößen aufgebaut



Für die Methode `AddNew` ist es notwendig, die Feldnamen und Werte als Array zu übergeben, wenn mehr als ein Name existiert. Die folgende Prozedur *form_to_array* übernimmt zwei Arrays und füllt diese mit Werten des gesendeten Formulars. Das Formular darf durchaus zusätzliche Felder, z.B. Hidden-Felder, enthalten.

```
sub form_to_array(byref arrFields, byref arrValues)
   dim i
   const suffix = "_RSF"
   i = 0
   for each formfield in Request.Form
    if instr(formfield, "_RSF") > 0 then
      redim preserve arrFields(i)
      redim preserve arrValues(i)
      arrFields(i) = left(formfield,
                          len(formfield) - len(suffix))
      arrValues(i) = Request.Form(formfield)
      i = i + 1
    end if
   next
end sub
```

Listing 3.9: form_to_array() aus der Datei open.inc.asp: Umwandeln eines kompletten Formulars in AddNew-gerechte Arrays

Wenn Sie solche Felder übertragen möchten, können Sie ein Suffix zu den Feldnamen verwenden, der diese eindeutig identifiziert. Im Beispiel in Listing 3.9 wurde das Suffix `_RSF` (RecordSet Fields) verwendet. Beim Übertragen der Feldnamen muss das Suffix natürlich wieder entfernt werden, wozu die `left`-Funktion verwendet wird. Der Name ist selbstverständlich willkürlich gewählt, eine funktionale Bedeutung steckt nicht dahinter.

Aktualisierung der Datenbank

Die Aktualisierung der Datenbank erfolgt in diesem Beispiel sofort. Sie können statt des Parameters `adLockOptimistic` auch `adLockBatchOptimistic` verwenden. Dann werden die Werte mit jedem Aufruf von `AddNew` im lokalen Objekt aktualisiert und müssen mit `UpdateBatch` in die Datenbank geschrieben werden. Der Status der Operation lässt sich mit der Eigenschaft `Status` überwachen. Solange ein Datensatz den Wert `adRecNew` erzeugt, ist er noch nicht in der Datenbank aktualisiert worden.

Im Falle der Übernahme von Daten aus einem Formular ist das nicht sinnvoll, da nur ein einziger Aufruf von `AddNew` erfolgt.

- `CancelUpdate`, Seite 75
- `Status`, Seite 112
- `Update`, Seite 94
- `UpdateBatch`, Seite 95

Cancel

Cancel

Diese Methode bricht laufende oder wartende, durch `Open` angestoßene asynchrone Operationen ab.

```
objRS.Cancel
```

Ein sinnvoller Einsatz ist der Abbruch sehr langwieriger Operationen. Für den Fall, dass seine Abfrage extrem lange dauert, weil z.B. eine einschränkende Bedingung vergessen wurde, kann im Skript mit dieser Methode ein Abbruch veranlasst werden.

CancelBatch

CancelBatch

Bricht die Änderungen von Feldern aus einem Batchlauf heraus ab und stellt den ursprünglichen Zustand wieder her.

```
objRS.CancelBatch(affected)
```

Sinnvoll bei auftretenden Fehlern. Der Wert für *affected* kann der folgenden Tabelle entnommen werden:

Parameter

Konstante	Beschreibung
`adAffectCurrent`	Bricht die Änderung nur für den aktuellen Datensatz ab.
`adAffectGroup`	Bricht Änderungen nur für die Datensätze ab, die von der `Filter`-Eigenschaft selektiert wurden.
`adAffectAll`	Bricht Änderungen für alle Datensätze ab (Das ist der Standardwert).
`adAffectAllChapters`	Betrifft alle Chapter einer Hierarchie auf derselben Ebene eines hierarchischen Datensatzes, unabhängig von irgendeinem Filter.

Probleme mit Filtern

Die Anwendung des Parameters `adAffectGroup` ist unter Umständen kritisch. Wenn Sie Änderungen abbrechen möchten und die betroffenen Datensätze vom aktuellen Filter nicht selektiert werden, wirkt sich der Befehl nicht aus. Dies ist immer genau dann der Fall, wenn Sie einen zeichenbasierten Filter verwendet haben. Die Gruppe wird also nur selektiert, wenn Sie Filter mit Hilfe von Arrays aus Lesezeichen erstellt haben, nicht mit Filterwörtern.

Im Gegensatz dazu wirkt auch `adAffectAll` in Abhängigkeit vom Filter. Diese Option wirkt sich nur auf die von einem zeichenfolgenbasierten Filter selektierten – also die sichtbaren – Datensätze aus. Filter mit Arrays aus Lesezeichen wirken jedoch nicht.

Wenn Sie tatsächlich alle Datensätze ansprechen möchten und keinen hierarchischen Datensatz verwenden, hilft `adAffectAllChapters`. Dies ist zwar weder logisch noch nachvollziehbar, aber es funktioniert.

- `Filter`, Seite 102
- `UpdateBatch`, Seite 95

CancelUpdate

CancelUpdate

Bricht Änderungen ab, bevor die `Update`-Methode aufgerufen wurde.

```
objRS.CancelUpdate
```

Die Methode macht auch die durch `AddNew` eingefügten Datensätze rückgängig. Als aktueller Datensatz wird der Datensatz eingestellt, der vor `AddNew` der aktuelle war.

Clone

Clone

Gibt eine Kopie eines Datensatzes zurück.

```
object objCloneRS = objRS.Clone(locktype)
```

Diese Methode erzeugt keinen weiteren, unabhängigen Datensatz. Es wird ein weiteres Datensatzobjekt erzeugt, das auf denselben Datensatz zeigt. Änderungen am Klon werden nicht im originalen Datensatz ausgeführt.

Parameter

Konstante	Beschreibung
`adLockUnspecified`	Der geklonte Satz übernimmt den Verriegelungstyp vom originalen.
`adLockReadOnly`	Der geklonte Satz kann nur gelesen werden.

Der Zugriff auf die Daten des Klons erfolgt unter Umständen schneller als auf das Original, da weitere Zugriffe auf die Datenbank selbst entfallen.

Diese Methode funktioniert nur, wenn Lesezeichen unterstützt werden. Lesezeichen sind zusätzliche Haltepunkte für den Datensatzzeiger. Um herauszufinden, ob Lesezeichen unterstützt werden, gehen Sie entsprechend folgendem Code vor:

```
set objRS = Server.CreateObject("ADODB.RecordSet")
objRS.Open "SELECT * FROM Products", objConn, adOpenStatic
if objRS.Supports(adBookmark) then
   set objClone = objRS.Clone
   echo "Lesezeichen werden unterst&uuml;tzt.<p>"
   while not objClone.EOF
      echo objClone("ProductName")
      echo " ==> $"
```

```
      echo formatnumber(objClone("UnitPrice"))
      echo "<br>"
      objClone.MoveNext
   wend
else
   echo "Lesezeichen werden nicht unters&uuml;tzt"
end if
```

Listing 3.10: RecordSet.Clone.asp: Klonen und Ausgeben eines Datensatzes

Wenn Sie Änderungen im Original ausführen, werden diese normalerweise im Klon reflektiert. Das funktioniert, bis das Original mit `Requery` aufgefrischt wird. Ab diesem Zeitpunkt ist der Klon abgekoppelt.

Lesezeichen, die Sie vom Original aus ermitteln, können auch auf den Klon angewendet werden. Dies ist in der Regel die einzige Möglichkeit, Lesezeichen zwischen Datensatzojekten auszutauschen.

▶ `Requery`, Seite 88

Close

Close

Schließt den Datensatz und gibt alle enthaltenen Daten frei.

objRS`.Close`

Wenn Änderungen im Datensatz erfolgten, führt die Methode `Close` zu folgenden Reaktionen:

- Im `Batchupdate`-Mode gehen alle Änderungen verloren.
- Im normalen `Update`-Mode wird ein Laufzeitfehler erzeugt.

Das Schließen führt nicht zum Zerstören des Objekts. Sie sollten deshalb zur Freigabe des Speichers folgendermaßen vorgehen:

```
objRS.Close
Set objRS = Nothing
```

Ein `RecordSet`-Objekt kann nur geschlossen werden, wenn es offen ist. Zur Vermeidung von Laufzeitfehlern gehen Sie folgendermaßen vor:

```
set objRS = Server.CreateObject("ADODB.RecordSet")
objRS.Open "SELECT * FROM Products", objConn
if objRS.State = adStateOpen then
   objRS.Close
   if objRS.State then
      echo "Datensatz wurde nicht geschlossen."
   else
      echo "Datensatz wurde geschlossen."
```

```
      set objRS = Nothing
   end if
end if
```

Listing 3.11: RecordSet.Close.asp: Schließen und Status überwachen

- `State`, Seite 111
- `Update`, Seite 94
- `UpdateBatch`, Seite 95

CompareBookmarks

Diese Methode vergleicht zwei Lesezeichen und gibt eine Information über die Unterschiede zurück.

Compare Bookmarks

```
long lngResult = objRS.CompareBookmarks(bookmark1, bookmark2)
```

Das Resultat *lngResult* nimmt einen der folgenden Werte an:

Parameter

Konstante	Beschreibung
adCompareLessThan	Das erste Lesezeichen liegt vor dem zweiten.
adCompareEqual	Beide Lesezeichen sind gleich.
adCompareGreaterThan	Das erste Lesezeichen liegt hinter dem zweiten.
adCompareNotEqual	Die Lesezeichen sind nicht gleich und befinden sich nicht in einer bestimmbaren Reihenfolge.
adCompareNotComparable	Ein Vergleich der Lesezeichen war nicht möglich.

Lesezeichen können nur verglichen werden, wenn es sich um denselben Datensatz oder einen Klon handelt. Lesezeichen existieren unabhängig von Filtern und Sortierprozessen.

Beachten Sie, dass diese Vergleichsmethode nicht die Daten vergleicht, auf die die Lesezeichen zeigen, sondern tatsächlich die Lesezeichen selbst.

Schon bei `Clone` wurde erwähnt, dass Lesezeichen nur zwischen Klonen kompatibel sind. Selbst dann, wenn zwei Datensatzobjekte auf derselben Abfrage basieren und »äußerlich« identisch sind, können Lesezeichen nicht verglichen werden. Wenn es dennoch funktioniert, ist dies eher Zufall als Methode. So enthalten clientseitige Datensätze nie Lesezeichen, in diesem Fall sind beide Argumente -1 – der Vergleich wird also positiv ausfallen.

Die Methode erzeugt einen Laufzeitfehler, wenn eines der beiden Argumente nicht existiert. Der Fehler kann abgefangen werden.

- Clone, Seite 75
- Filter, Seite 102
- Sort, Seite 109
- Bookmark, Seite 98

Delete

Delete

Diese Methode löscht den aktuellen Datensatz oder eine Gruppe von Datensätzen.

```
objRS.Delete([affected])
```

Der Parameter *affected* kann einen der folgenden Werte annehmen:

Parameter

Konstante	Beschreibung
adAffectCurrent	Löscht nur den aktuellen Datensatz (Standard).
adAffectGroup	Löscht die Datensätze, die von der Eigenschaft Filter selektiert wurden.
adAffectAllChapters	Löscht verbundene Datensätze.

Wenn Sie im Batch-Mode arbeiten, werden die betroffenen Datensätze nur zum Löschen markiert. Erst der Aufruf der UpdateBatch-Methode führt zum Löschen in der Datenbank.

- UpdateBatch, Seite 95

Find

Find

Diese Methode durchsucht einen Datensatz nach bestimmten Filterkriterien. Unter Umständen bietet sich der Einsatz an, um eine globale SELECT-Abfrage (SELECT * FROM table) mehrfach auszuwerten. Für eine einzige gefilterte Abfrage ist SELECT normalerweise schneller.

```
objRS.Find("filter", offset, direction, start)
```

Die Parameter können Sie der folgenden Tabelle entnehmen:

Parameter

Parameter	Beschreibung
`offset`	Verschiebung des Startpunkts der Suche, entweder vom aktuellen Datensatz oder, wenn *start* angegeben wurde, von dort. Die Angabe ist optional, der Standardwert ist 0.
`direction`	Gibt die Richtung an, in der gesucht wird (optional): • `adSearchForward`: vorwärts suchen (dies ist der Standardwert) • `adSearchBackward`: rückwärts suchen
`start`	Gibt den Startpunkt der Suche an (optional): • `adBookmarkCurrent`: beginnt beim aktuellen Datensatz (dies ist der Standardwert) • `adBookmarkFirst`: beginnt beim ersten Datensatz • `adBookmarkLast`: beginnt am Ende

Operatoren

Als Filter setzen Sie eine Zeichenkette ein, die etwa den Möglichkeiten entspricht, welche die `WHERE`-Bedingung in SQL bietet. Im Unterschied zu SQL ist als Platzhalterzeichen nicht nur `%`, sondern auch `*` zulässig. Allerdings lassen sich Ausdrücke nicht mit den logischen Operatoren AND, OR usw. kombinieren. Sie können nur eine einfache, eindimensionale Suche ausführen. Zeichenketten setzen Sie in einfache Anführungszeichen:

```
set objRS = Server.CreateObject("ADODB.RecordSet")
objRS.Open "SELECT * FROM Products", objConn, adOpenStatic
if (len(Request.Form("Filter")) > 0) then
   strFind = "ProductName " & Request.Form("Operator") & " '"
             & Request.Form("Search") & "'"
   objRS.Find strFind
end if
if objRS.EOF then
   echo "Name wurde nicht gefunden<br>"
else
   while not objRS.EOF
      echo objRS("ProductName") & "<br>"
      objRS.MoveNext
   wend
end if
```

Listing 3.12: RecordSet.Find.asp: Suchen eines Datensatzes (Ausschnitt)

Der in `WHERE`-Bedingungen beliebte Operator `LIKE` ist verfügbar:

```
objRS.Find "stadt LIKE 'Berlin*'"
```

Diese Variante findet alle Städtebezeichnungen, die mit Berlin beginnen, z. B. »Berlin-Kreuzberg«, »Berlin-Mitte«, aber auch »Berlinchen«.

GetRows

GetRows

Diese Methode liest mehrere Datensätze aus und überführt sie in ein Array. Die Weiterverarbeitung in einem Array ist unter Umständen einfacher und flexibler.

```
array = objRS.GetRows([rows], [start], [fields])
```

Überführt werden die Datensätze in ein zweidimensionales Array, das automatisch erzeugt wird. Die erste Dimension enthält die Feldbezeichnungen, die zweite Dimension zeigt auf die Datensätze. Standardmäßig werden alle Reihen übertragen. Die Parameter können Sie der folgenden Tabelle entnehmen:

Parameter

Parameter	Beschreibung
rows	Der Parameter zeigt an, wie viele Reihen vom Datensatzobjekt übertragen werden. Der Standardwert ist adGetRowsRest und überträgt alle Reihen.
start	Gibt ein Lesezeichen an, ab dem die Daten übertragen werden oder eine der Konstanten für Lesezeichen, die in der folgenden Tabelle beschrieben werden.
fields	Gibt an, welche Felder übertragen werden. Dies kann entweder eine Zeichenkette mit einem Feldnamen, ein Array mit Feldnamen oder Positionsnummern der Felder sein

Die Steuerung über Lesezeichen kann mit folgenden Konstanten erfolgen, die Sie für *start* einsetzen können:

Konstante	Beschreibung
adBookmarkCurrent	Ab dem aktuellen Datensatz
adBookmarkFirst	Ab dem ersten Datensatz
adBookmarkLast	Ab dem letzten Datensatz

Die Anzahl der übertragenen Datensätze können Sie mit Ubound ermitteln, wie im folgenden Beispiel gezeigt:

```
set objRS = Server.CreateObject("ADODB.RecordSet")
objRS.Open "SELECT * FROM Products", objConn, adOpenStatic
arrRS  = objRS.GetRows
intCol = Ubound(arrRS, 1)
intRow = Ubound(arrRS, 2)
for i = 0 To intRow
   echo "<tr>"
   for j = 0 To intCol
```

```
      echo "<td>" & arrRS(j, i) & "</td>"
   next
   echo "</tr>"
next
```

Listing 3.13: RecordSet.GetRows.asp: Tabelle komplett an Array übergeben

Die Methode weist in Bezug auf bestimmte Datentypen Einschränkungen auf. Memo-Felder oder Bilder (binäre Felder) werden nicht übertragen. Die entsprechenden Zellen des Arrays bleiben dann leer. Ein Laufzeitfehler erfolgt nicht.

Die Feldauswahl erfolgt am einfachsten mit einem Array:

```
set objRS = Server.CreateObject"ADODB.RecordSet")
objRS.Open "SELECT * FROM Products", objConn, adOpenStatic
arrFields = Array("ProductID", "ProductName", "UnitPrice")
arrRS = objRS.GetRows(, , arrFields)
intCol = Ubound(arrRS, 1)
intRow = Ubound(arrRS, 2)
for i = 0 To intRow
   echo "<tr>"
   for j = 0 To intCol
      echo "<td>" & arrRS(j, i) & "</td>"
   next
   echo "</tr>"
next
```

Listing 3.14: RecordSet.GetRows.2.asp: GetRows mit Spaltenauswahl

Beachten Sie hier auch die Schreibweise, wenn die vorderen optionalen Werte nicht angegeben wurden.

Die Ausgabe der Daten mit `GetRows` entspricht einem normalen Lesevorgang im Datensatz. Der Datensatzzeiger wird also auch weiter gesetzt. Wenn alle Datensätze ausgelesen werden, ist `EOF` anschließend `TRUE`. Wird nur ein Teil übergeben, steht der Zeiger auf dem Datensatz, der dem letzten gelesenen folgt.

`GetRows` ist bei einer großen Anzahl Datensätze nicht sehr effizient. Versuchen Sie, mehrere kleinere Blöcke anstatt eines großen zu lesen.

GetString

GetString

Diese Methode überführt den gesamten Datensatz in eine Zeichenkette.

```
string strTable = objRS.GetString([integer format],
                                  [integer anzahl],
                                  [string feldtrenner],
                                  [string reihentrenner],
                                  [string nullwert])
```

format kann nur den Wert `adClipString` annehmen, andere Werte sind zukünftigen Entwicklungen vorbehalten. *anzahl* bezeichnet die Anzahl der Datensätze. Mit *feldtrenner* und *reihentrenner* werden Trennzeichen spezifiziert. Der *nullwert* wird eingesetzt, wenn das Feld leer (`NULL`) ist. Alle Angaben sind optional. Der Standardwert für den *feldtrenner* ist der Tabulator, für den *reihentrenner* ein Zeilenvorschub.

Interessant ist der Einsatz zum Erzeugen einer HTML-Tabelle, wie im folgenden Beispiel gezeigt:

```
set objRS = Server.CreateObject("ADODB.RecordSet")
objRS.Open "SELECT * FROM Products", objConn, adOpenStatic
echo "<TABLE BORDER=1><TR><TD>"
strTable = objRS.GetString(2, -1, "</TD><TD>",
                           "</TD></TR><TR><TD>", " ")
echo strTable
echo "</TD></TR></TABLE>"
```

Listing 3.15: RecordSet.GetString.asp: Datensätze als Zeichenkette ausgeben

Dabei bestehen Feldtrennungen immer aus dem Wechsel einer Tabellenzelle (`</TD><TD>`), Zeilenwechsel zusätzlich aus dem Wechsel der Reihe (`</TD></TR><TR><TD>`). Damit leere Tabellenzellen das Layout nicht stören, sollten Sie ein HTML-Leerzeichen einsetzen (` `).

Interessant ist auch die Überführung einer Tabelle in eine CSV-Datei:

```
set objRS = Server.CreateObject("ADODB.RecordSet")
objRS.Open "SELECT * FROM Products", objConn
strFile = objRS.getString(2, -1, ";", chr(10)&chr(13))
' Textdatei erzeugen und fuellen (Schreibrechte erforderlich)
set objFO = Server.CreateObject("Scripting.FileSystemObject")
set objTO = objFO.CreateTextFile(Server.MapPath("data.txt"))
objTO.Write strFile
' Textdatei schliessen
objTO.Close
set objTO = Nothing
' Textdatei erneut oeffnen und auslesen
```

```
set objTO = objFO.OpenTextFile(Server.MapPath("data.txt"), 1)
while not objTO.AtEndOfStream
   echo objTO.ReadLine
   echo "<br>"
wend
```

Listing 3.16: RecordSet.GetString.CSV.asp: CSV-Datei erzeugen

Die komplette Tabelle befindet sich nun in für andere Programme lesbarer Form in der Datei »data.txt«. Als Trennzeichen wird das Semikolon eingesetzt, als Zeilentrenner CRLF (Wagenrücklauf und Zeilenvorschub), wie unter Windows üblich. Für den Export nach Unix würde der Zeilenvorschub `chr(13)` genügen.

Move

Bewegt den Datenbankzeiger vorwärts oder rückwärts.

Move

objRS.**Move**(integer *Records*, [integer *start*])

Ist *Records* positiv, wird vorwärts gezählt, ansonsten rückwärts. *start* gibt den Startpunkt vor. Dazu wird ein Lesezeichen genutzt, der reguläre Datenbankzeiger wird nicht vorher verschoben. Das Datensatzobjekt muss deshalb Lesezeichen unterstützen. Die Steuerung über Lesezeichen kann mit folgenden Konstanten erfolgen, die Sie für *start* einsetzen können:

Parameter

Konstante	Beschreibung
adBookmarkCurrent	Ab dem aktuellen Datensatz
adBookmarkFirst	Ab dem ersten Datensatz
adBookmarkLast	Ab dem letzten Datensatz

Die Bewegung des Datensatzzeigers kann durch den Typ des Datensatzes eingeschränkt sein. So ist ein Datensatz, der mit `adForwardOnly` erzeugt wurde, nicht in der Lage, den Zeiger rückwärts zu setzen. Wenn Sie einen Wert angeben, der den Zeiger außerhalb des Datensatzes platziert, wird der beim ersten Versuch auf `BOF` oder `EOF` gesetzt. Erfolgt dann ein erneuter Versuch, den Zeiger außerhalb der Grenzen zu setzen, wird ein Laufzeitfehler erzeugt.

Wurden Daten geändert und die `Update`-Methode wurde nicht verwendet, wird sie von `Move` implizit aufgerufen. Wenn Sie Änderungen nicht ausführen möchten, muss zuvor `CancelUpdate` aufgerufen werden:

- `Update`, Seite 94
- `CancelUpdate`, Seite 75
- `MoveFirst`, Seite 84
- `MoveLast`, Seite 84

MoveFirst

MoveFirst Bewegt den Datenbankzeiger zum ersten Datensatz.

```
objRS.MoveFirst
```

Diese Methode benötigt keine Parameter. Es gelten auch die bei `Move` gemachten Aussagen.

▶ `Move`, Seite 83

MoveLast

MoveLast Bewegt den Datenbankzeiger zum letzten Datensatz.

```
objRS.MoveLast
```

Diese Methode benötigt keine Parameter. Es gelten auch die bei `Move` gemachten Aussagen.

▶ `Move`, Seite 83

MoveNext

MoveNext Bewegt den Datenbankzeiger zum nächsten Datensatz.

```
objRS.MoveNext
```

Diese Methode benötigt keine Parameter. Es gelten auch die bei `Move` gemachten Aussagen.

▶ `Move`, Seite 83

MovePrevious

MovePrevious Bewegt den Datenbankzeiger zum vorhergehenden Datensatz.

```
objRS.MovePrevious
```

Diese Methode benötigt keine Parameter. Es gelten auch die bei `Move` gemachten Aussagen.

▶ `Move`, Seite 83

NextRecordSet

NextRecordSet Wenn mehrere Datensätze zurückgegeben werden, dient diese Methode dazu, den aktuellen Datensatz zu löschen und zum nächsten zu gehen.

```
Set objRS2 = objRS.NextRecordSet([Affected])
```

Wird die Variable *Affected* angegeben, enthält sie nach der Operation die Anzahl der betroffenen Datensätze. Dies ist nur sinnvoll, wenn es sich um eine Operation handelt, die keine Datensätze zurückgibt.

Mehrere Datensätze entstehen, wenn verbundene SQL-Anweisungen ausgeführt werden, z.B. in gespeicherten Prozeduren. SQL-Anweisungen lassen sich aber auch wie im folgenden Listing gezeigt kombinieren – getrennt durch Semikola:

```
set objRS = Server.CreateObject("ADODB.RecordSet")
strQuery = "SELECT * FROM Products;"
strQuery = strQuery & "SELECT * FROM Suppliers;"
strQuery = strQuery & "SELECT * FROM Shippers;"
objRS.Open strQuery, objConn
show_table(objRS)
echo "<hr>"
set objRS = objRS.NextRecordSet
show_table(objRS)
echo "<hr>"
set objRS = objRS.NextRecordSet
show_table(objRS)
```

Listing 3.17: RecordSet.NextRecordset.asp: Auswahl mehrerer unabhängiger Datensätze aus einer Abfrage. Die Sub-Prozedur show_table gibt eine beliebige Tabelle im HTML-Format aus (siehe open.inc.asp)

Ein solcher Block wird von jeder SQL-Datenbank ausgeführt. Da Satzaufbau und Feldnamen jeweils unterschiedlich ist, kann ein einzelnes Datensatzobjekt dies nicht verarbeiten. Wenn Sie ein solches Gebilde senden, wird der erste `SELECT`-Befehl ausgeführt. Mit dem ersten `NextRecordSet` wird ein neues Datensatzobjekt erzeugt und der zweite Befehl ausgeführt usw. Bei der Berechnung des Laufzeitverhaltens ist zu beachten, dass die Ausführung tatsächlich erst erfolgt, wenn die nächste Anweisung mit `NextRecordSet` angefordert wird.

Manche SQL-Abfragen erzeugen auch multiple Datensätze, wenn Gruppierungsoperatoren wie `COMPUTE` eingeführt werden. Jede Gruppierung erzeugt dann unter Umständen zwei Datensätze: einen für die Daten und einen für das berechnete Zwischenergebnis. Noch stärker sind Rückgaben strukturiert, wenn die Anweisungen `WITH CUBE` oder `ROLLUP` verwendet werden.

Open

Open

Öffnet ein Datensatzobjekt. Das Objekt hat einen eigenen Datensatzzeiger, der unabhängig von der Tabelle arbeitet.

```
objRS.Open([source], [connection], [cursor], [lock], [option])
```

source ist der Name eines `Command`-Objekts, einer Tabelle, einer SQL-Prozedur oder eine vollständige SQL-Anweisung. *connection* ist der Name eines `Connection`-Objekts. Der Zeiger *cursor* kann eine der folgenden Eigenschaften haben:

Parameter

Konstante	Beschreibung
`adOpenForwardOnly`	Der Zeiger kann sich in der Tabelle nur vorwärts bewegen (Standard).
`adOpenKeyset`	Der Zeiger reagiert auf Änderungen und Löschen von Datensätzen durch andere Nutzer, Einfügungen neuer Datensätze werden nicht übertragen.
`adOpenDynamic`	Der Zeiger überträgt alle Einflüsse anderer Nutzer.
`adOpenStatic`	Der Zeiger reagiert auf keinerlei Einflüsse anderer Nutzer.

Das Verhalten des Datensatzes bei Zugriffen anderer Nutzer auf dieselbe Tabelle wird mit *lock* gesteuert, wofür Sie folgende Werte einsetzen können:

Konstante	Beschreibung
`adLockReadOnly`	Daten können nicht geändert werden (Standard).
`adLockPessimistic`	Der Server verriegelt den Datensatz, wenn mit der Änderung begonnen wird (empfohlen).
`adLockOptimistic`	Die Verriegelung erfolgt nur, wenn ein Update-Kommando ausgeführt wird.
`adLockBatchOptimistic`	Die Verriegelung erfolgt nur, wenn ein Update-Kommando im Batchmodus ausgeführt wird.

Der Parameter *option* gibt an, welche Art Kommando ausgeführt wird. Der SQL Server benötigt diese Angabe nicht unbedingt, sie beschleunigt aber die Ausführung. Sie können folgende Werte einsetzen:

Konstante	Beschreibung
adCMDText	SQL-Kommando
adCMDTable	Tabellenname
adCMDStoredProc	Eine gespeicherte Prozedur (in SQL!)
adCMDUnknown	Nicht bekannt (Standardwert)
adCmdFile	Das Kommando ist ein gespeicherter Datensatz
adCmdTableDirect	Tabellenname
adAsyncFetch	Die Datensätze werden asynchron gelesen
adAsyncFetchNonBlocking	Die Datensätze werden asynchron gelesen, sie werden dabei aber nicht blockiert

Die Ausführung entspricht dem Verhalten beim Umgang mit dem Command-Objekt. Sie finden dort eine nähere Erläuterung. Die beiden Operatoren adAsyncFetch und adAsyncFetchNonBlocking können mittels Or mit den anderen kombiniert werden.

```
strQuery = "SELECT * FROM Products"
objRS.Open (strQuery, objConn, adOpenKeyset, adLockOptimistic,
           (adCmdText Or adAsynchFetch)
```

Mit adAsyncFetch wird gesteuert, wie die Methode sich verhält, wenn Datensätze nicht verfügbar sind, z.B. weil andere Nutzer darauf schreibend zugreifen. Wenn Sie adAsyncFetch angeben, wird gewartet, bis der Datensatz wieder frei ist, mit adAsyncFetchNonBlocking erhalten Sie keine Daten und EOF wird TRUE. Normalerweise wird die Methode freigegeben und das Skript weiter ausgeführt, wenn noch Datensätze von einem im Hintergrund laufenden Thread geholt werden. Werden diese jedoch benötigt, stoppt der Hauptthread und wartet, bis die Daten da sind. Auch dieses Verhalten wird mit adAsynchFetch erreicht.

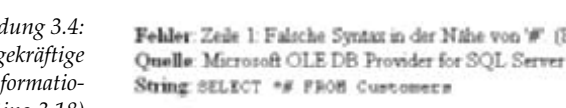
Bei Open treten erfahrungsgemäß die meisten Fehler auf. Es ist sinnvoll, hier eine Auswertung der Errors-Kollektion vorzunehmen. Das folgende Listing zeigt die Definition einer verbesserten Open-Funktion in OPEN.INC.ASP, *open_query()*.

```
sub open_query(byref objRecordSet, strQuery, objConnection,
               intLockType, intCursorType, intOptions)
   dim errorfield
   on error resume next
   objRecordSet.Open strQuery, objConnection, intLockType,
                     intCursorType, intOptions
   if Err.Number <> 0 then
      for each errorfield in objConnection.Errors
        echo "<b>Fehler</b>: " & errorfield.Description
             & " (" & hex(errorfield.Number) & ")<br>"
```

```
        echo "<b>Quelle</b>: " & errorfield.Source & "<br>"
        echo "<b>String</b>: <code>" & strQuery & "</code><p>"
      next
   end if
   on error goto 0
end sub
```

Listing 3.18: open_query() in open.inc.asp: komfortable Prozedur zum Abfangen von SQL-Fehlern

Abbildung 3.4: Aussagekräftige Fehlerinformationen (Listing 3.18)



Mehr Informationen zum Umgang mit dem `Error`-Objekt und der `Errors`-Kollektion finden Sie in Abschnitt 3.8 Error ab Seite 153 und Abschnitt 4.3 Errors ab Seite 165.

- `Command`, Seite 122
- `Close`, Seite 76

Requery

Requery

Frischt alle Datensätze neu auf, indem die Befehle erneut ausgeführt werden, die zur Erzeugung des Datensatzobjekts führten.

objRS.Requery([*option*])

Der Parameter *option* entspricht dem der Methode `Open`. Die Angabe ist optional und hat auch keinen Effekt, außer zur Optimierung. Die einzigen Änderungen, die tatsächlich ausgeführt werden, sind `adAsynchFetch` und `adAsynchFetchNonBlocking`, wie bei `Open` bereits beschrieben.

Intern ruft `Requery` erst `Close` auf, um dann die Abfrage erneut mit `Open` auszuführen. Dadurch kann sich der Inhalt des Datensatzobjekts dramatisch ändern. Wenn Sie Lesezeichen verwenden und diese angelegt haben, bevor `Requery` aufgerufen wurde, funktioniert Ihr Skript möglicherweise nicht mehr korrekt. Vielleicht hilft in diesen Fällen `Resync`.

- `Open`, Seite 86
- `Close`, Seite 76
- `Resync`

Resync

Resync

Synchronisiert die Datensätze in der Tabelle mit denen im Datensatzobjekt. Hat keine Auswirkungen auf neu der Tabelle hinzugefügte Datensätze.

objRS.Resync([*affected*], [*values*])

Parameter

Der Parameter *affected* kann sein:

Konstante	Beschreibung
adAffectCurrent	Synchronisation für den aktuellen Datensatz.
adAffectGroup	Synchronisation nur für die Datensätze, die von der Filter-Eigenschaft selektiert wurden.
adAffectAll	Synchronisation für alle Datensätze (Standard).
adAffectChapters	Auswirkung auf alle Chapter der gleichen Ebene eines hierarchischen Datensatzobjekts

Für den optionalen Parameter *values* kann Folgendes eingesetzt werden:

Konstante	Beschreibung
adResyncAllValues	Frischt alle Werte auf, wartende Änderungen werden nun geschrieben.
adResyncUnderlyingValues	Hier werden nur die Werte aufgefrischt, die nicht zu Änderungen in der Datenbank führen.

Im Gegensatz zu Requery wird die Abfrage nicht erneut ausgeführt.

Datensatzobjekte, die serverseitig existieren, können nicht mit Resync erneuert werden. Dies führt zu einem Laufzeitfehler. Clientseitige Datensatzobjekte dürfen nicht schreibgeschützt sein.

▶ UnderlyingValue, Seite 134

Save

Diese Methode speichert einen Datensatz komplett in einer Datei.

Save

objRS.Save(*file*, [*persist*])

Gespeichert wird nur die Auswahl, die durch Filter getroffen wurde, wenn ein Filter aktiv ist. Ist asynchroner Zugriff eingestellt, blockiert Save die gesamte Tabelle, bis der Lesevorgang abgeschlossen wurde. Nach der Ausführung wird der Datensatzzeiger auf den ersten Datensatz gesetzt. *file* kann auch eine Referenz zu einem Stream-Objekt sein. Die Angabe des Dateinamens ist optional, wenn Save bereits einmal mit Dateiname aufgerufen wurde. Künftige Zugriffe verwenden dann denselben Dateinamen und die Datensätze werden angehängt.

persist kann einen der folgenden Werte annehmen:

Parameter

Konstante	Beschreibung
adPersistADTG	ADTG steht für »Advanced Data TableGram«. Dies ist der Standardwert und bestimmt das Ausgabeformat.
adPersistXML	XML steht für »Extensible Markup Language«. Das XML-Format entspricht dem DOM-Tree.

Beispiel

Das folgende Beispiel zeigt die Anwendung. Die Ausgabe der XML-Datei erfolgt mit einer einfachen Prozedur zum zeilenweisen Lesen der Datei, die mit `HTMLEncode` für eine sichtbare Ausgabe im Browser sorgt.

```
set objRS = Server.CreateObject("ADODB.RecordSet")
set objFO = Server.CreateObject("Scripting.FileSystemObject")
objRS.Open "SELECT * FROM Suppliers", objConn
' Lösche Datei, wenn sie bereits existiert
if objFO.FileExists(strFile) then
   objFO.DeleteFile strFile, TRUE
end if
objRS.Save strFile, 1' adPersistXML
set objTO = objFO.OpenTextFile(strFile, 1)
show_xml(objTO)
objTO.Close
```

Listing 3.19: RecordSet.Save.asp: Erzeugen einer XML-Datei

Abbildung 3.5: Die XML-Version der Tabelle Suppliers

- `Stream`, im Abschnitt 3.7, Seite 141
- `SaveToFile`, im Abschnitt 3.7, Seite 147
- `Open`, Seite 86

Seek

Diese Methode sucht in der Datenbank. Der Vorgang ähnelt `Find`, die Ausführung erfolgt jedoch durch den OLEDB-Provider und nicht durch die Datenbank oder ADO.

objRS.Seek array *Index*, int *SeekOption*

Index ist ein Array von Feldwerten, deren Elemente den Spalten des Index entsprechen. *SeekOption* entnehmen Sie der folgenden Tabelle:

Parameter

Konstante	Beschreibung
adSeekFirstEQ	Ermittelt den ersten Schlüssel.
adSeekLastEQ	Ermittelt den letzten Schlüssel.
adSeekAfterEQ	Sucht den ersten Schlüssel. Wird dieser nicht gefunden, wird der Schlüssel zurückgegeben, der der erwarteten Position folgt.
adSeekBeforeEQ	Sucht den ersten Schlüssel. Wird dieser nicht gefunden, wird der Schlüssel zurückgegeben, der vor der erwarteten Position liegt.
adSeekBefore	Ermittelt den Schlüssel vor dem lokalisierten Schlüssel.

Der einzige Provider, der derzeit diese Methode unterstützt, ist der Jet-Provider für Access 2000. Mit SQL Server können Sie `Seek` nicht verwenden. Das folgende Beispiel zeigt dies und nutzt die deutsche Version der NorthWind-Datenbank, NORDWIND.MDB.

```
strConn = "Provider=Microsoft.Jet.OLEDB.4.0;
          Data Source=d:\w2k\programme\microsoft
                       office\office\samples\nordwind.mdb"
set objRS = Server.CreateObject("ADODB.RecordSet")
objRS.Open "Artikel", strConn, adOpenStatic,
                      adLockReadOnly, adCmdTableDirect
if (len(Request.Form("Filter")) > 0) then
   strFind = trim(Request.Form("Search"))
   objRS.Index = "ArtikelName"
   objRS.MoveFirst
   objRS.Seek Array(strFind)
end if
if objRS.EOF then
   echo "Name wurde nicht gefunden<br>"
else
   echo "Name gefunden; Ausgabe der Liste ab Fundstelle:<br>"
   while not objRS.EOF
      echo objRS("ArtikelName") & "<br>"
```

```
      objRS.MoveNext
   wend
end if
```

Listing 3.20: RecordSet.Seek.asp: Nutzung der Methoden Index und Seek

Das Array enthält die Suchwörter für jeden Index der Tabelle, genau in der definierten Reihenfolge. Im Beispiel wird nur ein Index auf die Spalte *Artikelname* verwendet.

Supports

Supports

Gibt `TRUE` zurück, wenn der Datensatz eine der folgenden Eigenschaften unterstützt.

```
boolean blnResult = objRS.Supports(option)
```

Als *option* wird einer der folgenden Werte angegeben. Unterstützt der Datensatz die betreffende Eigenschaft, wird *blnResult* `TRUE`. Die Konstanten für den Parameter können Sie der folgenden Tabelle entnehmen:

Parameter

Konstante	Beschreibung
`adAddNew`	Datensatzzeiger unterstützt die Methode `AddNew`.
`adApproxPosition`	Datensatzzeiger unterstützt die Methoden `AbsolutePosition` und `AbsolutePage`.
`adBookmark`	Datensatzzeiger unterstützt die `Bookmark`-Eigenschaft (Lesezeichen).
`adDelete`	Datensatzzeiger unterstützt die Methode `Delete`.
`adHoldRecords`	Die Eigenschaft zeigt, dass beim Lesen weiterer Datensätze bereits erfolgte Änderungen zurückgeschrieben werden.
`adMovePrevious`	Datensatzzeiger unterstützt die Methode `MovePrevious`.
`adResync`	Datensatzzeiger unterstützt die Methode `Resync`.
`adUpdate`	Datensatzzeiger unterstützt die Methode `Update`.
`adUpdateBatch`	Datensatzzeiger unterstützt die Methode `Update` in Stapeldateien.
`adNotify`	Zeigt an, dass der Datensatz Nachrichten erkennt und Ereignisse produziert. Dies ist für ASP nicht zutreffend, da Ereignisse nicht unterstützt werden.

Das folgende Beispiel zeigt die Anwendung der Methode:

```
set objRS = Server.CreateObject("ADODB.RecordSet")
objRS.Open "SELECT * FROM Products", objConn,
           Request.Form("type")
if objRS.Supports(adAddNew) then
   echo "Datensatzzeiger unterstützt die Methode
         <b>AddNew</b>.<br>"
if objRS.Supports(adApproxPosition) then
   echo "Datensatzzeiger unterstützt die Methoden
         <b>AbsolutePosition</b> und <b>AbsolutePage</b>.<br>"
if objRS.Supports(adBookmark) then
   echo "Datensatzzeiger unterstützt die <b>Bookmark</b>-
         Eigenschaft (Lesezeichen).<br>"
if objRS.Supports(adDelete) then
   echo "Datensatzzeiger unterstützt die Methode
         <b>Delete</b>.<br>"
if objRS.Supports(adHoldRecords) then
   echo "Die Eigenschaft zeigt, dass beim Lesen weiterer
         Datensätze bereits erfolgte Änderungen
         zurückgeschrieben werden.<br>"
if objRS.Supports(adMovePrevious) then
    echo "Datensatzzeiger unterstützt die Methode
          <b>MovePrevious</b>.<br>"
if objRS.Supports(adResync) then
   echo "Datensatzzeiger unterstützt die Methode
         <b>Resync</b>.<br>"
if objRS.Supports(adUpdate) then
   echo "Datensatzzeiger unterstützt die Methode
         <b>Update</b>.<br>"
if objRS.Supports(adUpdateBatch) then
   echo "Datensatzzeiger unterstützt die Methode
         <b>Update</b> in Stapeldateien.<br>"
```

Listing 3.21: RecordSet.Supports.asp: Anzeige der Eigenschaften in Abhängigkeit vom Zeigertyp des Datensatzes

Bei der Auswertung der Eigenschaft `adUpdate` ist zu beachten, dass `Support` hier nur feststellt, ob überhaupt Spalten geändert werden dürfen. Wenn Sie mit Sichten (Views) arbeiten und einige Spalten geändert werden dürfen, ist die Eigenschaft `TRUE`. Dennoch kann es beim Zugriff auf gesperrte Spalten zu einem Laufzeitfehler kommen.

Update

Update

Sichert alle neu hinzugefügten oder geänderten Datensätze im Datensatzobjekt in die korrespondierende Tabelle.

objRS.Update([*fields*], [*values*])

fields ist der Name des oder der Felder, die geändert werden. Mehrere Felder lassen sich durch ein Array angegeben. *values* sind die neuen Werte der Felder. Wenn mehrere Felder geändert werden, muss ein Array verwendet werden.

Die Update-Methode wird aufgerufen, nachdem Werte eines änderbaren Datensatzes beschrieben wurden:

```
objRS("vorname").Value  = "Jörg"
objRS("nachname").Value = "Krause"
objRS.Update
```

Außerdem ist die Angabe von Feldname und Wert auch mit der folgenden Syntax möglich:

```
set objRS = Server.CreateObject("ADODB.RecordSet")
strQuery = "SELECT * FROM Products"
objRS.Open strQuery, objConn, adOpenStatic, adLockOptimistic
call show_records(objRS, 10)
objRS.MoveFirst
while not objRS.EOF
   objRS.Update "UnitPrice", objRS("UnitPrice")
   objRS("UnitPrice").Value = objRS("UnitPrice") / 1.10
   objRS.MoveNext
wend
echo "<hr>"
objRS.MoveFirst
call show_records(objRS, 10)
```

Listing 3.22: RecordSet.Update.asp: Anwenden der Update-Methode. Die Prozedur show_records zeigt eine Anzahl Datensätze an.

Wenn viele Werte geändert werden müssen, können Arrays eingesetzt werden:

```
arrFields = Array("id", "name", "strasse", "stadt")
arrValues = Array(1, "Jörg Krause", "Planufer", "Berlin")
objRS.Update arrFields, arrValues
```

Der Umweg über Variablen ist natürlich optional:

```
objRS.Update Array("vorname", "name"),
```

```
            Array("Clemens", "Krause")
```

- UpdateBatch, Seite 95
- Open, Seite 86

UpdateBatch

UpdateBatch

Im Batchmode werden mit dieser Methode alle Datensätze gesichert, die neu sind oder verändert wurden.

objRS.UpdateBatch([*affected*])

Parameter

Der Parameter *affected* kann sein:

Konstante	Beschreibung
adAffectCurrent	Änderungen erfolgen nur für den aktuellen Datensatz.
adAffectGroup	Änderungen erfolgen für die Datensätze, die von der Filter-Eigenschaft selektiert wurden.
adAffectAll	Änderungen werden für alle Datensätze durchgeführt (Standard).

Der Batchmode wird aktiviert, indem ein Datensatzobjekt mit der Eigenschaft adLockBatchOptimistic erzeugt wird. Diese Methode steigert unter Umständen die Performance.

- Open, Seite 86
- Update, Seite 94

Beispiel

```
set objRS = Server.CreateObject("ADODB.RecordSet")
strQuery = "SELECT * FROM Products"
objRS.Open strQuery, objConn, adOpenStatic, adLockOptimistic
call show_records(objRS, 10)
objRS.MoveFirst
while not objRS.EOF
   objRS("UnitPrice").Value = objRS("UnitPrice") * 1.10
   objRS.MoveNext
wend
echo "<hr>"
objRS.UpdateBatch
objRS.MoveFirst
call show_records(objRS, 10)
```

Listing 3.23: RecordSet.UpdateBatch.asp: Updaten mit der Batch-Methode

3.2.4 Eigenschaften

Die Anwendung der Eigenschaften setzt voraus, dass ein `RecordSet`-Objekt existiert. Sie können aber einige Eigenschaften auch dann schon zuweisen, wenn noch keine Daten abgerufen wurden.

AbsolutePage

AbsolutePage

Nummer der aktuellen Seite bei seitenweiser Ausgabe. Die erste Seite hat den Wert 1.

```
long lngPosition = objRS.AbsolutePage
objRS.AbsolutePage = long lngPosition
```

Folgende spezielle Werte kann der Eigenschaft *lngPosition* zugewiesen, alle anderen Werte werden als Seitenzahl interpretiert. Die Parameter können Sie der folgenden Tabelle entnehmen:

Parameter

Konstante	Beschreibung
adPosUnknown	Aktueller Datensatz ist leer oder die Seitennummer ist unbekannt oder der Datensatz unterstützt Seiten nicht.
adPosBOF	Die Eigenschaft BOF ist TRUE.
adPosEOF	Die Eigenschaft EOF ist TRUE.

Nicht alle Provider unterstützen diese Eigenschaft. Sie können die Methode `Supports` mit Konstanten `adApproxPosition` prüfen, um die Unterstützung festzustellen.

- `PageSize`, Seite 108
- `PageCount`, Seite 107

AbsolutePosition

AbsolutePosition

Ergibt die absolute Position des Datensatzzeigers. Wenn die Eigenschaft geschrieben wird, setzt dies den Zeiger auf die neue absolute Position.

```
long lngPosition = objRS.AbsolutePosition
objRS.AbsolutePosition = long lngPostion
```

Nutzen Sie folgende Konstanten:

Parameter

Konstante	Beschreibung
adPosUnknown	Aktueller Datensatz ist leer oder die Seitennummer ist unbekannt oder der Datensatz unterstützt Seiten nicht.

Konstante	Beschreibung
adPosBOF	Die Eigenschaft BOF ist TRUE.
adPosEOF	Die Eigenschaft EOF ist TRUE.

ActiveCommand

ActiveCommand

Wenn das Datensatzobjekt durch Command erzeugt wurde, erlaubt diese Eigenschaft die Erzeugung eines daraus abgeleiteten Kommandoobjekts.

```
Set objCommand = objRS.ActiveCommand
```

Die Eigenschaft ist NULL, wenn das Datensatzobjekt nicht durch ein Kommandoobjekt erzeugt wurde. Der Zugriff kann aber auch direkt auf Eigenschaften des Command-Objekts erfolgen, sodass Sie praktisch kein Command-Objekt erzeugen müssen, wie das folgende Beispiel zeigt:

```
set objRS = Server.CreateObject("ADODB.RecordSet")
strQuery = "SELECT * FROM Products WHERE ProductID > 0"
objRS.Open strQuery, objConn, adOpenStatic, adLockOptimistic
echo "Folgendes Kommando wurde ausgef&uuml;hrt:<br>"
echo "<pre>"
echo objRS.ActiveCommand.CommandText
echo "</pre>"
```

Listing 3.24: RecordSet.ActiveCommand.asp: Aktuelles Kommando anzeigen

▶ Command, Seite 122

BOF

BOF

Ist TRUE, wenn sich der Datensatzzeiger noch vor dem ersten Datensatz am Dateianfang befindet (BOF = *Begin of File).*

```
boolean blnBOF = objRS.BOF
```

Wenn Sie feststellen möchten, ob überhaupt Datensätze zurückgegeben wurden und die Eigenschaft RecordCount nicht verwendet werden kann, prüfen Sie sowohl EOF als auch BOF auf TRUE:

```
if objRS.EOF and objRS.BOF then
   Response.Write "Abfrage lieferte keine Daten.<br>"
end if
```

▶ EOF, Seite 101

Bookmark

Bookmark

Diese Eigenschaft gibt das Lesezeichen für den aktuellen Datensatz zurück. Wenn diese Eigenschaft mit einem Lesezeichen gesetzt wird, kann der aktuelle Datensatz mit diesem Lesezeichen identifiziert werden.

```
long lngBkmrk = objRS.Bookmark
objRS.Bookmark = long lngBkmrk
```

Lesezeichen zwischen verschiedenen Datensätzen sind nicht austauschbar. Die einzige Ausnahme bilden geklonte Datensätze.

Sie können mit dieser Eigenschaft eine bestimmte Position des Datensatzzeigers speichern, die sich bei anderen Operationen verschiebt, wie z.B. bei `Move` oder `Find`:

```
set objRS = Server.CreateObject("ADODB.RecordSet")
strQuery = "SELECT * FROM Products WHERE ProductID > 0"
objRS.Open strQuery, objConn, adOpenStatic, adLockOptimistic
objRS.MoveFirst
echo "Erster: " & objRS("ProductName") & "<br>"
intBookmark = objRS.Bookmark
objRS.MoveLast
echo "Letzter: " & objRS("ProductName") & "<br>"
objRS.Bookmark = intBookmark
echo "Gemerkt: " & objRS("ProductName") & "<br>"
```

Listing 3.25: RecordsSet.Bookmark.asp: Positionen mit Lesezeichen merken

Lesezeichen werden von verschiedenen Providern unterschiedlich implementiert. Ein Skript, das mit absoluten Zahlen arbeitet und mit Access läuft, muss mit SQL Server nicht funktionieren. Sie sollten Lesezeichen nie absolut betrachten, sondern die Werte nur als relative Verweise auf Datensätze ansehen. Auch beim erneuten Abruf der Daten kann die innere Struktur des Datensatzes variieren – entsprechend variiert auch die Liste der Lesezeichen. Sie sollten deshalb nie Lesezeichen einsetzen, wenn über die Grenzen eines Skripts hinaus operiert wird. Dies gilt auch, wenn sich ein Skript selbst aufruft.

- `Clone`, Seite 75
- `Find`, Seite 78
- `Move`, Seite 83

CacheSize

CacheSize

Gibt an, wie viele Datensätze des aktuellen Datensatzobjekts im lokalen Speicher (RAM) gehalten werden.

```
long lngRecords = objRS.CacheSize
objRS.CacheSize = long lngRecords
```

Der Standardwert ist 1. Wenn die Anzahl der nachfolgend bearbeiteten Datensätze bekannt ist, kann der Einsatz die Performance steigern, da mehr Datensätze im lokalen Speicher gehalten werden. Dadurch steigt aber auch der Speicherverbrauch. Die Änderung des Wertes wirkt sich erst aus, wenn Daten aus der Datenquelle gelesen werden. Die Änderung an einem bestehenden Datensatz hat keine Wirkung. Sinnvoll ist der Einsatz vor dem Öffnen des Datensatzes.

Ein größerer Cache kann sich auf verschiedene Methoden auswirken. Änderungen an Datensätzen haben keinen Einfluss auf den Cache. Um den Cache zu aktualisieren, müssen Sie `Resync` verwenden. Das führt dazu, dass die Methoden `Move`, `MoveNext`, `MovePrevious` und `MoveFirst`, `MoveLast` auf einen gelöschten Datensatz zeigen, wenn die Ausführung im Cache gelingt und der Zieldatensatz zugleich im übergeordneten Datensatz gelöscht wurde. Es ist deshalb notwendig, zuvor `Resync` aufzurufen.

- `Resync`, Seite 88
- `Move`, Seite 83

CursorLocation

CursorLocation

Diese Eigenschaft bestimmt, wo der Datensatzzeiger gespeichert wird.

```
integer intPos = objRS.CursorLocation
objRS.CursorLocation = integer intPos
```

Die möglichen Parameter des Datenbankzeigers entnehmen Sie der folgenden Tabelle:

Parameter

Konstante	Beschreibung
`adUseClient`	Clientseitiger Zeiger
`adUseServer`	Serverseitiger Zeiger oder vom Treiber

Bei parallelen (konkurrierenden) Zugriffen sind serverseitige Zeiger effizienter, soweit sie vom Provider unterstützt werden. Wenn Sie dagegen Datensätze abkoppeln (z. B. mit `Clone`), müssen Sie clientseitige Zeiger verwenden, die dann auch unabhängig von einer direkten Verbindung zur Datenbank existieren können.

Wenn Sie `CursorLocation` nicht definieren, wird der Wert der Eigenschaft `CursorLocation` des Objekts `Connection` verwendet. Wenn der Wert dieser Eigenschaft jedoch überschrieben wird, erfolgt keine Synchronisation in `Connection`. Andere abgeleitete `RecordSet`-Objekte erben dann wieder den ursprünglichen Wert.

Zu Zeigern finden Sie mehr Informationen in Abschnitt 5.2 Datenbankzeiger ab Seite 197.

CursorType

CursorType

Diese Eigenschaft bestimmt, welchen Typ der Datensatzzeiger hat.

```
integer intPos = objRS.CursorType
objRS.CursorType = integer intPos
```

Der Zeiger kann eine der folgenden Eigenschaften haben:

Parameter

Konstante	Beschreibung
`adOpenForwardOnly`	Zeiger kann sich nur vorwärts in der Tabelle bewegen (Standard).
`adOpenKeyset`	Zeiger reagiert auf Änderungen und Löschungen von Datensätzen durch andere Nutzer, Einfügungen neuer Datensätze werden nicht übertragen.
`adOpenDynamic`	Zeiger überträgt alle Einflüsse anderer Nutzer.
`adOpenStatic`	Zeiger reagiert auf keinerlei Einflüsse anderer Nutzer.

Einige Provider unterstützen nicht alle Zeigertypen. Sie können dies prüfen, indem die Eigenschaft gesetzt und nach dem Lesen der Datensätze überprüft wird, ob sich die Eigenschaft verändert hat:

```
set objRS = Server.CreateObject("ADODB.RecordSet")
strQuery = "SELECT * FROM Products WHERE ProductID > 0"
' Standardzeiger (adOpenForwardOnly)
objRS.Open strQuery, objConn
echo "adOpenForwardOnly:<br>"
check_AddNew(objRS)
objRS.Close
' anderer Zeiger
objRS.CursorType = adOpenStatic
objRS.Open strQuery, objConn
echo "adOpenStatic:<br>"
check_AddNew(objRS)
```

Listing 3.26: RecordSet.CursorType.asp: Einstellung des Zeigertyps

Zu Zeigern finden Sie mehr Informationen in Abschnitt 5.2 Datenbankzeiger ab Seite 197.

▶ Open, Seite 86

EditMode

EditMode

Gibt eine Konstante zurück, die den Editiermodus repräsentiert. Die Eigenschaft kann nur gelesen werden.

Parameter

```
long lngEdit = objRS.EditMode
```

Der Rückgabewert kann der folgenden Tabelle entnommen werden:

Konstante	Beschreibung
adEditNone	Keine Änderungen im Moment
adEditInProgress	Der aktuelle Datensatz wurde geändert, ist aber noch nicht gespeichert.
adEditAdd	Die Methode AddNew wurde aufgerufen.

Wenn Sie eine Applikation haben, die es Nutzern erlaubt, Datensätze zu verändern und zugleich zu blättern, so wird normalerweise bei jeder Bewegung des Datensatzzeigers implizit die Update-Methode aufgerufen.

▶ Update, Seite 94

▶ CancelUpdate, Seite 75

EOF

EOF

Wird TRUE, wenn das Ende der Tabelle erreicht wurde. Der Zeiger steht hinter dem letzten Datensatz.

```
blnEOF = objRS.EOF
```

Die Anwendung finden Sie in nahezu jedem Listing. Typisch ist die Nutzung als Abbruchbedingung in einer while-Schleife:

```
while not objRS.EOF
   '... Aktionen
wend
```

Beachten Sie, dass der Zustand EOF = TRUE zum Abbruch führen soll, die while-Schleife dagegen solange durchlaufen wird, wie die Bedingung FALSE ist. Deshalb wird der Wert mit not negiert.

▶ BOF, Seite 97

Filter

Filter

Gibt einen Filter an. Der Filter kann eine Auswahlbedingung oder eine Liste von Lesezeichen sein.

```
string strFilter = objRS.Filter
objRs.Filter = long lngBkmrk
objRS.Filter = string strFilter
```

Der Filter kann eine Zeichenkette mit SQL-Filterzeichen und Platzhalterzeichen sein, ein Array mit Lesezeichen oder eine der folgenden Konstanten:

Parameter

Konstante	Beschreibung
adFilterNone	Entfernt den aktuellen Filter.
adFilterPendingRecords	Filtert im Batchmode die Datensätze, die geändert, aber noch nicht an den Server gesendet wurden.
adFilterAffectedRecords	Filtert die Datensätze, die von der letzten Delete-, Resync-, UpdateBatch- oder CancelBatch-Methode betroffen waren.
adFilterFetchedRecords	Filtert den letzten Datensatz (aktueller Datensatz).
adFilterPredictate	Zeigt zum Löschen markierte Datensätze.
adFilterConflictRecords	Zeigt Datensätze, die beim letzten Batch-Update miteinander in Konflikt stehen.

Als Filtereigenschaft kann eine Auswahlbedingung ähnlich wie WHERE in SQL dienen. Im Gegensatz zu Find sind auch die logischen Operatoren And, Or oder Not erlaubt.

```
set objRS = Server.CreateObject("ADODB.RecordSet")
objRS.Open "SELECT * FROM Products", objConn, adOpenStatic
echo "<b>Hinweise:</b>"
echo "<ul><li>Sie k&ouml;nnen logische Operatoren wie
      <code>and</code> oder <code>or</code> verwenden.<br>"
echo "<li>Folgende Feldnamen sind zul&auml;ssig:<br><pre>"
for each fname in objRS.Fields
    echo fname.name & ", "
next
echo "</pre>"
echo "<li>Setzen Sie Zeichenketten in einfache
      Anf&uuml;hrungszeichen, z.B. ProductName = 'Chai'</ul>"
echo "<hr noshade size=2>"
if (len(Request.Form("Filter")) > 0) then
    on error resume next
```

```
    strFilter = trim(Request.Form("search"))
    objRS.Filter = strFilter
    if Err.Number > 0 then
       echo "<div style=""color:red""><b>Fehler:</b>"
       echo "Der Ausdruck war nicht korrekt ==> "
             & Err.Description
       echo "</div>"
    end if
    on error goto 0
else
    objRS.Filter = adFilterNone
end if
if objRS.EOF then
    echo "Mit diesen Kriterien wurde kein Eintrag
          gefunden<br>"
else
    while not objRS.EOF
       echo objRS("ProductName") & "<br>"
       objRS.MoveNext
    wend
end if
```

Listing 3.27: RecordSet.Filter.asp: Anwendung der Filter-Eigenschaft. Der Inhalt des Filters wird aus einem Formularfeld mit dem Namen Search übergeben.

Um gezielt einige Datensätze auszuwählen, können Sie ein Array anlegen und dort Lesezeichen speichern. Dieses Array kann dann als Filter dienen:

```
Dim arrBkmrk(10)
arrBkmrk(0) = objRS.Bookmark
' verschiedene Operationen
arrBkmrk(1) = objRS.Bookmark
' verschiedene Operationen
objRS.Filter = arrBkmrk
```

Zur Angabe der Bedingungen beachten Sie, dass als Platzhalterzeichen * und % eingesetzt werden können. Außerdem können Datumsliterale verwendet werden, wie es in VBScript erlaubt ist:

```
objRS.Filter = "datum > #26.05.2000#"
```

Falls Spaltennamen Leerzeichen enthalten, setzen Sie diese in eckige Klammern:

```
objRS.Filter = "[Product Name] = 'Chai'"
```

Einfache Anführungszeichen werden angegeben, indem sie verdoppelt werden:

```
objRS.Filter = "Name = 'Ecki''s Laden'"
```

Das folgende Beispiel zeigt eine komplexere Applikation, mit der eine wiederholende Auswahl erfolgt.

```
set objRS = Server.CreateObject("ADODB.RecordSet")
objRS.Open "SELECT * FROM Products", objConn, adOpenStatic
if (len(Request.Form("Filter")) > 0) then
   arrSelection = split(Request.Form("Selection"), ",")
   strSelection = ""
   for each e in arrSelection
     strSelection = strSelection & "ProductID = " & e & " OR "
   next
   strSelection = strSelection & "ProductID < 0"
   objRS.Filter = strSelection
else
   objRS.Filter = adFilterNone
end if
if objRS.EOF then
   echo "Mit diesen Kriterien wurde kein Eintrag gefunden<br>"
else
   s = 1
   echo "<table><tr>"
   while not objRS.EOF
      echo "<td><input type=""Checkbox"" name=""Selection""
            value=""" & objRS("ProductID") & """></td>"
      echo "<td>" & objRS("ProductName") & "</td>"
      objRS.MoveNext
      if s mod 4 = 0 then
         echo "<tr></tr>"
      end if
      s = s + 1
   wend
   echo "</table>"
end if
```

Listing 3.28: RecordSet.Filter.2.asp: Auswahl von Datensätzen mit Kontrollkästchen und Anwendung der Filter-Eigenschaft

- `Update`, Seite 94
- `Bookmark`, Seite 98
- `Find`, Seite 78

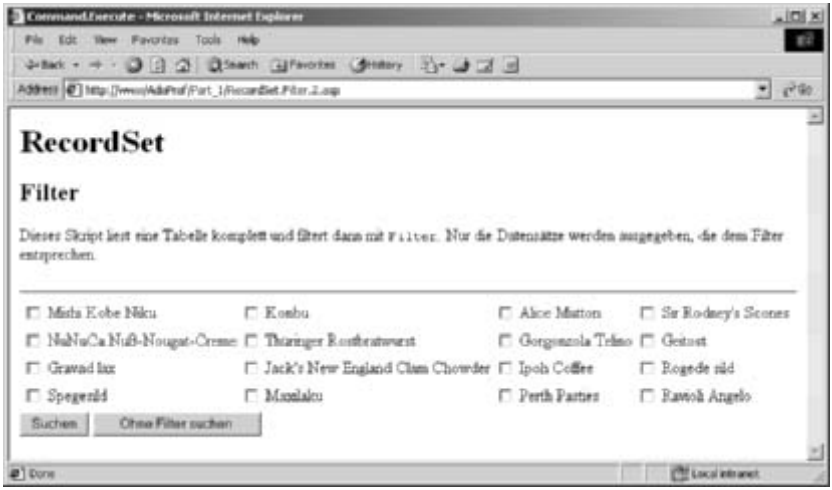


Abbildung 3.6: Auswahl aus der Tabelle Products (Northwind-Datenbank), mit Hilfe des Skripts in Listing 3.28

Index

Index

Die `Index`-Eigenschaft wird nur vom OLEDB-Jet.4.0-Treiber und damit nur von Access-Datenbanken unterstützt.

```
objRS.Index(string fieldname)
```

Um diese Eigenschaft nutzen zu können, müssen Sie die Eigenschaft `CommandType` auf `adCmdTable` setzen. Die `Supports`-Methode muss außerdem `adIndex` akzeptieren.

Die Anwendung ist in der Regel zusammen mit `Seek` sinnvoll, eine Methode, die auch nur von dem erwähnten Provider unterstützt wird. Sie finden bei `Seek` ein entsprechendes Beispiel.

▶ `Seek`, Seite 91

LockType

LockType

Art der verwendeten Verriegelung, wenn der Datensatz geöffnet wird.

```
integer intType = objRS.LockType
objRS.LockType = integer intType
```

Parameter

Sie können eine der folgenden Konstanten zuweisen:

Konstante	Beschreibung
`adLockReadOnly`	Daten können nicht geändert werden (Standard).
`adLockPessimistic`	Der Server verriegelt den Datensatz, wenn mit der Änderung begonnen wird.

Konstante	Beschreibung
adLockOptimistic	Die Verriegelung erfolgt nur, wenn ein Update-Kommando ausgeführt wird.
adLockBatchOptimistic	Die Verriegelung erfolgt nur, wenn ein Update-Kommando im Batchmodus ausgeführt wird.

Die Option adLockPessimistic kann nicht verwendet werden, wenn Cursor clientseitig existieren. Auch in allen anderen Fällen ist diese Methode zwar die in Bezug auf die Datenkonsistenz sicherste, praktisch ist sie jedoch kritisch in der Anwendung. Um den Zustand zu erhalten, muss zwischen Programm und SQL Server eine permanente Verbindung bestehen. Das ist ineffizient bzw. bei manchen Systemen unmöglich. Die Methode adLockOptimistic ist effizienter, führt aber in manchen Fällen zu inkonsistenten Zuständen. Wenn zwei Nutzer einen Datensatz öffnen und verändern, kann nur ein Datensatz zurückgeschrieben werden. Dies führt zu einem Laufzeitfehler, wenn der zweite Datensatz ebenfalls aktualisiert wird. Sie müssen diesen Fehler abfangen und durch eigene Routinen verarbeiten.

MarshalOptions

MarshalOptions

Gibt die Ordnung an, in der eine Gruppe Datensätze zurückgeschrieben wird.

```
integer intMarshal = objRS.MarshalOptions
objRS. MarshalOptions = integer intMarshal
```

Die Parameter können Sie der folgenden Tabelle entnehmen:

Parameter

Konstante	Beschreibung
adMarshalAll	Alle Datensätze werden zum Server gesendet (Standard).
adMarshalModifiedOnly	Nur die geänderten Datensätze werden zum Server zurückgesendet.

Diese Eigenschaft ist nur anwendbar, wenn clientseitige Datensätze verwendet werden. Die Eigenschaft adMarshalModifiedOnly kann die Leistung deutlich verbessern, weil die zu übertragene Datenmenge reduziert wird. Der Einsatz ist vor allem da zu sehen, wo Datensatzobjekte über mehrere Maschinen hinweg an andere Prozesse gesendet werden. Sie verringern mit der Option adMarshalModifiedOnly die erforderliche Bandbreite. Solche verteilten Applikationen finden aber im Zusammenhang mit ASP kaum Einsatz. Falls Sie selbst ASP-Objekte programmieren, könnte das trotzdem interessant sein.

MaxRecords

MaxRecords

Anzahl der Datensätze, die zurückgegeben werden. Der Standardwert ist 0, damit werden alle Sätze zurückgegeben.

```
long lngMaxRec = objRS.MaxRecords
objRS.MaxRecords = long lngMaxRec
```

Diese Eigenschaft kann nur gesetzt werden, wenn der Datensatz noch nicht geöffnet wurde, also noch keine Daten gelesen wurden. Danach kann die Eigenschaft nur noch gelesen werden.

Sie können diese Eigenschaft einsetzen, um die Anzahl der zu lesenden Datensätze zu beschränken, wenn eine Tabelle sehr groß ist und eine allgemeine Suchanfrage eine große Anzahl Datensätze zurückgeben würde.

Die Implementierung ist relativ unterschiedlich, es ist nicht sicher, dass `MaxRecords` mit allen Datenbankprovidern funktioniert. Die eigentliche Abarbeitung wird nämlich nicht in ADO erfolgen, sondern in der Datenbank. Sie können die Aufgabe aber auch vom Provider erledigen lassen, was zumindest OLE DB unterstützt. Das folgende Beispiel zeigt dies:

```
set objRS = Server.CreateObject("ADODB.RecordSet")
if (len(Request.Form("Filter")) > 0) then
   MaxRecords = "TOP " & Cint(Request.Form("MaxRecords"))
else
   MaxRecords = ""
end if
objRS.Open "SELECT " & MaxRecords & " * FROM Products",
           objConn, adOpenStatic
while not objRS.EOF
   echo objRS("ProductID") & " => "
        & objRS("ProductName") & "<br>"
   objRS.MoveNext
wend
```

Listing 3.29: RecordSet.MaxRecords.asp: Alternative Nutzung des OLEDB-Providers, wenn MaxRecords nicht implementiert ist.

PageCount

PageCount

Ermittelt die Anzahl der Seiten, die entstehen, wenn das Datensatzobjekt in Seiten unterteilt wird.

```
long lngPage = objRS.PageCount
```

Wenn die Eigenschaft `PageSize` nicht gesetzt wurde, wird -1 zurückgegeben. Die Zählung stimmt mit der von `AbsolutePage` überein. So können Sie mit der folgenden Anwendung zur letzten Seite springen:

```
set objRS = Server.CreateObject("ADODB.RecordSet")
PageSize = Request.Form("PageSize")
if len(PageSize) > 0 then
   objRS.PageSize = PageSize
end if
objRS.Open "SELECT * FROM Products", objConn, adOpenStatic
echo "Bei einer Seitengr&ouml;&szlig;e von " & objRS.PageSize
echo " enth&auml;lt die Tabelle " & objRS.PageCount & "
       Seiten."
```

Listing 3.30: RecordSet.PageCount.asp: Anzahl der Seiten bei seitenweiser Ausgabe ermitteln, die Seitengröße selbst wird mit PageSize vorgegeben

Die Möglichkeit, Datensätze in Seiten zu unterteilen, muss vom Provider unterstützt werden. Teilweise werden auch nur einige der Eigenschaften unterstützt.

▶ PageSize

PageSize

PageSize

Größe einer Seite, auf deren Grundlage `PageCount` berechnet wird. Mit dieser Eigenschaft werden lange Datensätze in Seiten unterteilt.

```
long lngPgsz  = objRS.PageSize
objRS.PageSize = long lngPgsz
```

Die Unterteilung von umfangreichen Abfragen erlaubt vor allem eine bessere Navigation. Trotzdem werden alle Daten abgerufen. Die Eigenschaft korrespondiert auch nicht mit dem internen Cache.

Das Unterteilen der Datensätze in Seiten muss vom Provider unterstützt werden. Teilweise werden auch nur einige der Eigenschaften unterstützt.

Ein Beispiel finden Sie in Listing 3.30.

▶ PageCount, Seite 107

RecordCount

RecordCount

Anzahl der Datensätze des Datensatzobjekts. Gibt den Wert -1 zurück, wenn die Anzahl nicht festgestellt werden kann.

```
long lngRecords = objRS.RecordCount
```

Diese Eigenschaft kann nur dann die korrekte Anzahl Datensätze liefern, wenn der Provider synchron abgefragt wurde. Um bei einer asynchronen Abfrage ein korrektes Ergebnis zu erzielen, ist es empfehlendwert, zuvor die Methode `MoveLast` aufzurufen. Der Provider wird dadurch gezwungen,

alle Datensätze der Abfrage auszuliefern und kann auch die Anzahl berechnen.

```
set objRS = Server.CreateObject("ADODB.RecordSet")
strSQL = "SELECT * FROM Products"
objRS.Open strSQL, objConn, adOpenStatic
if objRS.Supports(adBookmark) then
   echo "Diese Tabelle enth&auml;lt <b>" & objRS.RecordCount
         & "</b>"
   echo " Datens&auml;tze. Abfrage:<br>"
   echo "<pre>" & strSQL & "</pre>"
else
   echo "Anzahl konnte nicht ermittelt werden."
end if
```

Listing 3.31: RecordSet.RecordCount.asp: Anzahl der Datensätze ermitteln

Sort

Diese Eigenschaft bestimmt eine oder mehrere Spalten, nach denen sortiert wird.

Sort

```
string strSort = objRS.Sort
objRS.Sort = string strSort
```

Die Zeichenkette *strSort* enthält eine durch Komma separierte Liste mit Spaltennamen, die jeweils von den Schlüsseln `ASC` (*ascending*, aufsteigend) oder `DESC` (*descending*, absteigend) begleitet werden.

```
objRS.Sort = "name ASC, title DESC"
```

Das Sortieren erfolgt im Datensatzobjekt, nicht in der Datenbank. Es ist bei einem umfangreichen Datensatz sinnvoll, mit lokalen Indizes zu arbeiten. Dies wird durch die Eigenschaft `Optimize` erreicht:

```
objRS("titel").Properties("Optimize") = TRUE
```

Diese Funktionalität wird von ADO implementiert, die Sortiwerung erfolgt also nicht in der Datenbank. Sie müssen daher einen clientseitigen Datensatz verwenden:

```
objRS.CursorLocation = adUseClient
```

Das folgende Beispiel zeigt, wie Sie eine Tabelle universell sortieren:

```
set objRS = Server.CreateObject("ADODB.RecordSet")
objRS.CursorLocation = adUseClient
objRS.Open "SELECT * FROM Products", objConn, adOpenStatic
```

```
objRS.Sort = Request.QueryString("sort") & " "
             & Request.QueryString("dir")
echo "<table border=1>"
for each e in objRS.Fields
   echo "<th>"
   echo "<a href=""" & ASP_SELF & "?sort=" & e.name &
        "&dir=desc"">[+]</a><br>"
echo e.name & "<br>"
echo "<a href=""" & ASP_SELF & "?sort=" & e.name &
     "&dir=asc"">[-]</a>"
echo "</th>"
next
while not objRS.EOF
   echo "<tr>"
   for each e in objRS.Fields
      echo "<td>" & e & "</td>"
   next
   objRS.MoveNext
echo "</tr>"
wend
echo "</table>"
```

Listing 3.32: RecordSet.Sort.asp: Universelle Sortierfunktion

Abbildung 3.7: Listing 3.32 in Aktion: Universelle Sortierung

RecordSet

Sort

Sie können mit Hilfe der Schalter in der Kopfzeile die Sortierung der Tabelle verändern

[+] ProductID [-]	[+] ProductName [-]	[+] SupplierID [-]	[+] CategoryID [-]	[+] QuantityPerUnit [-]	[+] UnitPrice [-]	[+] UnitsInStock [-]	[+] UnitsOnOrder [-]	[+] ReorderLevel [-]	[+] Discontinued [-]
47	Zaanse koeken	22	3	10 - 4 oz boxes	10,45	36	0	5	Falsch
64	Wimmers gute Semmelknödel	12	5	20 bags x 4 pieces	33,875	22	80	30	Falsch
63	Vegie-spread	7	2	15 - 625 g jars	43,29	24	0	5	Falsch
60	Valkoinen suklaa	23	3	12 - 100 g bars	17,075	65	0	30	Falsch
7	Uncle Bob's Organic Dried Pears	3	7	12 - 1 lb pkgs.	33	15	0	10	Falsch
23	Tunnbröd	9	5	12 - 250 g pkgs.	9,9	61	0	25	Falsch
54	Tourtière	25	6	16 pies	8,195	21	0	10	Falsch
14	Tofu	6	7	40 - 100 g pkgs.	23,25	35	0	5	Falsch
29	Thüringer Rostbratwurst	12	6	50 bags x 30 sausgs.	123,79	0	0	0	Wahr
19	Teatime Chocolate Biscuits	8	3	10 boxes x 12 pieces	9,2	25	0	5	Falsch
62	Tarte au sucre	29	3	48 pies	49,3	17	0	0	Falsch

▶ Properties, Abschnitt 4.2, Seite 164

Source

Source

Name des Command-Objekts, einer SQL-Anweisung, einer Tabelle oder einer SQL-Prozedur.

```
string strSource = objRS.Source
objRS.Source = string strSource
Set objRS.Source = varSource
```

Wenn Sie als Kommando adCmdTable verwenden und nur den Tabellennamen übermitteln, wird die komplette SQL-Anweisung zurückgegeben, also: »SELECT * FROM tabelle«.

Werden Abfragen mit Parametern eingesetzt, wird die ursprüngliche Abfrage zurückgegeben, also mit Platzhaltern für die Parameter.

State

State

Status des Datensatzobjekts; kann nur gelesen werden.

```
integer intState = objRS.State
```

Die Eigenschaft kann folgende Werte annehmen:

Parameter

Konstante	Beschreibung
adStateClosed	Das Objekt ist geschlossen.
adStateOpen	Das Objekt ist geöffnet.
adStateConnecting	Nicht anwendbar
adStateExecuting	Die Abfrage wird gerade ausgeführt.
adStateFetching	Die Daten der Abfrage werden gerade übertragen.

Es ist möglich, dass zwei Werte kombiniert auftreten. Wenn Sie asynchrone Abfragen starten und das RecordSet-Objekt bereits geöffnet ist (weil die Abfrage erfolgreich war), aber immer noch Datensätze geholt werden (weil adAsynchFetch erlaubt ist), dann werden adStateOpen und adStateFetching gleichzeitig aktiv sein. Sie können das folgendermaßen testen:

```
intRSState = objRS.State
if intRSState = adStateOpen And adStateFetching then
   … 'Aktionen
end if
```

▶ Open, Seite 86

Status

Status

Status des aktuellen Datensatzes.

```
integer intStatus = objRS.Status
```

Eine oder mehrere Konstanten geben Auskunft über den Zustand:

Parameter

Konstante	Beschreibung
adRecOK	Der Datensatz wurde erfolgreich geändert.
adRecNew	Der Datensatz ist neu.
adRecModified	Der Datensatz wurde geändert.
adRecDeleted	Der Datensatz wurde gelöscht.
adRecUnmodified	Der Datensatz wurde nicht geändert.
adRecInvalid	Der Datensatz wurde nicht gesichert, weil ein Lesezeichen falsch war.
adRecMultipleChanges	Der Datensatz wurde nicht gesichert, weil es Änderungen an mehreren Datensätzen gab.
adRecPendingChanges	Der Datensatz wurde nicht gesichert, weil eine Einfügung Vorrang hatte.
adRecCanceled	Der Datensatz wurde nicht gesichert, weil die Operation abgebrochen wurde.
adRecCantRelease	Der Datensatz wurde nicht gesichert, weil der Datensatz verriegelt war.
adRecConcurrencyViolation	Der Datensatz wurde nicht gesichert, weil ein anderer Nutzer den Datensatz geöffnet hielt.
adRecIntegrityViolation	Der Datensatz wurde nicht gesichert, weil die Integrität der Datenbank verletzt wurde (Recht).
adRecMaxChangesExceeded	Der Datensatz wurde nicht gesichert, weil zu viele Änderungen gleichzeitig zu sichern waren.
adRecObjectOpen	Der Datensatz wurde nicht gesichert, weil es einen Konflikt mit einem Speicherobjekt gab.
adRecOutOfMemory	Der Datensatz wurde nicht gesichert, weil der Computer keinen freien Speicherplatz mehr hatte.
adRecPermissionDenied	Der Datensatz wurde nicht gesichert, weil der Nutzer nicht die erforderlichen Rechte hatte.

Konstante	Beschreibung
adRecSchemaViolation	Der Datensatz wurde nicht gesichert, weil die Struktur der zu Grunde liegenden Datenbank nicht beachtet wurde.
adRecDBDeleted	Der Datensatz wurde bereits gelöscht.

Die Eigenschaftswerte können kombiniert auftreten – sie verhalten sich wie eine Bitmaske. Sie müssen die Werte also auch bitweise auswerten. Ob es sinnvoll ist, die exotischen Werte auszuwerten, sei dahingestellt. Der Einsatz dürfte in der Regel auch nur der Fehlersicherung und Perfektionierung ohnehin funktionierender Skripte dienen.

StayInSynch

In einem hierarchischen Datensatzobjekt bestimmt diese Eigenschaft, ob ein abgeleitetes Objekt (Child) geändert werden soll, wenn das übergeordnete Objekt (Parent) geändert wurde.

StayInSynch

```
blnSynch = objRS.StayInSynch
objRS.StayInSynch = blnSynch
```

Der Standardwert ist TRUE. Mehr Informationen zu hierarchischen Datensätzen finden Sie in Abschnitt 5.1 *Data Shaping* ab Seite 179.

3.2.5 Kollektionen

Kollektionen werden in Kapitel 4 *Kollektionen ab Seite 152* ausführlicher behandelt. Zwei Kollektionen können von Connection abgeleitet werden:

- Properties. Diese Kollektion enthält Property-Objekte mit Eigenschaftsinformationen. Siehe Abschnitt 4.2 Properties ab Seite 164.
- Fields. Diese Kollektion enthält eines oder mehrere Field-Objekte mit Fehlerinformationen. Siehe Abschnitt 4.1 Fields ab Seite 159.

3.3 Record

Das Record-Objekt ist neu seit ADO 2.5 und erleichtert den Zugriff auf einzelne Datensätze. Erzeugt wird das Objekt mit:

```
Set objRC = CreateObject("ADODB.Record")
```

objRC wird in der folgenden Darstellung verwendet, um Methoden und Eigenschaften zu erläutern.

3.3.1 Einführung

Das `Record`-Objekt ist relativ neu und der Einsatz mag nicht sofort transparent erscheinen. Diese Einführung zeigt in kompakter Form Einsatzmöglichkeiten.

Nicht rechteckige Datensätze

`Record` ermöglicht den Aufbau von Datengebilden, die nicht tabellenförmig sind. Ein `RecordSet`-Objekt enthält immer rechteckige Daten – ein Gitter aus Zeilen und Spalten. Nun können bestimmte Datensätze zusätzliche Informationen enthalten, über die andere nicht verfügen. Für jede dieser Zusatzinformationen muss es einen Speicherplatz geben, der an den konkreten Datensatz gebunden ist. Diesen Platz stellt `Record` bereit.

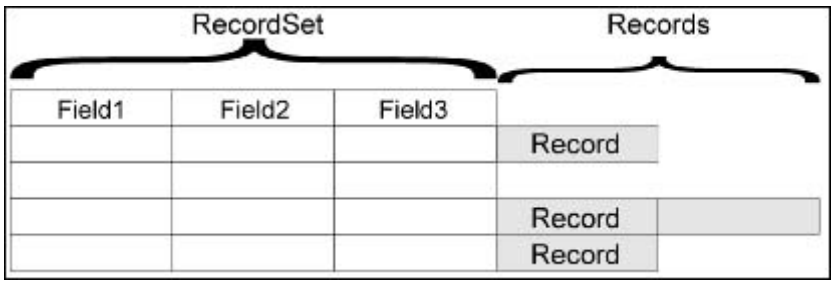


Abbildung 3.8: Möglicher Zusammenhang zwischen RecordSet und Record

Hierarchische Datensätze

Auch Hierarchien lassen sich mit `Record` abgebilden. Es gibt spezielle Methoden, die den Zugriff auf über- oder untergeordnete Objekte der Hierarchie erlauben. In einigen Fällen ist dadurch die realistische Abbildung von Daten besser möglich als mit flachen Tabellen.

Zugriff auf Verzeichnisse

`Record` berücksichtigt nicht nur eine Reihe in einem Datensatzobjekt, sondern auch eine Datei in einem Verzeichnis. Sie können damit auch Daten verwalten, die in hierarchischen Strukturen eines Dateisystems abgelegt sind. Dies ist in einigen Fällen komfortabler als mit `FileSystemObject`.

3.3.2 Übersicht

Methoden

- `Cancel`, Seite 115
- `Close`, Seite 115
- `CopyRecord`, Seite 116
- `DeleteRecord`, Seite 116

- GetChildren, Seite 117
- MoveRecord, Seite 117
- Open, Seite 118

Eigenschaften

- ActiveConnection, Seite 120
- Mode, Seite 120
- ParentURL, Seite 120
- RecordType, Seite 121
- Source, Seite 121
- State, Seite 121

Kollektionen

- Properties, Seite 159
- Fields, Seite 159

3.3.3 Methoden

Cancel

Cancel

Bricht eine Aktion ab, die von einer der folgenden Methoden gestartet wurde: CopyRecord, DeleteRecord, MoveRecord oder Open.

objRC.Cancel

- CopyRecord, Seite 116
- DeleteRecord, Seite 116
- MoveRecord, Seite 117
- Open, Seite 118

Close

Close

Diese Methode schließt das Objekt.

objRC.Close

Ein Laufzeitfehler wird erzeugt, wenn das Objekt nicht zuvor mit Open oder implizit geöffnet wurde.

- Open, Seite 118

CopyRecord

CopyRecord

Kopiert Dateien oder Verzeichnisse.

```
objRC.CopyRecord(Source, Destination,
                 [User], [Pass],
                 Options, Async)
```

Source und *Destination* geben Quelle und Ziel des Kopiervorgangs an. Verwendbar sind Pfadangaben oder URLs. Falls die Datenquelle geschützt ist, kann mit *User* und *Pass* die Identifizierung angegeben werden.

Werte für *Options* können der folgenden Tabelle entnommen werden:

Parameter

Konstante	Beschreibung
adCopyAllowEmulation	Erlaubt die Emulation der Kopie durch Herunterladen und anschließendes Hochladen der Daten, wenn sich diese auf zwei verschiedenen Servern befinden.
adCopyNonRecursive	Kopiert Verzeichnisse, aber nicht deren Unterverzeichnisse.
adCopyOverWrite	Erlaubt Überschreiben existierender Daten im Ziel.
adCopyUnspecified	Überschreiben ist nicht erlaubt und Rekursion wird ausgeführt (Standardwert).

Bei der Anwendung mit dem Parameter adCopyAllowEnumerate kann es bei unterschiedlichen Datenprovidern zum Datenverlust oder zu einem drastischen Performanceeinbruch kommen.

Asynch ist ein logischer Wert, der bestimmt, ob die Operation asynchron ausgeführt werden soll (TRUE) oder nicht (FALSE).

- DeleteRecord, Seite 116
- MoveRecord, Seite 117

DeleteRecord

DeleteRecord

Löscht einen Datensatz oder eine Datei.

```
objRC.DeleteRecord source, asynch
```

source gibt die Quelle des Vorgangs an. *asynch* ist ein logischer Wert, der bestimmt, ob die Operation asynchron ausgeführt werden soll (TRUE) oder nicht (FALSE).

- CopyRecord, Seite 116
- MoveRecord, Seite 117

GetChildren

Diese Methode überführt ein Verzeichnis in ein Datensatzobjekt. Damit können die Methoden und Eigenschaften teilweise auf Dateien und Verzeichnisse angewendet werden.

GetChildren

```
Set objRS = objRC.GetChildren
```

```
strURL = "http://www/"
strFolder = "ADOProf"
set objRC = Server.CreateObject("ADODB.Record")
objRC.Open "http://localhost/"
set objRCH = objRC.GetChildren
While Not objRCH.EOF
   if not objRCH("RESOURCE_ISCOLLECTION") then
      echo objRCH("RESOURCE_PARSENAME") & "<br>"
   end if
   objRCH.MoveNext
Wend
```

Listing 3.33: Record.OpenURL.asp: Zugriff auf ein Verzeichnis über das Web

▶ RecordSet-Objekt, Seite 69

MoveRecord

Verschiebt Dateien oder Verzeichnisse.

MoveRecord

```
objRC.MoveRecord(Source, Destination,
                 [User], [Pass],
                 Options, Async)
```

Source und *Destination* geben Quelle und Ziel des Kopiervorgangs an. Verwendbar sind Pfadangaben oder URLs. Falls die Datenquelle geschützt ist, kann mit *User* und *Pass* die Identifizierung angegeben werden.

Werte für *Options* kann der folgenden Tabelle entnommen werden:

Parameter

Konstante	Beschreibung
adCopyAllowEmulation	Erlaubt die Emulation der Verschiebung durch herunterladen und anschließendes Hochladen der Daten, wenn sich diese auf zwei verschiedenen Servern befinden.
adCopyNonRecursive	Verschiebt Verzeichnisse, aber nicht deren Unterverzeichnisse.

Konstante	Beschreibung
adCopyOverWrite	Erlaubt das Überschreiben existierender Daten im Ziel.
adCopyUnspecified	Standardwert: Überschreiben ist nicht erlaubt und Rekursion wird ausgeführt.

Bei der Anwendung mit dem Parameter adCopyAllowEnumerate kann es bei unterschiedlichen Datenprovidern zum Datenverlust oder zu einem drastischen Performanceeinbruch kommen.

Asynch ist ein logischer Wert, der bestimmt, ob die Operation asynchron ausgeführt werden soll (TRUE) oder nicht (FALSE).

- DeleteRecord, Seite 116
- CopyRecord, Seite 116

Open

Open

Öffnet ein vorhandenes Record-Objekt oder erzeugt eine Datei oder ein Verzeichnis.

```
objRec.Open Source, ActiveConnection, Mode,
            CreateOptions, Options,
            [UserName], [Password]
```

Source gibt die Quelle an, entweder eine URL, einen Dateinamen mit Pfadangaben oder einen Datensatz eines RecordSet-Objekts. *ActiveConnection* bezeichnet eine Verbindung zu einem Datenprovider. *UserName* und *Password* sind optional und dienen der Authentifizierung, wenn die Quelle geschützt ist.

Mode kann einen der folgenden Werte annehmen:

Parameter

Konstante	Beschreibung
adModeRead	Öffnen nur zum Lesen
adModeReadWrite	Öffnen zum Lesen und Schreiben
adModeRecursive	Wird in Verbindung mit adModeShareDenyNone, adModeShareDenyWrite oder adModeShareDenyRead verwendet, um die Zugriffsrechte auf alle abhängigen Datensätze oder Unterverzeichnisse zu übertragen. Kann nicht mit adModeShareDenyNone alleine verwendet werden.
adModeShareDenyNone	Erlaubt grundsätzlich andere Zugriffe
adModeShareDenyRead	Erlaubt anderen den Schreib-, aber keinen Lesezugriff

Konstante	Beschreibung
adModeShareDenyWrite	Erlaubt anderen den Lese-, aber keinen Schreibzugriff
adModeShareExclusive	Öffnet exklusiv, andere Nutzer haben keinen Zugriff
adModeUnknown	Standardwert: keine Angaben zu den Rechten
adModeWrite	Öffnet nur zum Schreiben

CreateOptions kann folgende Werte annehmen:

Konstante	Beschreibung
adCreateCollection	Erzeugt eine neue Datei, anstatt eine vorhandene zu öffnen. Existiert die Datei schon, wird ein Laufzeitfehler erzeugt, es sei denn, Sie kombinieren adCreateCollection mit adOpenIfExists oder adCreateOverwrite.
adCreateNonCollection	Erzeugt einen Datensatz vom Typ adSimpleRecord
adCreateOverwrite	Modifiziert adCreateCollection, adCreateNonCollection und adCreateStructDoc um das Überschreiben zu erlauben.
adCreateStructDoc	Erzeugt einen Datensatz vom Typ adStructDoc .
adFailIfNotExists	Standardwert. Ein Laufzeitfehler wird erzeugt, wenn die Datei oder der Datensatz nicht existiert.
adOpenIfExists	Modifiziert adCreateCollection, adCreateNonCollection und adCreateStructDoc um das Öffnen zu erlauben.

Die Werte für *Options* können Sie der folgenden Tabelle entnommen werden:

Konstante	Beschreibung
adDelayFetchFields	Der Provider liest Daten nicht beim Erzeugen des Objekts, sondern beim ersten Zugriff auf Daten.
adDelayFetchStream	Der Provider liest Daten nicht beim Erzeugen des Objekts, sondern beim ersten Zugriff auf den Datenstrom.
adOpenAsync	Der asynchrone Modus wird verwendet
adOpenRecordUnspecified	Standardwert, keine weitere Angaben
adOpenSource	Öffnet die Quelle, wenn es sich um ein Skript handelt, anstatt das Skript auszuführen.

3.3.4 Eigenschaften

ActiveConnection

ActiveConnection

Verbindungsparameter eines `Record`-Objekts. Erzeugt entweder ein neues Verbindungsobjekt aus den Daten oder gibt nur die Zeichenkette zurück, die als Verbindungszeichenfolge bezeichnet wird.

```
objRC.ActiveConnection = string strConn
string strConn = objRC.ActiveConnection
```

Die Angaben für *strConn* entsprechen denen des Objekts `Connection`. Bei Zuweisung kann sowohl eine Zeichenkette als auch ein Objekt verwendet werden.

▶ `Connection`, Abschnitt 3.1 ab Seite 47

Mode

Mode

Ermittelt oder setzt die Rechte zum Ändern von Daten.

```
objConnection.Mode = long lngMode
long lngMode = objConnection.Mode
```

lngMode kann einen der folgenden Werte annehmen:

Parameter

Konstante	Beschreibung
`adModeUnknown`	Unbestimmt (Standard)
`adModeRead`	Nur Leserecht
`adModeWrite`	Nur Schreibrecht
`adModeReadWrite`	Schreib- und Leserecht
`adModeShareDenyRead`	Verhindert, dass andere eine Verbindung zum Lesen öffnen können.
`adModeShareDenyWrite`	Verhindert, dass andere eine Verbindung zum Schreiben öffnen können.
`adModeShareExclusive`	Verhindert, dass andere eine Verbindung zum Schreiben oder Lesen öffnen können.
`adModeShareDenyNone`	Verhindert, dass andere eine Verbindung öffnen können.

ParentURL

ParentURL

Gibt den übergeordneten Pfad einer URL oder das übergeordnete `Record`-Objekt zurück.

```
string strPath = objRC.ParentURL
```

▶ Open, Seite 118

RecordType

RecordType

Gibt den Typ des Objekts zurück.

```
long lngType = objRC.RecordType
```

lngType kann dabei einen der folgenden Werte annehmen:

Parameter

Konstante	Beschreibung
adSimpleRecord	Ein einfaches Objekt ohne Unterknoten
adCollectionRecord	Eine Kollektion mit Unterknoten
adStructDoc	Strukturierte Dokumente

Source

Source

Diese Eigenschaft gibt die Quelle eines Record-Objekts zurück, also entweder die URL oder einen Verweis auf das zu Grunde liegende RecordSet-Objekt.

```
string varSource = objRC.Source
objRC.Source = string varSource
```

▶ Open, Seite 118

State

State

Diese Eigenschaft gibt den Status des Record-Objekts zurück.

```
long lngState = objRC.State
```

lngState kann einen der folgenden Werte annehmen:

Parameter

Konstante	Beschreibung
adStateClosed	Das Objekt ist geschlossen (Standardwert).
adStateOpen	Das Objekt ist noch offen.
adStateExecuting	Das Kommando wird gerade ausgeführt.
adStateFetching	Das Kommando holt gerade Datensätze.

3.4 Command

Das `Command`-Objekt erzeugen Sie mit dem folgenden Befehl:

```
Set objCommand = CreateObject("ADODB.Command")
```

Die Objektvariable *objCommand* wird in allen folgenden Syntaxdiagrammen dieses Abschnitts vorausgesetzt.

3.4.1 Einführung

`Command` müssen Sie nicht für jede Abfrage einsetzen. Der Schwerpunkt liegt in der Erzeugung und Verwaltung von `Parameters`-Kollektionen, die zur Übergabe von Werten an gespeicherte Prozeduren dienen. Unabhängig davon spricht nichts dagegen, auch einfache SQL-Anweisungen mit `Command` abzusetzen, nur werden Sie mit `Connection.Execute` schneller zum Ziel kommen.

Parameter sind sinnvoll, wenn Sie damit ein einziges Kommando bedienen können und unterschiedliche Aktionen auslösen. SQL Server wird das Kommando in eine gespeicherte Prozedur kompilieren und schneller ausführen als normale Anweisungen.

3.4.2 Übersicht

Methoden

- `Cancel`, Seite 123
- `CreateParameter`, Seite 123
- `Execute`, Seite 125

Eigenschaften

- `ActiveConnection`, Seite 126
- `CommandText`, Seite 126
- `CommandTimeOut`, Seite 127
- `CommandType`, Seite 127
- `Name`, Seite 128
- `Prepared`, Seite 128
- `State`, Seite 129

Kollektionen

- `Properties`-Kollektion, Abschnitt 4.2 ab Seite 164
- `Parameters`-Kollektion, Abschnitt 4.4 ab Seite 167

3.4.3 Methoden

Cancel

Cancel

Diese Methode bricht eine noch laufende Befehlsausführung mit `Execute` ab.

objCommand.Cancel

- `Execute`, Seite 125

CreateParameter

CreateParameter

Diese Methode erzeugt einen neuen Parameter. Parameter dienen der Übergabe von Werten von und zu gespeicherten Prozeduren. Alle Parameter bilden eine Kollektion.

objCommand.CreateParameter([*name*],[*type*],[*dir*],[*size*],[*value*])

name ist die Bezeichnung des Parameters. *type* ist der Datentyp; für die Angabe sind folgende Konstanten zulässig:

Parameter

Konstante	Wert	Beschreibung
adBigInt	20	8-Byte-Ganzzahl mit Vorzeichen
adBinary	128	Binärzahl
adBoolean	11	Boolescher Wert
adBSTR	8	Unicode-Zeichenkette, die mit /0 endet
adChar	129	Zeichenkette
adCurrency	6	Währung
adDBDate	133	Datum (yyyymmdd)
adDBTime	134	Zeitwert (hhmmss)
adDBTimeStamp	135	Datum und Zeit (yyyymmddhhmmss.milliardstel)
adDecimal	14	Exakter numerischer Wert
adDouble	5	Doppelt genaue Gleitkommazahl
adEmpty	0	Kein Wert
adError	10	32-Bit-Fehlercode
adGUID	72	Globale einmalige ID
adIDispatch	9	Zeiger auf die ID eines OLE-Objekts

Konstante	Wert	Beschreibung
adInteger	3	4 Byte Integer mit Vorzeichen
adIUnknown	13	Zeiger auf IUnknown eines OLE-Objekts
adLongVarBinary	205	Langer Binärwert
adLongVarChar	201	Lange Zeichenkette
adLongVarWChar	203	Durch /0 begrenzte Zeichenkette
adNumeric	131	Exakter Zahlenwert
adSingle	4	Einfache Gleitkommazahl
adSmallInt	2	2 Byte Integer mit Vorzeichen
adTinyInt	16	1 Byte Integer mit Vorzeichen
adUnsignedBigInt	21	8 Byte Integer ohne Vorzeichen
adUnsignedInt	19	4 Byte Integer ohne Vorzeichen
adUnsignedSmallInt	18	2 Byte Integer ohne Vorzeichen
adUnsignedTinyInt	17	1 Byte Integer ohne Vorzeichen
adUserDefined	132	Benutzerdefinierter Typ
adVarBinary	204	Binärwert
adVarChar	200	Zeichenkette
adVariant	12	OLE-Variante
asVarWChar	202	Unicode-Zeichenkette mit /0 beendet
adWChar	130	Unicode-Zeichenkette mit /0 beendet

Der Parameter *dir* gibt an, ob der Parameter gelesen oder geschrieben werden kann. Zulässige Werte sind:

Konstante	Wert	Beschreibung
adBigInt	20	8-Byte-Ganzzahl mit Vorzeichen
adParamInput	1	Eingabewert (Standard)
adParamOutput	2	Ausgabewert
adParamInputOutput	3	Beide Richtungen
adParamReturnValue	4	Gibt nur einen Wert zurück

Die maximale Länge der Parameter in Zeichen oder Byte gibt *size* vor. Der eigentliche Wert des Parameters steht in *value*, wenn der Wert an eine Prozedur übergeben wird.

Execute

Execute

`Execute` führt das vorbereitete Kommando aus. Es kann sich um eine gespeicherte Prozedur oder eine SQL-Anweisung handeln. Falls die Anweisung Datensätze erzeugt, wird ein Datensatzobjekt erzeugt.

```
objCommand.Execute([affected],[parameters],[options])
set objRS =
    objCommand.Execute([affected],[parameters],[options])
```

Der Parameter *affected* ist eine Variable, deren Wert geändert wird und nach der Ausführung die Anzahl der bearbeiteten Datensätze enthält. Dies gilt nur für Anweisungen, die Datensätze verändern; `SELECT`-Anweisungen setzen diesen Wert nicht. *parameters* ist ein Array von Parametern für gespeicherte Prozeduren. Wenn die Prozedur auch Werte unter dem gleichen Namen zurückgibt, werden diese Werte hier nicht geändert (das Array wird nicht geschrieben). Mit Hilfe des Parameters *option* geben Sie an, welche Art Kommando ausgeführt werden soll. Die Angabe ist optional und steigert lediglich die Performance bei der ersten Ausführung.

Parameter

Konstante	Beschreibung
adCMDText	Beliebige SQL-Anweisung
adCMDTable	Tabellenname (entspricht `SELECT * FROM tabelle`)
adCMDTableDirect	Tabellenname (sendet nur den Namen)
adCMDStoredProc	Eine gespeicherte Prozedur in SQL
adCMDUnknown	Nicht bekannt (Standardwert)
adAsynchFetch	Führt im asynchronen Mode aus
adAsynchFetchNonBlocking	Führt im asynchronen geblockten Mode aus
adAynchExecute	Führt im asynchronen Mode aus

Mit `adAsyncFetch` wird gesteuert, wie die Methode sich verhält, wenn Datensätze nicht verfügbar sind, z.B. weil andere Nutzer darauf schreibend zugreifen. Wenn Sie `adAsyncFetch` angeben, wird gewartet, bis der Datensatz wieder frei ist, mit `adAsyncFetchNonBlocking` erhalten Sie keine Daten und `EOF` wird `TRUE`. Diese Parameter werden mit den anderen durch `Or` kombiniert:

```
objComm.Execute , , adCmdStoredProc Or adAsynchExecute
```

Das folgende Beispiel zeigt, wie ein Datensatzobjekt erzeugt wird:

```
strQuery = "Customers"
set objCommand = Server.CreateObject("ADODB.Command")
objCommand.ActiveConnection = objConn
```

```
objCommand.CommandText = strQuery
objCommand.CommandType = adCMDTable
set objRS = objCommand.Execute
show_table(objRS)
```

Listing 3.34: Command.Execute.asp: Minimale Nutzung des `Command`-Objekts

Die Funktion *show_table()* ist in OPEN.INC.ASP definiert und gibt eine komplette Tabelle aus.

3.4.4 Eigenschaften

ActiveConnection

ActiveConnection

Name der aktuellen Verbindung zur Datenquelle. Erzeugt entweder ein neues Verbindungsobjekt aus den Daten oder gibt nur die Zeichenkette zurück, die als Verbindungszeichenfolge bezeichnet wird.

```
Set objConnection = objCommand.ActiveConnection
Set objCommand.ActiveConnection = objConnection
objCommand.ActiveConnection = string strConn
string strConn = objCommand.ActiveConnection
```

Die Angaben für `strConn` entsprechen denen des Objekts `Connection`. Ein Beispiel finden Sie in Listing 3.34.

▶ `Connection`, Abschnitt 3.1, Seite 47

CommandText

CommandText

Setzt den Text, der mit `Execute` ausgeführt werden soll.

```
objCommand.CommandText = string strComm
string strComm = objCommand.CommandText
```

Für `strComm` kann eine SQL-Anweisung, ein Tabellenname, eine gespeicherte Prozedur oder ein an den Provider gerichtetes Kommando stehen. Aus der Anweisung lässt sich eine gespeicherte Prozedur erzeugen, wenn der Parameter `Prepared` gesetzt wurde. Das verbessert die Performance, wenn der Aufruf häufiger mit wechselnden Parametern erfolgt. Parameter für gespeicherte Prozeduren können mit der `Parameter`-Kollektion oder direkt übergeben werden:

```
objComm.CommandText = "my_procedure 'name', 'kennwort'"
objComm.CommandType = adCmdProcedure
objComm.Execute
```

Die Übergabe kann aber auch als Array erfolgen, wie bei `Execute` beschrieben:

```
objComm.CommandText = "my_procedure"
objComm.CommandType = adCmdProcedure
arrParameters = Array("name", "kennwort")
objComm.Execute , , arrParameters
```


- `Execute`, Seite 125
- `CommandType`, Seite 127
- `Parameters`, Abschnitt 4.4, ab Seite 167

CommandTimeOut

Command TimeOut

Zeit in Millisekunden, die auf eine Antwort gewartet wird. Der Standardwert ist 30 Sekunden.

```
long lngTime = objCommand.CommandTimeout
objCommand.CommandTimeout = long lngTime
```

Wenn ein Kommando mehr Zeit in Anspruch nimmt, wird es abgebrochen und ein Laufzeitfehler erzeugt. Dieser Wert hat keinen Zusammenhang mit `CommandTimeout` des `Connection`-Objekts.

CommandType

CommandType

Art des Kommandos. Entspricht etwa dem Parameter *option* der Methode `Execute`.

```
long lngType = objCommand.CommandType
objCommand.CommandType = long lngType
```

Den Parameter können Sie der folgenden Tabelle entnehmen:

Parameter

Konstante	Beschreibung
`adCMDText`	Beliebige SQL-Anweisung
`adCMDTable`	Tabellenname (entspricht `SELECT * FROM tabelle`)
`adCMDTableDirect`	Tabellenname (sendet nur den Namen)
`adCMDStoredProc`	Eine gespeicherte Prozedur in SQL (`CALL proc`)
`adCMDUnknown`	Nicht bekannt (Standardwert)
`adExecuteNoRecords`	Anweisung gibt keine Datensätze zurück

Ein Beispiel finden Sie in Listing 3.34.

Neben normalen Abfragen sind parametrisierte Kommandos möglich. Eine normale Abfrage kennen Sie sicher:

```
SELECT * FROM Customers WHERE CustID = 'ALFKI'
```

Mit Parameter wird der variable Wert einfach ersetzt:

```
SELECT * FROM Customers WHERE CustID = ?
```

▶ Execute, Seite 125

Name

Name

Der Name des Kommandos, die Angabe ist optional – Kommandos müssen nicht benannt werden.

```
string strName = objComm.Name
objComm.Name = string strName
```

Die Anwendung ist möglich, wenn Sie mehrere Kommandos zur Ausführung vorbereiten und in einer Kollektion speichern möchten. Der Aufruf könnte dann über den Namen erfolgen. In einem anderen Fall lassen sich die Parameter eines Kommandos nach den aktuellen Bedingungen beim Ablauf eines Skripts einstellen. Zu einem späteren Zeitpunkt wird das so präparierte Kommando mit seinem Namen aufgerufen – unabhängig davon, welche Parameter zuvor eingestellt wurden. Skripte werden dadurch möglicherweise besser strukturiert.

Der Name muss vergeben werden, bevor die Zuweisung einer Verbindung mit `ActiveConnection` erfolgt. Dafür steht der Name anschließend als Methode im `Connection`-Objekt zur Verfügung.

▶ ActiveConnection, Seite 126

Prepared

Prepared

Weist den Server an, das Kommando zu kompilieren. Das verlangsamt die erste Ausführung, beschleunigt jedoch alle folgenden.

```
boolean blnVal = objComm.Prepared
objComm.Prepared = boolean blnVal
```

blnVal kann `TRUE` oder `FALSE` sein, `FALSE` ist der Standardwert. Wenn diese Eigenschaft zusammen mit der Verbindungseigenschaft `Use Procedure for Prepare` eingesetzt wird, erstellt der SQL Server dafür eine temporäre gespeicherte Prozedur. Dieser Aufwand lohnt, wenn die Ausführung mehr als drei Mal hintereinander erfolgt.

State

Status des aktuellen Kommandos.

```
long lngState = objComm.State
```

Folgende Konstanten werden zurückgegeben:

Konstante	Beschreibung
adStateClosed	Das Objekt ist geschlossen (Standardwert).
adStateOpen	Das Objekt ist noch offen.
adStateExecuting	Das Kommando wird gerade ausgeführt.
adStateFetching	Das Kommando holt gerade Datensätze.

Die Anwendung ist sinnvoll, wenn asynchrone Zugriffe erfolgen und der Nutzer über die verzögerte Ausführung aufgeklärt werden soll:

```
If objComm.State = adStateExecuting Then
   Response.Write "Bitte warte ... <br>"
   Response.Flush
End If
```

▶ Execute, Seite 125

3.4.5 Kollektionen

Es gibt zwei Kollektionen, die sich von Command abgeleitet lassen:

▶ Properties-Kollektion, Abschnitt 4.2 ab Seite 164

▶ Parameters-Kollektion, Abschnitt 4.4 ab Seite 167

Diese Kollektion sind eigenständige Objektsammlungen und werden deshalb im Kapitel 4 *Kollektionen* beschrieben.

3.5 Field

Das Field-Objekt repräsentiert ein einzelnes Feld oder eine Spalte. Wenn eine Spalte oder Auswahl von Feldern verwaltet wird, entsteht eine Kollektion. Die nur für Kollektionen geltenden Methoden und Eigenschaften werden im Abschnitt 4.1 *Fields* ab Seite 159 beschrieben.

3.5.1 Einführung

Um ein Field-Objekt direkt zu erzeugen, nutzen Sie folgenden Aufruf.

```
Set objField = Server.CreateObject("ADODB.Field")
```

Das dürfte ein eher selten beschrittener Weg sein, denn `Field`-Objekte (dann als `Fields`-Kollektion) sind in jedem `Record`- und `RecordSet`-Objekt enthalten. Aus Performancegründen kann es sinnvoll sein, `Field`-Objekte zu verwenden, denn der Zugriff auf spezifische Methoden wird verkürzt, wenn ADO nicht zusätzlich das gesamte `RecordSet`-Objekt bearbeiten muss. Auf die Methoden und Eigenschaften des `Field`-Objekts greifen Sie über ein `RecordSet`-Objekt mit folgender Syntax zu:

```
Set objRS = Server.CreateObject("ADODB.RecordSet")
... ' Aktionen, die objRS füllen
Response.Write objRS.Fields("Name").Type
Response.Write objRS.Fields(0).Type
```

Um direkt aus dem `RecordSet`-Objekt das `Field`-Objekt zu gewinnen, gehen Sie folgendermaßen vor:

```
Set objField = objRS.Fields("Name")
```

Auf dieses Objekt können Sie nun die nachfolgend beschriebenen Methoden und Eigenschaften direkt anwenden:

```
Response.Write objField.Type
```

3.5.2 Übersicht

Methoden

- `AppendChunk`, Seite 131
- `GetChunk`, Seite 131

Eigenschaften

- `ActualSize`, Seite 131
- `Attributes`, Seite 132
- `DefinedSize`, Seite 133
- `Name`, Seite 133
- `NumericScale`, Seite 134
- `OriginalValue`, Seite 134
- `Precision`, Seite 134
- `Type`, Seite 134
- `UnderlyingValue`, Seite 134
- `Value`, Seite 135

Kollektionen

- Abschnitt 3.5.5 Kollektionen ab Seite 135

3.5.3 Methoden

AppendChunk

AppendChunk

Diese Methode hängt Daten an eine großes Text- oder Binärdatenfeld an.

```
objField.AppendChunk(Data)
```

Die Methode wird meist zum Umgang mit Bilddaten eingesetzt. Das folgende Beispiel liest die GIF- und JPG-Bilder eines Verzeichnisses in eine Datenbank ein. Das Speichern von binären Daten in einer Datenbank ist generell kritisch, denn solche Daten profitieren nicht von den vielfältigen Such- und Sortiermöglichkeiten, blähen aber die Datenbankdateien auf und machen die gesamte Datenbank langsam.

Wenn Sie dagegen mit einem Bildkatalog arbeiten und kaum zusätzliche Daten verwalten, kann dies in einer Datenbank komfortabler sein. Generell ist der Umgang mit binären Daten nicht einfach. Wenn es sich nicht vermeiden lässt, sollten Sie unbedingt einen Blick auf das Objekt `Streams` werfen, das dafür besser geeignet ist.

- `GetChunk`
- `ActualSize`, Seite 131

GetChunk

GetChunk

`GetChunk` holt binäre Daten aus einem Feld und gibt sie zurück.

```
binData = objField.GetChunk(integer len)
```

Der Parameter *len* gibt an, wie viele Bytes aus dem Feld gelesen werden sollen. Dabei wird ein interner Zeiger gesetzt, unmittelbar folgende Aufrufe der Methode setzen an der letzten Stelle fort. Sie können die Daten so stückchenweise lesen.

- `AppendChunk`, Seite 131
- `ActualSize`, Seite 131

3.5.4 Eigenschaften

ActualSize

ActualSize

Diese Eigenschaft zeigt die Größe eines Feldes an.

```
long lngSize = objField.ActualSize
objField.ActualSize = long lngSize
```

Die Eigenschaft kann geschrieben werden, wenn es sich um Felder mit variabler Größe handelt (z.B. `varchar`). Um die Größe eines Feldes von vornherein festzulegen, setzen Sie besser `DefinedSize` ein.

Wenn Unicode-Texte verwendet werden, kann `ActualSize` den Wert von `DefinedSize` überschreiten. `DefinedSize` gibt die Anzahl der Zeichen zurück, `ActualSize` dagegen die Größe in Bytes. Unicode-Zeichen können 16 Bit groß sein, entsprechend können die Werte differieren. Wenn Sie Skripte mit diesen Eigenschaften schreiben, sollten Sie die möglichen Formate berücksichtigen und gegebenenfalls Korrekturfunktionen erstellen, die dies umrechnen.

▶ `DefinedSize`, Seite 133

Attributes

Attributes

Diese Eigenschaft zeigt einige charakteristische Eigenschaften des Feldes an.

```
integer intAttr = objField.Attributes
```

Zurückgegeben wird eine oder mehrere der folgenden Konstanten:

Konstante	Beschreibung
`adFldMayDefer`	Der Wert des Feldes befindet sich nicht im Datensatz und ist nur bei explizitem Zugriff lesbar.
`adFldUpdatable`	Das Feld kann beschrieben werden.
`adFldUnKnown-Updatable`	Es ist nicht bekannt, ob Schreiben zulässig ist.
`adFldFixed`	Das Feld enthält Daten konstanter Größe.
`adFldIsNullable`	`NULL` ist erlaubt.
`adFldMayBeNull`	`NULL` kann gelesen werden.
`adFldLong`	`AppendChunk` und `GetChunk` sind erlaubt.
`adFldRowID`	Das Feld ist ein `IDENTITY`-Feld.
`adFldRowVersion`	Das Feld ist ein `TIMESTAMP`-Feld.
`adFldCacheDeferred`	Die Daten werden vom und in den Cache geliefert.
`adFldIsChapter`	Entspricht einem Chapter in einer hierarchischen Datengruppe.
`adFldNegativeScale`	Das Feld ist negativ skaliert, der Skalierungsfaktor bestimmt die Anzahl der *Vorkommastellen*, um die die Ausgabe verschoben wird.
`adFldKeyColumn`	Das Feld ist ein Primärschlüssel.

Konstante	Beschreibung
adFldIsRowURL	Das Feld zeigt die Position im Record-Objekt an.
adFldIsDefaultStream	Das Feld enthält den Standarddatenstrom.
adFldIsCollection	Das Feld enthält eine Kollektion.

Die Konstanten lassen sich kombinieren. Dadurch kann der Rückgabewert nicht direkt gelesen werden. Sie können folgende Schreibweise zur Ermittlung einer bestimmten Eigenschaft verwenden:

```
If (objField.Attributes And adFldIsNullable) Then
    Response.Write "<pre>NULL</pre> ist erlaubt<p>"
End If
```

Um zwei Eigenschaften abzufragen, gehen Sie folgendermaßen vor:

```
If (objField.Attributes And (adFldIsNullable + adFldFixed) Then
    Response.Write "<pre>NULL</pre> ist erlaubt<br>"
    Response.Write "Das Feld hat eine Konstante Länge<br>"
End If
```

- NumericScale, Seite 134
- ActualSize, Seite 131
- DefinedSize, Seite 133

DefinedSize

DefinedSize

Diese Eigenschaft gibt den definierten Platzverbrauch in Zeichen eines Feldes zurück.

```
integer intLong = objField.DefinedSize
```

Ob das Feld konkret gefüllt ist, spielt keine Rolle. Wenn als SQL-Datentyp varchar(255) angegeben wurde, gibt die Eigenschaft 255 zurück, auch wenn der Inhalt weniger Zeichen hat.

Name

Name

Diese Eigenschaft gibt den Namen des Feldes zurück.

```
string strName = objField.Name
```

Sie ermitteln damit Feldnamen, ohne diese zu kennen. Dazu wird die Fields-Kollektion durchlaufen:

```
for each feld in objRec.Name
   Response.Write feld.Name & " -> "
next
```

▶ Record, Abschnitt 3.3 ab Seite 113

NumericScale

NumericScale

Mit dieser Eigenschaft ermitteln Sie die Anzahl der Nachkommastellen, wie sie definiert wurden:

```
integer intLong = objField.NumericScale
```

▶ DefinedSize, Seite 133

OriginalValue

OriginalValue

Diese Eigenschaft enthält den Wert des Feldes vor der letzten Änderung. Dieser Wert wird eingesetzt, wenn die Methoden Cancel oder CancelUpdate aufgerufen werden.

```
varVar = objField.OriginalValue
```

▶ Cancel, Seite 115 und Abschnitt 3.3 Record, ab Seite 113

Precision

Precision

Mit dieser Eigenschaft ermittelt man die Genauigkeit von numerischen Werten. Dabei wird die Anzahl der möglichen Stellen vor dem Komma zurückgegeben.

```
long lngPrecision = objField.OriginalValue
```

Type

Type

Diese Eigenschaft gibt den Datentyp zurück. Der Wert entspricht den ADO-Konstanten für Datentypen.

```
integer intType = objField.Type
```

Der konkrete Wert hängt auch vom Provider und den in der Datenbank verfügbaren Datentypen ab.

UnderlyingValue

UnderlyingValue

Diese Eigenschaft repräsentiert den Wert des Feldes in der Datenbank. Dieser Wert kann von Value abweichen, wenn im lokalen Datensatzobjekt Änderungen vorgenommen wurden und noch kein Update erfolgt.

```
variant varVar = objField.UnderlyingValue
```

▶ Value

Value

Diese Eigenschaft enthält den aktuellen Wert des Feldes im lokalen Datensatz.

```
variant varVar = objField.Value
```

Beachten Sie den Unterschied zu `OriginalValue` und `UnderlyingValue` bei Änderungen. `Value` enthält den geänderten Wert, `OriginalValue` den Wert vor der letzten Änderung, bevor `Update` aufgerufen wurde, und `UnderlyingValue` den in der Datenbank gespeicherten Wert.

- `OriginalValue`, Seite 134
- `UnderlyingValue`, Seite 134

3.5.5 Kollektionen

Das Objekt `Field` kann folgende Kollektion enthalten:

- `Properties`, Abschnitt 4.2 ab Seite 164

3.5.6 Beispiele

Das folgende Beispiel zeigt, wie die `Field`-Eigenschaften eingesetzt werden können:

```
echo "<table border=1><tr>"
for each feld in objRS.Fields
   echo "<th colspan=2>" & feld.Name & "</th>"
next
echo "</tr>"
while not objRS.EOF
   echo "<tr>"
   for each feld in objRS.Fields
       echo "<td>"
       echo feld.Value
       echo "</td><td nowrap>"
       echo "DS: " & feld.DefinedSize & "<br>"
       echo "NS: " & feld.NumericScale & "<br>"
       echo "PR: " & feld.Precision & "<br>"
       echo datatype(cstr(feld.Type))
       echo "</td>"
   next
   echo "</tr>"
```

```
   objRS.MoveNext
wend
echo "</tr></table>"
```

Listing 3.35: field_properties.asp: Anwendung verschiedener `Field`-Properties

Sie können an diesem Beispiel gut erkennen, wie die Werte interpretiert werden. Wenn Sie die Beispiele von der Website zum Buch laden, finden Sie dort auch eine Include-Datei, die die `DataType`-Werte umkehrt und zu den numerischen Werten den Namen ausgibt. Die Datei erzeugt ein Dictionary namens *`datatype`*. Einbinden können Sie dieses Modul wie folgt:

```
<!-- #include file="datatypes.ado.inc.asp" -->
```

3.6 Property

Das `Property`-Objekt enthält Eigenschaften anderer Objekte. Es tritt normalerweise nicht allein auf, sondern als Kollektion von Eigenschaften, die direkt aus einem der folgenden Objekte abgeleitet werden:

- `Connection`, Seite 47
- `Command`, Seite 122
- `RecordSet`, Seite 69
- `Field`, Seite 129

3.6.1 Einführung

Ableitung des Objekts

Das `Property`-Objekt wird nur selten direkt verwendet. Üblich ist die Ableitung aus einem der bereits genannten Objekte:

```
Response.Write objConn.Properties("User ID").Attributes
```

In diesem Fall wird die Kollektion `Properties` angesprochen und innerhalb dieser Kollektion das Objekt »User ID«. Im Anhang A.1 Properties-Kollektion ab Seite 403 finden Sie eine Liste aller zulässigen Namen für `Properties`. Alle folgenden Eigenschaften beziehen sich auf jeweils eines dieser Elemente.

3.6.2 Übersicht

Eigenschaften

- `Attributes`, Seite 137
- `Name`, Seite 139

- Type, Seite 139
- Value, Seite 140

3.6.3 Methoden

Das Objekt Property hat keine Methoden.

3.6.4 Eigenschaften

Attributes

Diese Eigenschaft ermittelt einige Charakteristika des Objekts.

Attributes

```
long lngAttributes = objProperty.Attributes
```

Die Rückgabe ist ein numerischer Wert, der die Summe der folgenden Konstanten darstellt:

Konstante	Beschreibung
adPropNotSupported	Die Eigenschaft wird nicht unterstützt.
adPropRequired	Diese Eigenschaft wird unterstützt und muss gesetzt werden, bevor die Abfrage ausgeführt wird.
adPropOptional	Diese Eigenschaft muss nicht spezifiziert werden.
adPropRead	Diese Eigenschaft kann gelesen werden.
adPropWrite	Diese Eigenschaft kann geschrieben werden.

Aus der Kombination der Konstanten ergeben sich typische Attribute der benannten Eigenschaften. Das folgende Beispiel ermittelt dies für verschiedene typische Eigenschaften des Objekts Connection. Dieses Beispiel bezieht auch die anderen Eigenschaften des Objekts Properties mit ein.

Beispiel

Der erste Teil zeigt, wie Sie auf die gesamte Kollektion zugreifen können:

```
<form method="post" action="<% = ASP_SELF %>">
Bitte w&auml;hlen Sie eine Eigenschaft aus:
<select name="property" size="1">
  <%
  open()
  dim prop
  for each prop in objConn.Properties
     echo "<option value=""" & prop.Name & """"
```

```
    if Request.Form("property") = prop.Name then
       echo " selected "
    end if
    echo ">" & prop.Name
  next
  %>
</select>
<input type="Submit" value="Attributes anzeigen">
</form>
```

Der zweite Teil wertet die mit dem Formular ausgewählte Eigenschaft aus. Das `Dictionary` *datatype* wird in der Datei DATATYPES.ADO.INC.ASP aufgebaut und stellt Datentypen mit Klartextnamen dar.

```
<%
dim strProperty, varAttr, objProp
set objProp = objConn.Properties
' Datenverbindung herstellen
strProperty = Request.Form("property")
if len(strProperty) > 0 then
   echo "Es wurden folgende Attribute f&uuml;r die Eigenschaft
         <code>$strProperty</code> ermittelt: "
   varAttr = objProp(strProperty).Attributes
   echo "<ul>"
   echo "<li>Eigenschaftswert: $varAttr"
   if varAttr = 0 then
       echo "<li>Diese Eigenschaft wird nicht
                 unterst&uuml;tzt"
   else
       if varAttr and adPropRequired then
          echo "<li>Muss vor der Abfrage spezifiziert werden."
       end if
       if varAttr and adPropOptional then
          echo "<li>Die Angabe der Eigenschaft ist optional."
       end if
       if (varAttr and adPropRead) then
          echo "<li>Eigenschaft ist lesbar"
       end if
       if (varAttr and adPropWrite) and
          (varAttr and adPropRead) then
           echo "<li>Eigenschaft kann geschrieben werden"
       end if
   end if
   echo "<li>Datentyp: <b>"
         & datatype(cstr(objProp(strProperty).Type)) & "</b>"
   echo "<li>Der aktuelle Wert ist: <b>"
```

```
        & objProp(strProperty).Value & "</b>"
   echo "</ul>"
end if
%>
```

Listing 3.36: Properties.Attributes.asp: Anzeige der Elemente der Kollektion Properties

Abbildung 3.9: Ausgabe von Listing 3.36

- Name, Seite 139
- Type, Seite 139
- Value, Seite 140

Name

Name

Diese Eigenschaft gibt den Namen der Property zurück.

```
string strName = objProperty.Name
```

Ein Beispiel finden Sie in Listing 3.36.

Type

Type

Diese Eigenschaft gibt den Datentyp im ADO-Format zurück.

```
long lngType = objProperty.Type
```

Ein Beispiel finden Sie in Listing 3.36.

Value

Value

Mit dieser Eigenschaft erhalten Sie Zugriff auf den Wert. Wenn die Eigenschaft geschrieben werden kann, ist auch `Value` schreibbar.

```
variant varValue = objProperty.Value
objProperty.Value = variant varValue
```

Das folgende Beispiel erzeugt passende Eingabefelder für die Werte und sperrt das Feld, wenn die Änderung nicht mehr möglich ist. Das Formular mit der Auswahl der `Properties` entspricht dem in Listing 3.36 gezeigten.

```
set objProp = objConn.Properties
' Datenverbindung herstellen
strProperty = Request.Form("property")
if len(strProperty) > 0 then
   intType = cint(objProp(strProperty).Type)
   varValue =  objProp(strProperty).Value
   varAttr = objProp(strProperty).Attributes
   if varAttr = 0 then
       echo "Diese Eigenschaft wird
             nicht unterst&uuml;tzt.<br>"
   else
      if varAttr and adPropRequired then
         echo "Muss vor der Abfrage spezifiziert werden.
            &Auml;nderungen sind nicht mehr m&ouml;glich.<br>"
     disabled =  "disabled"
     elseif (varAttr and adPropWrite) then
        echo "Eigenschaft kann geschrieben werden"
     end if
     echo "<br><code> $strProperty </code>"
      select case intType
        case 16, 2, 3, 20, 17, 18, 19, 21:
            echo "<input type=text size=10 name=prop
                  value=""$varValue"" $disabled> (Integer)"
        case 4, 5, 14, 131:
            echo "<input type=text size=10 name=prop
                  value=""$varValue"" $disabled>
                  (Gleitkommazahl)"
    case 6:
            echo "<input type=text size=10 name=prop
                  value=""$varValue"" $disabled>
                  (W&auml;hrung)"
        case 11:
       if varValue = TRUE then checked = "checked"
              echo "<input type=checkbox name=prop $checked
                    $disabled> (Boolean)"
```

```
        case 7, 133, 134, 135:
            echo "<input type=text size=10 name=prop
                  value=""$varValue"" $disabled> (Datum)"
        case 8, 129, 200, 201, 130, 202, 203:
            echo "<input type=text size=10 name=prop
                  value=""$varValue"" $disabled> (String)"
        case else:
            echo "Datentyp wird nicht unterst&uuml;tzt."
       end select
   end if
   echo "</ul>"
end if
```

Listing 3.37: Properties.Value.asp: Erzeugen von Formularfeldern in Abhängigkeit vom Typ der Eigenschaft

Abbildung 3.10: Listing 3.37 mit einer Booleschen, gesperrten Eigenschaft

3.7 Stream

Mit dem `Stream`-Objekt kann der Zugriff auf Textdateien oder binäre Daten erfolgen. Die Einführung in ADO 2.5 erfolgte vor allem in Hinblick auf die Verarbeitung von Textdateien, die für XML notwendig sind.

3.7.1 Einführung

Die Anwendung basiert auf folgendem Code:

```
objStream = Server.CreateObject("ADODB.Stream")
```

Stream ergänzt in hervorragender Weise das Objekt Record. Mit Record können Sie auf die Struktur von Dokumenten und deren Position im Dateisystem zugreifen. Stream erlaubt dann den Zugriff auf den Inhalt der Dokumente. Der Inhalt kann mit Stream-Methoden natürlich auch verändert und geschrieben werden.

Stream verfügt über die Eigenschaft der Persistenz. Außerdem können große Binärobjekts (BLOBs), wie beispielsweise Bilder, verarbeitet werden. Damit stellt Stream auch eine Alternative zu FileSystemObject und vor allem zu den unhandlichen Methoden GetChunk und AppendChunk des Field-Objekts dar.

3.7.2 Übersicht

Die folgende Übersicht zeigt alle Methoden und Eigenschaften, die nachfolgend vorgestellt werden. Die für ASP unbedeutenden Ereignisse sind nicht aufgeführt. Sie finden dazu Informationen in der Online-Dokumentation.

Methoden

- Cancel, Seite 143
- Close, Seite 143
- CopyTo, Seite 143
- Flush, Seite 144
- LoadFromFile, Seite 144
- Open, Seite 145
- Read, Seite 146
- ReadText, Seite 146
- SaveToFile, Seite 147
- SetEOS, Seite 147
- SkipLine, Seite 148
- Write, Seite 148

WriteText, Seite 148

Eigenschaften

- CharSet, Seite 149
- EOS, Seite 150
- LineSeparator, Seite 150
- Mode, Seite 150
- Position, Seite 151
- Size, Seite 151
- State, Seite 151
- Type, Seite 152

Kollektionen

Das Stream-Objekt bildet keine Kollektionen.

3.7.3 Methoden

Cancel

Cancel

Mit dieser Methode wird eine laufende Open-Methode abgebrochen.

objStream.Cancel

Die Methode erzeugt einen Laufzeitfehler, wenn Open nicht mit dem Parameter adRunAsync aufgerufen wurde.

- Open, Seite 145
- Close, Seite 143

Close

Close

Diese Methode schließt das Stream-Objekt und gibt den verwendeten Speicher frei.

objStream.Close

Die Methode erzeugt einen Laufzeitfehler, wenn zuvor Open nicht aufgerufen wurde.

- Open, Seite 145

CopyTo

CopyTo

Hiermit werden eine Anzahl Zeichen oder Bytes von einem Stream-Objekt zu einem anderen kopiert. Beide Objekte müssen vom gleichen Typ sein.

objStream.CopyTo *objAnotherStream*, *Number*

objAnotherStream ist eine Objektvariable eines anderen Stream-Objekts, *Number* die Anzahl der zu kopierenden Zeichen. *Number* ist optional, standardmäßig werden alle Zeichen bis EOS (*End Of Stream*) kopiert. Der Kopiervorgang beginnt an der aktuellen Position des Dateizeigers, der durch die Eigenschaft Position festgelegt wird. Nach dem Kopiervorgang steht der Zeiger am Ende des kopierten Blockes. Der Kopiervorgang fügt die Zeichen im Ziel ebenfalls ab der Position ein, die durch Position festgelegt wurde. Befinden sich mehr Zeichen im Ziel, bleiben die restlichen Zeichen erhalten. Wenn Sie dies nicht wünschen, setzen Sie die Position des Dateiendes mit SetEOS.

- Open, Seite 145
- EOS, Seite 150
- Position, Seite 151
- SetEOS, Seite 147

▶ Type, Seite 152

Flush

Flush

Sendet den Inhalt des Stream-Objekts sofort an das zu Grunde liegende Objekt oder die Datenquelle, z.B. eine URL.

objStream.Flush

Wenn Sie Close aufrufen, wird Flush implizit ausgeführt.

▶ Close, Seite 143

LoadFromFile

LoadFromFile

Mit dieser Methode wird der Inhalt einer Datei in ein Stream-Objekt geladen.

objStream.LoadFromFile *Filename*

Filename muss ein existierender Dateiname mit oder ohne Pfadangabe sein. Das Format kann ein absoluter oder ein UNC-Pfad sein. Das Stream-Objekt muss bereits mit Open geöffnet sein. Existierende Daten werden gelöscht. Position wird auf das erste Zeichen gesetzt.

```
dim objStream, strFile
set objStream = Server.CreateObject("ADODB.Stream")
strFile = Server.Mappath("save.txt")
objStream.Type = adTypeText
objStream.Charset = "ISO-8859-1"
objStream.Open
objStream.LoadFromFile strFile
%>
Ausgabe der geladenen Datei "<% = strFile %>":
<p>
<%
dim strDummy
while not objStream.EOS
   strDummy = objStream.ReadText(1)
   Response.Write "<code>" & encode(strDummy) & "</code>"
wend
```

Listing 3.38: Stream.LoadFromFile.asp: Einlesen einer Textdatei

Das zeichenweise Auslesen ist natürlich nur sinnvoll, wenn Sie die Zeichen auch einzeln verarbeiten möchten. Mit dem Parameter -1 können Sie die gesamte Datei auf einmal auslesen:

```
strDummy = objStream.ReadText(-1)
```

```
Response.Write "<code>" & encode(strDummy) & "</code>"
```

Listing 3.39: Stream.LoadFromFile.2.asp: Alles auslesen

- Open, Seite 145
- EOS, Seite 150
- Position, Seite 151
- ReadText, Seite 146

Open

Open

Open öffnet ein Stream-Objekt.

objStream.Open *Source*, *Mode*, *OpenOptions*, *UserName*, *Password*

Der Parameter *Source* kann ein Pfad, ein UNC-Pfad, eine URL oder ein Record-Objekt sein. Die Angabe ist optional, ohne Angabe wird ein unbestimmtes Stream-Objekt erzeugt und geöffnet, das sich mit Hilfe der Eigenschaften modifizieren lässt.

Mode kann einen der folgenden Werte annehmen:

Parameter

Konstante	Beschreibung
adModeUnknown	Unbestimmt (Standard)
adModeRead	Nur Leserecht
adModeWrite	Nur Schreibrecht
adModeReadWrite	Schreib- und Leserecht
adModeShareDenyRead	Verhindert, dass andere eine Verbindung zum Lesen öffnen können.
adModeShareDenyWrite	Verhindert, dass andere eine Verbindung zum Schreiben öffnen können.
adModeShareExclusive	Verhindert, dass andere eine Verbindung zum Schreiben oder Lesen öffnen können.
adModeShareDenyNone	Verhindert, dass andere eine Verbindung öffnen können.

OpenOptions kann einer der folgenden Werte sein:

Konstante	Bemerkung
adOpenStreamAsync	Öffnet das Objekt im asynchronen Mode
adOpenStreamFromRecord	Wird verwendet, wenn der Inhalt ein bereits geöffnetes Record-Objekt ist. Standardmäßig wird eine Verzeichnisstruktur erwartet.
adOpenStreamUnspecified	Standardwert, das Objekt wird mit unbestimmten Werten geöffnet.

UserName und *Password* dienen dazu, den Zugriff zu authentifizieren, wenn die Quelle geschützt ist. Die Angabe ist optional. Wenn die Quelle ein Record-Objekt ist, werden *UserName* und *Password* niemals verlangt.

Wenn ein Stream-Objekt unbestimmt geöffnet wurde, ist die Eigenschaft Size gleich Null.

- Close, Seite 143
- SaveToFile, Seite 147
- Size, Seite 151

Read

Read

Liest eine Anzahl Bytes aus dem Stream-Objekt in eine Variable.

```
varData = objStream.Read(Number)
```

Gelesen wird ab Position die Anzahl *Number* Bytes. Für das Lesen von Texten wird besser ReadText verwendet.

- Open, Seite 145
- Position, Seite 151
- ReadText, Seite 146

ReadText

ReadText

Liest eine Anzahl Zeichen aus dem Stream-Objekt in eine Variable; das Objekt muss vom Typ adTypeText sein.

```
strData = objStream.ReadText(Number)
```

Gelesen wird ab Position die Anzahl *Number* Zeichen. Statt der Anzahl kann auch eine der folgenden Konstanten eingesetzt werden:

Parameter

Konstante	Bemerkung
adReadAll	Liest alles von Position bis EOS (-1)
adReadLine	Liest die nächste Zeile der Textdatei

Die Unterteilung einer Textdatei in Zeilen erfolgt anhand der Eigenschaft LineSeparator.

Für das Lesen von binären Daten verwendet man besser Read.

- LineSeparator, Seite 150
- Open, Seite 145
- Position, Seite 151
- Read, Seite 146

SaveToFile

SaveToFile

Diese Methode schreibt den Inhalt des Stream-Objekts in eine Datei. Das Objekt muss zuvor geöffnet werden.

objStream.SaveToFile string *FileName*, integer *SaveOptions*

FileName ist ein vollständiger lokaler Pfad oder ein UNC-Pfad. *SaveOptions* können Sie der folgenden Tabelle entnehmen, die Angabe ist optional:

Parameter

Konstante	Bemerkung
adSaveCreateNotExists	Wenn die Datei nicht existiert, wird sie angelegt. Dies ist der Standardwert.
adSaveCreateOverWrite	Wenn die Datei bereits existiert, wird sie überschrieben.

- CopyTo, Seite 143
- Read, Seite 146
- ReadText, Seite 146
- Open, Seite 145

SetEOS

SetEOS

Diese Methode schneidet die Datei an der aktuell durch Position markierten Stelle ab.

objStream.SetEOS

Die Anwendung ist für CopyTo, Write und WriteText sinnvoll, wo keine Kürzung einer vorher längeren Datei erfolgt. Wenn Sie SetEOS unmittelbar nach

dem Öffnen aufrufen, werden alle Daten gelöscht, weil zu diesem Zeitpunkt `Position` auf dem Dateianfang steht.

- `CopyTo`, Seite 143
- `EOS`, Seite 150
- `Position`, Seite 151
- `Write`, Seite 148
- `WriteText`, Seite 148

SkipLine

SkipLine

Wenn das `Stream`-Objekt vom Typ `adTypeText` ist, wird der interne Dateizeiger eine Zeile weitergesetzt. Daten werden nicht zurückgegeben. Die Zeilentrennung erfolgt durch `LineSeparator`.

```
objStream.SkipLine
```

Wenn Sie nach `EOS` die Methode anwenden, verbleibt der Zeiger auf `EOS`. Ein Laufzeitfehler entsteht jedoch nicht.

- `EOS`, Seite 150
- `LineSeparator`, Seite 150
- `Type`, Seite 152

Write

Write

Diese Methode schreibt ab der durch `Position` angezeigten Stelle den Inhalt einer Variablen in das `Stream`-Objekt.

```
objStream.Write Data
```

Vorhandene Bytes werden überschrieben, der überhängende Teil der bestehenden Daten wird jedoch nicht gelöscht. Setzen Sie dazu anschließend `SetEOS` ein. Für Textdateien wird `WriteText` verwendet.

- `Position`, Seite 151
- `SetEOS`, Seite 147
- `WriteText`, Seite 148

WriteText

WriteText

Diese Methode schreibt ab der durch `Position` angezeigten Stelle den Inhalt einer Zeichenkettenvariablen in das `Stream`-Objekt.

objStream.WriteText *Data*

Vorhandene Zeichen werden überschrieben, der überhängende Teil der bestehenden Daten wird jedoch nicht gelöscht. Setzen Sie dazu anschließend SetEOS ein. Für Binärdateien wird Write verwendet.

- Position, Seite 151
- SetEOS, Seite 147
- Write, Seite 148

3.7.4 Eigenschaften

Charset

Charset

Diese Eigenschaft bestimmt den Zeichensatz, der einem Stream-Objekt vom Typ adTypeText zu Grunde liegt.

objStream.Charset = string *strChar*
strChar = *objStream*.Charset

Möglich sind alle Werte, die typischerweise als Bezeichnungen für Zeichensätze unter Windows zulässig sind, z. B. »iso-8859-a« oder »windows-1252«. Der Standardwert ist »unicode«.

Beachten Sie, dass ASP Unicode nicht anzeigen kann. Wenn Sie Unicode-Daten ungefiltert anzeigen, werden Fragezeichen (ASCII-Code 63) ausgegeben. Der folgende Code reagiert darauf:

Beispiel

```
objStream = Server.CreateObject("ADODB.Stream")
strFile = objStream.ReadText(adReadAll)
if asc(mid(strFile, 1, 1)) = 63 then
   objStream.Charset = "ascii"
   objStream.Type = adTypeText
end if
Response.Write objStream.ReadText(adReadAll)
```

Zuerst wird der Inhalt gelesen. Besteht er nur aus Fragezeichen, dann wird der Ausgabe-Zeichensatz »ascii« eingestellt. Dann wird der Stream erneut mit ReadText gelesen. Jetzt kann der Inhalt angezeigt werden.

Die Veränderung der Eigenschaft ist nur erlaubt, wenn Position gleich 0 ist. Die ist normalerweise nur unmittelbar nach dem Öffnen der Datei zulässig.

- Position, Seite 151
- Open, Seite 145

EOS

EOS

Diese Eigenschaft ist `TRUE`, wenn sich der Dateizeiger `Position` am Dateiende befindet.

```
boolean blnEOS = objStream.EOS
```

Wenn die Datei an der aktuellen Position abgeschnitten werden soll, wird `SetEOS` eingesetzt. Die Anwendung von `EOS` erfolgt normalerweise in Bedingungsabfragen:

```
while not objStream.EOS
  Response.Write objStream.ReadText(adReadLine) & "<br>"
wend
```

- `Position`, Seite 151
- `SetEOS`, Seite 147

LineSeparator

LineSeparator

Diese Eigenschaft bestimmt, welches Trennzeichen für das Zeilenende bestimmt wird.

```
objStream.LineSeparator = integer intCode
integer intCode = objStream.LineSeparator
```

Die möglichen Einstellungen entnehmen Sie der folgenden Tabelle:

Parameter

Konstante	Bemerkung
`adCR`	Wagenrücklauf (CR, Chr(13))
`adCRLF`	Wagenrücklauf und Zeilenvorschub (CRLF, Chr(13) und Chr(10))
`adLF`	Nur Zeilenvorschub (LF, Chr(10))

`adCRLF` ist der Standardwert (-1), alle anderen Konstanten entsprechen dem jeweiligen ASCII-Code, d.h., `adCR` ist gleich 13 usw.

- `ReadText`, Seite 146

Mode

Mode

Ermittelt oder setzt die Rechte zum Ändern von Daten.

```
objStream.Mode = long lngMode
long lngMode = objStream.Mode
```

lngMode kann einen der folgenden Werte annehmen:

Parameter

Konstante	Beschreibung
adModeUnknown	Unbestimmt (Standard)
adModeRead	Nur Leserecht
adModeWrite	Nur Schreibrecht
adModeReadWrite	Schreib- und Leserecht
adModeShareDenyRead	Verhindert, dass andere eine Verbindung zum Lesen öffnen können.
adModeShareDenyWrite	Verhindert, dass andere eine Verbindung zum Schreiben öffnen können.
adModeShareExclusive	Verhindert, dass andere eine Verbindung zum Schreiben oder Lesen öffnen können.
adModeShareDenyNone	Verhindert, dass andere eine Verbindung öffnen können.

Position

Position

Diese Eigenschaft zeigt auf das aktuelle Zeichen des Datenstroms. Das erste Byte hat die Position 0.

```
objStream.Position = long lngPosition
long lngPosition  = objStream.Position
```

Wenn die Datei Unicode-Daten enthält, besteht jedes gültige Zeichen aus zwei Bytes. Der Zeiger sollte deshalb immer auf gerade Zahlen eingestellt werden. `Position` arbeitet unabhängig vom Inhalt immer byteweise.

▶ `Size`, Seite 151

Size

Size

Die `Size`-Eigenschaft zeigt die Länge des Datenstromobjekts in Bytes an. Kann die Größe nicht ermittelt werden, wird -1 zurückgegeben.

```
long lngSize = objStream.Size
```

▶ `Position`, Seite 151

State

State

Diese Eigenschaft gibt den Status des `Stream`-Objekts zurück.

```
long lngState = objStream.State
```

lngState kann einen der folgenden Werte annehmen:

Konstante	Beschreibung
adStateClosed	Das Objekt ist geschlossen (Standardwert).
adStateOpen	Das Objekt ist noch offen.
adStateExecuting	Das Kommando wird gerade ausgeführt.
adStateFetching	Das Kommando holt gerade Datensätze.

Type

Type

Diese Eigenschaft zeigt an, welcher Datentyp sich im Datenstrom befindet.

```
integer intType = objStream.Type
```

intType kann einen der folgenden Werte annehmen:

Konstante	Beschreibung
adTypeBinary	Das Stream-Objekt enthält binäre Daten.
adTypeText	Es handelt sich um Textdaten (Standardwert).

Beispiel

```
set objFO = Server.CreateObject("Scripting.Filesystemobject")
set objF = objFO.GetFolder(Server.Mappath("images"))
for each f in objF.Files
   filename = objFO.GetBasename(f) & "."
              & objFO.GetExtensionName(f)
   echo "Bild <a href=""Stream.BLOB.Show.asp?file=$filename"">
         $filename</a> <br>"
next
```

Listing 3.40: Stream.BLOB.asp: Einlesen eines Verzeichnisses mit Bildern und Übergabe an das Ausgabeskript Stream.BLOB.Show.asp.

Das Antwortskript sendet das ausgewählte Bild dann direkt zum Browser:

```
<%
dim objStream, adTypeBinary
adTypeBinary = 1
Set objStream = Server.CreateObject("ADODB.Stream")
objStream.Type = adTypeBinary
objStream.Open
objStream.LoadFromFile = Server.Mappath("images/"
                         & Request.QueryString("file"))
Response.ContentType = "image/gif"
Response.BinaryWrite objStream.Read
```

```
objStream.Close
set objStream = Nothing
%>
```

Listing 3.41: Stream.BLOB.Show.asp: Ausgabe der Bilddaten

Generell kann `Stream` eingesetzt werden, wenn Dateizugriffe auf große ASCII- oder Binärdaten erfolgen. Außerdem ist der Einsatz zusammen mit XML möglich. Dies wird im Abschnitt 5.4 *XML* ab Seite 215 näher behandelt.

3.8 Error

Das `Error`-Objekt verwaltet Fehlerausgaben. Es ist immer besser, mögliche Fehlermeldungen abzufangen und gezielt zu verarbeiten, als dies der Laufzeitumgebung zu überlassen.

3.8.1 Einführung

Es gibt zwei Möglichkeiten des Zugriffs auf Fehlerinformationen. Zunächst ist das `Errors`-Objekt von `Connection` ableitbar. Wenn Sie ein bereits explizites `Connection`-Objekt *`objConn`* haben, gibt es diese Zugriffsmöglichkeit:

```
objConn.Errors(0).<property>
```

Dabei steht *<property>* für eine der nachfolgend beschriebenen Eigenschaften. Auf die Verbindung können Sie auch indirekt über das `RecordSet`-Objekt zugreifen – über die Eigenschaft `ActiveConnection`. Zulässig ist deshalb auch der folgende Aufruf:

```
objRS.ActiveConnection.Errors(0).<property>
```

Mehrere Fehler auswerten

Es besteht die Möglichkeit, dass ein Provider mehrere Fehler zurückliefert. Normalerweise gehen diese verloren, wenn Sie mit der oben gezeigten Methode nur den ersten Fehler `(0)` abrufen. Sie können die Kollektion aber mit `For Each` durchlaufen:

```
For Each error in objConn.Errors
   Response.Write "Fehler: " & error.Number & "<br>"
   Response.Write error.Description & "<hr>"
Next
```

3.8.2 Übersicht

Eigenschaften

- `Description`, Seite 154
- `HelpContext`, Seite 154
- `HelpFile`, Seite 154
- `NativeError`, Seite 155
- `Number`, Seite 155
- `Source`, Seite 155
- `SQLState`, Seite 155

3.8.3 Methoden

Die Kollektion des Objekts `Errors` hat keine Methoden.

3.8.4 Eigenschaften

Description

Description

Diese Eigenschaft enthält den beschreibenden Text des letzten Fehlers.

```
string strText = objError.Description
```

Es handelt sich hier um die Standardeigenschaft, die Angabe kann deshalb entfallen.

HelpContext

HelpContext

Diese Eigenschaft gibt die ID in einer verbundenen Hilfedatei zurück. Unter ASP ist dies normalerweise wenig sinnvoll.

```
long lngContext = objError.HelpContext
```

- `HelpFile`

HelpFile

HelpFile

Diese Eigenschaft gibt den Dateinamen der Hilfedatei zurück. Unter ASP ist dies normalerweise wenig sinnvoll.

```
strFile = objError.HelpFile
```

- `HelpContext`

NativeError

NativeError

ADO interpretiert normalerweise Fehler und gibt eine eigene Fehlernummer zurück. Der Provider ist aber möglicherweise in der Lage, mehrere Fehlernummern zu erzeugen, als ADO kennt. Dies ist der Fall, wenn Sie in gespeicherten Prozeduren eigene Fehlercodes erzeugen. Die Eigenschaft `NativeError` enthält diese ursprünglichen Fehlerwerte. Welcher konkrete Inhalt sich dahinter verbirgt, hängt von der Applikation und der Datenbank ab.

```
long lngErrorCode = objError.NativeError
```

Es gibt zwei typische Fehlercodes in ADO, die darauf hinweisen, dass der Provider einen speziellen Fehlercode gesendet hat: -2 147 217 900 und -2 147 467 259.

Number

Number

Diese Eigenschaft enthält den von ADO verwendeten, internen Fehlercode. Eine Auflistung aller Fehler finden Sie im Anhang.

```
long lngNumber = objError.Number
```

▶ NativeError, Seite 155

Source

Source

Diese Eigenschaft enthält eine Zeichenkette mit dem Namen des Objekts, das den Fehler ursprünglich erzeugt hat. Wenn der Provider den Fehler erzeugt, funktioniert dies nicht.

```
string strSource = objError.Source
```

SQLState

SQLState

Der Provider erkennt die Fehlermeldungen der Datenbank beim Abarbeiten von SQL-Kommandos und legt den fünfstelligen SQL-Code dann in der Eigenschaft `SQLState` ab.

```
long lngContext = objError.SQLState
```

Beispiele

Das folgende Beispiel zeigt, wie sich Fehlermeldungen anzeigen lassen:

```
<%
on error resume next
strSQL = "SELECT * FROM no_table_here"
Set objRS = Server.CreateObject("ADODB.RecordSet")
```

```
objRS.Open strSQL, objConn, adOpenStatic, adLockReadOnly
%>
<h3>Fehlerausgabe des ASP-<code>Err</code>-Objekts</h3>
<%
if Err.number <> 0 then
   echo Err.description & "<br>Fehlernummer: " & Err.number
   Err.Clear
else
   echo "Es traten keine Fehler auf."
end if
%>
<h3>Fehlerausgabe der ADO-<code>Error</code>-Kollektion</h3>
<%
for each error in objRS.ActiveConnection.Errors
   echo "<b>Beschreibung</b>: " & error.Description & "<br>"
   echo "<b>NativeError</b>: " & error.NativeError & "<br>"
   echo "<b>Fehlernummer</b>: " & error.Number & "<br>"
   echo "<b>Fehlerquelle</b>: " & error.Source  & "<br>"
   echo "<b>SQL-Status</b>: " & error.SQLState & "<br>"
next
%>
```

Listing 3.42: check_errors.asp, Abfangen und Ausgeben von Fehlern

Der SQL-Status ist manchmal recht aufschlussreich. Ich habe aus den Fehlercodes der Dokumentation eine Datei kompiliert, die alle Fehlercodes und die (englischen) Beschreibungen als `Dictionary`-Kollektion abbildet. Sie können dann `SQLState` folgendermaßen verwenden:

```
echo "<b>SQL-Status</b>: " & error.SQLState & " = "
     & sql_error(error.SQLState) & "<br>"
```

Die komplette Fehlerliste finden Sie in Anhang A.5 Fehlercodes in ADO ab Seite 465.

Die verschiedenen Fehler haben unterschiedliche Beschreibungen, die auf die tatsächliche Ursache hinweisen. Möglicherweise ist es für das Debuggen von Skripten sinnvoll, mehrere Varianten präsentiert zu bekommen. Die Dateien ERRORS.SQL.INC.ASP und ERRORS.ADO.INC.ASP enthalten die entsprechenden Übersetzungen der Fehlernummern in Beschreibungen (in Englisch). Die Anwendung ist einfach, wie das folgende Beispiel zeigt:

```
<!-- #include file="errors.sql.inc.asp" -->
<!-- #include file="errors.ado.inc.asp" -->
<%
for each error in objRS.ActiveConnection.Errors
   echo "<b>ADO-Beschreibung</b>: "
```

```
        & error.Description & "<br>"
   echo "<b>ADO-Fehlercodes</b>: "
        & ado_error(error.Number) & "<br>"
   echo "<b>SQL-Status</b>: " & sql_error(error.SQLState)
        & "<br>"
next
```

Listing 3.43: check_errors_c.asp: Ausgabe aller verfügbaren Fehlerbeschreibungen

Abbildung 3.11: So werden Fehler transparent: gleiche Ursache, unterschiedliche Wirkung

In diesem Fall (siehe Abbildung 3.11) mag Ihnen die deutsche Beschreibung am besten erscheinen. Bei anderen Fehlern ist dies nicht der Fall. Letztendlich bekommen Sie zuverlässige Aussagen nur durch die Auswertung aller Fehler.

4 Die Kollektionen der Objekte

Kollektionen sind Sammlungen von Objekten, die aus globalen Objekten abgeleitet werden. Auf Kollektionen können Sie die üblichen VBScript-Funktionen wie `Add`, `Count` usw. anwenden.

4.1 Fields

Voraussetzung für die Anwendung der Methoden und Eigenschaften ist ein Feldobjekt.

4.1.1 Übersicht

Methoden

- `Append`, Seite 159
- `CancelUpdate`, Seite 161
- `Delete`, Seite 161
- `Refresh`, Seite 162
- `Resync`, Seite 163

Eigenschaften

- `Count`, Seite 164
- `Item`, Seite 164

4.1.2 Methoden

Append

Append

Diese Methode fügt der Kollektion ein weiteres Feld hinzu. Die Attribute können mit übergeben werden.

```
objField.Append(name, type, [size,] [attributes])
```

Der *name* ist der künftige Name des Feldes, *type* der Datentyp (siehe Tabelle). Falls es sich um ein Zeichenkette oder Zahlenfeld handelt, lässt sich mit *size* die Größe angegeben. Die Angabe ist optional, der Standardwert wird von *type* übernommen. Feldattribute können mit *attributes* festgelegt werden.

Konstante	Wert	Beschreibung
adBigInt	20	8-Byte-Ganzzahl mit Vorzeichen
adBinary	128	Binärzahl
adBoolean	11	Boolescher Wert
adBSTR	8	Unicode-Zeichenkette, die mit /0 endet
adChar	129	Zeichenkette
adCurrency	6	Währung
adDBDate	133	Datum (yyyymmdd)
adDBTime	134	Zeitwert (hhmmss)
adDBTimeStamp	135	Datum und Zeit (yyyymmddhhmmss.milliardstel)
adDecimal	14	Exakter numerischer Wert
adDouble	5	Doppelt genaue Gleitkommazahl
adEmpty	0	Kein Wert
adError	10	32-Bit-Fehlercode
adGUID	72	Globale einmalige ID
adIDispatch	9	Zeiger auf die ID eines OLE-Objekts
adInteger	3	4 Byte Integer mit Vorzeichen
adIUnknown	13	Zeiger auf IUnknown eines OLE-Objekts
adLongVarBinary	205	Langer Binärwert
adLongVarChar	201	Lange Zeichenkette
adLongVarWChar	203	Durch /0 begrenzte Zeichenkette
adNumeric	131	Exakter Zahlenwert
adSingle	4	Einfache Gleitkommazahl
adSmallInt	2	2 Byte Integer mit Vorzeichen
adTinyInt	16	1 Byte Integer mit Vorzeichen
adUnsignedBigInt	21	8 Byte Integer ohne Vorzeichen
adUnsignedInt	19	4 Byte Integer ohne Vorzeichen
adUnsignedSmallInt	18	2 Byte Integer ohne Vorzeichen
adUnsignedTinyInt	17	1 Byte Integer ohne Vorzeichen
adUserDefined	132	Benutzerdefinierter Typ
adVarBinary	204	Binärwert
adVarChar	200	Zeichenkette
adVariant	12	OLE-Variante
asVarWChar	202	Unicode-Zeichenkette mit /0 beendet
adWChar	130	Unicode-Zeichenkette mit /0 beendet

Eine praktische Anwendung der Fields-Kollektion wurde bereits mehrfach gezeigt: die Ausgabe der Kopfzeile für eine Tabelle ohne konkrete Angabe der Namen. Als einfache Liste könnte das folgenermaßen aussehen:

```
for each e in objRS.Fields
   echo e.Name & " | "
next
```

Dabei wird die Eigenschaft Name des aus Fields abgeleiteten Objekts Field verwendet. Auf die Elemente der Fields-Kollektion kann aber nicht nur zugegriffen werden, es können auch neue Elemente gelöscht werden. Damit werden Tabellen um Felder erweitert. Tatsächlich führt dies nicht zu Änderungen an der Tabelle in der Datenbank, sondern nur in ADO, also auf (hier) Client-Seite. Der Anwendungsbereich liegt also eher bei der Verarbeitung temporärer Daten, die mit Hilfe von RecordSet-Objekten bequemer durchgeführt werden können. Das folgende Beispiel zeigt, wie eine Fields-Kollektion aus einer existierenden Tabelle abgeleitet und weiterverarbeitet wird.

▶ Delete

CancelUpdate

CancelUpdate

Bricht die Ausführung eines Updates für den Datensatz ab. Feldbezogene Abbrüche sind nicht möglich.

objField.CancelUpdate

Diese Methode kann nur verwendet werden, wenn *objField* von einem Record-Objekt abstammt. Bei RecordSet tritt ein Laufzeitfehler auf. Verwenden Sie statt dessen die Methode CancelUpdate des RecordSet-Objekts.

Delete

Delete

Delete entfernt ein Feld aus der Feldkollektion. Die Angabe kann mit Hilfe des Namens oder des Indizes des Feldes erfolgen.

objField.Delete(*index*)

Das Löschen erfolgt ohne weitere Rückfrage.

```
objRS.Fields.Delete("age")
objRS.Fields.Delete(2)
```

▶ Append, Seite 159

Refresh

Refresh

Diese Methode aktualisiert die Feldliste in der Kollektion. Um Daten zu aktualisieren, verwenden Sie besser die Methode `Requery` des Objekts `RecordSet`.

objFields.Refresh

▶ `Requery`, Seite 88

Beispiel

Das folgende Beispiel zeigt die Anwendung der Methoden `Append` der `Fields`-Kollektion und verschiedener Eigenschaften:

```
set objRS = Server.CreateObject("ADODB.RecordSet")
set objRX = Server.CreateObject("ADODB.RecordSet")
strQuery = "Products"
objRS.Open strQuery, objConn, adOpenStatic, adLockOptimistic
echo "Die Tabelle hat " & objRS.Fields.Count & " Spalten.<p>"
' Auslesen der Felder und uebertragen in virtellen RecordSet
echo "Liste der Felder der Originaltabelle:<br>"
echo "<pre>"
for each e in objRS.Fields
    echo e.Name & "|"
   objRX.Fields.Append e.Name, e.Type, e.DefinedSize,
                       e.Attributes
next
echo "</pre>"
objRX.Fields.Append "Price_DM", adDouble
objRX.Open
while not objRS.EOF
   objRX.AddNew
   for each e in objRS.Fields
      objRX(e.name) = objRS(e.name)
   next
   objRX("Price_DM") = objRS("UnitPrice") * 2.24
   objRX.Update
   objRS.MoveNext
wend
objRS.Close
objRX.MoveFirst
echo "Liste der Felder der tempor&auml;ren Tabelle:<br>"
echo "<pre>"
for each e in objRX.Fields
    echo e.name & "|"
next
echo "</pre>"
```

```
echo "<hr noshade size=2>"
echo "<b>UnitPrice = Price_DM</b><br>"
while not objRX.EOF
   echo "US$" & formatnumber(objRX("UnitPrice")) & " = "
   echo "DM " & formatnumber(objRX("Price_DM")) & "<br>"
   objRX.MoveNext
wend
```

Listing 4.1: Fields.Append.asp: Umgang mit temporären RecordSet-Objekten zur Speicherung von Daten außerhalb der Datenbank

Die Eigenschaften des `Field`-Objekts wurden bereits in Abschnitt 3.5 Field ab Seite 129 beschrieben. Die folgenden Eigenschaften beziehen sich dagegen auf die `Fields`-Kollektion.

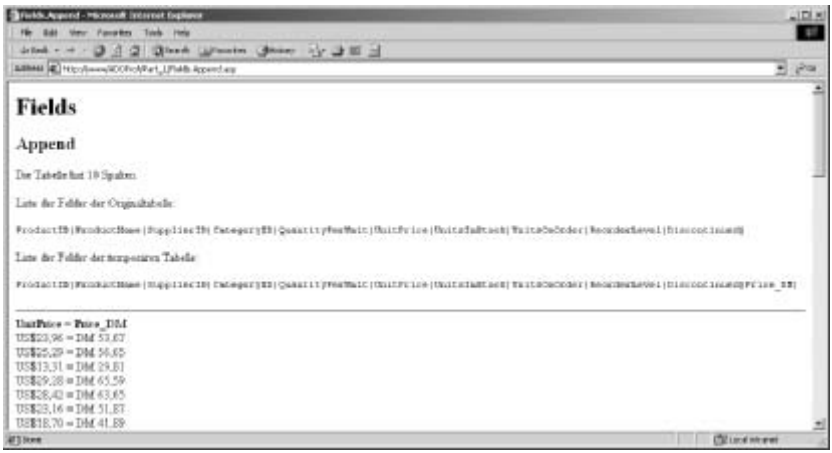


Abbildung 4.1: Ausgabe von Listing 4.1

Resync

`Resync` arbeitet analog der `Resync`-Methode des `RecordSet`-Objekts. Die Anwendung darf nur für `Field`-Objekte erfolgen, die von `Record` abgeleitet wurden.

Resync

objField`.Resync long` *lngResync*

Der Parameter *lngResync* kann folgende Werte annehmen:

Konstante	Wert	Beschreibung
`adResyncAllValues`	2	Daten werden überschrieben und wartende Änderungen gehen verloren (Standardwert).
`adResyncUnderlying-Values`	1	Daten werden nicht überschrieben und wartende Änderungen gehen nicht verloren.

4.1.3 Eigenschaften

Count

Count

Mit `Count` lässt sich die Anzahl der Elemente der Kollektion festgestellen.

```
long lngFields = objField.Count
```

Verwenden Sie `Count`, um Feldlisten mit `For...Next` zu durchlaufen. Einfacher ist jedoch in den meisten Fällen der Zugriff mit `For Each...Next`.

Item

Item

Diese Eigenschaft gibt den Feldinhalt wieder.

```
data = objField.Item(variant Index)
```

Dabei handelt es sich um die Standardeigenschaft. Da `Fields` die Standardkollektion ist, gibt es mehrere verkürzte Schreibweisen:

```
objField.Item(2)
objField.Item("name")
objRS.Fields.Item(2)
objRS.Fields.Item("name")
objRS.Fields("name")
objRS("name")
```

Der Index kann der numerische Index des Feldes oder sein Name sein.

▶ `RecordSet`, Abschnitt 3.2, Seite 69

4.2 Properties

Das `Property`-Objekt enthält Angaben zur Art des Objekts. Auch dieses Objekt kann Kollektionen bilden, die in diesem Abschnitt erläutert werden. `Property`-Objekte kommen als Teil folgender Objekte vor:

- `Connection`
- `Command`
- `RecordSet`
- `Field`

Anwendungsbeispiele finden Sie beim Objekt `Property`, Abschnitt 3.6 ab Seite 136.

4.2.1 Übersicht

Eigenschaften

- `Count`, Seite 165
- `Item`, Seite 165

4.2.2 Methoden

Die `Properties`-Kollektion hat keine Methoden.

4.2.3 Eigenschaften

Count

Mit `Count` kann die Anzahl der Elemente der Kollektion festgestellt werden.

Count

```
lngFields = objProperties.Count
```

Verwenden Sie `Count`, um Feldlisten mit `For...Next` zu durchlaufen. Einfacher ist jedoch in den meisten Fällen der Zugriff mit `For Each...Next`.

Item

Diese Eigenschaft gibt den Feldinhalt wieder.

Item

```
data = objProperties.Item(variant Index)
```

Dabei handelt es sich um die Standardeigenschaft. Die Schreibweise kann deshalb auch verkürzt werden:

```
objProperties.Item(2)
objproperties.Item("name")
objProperties(2)
objproperties("name")
```

Der Index kann der numerische Index der Eigenschaft oder sein Name sein.

- `RecordSet`, Abschnitt 3.2, Seite 69

4.3 Errors

Die `Errors`-Kollektion kann aus mehreren Objekten bestehen, die Fehlerinformationen enthalten.

Wenn Sie diese Methoden und Eigenschaften nutzen möchten, müssen Sie über eine Instanz der `Errors`-Kollektion verfügen, z. B. mit der folgenden Anweisung:

```
set objErr = objConn.Errors(0)
```

objErr wird in den folgenden Syntaxdiagrammen als Variable verwendet.

4.3.1 Übersicht

Methoden

- `Clear`, Seite 166
- `Refresh`, Seite 166

Eigenschaften

- `Count`, Seite 166
- `Item`, Seite 167

4.3.2 Methoden

Clear

Clear

Diese Methode entfernt alle `Error`-Objekte aus der `Errors`-Kollektion. Diese Methode wird beim Auftreten eines Fehlers implizit aufgerufen, sodass Sie immer nur über den letzten Fehler (oder die Fehlergruppe) verfügen können.

objErr.Clear

Refresh

Refresh

Diese Methode holt aktuelle Informationen vom Provider. Damit können Sie sicherstellen, dass die enthaltenen Daten aktuell sind.

objErr.Refresh

4.3.3 Eigenschaften

Count

Count

Gibt die Anzahl der `Error`-Objekte in der Kollektion zurück.

```
long lngCount = objErr.Count
```

Item

Erlaubt den Zugriff auf ein einzelnes `Error`-Objekt in der Kollektion mit Hilfe des internen Index.

```
set objError = objErr.Item( variant index)
```

index ist ein numerischer Wert, beginnend mit 1:

```
set objError = objErr.Item(1)
```

`Item` ist die Standardeigenschaft der Kollektion, die Angabe kann deshalb entfallen.

4.4 Parameters

Eine `Parameters`-Kollektion wird mit folgender Anweisung erzeugt:

```
colParm = objComm.Parameters
```

In den folgenden Syntaxdiagrammen bezieht sich die Variable *colParm* auf diese Anweisung.

4.4.1 Übersicht

Methoden

- `AppendChunk`, Seite 168

Eigenschaften

- `Attributes`, Seite 171
- `Direction`, Seite 171
- `Name`, Seite 172
- `NumericScale`, Seite 172
- `Precision`, Seite 172
- `Size`, Seite 173
- `Type`, Seite 173
- `Value`, Seite 174

Kollektionen

- `Append`, Seite 175
- `Refresh`, Seite 176
- `Count`, Seite 176
- `Item`, Seite 177

4.4.2 Methoden

AppendChunk

AppendChunk

Mit dieser Methode werden einem `Parameter`-Objekt binäre Daten hinzugefügt.

```
colParm.AppendChunk(data)
```

data ist eine Variable, die binäre Daten enthält. In VBScript können Variablen bis zu 2 Gbytes Daten enthalten. Diese Methode entspricht `AppendChunk` der `Fields`-Kollektion.

Beispiel

Das folgende Beispiel erweitert die Northwind-Datenbank um eine weitere Tabelle und eine gespeicherte Prozedur, die Sie zuvor anlegen müssen. Folgen Sie dieser Definition:

Beispieltabelle covers

```
CREATE TABLE [dbo].[covers] (
   [id] [int] IDENTITY (1, 1) NOT NULL ,
   [cover] [image] NULL ,
   [name] [varchar] (255) NULL
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
```

Gespeicherte Prozedur Update_Cover

```
CREATE PROCEDURE Update_Cover
   @ID int,
   @CoverImage image
AS
  UPDATE covers
  SET cover = @Coverimage
  WHERE ID = @ID
```

Nun folgt das Skript zum Füllen der Tabelle mit den Bildinformationen:

```
dim adTypeBinary, s
dim strQuery, objStream, objCmd
dim f, filename, objF, fileid
adTypeBinary = 1
if open() then
' Stream-Objekt vorbereiten
set objStream = Server.CreateObject("ADODB.Stream")
objStream.Type = adTypeBinary
objStream.Open
' Datensatzobjekt vorbereiten
set objRS = Server.CreateObject("ADODB.RecordSet")
' Zuerst wird die Tabelle geloescht
' ********************************************************
```

```
' Entkommentieren Sie die folgende Anweisung nach dem ersten
' Ablaufen des Skripts, um Fehlermeldungen zu vermeiden
' ********************************************************
' objConn.Execute "DELETE FROM Covers"
' ********************************************************
' Dann wird das leere Datensatzobjekt vorbereitet
objRS.Open "SELECT * FROM Covers", objConn, adOpenDynamic,
adLockOptimistic
' Verzeichnis mit Bilder einlesen
set objFO = Server.CreateObject("Scripting.Filesystemobject")
set objF = objFO.GetFolder(Server.Mappath("images"))
' Jede Datei wird eingelesen, der Name gespeichert, die Prozedur
aktualisiert das dann
for each f in objF.Files
   ' Dateiname holen
   filename = objFO.GetBasename(f) & "."
              & objFO.GetExtensionName(f)
   ' Dateiname in Feld speichern
   objRS.AddNew ("name"), filename
next
' Datenbank aktualisieren
objRS.UpdateBatch
objRS.MoveFirst
' Vorbereiten des Kommando-Objekts und seiner Parameter
set objCmd = Server.CreateObject("ADODB.Command")
with objCmd
   .ActiveConnection = objConn
   .CommandText = "Update_Cover"
   .CommandType = adCmdStoredProc
   .Parameters.Append .CreateParameter("@ID",
                        adInteger, adParamInput, 16)
   .Parameters.Append .CreateParameter("@Coverimage",
                        adVarBinary, adParamInput, 2147483647)
end with
' Auslesen der Tabellen und aktualisieren der Bilder
while not objRS.EOF
   ' Bilddaten in Stream-Objekt laden
   objStream.LoadFromFile = Server.Mappath("images/"
                                          & objRS("name"))
   ' variable Parameter füllen
   objCmd.Parameters("@ID").Value = objRS("ID")
   objCmd.Parameters("@Coverimage").AppendChunk objStream.Read
   ' gespeicherte Prozedur ausführen
   objCmd.Execute
   objRS.MoveNext
```

```
wend
' Wenn alles funktioniert hat, kann die Tabelle komplett
' ausgelesen werden
objRS.MoveFirst
echo "<table border=1><tr>"
s = 1
while not objRS.EOF
   filename = objRS("name")
   fileid = objRS("ID")
   echo "<td>$filename</td>"
   echo "<td><img
src=""Properties.AppendChunk.Show.asp?fileid=$fileid""></td>"
   if s mod 3 = 0 then
      echo "</tr><tr>"
   end if
   s = s + 1
   objRS.MoveNext
wend
echo "</tr></table>"
end if
```

Listing 4.2: Properties.AppendChunk.asp: Auslesen von Bildern und Ablegen in einer Datenbank mit Hilfe einer gespeicherten Prozedur

Beachten Sie, dass die Bilder nicht direkt eingebunden werden können (da sie ja nicht von der Festplatte gelesen werden sollen), sondern jedes <IMG>-Tag einen Aufruf eines weiteres ASP-Skripts enthält. Dieses holt das passende Bild aus der Datenbank und sendet es mit dem richtigen HTTP-Header an den Browser:

```
set objRS = Server.CreateObject("ADODB.RecordSet")
if open() then
   objRS.Open "SELECT * FROM covers WHERE ID="
               & Request.QueryString("fileid"), objConn
   Response.ContentType = "image/gif"' Header erzeugen
   Response.BinaryWrite objRS("cover")' Bild ausgeben
else
   Response.ContentType = "text/html"
   Response.Write "Fehler: Kein Bild"
end if
```

Listing 4.3: Properties.Appendchunk.Show.asp: Ausgabe eines Bildes direkt aus der Datenbank heraus

Das hier gezeigte Verfahren ist, wenn es nur um eine einfache Bildausgabe geht, nicht besonders effizient. Interessanter sind die Manipulationsmöglichkeiten der Bilder oder die flexiblere Verwaltung in der Datenbank. Zumindest bei vielen kleineren Bildern, wie die im Beispiel verwendeten, nur wenige KByte großen JPGs, ist der SQL Server respektabel schnell.

▶ `Fields`-Kollektion, Seite 159

4.4.3 Eigenschaften

Attributes

Diese Eigenschaft beschreibt ein Element der Kollektion.

Attributes

Parameter

```
long lngAttr = colParm.Attributes
colParm.Attributes = long lngAttr
```

Für *lngAttr* kann eine der folgenden Konstanten eingesetzt werden:

Konstante	Beschreibung
`adParamSigned`	Der Parameter akzeptiert Vorzeichen.
`adParamNullable`	`Null` ist als Parameter erlaubt.
`adParamLong`	Der Parameter akzeptiert binäre Daten.

Die Konstanten können zum Schreiben mit `Or` kombiniert werden:

```
colParm.Attributes = adParamSigned Or adParamNullable
```

Beim Lesen wird entsprechend `And` verwendet:

```
lngAttr = colParm.Attributes
If lngAttr And (adParamSigned Or adParamNullable) Then
   Response.Write "Vorzeichen ist erlaubt.<br>"
   Response.Write "<pre>Null</pre> ist erlaubt.<br>"
End If
```

▶ `Size`, Seite 173

▶ `Type`, Seite 174

Direction

Diese Eigenschaft gibt an, ob die Parameter Eingabewerte, Ausgabewerte oder beides darstellen.

Direction

```
long lngDir = colParm.Direction
colParm.Direction = long lngDir
```

Den Parameter können Sie der folgenden Tabelle entnehmen:

Parameter

Konstante	Beschreibung
adParamUnknown	Richtung ist nicht bekannt
adParamInput	Eingabewert, Übergabe an eine gespeicherte Prozedur
adParamOutput	Ausgabewert, Rückgabe von einer gespeicherten Prozedur
adParamInputOutput	Ein- und Ausgabewert
adParamReturnValue	Wert wird modifiziert

Bei der Definition von gespeicherten Prozeduren werden Ausgabeparameter mit dem Schlüsselwort `OUTPUT` deklariert. Prozeduren können auch einen normalen Rückgabewert haben, dieser wird als `"RETURN_VALUE"` in der `Parameters`-Kollektion erzeugt. Für diesen Wert ist `Direction` immer `adParamOutput`.

Name

Name

`Name` ist eine Bezeichnung für die `Parameter`-Kollektion.

```
string strName = colParm.Name
colParm.Name = string strName
```

NumericScale

NumericScale

Diese Eigenschaft setzt oder liest die Stellenanzahl, wenn es sich bei dem Element der Kollektion um einen numerischen Datentyp handelt.

```
integer intScale = colParm.NumericScale
colParm.NumericScale = integer intScale
```

Der Rückgabewert entspricht der Anzahl der Dezimalstellen.

- `Precision`
- `Size`, Seite 173
- `Type`, Seite 174

Precision

Precision

Diese Eigenschaft setzt oder liest die Genauigkeit, wenn es sich bei dem Element der Kollektion um einen numerischen Datentyp handelt.

```
integer intPrec = colParm.Precision
colParm.Precision = integer intPrec
```

Der Rückgabewert entspricht der Genauigkeit der Zahl in Ziffern.

- NumericScale
- Size, Seite 173
- Type, Seite 174

Size

Size

Diese Eigenschaft setzt oder liest die Genauigkeit, wenn es sich bei dem Element der Kollektion um einen numerischen Datentyp handelt.

```
integer intSize = colParm.Size
colParm.Size = integer intSize
```

Der Rückgabewert entspricht der Anzahl Zeichen, die das Element benötigt. Beachten Sie, dass diese Eigenschaft den zulässigen Wert ausgibt, bei binären Daten also 2 147 483 647, entsprechend 2 Gbyte. Ein Feld mit dem Datentyp CHAR(8) würde entsprechend 8 zurückgeben, auch wenn nur vier Zeichen enthalten sind.

- NumericScale, Seite 172
- Precision, Seite 172
- Type, Seite 174

Type

Type

Diese Eigenschaft stellt den Datentyp eines Elements der Kollektion fest.

```
long lngType = colParm.Type
colParm.Type = long lngType
```

Die folgende Tabelle zeigt die Datentypen des SQL Servers und Access und die Zuordnung der entsprechenden Konstanten.

Konstante	SQL Server	Access
adVarBinary	Binary	Binary (Binär)
adVarBinary	VarBinary	
adVarChar	Char	Text
adVarChar	VarChar	
adDBTimeStamp	DateTime	
adDBTimeStamp	SmallDateTime	
adSingle	Float	Single (Fließkommazahl)

Konstante	SQL Server	Access
adSingle	Real	
adInteger	Int	Long
adSmallInt	SmallInt	Integer (Ganzzahl)
adUnsignedTinyInt	TinyInt	Byte
adCurrency	Money	Currency (Währung)
adCurrency	SmallMoney	
adBoolean	Bit	Y/N (JA/NEIN)
adVarBinary	TimeStamp	
adVarChar	Text	Memo
advarBinary	Image	OLE-Object
adDouble		Double
adDate		Date/Time (Datum/Uhrzeit)
adGUID		Replication ID
adEmpty		Value

Die Zuordnung der Datentypen ist unter Umständen nicht eindeutig. Wenn Sie feststellen möchten, welche Werte ADO annimmt, lesen Sie die Eigenschaft und nutzen Sie die vorgeschlagenen Werte.

Value

Value

Diese Eigenschaft liest oder setzt den Inhalt eines Elements. Sie können das auch für binäre Werte nutzen.

```
data = colParm.Value
colParm.Value = data
```

Die Anwendung auf ein Datensatzobjekt könnte folgendermaßen aussehen:

```
objRS.Parameters("name").Value = "Krause"
```


- Type, Seite 174
- RecordSet-Objekt, Abschnitt 3.2 ab Seite 69

4.4.4 Kollektionen

Die Parameter-Kollektion kann selbst wiederum eine Kollektion enthalten, wenn eine gespeicherte Prozedur mehrere Werte zurückgibt. Das passiert, wenn sowohl Parameter als auch Datensätze erzeugt werden. Die folgenden Methoden und Eigenschaften entsprechen den Standardmethoden und

-eigenschaften aller Kollektionen. Sie sollen anhand eines Beispiels erläutert werden.

Die folgende gespeicherte Prozedur gibt einen Wert, einen Parameter und ein Datensatzobjekt zurück.

Rückgabewerte einer gespeicherten Prozedur

```
CREATE PROCEDURE artikelname_by_lang
   @lang char(2)
   @lcode int OUTPUT
AS
BEGIN
   SELECT code AS lcode FROM langtable WHERE lang = @lang
   SELECT *, COUNT(id) AS anum FROM artikel WHERE lang = @lang
   SELECT @lcode = code
   SELECT @ok = anum
   RETURN @ok
END
```

Diese Prozedur gibt drei Elemente zurück:

- Einen Datensatz mit den Artikeln der ausgewählten Sprache
- Den Ländercode aus der Tabelle *langtable*
- Die Anzahl der gefundenen Elemente als Rückgabewert

Der folgende Code zeigt, wie Sie diese Prozedur aufrufen:

```
objComm.CommandText = "artikelname_by_lang"
objComm.CommandType = adCmdStoredProc
arrParam = Array("DE", intArtikelanzahl)
Set objRS = objComm.Execute ( , arrParam)
```

Der Datensatz ist nun über *objRS* verfügbar. Die anderen beiden Werte stehen in zwei `Parameter`-Elementen zur Verfügung. Der Rückgabewert wird in `"RETURN_VALUE"` übergeben, der Parameter *lcode* in `"lcode"`.

Methoden

Append

Append

Diese Methode fügt an die `Parameter`-Kollektion ein weiteres Objekt an. Dieses lässt sich vorher mit den bereits zuvor beschriebenen Eigenschaften modifizieren.

colParm.Append *objParm*

Das Objekt *objParam* wird mit `CreateParameters` erzeugt:

```
Set objParam = objComm.CreateParameters
objParam.Name = "Laendercode"
objParam.Type = adVarChar
objParam.Direction = adParamInput
objParam.Size = 2
objComm.Parameters.Append objParam
```

Dies ist ein fiktives Beispiel, kein vollständiger Code.

Delete

Delete

Entfernt ein Parameterobjekt aus der `Parameters`-Kollektion

```
colParm.Delete(variant index)
```

index ist der Name oder die Nummer des Objekts in der Kollektion.

Refresh

Refresh

Die Objekte der Kollektion werden mit dieser Methode aktualisiert.

```
colParm.Refresh
```

Dabei werden auch die Eigenschaften der Parameter gesetzt. Im Gegensatz zur Benutzung der entsprechenden Eigenschaften im Skript erfolgt der Abruf über den Provider langsamer. Eine typische Anwendung ist der erstmalige Abruf der Parameter mit `Refresh` und nachfolgend eine Kopie und Modifikation der Objektvariablen. Diese Methode ist bei der mehrfachen Verwendung eines parameterisierten Kommandos etwas schneller als die fortlaufende, direkte Übergabe der Argumente. In »klassischen« ASP-Skripte, wo nur eine Abfrage pro Ablauf erfolgt und verschiedene Nutzer unterschiedliche Abfragen produzieren, ist der Einsatz nicht besonders sinnvoll.

Eigenschaften

Count

Count

Diese Eigenschaft gibt die Anzahl Objekte in der `Parameters`-Kollektion zurück.

```
long lngNumber = objParam.Count
```

Statt `Count` kann auch `For Each ... Next` verwendet werden, um alle Objekte der Kollektion zu erreichen.

Item

Item

Diese Eigenschaft greift auf ein Objekt der Kollektion zu.

```
varElement = objParam.Item(variant index)
```

`Item` ist die Standardeigenschaft des Objekts und muss deshalb nicht angegeben werden. Die folgenden Codes sind äquivalent:

```
objRS.Parameters.Item(1)
objRS.Parameters(1)
objRS.Parameters.Item("feldname")
objRS.Parameters("feldname")
```

5 Spezielle Techniken

Dieses Kapitel behandelt einige seltener angewandte Methoden und Techniken, die erst in neueren Versionen von ADO unterstützt werden.

5.1 Data Shaping

Ein typische Situation beim Umgang mit realen Daten ist die Verknüpfung von Tabellen. Fast schon klassisch ist das Problem der Bestelltabelle. Ein Kunde hat mehrere Bestellungen, die in einer separaten Tabelle liegen. Wenn die Bestellungen aller Kunden angezeigt werden sollen, müssen beide Tabellen gelesen werden. Damit das funktioniert, werden Fremdschlüssel definiert.

Nachteilig wirkt sich dabei aus, dass die dafür normalerweise verwendete Abfrage mit `JOIN` keine Aktualisierung zulässt. Das Schreiben der Bestellungen neuer Kunden verlangt immer zwei Tabellenzugriffe. Wenn zu jeder Bestellung mehrere weitere Bestellinformationen in einer dritten Tabelle liegen – auch dies ist in größeren Szenarios typisch –, ist der Aufwand zum Schreiben des Codes und auch für die Datenbank selbst bei der Ausführung enorm. Auch die Trennung in mehrere Datensatzobjekte und die Verarbeitung im Code ist keine Lösung. Hier würde die Abarbeitung noch ineffizienter erfolgen.

Data Shaping löst diese Probleme in einem Zuge. Es lassen sich beliebig tiefe Hierarchien aufbauen, die alle nötigen Beziehungen enthalten. Sie können über alle Ebenen Aggregat-Funktionen anwenden und – das ist besonders interessant – Updates ausführen.

5.1.1 Wie Data Shaping funktioniert

Data Shaping geht zwar mit hierarchischen Beziehungen zwischen Tabellen um, verlangt aber keineswegs nach bestehenden Fremdschlüsseln. Tatsächlich muss man sich von dem Gedanken lösen, dass alle Prozesse in der Datenbank ablaufen müssen. Data Shaping ist ein Funktion, die ADO mitbringt. Entsprechend wird ein spezieller Datenprovider verwendet, der die Umsetzung in SQL übernimmt. Das Konzept stammt übrigens aus einer früheren Version von Foxpro und wurde erst mit Version 2.0 in ADO integriert.

Da ADO auf OLEDB aufbaut, wird für Data Shaping auch OLEDB verwendet. Tatsächlich stellt OLEDB den entsprechenden Provider zur Verfügung – einschließlich einer eigenen Abfragesprache, mit der Hierarchien verwaltet

werden können. Die folgende Grafik zeigt die Zusammenhänge zwischen OLEDB, ADO und dem Data Shape-Provider:

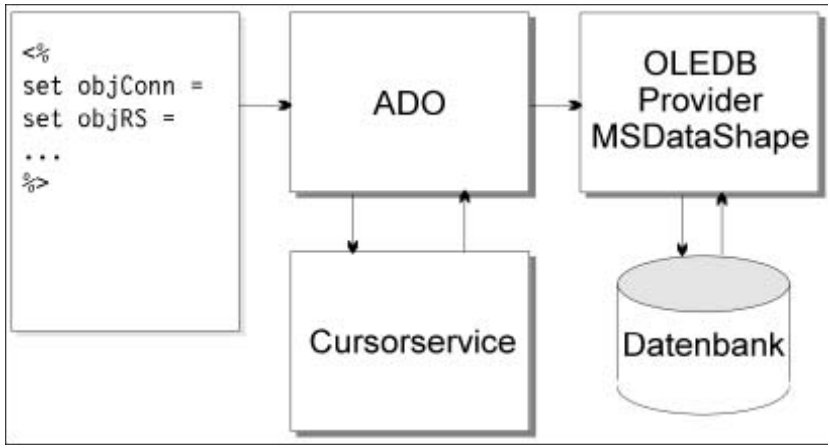


Abbildung 5.1: Zusammenhang zwischen OLE DB, ADO und dem Data Shape-Provider

Mit ASP schicken Sie jetzt ein `SHAPE`-Kommando an ADO. Dabei wird der *Data Shape-Provider* als *Service Provider* angegeben. OLEDB ist nun der sogenannte *Data Provider*. ADO schickt das `SHAPE`-Kommando an den *Data Shape-Provider* und dieser benutzt den angegebenen *Data Provider*, um die Daten aus der Datenbank anzufordern. Die vorverarbeiteten Daten werden an den *Cursor Service* übergeben, der für die Navigation und Änderungen im Datensatzobjekt zuständig ist. Die OLEDB-Ebene wird verlassen, und ADO bekommt die Datensätze zur Verwendung übergeben. Die ASP-Applikation kann die Daten auslesen, darstellen und modifizieren, so wie Sie es bisher auch gewohnt waren.

Syntax des Data Shape-Providers

Bisher wurde eine OLEDB-Verbindung nach folgendem Muster verwendet:

```
strConnection = "Provider         = SQLOLEDB;
                 Data Source      = WWW;
                 Initital Catalog = NorthWind;
                 User Id          = sa;
                 Password         = "
```

Der Daten-Provider ist in diesem Fall SQLOLEDB, also die OLEDB-Schnittstelle zum SQL Server. Der Data Shape-Provider schiebt sich nun dazwischen, was auch die Verbindungszeichenkette reflektiert:

```
strConnection = "Provider         = MSDataShape;
                 Data Provider    = SQLOLEDB;
                 Data Source      = WWW;
                 Initital Catalog = NorthWind;
```

```
                User Id          = sa;
                Password         = "
```

Wenn Ihnen diese Version nicht aussagekräftig genug erscheint, können Sie auch den Provider mit der Eigenschaft `Provider` des `Connection`-Objekts direkt setzen:

```
set objConn = Server.CreateObject
objConn.Provider = "MSDataShape"
strConnection = "Data Provider    = SQLOLEDB;
                Data Source       = WWW;
                Initital Catalog = NorthWind;
                User Id           = sa;
                Password          = "
objConn.Open strConnection
```

Datentyp

Neuer Datentyp

Der Inhalt eines Shape-Datensatzes besteht aus den Feldern der Haupttabelle und einem weiteren Feld, dem *Chapter.*

adChapter

Der Datentyp dieser Untertabellen ist `adChapter`.

Syntaxbeispiel

Einführung in die Shape-Sprache

Die Shape-Sprache ist ebenso mächtig wie primitiv, der Lernaufwand dürfte sich in Grenzen halten. Eine SQL-Abfrage könnte nun folgendermaßen aussehen:

```
strSQL = "SHAPE {SELECT * FROM Customers} AS CustomTable
          APPEND ({SELECT * FROM Orders} AS OrderTable
          RELATE CustomerID TO CustomerID) AS Orders
```

SHAPE
APPEND
RELATE

Die Abfrage besteht also aus drei Teilen:

- `SHAPE` definiert die Haupttabelle.
- `APPEND` definiert die Untertabelle.
- `RELATE` beschreibt die Beziehung. Hier wird immer zuerst die Haupttabelle genannt. Auch wenn die Spaltenbezeichnungen in Haupt- und Untertabelle gleich sind, muss keine Tabelle angegeben werden.

Aliase

Die eigentliche Abfrage wird in geschweifte Klammern gesetzt, wobei hier alle `SELECT`-Anweisungen zulässig sind. Die Haupttabelle bekommt den Alias *CustomTable*, die Untertabelle den Alias *OrderTable*. Die Spalte in *CustomTable*, deren Felder jeweils eine Untertabelle enthalten, trägt den Namen *Orders*.

Interne Namen

Wenn Sie keine Aliase für die Relationen benennen, erhalten diese die Bezeichnungen `Chapter1`, `Chapter2` usw.

5.1.2 Beispiel

Das folgende Beispiel zeigt eine komplexe Abfrage der Northwind-Datenbank: die vollständige Liste aller Bestellungen, sortiert nach Kunden und mit Angaben zu Kundennamen, Bestelldatum, Artikelname und Preis. Damit werden vier Tabellen verknüpft. Das Shape-Kommando verzichtet auf Aliase für die Tabellen mit den Ursprungsdaten, benennt aber Relationen um.

```
s= ""
s=s & "SHAPE {SELECT * FROM Customers}"
s=s & "APPEND ( "
s=s & "  (SHAPE {SELECT * FROM Orders}"
s=s & "   APPEND ("
s=s & "     (SHAPE {SELECT * FROM [Order Details]}"
s=s & "      APPEND ({SELECT * FROM Products}"
s=s & "      RELATE ProductID TO ProductID) AS product_child)"
s=s & "   RELATE OrderID TO OrderID) AS order_child) "
s=s & "RELATE CustomerID TO CustomerID) AS customer_child"
set objCustomer = Server.CreateObject("ADODB.RecordSet")
objCustomer.Open strShape, objConn
while not objCustomer.EOF
  echo "<b>KUNDE</b>: " & objCustomer("CompanyName")
  echo ", " & objCustomer("ContactName") & "<br>"
  set objOrder = objCustomer("customer_child").value
  echo "<ul>"
  while not objOrder.EOF
    echo "<li><b>BESTELLUNG VOM</b>: "
          & objOrder("OrderDate") & "<br>"
    echo "<ul>"
    set objDetail = objOrder("order_child").value
    while not objDetail.EOF
       set objProduct = objDetail("product_child").value
       echo "<li>"
       echo "<b>" & objDetail("Quantity") & "</b> St&uuml;ck "
       while not objProduct.EOF
          echo "<b>" & objProduct("ProductName") & "</b>"
          echo " zum Preis von je DM "
               & formatnumber(objProduct("UnitPrice"))
          objProduct.MoveNext
       wend
       objDetail.MoveNext
    wend
    echo "</ul>"
    objOrder.MoveNext
  wend
```

```
  echo "</ul>"
  objCustomer.MoveNext
wend
```

Listing 5.1: DataShape.Orders.asp: Ausgabe einer komplexen Beziehung

Wie es funktioniert

Wie funktioniert das nun? Das Shape-Kommando stellt alle Relationen bereit. Zuerst wird das umgebende Kommando zum Abrufen der Kundentabelle angegeben. Damit ist klar, dass die Ausgabe nach Kunden erfolgt:

```
SHAPE {SELECT * FROM Customers}
   APPEND (
   …
   RELATE CustomerID TO CustomerID) AS customer_child
```

Der Zugriff auf die nächstinnere Ebene mit den Bestellinformationen erfolgt dann mit:

```
set objOrder = objCustomer("customer_child").value
```

Die Bestellinformationen werden folgendermaßen definiert:

```
(SHAPE {SELECT * FROM Orders}
   APPEND (
   …
   RELATE OrderID TO OrderID) AS order_child)
```

Der Zugriff erfolgt über den Alias `order_child`:

```
set objDetail = objOrder("order_child").value
```

In den Bestellinformationen stehen die Produktnummern, die nun noch verknüpft werden müssen:

```
(SHAPE {SELECT * FROM [Order Details]}
   APPEND ({SELECT * FROM Products}
   RELATE ProductID TO ProductID) AS product_child)
```

Der Zugriff auf diese Details erfolgt mit folgendem Alias:

```
set objProduct = objDetail("product_child").value
```

Das Ergebnis ist durchaus überzeugend. Der Code ist ausgesprochen kompakt, von den wenigen Zeilen kümmern sich einige sogar um eine ansprechende Gestaltung.

Abbildung 5.2: Ausgabe des Skripts aus Listing 5.1

5.1.3 Weitere Techniken

Bei der Ausführung des letzten Skripts werden Sie eine verhältnismäßig lange Wartezeit bemerkt haben. Die vierfache Verschachtelung benötigt einiges an Rechenleistung. Nun ist es nicht unbedingt sinnvoll, alle Daten aller Kunden anzeigen zu lassen. Normalerweise dürften Sie sich nur für eine bestimmte Bestellhistorie interessieren. Die Abfragen sind deshalb parametrisierbar. Abgesehen von der festen Verknüpfung mit `WHERE`-Bedingungen können hiermit die Parameter der `WHERE`-Bedingung freigestellt werden. Dies erfolgt durch ein ?-Zeichen im Kontext des `SHAPE`-Kommandos.

Das `SHAPE`-Kommando des letzten Beispiels könnte man auch wie folgt schreiben:

```
SHAPE {SELECT * FROM Customers
                WHERE CustomerID = '" & strCustID & "'}
APPEND(
      (SHAPE {SELECT * FROM Orders
              WHERE CustomerID = ? ORDER BY OrderDate DESC}
```

```
      APPEND(
            (SHAPE {SELECT * FROM [Order Details]}
             APPEND ({SELECT * FROM Products}
             RELATE ProductID TO ProductID) AS product_child)
        RELATE OrderID TO OrderID) AS order_child)
RELATE CustomerID TO PARAMETER 0) AS customer_child
```

Mehrere Dinge sind hier neu. Zum einen wird mit der Übergabe der Kunden-ID eine Vorauswahl getroffen. Damit die folgende Abfrage diese Vorauswahl aber auch mitbekommt, wird eine parametrisierte Abfrage eingeführt. Die zweite `WHERE`-Bedingung heißt nun:

```
WHERE CustomerID = ?
```

Das Fragezeichen wird nicht weiter aufgelöst, es ist nur die Relation zu der mit `RELATE` definierten Verbindung:

```
RELATE CustomerID TO PARAMETER 0
```

Was immer die Vorauswahl (linker Teil von *`CustomerID`*, also durch die erste `WHERE`-Bedingung ausgewählt) enthält, es wird in den `PARAMETER 0` übertragen. Dieser wird an das »0«-te Fragezeichen gesendet.

Weitere Bedingungen

Unabhängig davon sind auch zusätzliche Bedingungen möglich. Im Beispiel wird die Anzeige der Bestellungen nach dem Bestelldatum sortiert, wobei die jüngsten Bestellungen zuerst erscheinen:

```
ORDER BY OrderDate DESC
```

Hier das vollständige Listing der verbesserten Version:

```
<html>
<body>
<h1>DataShape</h1>
Bitte w&auml;hlen Sie einen Kunden aus:
<br>
<form action="<% = ASP_SELF %>" method="post">
<select name="CustID">
<%
set objRS = Server.CreateObject("ADODB.RecordSet")
if open() then
   objRS.Open "SELECT * FROM Customers", objConn
   while not objRS.EOF
      echo "<option value=""" & objRS("CustomerID") & """>"
           & objRS("CompanyName")
      objRS.MoveNext
```

```
   wend
   objRS.Close
   objConn.Close
end if
%>
</select>
<input type="Submit" value="Bestelldaten ansehen">
</form>
<%
dim strShape, objOrder, objCustomer, objProduct
dim objDetail, strCustID
' Uebermittelte Kundennummer erfassen
strCustID = Request.Form("CustID")
if open_shape() and len(strCustID) > 0 then
   s="
   SHAPE {SELECT * FROM Customers
       WHERE CustomerID = '" & strCustID & "'}"
   APPEND (
       (SHAPE {SELECT * FROM Orders
               WHERE CustomerID = ?
               ORDER BY OrderDate DESC}
        APPEND (
             (SHAPE {SELECT * FROM [Order Details]}
              APPEND ({SELECT * FROM Products}
              RELATE ProductID TO ProductID) AS product_child)
        RELATE OrderID TO OrderID) AS order_child)
   RELATE CustomerID TO PARAMETER 0) AS customer_child"
   set objCustomer = Server.CreateObject("ADODB.RecordSet")
   objCustomer.Open strShape, objConn
   while not objCustomer.EOF
   echo "<b>KUNDE</b>: " & objCustomer("CompanyName")
   echo ", " & objCustomer("ContactName") & "<br>"
   set objOrder = objCustomer("customer_child").value
   echo "<ul>"
   while not objOrder.EOF
      echo "<li><b>BESTELLUNG VOM</b>: "
           & objOrder("OrderDate") & "<br>"
      echo "<ul>"
      set objDetail = objOrder("order_child").value
      while not objDetail.EOF
         set objProduct = objDetail("product_child").value
         echo "<li>"
         echo "<b>" & objDetail("Quantity") & "</b> Stück "
         while not objProduct.EOF
            echo "<b>" & objProduct("ProductName") & "</b>"
```

```
            echo " zum Preis von je DM "
                 & formatnumber(objProduct("UnitPrice"))
            objProduct.MoveNext
         wend
         objDetail.MoveNext
      wend
      echo "</ul>"
      objOrder.MoveNext
   wend
   echo "</ul>"
   objCustomer.MoveNext
   wend
end if
%>

</body>
</html>
```

Listing 5.2: DataShape.Orders.Param.asp: Nutzung parametrisierte Abfragen

Abbildung 5.3: Skript aus Listing 5.2 in Aktion

Dieses Skript ist bedeutend schneller. Die Parametrisierung verhindert tatsächlich, dass alle Daten von der Datenbank gelesen werden. Mit ASP nützt Ihnen das nicht so viel. Wenn Sie dagegen mit Visual Basic arbeiten würden, wäre ein Durchblättern aller Sätze ausgesprochen aufwändig. Mit Hilfe der Parameter ließe sich der nötige Zeitaufwand stark reduzieren. ASP verwirft

Besonderheiten unter ASP

dagegen am Ende des Skripts alle Datensätze und baut die Abfrage auf der nächsten Seite neu auf. Zum Blättern bleibt also nur der bereits gezeigte Weg. Ein persistentes Speichern der Daten der Shape-Abfrage in einem `Stream`- oder `Record`-Objekt ist leider nicht möglich.

Berechnungen in SHAPE-Kommandos

Bei dem Beispiel mit den Bestellinformationen wäre es sinnvoll, auch die Gesamtsumme einer Bestellung anzeigen zu können. Die Erweiterung erfolgt wiederum vor allem im Shape-Kommando und erfordert nur SQL-Kenntnisse. Die Erweiterung ist zwar ein Bestandteil der Shape-Sprache, sie ist aber mit SQL praktisch identisch. Lediglich die Anordnung sieht anders aus:

```
SHAPE {SELECT * FROM Customers
       WHERE CustomerID = '" & strCustID & "'}
APPEND(
      (SHAPE {SELECT * FROM Orders
              WHERE CustomerID = ? ORDER BY OrderDate DESC}
       APPEND(
             (SHAPE {SELECT [Order Details].*,
                     Quantity * UnitPrice AS ItemTotal
                     FROM [Order Details] WHERE OrderID = ?}
              APPEND ({SELECT * FROM Products}
              RELATE ProductID TO ProductID) AS product_child)
       RELATE OrderID TO PARAMETER 0) AS order_child,
       SUM (order_child.ItemTotal) AS SumTotal,
       COUNT (order_child.OrderID) AS CountOrders)
RELATE CustomerID TO PARAMETER 0) AS customer_child,
SUM (customer_child.order_child.ItemTotal) AS OrdersTotal
```

Das vollständige Listing zeigt, wie die Werte wieder ausgelesen werden können, die `SHAPE`-Kommandos sind fett gedruckt:

```
<html>
<body>
<h1>DataShape</h1>
Bitte w&auml;hlen Sie einen Kunden aus:
<br>
<form action="<% = ASP_SELF %>" method="post">
<select name="CustID">
<%
set objRS = Server.CreateObject("ADODB.RecordSet")
if open() then
   objRS.Open "SELECT * FROM Customers", objConn
   while not objRS.EOF
      echo "<option value=""" & objRS("CustomerID") & """>"
```

```
            & objRS("CompanyName")
      objRS.MoveNext
   wend
   objRS.Close
   objConn.Close
end if
%>
</select>
<input type="Submit" value="Bestelldaten ansehen">
</form>
<%
dim s, objOrder, objCustomer, objProduct, objDetail, strCustID
' Uebermittelte Kundennummer erfassen
strCustID = Request.Form("CustID")
if open_shape() and len(strCustID) > 0 then
  s = "
SHAPE {SELECT * FROM Customers
       WHERE CustomerID = '" & strCustID & "'}"
APPEND (
     (SHAPE {SELECT * FROM Orders
             WHERE CustomerID = ? ORDER BY OrderDate DESC}
      APPEND (
             (SHAPE {SELECT [Order Details].*,
                            Quantity * UnitPrice AS ItemTotal
                     FROM [Order Details]
                     WHERE OrderID = ?}
              APPEND ({SELECT * FROM Products}
              RELATE ProductID TO ProductID) AS product_child)
      RELATE OrderID TO PARAMETER 0) AS order_child,
      SUM (order_child.ItemTotal) AS SumTotal,
      COUNT (order_child.OrderID) AS CountOrders)
RELATE CustomerID TO PARAMETER 0) AS customer_child,
SUM (customer_child.order_child.ItemTotal) AS OrdersTotal"
 set objCustomer = Server.CreateObject("ADODB.RecordSet")
 objCustomer.Open s, objConn
 while not objCustomer.EOF
   echo "<b>KUNDE</b>: " & objCustomer("CompanyName")
   echo ", " & objCustomer("ContactName") & "<br>"
   echo "<b>INFO:</b> Gesamtwert aller Bestellungen: DM "
         & formatnumber(objCustomer("OrdersTotal"))
   set objOrder = objCustomer("customer_child").value
   echo "<ul>"
   while not objOrder.EOF
      echo "<li><b>BESTELLUNG VOM</b>: "
            & objOrder("OrderDate") & "<br>"
```

```
      echo "<li><b>INFO:</b> DM "
            & formatnumber(objOrder("SumTotal")) & ", "
            & objOrder("CountOrders") & " Artikel"
      echo "<ul>"
      set objDetail = objOrder("order_child").value
      while not objDetail.EOF
         set objProduct = objDetail("product_child").value
         echo "<li>"
         echo "<b>" & objDetail("Quantity") & "</b> Stück "
         while not objProduct.EOF
            echo "<b>" & objProduct("ProductName") & "</b>"
            echo " zum Preis von je DM "
                 & formatnumber(objDetail("UnitPrice"))
            echo " (Zwischensumme: "
                 & formatnumber(objDetail("ItemTotal")) & ")"
            objProduct.MoveNext
         wend
         objDetail.MoveNext
      wend
      echo "</ul>"
      objOrder.MoveNext
   wend
   echo "</ul>"
   objCustomer.MoveNext
 wend
end if
%>
</body>
</html>
```

Listing 5.3: DataShape.Order.Count.asp: Berechnungen mit Data Shaping

5.1.4 Shape im Detail

Die bisherigen Darstellungen sollten ausreichen, um klarer werden zu lassen, was Sie mit der Shape-Sprache erreichen können. Für eigene Projekte benötigen Sie etwas mehr Informationen, die in diesem Abschnitt überblicksartig zusammengefasst sind.

Aufbau des SHAPE-Kommandos

Das SHAPE-Kommando hat eine sehr klare Syntax, die aber durch vielfältige Verschachtelungen in der Praxis zu relativ unübersichtlichen Konstrukten führen kann. Der Aufbau einer Relation erfolgt nach folgenden Muster:

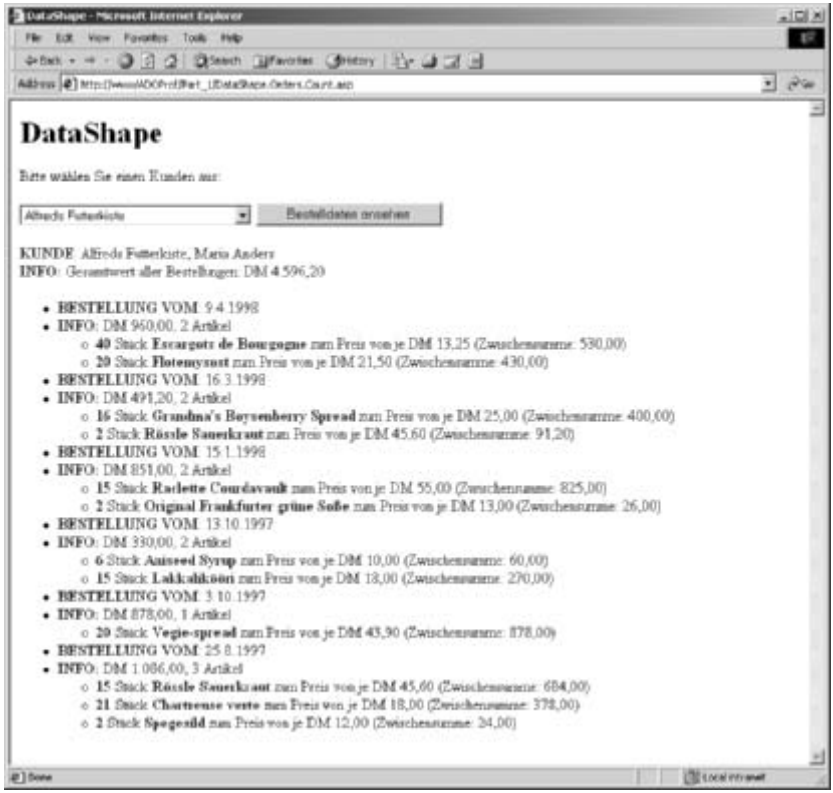


Abbildung 5.4: Anzeige diverser Zwischensummen (Ausgabe von Listing 5.3

Syntax

```
SHAPE [{parent-command} [[AS] parent-alias]]
      APPEND ( {column-list} [ [[AS] child-alias]
     [RELATE parent-column TO child-column], ... ] )
     [[AS] chapter-alias]
     [, ... ]
```

Hinter dem Schlüsselwort `SHAPE` folgt das Kommando zum Erzeugen der Haupttabelle (*`parent-command`*). Der Abfrage kann ein Alias (*`parent-alias`*) zugeteilt werden. Der Alias ist immer optional. Allerdings ist die Abfrage möglicherweise nicht eindeutig, wenn Tabellen mehrfach angesprochen werden. Dann können Sie die Benennung mit Aliasen eindeutig gestalten.

parent-command

Das Kommando kann folgenden Aufbau haben:

- Ein Provider-Kommando in geschweiften Klammern `{}`. Dieses Kommando sollte einen Datensatz zurückgeben. Normalerweise wird dort eine `SELECT`-Abfrage stehen oder der Aufruf einer gespeicherten Prozedur mit `CALL`.
- Ein weiteres, verschachteltes Shape-Kommando.
- Das Kommando `TABLE` *`tabellenname`*.

column-list Dann folgt das Schlüsselwort APPEND, mit dem der Abfrage eine weitere Spalte hinzugefügt wird. Diese Spalte enthält in jedem Feld den Verweis auf die jeweils zugeordnete Untertabelle. Die Abfrage für die Untertabelle wird mit *column-list* definiert. Der gesamte Ausdruck einschließlich der mit RELATE eingeleiteten Relation steht in runden Klammern (). Dabei kann es sich um Folgendes handeln:

- Eine SELECT-Abfrage in geschweiften Klammern.
- Der Name eines Datensatzes aus einem anderen Shape-Kommando desselben Ausdrucks.
- Ein weiteres Shape-Kommando.
- TABLE *tabellenname*.
- Eine aggregierte Spalte, wie z.B. AVG oder SUM.
- Eine berechnete Spalte (COMPUTE).
- Eine neue, leere Spalte, die mit dem Schlüsselwort NEW eingeleitet wird.

parent-column Es folgt nun das Schlüsselwort RELATE zur Definition der Beziehung, der die mit APPEND erzeugte Spalte von Untertabellen gehorcht. Hier geben Sie an der Stelle *parent-column* zuerst einen Spaltennamen der mit *parent-command* erzeugten Abfrage an.

child-column An der Stelle *child-command* steht ein Spaltenname aus der Untertabelle.

chapter-alias Die ganze Konstruktion wird als Chapter bezeichnet. ADO benennt Chapter fortlaufend als Chapter1, Chapter2 usw. Das ist nicht besonders programmierfreundlich. Sie können deshalb dem Chapter ein weiteres Alias geben.

Die ganze Konstruktion *parent-column* TO *child-column* ist nicht auf einen Ausdruck beschränkt. Wenn Sie mehrere Abhängigkeiten berücksichtigen müssen, können Sie diese als kommaseparierte Liste angeben:

```
parent1 TO child1, parent2 TO child2
```

Als Verknüpfung gilt immer UND, es müssen also alle Bedingungen erfüllt sein.

Auch die Bedingung hinter APPEND ist eine Liste. Sie können also mit APPEND nicht nur eine Spalte der Haupttabelle hinzufügen, sondern mehrere. Diese stellen dann keine Hierarchie dar, sondern sind gleichberechtigt.

Interne Arbeitsweise der nicht parametrisierten Kommandos

Wenn ADO das SHAPE-Kommando ausgeführt, fragt der Provider zuerst die Haupttabelle ab. Dann wird die Untertabelle gelesen – oder mehrere, falls eine Liste angegeben wurde. Bei normalen Relationen erfolgt nun für jeden Datensatz in der Haupttabelle die Untersuchung, ob in der Untertabelle ein passender Datensatz existiert. Was zueinander passen soll, definiert RELATE. Solche Relationen sind oft 1:n-Beziehungen. Das neue Feld in der Haupt-

tabelle kann also auf mehrere Datensätze in der Untertabelle verweisen. Der spezielle Datentyp ist `Chapter` (ADO-Konstante `adChapter`).

Wenn ohne Parameter gearbeitet wird, liest ADO dennoch immer die gesamte Untertabelle. Angezeigt werden natürlich nur die Datensätze, die die Relation erfüllen.

Arbeitsweise parametrisierter Kommandos

Der Umgang mit vielen sehr großen Datensätzen kann ineffizient sein, wenn immer die kompletten Untertabellen gelesen werden. Der Effekt tritt besonders deutlich zu Tage, wenn die Haupttabelle sehr klein ist und nur wenige Relationen die Bedingung erfüllen, während die Untertabelle aber extrem groß ist.

Bei einem parametrisierten Kommando werden schon bei der ersten Abfrage nur die Datensätze gelesen, die im speziellen Fall benötigt werden. ADO erzeugt intern aus der Parameterangabe eine weitere `WHERE`-Bedingung.

Den Einbau des Parameters erkennen Sie an folgender Syntax:

Syntax

```
SHAPE {parent-command}
   APPEND ({child-command WHERE child-column-name = ?}
           RELATE parent-column-name TO PARAMETER 0)
```

Das Fragezeichen ist ein Platzhalter für die Spalte, die in der Relation verwendet wird.

Bei der Ausführung einer solchen Abfrage wird zuerst die Haupttabelle gelesen. Dann wird die Chapter-Spalte mit `APPEND` angefügt. Erfolgt nun der Zugriff auf einen spezifischen Datensatz, so wird der Wert anstatt des Platzhalters eingesetzt und nur die in diesem Augenblick benötigten Daten werden gelesen.

ASP-Probleme

Mit ASP ist das nicht immer so einfach zu handhaben, wie z.B. mit Visual Basic. Normalerweise beendet der Provider die Verbindung am Ende eines Skripts. Das nächste Skript, das vielleicht nur ein Weiterblättern ausführt, muss die Verbindung erneut aufbauen. Damit gehen aber die bisherigen Datensätze verloren. Das erneute Lesen mit Parameter erfolgt dann nicht schneller als ohne. Wenn Sie sowieso alle Datensätze ausgeben, ergeben Parameter keinen Sinn.

Datensätze aus `SHAPE`-Kommandos besitzen keine Möglichkeit, persistent gemacht zu werden, wie dies mit normalen ADO-`RecordSet`-Objekten der Fall ist.

Hybride Kommandos

Manchmal werden Parameter nur in einige Teilen der Abfrage verwendet. Das folgende fiktive Beispiel zeigt das:

```
SHAPE {SELECT * FROM Customers}
   APPEND({SELECT * FROM Orders WHERE OrderDate = #2000-10-01}
           RELATE CustomerDate TO PARAMETER 0,
                  CustomerID TO CustomerID)
```

5.1.5 Berechnungen und Aggregierungen

Innerhalb der Abfrage können vielfältige Berechnungen angestellt werden, die mit einer Mischung aus SQL und VB.NET recht komfortabel ausfallen. Ob dies im Sinne der Praxistauglichkeit ist, muss jeder selbst beurteilen, als besonders schnell hat sich diese Methode der Verlagerung der Rechenlast in den Provider nicht herausgestellt. Die folgende Tabelle zeigt, welche Aggregatfunktionen zulässig sind:

Tabelle 5.1: Aggregatfunktionen

Aggregatfunktion	Beschreibung
`SUM(chapter-alias.column-name)`	Summe
`AVG(chapter-alias.column-name)`	Mittelwert
`MAX(chapter-alias.column-name)`	Maximum
`MIN(chapter-alias.column-name)`	Minimum
`COUNT(chapter-alias[.column-name])`	Anzahl der Elemente
`STDEV(chapter-alias.column-name)`	Statistische Standardabweichung
`ANY(chapter-alias.column-name)`	Der Wert einer Spalte, wo alle Werte der Spalte gleich sind
Berechnete Ausdrücke	
`CALC(expression)`	Berechnung mit Hilfe einer VBA-Funktion. Mögliche Funktionen entnehmen Sie der folgenden Tabelle.

Die folgende Tabelle zeigt die zulässigen VBA-Funktionen. Das ist mehr, als Ihnen VBScript bieten kann. Eine Beschreibung finden Sie in der SDK-Dokumentation.

Tabelle 5.2: VBA-Funktionen für CALC

Abs	Asc	Atn	CBool	CByte	CCur
CDate	CDbl	Chr	ChrB	ChrW	Chr$
ChrB$	CInt	CLng	Cos	CSng	CStr
Cvar	CVDate	CVErr	Date	Date$	DateAdd
DateDiff	DatePart	DateSerial	DateValue	Day	DDB

Error	Error$	Exp	Fix	Format	Format$
FV	Hex	Hex$	Hour	IIF	InStr
Int	IPmt	IRR	IsDate	IsEmpty	IsError
IsNull	IsNumeric	IsObject	LCase	LCase$	Left
LeftB	Left$	LeftB$	Len	Log	LTrim
LTrim$	Mid	Mid$	Minute	MIRR	Month
Now	NPer	NPV	Oct	Oct$	Pmt
PPmt	PV	QBColor	Rate	RGB	Right
RightB	Right$	RightB$	Rnd	RTrim	RTrim$
Second	Sgn	Sin	SLN	Space	Space$
Sqr	Str	Str$	StrComp	StrConv	String
String$	SYD	Tan	Time	Time$	Timer
TimeSerial	TimeValue	Trim	Trim$	TypeName	UCase
UCase$	Val	VarType	Weekday	Year	

5.1.6 Berechnungen mit COMPUTE

COMPUTE

Neben der spaltenweisen Berechnung mit den Aggregatfunktionen, die bereits in den Beispielen am Anfang des Abschnitts gezeigt wurden, können Berechnungen auch mit `COMPUTE` ausgeführt werden.

Syntax

```
SHAPE child-command [AS] child-alias
      COMPUTE child-alias [[AS] name], [appended-column-list]
      [BY grp-field-list]
```

Allgemein berechnet `COMPUTE` Gruppen und wendet Aggregatfunktionen auf solche Gruppen an. Das Verhalten gleicht praktisch dem des SQL-Kommandos `COMPUTE`. Eine typische Anwendung sind Zwischensummen einer Tabelle. In der Shape-Sprache führt dies zu neuen Untertabellen.

Beispiel

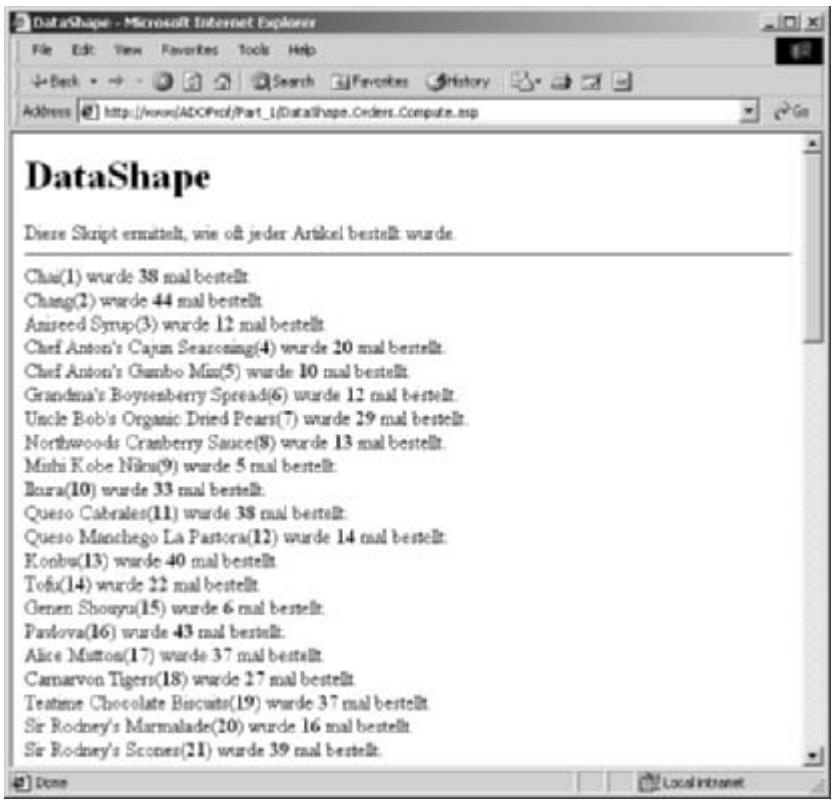
Das folgende Beispiel zeigt, wie `COMPUTE` angewendet wird.

```
strShape = "
 SHAPE(
 SHAPE {SELECT * FROM [Order Details]} AS order_details
        APPEND ({SELECT * FROM Products}
        RELATE ProductID TO ProductID))
     COMPUTE order_details,
     COUNT(order_details.Quantity) AS product_sum
     BY ProductID"
set objRS = Server.CreateObject("ADODB.RecordSet")
objRS.Open strShape, objConn
```

```
while not objRS.EOF
   set objProduct = objRS("order_details").value
   set objProductDetails = objProduct("product_details").value
   echo objProductDetails("ProductName") & "(<b>"
        & objRS("ProductID") & "</b>)"
   echo " wurde <b>" & objRS("product_sum")
        & "</b> mal bestellt.<br>"
   objRS.MoveNext
wend
```

Listing 5.4: DataShape.Orders.Compute.asp: Gruppierung von Daten mit COMPUTE

Abbildung 5.5: Ausgabe des Skripts aus Listing 5.4



Hinzufügen leerer Felder

Die bisherigen Verfahren generierten neue Felder immer dann, wenn eine Untertabelle erzeugt wurde. Sie können aber auch weitere leere Felder erstellen, um dort Daten zur Laufzeit des Skripts zu speichern. Wie bereits am Anfang erwähnt, reflektiert das Shape-Gebilde Datenänderungen (mit der Methode `Update` ausgeführt) in der Datenbank. Eigene Felder dagegen werden nicht in die Datenbank übernommen. Der Einsatz bleibt also auf

temporäre Daten beschränkt. Spinnt man diesen Gedanken weiter, so könnte man auf diese Art und Weise einen ganzen Datensatz aufbauen. Es gibt nun die Möglichkeit, auch ohne Verbindung zur Datenbank zu arbeiten – damit ist eine komplette Hierarchie auch ohne Datenbankzugriff verfügbar.

Provider NONE

Der Provider, der keiner ist, wird `NONE` genannt. Dieser Name ist praktisch nur ein Platzhalter, um der Syntax der Verbindungszeichenfolge zu genügen:

```
objConn.Open "Provider=MSDataShape;Data Provider=NONE;"
```

5.2 Datenbankzeiger

Die Zeigertechniken sind für Datenbankprogrammierer von grundlegender Bedeutung. Auch wenn ADO viele Prozesse intern abbildet und man nicht in jeder Anwendung bewusst Zeiger programmieren muss, erleichtern Kenntnisse der Arbeitsweise die Arbeit.

5.2.1 Zeigertypen

Grundlegende Zeigertechniken

Wenn Zugriffe auf Datenbanken erfolgen, wird ein interner Zeiger gebildet, der den aktuellen Datensatz adressiert. Schreib- und Leseoperationen, die nicht mit einem speziellen Auswahlbefehl verbunden sind, beziehen sich dann automatisch auf diesen aktuellen Datensatz. Wird einen Tabelle sequenziell durchlaufen, inkrementiert der interne Zeiger und stellt so einen Datensatz nach dem anderen zur Verfügung.

ADO kennt zwei grundlegende Typen von Zeigern:

- Serverseitig **Server**
- Clientseitig **Client**

Grundsätzlich ist hier die Bedeutung der Begriffe Client und Server zu beachten. Als Server wird der SQL-Datenbankserver verstanden, also der SQL Server oder MS Access. Als Client wird ADO eingesetzt, die Abstraktionsebene der Programmierumgebung. Client und Server können also, auch wenn dies die Begriffe nicht nahe legen, auf demselben Computer laufen.

Naturgemäß sind Zeiger im Server schneller, weil der Weg des Aufrufs zum Server und zurück entfällt. Allerdings unterstützt nicht jede Datenbank alle Zeigervarianten. ADO kann dies teilweise ausgleichen, wenn auch zu Lasten der Leistung.

Zugriffstypen

Eng im Zusammenhang mit Zeigern stehen auch die Zugriffstypen. Damit wird die Art der Verriegelung der Datensätze bei parallelen Zugriffen gesteuert. Folgende Typen sind möglich:

- adOpenDynamic
- adOpenForwardOnly
- adOpenKeyset
- adOpenStatic

Zugriffsteuerung

Diese Typen beziehen sich auf die Art, wie der Zeiger bewegt werden kann. Die Zugriffssteuerung erfolgt mit:

- adLockBatchOptimistic
- adLockOptimistic
- adLockPessimistic
- adLockReadOnly

Diese Typen werden im nächsten Abschnitt genauer diskutiert.

5.2.2 Die Zeiger in ADO

Dieser Abschnitt zeigt die Berührungspunkte mit Zeigern in ADO, die verwendeten Konstanten und deren Bedeutung.

Überblick über die Zeigertypen (CursorType)

Vortwärtszeiger: ad OpenForwardOnly

Der Standardzeiger für serverseitige Zeiger ist der Vorwärtszeiger, der mit `adOpenForwardOnly` erzeugt wird. Dieser Zeiger kann sich in einer Tabelle nur vorwärts bewegen. Entsprechend ist die einzige zulässige Methode zum Bewegen des Zeigers `MoveNext`. Da das Auslesen von Tabellen ohne weitere Bearbeitung sehr viel häufiger vorkommt als das Ändern von Werten, ist diese Festlegung sinnvoll. Wenn Sie sich rückwärts durch eine Tabelle bewegen möchten, können Sie vorher ein Sortierkriterium festlegen. Der Zeiger bewegt sich ja nicht direkt in der Tabelle, sondern in der temporären Ergebnistabelle der Anfrage. Es ist intelligent, den SQL-Ausdruck so zu modifizieren, dass trotz der Einschränkung des Vorwärtszeigers die Bewegungsrichtung verändert werden kann.

Statischer Zeiger: adOpenStatic

Für clientseitige Zeiger steht nur die Option `adOpenStatic` zur Verfügung. Dieser Zeiger hat keinerlei Verbindung mit der darunterliegenden Tabelle, unterstützt aber die Bewegung in beide Richtungen. Änderungen an den originalen Datensätzen werden nicht reflektiert. Unabhängig davon kann die Abfrage aber mit `Requery` aktualisiert werden.

Schlüsselgruppenzeiger: adOpenKeyset

Der Zeiger `adOpenKeyset` steht wiederum nur serverseitig zur Verfügung. Er verbindet die Originaltabelle mit dem Abfrageergebnis und überträgt Änderungen sofort. Zeiger können sich in beide Richtungen bewegen.

Dynamischer Zeiger: adOpenDynamic

Der Zeiger `adOpenDynamic` steht ebenfalls nur serverseitig zur Verfügung. Er verbindet die Originaltabelle mit dem Abfrageergebnis und überträgt außer Änderungen auch Lösch- und Einfügevorgänge sofort. Zeiger können sich in beide Richtungen bewegen.

Eine detaillierte Betrachtung finden Sie im Abschnitt Eigenschaften client- und serverseitiger Zeiger ab Seite 199.

Verriegelung im Überblick (LockType)

adLockReadOnly

Eine Abfrage, die mit der Zeigerart `adLockReadOnly` geöffnet wurde, kann nur gelesen werden.

adLockPessimistic

Mit `adLockPessimistic` verriegelt der Server den Datensatz, wenn der Nutzer mit der Änderung beginnt. Die Freigabe erfolgt erst wieder, wenn der Vorgang beendet wurde. Ob tatsächlich Änderungen durchgeführt wurden, ist hierbei nicht von Bedeutung.

adLockOptimistic

Mit `adLockOptimistic` verriegelt der Server den Datensatz erst, wenn die Methode `Update` aufgerufen wurde. Alle bis dahin bereits erfassten Änderungen werden nicht reflektiert.

adLock BatchOptimistic

Stehen mehrere Änderungen an, kann `adLockBatchOptimistic` eingesetzt werden. Alle Änderungen werden gespeichert und erst dann ausgeführt und in anderen Abfragen repliziert, wenn `UpdateBatch` aufgerufen wurde.

5.2.3 Die Zeigertypen

Bereits bei der Vorstellung der Konstanten zur Steuerung des Zeigertyps wurde klar, dass es viele Arten von Zeigern gibt und deren Anwendungsmöglichkeiten sehr unterschiedlich aussehen. In einigen Literaturquellen wird auch davon abgeraten, Zeiger zu verwenden. Statt dessen werden clientseitige Techniken empfohlen. Fehler im Umgang mit Zeigern äußern sich nicht unbedingt in Fehlfunktionen, sondern oft nur in Leistungseinbußen. Welche Zeigertypen welche Eigenschaften haben, verdient deshalb eine nähere Betrachtung.

Zeiger werden von ADO vielfältig unterstützt, für den ASP-Programmierer, der mit Datenbanken arbeitet, ist eine genaue Kenntnis der Techniken unbedingt notwendig. Skripte sind bei gut laufenden Sites hochfrequentiert und Lastprobleme lassen schnell die gesamte Anwendung zusammenbrechen. Dabei können die Unterschiede beim Abarbeiten eines Skripts zwischen dem einen oder anderen Zeigertyp schon den Faktor 1:1000 erreichen.

Eigenschaften client- und serverseitiger Zeiger

Grundsätzlich erlauben Zeiger den Umgang mit selbstständigen, von der Datenbank abgekoppelten Datensatzobjekten. Erst mit der Möglichkeit, Zeiger auch ohne die Datenbank zu nutzen, können Sie performante Anwendungen schreiben. OLEDB stellt dazu den so genannten Client Cursor Service (CCS) zur Verfügung. Damit können Navigationen zwischen Datensätzen ohne Zugriff auf die Datenbank ablaufen. Zeiger werden zwar auch von SQL Server bereitgestellt, die Verwendung ist aber nicht unproblematisch. Praktisch dienen Zeiger fast immer der Navigation – der Bewegung von einem Datensatz zum nächsten. Das ist mit Websites meist nicht

so gut zu realisieren, deshalb wird das Problem in ASP-Büchern nur am Rande behandelt. Meines Erachtens ist das nicht gerechtfertigt, denn damit bleiben Zeiger bei Optimierungsbetrachtungen außen vor – und die schlechte Performance der Site wird dann in der mangelnden Software oder Hardware gesucht. Dabei wäre ein optimiertes Skript allein die Lösung.

Serverseitige Zeiger

Serverseitige Zeiger gibt es in verschiedenen Ausführungen:

- *Vorwärts-Zeiger*: Dieser Zeiger liefert immer eine Zeile der Abfrage und kann nur vorwärts bewegt werden. In den meisten Fällen lassen sich Daten damit auch aktualisieren.
- *FireHose-Zeiger:* Dieser Zeiger ist auch ein Vorwärts-Zeiger, der aber zusätzlich scheibgeschützt ist. Damit können Daten nur gelesen werden. Er ist besonders schnell.
- *Statischer Zeiger*: Dieser Zeiger bildet das Abfrage-Objekt komplett und unveränderlich (statisch) ab. Sie können sich deshalb frei in alle Richtungen bewegen, auch rückwärts. Standardmäßig sind solche Zeiger schreibgeschützt.
- *Schlüsselgruppenzeiger*: Ein Schlüsselgruppezeiger enthält nur Schlüssel der abgerufenen Daten und greift darauf nach Bedarf zu. Die eigentlichen Daten sind davon völlig frei und können gelesen und aktualisiert werden. Als Schlüssel dient normalerweise der Primärschlüssel. Die Schlüsselgruppe selbst ist jedoch statisch. Wenn andere Benutzer Datensätze zur Datenbank hinzufügen, dann erscheinen diese nicht in der Abfrage, denn der Schlüssel darauf ist nicht in der Schlüsselgruppe enthalten und der Zeiger kann die neuen Datensätze nicht finden.
- *Dynamische Zeiger:* Dieser Zeiger ist ebenso ein Schlüsselgruppenzeiger. Allerdings ist auch die Schlüsselgruppe selbst dynamisch – Änderungen an den der Abfrage zugrunde liegenden Daten sind deshalb sichtbar.

Clientseitige Zeiger

Bis hierhin sind alle Zeiger auf die Unterstützung der Datenbank angewiesen. ADO holt die Schlüssel oder den aktuellen Datensatz und gibt ihn aus. Mit dem `RecordSet`-Objekt erhalten Sie Zugriff darauf. Werden jetzt zusätzliche Daten benötigt, sind weitere Zugriffe auf die Datenbank nötig. Schneller wäre es, wenn die gesamte Tabelle mit den Daten der aktuellen Abfrage übertragen würde und ADO die Navigation der Zeiger vornimmt und ausgewählte Daten bereitstellt. Nachteilig wirkt sich natürlich aus, dass sehr große Tabellen geladen werden, was möglicherweise Speicherprobleme bereitet, wenn gleichzeitig sehr viele Nutzer online sind. Clientseitige Zeiger sind also nicht in jedem Fall schneller. Allerdings ist die Umsetzung serverseitiger Zeiger in SQL Server nicht so gut gelungen, dass dies eine echte Alternative wäre.

Arbeitsweise clientseitiger Zeiger

Für den korrekten Umgang mit clientseitigen Zeigern sind einige Kenntnisse über deren Funktionsweise notwendig.

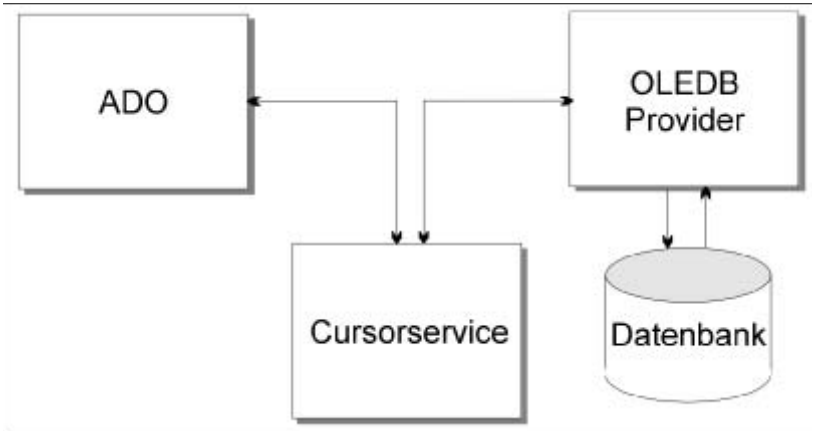


Abbildung 5.6: Datenfluss mit clientseitigen Zeigern

Wenn eine Abfrage erfolgt, wird diese von ADO direkt an den OLEDB-Provider gesendet. Bei einer normalen `SELECT`-Anweisung wird diese an die Datenbank unverändert weitergereicht. Bei Kommandos wie `adCmdTable` wird der Parameter als Tabellenname interpretiert und vom OLEDB-Provider in ein gültiges SQL-Kommando transformiert. Der Abfruf der Daten erfolgt serverseitig immer mit dem FireHose-Zeiger. Für diesen Zeigertyp ist der SQL Server optimiert. ADO speichert das gesamte Abfrageergebnis im Cursorservice (siehe Abbildung 5.6). Jetzt kann der Cursorservice entscheiden, welche Arten von Zeigern er anbietet und wie die Verwaltung erfolgt. Zugriffe auf die Datenbank erfolgen im Weiteren nur in besonderen Situationen.

Solange nur navigiert wird, muss die Datenbank nicht angesprochen werden. Die Ausgabe einer Tabelle oder die Anzeige eines einzelnen Datensatzes erfolgt deshalb sehr schnell. Dennoch ist auch der clientseitige Datensatz lässt sich aktualisieren. Dies mag nicht sofort einleuchten, weil serverseitig ja der per Definition schreibgeschützte FireHose-Zeiger verwendet wird. Was passiert nun intern? Viele Entwickler neigen auch dazu, sofort Schlüsselgruppenzeiger zu verwenden, um die Abfrage aktualisierbar zu halten. Das ist nicht notwendig. Sie können die Datensätze aktualisieren, obwohl serverseitig der FireHose-Zeiger verwendet wird. In solchen Fällen wartet ADO bis zur Ausführung der Methode `Update` oder `UpdateBatch`. Beim Aufruf werden die erfolgten Änderungen analysiert. Daraus generiert ADO eine Aktualisierungsabfrage, also eine `UPDATE`-, `INSERT` oder `DELETE`-Anweisung, die nichts mit Zeigern zu tun hat und die direkt vom DBMS ausgeführt werden kann. Die Analyse kostet zwar auch Zeit, wenn das Verhältnis zwischen Lesen und Schreiben aber stark zugunsten des Lesens ausgeprägt ist, dann hat diese Methode klare Leistungsvorteile.

Tipps für clientseitige Zeiger

Wenn Sie clientseitige Zeiger einsetzen – Sie sollten dies unbedingt tun – dann helfen die folgenden Tipps, die Applikation effizient zu gestalten:

- Versuchen Sie, die Abfrage so zu gestalten, dass so wenig Daten wie möglich abgefragt werden. Verwenden Sie Spaltenlisten anstatt des universellen Operators `*` in einer `SELECT * FROM table`-Anweisung.
- Versuchen Sie, Aktualisierungen selbst in `UPDATE`- oder `INSERT`-Anweisungen darzustellen und nicht mit `AddNew` zu arbeiten.
- Speichern Sie hochfrequentierte Datensatzobjekte, die sich nicht oder selten ändern, in XML-Objekten, sodass der Zugriff ohne Belastung der Datenbank stattfindet.

5.2.4 Verwendung clientseitiger Zeiger

Clientseitige Zeiger erreichen Sie, indem die Eigenschaft `CursorLocation` folgendermaßen gesetzt wird:

```
objRS.CursorLocation = adUseCLient
```

Diese Eigenschaft müssen Sie setzen, *bevor* das Datensatzobjekt geöffnet wird. Als Zeigertyp für das Datensatzobjekt können Sie nun nur noch `adOpenStatic` verwenden.

Vererbung von Eigenschaften

Wenn Sie die Eigenschaft `CursorLocation` nicht im `RecordSet`-Objekt, sondern in `Connection` setzen, erben alle neuen `RecordSet`-Objekte die Einstellung. Sie können den Wert dennoch überschreiben. Falls Sie Datensätze nicht explizit erzeugen, sondern mit der `Execute`-Methode des Objekts `Connection`, dann können Sie nur die Eigenschaft `CursorLocation` dieses Objekts verwenden.

Besondere Eigenschaften clientseitiger Zeiger

Clientseitige Zeiger haben einige Eigenschaften, die von serverseitigen Zeigern nicht bekannt sind:

- *Abgekoppelte Datensatzobjekte*: Sie können die Verbindung zur Datenbank nach der Datenübertragung trennen und trotzdem mit den Daten weiterarbeiten.
- *Erstellte Datensatzobjekte*: Sie können Datensatzobjekte ohne Verbindung zu einer Datenbank »zusammenbauen« und dorthinein Daten speichern.

Beide Techniken werden nachfolgend kurz vorgestellt, weil es einige Aspekte dabei zu beachten gibt, wenn Sie sich frustrierende Fehlversuche ersparen wollen.

Abgekoppelte Datensatzobjekte

Solche Objekte entstehen, wenn Sie nach dem Lesen der Daten die Verbindung trennen. Das kann in ADO erfolgen, indem die Eigenschaft `ActiveConnection` des `RecordSet`-Objekts geleert wird. Das folgende Listing zeigt, wie das aussieht:

```
set objRS = Server.CreateObject("ADODB.RecordSet")
objRS.CursorLocation = adUseClient
objRS.Open "SELECT * FROM Products", objConn, adOpenStatic
objRS.ActiveConnection = Nothing
show_table(objRS)
```

Listing 5.5: RecordSet.ClientCursor.Disconnect.asp: Abkoppeln eines Datensatzobjekts

Dies ist nicht nur sehr einfach – es ist auch der einzige Weg, dies zu tun. Sie müssen allerdings darauf achten, genau diese zwei Eigenschaften in dieser Reihenfolge einzustellen:

Reihenfolge der Eigenschaftszuweisungen beachten!

```
CursorLocation = adUseClient
```

Abfragen der Daten

```
ActiveConnection = Nothing
```

Wenn Sie die folgende Fehlermeldung beim Löschen der Verbindung erhalten, verfügen Sie nicht über einen clientseitigen Zeiger:

- Fehlertyp:
  ADODB.Recordset (0x800A0E79)
  Operation is not allowed when the object is open.
  /adoprof/Part_1/RecordSet.ClientCursor.Disconnect.asp, line 28

Abbildung 5.7: Fehler, wenn die Verbindung zu einem serverseitigen Datensatz gelöscht wird

Probleme mit Connection

Wenn ein `Connection`-Objekt verwendet wird, gibt es eine Beziehung zwischen diesem und der Eigenschaft `ActiveConnection` des `RecordSet`-Objekts. Allerdings nicht unbedingt in erwarteter Art und Weise. Wenn Sie eine Verbindung öffnen, Daten abholen und dann die Verbindung schließen, wird auch das abgeleitete `RecordSet`-Objekt geschlossen. Wenn Sie dann die Verbindung mit `ActiveConnection` trennen, haben Sie zwar wieder ein abgekoppeltes Datensatzobjekt – entgegen der Erwartung ist es aber geschlossen. Sie haben es natürlich nicht explizit geschlossen, dies hat die Methode `Close` des `Connection`-Objekts implizit getan.

Umgang mit Referenzen

Es ist möglich, Datensatzobjekte mehrfach anzusprechen. Der folgende Code-Ausschnitt zeigt dies:

```
set objRS = Server.CreateObject("ADODB.RecordSet")
objRS.CursorLocation = adUseClient
objRS.Open "SELECT * FROM Customers", objConn, adOpenStatic
set objChild = objRS
```

Sie können nun *objChild* ebenso wie *objRS* verwenden. Die Darstellung lässt vermuten, dass das Objekt *objChild* unabhängig von *objRS* ist. Genau dass ist nicht der Fall. Tatsächlich wird – Sie werden es bemerken, wenn die Datensatz-Objekte extrem groß sind – nur eine Referenz auf das Objekt im Speicher übertragen. Ein Datensatzobjekt repräsentiert nur Daten, die vom Cursorservice verwaltet werden. Dort existieren sie aber nur einmal, egal wie viele Referenzen es gibt. Schließt nun eines der Objekte die Verbindung, dann gilt dies für alle anderen Referenzen ebenso.

Der folgende Code kann deshalb nicht funktionieren:

```
set objRS = Server.CreateObject("ADODB.RecordSet")
objRS.CursorLocation = adUseClient
objRS.Open "SELECT * FROM Customers", objConn, adOpenStatic
set objChild = objRS
objRS.Close
while not objChild.EOF
  Response.Write objChild("CustomID") & "<br>"
wend
```

Mit `objRS.Close` wird auch *objChild* geschlossen. Wie ist das Problem nun zu lösen, wenn Sie genau so aber programmieren möchten? Verwenden Sie Klone. Das folgende Listing zeigt dies in der einfachsten Form. Bedenken Sie jedoch, dass die Methode `Clone` nur funktioniert, wenn Lesezeichen unterstützt werden:

```
set objRS = Server.CreateObject("ADODB.RecordSet")
objRS.CursorLocation = adUseClient
objRS.Open "SELECT * FROM Products", objConn, adOpenStatic
if objRS.Supports(adBookmark) then
   set objClone = objRS.Clone
   objRS.Close
   if objClone.CursorLocation = adUseClient the
      echo "Klone ist ein clientseitiger Datensatz!"
   end if
end if
```

Listing 5.6: RecordSet.ClientCursor.Clone.asp: Schließen eines entkoppelten Datensatzobjekts; erst wird der Klon erzeugt, dann kann die Quelle geschlossen werden.

Sie können bei der Ausführung des letzten Beispiels erkennen, dass auch der Klon nur clientseitig existiert.

Wenn Sie nun aber eine der Referenzen entfernen möchten, muss es einen anderen Weg geben. Sie verwenden in diesem Fall die bekannte Zuweisung von `Nothing`:

```
set objRS = Server.CreateObject("ADODB.RecordSet")
objRS.CursorLocation = adUseClient
```

```
objRS.Open "SELECT * FROM Customers", objConn, adOpenStatic
set objChild = objRS
objRS = Nothing
while not objChild.EOF
  Response.Write objChild("CustomID") & "<br>"
wend
```

So funktioniert es erwartungsgemäß. Hier wird das Objekt im Cursorservice nicht berührt.

Ankoppeln abgekoppelter Datensatzobjekte

Nachdem ein Datensatzobjekt eine Zeitlang abgekoppelt existiert hat, könnte es notwendig werden, die Verbindung wieder herzustellen., z.B. um nun aktualisierte Daten zurückzuschreiben. Dazu wird die Methode `Resync` verwendet.

Das folgende Listing erzeugt ein Datensatzobjekt, koppelt es ab, ändert es und fügt mit `INSERT` einen Datensatz hinzu:

```
set objRS = Server.CreateObject("ADODB.RecordSet")
objRS.CursorLocation = adUseClient
objRS.Open "SELECT * FROM Customers", objConn, adOpenStatic,
           adLockBatchOptimistic
if objRS.Supports(adBookmark)
   and objRs.LockType = adLockBatchOptimistic then
   objRS.ActiveConnection = Nothing
   echo "Datensatz enthält " & objRS.RecordCount & " Elemente"
   intKey = cint(rnd * 1000) + 1
   echo "Füge Schlüssel $intKey hinzu.<br>"
   strSQL = "INSERT INTO Customers (CustomerID, CompanyName)
             VALUES ('" & cstr(intKey) & "', 'TestName')"
   objConn.Execute strSQL
   echo "Datensatz enthält " & objRS.RecordCount & " Elemente"
   objRS.Update
end if
objRS.MoveFirst
show_table(objRS)
```

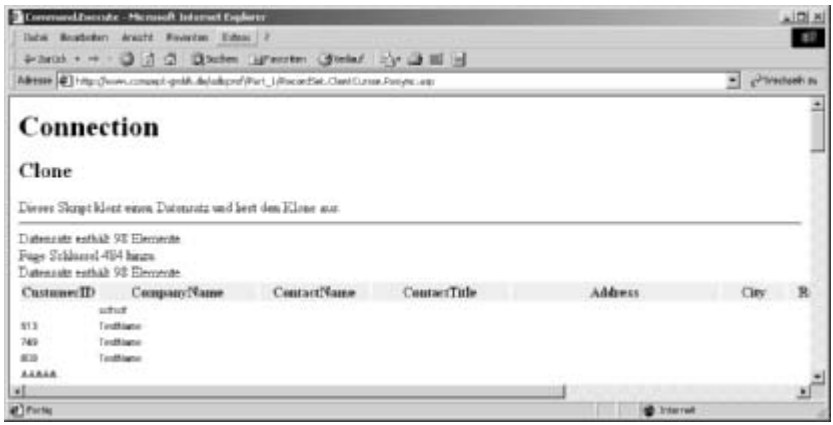
Die Tabelle enthält den neuen Datensatz nicht, denn die `INSERT`-Anweisung geht am Cursorservice völlig vorbei. Wenn Sie die neuen Daten nicht benötigen, sollten Sie dies auch so lassen. Wenn Sie die neue Tabelle aber anzeigen wollen, hilft – theoretisch – `Resync`:

```
objRS.ActiveConnection = objConn
objRS.Resync adAffectAll, adResyncAllValues
objRS.ActiveConnection = Nothing
show_table(objRS)
```

Listing 5.7: RecordSet.ClientCursor.Resync.asp: Synchronisation eines abgekoppelten Datensatzes mit der Datenquelle

Lassen Sie sich im Beispiel nicht von den per Zufallsgenerator erzeugten Schlüsseln irritieren. Das hat nichts mit dem Problem zu tun. Entscheidend ist der Umgang mit `ActiveConnection` vor und nach dem Aufruf von `Resync`. Leider funktioniert das nur, wenn Sie die Änderung mit `UPDATE` vornehmen. Das im Listing verwendete `INSERT` führt nicht dazu, dass der Datensatz erscheint:

Abbildung 5.8: Listing 5.7 funktioniert nicht richtig



In der Abbildung ist zu sehen, dass der Schlüssel 484 erzeugt wird, in der Liste aber nicht erscheint. Alle früheren Versuche sind dagegen zu sehen – `INSERT` wurde also korrekt ausgeführt.

Wie Resync funktioniert

`Resync` arbeitet auf Satzebene. Es wird also nicht die Tabelle aktualisiert, sondern nur der aktuelle Datensatz, der verändert wurde. Hinzugefügte Datensätze bleiben dagegen unsichtbar. Das Problem kann aber leicht behoben werden, indem Sie `Requery` verwenden. Diese Methode führt die ursprüngliche Abfrage erneut aus und holt damit auch die neuen Datensätze – wenn sie Bestandteil der Abfrage sind.

Daten zur Quelle aktualisieren

Das folgende Beispiel zeigt den umgekehrten Weg. Hier werden Änderungen im abgekoppelten Datensatzobjekt vorgenommen. Nun soll die Datenbank aktualisiert werden.

```
set objRS = Server.CreateObject("ADODB.RecordSet")
objRS.CursorLocation = adUseClient
objRS.Open "SELECT * FROM Customers", objConn, adOpenStatic,
           adLockBatchOptimistic
if objRS.Supports(adBookmark)
   and objRs.LockType = adLockBatchOptimistic then
   objRS.ActiveConnection = Nothing
   echo "Datensatz enthält " & objRS.RecordCount & " Elemente"
   intKey = cint(rnd * 1000) + 1
```

```
   echo "Füge Schlüssel $intKey hinzu.<br>"
   objRS.AddNew Array("CustomerID", "CompanyName"),
                Array(cstr(intKey), "TestName")
   echo "Datensatz enthält " & objRS.RecordCount & " Elemente"
end if
objRS.MoveFirst
objRS.ActiveConnection = objConn
objRS.UpdateBatch
objRS.ActiveConnection = Nothing
show_table(objRS)
```

Listing 5.8: RecordSet.ClientCursor.UpdateBatch.asp: Aktualisieren der Daten in der Quelle

Die Methode `UpdateBatch` schreibt alle Änderungen in die Datenbank. Natürlich dürfen Sie nicht vergessen, die Verbindung für diesen Moment wiederherzustellen.

Problemdiskussion

Die Anwendung der Methode erscheint zwar einfach, kann aber in komplexen Umgebungen Probleme bereiten. ADO erzeugt zur Ausführung intern passende SQL-Anweisungen, `INSERT`, `UPDATE` oder `DELETE`-Anweisungen. Hat sich der Inhalt jedoch geändert, kann SQL Server eventuell die Datensätze nicht mehr lokalisieren, die ADO aktualisieren will. Deshalb müssen Sie die möglichen Konfliktszenarien kennen und beachten. Folgende Konstellationen kann es geben:

- `UPDATE`-Probleme:
  - Das Ziel wurde gelöscht.
  - Das Ziel wurde verändert.

Diese Probleme kann die automatische Synchronisation mit `UpdateBatch` nicht lösen. Gehen Sie folgendermaßen vor:

Versuchen Sie zu erkennen, welche Art von Konflikt auftrat. Sie können Datensätze, die Konflikte verursachen, mit der `Filter`-Methode auswählen. Setzen Sie den Filter auf `adFilterConflictingRecorsdSet`.

Erneuern Sie den Datensatz, sodass die Eigenschaften `UnderLyingValue` und `Value` identisch sind. Dazu kann `Resync` verwendet werden.

Entscheiden Sie nur, ob Sie die alten Daten überschreiben oder die Aktion abbrechen möchten. `UpdateBatch` überschreibt, wenn zuvor `Resync` aufgerufen wurde, immer erfolgreich. Wenn Sie das nicht wünschen, rufen Sie `CancelBatch` auf.

Feineinstellung der Synchronisation

Die Verwaltung der lokalen Datensätze durch den Cursorservice ist vom Programmierer anpassbar. Dies ist besonders dann wichtig, wenn das Verhalten der Synchronisation nicht den Erwartungen entspricht. Drei Eigenschaften können modifiziert werden:

- `Batch Size`. Damit bestimmen Sie die Größe des Zwischenspeichers der `UpdateBatch`-Methode.
- `Update Resynch`. Damit können Sie festlegen, wie die Resynchronisation abläuft, wenn Konflikte auftreten, d.h., welche Datensätzen wirklich aktualisiert werden.
- `Update Criteria`. Hiermit lässt sich festlegen, wie exakt die Konflikterkennung arbeiten soll.

Batch Size

Normalerweise werden mehrere `UPDATE`-Operationen zusammengefasst. Der Standardwert beträgt 15. Das ist aus Performancegründen sinnvoll, führt aber zu Problemen, wenn häufig Konflikte auftreten. Die Ausführung eines Blocks scheitert, wenn in einer der Anweisungen ein Fehler auftritt. Bei großen Werten ist die Chance, dass praktisch jeder Versuch fehlschlägt, groß. Andererseits könnten Sie versucht sein, bei seltenen oder unwahrscheinlichen Konflikten den Wert höher zu setzen. Die Anweisung nutzt das `Property`-Objekt:

```
objRS.Properties("Batch Size").Value = 1
```

Damit wird jede Anweisung einzeln abgesetzt. Sehr große Werte, erfahrungsgemäß sind das Angaben über 50, benötigen viel Speicher. Sie sollten hier sorgfältig testen, ob der Server unter Last anfängt, Auslagerungsspeicher zu verwenden, und den Wert stufenweise wieder reduzieren. Befindet sich der SQL Server im Netzwerk auf einem eigenen Computer, beachten Sie, dass riesige Anweisungsfolgen Netzwerkbandbreite benötigen.

Update Resync

Die Eigenschaft `Update Resync` bestimmt, welche Datensätze von einer Änderung betroffen sind. Sie können die Werte folgendermaßen setzen:

```
objRS.Properties("Update Resync").Value = <adResyncWert>
```

Für *`adResyncWert`* setzen Sie einen der folgenden Werte ein:

- `adResyncNone`. Dieser Wert unterdrückt `UpdateBatch`. Der Einsatz dürfte nur in der Debugging-Phase sinnvoll sein.
- `adResyncAutoIncrement`. Dies ist der Standardwert. Beim Hinzufügen werden Spalten mit `AutoIncrement`-Werten erhöht und der Datensatz wird eingefügt.
- `adResyncConflicts`. Diese Eigenschaft bestimmt, dass Werte, die Konflikte verursachen, aktualisiert werden. Dies spart `Resync`, gegebenenfalls werden Werte aber auch unkontrolliert überschrieben.
- `adResyncUpdates`. Diese Eigenschaft löst generell ein Auffrischen aller Werte aus, egal ob diese Konflikte verursachen oder nicht. Sie sollten diesen Wert bei `UPDATE` setzen.
- `adResyncInserts`. Neu hinzugefügte Datensätze werden sofort sichtbar, da nach `INSERT` implizit `Requery` aufgerufen wird.

- `adResyncAll`. Hiermit werden alle Datensätze neu eingelesen, unabhängig von der verwendeten Methode.

Die Konstanten sind Bitwerte und können mit `And` kombiniert werden. Wenn Sie Datensätze sowohl hinzufügen als auch ändern, wäre folgende Angabe zu empfehlen:

```
objRS.Properties("Update Resync").Value =
   adResyncInsert And adResyncUpdate
```

Update Criteria

Diese Eigenschaft bestimmt, wie genau die Anweisungen überprüfen, ob Konflikte auftreten. Konflikte können unter anderem daran erkannt werden, ob sich Werte geändert haben. Für eine vollständige Prüfung wäre aber eine Prüfung jedes Felds nötig, was bei großen Tabellen einiges an Zeit kostet. In der Praxis wird man deshalb immer die kleinstmögliche, noch zuverlässige Methode wählen. Die Einstellung nehmen Sie mit `Update Criteria` folgendermaßen vor:

```
objRS.Properties("Update Criteria").Value = <adCriteriaWert>
```

Die möglichen Konstanten sind:

- `adCriteriaKey`. Nur Schlüsselspalten werden zur Konflikterkennung überprüft. Das ist praktisch der Wert, der keine zusätzliche Prüfung ausführt, denn die Schlüsselspalte wird zur Selektion des zu ändernden Datensatzes ohnehin ausgewählt.
- `adCriteriaAllCols`. Alle Spalten werden überprüft. Dies ist die sicherste, aber auch die langsamste Einstellung.
- `adCriteriaUpdCols`. Diese Methode prüft alle Spalten nur, wenn `UPDATE` ausgeführt wird.
- `adCriteriaTimeStamp`. Die Prüfung erfolgt anhand eines Zeitstempels des Datensatzes. Das setzt voraus, dass Zeitstempel geführt werden.

Eine Anmerkung zur Prüfung, die den Vorgang etwas transparenter macht: ADO muss hier zwischen den alten und neuen Werten unterscheiden können. Dazu wird der Wert der `Field`-Eigenschaft `UnderLyingValue` mit `Value` verglichen. Dies ist nicht unbedingt eine Methode, die besonders schnell ist.

Zugriffe auf mehrere Tabellen

Alle vorangegangenen Beispiele gingen davon aus, dass der Datensatz nur einer Tabelle entstammt. Prinzipiell können Sie in einem Datensatzobjekt aber auch das Ergebnis eines `JOIN` speichern. Dann müssen bei Änderungen mehrere Tabellen aktualisiert werden. Es gibt verschiedene Strategien, die hier auftretenden Probleme zu lösen. Die sicherste und empfehlenswerteste Methode ist, dies nicht zu tun. Schreiben Sie keine `JOIN`-Abfragen zurück. So elegant dies erscheinen mag, es birgt enorme Risiken. ADO wird bei `UPDATE` mehrere Anweisungen erstellen – für jede Tabelle eine. Nun kann es vor-

kommen, dass eine der Anweisungen erfolgreich abgeschlossen wird, andere dagegen mit Konflikten beendet werden und an der Datenbank keine Änderungen vornehmen. Schnell sind Ihre Daten inkonsistent. Auch wenn Sie alles richtig gemacht haben, muss es nicht funktionieren. So könnten Sie Einschränkungen, wie z.B. Primary-Key-Constraints, definiert haben. Diese erzwingen aber eine bestimmte Reihenfolge, in der voneinander abhängige Tabellen befüllt werden. Diese Reihenfolge können Sie mit `UpdateBatch` garantieren, indem Sie die Eigenschaft `Unique Table` setzen. Das ist aber ziemlich umständlich. Der folgende Code zeigt einen Ansatz, wenn Sie es dennoch versuchen möchten:

```
objRS.Properties("Unique Table").Value = "Tabelle1"
... Aktionen, die zu Änderungen an Tabelle1 führen
objRS.Properties("Unique Table").Value = "Tabelle2"
... Aktionen, die zu Änderungen an Tabelle2 führen
objRS.UpdateBatch
```

Hier werden zwei verbundene Tabellen *Tabelle1* und *Tabelle2* einzeln bedient, wobei davon ausgegangen wird, dass die Änderungen in *Tabelle2* nur erfolgen können, wenn *Tabelle1* zuvor in einer bestimmten Weise bedient wurde.

5.3 Betrachtungen zur Optimierung

Der folgende Abschnitt beschäftigt sich mit zwei Ansätzen, die der Optimierung dienen. Der Einsatz der hier gezeigten Techniken kann aber auch Problemstellungen des Alltags verblüffend einfach lösen.

5.3.1 Hinweise zu Leistungstests

Datenbankgestützte Applikationen sind komplizierte Gebilde. Optimierungen an einer einzigen Stelle wirken sich oft kaum aus. Sie müssen deshalb erkennen können, wo die Schwachpunkte liegen. Generell sollten Sie zuerst feststellen, welcher Teil der Applikation am meisten Zeit benötigt. Hier lohnt sich der Optimierungsaufwand. Generell bestehen Skripte aus drei Teilen, die getrennt betrachtet werden sollten:

- Datenbankzugriff
- Bearbeitung der Daten
- Ausgabe der Daten an den Nutzer

Datenbankzugriff Hier fällt vor allem die Zeit zur Generierung der Daten an. Der SQL Server muss die empfangene Anweisung interpretieren, die Daten aus den Tabellen holen, manchmal noch Berechnungen anstellen und dann die fertigen Daten auf den Weg schicken.

Bearbeitung

Bei der Bearbeitung achten Sie vor allem auf Vorgänge, die in Schleifen ablaufen. Wiederholungen wirken sich drastisch auf die Leistung aus.

Datenausgabe

Bei Internetanwendungen ist die Ausgabe der Daten an den Nutzer der zeitaufwändigste Teil. Hier werden HTML-Seiten erzeugt und an den Browser übertragen. Optimieren können Sie vor allem die Größe des erzeugten Codes. Es hilft nichts, wenn Sie bei der Abfrage der Daten einige Millisekunden herausholen, die Übertragung der Daten zum Nutzer aber mehrere Sekunden in Anspruch nimmt.

Praktische Hilfe zur Leistungsoptimierung

Die folgenden Tests fokussieren auf die Datenabfrage und den Umgang mit den fertigen Ergebnissen. Die Optimierung von HTML ist nicht Gegenstand dieses Buches. Untersucht wird vor allem das Verhalten von ADO bei unterschiedlichen Techniken des Zugriffs und der Zeigerverwaltung.

5.3.2 Benchmarks

Die folgenden Abbildungen zeigen einige Benchmarks, die verschiedene Zustände der Eigenschaften des `RecordSet`-Objekts bewerten. Verwendet wurde ein primitives Skript, das eine Zeitmessung vornimmt. Als Zeitmesser dient eine einfache ActiveX-Komponente, die Sie auch auf der Website finden. Dieses Verfahren ist notwendig, da VBScript keine Zeitmessung im Millisekundenbereich zulässt.

```
function Benchmark (objR, Par1, Par2)
   on error resume next
   objR.Open , , Par1, Par2
   if Err.Number <> 0 then
     Benchmark = Err.Description
     Err.Clear
     objRS.Close
     exit function
   end if
   objTimer.StartTiming()
   for i = 0 to 10
     while not objRS.EOF
       objRS.MoveNext
     wend
    objRS.MoveFirst
   next
   objTimer.StopTiming()
   objR.Close
   Benchmark = objTimer.TotalTime / 100
end function
```

Listing 5.9: Funktion zur Kontrolle des Verhaltens der Zeiger in einem Datensatzobjekt

```
function Benchmark (objR, Par1, Par2)
   on error resume next
   objTimer.StartTiming()
   for i = 0 to 50
      objR.Open , , Par1, Par2
      if Err.Number <> 0 then
         Benchmark = Err.Description
         Err.Clear
         objRS.Close
         exit function
      end if
      objR.Close
   next
   objTimer.StopTiming()
   Benchmark = objTimer.TotalTime / 100
end function
```

Listing 5.10: Funktion zur Messung des Verbindungsaufbaus zum Datensatz

Die DLL zur Zeitmessung binden Sie folgendermaßen ein:

```
set objTimer = Server.CreateObject("Timer.Benchmark")
```

Ergebnisse der Benchmarks mit RecordSet

Öffnen und Schließen

Das erste Chart zeigt das Öffnen und Schließen eines Datensatzes. Verwendet werden serverseitige Zeiger. In das Ergebnis fließt die Abfrage der Daten mit ein, also die Interpretation der SQL-Anweisung.

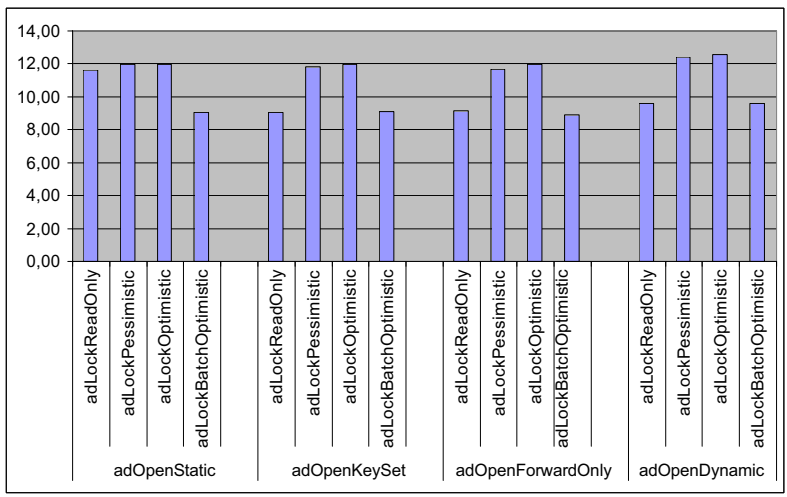


Abbildung 5.9: Öffnen und Schließen von RecordSet, adUseServer

Die nächste Variante zeigt denselben Vorgang, aber mit clientseitigen Zeigern:

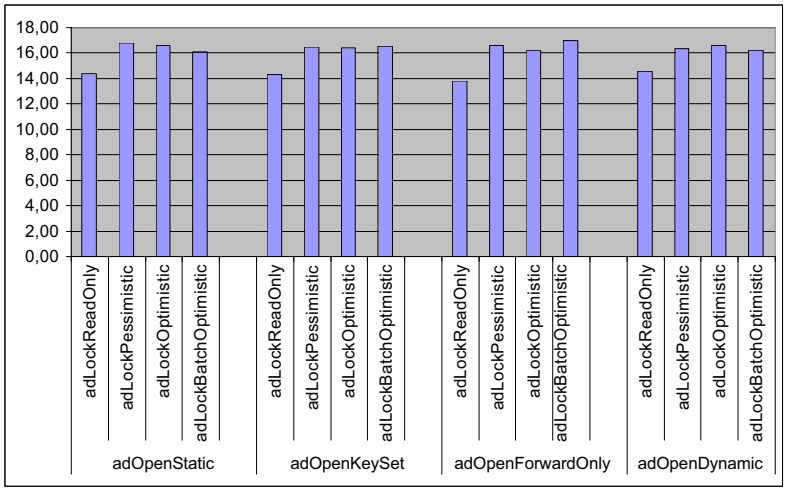


Abbildung 5.10: Öffnen und Schließen von RecordSet, adUseClient

Das Ergebnis ist keineswegs überraschend. Zum einen sind die Unterschiede zwischen den Zugriffmethoden marginal. Lesen oder Schreiben spielt aber beim Öffnen die geringste Rolle. Das die clientseitigen Zeiger langsamer sind, war auch zu erwarten. Immerhin müssen die Daten noch zum Client übertragen werden, was der serverseitige Zeiger erspart.

Durchlaufen des Datensatzes

Wenn die Daten einmal vorliegen, erfolgt der Zugriff auf die einzelnen Datensätze unabhängig von der Interpretation des Kommandos. Jetzt wirken sich die Unterschiede zwischen client- und serverseitigen Zeigern drastisch zugunsten der clientseitigen Zeiger aus.

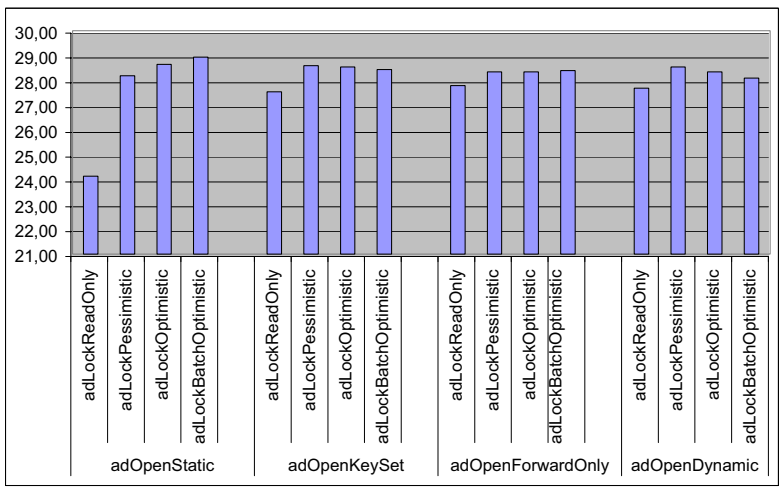
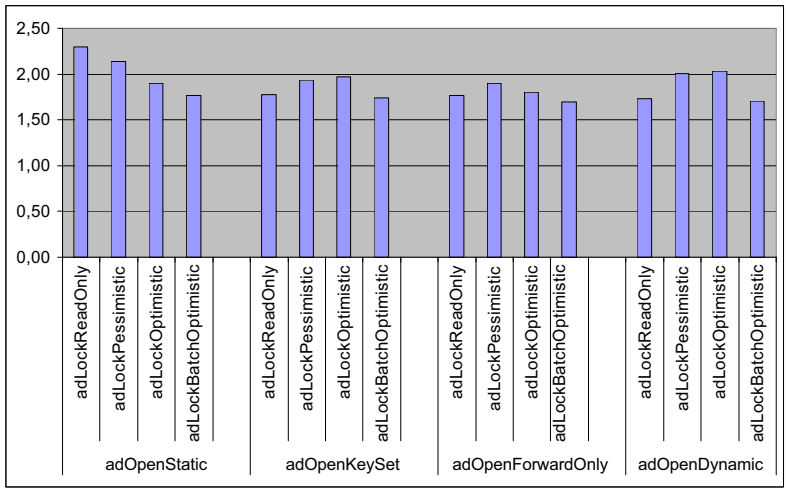


Abbildung 5.11: MoveNext mit adUseServer

Im Gegensatz dazu ist der clientseitige Zeiger eine Größenordnung schneller.

Abbildung 5.12: MoveNext mit asUseClient



Auswertung

Die Werte sind direkt vergleichbar, andere Parameter wurden nicht geändert. Generell ist eine Tendenz zu Gunsten von `adLockBatchOptimistic` festzustellen, wenn mit clientseitigen Zeigern gearbeitet wird. Die Vorteile von `adForwardOnly` sind zwar erkennbar, halten sich aber in Grenzen.

Die Ergebnisse zeigen eine starke Abhängigkeit von der Größe der Datenbank. Allgemeingültige Aussagen lassen sich mit diesen wenigen Tests nur schwer treffen. Sie können aber erkennen, dass sich die üblichen Empfehlungen – clientseitige Zeiger und Batch-Mode – gut messtechnisch bestätigen lassen.

Treiber und Provider

Auch der verwendete Treiber spielt eine Rolle. Da OLEDB oder ODBC aber nur beim Transport der Daten von und zum Server in Aktion treten, wirken sich diese Unterschiede nur bei `Open` oder `Execute` aus, nicht jedoch bei den häufiger verwendeten `Move`-Methoden. Generell bestätigen Tests nur die schon im Kapitel 1 gemachten Aussagen, dass native Treiber schneller sind.

Gespeicherte Prozeduren

Wenn Kommandos mehrfach ausgeführt werden, lohnen sich gespeicherte Prozeduren. Auch die mit `Prepared` automatisch erzeugten temporären Prozeduren bringen Leistungsvorteile, wenn der Aufruf innerhalb eines Skripts mehrfach erfolgt. Der Zuwachs liegt aber nur bei Faktor 2 bis 3.

5.4 XML

ADO und XML werden in der letzten Zeit immer wieder zusammen erwähnt. Was konkret bereits nutzbar ist – XML (*Extensible Markup Language*) ist in einer sehr dynamischen Entwicklungsphase – und was ADO damit zu tun hat, wird in diesem Abschnitt behandelt.

5.4.1 Einführung in XML

Dies ist ein kompaktes Buch zu ADO. Entsprechend knapp fällt auch die Einführung in XML aus – informieren Sie sich bei Bedarf in entsprechend spezialisierten Büchern.

Warum XML?

XML ist ein standardisiertes Format für Daten. Dabei wird nicht genau unterschieden, um welche Art von Daten es sich handelt. XML stellt lediglich eine bestimmte Form der Repräsentation von Daten dar. Die konkrete Umsetzung obliegt einem davon abgeleiteten Dialekt, von denen inzwischen Hunderte existieren.

Entsprechend gibt es inzwischen viele Anwendungen, die mit XML umgehen. Dies bedeutet nicht zwangsläufig, dass alle Anwendungen dieser Art nun mit anderen interoperabel sind – aber zumindest in den Fällen, wo es tatsächlich gewollt ist, erscheint dies leicht realisierbar.

XML ist ein offenes Format, das als Standard vom W3C definiert ist. Es kann lizenzfrei verwendet werden, ähnlich wie HTML. XML ist eine vereinfachte Version der Meta-Sprache SGML, von der auch HTML abgeleitet wurde. XML-Daten sind von Menschen und Maschinen gleichermaßen lesbar, was die Entwicklung stark vereinfacht.

Wie XML aussieht

Im ersten Teil zu den Standard-ADO-Objekten wurde bereits ein wenig XML erzeugt. Mit der Methode `Save` des `RecordSet`-Objekts ließ sich der Inhalt des Datensatz-Objekts in einer XML-Datei speichern. Die so entstandene Datei sieht folgendermaßen aus:

```
<xml xmlns:s='uuid:BDC6E3F0-6DA3-11d1-A2A3-00AA00C14882'
     xmlns:dt='uuid:C2F41010-65B3-11d1-A29F-00AA00C14882'
     xmlns:rs='urn:schemas-microsoft-com:rowset'
     xmlns:z='#RowsetSchema'>
<s:Schema id='RowsetSchema'>
  <s:ElementType name='row'
                 content='eltOnly'
                 rs:CommandTimeout='30'>
   <s:AttributeType name='ShipperID' rs:number='1'>
      <s:datatype dt:type='int'
```

```
                dt:maxLength='4'
                rs:precision='10'
                rs:fixedlength='TRUE'
                rs:maybenull='FALSE'/>
   </s:AttributeType>
   <s:AttributeType name='CompanyName'
                  rs:number='2'
                  rs:writeunknown='TRUE'>
      <s:datatype dt:type='string'
                dt:maxLength='40'
                rs:maybenull='FALSE'/>
   </s:AttributeType>
   <s:AttributeType name='Phone'
                  rs:number='3'
                  rs:nullable='TRUE'
                  rs:writeunknown='TRUE'>
      <s:datatype dt:type='string' dt:maxLength='24'/>
   </s:AttributeType>
   <s:extends type='rs:rowbase'/>
 </s:ElementType>
</s:Schema>
<rs:data>
   <z:row ShipperID='1' CompanyName='Speedy Express'
                        Phone='(503) 555-9831'/>
   <z:row ShipperID='2' CompanyName='United Package'
                        Phone='(503) 555-3199'/>
   <z:row ShipperID='3' CompanyName='Federal Shipping'
                        Phone='(503) 555-9931'/>
</rs:data>
</xml>
```

Listing 5.11: XML-Ansicht eines Datensatzobjekts

Wesentliche Eigenschaften

Der Aufbau ist streng strukturiert – ein Wesensmerkmal von XML. Unterscheiden lassen sich folgende Teile:

- *Tag*. Ein Tag ist eine Auszeichnungsmarke der Sprache, die in spitzen Klammern steht: `<xml>`. Es gibt zwei Arten von Tags:
  - Container, die noch ein schließendes Tag besitzen:

    `<xml></xml>`
  - Tags, die »sich selbst« schließen: `<z:row ... />`
- *Attribut*. Damit sind Eigenschaften eines Tags benannt.

- *Element*. Damit wird ein vollständiges Tag mit Attributen, Inhalt und dem öffnenden und schließenden Tag bezeichnet.

Wohlgeformte Dokumente

In der XML-Welt gibt es den Begriff des wohlgeformten Dokuments. Damit ist ein Dokument gemeint, das folgende Eigenschaften aufweist:

- Jedes Tag muss geschlossen werden.
- Argumente der Attribute müssen in Anführungszeichen stehen.
- Groß- und Kleinschreibung wird bei Tag-Namen unterschieden.
- Es darf nur ein Wurzelelement geben.

Daraus lässt sich ableiten, dass die folgenden Darstellungen in XML nicht legal sind:

```
<orderdate date=26.05.2000></orderdate>
```

In diesem Tag fehlen die Anführungszeichen für das Argument `date`.

```
<orderdate date="26.05.2000"></ORDERDATE>
```

Hier wurde die Regel Groß- und Kleinschreibung verletzt – das schließende Tag entspricht nicht dem öffnenden.

```
<orderdate date="26.05.2000">
```

Hier fehlt das schließende Tag.

Korrekt sind dagegen die folgenden beiden Varianten. Zuerst die Version mit schließendem Tag:

```
<orderdate date="26.05.2000"></orderdate>
```

Auch eine »verkürzte« Schreibweise ist zulässig, wenn das Tag keinen Inhalt hat und sich selbst schließt:

```
<orderdate date="26.05.2000" />
```

Ob Tags selbst groß oder klein geschrieben werden, spielt dagegen keine Rolle, auch wenn sich Kleinschreibung durchgesetzt hat.

Wenn Sie das Dokument aus Listing 5.11 betrachten, fällt noch eine weitere Eigenschaft auf: Es gibt ein umschließendes Tag `<xml>`. Dies muss nicht `<xml>` sein – es muss aber genau ein sogenanntes Wurzel-Tag geben. Folgende Dokumentstruktur ist unzulässig:

```
<xml>
  <data>...</data>
</xml>
<xml:second>
```

```
  <data>...</data>
</xml:second>
```

Korrekt wäre dagegen diese Variante:

```
<xml>
  <xml:first>
    <data>...</data>
  </xml:first>
  <xml:second>
    <data>...</data>
  </xml:second>
</xml>
```

Verarbeitung von XML-Daten

Auch für die Verarbeitung von Daten in XML gibt es einige simple Vorschriften. Es existieren Programme zur Prüfung der Eigenschaften von XML-Dokumenten, so genannte XML-Validatoren, die auf Fehler hinweisen.

Version

Die erste Anweisung beinhaltet die Version, in der das Dokument verfasst ist. Derzeit gibt es nur XML der Version 1.0. Entsprechend wird das umschließende XML-Tag folgendermaßen erweitert:

```
<xml version="1.0"?>
```

Beachten Sie das Fragezeichen am Ende des Tags, es weist darauf hin, dass es sich hier um eine Prozessanweisung handelt.

Sprachen und Zeichensatz

Die zweite Anweisung beinhaltet die Sprache und den darauf abgestimmten Zeichensatz. Für den Gebrauch mit mehreren westlichen Sprachen eignet sich UTF-8 (8 Bit Unicode). Bei rein lateinischen Buchstaben, wie sie im Englischen gebraucht werden, reicht auch der Standardzeichensatz ISO-8859-1. Die folgende Tabelle zeigt die möglichen Varianten:

Tabelle 5.3: Zeichensätze

Zeichensatz	Sprache
UTF-8	Unicode, 8 Bit
ISO-8859-1	Westliches Europa
ISO-8859-2	Osteuropa
ISO-8859-3	Südost-Europa
ISO-8859-4	Skandinavische Sprachen
ISO-8859-5	Latein und Kyrillisch
ISO-8859-6	Latein und Arabisch
ISO-8859-7	Latein und Griechisch

Zeichensatz	Sprache
ISO-8859-8	Latein und Hebräisch
ISO-8859-9	Latein und Türkisch
EUC-JP, Shift_JIS	Japanisch

Das <xml>-Tag wird nun um eine entsprechende Anweisung erweitert:

```
<xml version="1.0" encoding="iso-8859-1"?>
```

Sonderzeichen

Ähnlich wie in HTML gibt es in XML Sonderzeichen, die eine besondere Bedeutung haben. Die folgende Tabelle zeigt diese Zeichen und die Notation im Datenbereich:

Tabelle 5.4: Sonderzeichen in XML

Zeichen	Entität	Hexadezimal	Dezimal
&	&amp;	&#x26;	&#38;
<	&lt;	&#x3C;	&#60;
>	&gt;	&#x3E;	&#62;
"	&quot;	&#x22;	&#34;
'	&apos;	&#x27;	&#39;

Es ist also *nicht* notwendig, Umlaute zu codieren, wenn diese vom gewählten Zeichensatz unterstützt werden. Besonders mit UTF-8 können so Datenimporte und -exporte leichter realisiert werden. Der folgende Text ist nicht zulässig:

```
<data>Müller & Brüder</data>
```

Zulässig ist dagegen:

```
<data>Müller &amp; Brüder</data>
```

Wenn Sie Daten mit entsprechenden Funktionen in ADO ins XML-Format transformieren, müssen Sie das nicht weiter beachten – ADO erledigt das zuverlässig. Bei der Erstellung von XML-Dateien mit eigenen Skripten oder von Hand sind Sie natürlich selbst dafür verantwortlich.

Unabhängig davon ist es eine gute Idee, die in HTML üblichen Entitäten, z.B. © für ©, auch in XML zu verwenden, vor allem wenn die Ausgabe später in HTML erfolgt.

Definition der Dokumentstruktur

Wie bisher zu sehen war, ist die Ausgestaltung der Tags offensichtlich nicht Bestandteil des Standards XML, sondern der Applikation. Sie müssen also dem Empfänger einer XML-Datei eine Information darüber zukommen las-

sen, wie die Tags zu interpretieren sind. Es versteht sich von selbst, dass auch diese Form maschinenlesbar sein muss.

DTD

Es gibt zwei Ansätze, dies zu realisieren: Document Type Definitions (DTDs) und Schemas. DTDs gehören zum aktuellen XML-Standard und nutzen eine eigene Beschreibungssprache, wie das folgende Beispiel zeigt:

```
<!DOCTYPE sales [
   <!ELEMENT sales (sale)*>
   <!ELEMENT sale (id|order|date)*>
   <!ELEMENT id (#PCDATA)
   <!ELEMENT order (#PCDATA)
   <!ELEMENT date (#PCDATA)
]>
```

Listing 5.12: Einfache DTD

Der Aufbau ist relativ einfach, wenn die Struktur bekannt ist. Das erste Element `DOCTYPE` definiert das Wurzelelement:

```
<!DOCTYPE wurzel_element [
```

Dann werden Elemente definiert. Im Beispiel wird ein Element *sale* erzeugt, das selbst weitere Elemente enthalten darf. `PCDATA` definiert den Inhalt der Elemente als lesbare Zeichen, Parsed Character Data. Attribute werden in diesem einfachen Beispiel nicht definiert.

Es gibt im Web einen XML-Validator, der ein Dokument auf der Grundlage einer DTD überprüft. Sie können sich dieses Werkzeug von der folgenden Adresse laden:

- *http://www.microsoft.com/downloads/internet/samples/xml/ xml_validator*

Schemas

Die bislang übliche Methode der Definition – DTDs – weist einige schwerwiegende Nachteile auf:

- Es gibt keine Möglichkeit, Datentypen zu definieren – alle Daten sind immer »PCDATA«.
- Die Definition selbst erfolgt nicht in XML. Das ist zwar mehr ein philosophisches Thema, nichtsdestotrotz für die Zukunft sehr kritisch zu betrachten.
- Das Format ist nicht erweiterbar, Änderungen bedürfen einer neuen DTD.
- Es gibt nur eine DTD pro Dokument, Kombinationen und Vererbungen sind nicht vorgesehen.

Microsoft hat deshalb mit einer Handvoll kleinerer Firmen eine andere Form der Strukturdefinition entwickelt. Man mag diesen Weg kritisch betrachten, in Anbetracht des frühen Entwicklungsstadiums ist das Einbrin-

gen alternativer Vorschläge zur Vermeidung historischer Fehler aber durchaus legitim. Auch wenn einige Entwickler aus Gründen persönlicher Abneigung gegen Microsoft streng auf die aktuellen W3C-Standards setzen, soll hier der Alternative doch ein breiterer Raum eingeräumt werden – gerade weil diese Lösung intelligenter und konsequenter ist.

Schemas definieren die Struktur der XML-Dokumente. Sie sind selbst in XML geschrieben. Das folgende Listing zeigt ein Beispiel:

```
<?XML version="1.0" encoding="UTF-8" ?>
  <schema id="rss">
    <ElementType name="id" dt="i4" />
    <ElementType name="order" dt="string" />
    <ElementType name="date" dt="date" />
    <ElementType name="sale" model="closed" content="eltOnly">
       <element type="id" />
       <element type="order" />
       <element type="sale" />
   </ElementType>
   <ElementType name="sale" model="closed" content="eltOnly">
       <element type="sale" maxOccurs="*" />
   </ElementType>
</schema>
```

Listing 5.13: Schema als Alternative zur DTD in Listing 5.12 (Ausschnitt)

Neu ist hier die Definition von Datentypen im Namensraum `dt:` (auf Namensräume wird im Anschluss an diesen Abschnitt eingegangen). Sie können also exakt bestimmen, was der Inhalt eines Felds sein darf – für die Speicherung von Daten aus Datenbanken ist das unbedingt notwendig.

Schemas im Detail

Das Element `Elementtype` definiert den umgebenden Container. Die Attribute haben folgende Bedeutung:

- `name`: Name des Tags
- `model`: *`closed`* steht für Tags, die selbst nur definierte Tags enthalten dürfen.
- `content`: *`eltOnly`* steht für »elements only«, dieses Tag darf nur andere im Schema definierte Elemente enthalten, keine Daten.

Normalerweise kann jedes Tag nur ein Mal im umgebenden Container erscheinen. Mit `maxOccurs="*"` wird dies modifiziert: Das Sternchen steht für »ein oder beliebig viele«.

Namensräume

Betrachten Sie noch einmal einen Ausschnitt aus der Datei SAVE.TXT, die direkt aus einem ADO-Objekt erzeugt wurde:

```
<s:Schema id='RowsetSchema'>
<s:ElementType name='row'
               content='eltOnly'
               rs:CommandTimeout='30'>
<s:AttributeType name='ShipperID' rs:number='1'>
  <s:datatype dt:type='int'
              dt:maxLength='4'
              rs:precision='10'
              rs:fixedlength='TRUE'
              rs:maybenull='FALSE'/>
</s:AttributeType>
```

Listing 5.14: Ausschnitt aus einer realen XML-Datei

Dies sieht offensichtlich etwas anders aus: komplexer als das mit dem gezeigten Schema möglich wäre. Das korrekte Schema erscheint dann folgendermaßen:

```
<xml xmlns:s="uuid:BDC6E3F0-6DA3-11d1-A2A3-00AA00C14882"
     xmlns:dt="uuid:C2F41010-65B3-11d1-A29F-00AA00C14882"
     xmlns:rs="urn:schemas-microsoft-com:rowset"
     xmlns:z="#RowsetSchema">
<s:Schema id="RowsetSchema">
  <s:ElementType name="row"
                 content="eltOnly"
                 s:updatable="TRUE">
    <s:AttributeType name="ShipperID"
                     rs:number="1"
                     rs:basetable="shippers"
                     rs:basecolumn="ShipperID"
                     rs:keycolumn="TRUE">
                     <s:datatype
                        dt:type="int"
                        dt:maxLength="4"
                        rs:precision="10"
                        rs:fixedlength="TRUE"
                        rs:maybenull="FALSE"/>
    </s:AttributeType>
    <s:AttributeType name="CompanyName"
                     rs:number="2"
                     rs:nullable="TRUE"
                     rs:write="TRUE"
```

```
                    rs:basetable="shippers"
                    rs:basecolumn="CompanyName">
                    <s:datatype
                       dt:type="string"
                       dt:maxLength="40" />
   </s:AttributeType>
   <s:AttributeType name="Phone"
                    rs:number="3"
                    rs:nullable="TRUE"
                    rs:write="TRUE"
                    rs:basetable="shippers"
                    rs:basecolumn="Phone">
                    <s:datatype
                       dt:type="string"
                       dt:maxLength="24"/>
  </s:AttributeType>
  <s:extends type="rs:rowbase"/>
 </s:ElementType>
</s:Schema>
```

Listing 5.15: Komplexes Schema für Datensätze

Namensräume

In diesem Schema werden vier Namensräume verwendet. Namensräume trennen Bezeichnungen von Attributen, sodass keine Namenskonflikte entstehen. Die Notation ist einfach:

```
namensraum:tag
```

Vor dem Doppelpunkt steht die Bezeichnung des Namensraums, danach das Element aus diesem Bereich. Definiert werden Namensräume durch Verweise auf interne oder externe Bibliotheken, auch über UUIDs:

```
xmlns:s="uuid:BDC6E3F0-6DA3-11d1-A2A3-00AA00C14882"
xmlns:dt="uuid:C2F41010-65B3-11d1-A29F-00AA00C14882"
xmlns:rs="urn:schemas-microsoft-com:rowset"
xmlns:z="#RowsetSchema">
```

Bedeutung der Namensräume

Die vier Namensräume haben folgende Bedeutung:

- `s`: Schema
- `dt`: Datentypen
- `rs`: RecordSet (Datensatz)
- `z`: Erweiterung der Reihendefinition

Datentypen

Die Datentypen, die im Namensraum `dt` verwendet werden können, finden Sie in der folgenden Tabelle:

Tabelle 5.5: Datentypen in XML-Schemas

Datentyp	Beschreibung
`bin.base64`	Binär
`bin.hex`	Hexadezimale Oktetts
`boolean`	Wahrheitswert
`char`	1 Zeichen
`date`	Datum im Format yyyy-mm-tt
`dateTime`	Datum mit Zeit: yyyy-mm-ttThh:mm:ss
`dataTime.tz`	yyyy-mm-ttThh:mm:ss-hh:mm, wobei der letzte Teil die Differenz zur lokalen Zeitzone bestimmt
`fixed.14.4`	Gleitkommazahl mit 14 Stellen vor und 4 Stellen nach dem Komma
`float`	Gleitkommazahl
`int`	Integer
`number`	Alternative Bezeichnung für `float`
`time`	Zeit
`time.tz`	Zeit mit Zeitzonenangabe
`i1`	Integer, 1 Byte
`i2`	Integer, 2 Bytes
`i4`	Integer, 4 Bytes
`r4`	Gleitkommazahl, 4 Bytes
`r8`	Gleitkommazahl, 8 Bytes
`ui1`	Integer, 1 Byte, ohne Vorzeichen
`ui2`	Integer, 2 Bytes, ohne Vorzeichen
`ui4`	Integer, 4 Bytes, ohne Vorzeichen
`uri`	URI
`uuid`	UUID

Zusätzlich zu diesen von Microsoft vorgeschlagenen Typen hat W3C die Liste um einige primitive Typen ergänzt:

Datentyp	Entspricht Element in einer DTD
`entity`	ENTITY
`entitities`	ENTITIES
`enumeration`	Aufzählungstyp
`id`	ID
`idref`	IDREF
`idrefs`	IDREFS
`nmtoken`	NMTOKEN
`nmtokens`	NMTOKENS
`notation`	NOTATION
`string`	Zeichenkettentyp (praktisch `PCDATA`)

Tabelle 5.6: Primitive dt:-Elemente

Attribute von ElementType

ElementType

`ElementType` darf verschiedene Attribute enthalten, die Sie der folgenden Tabelle entnehmen können:

Attribut	Beschreibung
`content`	Art des Inhalts. Darf Folgendes sein: • `empty`. Element darf kein Inhalt haben. • `textOnly`. Nur Text ist erlaubt. • `eltOnly`. Nur andere Elemente sind erlaubt. • `mixed`. Andere Elemente und Text dürfen enthalten sein.
`dt:type`	Datentyp, wie zuvor beschrieben
`model`	Art des Modells: • `closed`. Nur definierte Elemente sind erlaubt. • `open`. Auch andere Elemente mit eigenem Namensraum sind möglich.
`name`	Name
`order`	Reihenfolge der Elemente: • `one`. Nur ein Element ist erlaubt. • `seq`. Die Elemente müssen in der Reihenfolge erscheinen, wie sie definiert wurden. • `many`. Die Reihenfolge spielt keine Rolle, es müssen auch nicht alle Elemente genutzt werden.

Tabelle 5.7: Attribute von ElementType

Attribute Die Attribute können mit den folgenden Elementen definiert werden:

Attribut	Beschreibung
`default`	Standardwert
`dt:type`	Datentyp
`dt:values`	Wenn der Datentyp eine Aufzählung ist, stehen hier die Elemente.
`name`	Name
`required`	`yes` oder `no`, je nachdem, ob das Element erforderlich ist

Im Beispiel in Listing 5.15 fanden Sie weitere Attribute, z.B. `rs:keycolumn="TRUE"`. *keycolumn* ist kein Standardattribut, sondern eine Erweiterung des Namenraums `rs` (dies steht für *RecordSet*). Dies macht die Kraft von XML aus: Es ist unbegrenzt erweiterbar.

5.4.2 Praktische Anwendung

Ein praktischer Weg der Verwendung von XML-Daten ist die Persistenz. Darunter versteht man die Speicherung von Datensätzen bzw. Datenabfragen außerhalb der Datenbank. Nach der Abfrage werden die Daten im lokalen Datensatzobjekt gehalten – normalerweise also in einem bestimmten Speicherbereich des Hauptspeichers des Webservers. Das Speichern, Versenden und Reaktivieren ist schwierig. Sie werden bisher Ihre Applikationen auch so programmiert haben, dass diese das Ergebnis der Abfrage am Ende des Skripts verwerfen und dann auf der nächsten Seite erneut die Datenbank abfragen. Persistente Datensätze in der klassischen ADO-Programmierung konnten immerhin auf eine bereits bestehende Verbindung zum Datenbankserver zugreifen. Die Abfrage selbst musste dennoch erneut initiiert werden.

Nutzwert

Schon bei kleinen Projekten können Sie von XML profitieren. Das Durchblättern von längeren Tabellen ist ein typisches Problem. Herkömmlich wird die Tabelle immer wieder abgefragt und dann mit Hilfe der Zeigersteuerung in Seiten und Abschnitte zerlegt. Mit XML benötigen Sie nur eine Abfrage, die Daten selbst werden als Datei persistent gemacht und stehen so schneller zur Verfügung.

```
<% option explicit %>
<html>
<body>
<h1>XML</h1>
<h2>Persistenz</h2>
<%
dim objFO, strFile, field, position, records
```

```
position = 1
strFile = Server.MapPath("customer.table.xml")
if open() then
   set objRS = Server.CreateObject("ADODB.RecordSet")
   set objFO = Server.CreateObject
               ("Scripting.FileSystemObject")
   if objFO.FileExists(strFile) then
      objRS.Open strFile
   else
      objRS.Open "SELECT * FROM Customers", objConn,
                 adOpenKeySet, adLockOptimistic
      objRS.Save strFile, adPersistXML
   end if
   records = objRS.RecordCount
   if len(Request.Form("position-start")) > 0 then
      position = 1
   elseif len(Request.Form("position-end")) > 0 then
      position = records
   elseif len(Request.Form("position-back")) > 0 then
      position = Request.Form("position") - 1
   elseif len(Request.Form("position-next")) > 0 then
      position = Request.Form("position") + 1
   elseif len(Request.Form("position-next5")) > 0 then
       position = Request.Form("position") + 5
   elseif len(Request.Form("position-back5")) > 0 then
       position = Request.Form("position") - 5
   end if
   objRS.AbsolutePosition = position
   echo "Tabelle enth&auml;lt <b>$records</b>
         Datens&auml;tze<br>"
   echo "Aktueller Datensatz Nr. <b>$position</b><br>"
   echo "<table>"
   for each field in objRS.Fields
      echo "<tr><td bgcolor=#eeeeee>" & field.name
     echo "</td><td>" & field.Value & "</td></tr>"
   next
   echo "</table>"
end if
%>
<form method="post" action="<% = ASP_SELF %>">
<input type="Hidden" name="position" value="<% = position %>">
<input type="Submit" name="position-start" value="|<"
  <% if position = 1 then echo "disabled" %>
 style="width:30px">
<input type="Submit" name="position-back5" value="-5"
```

```
  <% if position < 6 then echo "disabled" %>
 style="width:30px">
<input type="Submit" name="position-back" value="<"
  <% if position = 1 then echo "disabled" %>
 style="width:30px">
<input type="Button" value="<% = position %>" disabled
style="width:50px">
<input type="Submit" name="position-next" value=">"
   <% if position = records then echo "disabled" %>
 style="width:30px">
<input type="Submit" name="position-next5" value="+5"
   <% if position > records - 6 then echo "disabled" %>
 style="width:30px">
<input type="Submit" name="position-end" value=">|"
   <% if position = records then echo "disabled" %>
 style="width:30px">
</form>
</body>
</html>
```

Listing 5.16: Speicherung einer Datenbankabfrage in einer XML-Datei

Das Skript besteht im Wesentlichen aus zwei Teilen: der Erzeugung und Aktivierung der Datenbankabfrage und dem Formular mit den Schaltflächen zum Durchlaufen der Datensätze sowie deren Auswertung.

Wie es funktioniert

Die Abspeicherung des Datensatzobjekts erfolgt in einer Datei namens CUSTOMER.TABLE.XML.

```
strFile = Server.MapPath("customer.table.xml")
```

Dann wird die Verbindung geöffnet und zuerst untersucht, ob die Datei bereits existiert:

```
if open() then
   set objRS = Server.CreateObject("ADODB.RecordSet")
   set objFO = Server.CreateObject
               ("Scripting.FileSystemObject")
```

Ist das der Fall, wird die Datei in das Datensatzobjekt eingelesen:

```
   if objFO.FileExists(strFile) then
      objRS.Open strFile
```

Andernfalls wird die Datenbankabfrage initiiert:

```
   else
      objRS.Open "SELECT * FROM Customers", objConn,
                 adOpenKeySet, adLockOptimistic
```

Das neu befüllte Datensatzobjekt wird als XML-Datei gespeichert und steht beim nächsten Aufruf allen Nutzern zur Verfügung:

```
      objRS.Save strFile, adPersistXML
   end if
```

Dann wird die Anzahl der Datensätze und damit die höchste Nummer festgestellt:

```
   records = objRS.RecordCount
```

Jetzt werden die Eingaben des Formulars ermittelt und die Nummer des aktuellen Datensatzes wird ausgewählt:

```
   if len(Request.Form("position-start")) > 0 then
      position = 1
   elseif len(Request.Form("position-end")) > 0 then
      position = records
   elseif len(Request.Form("position-back")) > 0 then
      position = Request.Form("position") - 1
   elseif len(Request.Form("position-next")) > 0 then
      position = Request.Form("position") + 1
   elseif len(Request.Form("position-next5")) > 0 then
       position = Request.Form("position") + 5
   elseif len(Request.Form("position-back5")) > 0 then
       position = Request.Form("position") - 5
   end if
```

Anschließend wird der aktuelle Datensatz ausgewählt:

```
   objRS.AbsolutePosition = position
```

Jetzt erfolgt die Darstellung des Datensatzes in einer Tabelle und die Generierung der Schaltflächen. Das Formular enthält noch einige Abfragen, mit denen Schaltflächen am Anfang und am Ende der Tabelle gesperrt werden, andernfalls würde beim Aufruf der Methode `AbsolutePosition` ein Laufzeitfehler entstehen.

Diskussion

Es ist sicher in der Praxis notwendig, weitere Fehlerroutinen einzubinden. Außerdem wäre es sinnvoll, einen Mechanismus zuer Sicherung der Aktualität des persistenten Datensatzobjekts einzuführen. Beispielsweise könnten Sie die Datei zum Beginn jeder Stunde löschen, der erneute Aufbau erfolgt automatisch. Insgesamt ist bei dieser Lösung eine deutliche Steigerung der Zugriffsgeschwindigkeit feststellbar, denn unzählige Datenbankzugriffe entfallen. Die Datei hält ASP im internen Cache, sodass auch hier kaum Verzögerungen eintreten. Bei komplexen Datenbankabfragen mit Berechnungen und Verknüpfungen treten die Vorteile noch drastischer hervor.

Abbildung 5.13: Durchlaufen eines persistenten Datensatzes über eine XML-Datei

Umgang mit XML-Dateien: XSL und XSLT

Der Internet Explorer kann ab Version 5 XML-Dateien darstellen. – und das tatsächlich in durchaus praktikabler Form, nämlich unter Verwendung von XSL-Vorlagen. XSL steht für *Extensible Style Sheet Language* und stellt so etwas wie CSS für XSL dar. Tatsächlich ist der Umgang mit Daten ungleich komplizierter als der mit gestalterischen Elementen. Denn viele Aktionen, die bislang mit ADO-Objekten ausgeführt werden, könnten nun auch mit XML erfolgen. Also muss XSL auch über Mechanismen zum Filtern, Sortieren und Blättern verfügen. Diese Sprache ist noch in der Entwicklung und weit entfernt von allgemeiner Verfügbarkeit. Trotzdem ist es ein dankbares Projekt, mit realen Daten und XSL umzugehen.

XML-Daten zum Browser senden

Der erste Schritt besteht darin, die XML-Daten zum Browser zu senden. Das folgende Skript zeigt, wie dies funktioniert:

```
<!-- #include file="open.inc.asp" -->
<?xml version='1.0' encoding='UTF-8'?>
<%
if open() then
   set objRS = Server.CreateObject("ADODB.RecordSet")
   objRS.Open "SELECT * FROM Customers", objConn
```

```
   objRS.Save Response, adPersistXML
end if
%>
```

Listing 5.17: XML.Stream.asp: Ausgabe des XML-Streams direkt zum Browser

Die Kopfzeile ist schon am Anfang des Abschnitts vorgestellt worden. Beachten Sie, dass hier `encoding='UTF-8'` eingesetzt wurde, damit kommen die vielen fremdsprachlichen Elemente der *NorthWind*-Datenbank direkt zur Anzeige.

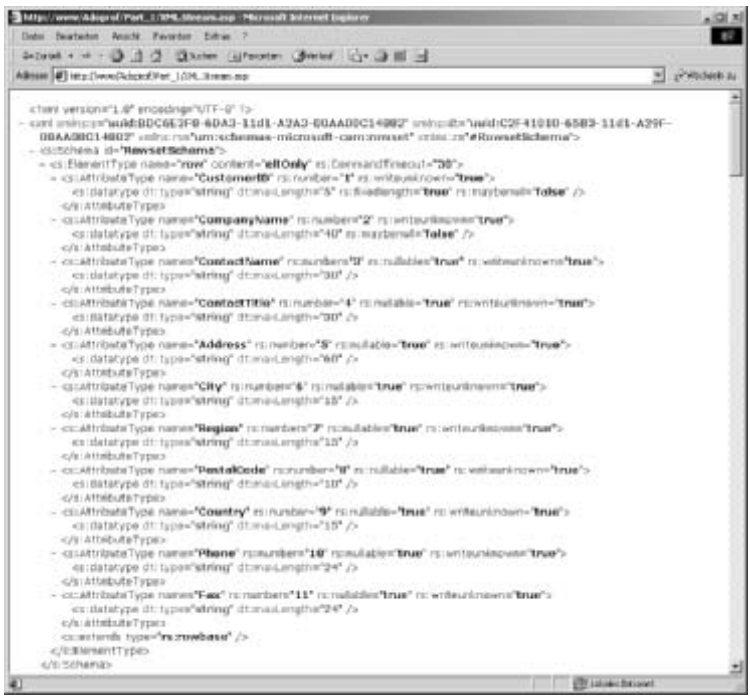


Abbildung 5.14: Schema-Definition der XML-Datei im IE5

Neben den Daten enthält die Datei auch das Schema. Sie können die korrespondierenden Tags mit den Minus-Zeichen zuklappen und mit den Plus-Zeichen aufklappen.

Falls Sie statt der Umlaute kryptische Zeichen sehen, wurde der Parameter für `encoding` falsch gesetzt.

Bis dahin ist das sicher nicht mehr als ein netter Effekt. Nutzer können Sie mit solchen Daten nicht konfrontieren. Dabei hilft aber ein weiterer Mechanismus: XSL bzw. die davon abgeleitete Transformationssprache XSLT. Außerdem kann eine grundlegende Formatierung mit CSS *(Cascading Style Sheets)* erfolgen.

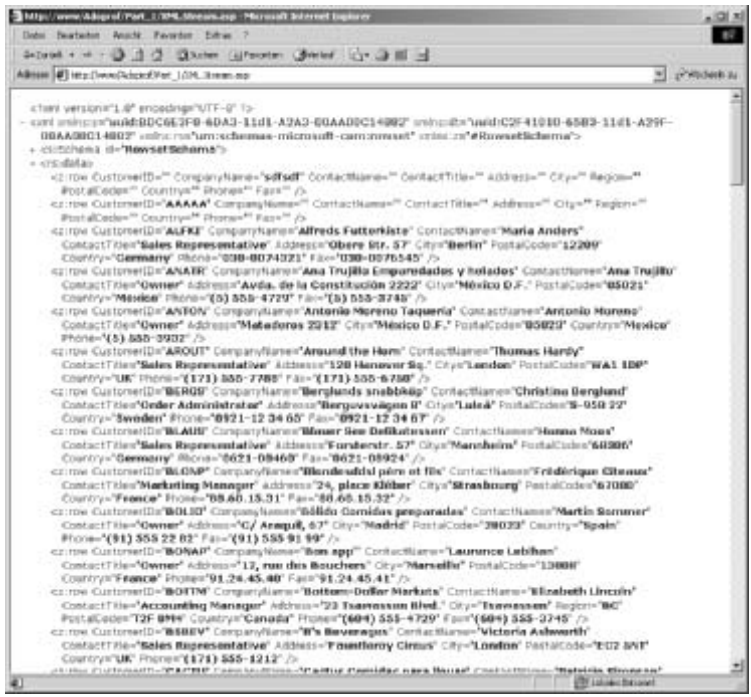


Abbildung 5.15: Daten aus der Tabelle Customers in XML

XSLT und CSS

XSLT im IE 5

Der im IE 5 implementierte XSLT-Mechanismus lässt sich heute schon nutzen, auch wenn der praktische Einsatz kritisch betrachtet werden muss. Die Sprache ist noch nicht endgültig standardisiert, kann sich also noch ändern – beim derzeitigen Stand sind auch grundlegende Änderungen denkbar. Sie sollten also nicht mehr in Zeit investieren, als es die hier gezeigten Experimente erfordern. Das Beispiel zeigt aber deutlich, wohin die Reise geht und was in den nächsten Jahren zu erwarten ist.

CSS

Mittels CSS lässt sich das Dokument in Form bringen. Die einfachste Form benennt eine CSS-Datei und die Tags werden, analog der Anwendung bei HTML, entsprechend der XML-Datei benannt. Namensräume werden unverändert mitgeführt.

XSL

Mit XSL stehen Ihnen noch mehr Möglichkeiten offen. Stylesheets werden hier ebenso integriert. Das folgende Listing zeigt die Integration der XSL-Datei:

```
<!-- #include file="open.inc.asp" -->
<?xml version='1.0' encoding='UTF-8'?>
<?xml-stylesheet href='test.xsl' type='text/xsl'?>
<%
if open() then
   set objRS = Server.CreateObject("ADODB.RecordSet")
```

```
   objRS.Open "SELECT * FROM Customers", objConn
   objRS.Save Response, adPersistXML
end if
%>
```

Listing 5.18: XML.Stream.xsl.asp: Ausgabe direkt an den Browser per XML und mit Formatierung per XSL

Die entscheidende Zeile ist einfach aufgebaut:

```
<?xml-stylesheet href='test.xsl' type='text/xsl'?>
```

Das Attribut *href* zeigt auf die URL der Style-Datei, mit *type* wird der MIME-Typ angegeben. Die eigentliche Leistung besteht im Erstellen der XSL-Datei. Hier soll nur ein Beispiel gezeigt werden – die komplette Darstellung würde ein eigenes Buchprojekt ergeben. XSL ist sehr umfangreich und keineswegs trivial.

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template>
<style>
td { font-family:Arial; font-size:10pt; color:blue; }
</style>
<table border="1">
   <tr>
      <th>CustomerID</th>
      <th>CompanyName</th>
      <th>ContactName</th>
      <th>City</th>
      <th>Country</th>
      <th>Phone</th>
   </tr>
 <xsl:for-each
  select="xml/rs:data/z:row[@CustomerID != ''
          and @CompanyName != '']"
   order-by="+ @City">
  <tr>
     <td><xsl:value-of select="./@CustomerID"/></td>
     <td><xsl:value-of select="./@CompanyName"/></td>
     <td><xsl:value-of select="./@ContactName"/></td>
     <td><xsl:value-of select="./@City"/></td>
     <td><xsl:value-of select="./@Country"/></td>
     <td><xsl:value-of select="./@Phone"/></td>
  </tr>
 </xsl:for-each>
 <hr/>
</table>
```

```
</xsl:template>
</xsl:stylesheet>
```

Listing 5.19: test.xsl: die XSL-Steuerdatei zur Anzeige der Daten

Bevor Sie sich näher mit der Konstruktion der XSL-Datei befassen, soll das Ergebnis im Browser gezeigt werden (siehe Abbildung 5.17):

Offensichtlich wird die zeilenweise Ausgabe nicht mehr auf der ASP-Seite ausgeführt, sondern direkt im Browser – unter Aufsicht der XSL-Datei.

Wie es funktioniert

Zuerst beginnt jede XSL-Datei mit einem speziellen Kopf. Üblich ist die Definition eines eigenen Namensraums »xsl«, damit es keine Namenskonflikte gibt:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
```

Dann wird die gesamte Datei als Vorlage (Template) betrachtet: Der Browser weiß jetzt also, dass er die Platzhalter mit Werten aus der XML-Datei füllen muss:

```
<xsl:template>
```

Abbildung 5.16: Ausgabe des Skripts aus Listing 5.18

CustomerID	CompanyName	ContactName	City	Country	Phone
DRACD	Drachenblut Delikatessen	Sven Ottlieb	Aachen	Germany	0241-039123
RATTC	Rattlesnake Canyon Grocery	Paula Wilson	Albuquerque	USA	(505) 555-5939
OLDWO	Old World Delicatessen	Rene Phillips	Anchorage	USA	(907) 555-7584
VAFFE	Vaffeljernet	Palle Ibsen	Århus	Denmark	86 21 32 43
GALED	Galería del gastrónomo	Eduardo Saavedra	Barcelona	Spain	(93) 203 4560
LILAS	LILA-Supermercado	Carlos González	Barquisimeto	Venezuela	(9) 331-6954
MAGAA	Magazzini Alimentari Riuniti	Giovanni Rovelli	Bergamo	Italy	035-640230
ALFKI	Alfreds Futterkiste	Maria Anders	Berlin	Germany	030-0074321
TESTA	Testfirma	Ludwig Tester	Berlin	Deutschland	+49 / 123456789
CHOPS	Chop-suey Chinese	Yang Wang	Bern	Switzerland	0452-076545
SAVEA	Save-a-lot Markets	Jose Pavarotti	Boise	USA	(208) 555-8097
FOLKO	Folk och fä HB	Maria Larsson	Bräcke	Sweden	0695-34 67 21
KOENE	Königlich Essen	Philip Cramer	Brandenburg	Germany	0555-09876
MAISD	Maison Dewey	Catherine Dewey	Bruxelles	Belgium	(02) 201 24 67
CACTU	Cactus Comidas para llevar	Patricio Simpson	Buenos Aires	Argentina	(1) 135-5555
OCEAN	Océano Atlántico Ltda.	Yvonne Moncada	Buenos Aires	Argentina	(1) 135-5333
RANCH	Rancho grande	Sergio Gutiérrez	Buenos Aires	Argentina	(1) 123-5555
THECR	The Cracker Box	Liu Wong	Butte	USA	(406) 555-5834
GOURL	Gourmet Lanchonetes	André Fonseca	Campinas	Brazil	(11) 555-9482
GROSR	GROSELLA-Restaurante	Manuel Pereira	Caracas	Venezuela	(2) 283-2951
SUPRD	Suprêmes délices	Pascale Cartrain	Charleroi	Belgium	(071) 23 67 22 20
HUNGO	Hungry Owl All-Night Grocers	Patricia McKenna	Cork	Ireland	2967 542
ISLAT	Island Trading	Helen Bennett	Cowes	UK	(198) 555-8888
QUICK	QUICK-Stop	Horst Kloss	Cunewalde	Germany	0372-035188
HUNGC	Hungry Coyote Import Store	Yoshi Latimer	Elgin	USA	(503) 555-6874
GREAL	Great Lakes Food Market	Howard Snyder	Eugene	USA	(503) 555-7555
LEHMS	Lehmanns Marktstand	Renate Messner	Frankfurt a.M.	Germany	069-0245984

Zwischen den speziellen XSL-Elementen können Sie alles aufführen, was in HTML erlaubt ist – wohlgeformt nach der XML-Norm natürlich. Hier folgt ein normaler Stylesheet für die Formatierung der Tabellenzellen:

```
<style>
td { font-family:Arial; font-size:10pt; color:blue; }
</style>
```

Dann wird eine Tabelle definiert, dies ist pures HTML:

```
<table border="1">
   <tr>
      <th>CustomerID</th>
      <th>CompanyName</th>
      <th>ContactName</th>
      <th>City</th>
      <th>Country</th>
      <th>Phone</th>
   </tr>
```

Die Reihen der Tabelle sollen nun mit Werten gefüllt werden. Dazu wird die XSL-Transform-Anweisung `for-each` verwendet. Als Argument dient `select` zur Auswahl der XML-Tags und `order-by` zum Sortieren der Ausgabe. Hinter `select` müssen Sie immer den vollen »Pfad« der Tags angeben, inklusive der Namensräume, hier also `xml/rs:data/z:row`. Wenn sich die Daten im Container eines solchen Tags befinden, ist die Angabe fertig:

```
 <xsl:for-each
  select="xml/rs:data/z:row[@CustomerID != ''
          and @CompanyName != '']"
   order-by="+ @City">
```

Da das hier verwendete Schema aber die Daten in Attribute packt, müssen noch Attribute ausgewählt werden. Letztere stehen in eckigen Klammern und werden mit einem `@` eingeleitet. In der `for-each`-Anweisung erfolgt nun eine Selektion, ähnlich einer `if`-Anweisung in VBScript. Die Auswahl `[@CustomerID != '' and @CompanyName != '']"` bestimmt, dass nur die Datensätze angezeigt werden, deren Werte für *CustomerID* und *CompanyName* belegt sind. XSL kennt hier das gesamte Spektrum an Operatoren.

```
<xsl:for-each
  select="xml/rs:data/z:row[@CustomerID != ''
          and @CompanyName != '']"
   order-by="+ @City">
```

Innerhalb der Schleife müssen nun die Werte tatsächlich abgerufen werden; dies erfolgt mit `value-of`. Der Präfix `./` ruft das gesamte umgebende Tag auf und spart etwas Schreibarbeit. Ansonsten erfolgt hier der Zugriff auf die Attribute:

```
  <tr>
     <td><xsl:value-of select="./@CustomerID"/></td>
     <td><xsl:value-of select="./@CompanyName"/></td>
     <td><xsl:value-of select="./@ContactName"/></td>
     <td><xsl:value-of select="./@City"/></td>
     <td><xsl:value-of select="./@Country"/></td>
     <td><xsl:value-of select="./@Phone"/></td>
  </tr>
```

Wenn die Daten im Container stehen, würde man `select="."` schreiben.

Am Ende darf man nicht vergessen, alle Tags wieder zu schließen:

```
 </xsl:for-each>
</table>
</xsl:template>
</xsl:stylesheet>
```

5.4.3 XSL und ASP kombinieren

Die XSL-Datei kann natürlich auch per ASP erzeugt werden. In der Kombination ergeben sich faszinierende Möglichkeiten. Das folgende Beispiel zeigt die etwas modifizierte Ausgangsdatei. Praktisch wird nur ein Parameter aus der URL extrahiert und – darauf kommt es hier an – die Datei TEXT.XSL wurde durch ein ASP-Skript XSL.ASP ersetzt. Dieses Skript muss sich nur darum kümmern, gültigen XSL-Code zu erzeugen.

```
<!-- #include file="open.inc.asp" -->
<% order = Request.QueryString("order") %>
<?xml version='1.0' encoding='UTF-8'?>
<?xml-stylesheet href='xsl.asp?order=<% = order %>' type='text/xsl'?>
<%
if open() then
   set objRS = Server.CreateObject("ADODB.RecordSet")
   objRS.Open "SELECT * FROM Customers", objConn
   objRS.Save Response, adPersistXML
end if
%>
```

Listing 5.20: XML.Stream.xsl.2.asp: Die XSL-Datei mutiert hier zum ASP-Skript

Nun stehen alle Möglichkeiten offen, die XSL-Datei interaktiv zu erstellen. Im Beispiel wird dies benutzt, um die Tabelle nach allen Spalten sortieren zu können:

```
<%
dim order, script, fieldlist, part
order = Request.QueryString("order")
Set fieldlist = Server.CreateObject("Scripting.Dictionary")
fieldlist.Add "CustomerID", "1"
fieldlist.Add "CompanyName", "2"
fieldlist.Add "ContactName", "3"
fieldlist.Add "City", "4"
fieldlist.Add "Country", "5"
fieldlist.Add "Phone", "6"
%>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template>
<style>
td { font-family:Arial; font-size:10pt; color:blue; }
</style>
<table border="1">
 <tr>
 <% for each part in fieldlist %>
  <% if order=part then %>
     <th bgcolor="Silver">
  <% else %>
     <th>
  <% end if %>
    <a href="?order=<% = part %>"><% = part %></a>
  </th>
 <% next %>
 </tr>
 <xsl:for-each select="xml/rs:data/z:row[@CustomerID != ''
      and @CompanyName != '']" order-by="+ @<% = order %>">
 <tr>
 <% for each part in fieldlist %>
  <td><xsl:value-of select="./@<% = part %>"/></td>
 <% next %>
 </tr>
 </xsl:for-each>
 <hr/>
</table>
</xsl:template>
</xsl:stylesheet>
```

Listing 5.21: xsl.asp: auch XSL-Dateien lassen sich dynamisch erzeugen

Zwei wesentliche Effekte werden hier benutzt: Zum einen wird die Feldliste dynamisch erzeugt, innerhalb der XSL-`for-each`-Anweisung finden Sie die VBScript-`For Each`-Anweisung. Der Parameter *order* wird dynamisch eingesetzt, je nachdem, welcher Teil der Kopfleiste angeklickt wurde (siehe Abbildung 5.17).

CustomerID	CompanyName	ContactName	City	Country	Phone
DRACD	Drachenblut Delikatessen	Sven Ottlieb	Aachen	Germany	0241-039123
RATTC	Rattlesnake Canyon Grocery	Paula Wilson	Albuquerque	USA	(505) 555-5939
OLDWO	Old World Delicatessen	Rene Phillips	Anchorage	USA	(907) 555-7584
VAFFE	Vaffeljernet	Palle Ibsen	Århus	Denmark	86 21 32 43
GALED	Galería del gastrónomo	Eduardo Saavedra	Barcelona	Spain	(93) 203 4560
LILAS	LILA-Supermercado	Carlos González	Barquisimeto	Venezuela	(9) 331-6954
MAGAA	Magazzini Alimentari Riuniti	Giovanni Rovelli	Bergamo	Italy	035-640230
ALFKI	Alfreds Futterkiste	Maria Anders	Berlin	Germany	030-0074321
TESTA	Testfirma	Ludwig Tester	Berlin	Deutschland	+49 / 123456789
CHOPS	Chop-suey Chinese	Yang Wang	Bern	Switzerland	0452-076545
SAVEA	Save-a-lot Markets	Jose Pavarotti	Boise	USA	(208) 555-8097
FOLKO	Folk och fä HB	Maria Larsson	Bräcke	Sweden	0695-34 67 21
KOENE	Königlich Essen	Philip Cramer	Brandenburg	Germany	0555-09876
MAISD	Maison Dewey	Catherine Dewey	Bruxelles	Belgium	(02) 201 24 67
CACTU	Cactus Comidas para llevar	Patricio Simpson	Buenos Aires	Argentina	(1) 135-5555
OCEAN	Océano Atlántico Ltda.	Yvonne Moncada	Buenos Aires	Argentina	(1) 135-5333
RANCH	Rancho grande	Sergio Gutiérrez	Buenos Aires	Argentina	(1) 123-5555
THECR	The Cracker Box	Liu Wong	Butte	USA	(406) 555-5834
GOURL	Gourmet Lanchonetes	André Fonseca	Campinas	Brazil	(11) 555-9482
GROSR	GROSELLA-Restaurante	Manuel Pereira	Caracas	Venezuela	(2) 283-2951
SUPRD	Suprêmes délices	Pascale Cartrain	Charleroi	Belgium	(071) 23 67 22 20
HUNGO	Hungry Owl All-Night Grocers	Patricia McKenna	Cork	Ireland	2967 542
ISLAT	Island Trading	Helen Bennett	Cowes	UK	(198) 555-8888
QUICK	QUICK-Stop	Horst Kloss	Cunewalde	Germany	0372-035188
HUNGC	Hungry Coyote Import Store	Yoshi Latimer	Elgin	USA	(503) 555-6874
GREAL	Great Lakes Food Market	Howard Snyder	Eugene	USA	(503) 555-7555
LEHMS	Lehmanns Marktstand	Renate Messner	Frankfurt a.M.	Germany	069-0245984

Abbildung 5.17: XML-Daten mit dynamischer Sortierung der Spalten. Die jeweils sortierte Spalte ist im Kopf grau hinterlegt

Diskussion

Die vorgestellten Methoden zeigen einen kleinen Ausschnitt der Möglichkeiten. Sie zeigen auch, wohin die Reise geht. Die eigentliche Botschaft für den Webentwickler ist Folgende:

Aufgrund der zunehmenden Komplexität der Applikationen, die oft mit leistungsfähigen Werkzeugen von wenigen Personen umgesetzt werden, müssen Sie bestimmte Techniken anwenden, um nicht die Übersicht zu verlieren. Eine typische Technik ist die Trennung zwischen Datenbank, Geschäftslogik und Oberfläche. In klassischen Applikationen wird diese als 3-Schicht-Modell (Tree-Tier-Model) bezeichnete Form schon lange genutzt. Die Webserverentwicklung hinkt hier um einiges hinterher. XML ist der Schlüssel, mit chaotischen Spagettiprogrammen Schluss zu machen und

konsequent zu entwickeln. Sie können nun die Datenbank programmieren und sich selbst Schnittstellen zurechtlegen, wobei die »schweren« Aufgaben z.B. durch gespeicherte Prozeduren erledigt werden. Dann verwenden Sie als Übergang zur Geschäftslogik OLEDB und programmieren in ADO und ASP alle Abläufe. Die Daten werden einheitlich in XML ausgegeben.

Dann erst wird die dritte Schicht mit der Oberfläche entworfen. Nutzerspezifische Modifikationen erfolgen mit XSLT oder ähnlichen Techniken. Die Vorteile liegen auf der Hand. Verlangt ein Nutzer eine andere Sortierung oder Darstellung, bleiben die mühevoll entworfenen Skripte gänzlich unberührt.

6 ADOX

ADOX steht für *ADO Extensions for DDL and Security.* DDL wiederum ist ein Akronym für *Data Definition Language.* ADOX wurde parallel zu ADOMD mit der ADO Version 2.1 eingeführt.

6.1 Grundlagen

DDL ist Ihnen vielleicht aus SQL bekannt. SQL lässt sich in drei Sprachteile zerlegen:

- DDL: *Data Definition Language*
- DML: *Data Manipulation Language*
- DCL: *Data Control Language*

Die DDL umfasst Anweisungen zum Erzeugen und Ändern der Datenbankstrukturen, also `CREATE`, `ALTER` usw. Mit der DML haben Sie in der Praxis häufiger zu tun, hier finden sich z.B. `INSERT` oder `DELETE`. Die DCL umfasst dagegen die Transaktionkontrolle und Systemanweisungen wie `BEGIN` oder `COMMIT`.

6.1.1 ADO universell

Bislang eignete sich ADO lediglich als Abstraktionsebene für die DML. Sie konnten mit Methoden wie `AddNew` anstatt `INSERT` arbeiten und Datensätze filtern, sortieren und berechnen. Das Anlegen von Tabellen oder gar Datenbanken gelingt nicht ohne den Zugriff auf SQL-Anweisungen. Hier ist ein Einsatzgebiet von ADOX zu finden. ADOX erlaubt den Zugriff auf die Metadaten wie Prozeduren und Systemobjekte sowie die Strukturen der Datenbank. Das alles natürlich ohne spezifische Kenntnisse der Datenbank selbst. Konsequenterweise – auch wenn es der Name nicht beinhaltet – kennt ADOX auch Objekte zum Zugriff auf die DCL. Damit ist das gesamte Spektrum an Anweisungen zur Datenbankmanipulation durch ADO abgedeckt.

Entwicklungspfad

ADOX befindet sich noch in der Entwicklung. Lediglich der Jet-Treiber stellt die gesamte Funktionalität zur Verfügung. Der SQL Server 7-Treiber unterstützt nicht alle Möglichkeiten. In der Praxis kann er aber durchaus eingesetzt werden. Die folgenden Beispiele sind alle mit Access 2000 erprobt worden, um sämtliche Möglichkeiten zu zeigen. Wenn Sie den SQL-Provider einsetzen, werden Sie gegebenenfalls mit der Fehlermeldung SCHNITTSTELLE NICHT UNTERSTÜTZT konfrontiert werden. Informieren Sie sich auf den Webseiten des SQL Servers, ob ein neuer Treiber vorliegt.

6.1.2 ADOX-Objektmodell

In diesem Abschnitt werden die Objekte vorgestellt, mit denen Sie in ADOX programmieren.

Objektdiagramm

Das folgende Diagramm zeigt die Objektstruktur für ADOX.

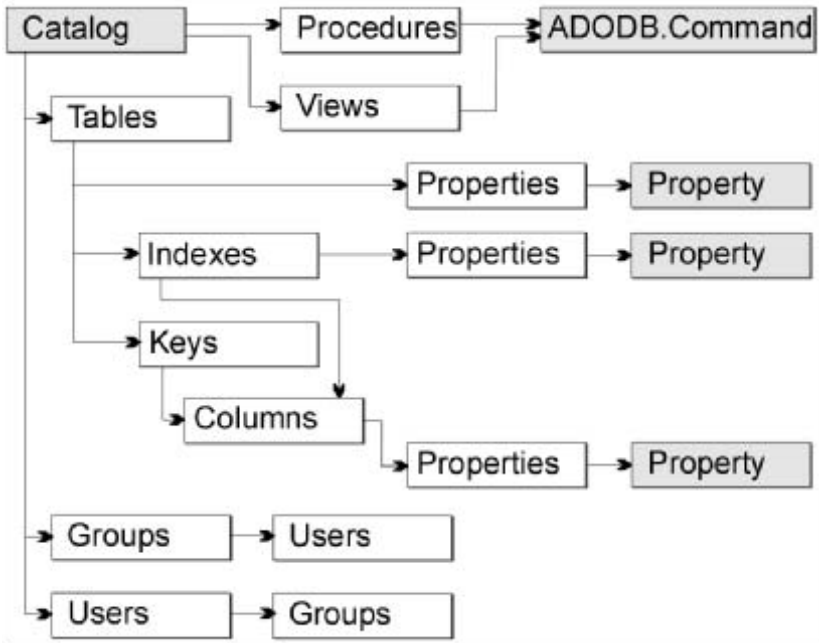


Abbildung 6.1: ADOX-Objektstruktur

Fast alle Objekte bilden Kollektionen. Das liegt in der Natur der Sache – die meisten Elemente einer Datenbank können mehrfach auftreten. Die Verbindung zwischen `Users` und `Groups` mag eigenartig aussehen, auch hier steckt aber eine klare Struktur dahinter: Gruppen enthalten Nutzer und Nutzer sind zu Gruppen zugehörig – jenachdem, wie herum man es betrachtet und administriert. Die Ableitung der Prozeduren und Sichten endet mit dem ADO-Objekt `Command`. Hier ist die Verknüpfung zu ADO offensichtlich, der gemischte Einsatz dürfte auch deshalb eher der Regelfall als die Ausnahme sein.

Übersicht ADOX-Objekte

Die Objekte `Table`, `Index` und `Column` haben eine weitere Kollektion `Properties`, die der von ADO entspricht. Siehe dazu auch Abschnitt 3.6 *Property* ab Seite 136 und Abschnitt 4.2 Properties ab Seite 164.

Objekt	Beschreibung
Catalog	Bescheibt die Datenbank
Column	Spalte einer Tabelle, eines Index oder Schlüssels
Group	Gruppenkonto mit Zugriffsrechten auf eine gesicherte Datenbank
Index	Index einer Tabelle
Key	Primär- oder Fremdschlüssel oder Unique-Feld einer Tabelle
Procedure	Gespeicherte Prozedur
Table	Tabelle, einschließlich der Spalten, Schlüssel und Indizes
User	Nutzerkonto mit Zugriffsrechten auf eine gesicherte Datenbank
View	Gefilterte Daten einer Tabelle oder Auswahl von Daten

Tabelle 6.1: ADOX-Objekte

Die Beziehungen zwischen den Objekten können Sie dem Objektmodell in Abbildung 6.1 entnehmen. Jedes Objekt kann Bestandteil der korrespondierenden Kollektion sein.

ADOX-Kollektionen

Die ADOX-Kollektionen stehen in direktem Zusammenhang mit den ADOX-Objekten – jedes Objekt, das mehr als einmal auftritt, ist Bestandteil einer Kollektion.

Collection	Description
Columns	Enthält alle Column-Objecte einer Tabelle, Index oder Schlüssel.
Groups	Enthält alle Group-Objecte eines Katalogs oder Nutzers.
Indexes	Enthält alle Index-Objekte einer Tabelle.
Keys	Enthält alle Key-Objekte einer Tabelle.
Procedures	Enthält alle Procedure-Objekte eines Katalogs.
Tables	Enthält alle Table-Objekte eines Katalogs.
Users	Enthält alle User-Objecte eines Katalogs oder einer Gruppe.
Views	Enthält alle View-Objekte eines Katalogs.

Tabelle 6.2: ADOX-Kollektionen

6.2 Funktionsweise

Dieser Abschnitt zeigt die Nutzung und einige einführende Beispiele für ADOX.

6.2.1 Zugriff auf ADOX

Um ADOX nutzen zu können, sollten Sie die Bibliothek referenzieren. Die steht zwar auch sonst zur Verfügung, da sie in der Registrierung angemeldet ist, wenn ADO 2.6 installiert wurde, mit der Referenzierung stehen aber auch die Konstanten zur Verfügung, die ADOVBS.INC nicht enthält. Es ist sinnvoll, die Referenzierung in der Datei global.asa einzutragen, um sie nicht in jedem Skript erneut aufführen zu müssen. Der folgende Tag erledigt dies:

```
<!-METADATA TYPE="TypeLib"
            NAME="Microsoft ADO Ext. 2.5 for DDL and Security"
            UUID="{00000600-0000-0010-8000-00AA006D2EA4}"
            Version="2.5" -->
```

Achten Sie auf die korrekte Übernahme der UUID-Nummer.

6.2.2 Anlegen einer neuen Datenbank

Der erste Schritt zur neuen Tabelle könnte das Anlegen einer völlig neuen Datenbank sein. Dazu benötigen Sie nicht mehr als eine Zeile ADOX-Code. Das folgende Listing zeigt es allerdings etwas komfortabler und mit »Beweisführung« – die Existenz der Datei wird mit dem `FileSystemObject` überprüft.

```
Set objCat = CreateObject("ADOX.Catalog")
strCat = Server.MapPath("NewCatalog.mdb")
echo "Tabelle wird in <b>$strCat</b> erzeugt.<br>"
strDB = "Provider=Microsoft.Jet.OLEDB.4.0; "
strDB = strDB & "Data Source= " & strCat
on error resume next
objCat.Create strDB
if Err.Number <> 0 then
   echo "<div style=""color:red""><b>Fehler:</b>"
   echo Err.Description
   echo "</div>"
end if
echo "Done..."
set objFO = Server.CreateObject("Scripting.FileSystemObject")
if objFO.FileExists(strCat) then
   echo "Datenbank wurde erfolgreich angelegt.<br>"
else
   echo "Es trat ein Fehler auf.<br>"
end if
```

Listing 6.1: ADOX.Catalog.CreateDB.asp: anlegen einer neuen Datenbank mit ADOX

Das `Catalog`-Objekt erlaubt es, sowohl neue Datenbanken anzulegen als auch bestehende zu öffnen. In diesem Beispiel wurde mit der `Create`-Methode eine neue Datenbank mit dem Namen NEWCATALOG.MDB angelegt. Notwendig sind dafür natürlich Schreibrechte. Durch die `MapPath`-Methode wird die Datenbank im Verzeichnis des aufrufenden ASP-Skripts platziert. Beim zweiten Aufruf erzeugt das Objekt einen Fehler – löschen und erneut anlegen kann das `Catalog`-Objekt nicht.

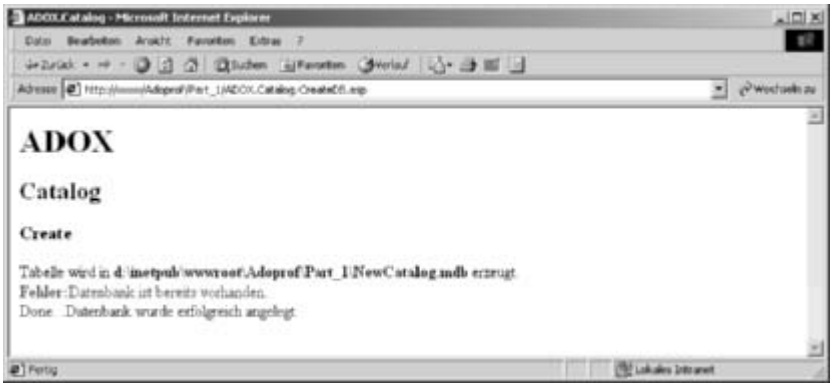


Abbildung 6.2: Beim zweiten Aufruf wird ein Fehler angezeigt – auch eine Form von Existenzbeweis

Das Abfangen der Fehler ist also immer eine gute Idee.

6.2.3 Anlegen einer Tabelle

Um eine Tabelle in der neuen Datenbank anzulegen, stellt ADOX das Objekt `Table` zur Verfügung. Das folgende Beispiel zeigt, dass Sie auch leere Tabellen (ohne Felder) in eine Datenbank einfügen können. Sie können diese Felder dann später vielfältig modifizieren. Der Zugriff auf die Datenbank erfolgt über die gewählte Verbindungszeichenfolge und die Eigenschaft `ActiveConnection` des Objekts `Catalog`:

```
Set objCat = CreateObject("ADOX.Catalog")
strCat = Server.MapPath("NewCatalog.mdb")
echo "Tabelle wird in <b>$strCat</b> erzeugt.<br>"
strDB = "Provider=Microsoft.Jet.OLEDB.4.0; "
strDB = strDB & "Data Source= " & strCat
on error resume next
objCat.ActiveConnection = strDB
Set objTbl = CreateObject("ADOX.Table")
objTbl.Name = "Customers"
objCat.Tables.Append objTbl
```

Listing 6.2: ADOX.Catalog.CreateTable.asp: Erzeugen einer leeren Tabelle

Sie müssen dazu nur ein `Table`-Objekt erstellen, einen eindeutigen Namen zuweisen und die Tabelle dann an die `Tables`-Kollektion anfügen. Dazu dient die Methode `Append`.

Hinzufügen von Feldern

Für das Hinzufügen von Feldern sollten Sie sich etwas mehr Zeit nehmen, da sehr viele Eigenschaften gesetzt werden können. Auch bei ADOX führen viele Wege zum Ziel, ähnlich wie bei ADO. Tabellendefinitionen bestehen aus mehreren Spaltendefinitionen. Dazu wird ein `Column`-Objekt erzeugt:

```
set objCol = CreateObject("ADOX.Column")
```

Im Anschluss daran erfolgt die Definition der Eigenschaften. Am übersichtlichsten ist die Benennung mit Hilfe der `with`-Anweisung:

```
with objCol
    .Name = "UserName"
    .Type = adVarWChar
    .DefinedSize = 64
end with
```

Dann wird das Spaltenobjekt an die Tabelle angefügt:

```
objTbl.Columns.Append objCol
```

Alternativ können alle Angaben auch in einer Zeile erfolgen:

```
objTbl.Columns.Append "UserName", adVarWChar, 64
```

Weitere Eigenschaften der Tabelle werden ähnlich erzeugt. Ein Autofeld lässt sich folgendermaßen erzeugen:

```
With objCol
   .Name = "AutoWertFeld"
   .Type = adInteger
End With
Set objCol.ParentCatalog = objCat
objCol.Properties("Autoincrement") = TRUE
objTbl.Columns.Append objCol
```

Anlegen eines Primärschlüssels

Für Primärschlüssel (und andere Schlüssel) wird das Objekt `Index` verwendet. Der folgende Ausschnitt zeigt die Vorgehensweise:

```
set objIdx = CreateObject("ADOX.Index")
with objIdx
   .Name = "UserID_PK"
```

```
   .IndexNulls = adIndexNullsDisallow
   .PrimaryKey = TRUE
   .Unique = TRUE
   .Columns.Append "UserID"
end with
set objTbl = objCat.Tables("Adressen")
objTbl.Indexes.Append objIdx
```

Die Eigenschaften sind eigentlich selbsterklärend. Hier ist die Verwandtschaft zu SQL unverkennbar. `Append` wird zweimal benötigt: zum einen wird die Spalte, an die der Schlüssel gebunden wird, angegeben. Zum anderen wird das `Index`-Objekt der Kollektion `Indexes` des `Table`-Objekts zugeordnet.

Sie können Indizes auch über mehrere Felder anlegen. Sie müssen nur `Columns.Append` mehrfach mit den gewünschten Feldern aufrufen.

Beispiel

Das folgende Beispiel zeigt das Zusammenspiel von ADO und ADOX – es überträgt die Struktur der Tabelle *Customers* der `Northwind`-Datenbank komplett in die neue Tabelle *Customers* der Access-Datenbank. Das Skript zeigt grob den Weg, den Sie gehen müssen, um mit ADOX-Datenbankobjekte anzulegen und zu manipulieren. Es ist keinesfalls perfekt, beispielsweise fehlen noch Index-Einträge und natürlich die Datenübertragung.

```
<%
on error resume next
dim objCat, strDB, strCat, objTbl, objFO, objCol,
    objColSchema, objIdxSchema
dim objSchema, strColName, strColType, strColSize,
    strTableName, strIndexName, intType, intColSize
dim blnPrimary_Key, blnUnique, intPrecision, intScale
set objCat = Server.CreateObject("ADOX.Catalog")
strCat = Server.MapPath("NewCatalog.mdb")
echo "Tabelle wird in <b>$strCat</b> erzeugt.<br>"
strDB = "Provider=Microsoft.Jet.OLEDB.4.0; "
strDB = strDB & "Data Source= " & strCat
echo "<br>Erzeuge Datenbank in <i>$strDB</i> ..."
'objCat.Create strDB
set objFO = Server.CreateObject("Scripting.FileSystemObject")
if objFO.FileExists(strCat) then
   echo "Datenbank wurde erfolgreich angelegt.<br>"
else
   echo "Es trat ein Fehler auf.<br>"
end if
objCat.ActiveConnection = strDB
if open() then
   set objSchema = objConn.OpenSchema(adSchemaTables,
```

```
                Array(Empty, Empty, Empty, "TABLE"))
' Liest alle Benutzer-Tabellen der DB aus
do while Not objSchema.EOF
   strTableName = objSchema("TABLE_NAME")
   set objTbl = Server.CreateObject("ADOX.Table")
   objTbl.Name = strTableName
   echo "<br>Erzeuge Tabelle <i>$strTableName</i> ..."
   objCat.Tables.Append objTbl
   set objColSchema = objConn.OpenSchema(adSchemaColumns,
                      Array(Empty, Empty, strTableName))
   while not objColSchema.EOF
      strColName = objColSchema("COLUMN_NAME")
      strColType = objColSchema("DATA_TYPE")
      strColSize = objColSchema("CHARACTER_MAXIMUM_LENGTH")
      if isNull(strColSize) then strColSize = 0
  select case strColType
        case adSmallInt, adInteger, adSingle, adCurrency:
          intPrecision = objColSchema("NUMERIC_PRECISION")
          intScale = objColSchema("NUMERIC_SCALE")
          intType = cint(strColType)
          intColSize = Cint(strColSize)
        case adBigInt:
          intType = adInteger
          intColSize = CLng(strColSize)
        case adChar, adWChar, adVarWChar:
          if strColSize <= 255 then
             intType = adVarWChar
             intColSize = CLng(strColSize)
          else
             intType = adLongVarBinary
             if strColSize >= 2^30 then
                intColSize = 2^30 - 1
             else
                intColSize = CLng(strColSize)
             end if
          end if
        case adBinary:
          intType = adLongVarBinary
          intColSize = 2^30
      end select
      if isNull(intScale) then intScale = 0
      if isNull(intPrecision) then intPrecision = 4
      ...
      set objCol = Server.CreateObject("ADOX.Column")
      with objCol
```

```
            .Name = strColName
            .Type = intType
            .DefinedSize = intColSize
            .Precision = intPrecision
            .NumericScale = intScale
         end with
         objTbl.Columns.Append objCol
         objColSchema.MoveNext
      wend
      objSchema.MoveNext
   loop
end if
%>
```

Listing 6.3: ADOX.Copytable.asp: Übertragung der Struktur einer SQL-Datenbank nach Access

Der erste Teil – das Anlegen der Datenbank – wurde bereits im ersten Beispiel dieses Kapitels behandelt. Im nächsten Schritt werden jetzt die Tabellen ausgelesen und angelegt. Zum Auslesen wird auf das Datenbankschema mit der Methode `OpenSchema` des ADO-`Connection`-Objekts zugegriffen:

```
set objSchema = objConn.OpenSchema(adSchemaTables,
                Array(Empty, Empty, Empty, "TABLE"))
```

Mit dieser Anweisung erhalten Sie in *objSchema* ein Datensatzobjekt, dass nur benutzerdefinierte Tabellen der Datenbank enthält. Diese Datensätze werden nun in einer Schleife durchlaufen; jeder Durchlauf erzeugt eine Tabelle:

```
do while Not objSchema.EOF
   strTableName = objSchema("TABLE_NAME")
   set objTbl = Server.CreateObject("ADOX.Table")
   objTbl.Name = strTableName
   ...
   objCat.Tables.Append objTbl
```

Dann werden die Spalten jeder Tabelle ausgelesen:

```
set objColSchema = objConn.OpenSchema(adSchemaColumns,
                   Array(Empty, Empty, strTableName))
```

Auch diese Liste von Spalten steht als Datensatzobjekt zur Verfügung. Die Spalteneigenschaften lassen sich einzeln abrufen:

```
while not objColSchema.EOF
   strColName = objColSchema("COLUMN_NAME")
   strColType = objColSchema("DATA_TYPE")
   strColSize = objColSchema("CHARACTER_MAXIMUM_LENGTH")
```

Jetzt folgen einige Anpassungen der Datentypen, die sich teilweise erheblich zwischen SQL Server und Access unterscheiden. Hier ist nur ein kleiner Teil angepasst worden, zugeschnitten auf die *NorthWind*-Datenbank.

Sind die Datentypen fertig aufgebaut, können die Spalten erzeugt werden:

```
set objCol = Server.CreateObject("ADOX.Column")
    with objCol
        .Name = strColName
        .Type = intType
        .DefinedSize = intColSize
        .Precision = intPrecision
        .NumericScale = intScale
     end with
objTbl.Columns.Append objCol
```

Jetzt werden die Schleifen geschlossen und das Skript ist fertig. Nach der Ausführung können Sie die fertigen Tabellen in Access sehen.

Abbildung 6.3: Kontrollausgabe des Skripts aus Listing 6.3

6.3 ADOX-Syntaxdiagramme

Dieser Abschnitt zeigt die exakte Syntax und alle Methoden und Eigenschaften des ADOX-Objektmodells.

6.3.1 Übersicht

6.3.2 Catalog

Eine Instanz des `Catalog`-Objekts legen Sie folgendermaßen an:

```
set objCatalog = Server.CreateObject("ADOX.Catalog")
```

Methoden

Create

Mit `Create` wird eine neue Datenbank erzeugt. Als Parameter wird eine ADO-Verbindungszeichenfolge erwartet.

Create

objCatalog.Create string "<vbz>"

Für <vbz> im Syntaxdiagramm setzen Sie beispielsweise Folgendes ein:

```
"Provider=Microsoft.Jet.OLEDB.4.0; Data Source=C:\dbase.mdb"
```

Der volle Funktionsumfang von ADOX wird derzeit nur von diesem Provider unterstützt.

GetObjectOwner

GetObjectOwner

Diese Methode ermittelt den Eigentümer eines Objekts.

```
Owner = objCatalog.GetObjectOwner(string ObjectName,
                                  integer ObjectType
                                [, guid ObjectTypeID])
```

Parameter	Beschreibung
ObjectName	Name des Objekts
ObjectType	Art des Objekts (siehe nächste Tabelle)
ObjectTypeID	GUID des Objekts (nur wenn ObjectType = adPermObjProviderSpecific ist)

Die folgende Tabelle zeigt alle möglichen Werte des Parameters ObjectType:

Konstante	Numerischer Wert	Beschreibung
adPermObjProviderSpecific	-1	Providerabhängig
adPermObjTable	1	Tabelle
adPermObjColumn	2	Spalte
adPermObjDataBase	3	Datenbank
adPermObjProcedure	4	Gespeicherte Prozedur
adPermObjView	5	Sicht (View)

▶ SetObjectOwner

SetObjectOwner

SetObjectOwner

Setzt die Eigenschaft Eigentümer für ein bestimmtes Objekt.

```
objCatalog.SetObjectOwner string ObjectName, integer ObjectType,
                          string OwnerName [, guid ObjectTypeID]
```

OwnerName ist der künftige Name des Eigentümers des Objekts. Alle übrigen Parameter entsprechen den bei GetObjectOwner beschriebenen.

Wenn Sie Eigentümer setzen, ist Folgendes zu beachten: Der Nutzer muss bereits in der User-Kollektion existieren. Außerdem muss die Sicherheitsdatenbank in Access 2000 explizit gesetzt worden sein. Dazu ist die Verbindungszeichenfolge für Create wie folgt zu erweitern:

```
"Provider = Microsoft.Jet.OLEDB.4.0;
 Data Source = c:\dbase.mdb;
 JET OLEDB:System database = c:\security\dbase.mdw; "
```

JET OLEDB: ist eine providerspezifische Erweiterung der Verbindungszeichenfolge. System database benennt dann die Sicherheitsdatenbank für Access.

Beispiel

```
objCat.Users.Append "Joerg", "Joergs_Kennwort"
objCat.SetObjectOwner "Customers", adPermObjectTable, "Joerg"
```

- GetObjectOwner
- Create
- User

Eigenschaften

ActiveConnection

ActiveConnection

Dieser Eigenschaft wird die Verbindungszeichenfolge übergeben. Alternativ kann auch ein ADO-Connection-Objekt angegeben werden.

```
objCatalog.ActiveConnection = object objConn
objCatalog.ActiveConnection = string "<vbz>"
```

- Create

Kollektion

Im Folgenden wird mit *objCatalogs* eine Kollektion von Catalog-Objekten bezeichnet. Catalog-Kollektionen können Objekte folgenden Typs enthalten:

- Tables
- Views
- Users
- Groups
- Procedure

Delete

Delete

Diese Methode entfernt ein Element der Kollektion.

```
objCatalogs.Delete string Name
```

6.3.3 Table

Ein Objekt vom Typ `Table` legen Sie folgendermaßen an:

```
set objTable = Server.CreateObject("ADOX.Table")
```

Der Zugriff auf Tabellen erfolgt entweder über den Namen oder den numerischen Index. Die folgenden beiden Varianten greifen dabei unmittelbar über das `Catalog`-Objekt zu:

```
objCat.Tables("TableName")
objCat.Tables(0)
```

Methoden

Append

Append Fügt eine Tabelle in die Kollektion der Tabellen ein.

```
objTable.Append string TableName
```

Statt des Namens kann auch ein anderes `Table`-Objekt angegeben werden.

Delete

Delete Diese Methode löscht die benannte Tabelle.

```
objTable. Delete string TableName
```

Eigenschaften

DateCreated

DateCreated Diese Eigenschaft gibt das Datum zurück, an dem die Tabelle erzeugt wurde.

```
date varDate = objTable.DateCreated
```

DateModified

DateModified Diese Eigenschaft gibt das Datum zurück, an dem die Tabelle zuletzt geändert wurde.

```
date varDate = objTable.DateModified
```

Name

Name Diese Eigenschaft gibt den Namen der Tabelle an.

```
string strName = objTable.Name
```

ParentCatalog

ParentCatalog

Gibt das `Catalog`-Objekt zurück oder lässt die Angabe eines solchen Objekts zu.

```
object objCat = objTable.ParentCatalog
objTable.ParentCatalog = objCat
```

Die Angabe ist notwendig, wenn providerspezifische Angaben zur Tabelle gemacht werden sollen.

Type

Type

Gibt den Typ der Tabelle an. Diese Eigenschaft kann nur gelesen werden.

```
string strType = objTable.Type
objTable.Type = strType
```

Die Zeichenkette kann folgende Werte annehmen:

- `TABLE`: normale, benutzerdefinierte Tabelle
- `SYSTEM TABLE`: die Systemtabelle
- `GLOBAL TEMPORY`: eine temporäre Tabelle

Kollektion

Sie können die üblichen Methoden für die Bearbeitung der Kollektion verwenden:

- `Item`: Zugriff auf eine Tabelle der Kollektion
- `Count`: Anzahl der Elemente
- `Delete`: Entfernen eines Eintrags
- `Refresh`: Auffrischen der Kollektion mit den Informationen aus der Datenbank

6.3.4 Index

Das Objekt `Index` wird verwendet, um Primärschlüssel und Indizes Tabellen hinzuzufügen.

Methoden

Das Objekt hat keine Methoden.

Eigenschaften

Clustered

Clustered

Diese Eigenschaft kann geschrieben und gelesen werden. Sie gibt `TRUE` zurück, wenn der Index clustered ist, sonst `FALSE`.

```
blnClust = objIndex.Clustered
objIndex.Clustered = blnClust
```

Wenn der Index bereits an die Kollektion angehängt wurde, kann die Eigenschaft nur noch gelesen werden. Der Standardwert ist `FALSE`.

IndexNulls

IndexNulls

Diese Eigenschaft gibt an, wie die Indizierung von `NULL`-Werten erfolgt.

```
objIndex.IndexNulls = integer value
```

Die folgende Tabelle gibt die zulässigen Parameter für *value an:*

value	Wert	Beschreibung
`adIndexNullsAllow`	0	Der Index erlaubt Felder mit `Null`-Werten in der Schlüsselspalte.
`adIndexNullsDisallow`	1	`Null`-Werte sind in der Schlüsselspalte nicht erlaubt (Standardwert). Werden `Null`-Werte gefunden, wird ein abfangbarer Laufzeitfehler generiert.
`adIndexNullsIgnore`	2	Der Index erlaubt Felder mit `Null`-Werten in der Schlüsselspalte, diese Reihen werden jedoch ignoriert. Fehler werden nicht erzeugt.
`adIndexNullsIgnoreAny`	4	Der Index erlaubt Felder mit `Null`-Werten in der Schlüsselspalte, diese Reihen werden jedoch ignoriert. Fehler werden nicht erzeugt. Falls der Index aus mehrere Spalten besteht, wird die Reihe auch dann nicht aufgenommen, wenn nur eines der Felder `Null` enthält.

Wenn der Index bereits an die Kollektion angehängt wurde, kann die Eigenschaft nur noch gelesen werden. Der Standardwert ist `adIndexNullsDisallow`.

- `Index`

Name

Name

Setzt oder liest den Namen des Index.

```
strIndexName = objIndex.Name
objIndex.Name = string strIndexName
```

```
strIndexName = objIndex.Name
objIndex.Name = strIndexName
```

PrimaryKey

PrimaryKey

Mit dieser Eigenschaft wird der Primärschlüssel festgelegt oder ermittelt. Sie gibt TRUE zurück, wenn der Index der Primärschlüssel ist, sonst FALSE.

```
boolean blnClust = objIndex.PrimaryKey
objIndex.PrimaryKey = blnClust
```

Wenn der Index bereits an die Kollektion angehängt wurde, kann die Eigenschaft nur noch gelesen werden. Der Standardwert ist FALSE.

Unique

Unique

Mit dieser Eigenschaft wird festgelegt oder ermittelt, ob der Index die UNIQUE ist. Sie gibt dann TRUE zurück, sonst FALSE.

```
boolean blnClust = objIndex.Unique
objIndex.Unique = blnClust
```

Wenn der Index bereits an die Kollektion angehängt wurde, kann die Eigenschaft nur noch gelesen werden. Der Standardwert ist FALSE.

Kollektion

Index kann zwei Kollektionen bilden: Columns und Properties. Die Kollektion Columns wird verwendet, wenn der Index mehrere Spalten umfasst. Die Kollektion Properties enthält weitere Eigenschaften.

6.3.5 Key

Dieses Objekt enthält einen Primär-, Fremd- oder Unique-Schlüssel einer Tabelle. Weist die Tabelle mehrere dieser Objekte auf, wird eine Kollektion gebildet.

Methoden

Key kennt keine Methoden. Für die Methoden der Kollektion siehe unter *Kollektion* in diesem Abschnitt.

Eigenschaften

DeleteRule

DeleteRule

Diese Eigenschaft legt fest, wie sich der Schlüssel verhält, wenn Reihen gelöscht werden, die Schlüssel enthalten.

```
long lngValue = objKey.DeleteRule
objKey.DeleteRule = lngValue
```

value	Wert	Beschreibung
adRICascade	1	Änderungen werden kaskadiert, d. h., auch der Fremdschlüssel wird gelöscht.
adRINone	0	Keine Aktion (Standardwert)
adRISetDefault	3	Der Fremdschlüssel wird auf den Standardwert gesetzt, wenn dieser definiert wurde.
adRISetNull	2	Der Fremdschlüssel wird auf Null gesetzt.

Wenn der Schlüssel bereits an eine Kollektion angefügt wurde, kann der Wert nur noch gelesen werden.

Name

Name

Setzt oder liest den Namen des Schlüssels.

```
string strIndexName = objIndex.Name
objIndex.Name = strIndexName

strIndexName = objIndex.Name
objIndex.Name = strIndexName
```

RelatedTable

RelatedTable

Diese Eigenschaft gibt an, mit welcher Tabelle der Schlüssel als Fremdschlüssel verbunden wird. Damit wird die eigentliche Relation definiert.

```
string strRelation = objKey.RelatedTable
objKey.RelatedTable = strRelation
```

Wenn RelatedTable gelesen wird, ohne dass zuvor eine Definition erfolgte, wird eine leere Zeichenkette zurückgegeben.

Type

Diese Eigenschaft legt fest, welchen Schlüsseltyp der Schlüssel hat.

Type

```
integer intType = objKey.Type
objKey.Type = intType
```

Der folgenden Tabelle können Sie die zulässigen Werte entnehmen:

value	Wert	Beschreibung
adKeyPrimary	1	Primärschlüssel (Standardwert)
adKeyForeign	2	Fremdschlüssel
adKeyUnique	3	Unique-Schlüssel

UpdateRule

Diese Eigenschaft legt fest, wie ein Primärschlüssel reagiert, wenn das Schlüsselfeld geändert wird.

UpdateRule

```
long lngValue = objKey.UpdateRule
objKey.UpdateRule = lngValue
```

value	Wert	Beschreibung
adRICascade	1	Änderungen werden kaskadiert, d.h., der Primärschlüssel wird geändert.
adRINone	0	Keine Aktion (Standardwert)
adRISetDefault	3	Der Primärschlüssel wird auf den Standardwert gesetzt, wenn dieser definiert wurde.
adRISetNull	2	Der Primärschlüssel wird auf Null gesetzt.

Wenn der Schlüssel bereits an eine Kollektion angefügt wurde, kann der Wert nur noch gelesen werden.

Kollektion

Die Kollektion vom Typ Key wird mit Column-Objekten gebildet. Mehr Informationen dazu finden Sie im nächsten Abschnitt *Column*.

6.3.6 Column

Ein neues Objekt vom Typ Column legen Sie folgendermaßen an:

```
set objCol = Server.CreateObject("ADOX.Column")
```

Methoden

Append

Append

Diese Methode fügt der `Columns`-Kollektion oder einem `Table`-Objekt eine neue Spalte hinzu.

```
objColumns.Append variant Column [, integer Type] [, long DefinedSize]
```

Die Parameter haben folgende Bedeutung:

- *Column*. Dies kann ein `Column`-Objekt sein, das der Kollektion hinzugefügt werden soll oder eine Zeichenkette, die den Namen der Spalte bestimmt, die erzeugt werden soll.
- *Type*. Dieser Wert ist optional. Sie können hier die unten aufgeführten Konstanten für den Datentyp einsetzen, um die Spalte mit den entsprechenden Eigenschaften zu erzeugen.
- *DefinedSize*. Dieser Wert ist ebenfalls optional. Sie können hier die Größe der Spalte angeben, soweit der gewählte Datentyp diese Eigenschaft unterstützt.

Wenn Sie versuchen, mit Hilfe dieser Methode eine Spalte einer `Index`-Kollektion hinzuzufügen, und die Spalte existiert noch nicht, erhalten Sie einen abfangbaren Laufzeitfehler.

Parameter

Konstante	Beschreibung	DefinedSize sinnvoll
`adBigInt`	8-Byte-Ganzzahl mit Vorzeichen	✓
`adBinary`	Binärzahl	✓
`adBoolean`	Boolescher Wert	
`adBSTR`	Unicode-Zeichenkette, mit /0 endet	✓
`adChar`	Zeichenkette	✓
`adCurrency`	Währung	✓
`adDBDate`	Datum (yyyymmdd)	
`adDBTime`	Zeitwert (hhmmss)	
`adDBTimeStamp`	Datum und Zeit (yyyymmdd-hhmmss.milliardstel)	
`adDecimal`	Exakter numerischer Wert	✓
`adDouble`	Doppelt genaue Gleitkommazahl	✓
`adInteger`	4 Byte Integer mit Vorzeichen	✓
`adLongVarBinary`	Langer Binärwert	✓
`adLongVarChar`	Lange Zeichenkette	?

Konstante	Beschreibung	DefinedSize sinnvoll
adLongVarWChar	Durch /0 begrenzte Zeichenkette	✓
adNumeric	Exakter Zahlenwert	✓
adSingle	Einfache Gleitkommazahl	✓
adSmallInt	2 Byte Integer mit Vorzeichen	✓
adTinyInt	1 Byte Integer mit Vorzeichen	✓
adUnsignedBigInt	8 Byte Integer ohne Vorzeichen	✓
adUnsignedInt	4 Byte Integer ohne Vorzeichen	✓
adUnsignedSmallInt	2 Byte Integer ohne Vorzeichen	✓
adUnsignedTinyInt	1 Byte Integer ohne Vorzeichen	✓
adVarBinary	Binärwert	✓
adVarChar	Zeichenkette	✓
adVariant	OLE-Variante	
asVarWChar	Unicode-Zeichenkette mit /0 beendet	✓
adWChar	Unicode-Zeichenkette mit /0 beendet	✓

Eigenschaften

Attributes

Attributes

Mit dieser Eigenschaft werden spezielle Charakteristika einer Spalte beschrieben.

```
integer intAttr = objColumn.Attributes
objColumn.Attributes = integer intAttr
```

Die folgende Tabelle zeigt die zulässigen Werte für *intAttr*:

Konstante	Wert	Beschreibung
adColFixed	1	Spalte hat eine feste Breite
adColNullable	2	Spalte darf Null-Werte enthalten

DefinedSize

DefinedSize

Beschreibt die definierte Breite der Spalte. Das entspricht der maximalen Anzahl Zeichen, die die Spalte aufnehmen kann.

```
lngSize = objColumn.DefinedSize
objColumn.DefinedSize = lngSize
```

Der Standardwert ist 0.

Name

Name

Setzt oder liest den Namen des Schlüssels.

```
strColumnName = objColumn.Name
objColumn.Name = strColumnName
```

Der Name muss eindeutig sein. Wenn `Name` gelesen wird, ohne dass zuvor eine Definition erfolgte, wird eine leere Zeichenkette zurückgegeben.

NumericScale

NumericScale

Diese Eigenschaft gibt, mit wie vielen Kommastellen ein numerischer Wert gespeichert wird.

```
intScale = objColumn.NumericScale
objColumn.NumericScale = intScale
```

Diese Eigenschaft wird nur eingesetzt, wenn Type den Wert `adDecimal` oder `adNumeric` hat. Für alle anderen Datentypen wird die Zuweisung ignoriert.

▶ `Type`, Seite 264

ParentCatalog

ParentCatalog

Durch die Zuweisung des übergeordneten Katalogs (in der Regel ist das eine Datenbank) kann der Zugriff auf providerspezifische Eigenschaften erfolgen.

```
Set objTbl.ParentCatalog = objCat
```

Der folgende Code-Ausschnitt zeigt, wie eine AUTOINCREMENT-Spalte angelegt wird:

```
Set objCnn = Server.CreateObject("ADODB.Connection")
Set objCat = Server.CreateObject("ADOX.Catalog")
Set objTbl = Server.CreateObject("ADOX.Table")
objCnn.Open "Provider=Microsoft.Jet.OLEDB.4.0;" & _
            "Data Source= c:\Program Files\" & _
            "Microsoft Office\Office\Samples\Northwind.mdb;"
Set objCat.ActiveConnection = objCnn
objTbl.Name = "MyContacts"
Set objTbl.ParentCatalog = objCat
objTbl.Columns.Append "ContactId", adInteger
objTbl.Columns("ContactId").Properties("AutoIncrement") = TRUE
```

```
objTbl.Columns.Append "CustomerID", adVarWChar
objTbl.Columns.Append "FirstName", adVarWChar
objTbl.Columns.Append "LastName", adVarWChar
objTbl.Columns.Append "Phone", adVarWChar, 20
objTbl.Columns.Append "Notes", adLongVarWChar
objCat.Tables.Append objTbl
Set objCat = Nothing
```

Listing 6.4: Anwendung von ParentCatalog zum Zugriff auf providerspezifische Eigenschaften wie z.B. »AutoIncrement«. Im Beispiel wird eine neue Tabelle mit dem Namen MyContacts erzeugt.

Precision

Precision

Diese Eigenschaft gibt, mit wie vielen Stellen vor dem Komma ein numerischer Wert gespeichert wird.

```
intScale = objColumn.Precision
objColumn.Precision = intScale
```

Diese Eigenschaft wird nur eingesetzt, wenn `Type` den Wert `adDecimal` oder `adNumeric` hat. Für alle anderen Datentypen wird die Zuweisung ignoriert.

▶ `Type`, Seite 264

RelatedColumn

RelatedColumn

Für Schlüsselspalten (und nur für solche) kann mit Hilfe dieser Eigenschaft festgelegt werden, zu welcher Spalte eine Relation besteht.

```
objCat.RelatedColumn = strColumn
strColumn = objCat.RelatedColumn
```

Angegeben wird der Name der Spalte. Das folgende Beispiel zeigt eine typische Anwendung:

```
Set objKey = Server.CreateObject("New ADOX.Key")
Set objCat = Server.CreateObject("ADOX.Catalog")
objCat.ActiveConnection = "Provider=Microsoft.Jet.OLEDB.4.0;"&
                          "Data Source=C:\Data\Northwind.mdb;"
objKey.Name = "KundenOrdnung"
objKey.Type = adKeyForeign
objKey.RelatedTable = "Customers"
objKey.Columns.Append "CustomerId"
objKey.Columns("CustomerId").RelatedColumn = "CustomerId"
objKey.UpdateRule = adRICascade
objCat.Tables("Orders").Keys.Append kyForeign
```

Listing 6.5: Definition eines Fremdschlüssels und Anwendung der Eigenschaft RelatedColumn, um die Relation herzustellen

SortOrder

SortOrder

Diese Eigenschaft legt fest, welche Sortierreihenfolge auf die Spalte vorzugsweise angewendet wird.

```
objCat.SortOrder = integer intOrder
integer intOrder = objCat.SortOrder
```

Wenn Sie häufig die Option `ORDER BY` einsetzen, sollten Sie die Sortierung mit dieser Eigenschaft vorab festlegen, um die nachträgliche Sortierung zu vermeiden.

Als Wert für *intOrder* können Sie die folgenden Konstanten einsetzen:

Konstante	Wert	Beschreibung
`adSortAscending`	1	Aufsteigend (Standardwert)
`adSortDescending`	2	Absteigend

Type

Type

Mit dieser Eigenschaft wird der Datentyp bestimmt. Die zulässigen Konstanten können Sie der Tabelle entnehmen.

```
objCat.Type = integer intType
integer intType = objCat.Type
```

Nicht alle Datenbanken können alle Typen unterstützen.

Konstante	Wert	Beschreibung
`adBigInt`	20	8-Byte-Ganzzahl mit Vorzeichen
`adBinary`	128	Binärzahl
`adBoolean`	11	Boolescher Wert
`adBSTR`	8	Unicode-Zeichenkette, mit /0 endet
`adChar`	129	Zeichenkette
`adCurrency`	6	Währung
`adDBDate`	133	Datum (yyyymmdd)
`adDBTime`	134	Zeitwert (hhmmss)
`adDBTimeStamp`	135	Datum und Zeit (yyyymmddhhmmss.milliardstel)
`adDecimal`	14	Exakter numerischer Wert
`adDouble`	5	Doppelt genaue Gleitkommazahl
`adEmpty`	0	Kein Wert
`adError`	10	32-Bit-Fehlercode

Konstante	Wert	Beschreibung
adGUID	72	Globale einmalige ID
adIDispatch	9	Zeiger auf die ID eine OLE-Objekts
adInteger	3	4 Byte Integer mit Vorzeichen
adIUnknown	13	Zeiger auf IUnknown eines OLE-Objekts
adLongVarBinary	205	Langer Binärwert
adLongVarChar	201	Lange Zeichenkette
adLongVarWChar	203	Durch /0 begrenzte Zeichenkette
adNumeric	131	Exakter Zahlenwert
adSingle	4	Einfache Gleitkommazahl
adSmallInt	2	2 Byte Integer mit Vorzeichen
adTinyInt	16	1 Byte Integer mit Vorzeichen
adUnsignedBigInt	21	8 Byte Integer ohne Vorzeichen
adUnsignedInt	19	4 Byte Integer ohne Vorzeichen
adUnsignedSmallInt	18	2 Byte Integer ohne Vorzeichen
adUnsignedTinyInt	17	1 Byte Integer ohne Vorzeichen
adUserDefined	132	Nutzerdefinierter Typ
adVarBinary	204	Binärwert
adVarChar	200	Zeichenkette
adVariant	12	OLE-Variante
asVarWChar	202	Unicode-Zeichenkette mit /0 beendet
adWChar	130	Unicode-Zeichenkette mit /0 beendet

Kollektion

`Column`-Objekte können Kollektionen vom Typ `Property` enthalten.

6.3.7 Group

Dieses Objekt enthält Informationen über eine Benutzergruppe. Mehrere solcher Angaben bilden eine Kollektion.

Methoden

Append

Diese Methode fügt der Kollektion eine weitere Gruppe hinzu.

Append

objGroups.Append *strGroupName*

GetPermissions

GetPermissions

Diese Methode ermittelt die Zugriffsrechte der entsprechenden Gruppe auf ein Objekt.

```
intPerm = objGroup.GetPermissions(string Name, integer ObjectType
                                  [, guid ObjectTypeID])
```

Konstante	Wert	Gruppenmitglieder haben das Recht...
adRightCreate	16 384 (&H4000)	…neue Objekte dieses Typs zu erzeugen (CREATE)
adRightDelete	65 536 (&H10000)	…Daten aus dem Objekt zu löschen (DELETE)
adRightDrop	256 (&H100)	…Objekte dieses Typs zu entfernen (DROP)
adRightExclusive	512 (&H200)	…exclusiv zuzugreifen
adRightExecute	536 870 912 (&H20000000)	…das Objekt auszuführen
adRightFull	268 435 456 (&H10000000)	Alle Zugriffsrechte ohne Einschränkungen
adRightInsert	32 768 (&H8000)	…Daten einzufügen
adRightMaximumAllowed	335 544 32 (&H2000000)	Die maximale Anzahl von Rechten, die der Provider unterstützt
adRightNone	0	Keine Rechte
adRightRead	-2 147 483 648 (&H80000000)	… Daten zu lesen
adRightReadDesign	1024 (&H400)	… die Struktur des Objekts zu lesen
adRightReadPermissions	131072 (&H20000)	… die Rechte des Objekts einzusehen, aber nicht, diese zu ändern
adRightReference	8192 (&H2000)	… eine Referenz auf das Objekt anzulegen
adRightUpdate	1073741824 (&H40000000)	… Veränderungen an den Daten auszulösen (UPDATE)

Konstante	Wert	Gruppenmitglieder haben das Recht...
adRightWithGrant	4096 (&H1000)	… anderen Nutzern Zugriffsrechte einzuräumen. Diese Rechte können nicht umfassender sein als die eigenen (GRANT, REMOVE).
adRightWriteDesign	2048 (&H800)	… die Struktur des Objekts zu verändern (ALTER)
adRightWriteOwner	524288 (&H80000)	… den Besitzer des Objekts zu ändern
adRightWritePermissions	262144 (&H40000)	…die Zugriffsrechte zu ändern.

Die Parameter haben folgende Bedeutung:

Parameter

- Name. Name des Objekts, für das Zugriffsrechte der Gruppe geändert werden sollen. Wenn der Name NULL ist, werden die globalen Zugriffsrechte des Gruppencontainers ermittelt.
- ObjectType. Diese Wert können Sie der folgenden Tabelle entnehmen:

Konstante	Wert	Das Objekt ist ...
adPermObjColumn	2	Spalte
adPermObjDatabase	3	Datenbank
adPermObjProcedure	4	gespeicherte Prozedur
adPermObjProviderSpecific	-1	Providerabhängig
adPermObjTable	1	Tabelle
adPermObjView	5	Sicht (View)

- ObjectTypeId. Optionale Angabe. GUID eines Objekts, das nicht von OLE DB verarbeitet wird. ObjectType muss dabei auf adPermObjProviderSpecific gesetzt werden, sonst wird dieser Parameter ignoriert.

SetPermissions

Diese Methode setzt Zugriffsrechte der Gruppe

SetPermissions

```
objGroup.SetPermissions string Name, integer ObjectType,
                        Action, Rights [, integer Inherit]
                      [, guid ObjectTypeID]
```

- geändert werden sollen. Wenn der Name *NULL* ist, werden die globalen Zugriffsrechte des Gruppencontainers ermittelt.
- *ObjectType*. Diese Wert können Sie der folgenden Tabelle entnehmen:

Konstante	Wert	Das Objekt ist ...
adPermObjColumn	2	Spalte
adPermObjDatabase	3	Datenbank
adPermObjProcedure	4	gespeicherte Prozedur
adPermObjProviderSpecific	-1	Providerabhängig
adPermObjTable	1	Tabelle
adPermObjView	5	Sicht (View)

- *Action*. Auszuführende Aktion. Eingesetzt werden können Konstanten aus der folgenden Tabelle:

Konstante	Wert	Beschreibung
adAccessDeny	3	Das Recht wird entzogen
adAccessGrant	1	Das Recht wird zusätzlich vergeben
adAccessRevoke	4	Explizite Zugriffsrechte werden entzogen
adAccessSet	2	Genau diese Rechte werden vergeben – alle nicht genannten werden entzogen

- *Rights*. Hiermit wird das eigentliche Recht bestimmt, das vergeben oder entzogen wird:

Konstante	Wert	Gruppenmitglieder haben das Recht...
adRightCreate	16 384 (&H4000)	…neue Objekte dieses Typs zu erzeugen (CREATE)
adRightDelete	65 536 (&H10000)	…Daten aus dem Objekt zu löschen (DELETE)
adRightDrop	256 (&H100)	…Objekte dieses Typs zu entfernen (DROP)
adRightExclusive	512 (&H200)	…exclusiv zuzugreifen
adRightExecute	536 870 912 (&H20000000)	…das Objekt auszuführen

Konstante	Wert	Gruppenmitglieder haben das Recht...
adRightFull	268 435 456 (&H10000000)	Alle Zugriffsrechte ohne Einschränkungen
adRightInsert	32 768 (&H8000)	…Daten einzufügen
adRightMaximumAllowed	335 544 32 (&H2000000)	Die maximale Anzahl von Rechten, die der Provider unterstützt
adRightNone	0	Keine Rechte
adRightRead	-2 147 483 648 (&H80000000)	… Daten zu lesen
adRightReadDesign	1 024 (&H400)	… die Struktur des Objekts zu lesen
adRightReadPermissions	131 072 (&H20000)	… die Rechte des Objekts einzusehen, aber nicht, diese zu ändern
adRightReference	8 192(&H2000)	… eine Referenz auf das Objekt anzulegen
adRightUpdate	1 073 741 824 (&H40000000)	… Veränderungen an den Daten auszulösen (UPDATE)
adRightWithGrant	4 096(&H1000)	… anderen Nutzern Zugriffsrechte einzuräumen. Diese Rechte können nicht umfassender sein als die eigenen (GRANT, REMOVE).
adRightWriteDesign	2 048(&H800)	… die Struktur des Objekts zu verändern (ALTER)
adRightWriteOwner	524 288(&H80000)	… den Besitzer des Objekts zu ändern
adRightWritePermissions	262 144(&H40000)	…die Zugriffsrechte zu ändern.

- *Inherit*. Optionaler Wert für die Art der Vererbung:

Konstante	Wert	Beschreibung
adInheritBoth	3	Objekte und Container, die von diesem Objekt abgeleitet werden, erben die Zugriffsrechte
adInheritContainers	2	Nur Container erben die Rechte
adInheritNone	0	Keine Vererbung (Standardwert)

Konstante	Wert	Beschreibung
adInheritNoPropagate	4	Die Attribute adInheritObjects und adInheritContainers werden nicht an die nächste Ebene weitergereicht, so dass eine erneute Vererbung nicht stattfinden kann.
adInheritObjects	1	Alle Objekt, die keine Container sind, erben die Zugriffsrechte

- *ObjectTypeID*. Optionale Angabe. GUID eines Objekts, das nicht von OLE DB verarbeitet wird. *ObjectType* muss dabei auf adPermObjProviderSpecific gesetzt werden, sonst wird dieser Parameter ignoriert.

Hinweise Wenn der Parameter Action auf adAccessRevoke gesetzt wird, überschreibt dies alle anderen Einstellungen.

Eigenschaften

Name

Name Setzt oder liest den Namen der Gruppe.

```
string strGroupName = objGroup.Name
objGroup.Name = string strGroupName
```

Der Name muss eindeutig sein. Wenn Name gelesen wird, ohne dass zuvor eine Definition erfolgte, wird eine leere Zeichenkette zurückgegeben.

Kollektion

Sie können die üblichen Methoden für die Bearbeitung der Kollektion verwenden:

- *Item*. Zugriff auf eine Tabelle der Kollektion
- *Count*. Anzahl der Elemente
- *Delete*. Entfernen eines Eintrags
- *Refresh*. Auffrischen der Kollektion mit den Informationen aus der Datenbank.

6.3.8 User

Dieses Objekt erlaubt den Zugriff auf die Benutzerrechte der Datenbank. Das User-Objekt bildet eine Kollektion Users. Diese ist dann Bestandteil des Group-Objekts. Umgekehrt können Sie in User eine Groups-Kollektion finden, nämlich dann, wenn der Benutzer Mitglied mehrere Gruppen ist.

Methoden

Append

Append

Diese Methode fügt der Kollektion einen weitere Benutzer hinzu.

```
objUsers.Append string strUserName
```

GetPermissions

GetPermissions

Diese Methode ermittelt die Zugriffsrechte des entsprechenden Benutzers auf ein Objekt.

```
intPerm = objGroup.GetPermissions(string Name, integer ObjectType
                                  [, guid ObjectTypeId])
```

Konstante	Wert	Benutzer hat das Recht...
adRightCreate	16 384 (&H4000)	…neue Objekte dieses Typs zu erzeugen (CREATE)
adRightDelete	65 536 (&H10000)	…Daten aus dem Objekt zu löschen (DELETE)
adRightDrop	256 (&H100)	…Objekte dieses Typs zu entfernen (DROP)
adRightExclusive	512 (&H200)	…exclusiv zuzugreifen
adRightExecute	536 870 912 (&H20000000)	…das Objekt auszuführen
adRightFull	268 435 456 (&H10000000)	Alle Zugriffsrechte ohne Einschränkungen
adRightInsert	32 768 (&H8000)	…Daten einzufügen
adRightMaximumAllowed	335 544 32 (&H2000000)	Die maximale Anzahl von Rechten, die der Provider unterstützt
adRightNone	0	Keine Rechte
adRightRead	-2 147 483 648 (&H80000000)	… Daten zu lesen
adRightReadDesign	1024 (&H400)	… die Struktur des Objekts zu lesen
adRightReadPermissions	131072 (&H20000)	… die Rechte des Objekts einzusehen, aber nicht, diese zu ändern

Konstante	Wert	Benutzer hat das Recht...
adRightReference	8192 (&H2000)	… eine Referenz auf das Objekt anzulegen
adRightUpdate	1073741824 (&H40000000)	… Veränderungen an den Daten auszulösen (UPDATE)
adRightWithGrant	4096 (&H1000)	… anderen Nutzern Zugriffsrechte einzuräumen. Diese Rechte können nicht umfassender sein als die eigenen (GRANT, REMOVE).
adRightWriteDesign	2048 (&H800)	… die Struktur des Objekts zu verändern (ALTER)
adRightWriteOwner	524288 (&H80000)	… den Besitzer des Objekts zu ändern
adRightWritePermissions	262144 (&H40000)	…die Zugriffsrechte zu ändern.

Parameter

Die Parameter haben folgende Bedeutung:

- *Name*. Name des Objekts, für das Zugriffsrechte des Benutzers geändert werden sollen. Wenn der Name NULL ist, werden die globalen Zugriffsrechte des Gruppencontainers ermittelt.
- *ObjectType*. Diese Wert können Sie der folgenden Tabelle entnehmen:

Konstante	Wert	Das Objekt ist ...
adPermObjColumn	2	Spalte
adPermObjDatabase	3	Datenbank
adPermObjProcedure	4	gespeicherte Prozedur
adPermObjProviderSpecific	-1	Providerabhängig
adPermObjTable	1	Tabelle
adPermObjView	5	Sicht (View)

- *ObjectTypeId*. Optionale Angabe. GUID eines Objekts, das nicht von OLE DB verarbeitet wird. ObjectType muss dabei auf adPermObjProviderSpecific gesetzt werden, sonst wird dieser Parameter ignoriert.

SetPermissions

SetPermissions

Diese Methode setzt Zugriffsrechte des Benutzers:

```
objGroup.SetPermissions string Name, integer ObjectType,
                        integer Action, Rights [, integer Inherit]
                       [, guid ObjectTypeId]
```

- *Name*. Name des Objekts, für das Zugriffsrechte des Benutzers geändert werden sollen. Wenn der Name NULL ist, werden die globalen Zugriffsrechte des Gruppencontainers ermittelt.
- *ObjectType*. Diese Wert können Sie der folgenden Tabelle entnehmen:

Konstante	Wert	Das Objekt ist ...
adPermObjColumn	2	Spalte
adPermObjDatabase	3	Datenbank
adPermObjProcedure	4	gespeicherte Prozedur
adPermObjProviderSpecific	-1	Providerabhängig
adPermObjTable	1	Tabelle
adPermObjView	5	Sicht (View)

- *Action*. Auszuführende Aktion. Eingesetzt werden können Konstanten aus der folgenden Tabelle:

Konstante	Wert	Beschreibung
adAccessDeny	3	Das Recht wird entzogen
adAccessGrant	1	Das Recht wird zusätzlich vergeben
adAccessRevoke	4	Explizite Zugriffsrechte werden entzogen
adAccessSet	2	Genau diese Rechte werden vergeben – alle nicht genannten werden entzogen

- *Rights*. Hiermit wird das eigentliche Recht bestimmt, das vergeben oder entzogen wird:

Konstante	Wert	Gruppenmitglieder haben das Recht...
adRightCreate	16 384 (&H4000)	…neue Objekte dieses Typs zu erzeugen (CREATE)
adRightDelete	65 536 (&H10000)	…Daten aus dem Objekt zu löschen (DELETE)
adRightDrop	256 (&H100)	…Objekte dieses Typs zu entfernen (DROP)
adRightExclusive	512 (&H200)	…exclusiv zuzugreifen

Konstante	Wert	Gruppenmitglieder haben das Recht...
adRightExecute	536 870 912 (&H20000000)	…das Objekt auszuführen
adRightFull	268 435 456 (&H10000000)	Alle Zugriffsrechte ohne Einschränkungen
adRightInsert	32 768 (&H8000)	…Daten einzufügen
adRightMaximumAllowed	335 544 32 (&H2000000)	Die maximale Anzahl von Rechten, die der Provider unterstützt
adRightNone	0	Keine Rechte
adRightRead	-2 147 483 648 (&H80000000)	… Daten zu lesen
adRightReadDesign	1 024 (&H400)	… die Struktur des Objekts zu lesen
adRightReadPermissions	131 072 (&H20000)	… die Rechte des Objekts einzusehen, aber nicht, diese zu ändern
adRightReference	8 192 (&H2000)	… eine Referenz auf das Objekt anzulegen
adRightUpdate	1 073 741 824 (&H40000000)	… Veränderungen an den Daten auszulösen (UPDATE)
adRightWithGrant	4 096 (&H1000)	… anderen Nutzern Zugriffsrechte einzuräumen. Diese Rechte können nicht umfassender sein als die eigenen (GRANT, REMOVE).
adRightWriteDesign	2 048 (&H800)	… die Struktur des Objekts zu verändern (ALTER)
adRightWriteOwner	524 288 (&H80000)	… den Besitzer des Objekts zu ändern
adRightWritePermissions	262 144 (&H40000)	…die Zugriffsrechte zu ändern.

- *Inherit*. Optionaler Wert für die Art der Vererbung:

Konstante	Wert	Beschreibung
adInheritBoth	3	Objekte und Container, die von diesem Objekt abgeleitet werden, erben die Zugriffsrechte
adInheritContainers	2	Nur Container erben die Rechte
adInheritNone	0	Keine Vererbung (Standardwert)
adInheritNoPropagate	4	Die Attribute adInheritObjects und adInheritContainers werden nicht an die nächste Ebene weitergereicht, so dass eine erneute Vererbung nicht stattfinden kann.
adInheritObjects	1	Alle Objekt, die keine Container sind, erben die Zugriffsrechte

- *ObjectTypeID*. Optionale Angabe. GUID eines Objekts, das nicht von OLE DB verarbeitet wird. *ObjectType* muss dabei auf adPermObjProviderSpecific gesetzt werden, sonst wird dieser Parameter ignoriert.

Hinweise

Wenn der Parameter Action auf adAccessRevoke gesetzt wird, überschreibt dies alle anderen Einstellungen.

ChangePassword

ChangePassword

Diese Methode ändert das Kennwort für einen User.

```
objUser.ChangePassword string OldPassword, string NewPassword
```

Parameter

- *OldPassword*. Eine Zeichenkette mit dem alten Kennwort. Wenn noch nie ein Kennwort vergeben wurde, tragen Sie hier eine leere Zeichenkette ein.
- *NewPassword*. Das neue Kennwort

Hinweise

Die Angabe des alten Kennwortes ist aus Sicherheitsgründen notwendig. Wenn der Provider keine Vertrauensstellungen unterstützt, ist diese Methode nicht anwendbar.

Eigenschaften

Name

Name

Setzt oder liest den Namen des Benutzer.

```
string strUserName = objUser.Name
objUser.Name = string strUserName
```

Der Name muss eindeutig sein. Wenn `Name` gelesen wird, ohne dass zuvor eine Definition erfolgte, wird eine leere Zeichenkette zurückgegeben.

Kollektion

Sie können die üblichen Methoden für die Bearbeitung der Kollektion verwenden:

- `Item`: Zugriff auf eine Tabelle der Kollektion
- `Count`: Anzahl der Elemente
- `Delete`: Entfernen eines Eintrags
- `Refresh`: Auffrischen der Kollektion mit den Informationen aus der Datenbank

6.3.9 Procedure

Dieses Objekt dient der Erzeugung und Verwaltung gespeicherter Prozeduren. Die so erzeugten Prozeduren können an ein ADO-`Command`-Objekt übergeben werden.

Oft werden die gespeicherte Prozeduren als Kollektion `Procedures` dem `Catalog`-Objekt entnommen.

Das folgende Beispiel zeigt, wie alle gespeicherten Prozeduren in einer Datenbank angezeigt werden können:

```
if open() then
   set objCat = Server.CreateObject("ADOX.Catalog")
   set objCat.ActiveConnection = objConn
   for each objProc in objCat.Procedures
      echo objProc.name & "<br>"
   next
end if
```

Listing 6.6: ADOX.Procedures.asp: Ausgabe aller Namen der in einer Datenbank gespeicherten Prozeduren

Methoden

Dieses Objekt hat keine Methoden.

Eigenschaften

Command

Command

Mit dieser Eigenschaft wird ein ADO-Command-Objekt an das ADOX-Procedure-Objekt übergeben oder von diesem entnommen:

```
objProc.Command = object objCommand
```

DateCreated

DateCreated

Zeigt das Datum an, an dem die Prozedur erzeugt wurde.

```
date datProc = objProc.DateCreated
```

DateModified

DateModified

Zeigt das Datum an, an dem die Prozedur zuletzt geändert wurde.

```
date datProc = objProc.DateModified
```

Name

Name

Zeigt oder setzt den Namen der Prozedur.

```
string strProcName = objProc.Name
objProc.Name = string strProcName
```

Der Name muss eindeutig sein. Wenn `Name` gelesen wird, ohne dass zuvor eine Definition erfolgte, wird eine leere Zeichenkette zurückgegeben.

Kollektion

Dieses Objekt bildet die Kollektion `Properties` im Objekt `Catalog`, nicht jedoch unabhängig davon. Selbst enthält es keine Kollektionen.

6.3.10 View

Mit diesem Objekt erzeugen und verändern Sie Sichten. Sichten bieten eine vordefinierte Abfrageform. Nicht alle Datenbanken unterstützen Sichten.

Methoden

Dieses Objekt hat keine Methoden.

Eigenschaften

Command

Command

Mit dieser Eigenschaft wird ein ADO-Command-Objekt an das ADOX-`View`-Objekt übergeben oder von diesem entnommen:

```
objView.Command = object objCommand
```

DateCreated

DateCreated

Zeigt das Datum an, an dem die Sicht erzeugt wurde.

```
date datView = objView.DateCreated
```

DateModified

DateModified

Zeigt das Datum an, an dem die Sicht zuletzt geändert wurde.

```
date datView = objView.DateModified
```

Name

Name

Zeigt oder setzt den Namen der Sicht.

```
string strViewName = objView.Name
objView.Name = string strViewName
```

Der Name muss eindeutig sein. Wenn `Name` gelesen wird, ohne dass zuvor eine Definition erfolgte, wird eine leere Zeichenkette zurückgegeben.

7 ADO MD

ADO MD steht für *ADO Multidimensional Data*. Damit steht ein Weg zur Verfügung, um Daten multidimensional zu verarbeiten. Diese Möglichkeit bietet der SQL Server ab Version 7 auch mit verschiedenen SQL-Anweisungen, wie z.B. `CUBE`. Mit ADO MD ist es nun möglich, diese Aufgabe vom OLE DB-Provider erledigen zu lassen und damit eine größere Unabhängigkeit von der Datenbank zu erreichen.

7.1 OLAP

Voraussetzung für die Nutzung von ADO MD ist der Zugriff auf eine entsprechende Datenbank. Sie benötigen dazu den SQL Server mit den OLAP-Komponenten. OLAP steht für *Online Analytical Processing* und dient als Oberbegriff für die computergestützte Auswertung komplexer, mehrdimensionaler Daten. Solche Daten fallen bei allen Arten verschachtelter Vorgänge an.

7.1.1 Installation der OLAP-Erweiterung

SQL Server

Die Installation der OLAP-Erweiterung erfolgt von der SQL Server-CD.

Wählen Sie dazu den Punkt SQL SERVER 7.0 OLAP SERVICES aus. Es startet ein Installationsassistent, der das Zielverzeichnis und den Programmordner für eine neue Programmgruppe abfragt. Die Installation benötigt etwa 60 Mbyte.

7.1.2 Beispieldatenbank

Nutzen Sie die Beispieldaten vom SQL Server!

Gerade für die ersten Schritte ist es schwer, passende Daten zu finden. Sie sollten daher mit den mitgelieferten Daten anfangen. Analog zur *Northwind*-Datenbank des SQL Servers ist im Lieferumgang der OLAP-Services die *FoodMart*-Datenbank enthalten. Auf diese Daten beziehen sich alle Beispiele dieses Abschnitts. Die Datenbank selbst liegt im Access-97-Format vor. Sie wird vom OLAP-Manager des SQL Servers über ODBC angesprochen. Wenn Sie nur mit dem SQL Server arbeiten, lohnt der Entwurf in Access nicht, Sie sollten dann die Werkzeuge des SQL Servers verwenden.

Abbildung 7.1: OLAP-Installation

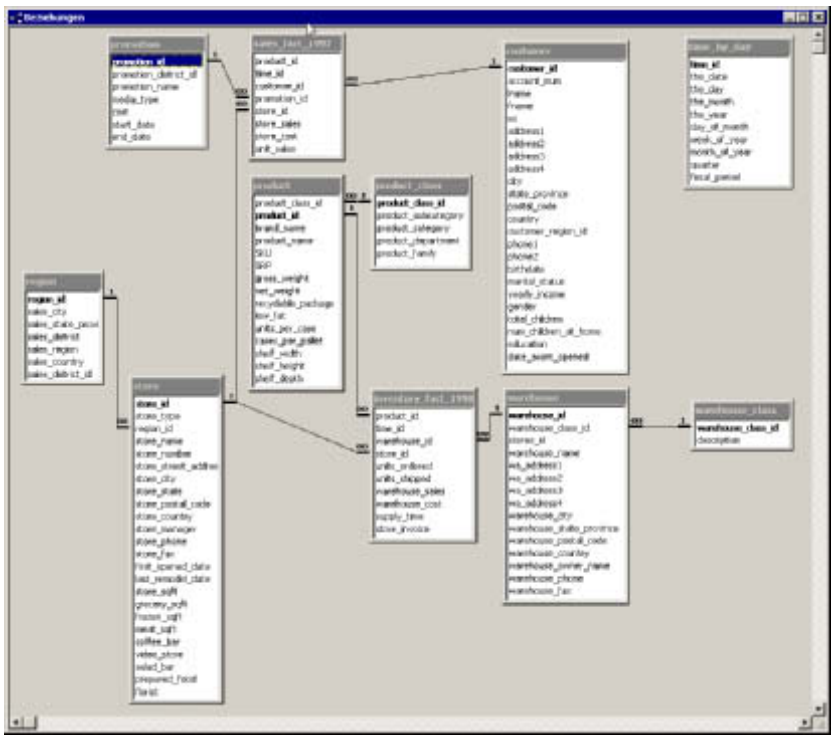


Abbildung 7.2: Ansicht der Food-Mart-Datenbank in Access mit den Tabellenbeziehungen

Nutzung der OLAP-Werkzeuge

Mit dem Start des SQL Servers, stehen die OLAP-Werkzeuge zur Verfügung. In der Regel finden Sie diese in der SQL Server-Programmgruppe unter OLAP SERVICES. Diese Oberfläche erscheint als MMC-Snap-In und dürfte wenig Schwierigkeiten bei der Bedienung bereiten. Die Abbildung der Daten und die Definition der mehrdimensionalen Modelle erfolgt im Zweig CUBES, die Verknüpfung mit den physikalischen Daten, also der flachen Darstellung, ist unter BIBLIOTHEKEN zu finden. Hier wird auch die Verknüpfung mit der ODBC-Quelle erstellt. Bei der Standardinstallation sind allerdings alle Angaben schon erfolgt. Sie müssen für *FoodMart* keine Einstellungen vornehmen. Ein Blick in die Dialogfelder lohnt sich aber als Vorbereitung auf dieses Kapitel.

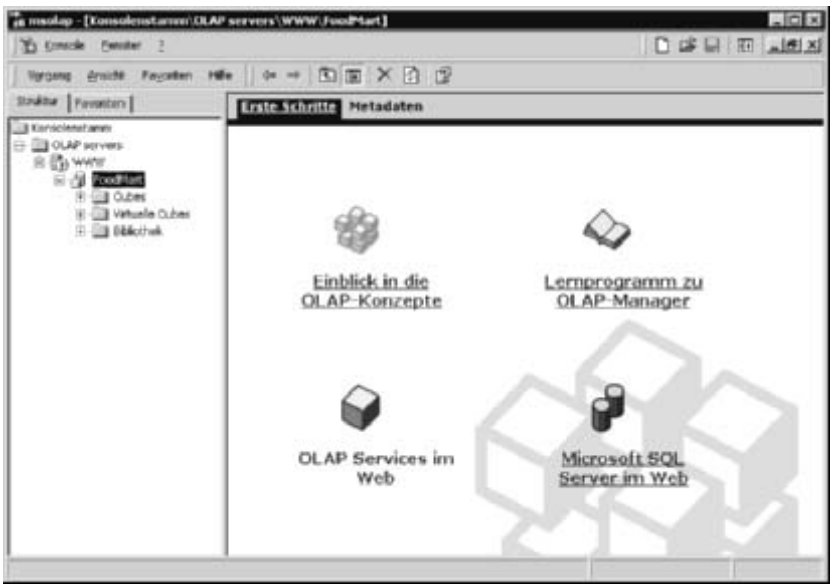


Abbildung 7.3: OLAP in Aktion: das Snap-In der OLAP Services (msolap)

7.1.3 Die Prozesssprache

Ebenso wie bei ADO wird die eigentliche Abfrage der Datenbank mit SQL-Kommandos initiiert. Der SQL Server bietet dazu einen speziellen Dialekt. Dieser Dialekt wird als MDX (*OLAP Extensions for SQL*) bezeichnet. Er ist in der Dokumentation vollständig dargestellt. Eine kurze Einführung soll an dieser Stelle genügen.

Die Abfragesprache ähnelt der bekannten SQL-Form. Die folgende Darstellung ist verhältnismäßig abstrakt, erleichtert aber das Lesen der nachfolgenden Abschnitte.

Abfrage von multidimensionalen Daten

Multidimensionale Daten werden in einem mehrachsigen System dargestellt. Die folgende Anweisung ruft solche Daten ab:

SELECT FROM

```
SELECT <axis> FROM <cube> WHERE <selection>
```

Die Auswahl der Achse erlaubt die Angabe des Ortes, wo die Daten im Cube gespeichert sind. Diese Sicht ist virtueller als bei einer einfachen Abfrage von Tabellen. Wie die Auswahl erfolgt, zeigt eine konkrete Abfrage:

ON COLUMNS

```
SELECT product.Children ON COLUMNS FROM Warehouse
```

Hier erfolgt die Auswahl nach Spalten. Auch die Auswahl nach Reihen ist möglich:

ON ROWS

```
SELECT Product.Children ON COLUMNS,
      [Store Type].Children ON ROWS
  FROM Warehouse
```

Der Name der Dimension *Store Type* wurde nur deshalb in eckige Klammern gesetzt, weil er ein Leerzeichen enthält. Insofern unterscheidet sich die Syntax nicht vom bekannten Transact-SQL.

Die letzte Abfrage verwendete bereits zwei Dimensionen: Spalten und Reihen bilden eine zweidimensionale Fläche. Um eine dritte Dimension hinzuzufügen, wird die Anweisung `ON PAGES` ergänzt:

```
SELECT Product.Children ON COLUMNS,
      [Store Type].Children ON ROWS
       Store.Children ON PAGES
  FROM Warehouse
```

Multiple Daten auf den Achsen

Auf jeder Achse befinden sich in einfachen Modellen nur einfache – eindimensionale – Daten. In der Praxis können aber auch dort mehrere Daten sein. Um eine solche Verknüpfung darzustellen, existiert ein dem `JOIN` ähnliches Verfahren: `CROSSJOIN`. Hier ein Beispiel:

```
SELECT Product.Children ON COLUMNS
     CROSSJOIN ([Store Type].Children,
                Store.Children) ON ROWS
```

Auf diese Weise werden zwei »DataSets« verknüpft. Angenommen, der erste Satz enthält folgende Daten:

```
Spielzeug
Spiele
```

Der zweite Satz enthält folgende Werte:

```
Bücher
Fahrräder
```

Die oben dargestellte Abfrage ergibt somit folgendes Bild:

```
Spielzeug, Bücher
Spielzeug, Fahrräder
Spiele, Bücher
Spiele, Fahrräder
```

Filtern

In der `WHERE`-Bedingung werden Filter für die ausgewählten Daten angegeben. Dies unterscheidet sich nur wenig von SQL. Das folgende Beispiel zeigt die Daten des Jahres 1999 an:

```
SELECT Product.Children ON COLUMNS,
      [Store Type].Children ON ROWS
   FROM Warehouse
   WHERE ([1999])
```

Dies ist eine verkürzte Form, denn der OLAP-Server erkennt, dass die Jahreszahl nur Bestandteil der Dimension *`time`* sein kann. Besser lesbar ist die folgende Variante:

```
SELECT Product.Children ON COLUMNS,
      [Store Type].Children ON ROWS
   FROM Warehouse
   WHERE ([Time.1999])
```

Vielleicht haben Sie eine Schreibweise wie `Time = 1999` erwartet. Die Daten sind in der OLAP-Welt aber als Objekte zu verstehen und lassen sich aus der übergeordneten Dimension abgeleiten. Deshalb ist die Schreibweise auch, ähnlich wie bei Objekten, mit einem Punkt als Trennzeichen möglich.

7.2 Einführung

Ebenso wie bei ADO und ADOX wird ein spezieller OLE DB-Provider für ADO MD benutzt, der so genannte MDP.

7.2.1 Einführung in multidimensionale Schemata

Das zentrale Objekt in ADO MD ist `Cube`. Dieses Objekt besteht aus einem strukturierten Satz von verwandten Dimensionen, Hierarchien, Ebenen und Mitgliedern.

Begriffe

Dimension Eine Dimension (`Dimension`) ist eine unabhängige Kategorie einer multidimensionalen Datenbank, die eine Geschäftseinheit abbildet. Eine Dimension enthält typischerweise Einheiten, die als Abfragekriterien für die Datenbank verwendet werden.

Hierarchie (Hierarchy) Eine Hierarchie (`Hierarchy`) ist eine Methode der Zusammenfassung der Elemente einer Dimension. Eine Dimension kann Elemente in verschiedenen Ebenen enthalten, die untereinander eine Art Eltern-Kind-Beziehung haben. Die Hierarchie definiert, wie diese Elemente verbunden sind.

Ebene (Level) Eine Ebene (`Level`) ist ein Schritt in der Zusammenfassung der Elemente in der Hierarchie. Enthält eine Dimension mehrere Schichten, bildet jede Schicht eine Ebene.

Mitglieder (Member) Als Mitglied (`Member`) wird ein Datenobjekt der Dimension bezeichnet. Sie können auch Überschriften oder Einheiten angeben, die beschreiben, wie Datenbankobjekte behandelt werden sollen.

Die ADO MD-Objekte

Cubes werden durch das `CubeDef`-Objekt in ADO MD repräsentiert. Die Darstellung der Dimensionen, Hierarchien, Ebenen und Mitglieder erfolgt durch entsprechende ADO MD-Objekte:

- `Dimension`
- `Hierarchy`
- `Level`
- `Member`

7.3 ADO MD-Objekte im Detail

Die oberste Instanz in ADO MD ist der Cube. Dieser repräsentiert eine Datenbank im herkömmlichen ADO.

7.3.1 Dimensionen

Wie eine Dimension in der Praxis aussieht, hängt vom verwendeten Geschäftsmodell ab. Üblicherweise ist eine Dimension ein Ausgangspunkt für die Auswahl von Daten. Daten können zueinander in einer sehr komplexen Beziehung stehen. ADO MD kann solche Beziehungen besser abbilden als ADO allein.

Beispiel-Szenario Stellen Sie sich vor, Sie haben einen Cube mit Verkaufsdaten. Dieser enthält fünf Dimension: *Verkaeufer, Regionen, Zeiten, Produkte, Umsaetze.* Umsaetze enthält die aktuellen Daten der Aktivitäten der Verkäufer, die übrigen

Dimensionen repräsentieren die unterlegten Daten und bieten Möglichkeiten der Strukturierung und Selektion.

Die Dimension *Regionen* hat folgende Mitglieder:

```
{Alle, Europa, Deutschland, Frankreich, England, Bayern, Brandenburg,
Berlin, Frankfurt ....}
```

Die Dimension *Verkaeufer* enthält folgende Namen:

```
{Alle, ...}
```

7.3.2 Hierarchien

Hierarchien definieren, wie die Ebenen der Dimension gruppiert werden können. Eine Dimension kann mehr als eine Hierarchie enthalten. Für die Dimension *Regionen* wäre eine mögliche Hierarchie folgende:

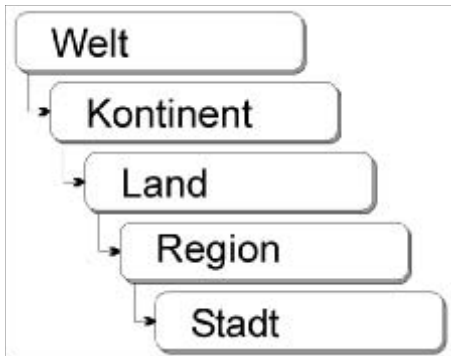


Abbildung 7.4: Hierarchie einer Dimension für geografische Informationen

7.3.3 Ebenen

Wenn Sie die in Abbildung 7.4 vorgestellte Hierarchie betrachten, lassen sich daraus leicht fünf Ebenen ableiten, hier mit entsprechenden Mitgliedern:

- Welt = {`All`}
- Kontinent = {Amerika, Asien, Europa, ...}
- Land = {USA, Kanada, Japan, Deutschland, Frankreich, ...}
- Region = {USA-Westcoast, USA-Eastcoast, Brandenburg, Bayern, ...}
- Stadt = {Berlin, New York, Ontario, …]

7.3.4 Mitglieder

Mitglieder am Ende der Hierarchie haben keine weiteren untergeordneten Einheiten, sie bilden die Blätter des Hierarchiebaums. Das Mitglied an der Wurzel hat keine übergeordnete Einheit (kein Elternteil). Mitglieder können also auf allen Ebenen angeordnet werden.

Die folgende Darstellung zeigt, wie die Elemente der Dimension Regionen zueinander in Beziehung stehen:

- `{All}` ist Elternteil von `{Europa, Amerika, Asien}`
- `{Europa}` ist Elternteil von `{Deutschland, Frankreich}`
- `{Deutschland}` ist Elternteil von `{Bayern, Brandenburg, ...}`
- `{Bayern}` ist Elternteil von `{München, Freiburg, Passau, ...}`

Mitglieder können in einer Dimension zugleich in mehreren Hierarchien abgebildet werden. Das folgende Beispiel zeigt, wie in der Dimension *Zeiten* damit umgegangen wird. Jedes Element der Dimension, das einen Verkaufszeitpunkt repräsentiert, kann beispielsweise in einer oder beiden der folgenden Hierarchien dargestellt werden:

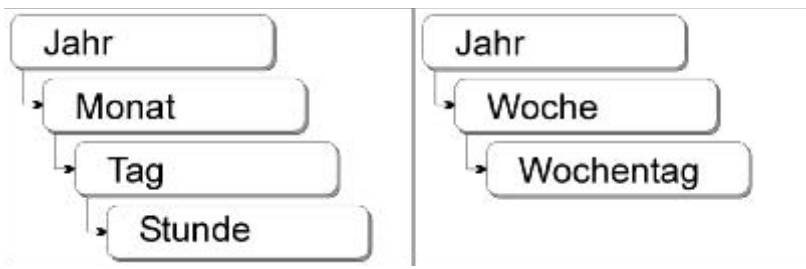


Abbildung 7.5: Verschiedene Hierarchien der Dimension Zeiten

Mitglieder existieren natürlich nur ein Mal in der Dimension, auch wenn Sie in mehreren Hierarchien erfasst werden. Jede diese Hierarchien muss aber nicht alle Mitglieder erfassen. So wären die in der Ebene *Stunde* aufgeführten Elemente nicht zwangsweise in der *Wochen*-Hierarchie (rechts in Abbildung 7.5) zu finden.

7.3.5 Mit multidimensionalen Daten arbeiten

Die Abfrageergebnisse unterscheiden sich von den aus ADO bekannten Datensätzen. Gesprochen wird nun von Zellsätzen, Zellen und Position auf Achsen.

Zellsätze und Achsen

Ein (Zellsatz) `Cellset` stellt das Ergebnis einer Abfrage multidimensionaler Daten dar. Im Gegensatz zum Datensatz, der Ergebnisliste einer normalen

Abfrage, ist ein Zellsatz eine Kollektion von Achsen. Normalerweise werden nicht mehr als vier Achsen benötigt, häufig nur ein oder zwei. Eine Achse ist eine Kollektion von Mitgliedern einer oder mehrerer Dimensionen, welche verwendet werden, um Werte im Cube zu spezifizieren.

Positionen

Eine Position ist ein Punkt auf einer Achse. Besteht eine Achse nur aus einer Untermenge einer Dimension, sind Positionen eine Untermenge der Mitglieder. Enthält eine Achse dagegen Daten aus mehreren Dimensionen, ist jede Position eine Komposition aus Einheiten mit `n` Komponenten, wobei `n` die Anzahl der Dimensionen ist. Jeder Teil der Komposition ist ein Mitglied einer entsprechenden Dimension.

Im bereits eingeführten Beispiel wären Produktdaten und regionale Abfragen gemeinsam abfragbar. Daraus könnte man ableiten, welche Artikel in welcher Region verkauft wurden – eine normalerweise relativ komplexe Abfrage. Auf der X-Achse des Zellsatzes stünden dann kombinierte Positionen, jeweils bestehend aus der Informationen zu Region (z.B. Berlin) und dem Produkt (Computer).

Zellen

Eine Zelle ist ein Objekt am Schnittpunkt der Achsen-Koordinaten. Jede Zelle enthält mehrere mit ihr verbundene Informationen und die eigentlichen Daten der Abfrage. Die Daten werden auch als formatierte Zeichenkette für die Ausgabe zur Benutzerschnittstelle vorgehalten. Jede Zelle kann nur einen Wert enthalten. Außerdem besitzt jede Zelle einen ordinalen Wert, beginnend mit 0 für die erste Zelle des Zellsatzes. Die Zellen zählen fortlaufend. Enthält ein Zellsatz sechzehn Spalten, hat die erste Zelle der zweiten Reihe (bei einem zweidimensionalen Zellsatz) den Wert 16.

7.3.6 Beispiel für einen Zellsatz

Betrachten Sie das bereits angesprochene Beispiel mit den Verkaufsinformationen genauer. Die verfügbaren Dimensionen waren:

- *Verkaeufer*
- *Regionen*
- *Zeiten*
- *Produkte*
- *Umsaetze*

Die folgende Abbildung zeigt eine explorative Darstellung des Zellsatzes für Verkäufe des Jahres 2000:

Abbildung 7.6: Zellsatz des Beispiel-Cube

		Jürgen Mayer				Bernd Schuster			
		Deutschland				Deutschland			
		Bayern			Berlin	Bayern			Berlin
		M	P	N		M	P	N	
2000	Q1	00	05	10	15	20	25	30	35
	Q2	01	06	11	16	21	26	31	36
	Q3	02	07	12	17	22	27	32	37
	Q4	03	08	13	18	23	28	33	38
	2001	04	09	14	19	24	29	34	39

Diese Tabelle ließe sich in jeder Richtung fortsetzen. Den Inhalt der Zellen hier die Zellennummer bildet. Die Charakteristik der Achsen ist Folgende:

- Dimensionen der Achsen: *Zeiten*, *Verkaeufer*, *Regionen*

  In Worten: »Zeige die Daten geordnet nach Zeiten, Verkäufern und Regionen an.« Dies bestimmt die Anordnung der Zellen.

- Filter-Dimensionen: *Umsaetze*, *Jahre*, *Produkte*

  In Worten: »Wähle die Daten nach Umsätzen, Jahren und Produkten aus.« Dies bestimmt den Inhalt der Zellen.

- Es werden zwei Achsen verwendet: X- und Y-Achse. Auf der X-Achse werden Verkäufer und Regionen abgebildet, die Y-Achse enthält die Zeiten.

  - X-Achse: Die X-Achse enthält zwei verschachtelte Dimensionen. In der ersten Ebene werden die Verkäufer dargestellt, in der zweiten und dritten Ebene zwei Ebenen der Hierarchie *Regionen*.

  - Y-Achse: Hier wird die Zeit in zwei Ebenen abgebildet, Jahre und Quartale.

Der gesamte Zellsatz enthält, so wie er in Abbildung 7.6 dargestellt wurde, acht Positionen auf der X-Achse: zwei Positionen liefern die ausgewählten Verkäufer und vier Positionen die Regionen (2 x 4 = 8). In der Mengendarstellung sieht das folgendermaßen aus:

```
{Verkaeufer, Region}
```

Für Jürgen Mayer ergibt sich folgende Mengendarstellung:

```
{{Jürgen Mayer, M}, {Jürgen Mayer, P}, {Jürgen Mayer, N},
 {Jürgen Mayer, Bayern}, {Jürgen Mayer, Berlin},
 {Jürgen Mayer, Deutschland}
}
```

Die Y-Achse hat nur eine Dimension: *Zeiten*. Dort sind folgende Positionen zu finden:

```
{Q1, Q2, Q3, Q4, 2001}
```

Der Zugriff auf diese Elemente erfolgt über entsprechende Objekte, die im nächsten Abschnitt ausführlich vorgestellt werden. Für die Elemente *Zellsatz*, *Zelle*, *Achse* und *Position* sind die Objekte `Cellset`, `Cell`, `Axis` und `Position` zuständig.

7.4 ADO und ADO MD im Vergleich

ADO und ADO MD besitzen unterschiedliche aber miteinander verwandte Objektmodelle. ADO liefert Objekte zum Verbinden mit Datenquellen, zur Ausführung von Kommandos auf einem Datenbankmanagementsystem, zum Ermitteln von Tabelleninformationen und Schemata-Daten sowie zur Fehlerbehandlung. ADO MD besitzt Objekte zum Abfragen multidimensionaler Daten und zum Ermitteln der Schemata multidimensionaler Datenquellen.

Bei der Arbeit mit multidimensionalen Daten, können Sie zwischen ADO, ADO MD oder der Kombination aus beidem wählen. Gerade in der Kombination verfügen Sie über eine außerordentlich leistungsstarke Programmierumgebung.

7.4.1 Multidimensionale Datenmodelle in ADO

Wenn Sie mit ADO arbeiten, werden die Daten einer multidimensionalen Datenquelle in einer flachen, tabellarischen Form dargestellt. Für Nutzer ist diese Sicht oft klarer und einfacher in der Handhabung. Diese Sicht erzielen Sie, wenn das ADO-Objekt `RecordSet` verwendet wird. Wenn Sie die Quelle des `Cellset`-Objekts als Quelle in der `Open`-Methode des `RecordSet`-Objekts verwenden, erhalten Sie eine solche tabellarische Sicht des Zellensatzes.

Auch die Sicht der schematischen Daten – also der Beschreibung der Struktur der Datenquelle – ist oft in tabellarischer Sicht einfacher zu beherrschen. Die ADO-Methode `OpenSchema` erlaubt derartige Abfragen. Der abgefragte Datensatz enthält dann Informationen über die Struktur der Tabellen. Um die Struktur eines Cube abzufragen, gibt es einige neue Werte für `SchemaEnum`:

- `adSchemaCubes`
- `adSchemaDimensions`
- `adSchemaHierarchies`

- `adSchemaLevels`
- `adSchemaMeasures`
- `adSchemaMembers`

Andere Konstanten aus ADO sind auch verwendbar, der Zugriff auf die Datei ADOVBS.INC sollte also auch in einer ADO MD-Umgebung gewährleistet sein.

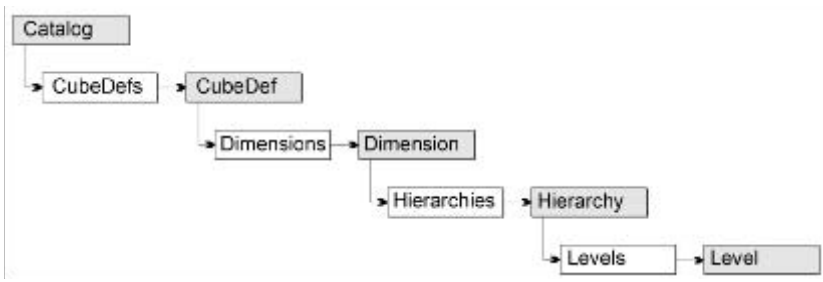
7.5 Die ADO MD-Objekte

ADO MD ist, ebenso wie ADO, eine ActiveX-Bibliothek. Die Programm-ID zur Referenzierung ist »ADOMD«. Der Dateiname der DLL dieser Bibliothek lautet DOMD.DLL.

7.5.1 ADO MD-Objektmodell

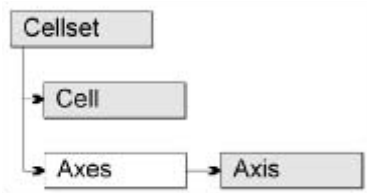
Die folgende Darstellung zeigt die grundlegenden Objekte (grau hinterlegt) und die Kollektionen, die aus diesen Objekten gebildet werden können (weißer Hintergrund).

Abbildung 7.7: Basisobjekte in ADO MD



Die Elemente werden in einer eigenen Objektstruktur dargestellt, wie die folgende Abbildung zeigt:

Abbildung 7.8: Struktur des Objekts CellSet und abgeleiteter Objekte



Die Objekte `Axis` und `Cell` enthalten eine Kollektion `Positions` aus `Position`-Objekten:

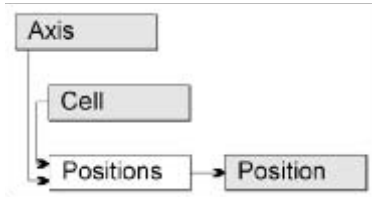


Abbildung 7.9: Ableitung der Position-Kollektion aus dem Axis- und dem Cell-Objekt

Die Objekte `Level` und `Position` wiederum können `Member`-Objekte enthalten:

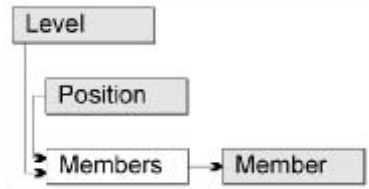


Abbildung 7.10: Ableitung des Objekts Member aus Level und Position

Das Objekt `Property`, üblicherweise als Sammlung in Form einer Kollektion `Properties` betrachtet, kann aus fast allen anderen Objekten abgeleitet werden:

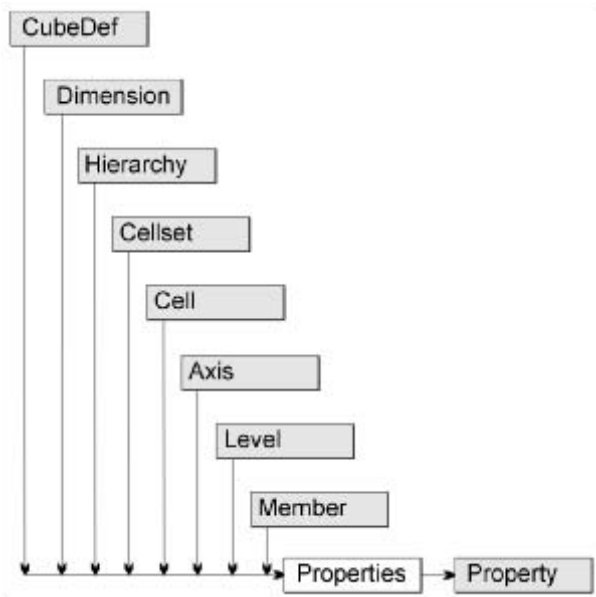


Abbildung 7.11: Ableitung des Objekts Property

7.5.2 ADO MD-Kollektionen

Wie bereits in der Vorstellung der Objekte zu sehen war, nutzt auch ADO MD an vielen Stellen Kollektionen, um zusammengehörige Objekte zu verwalten. Die folgende Tabelle zeigt alle Kollektionen:

Tabelle 7.1: Kollektionen in ADO MD

Kollektion	Beschreibung
`Axes`	Enthält Objekte des Typs `Axis`
`CubeDefs`	`CubeDefs` enthält alle multidimensionalen Objekte eines Katalogs (Cube)
`Dimensions`	Enthält die `Dimension`-Objekte
`Hierachies`	Enthält die `Hierarchy`-Objekte einer Dimension
`Levels`	Enthält die Ebenen (`Level`-Objekte) einer Hierarchie
`Members`	Enthält die Mitglieder einer Position oder Ebene
`Positions`	Enthält `Position`-Objekte auf einer Achse

7.6 ADO MD – Syntaxdiagramme

Dieser Abschnitt zeigt alle Objekte, Methoden und Eigenschaften aus der Bibliothek ADO MD und deren Einsatzformen.

7.6.1 Übersicht

- `Axis`, Seite 294
- `Catalog`, Seite 294
- `Cell`, Seite 296
- `Cellset`, Seite 297
- `CubeDef`, Seite 300
- `Dimension`, Seite 302
- `Hierarchy`, Seite 303
- `Level`, Seite 305
- `Member`, Seite 306
- `Position`, Seite 312

7.6.2 Den OLAP-Provider ansprechen

Für die Nutzung von ADO MD gelten folgende Voraussetzungen:

- Ein im Netzwerk oder lokal verfügbarer OLAP-Server, vorzugsweise der SQL Server 7 / 2000
- Eine Datenbank
- Der installierte Provider (Installation erfolgt von der SQL Server-CD zusammen mit den OLAP-Diensten)

Das oberste Objekt der Hierarchie ist der Katalog:

```
set objCat = Server.CreateObject("ADOMD.Catalog")
```

Anschließend wird die Verbindung zu diesem Katalog initiiert. Der Provider hat den Namen »MSOLAP«. Alle übrigen Parameter entsprechen denen der in Kapitel 1 vorgestellten Verbindungszeichenfolge:

```
objCat.ActiveConnection = "Data Source = <server>;
                           Catalog = <catname>;
                           Provider = MSOLAP"
```

Der Parameter *<server>* wird durch den Namen des SQL Servers ersetzt, *<catname>* entsprechend durch den Namen der verwendeten Datenbank, die hier als Katalog bezeichnet wird.

Vorbereitung der Beispiele

Für die folgenden Beispiele wird die Standarddatenbank *FoodMart* von Microsoft genutzt, die im Lieferumfang der OLAP-Services des SQL Servers enthalten ist. In der Datei OPEN.INC.ASP wird dazu eine Funktion definiert, die die entsprechenden Parameter einstellt und als Zeichenkette zurückgibt:

```
function open_olap()
  dim strProvider, strDataSrc, strCatalog, strUser,
      strPassword, strConnection
  strProvider = "MSOLAP"  ' Provider
  strDataSrc  = "WWW"     ' Name des Servers oder IP-Nummer
  strCatalog  = "FoodMart"' Name der Datenbank mit Musterdaten
  strUser     = "sa"      ' Nutzername des SQL Server
  strPassword = ""        ' Kennwort
  ' Ende anpassbarer Bereich
  strConnection = "Provider=" & strProvider
                & "; Data Source= " & strDataSrc
                & "; Catalog=" & strCatalog
                & "; User Id=" & strUser
                & "; Password=" & strPassword
  open_olap = strConnection
end function
```

Listing 7.1: Funktion `open_olap()` in open.inc.asp

Die Nutzung erfolgt, indem die Eigenschaft `ActiveConnection` mit dem Wert gesetzt wird:

```
objCat.ActiveConnection = open_olap()
```

7.6.3 Axis

Ein `Axis`-Objekt können Sie mit folgender Anweisung erzeugen:

```
set objAxis = Server.CreateObject("ADOMD.Axis")
```

Methoden

Dieses Objekt kennt keine Methoden.

Eigenschaften

DimensionCount

DimensionCount

Gibt die Anzahl der Dimensionen zurück. Die Eigenschaft ist nur lesbar.

```
long lngDim = objAxis.DimensionCount
```

Name

Name

Gibt den Namen des Objekts zurück.

```
string strName = objAxis.Name
```

Wenn ein Objekt numerisch referenziert werden kann, gibt die folgende Anweisung den Namen zurück:

```
objCube.CubeDefs(0).Name' ergibt "MyCube"
```

In diesem Fall können Sie auch den Namen zur Referenzierung nutzen:

```
objCube.CubeDefs("myCube")
```

7.6.4 Catalog

Ein neues `Catalog`-Objekt legen Sie mit der folgenden Anweisung an:

```
set objCatalog = Server.CreateObject("ADOMD.Catalog")
```

`Catalog` ist das höchste Objekt in der Hierarchie.

Das folgende Beispiel zeigt, wie Sie die Cubes eines Katalogs abfragen und anzeigen:

```
<%
```

```
dim objCat, objCub
set objCat = Server.CreateObject("ADOMD.Catalog")
objCat.ActiveConnection = open_olap()
for each objCub in objCat.CubeDefs
   echo objCub.Name & "<br>"
next
%>
```

Listing 7.2: ADOMD.Catalog.asp: Ausgabe aller Cubes eines Katalogs

Hier wird ein `Catalog`-Objekt angelegt, mit der Datenbank *FoodMart* verbunden und mit Hilfe der Kollektion `CubeDefs` werden die Namen der definierten Cubes ausgegeben.

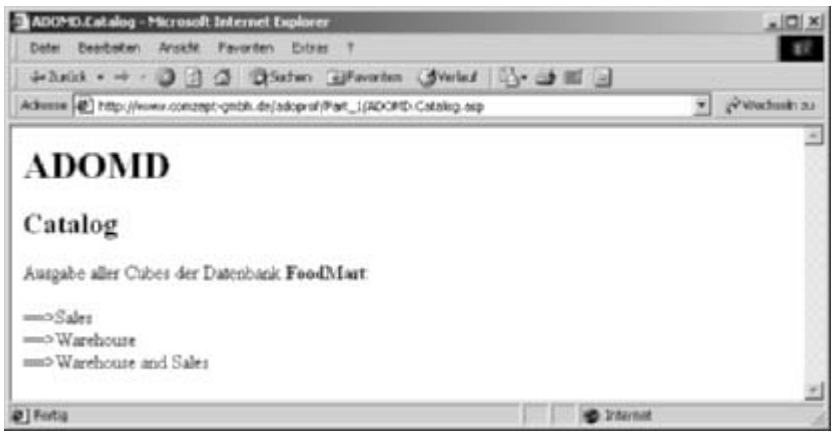


Abbildung 7.12: Cubes des Katalogs FoodMart (Listing 7.2)

Methoden

Dieses Objekt kennt keine Methoden.

Eigenschaften

ActiveConnection

ActiveConnection

Diese Eigenschaft bestimmt die Verbindung zur Datenbank. Übergeben lässt sich eine Verbindungszeichenfolge oder ein ADO-`Connection`-Objekt.

```
objCatalog.ActiveConnection = string strConn
objCatalog.ActiveConnection = object objConn
```

Wenn diese Eigenschaft auf `Nothing` gesetzt wird, wird das Objekt gelöscht. Das betrifft auch alle abgeleiteten Objekte.

Bei bereits stehender Verbindung zu einem `CellSet` darf die Eigenschaft nicht mehr geändert werden.

Verbindungen können auch zu einer CUB-Datei hergestellt werden. Ein Muster für die Verbindungszeichenfolge sieht dann folgendermaßen aus:

```
"Provider=msolap; Location=C:\mycube.cub"
```

Eine normale Verbindungszeichenfolge hat folgende Struktur:

```
"Provider=msolap;
 Data Source=<Server>;
 Catalog=<Katalog>"
```

Ersetzen Sie *<Server>* durch den Namen des OLAP-Servers und *<Katalog>* durch den Namen der Datenbank.

Name

Name

Gibt den Namen des Objekts zurück.

```
string strName = objAxis.Name
```

7.6.5 Cell

`Cell` wird mit der Methode `Item` des `CellSet`-Objekts erstellt:

```
set objCell = objCellSet.Item(intPos)
```

Dabei bezeichnet *intPos* die Position der Zelle im Zellensatz. `Cell` kann auch implizit aus `CellSet` heraus angesprochen werden, wenn eine Zelle eindeutig referenziert wird:

```
set objCellSet = Server.CreateObject("ADOMD.CellSet")
  ' ... diverse Aktionen mit Zellen
Response.Write objCellSet(zeile, spalte).<property>
```

Dabei ist *<property>* eine der nachfolgend beschriebenen Eigenschaften des Objekts `Cell`.

Methoden

Dieses Objekt kennt keine Methoden.

Eigenschaften

Neben den standardmäßig verfügbaren Eigenschaften besteht die Möglichkeit, dass der Provider weitere Eigenschaften zur Verfügung stellt. Diese werden hier nicht aufgeführt.

FormattedValue

Gibt den formatierten Wert einer Zelle zurück.

FormattedValue

```
string strVal = objCell.FormattedValue
```

Der formatierte Wert ergibt sich aus dem Datentyp und dem internen Wert. Wenn der Datentyp `Currency` ist und der Wert 99,90, dann wird die Zeichenkette »DM 99,90« zurückgegeben.

Ordinal

Diese Eigenschaft ergibt die fortlaufend gezählte Position der Zelle im Zellsatz. Da Zellsätze mehrdimensional sein können, lässt sich der Wert selten direkt ermitteln.

Ordinal

```
long lngPos = objCell.Ordinal
```

Die Anzahl der Zellen wird nach der folgenden Formel berechnet:

$$\sum_{i=0}^{p-1} S_i E_i \text{ wobei } E0 = 1 \text{ und } \mathrm{E_i} = \prod_{k=0}^{i-1} U_k$$

U_k bezeichnet die Anzahl der Elemente der Achse k. p ist die Anzahl der Achsen (entspricht den Dimensionen eines Arrays).

Value

Gibt den aktuellen Wert der Zelle zurück. Dieser Wert kann nur gelesen werden.

Value

```
variant varVal = objCell.Value
```

7.6.6 Cellset

Ein `CellSet` enthält meist das Ergebnis einer Abfrage. Der Zugriff auf die Zellen erfolgt über Koordinaten. Das `CellSet`-Objekt erzeugen Sie folgendermaßen:

```
set objCellSet = Server.CreateObject("ADOMD.CellSet")
```

Methoden

Close

Schließt ein `CellSet`-Objekt. Alle verbundenen Objekte gehen verloren.

Close

```
objCellSet.Close
```

Open

Open

Öffnet ein `CellSet`-Objekt. Das Objekt kann leer (ohne Parameter) oder mit Parametern geöffnet werden.

```
objCellSet.Open [string Query][, object ActiveConnection]
```

Der Parameter *Query* enthält eine gültige Abfrage, die zu einem mehrdimensionalen Objekt führt. *ActiveConnection* ist entweder eine Verbindungszeichenfolge oder ein ADO-`Connection`-Objekt zu einer OLAP-Datenquelle.

Eigenschaften

ActiveConnection

ActiveConnection

Diese Eigenschaft bestimmt die Verbindung zur Datenbank. Übergeben werden kann eine Verbindungszeichenfolge oder ein ADO-`Connection`-Objekt.

```
objCellSet.ActiveConnection = string strConn
objCellSet.ActiveConnection = object objConn
```

Wenn diese Eigenschaft auf `Nothing` gesetzt wird, wird das Objekt gelöscht. Das betrifft auch alle abgeleiteten Objekte.

Besteht die Verbindung zu einem `CellSet` bereits, darf die Eigenschaft nicht mehr geändert werden.

Verbindungen können auch zu einer CUB-Datei hergestellt werden. Ein Muster für die Verbindungszeichenfolge sieht dann folgendermaßen aus:

```
"Provider=msolap; Location=C:\mycube.cub"
```

Eine normale Verbindungszeichenfolge hat folgende Struktur:

```
"Provider=msolap;
 Data Source=<Server>;
 Initial Catalog=<Katalog>"
```

Ersetzen Sie *<Server>* durch den Namen des OLAP-Servers und *<Katalog>* durch den Namen der Datenbank.

FilterAxis

FilterAxis

Diese Eigenschaft gibt ein `Axis`-Objekt zurück, das als Ergebnis eines Filtervorgangs aus dem `CellSet` entsteht. Die Eigenschaft kann nur gelesen werden.

```
set object objAxis = objCellSet.FilterAxis
```

CellSet enthält auch eine Axis-Kollektion mit Axis-Objekten. Die durch FilterAxis zurückgegebene Achse ist nicht Bestandteil dieser Kollektion.

Item

Item

Gibt eine Zelle des Zellsatzes zurück.

```
set object objCell = objCellSet.Item(variant index)
set object objCell = objCellSet.Item(array array)
```

Die Zelle kann durch ihren numerischen Index *index* ausgewählt werden. Mehrdimensionale Zellsätze können auch mit Hilfe eines Arrays angesprochen werden, das für jede Dimension die Zellennummer auf der Achse oder den Namen der Achse beschreibt:

```
objCell = objCellSet.Item(Array("N1", "N2", "N3"))
```

Diese Eigenschaft ist die Standardeigenschaft und deshalb kann die Angabe entfallen. Dann verwenden Sie folgende Syntax:

```
Response.Write objCell(1, 2)
```

In diesem Fall wird in einem zweidimensionalen Feld die zweite Zelle der ersten Spalte angesprochen.

Source

Source

Quelle der MDX-Abfrage. Diese Zeichenkette spezifiziert die Daten und sollte eine gültige Abfrage in MDX-Syntax sein.

```
objCellSet.Source = string strQuery
string strQuery = objCellSet.Source
```

Die Eigenschaft kann nur so lange gesetzt werden, wie das CellSet-Objekt geschlossen ist.

- Open
- Close

State

State

Ergibt den Status des CellSet-Objekts: Offen oder Geschlossen.

```
integer intState = objCellSet.State
```

intState	Nummerischer Wert	Beschreibung
adStateClosed	0	Geschlossen
adStateOpen	1	Offen

7.6.7 CubeDef

Dieses Objekt erlaubt die Definition eines Cube. Ebenso ist auch die Abfrage der Eigenschaften und Elemente eines Cube möglich, was den häufigeren Fall darstellt, wenn die Definition in der Entwicklungsumgebung des SQL Servers erfolgt.

Methoden

Dieses Objekt kennt keine Methoden.

Eigenschaften

Die folgende Aufstellung enthält nur die sicher verfügbaren Eigenschaften. Weitere Eigenschaften kann der Provider zur Verfügung stellen. Zu diesen können die folgenden gehören:

- CatalogName (»CATALOG_NAME«). Name des Katalogs
- CreatedOn (»CREATED_ON«). Zeitpunkt der Erzeugung des Cube
- CubeGUID (»CUBE_GUID«). Cube GUID
- CubeName (»CUBE_NAME«). Name.
- CubeType (»CATALOG_NAME«). Typ.
- DataUpdatedBy (»DATA_UPDATED_BY«). Datum der letzten Änderung
- LastSchemaUpdate (»LAST_SCHEMA_UPDATE«). Letzte Änderung des Schemas
- SchemaName (»SCHEMA_NAME«). Name des Schemas
- SchemaUpdatedBy (»SCHEMA_UPDATED_BY«). User-ID des Users, der das Schema geändert hat.

Die Eigenschaften können auch über die Kollektion Properties gelesen werden. Dazu verwenden Sie die Attribute-Namen in Klammern. Das folgende Beispiel zeigt, wie Sie auf die Kollektion zugreifen können, ohne die Namen anzugeben:

```
<%
dim objCat, objCub, varProp
set objCat = Server.CreateObject("ADOMD.Catalog")
objCat.ActiveConnection = open_olap()
for each objCub in objCat.CubeDefs
```

```
   echo "Ausgabe der Eigenschaften f&uuml;r <b>"
         & objCub.Name & "</b>:<br>"
   echo "<ul>"
   for each varProp in objCub.Properties
      echo "<li>" & varProp.Name & " = " & varProp
   next
  echo "</ul>"
next
%>
```

Listing 7.3: ADOMD.CubeDef.Properties.asp: Ausgabe aller Eigenschaften

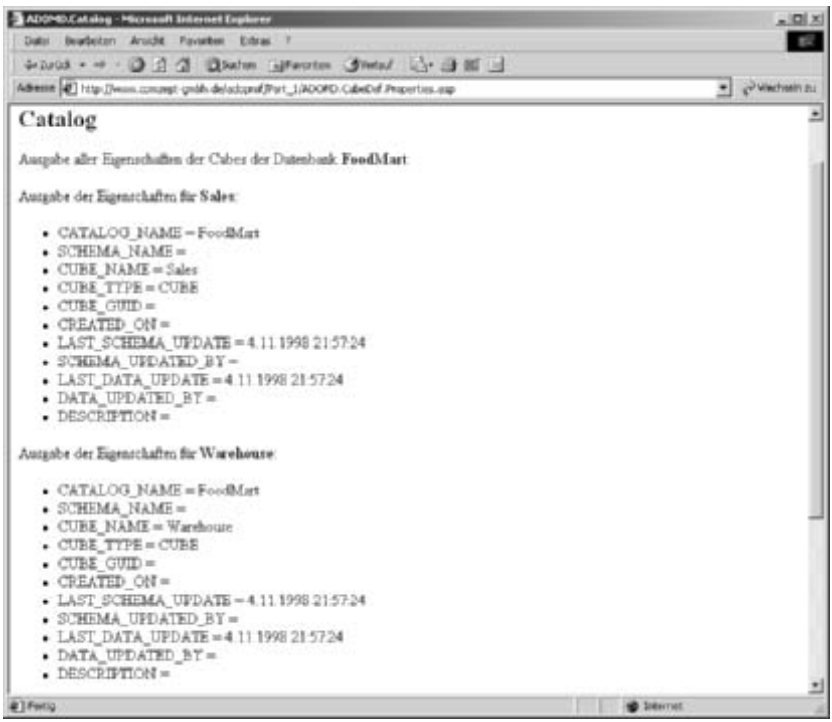


Abbildung 7.13: Ausgabe der Eigenschaften von Cubes

Description

Eine Beschreibung des Objekts. Diese Eigenschaft kann nur gelesen werden.

Description

```
string strDesc = objCube.Description
```

Name

Gibt den Namen des Objekts zurück.

Name

```
string strName = objAxis.Name
```

7.6.8 Dimension

Eine Dimension enthält eine Hierarchie von Mitgliedern. Sie wird mit der Methode `Dimensions` aus dem Objekt `CubeDef` abgeleitet:

```
objDimension = objCubeDef.Dimensions
```

Das folgende Beispiel zeigt alle Dimensionen des Cube *Sales* an:

```
<%
dim objCat, varDim
set objCat = Server.CreateObject("ADOMD.Catalog")
objCat.ActiveConnection = open_olap()
for each varDim in objCat.CubeDefs("Sales").Dimensions
   echo "==>" & varDim.Name & "<br>"
next
%>
```

Listing 7.4: ADOMD.Dimension.asp: Zugriff auf die Namen aller Dimensionen eines Cube

Abbildung 7.14: Anzeige aus Listing 7.4

Methoden

Dieses Objekt kennt keine Methoden.

Eigenschaften

Description

Description

Eine Beschreibung des Objekts. Diese Eigenschaft kann nur gelesen werden.

```
string strDesc = objDimension.Description
```

Name

Name

Gibt den Namen des Objekts zurück.

```
string strName = objDimension.Name
```

UniqueName

UniqueName

Gibt einen eindeutigen Namen für das Objekt zurück. Diese Eigenschaft kann nur gelesen werden.

```
string strUnique = objDimension.UniqueName
```

7.6.9 Hierarchy

Eine Hierarchie besteht aus Ebenen (`Levels`-Kollektionen) und Mitgliedern (`Members`-Kollektionen). Ein `Hierarchy`-Objekt ist Bestandteil der `Hierarchies`-Kollektion, die von einem `Dimension`-Objekt abgeleitet wird.

```
objHierarchy = objDimension.Hierarchies(0)
```

Auch zu Hierarchy existiert eine Kollektion. Allerdings kann der OLAP-Server des SQL Servers nur eine Hierarchie pro Dimension verwalten. Deshalb ist die Unterscheidung nicht sinnvoll und die Angabe eines Namens in der Administration nicht möglich. Die oben aufgeführte Form des Abrufs mit dem Index 0 ist deshalb grundsätzlich die einzige Version.

Properties

Auch zu `Hierarchy` existiert eine `Properties`-Kollektion. Die enthaltenen Werte ermittelt der folgende Code:

```
<%
dim objCat, varProp
set objCat = Server.CreateObject("ADOMD.Catalog")
objCat.ActiveConnection = open_olap()
echo "<ul>"
for each varProp in objCat.CubeDefs("Sales")
                          .Dimensions("Time")
                          .Hierarchies(0)
                          .Properties
    echo "<li>" & varProp.Name & " = " & varProp
```

```
next
echo "</ul>"
%>
```

Listing 7.5: ADOMD.Hierarchy.Properties.asp: Anzeige der Eigenschaften der Hierarchie der Dimension Time. Wenn Sie das Listing abtippen, schreiben Sie die Objektaufrufe hinter for each in eine Zeile ohne Leerzeichen.

Abbildung 7.15: Ausgabe von Listing 7.5

Wie immer bei Verwendung der Kollektion `Properties` können Sie auf das einzelne `Property`-Objekt folgendermaßen zugreifen:

```
object.Properties("CUBE_NAME")
```

Methoden

Dieses Objekt kennt keine Methoden.

Eigenschaften

Description

Description Eine Beschreibung des Objekts. Diese Eigenschaft kann nur gelesen werden.

```
string strDesc = objHierarchy.Description
```

Name

Name Gibt den Namen des Objekts zurück.

```
string strName = objHierarchy.Name
```

UniqueName

Gibt einen eindeutigen Namen für das Objekt zurück. Diese Eigenschaft kann nur gelesen werden.

```
string strUnique = objHierarchy.UniqueName
```

7.6.10 Level

Dieses Objekt ist Bestandteil der `Level`-Kollektion, die von `Hierarchy` abgeleitet wird. Bestandteil des Objekts kann die Kollektion `Members` oder `Properties` sein.

Das Beispiel gibt alle Level einer Dimension aus:

```
<%
dim objCat, varProp
set objCat = Server.CreateObject("ADOMD.Catalog")
objCat.ActiveConnection = open_olap()
echo "<ul>"
for each varProp in objCat.CubeDefs("Sales")
                          .Dimensions("Time")
                          .Hierarchies(0)
                          .Levels
      echo "<li>" & varProp.Name & " (Tiefe <i>"
                  & varProp.Depth & "</i>)"
next
echo "</ul>"
%>
```

Listing 7.6: ADOMD.Level.asp: Ausgabe der Level und ihrer Tiefe für die Dimension Times

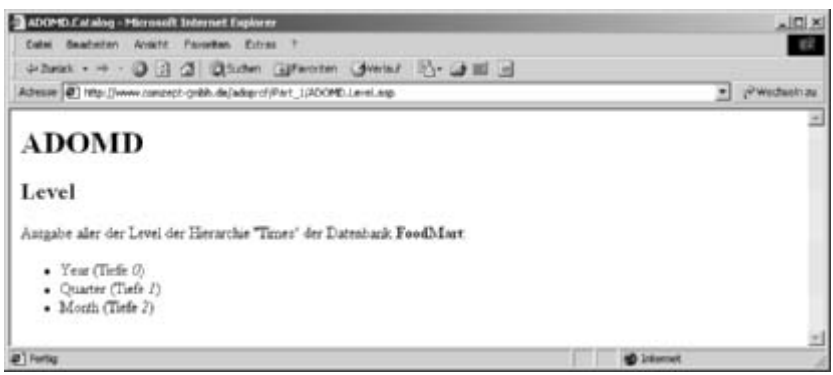


Abbildung 7.16: Ausgabe von Listing 7.6

Methoden

Dieses Objekt kennt keine Methoden.

Eigenschaften

Caption

Caption

Enthält die Überschrift der Liste von Mitgliedern der Ebene bei der Anzeige. Wenn Sie in ASP programmieren, muss der Wert explizit gelesen werden. Andere Programmierumgebungen können darauf möglicherweise implizit zugreifen.

```
string strCaption = objLevel.Caption
```

```
string strCaption = objLevel.Caption
```

Depth

Depth

Gibt die Tiefe der aktuellen Ebene zur Wurzel der Hierarchie an. Die Eigenschaft kann nur gelesen werden.

```
integer intLevel = objLevel.Depth
```

Description

Description

Eine Beschreibung des Objekts. Diese Eigenschaft kann nur gelesen werden.

```
string strDesc = objLevel.Description
```

Name

Name

Gibt den Namen des Objekts zurück.

```
string strName = objLevel.Name
```

UniqueName

UniqueName

Gibt einen eindeutigen Namen für das Objekt zurück. Diese Eigenschaft kann nur gelesen werden.

```
string strUnique = objLevel.UniqueName
```

7.6.11 Member

Mitglieder können an Ebenen (`Levels`-Kollektionen) oder Positionen auf Achsen gebunden sein. Entsprechend erfolgt die Ableitung aus `Levels` oder `Positions`:

```
colMembers = objLevel.Members
objMembers = colMembers.Item(0)
```

Das folgende Beispiel zeigt den Zugriff auf einige Mitglieder und deren Zuordnung:

```
<%
dim objCat, varProp, varMem
set objCat = Server.CreateObject("ADOMD.Catalog")
objCat.ActiveConnection = open_olap()
echo "<ul>"
on error resume next
for each varProp in objCat.CubeDefs("Sales")
                          .Dimensions("Time")
                          .Hierarchies(0)
                          .Levels
   echo "<li>" & varProp.Name
   echo "<ul>"
   for each varMem in varProp.Members
      echo "<li>" & varMem.Caption
      echo " (Sohn von <i>" & varMem.Parent.Caption & "</i>)"
      if Err.Number <> 0 then
         echo " (Kein &uuml;bergeordnetes Objekt)"
         Err.Clear
      end if
   next
   echo "</ul>"
next
echo "</ul>"
%>
```

Listing 7.7: ADOMD.Members.asp: Ausgabe einiger Mitglieder der Dimension Time

Beachten Sie die Ausgabe der Nachricht, dass kein übergeordnetes Objekt existiert. Dies erfolgt hier durch Abfangen der Fehlernummer. Korrekter wäre eine Auswertung der Fehlernummer 424. Der Text lautet »Objekt erforderlich«.

Fehler 424

Methoden

Dieses Objekt kennt keine Methoden.

Abbildung 7.17: Ausgabe von Listing 7.7

Eigenschaften

Caption

Caption

Enthält die Überschrift der Liste von Mitgliedern der Ebene bei der Anzeige. Wenn Sie in ASP programmieren, muss der Wert explizit gelesen werden. Andere Programmierumgebungen können darauf möglicherweise implizit zugreifen.

```
string strCaption = objLevel.Caption
```

```
string strCaption = objLevel.Caption
```

ChildCount

Diese Eigenschaft ermittelt die Anzahl Mitglieder, die in der Hierarchie dem aktuellen Mitglied untergeordnet sind.

```
integer intChildCount = objMember.ChildCount
```

Das untergeordnete Mitglied selbst lässt sich mit `Children` ermitteln. Die Eigenschaft kann nur gelesen werden. Wenn `Member` von `Position` abgeleitet

wurde, ist der maximale Wert 65 535. Die Applikation sollte dies berücksichtigen, und den Wert als größer oder gleich 65 535 interpretieren. Für eine exakte Ermittlung kann auch die Eigenschaft `Count` der `Members`-Kollektion verwendet werden, die jedoch bei sehr großen Objekten langsamer ist.

- `Children`

Children

Children

Diese Eigenschaft gibt eine Kollektion von Member-Objekten zurück, die dem aktuellen Mitglied untergeordnet sind. Die Eigenschaft kann nur gelesen werden.

```
collection colMembers = objMember.Children
```

`Children` kann nur für Mitglieder verwendet werden, die von `Level` abstammen. Für `Positions`-Mitglieder gilt diese Eigenschaft nicht.

Eine Erweiterung des Beispiels aus Listing 7.7 zeigt der folgende Code:

```
if varMem.childCount > 0 then
   dim varChild
   echo "<br><u>Dieses Mitglied hat weitere
         Mitglieder:</u><br>"
   for each varChild in varMem.Children
      echo varChild.Caption & "<br>"
   next
end if
```

Listing 7.8: ADOMD.Members.Children.asp: Ermittlung der untergeordneten Mitglieder

- `ChildCount`

Description

Description

Eine Beschreibung des Objekts. Diese Eigenschaft kann nur gelesen werden.

```
string strDesc = objMember.Description
```

DrilledDown

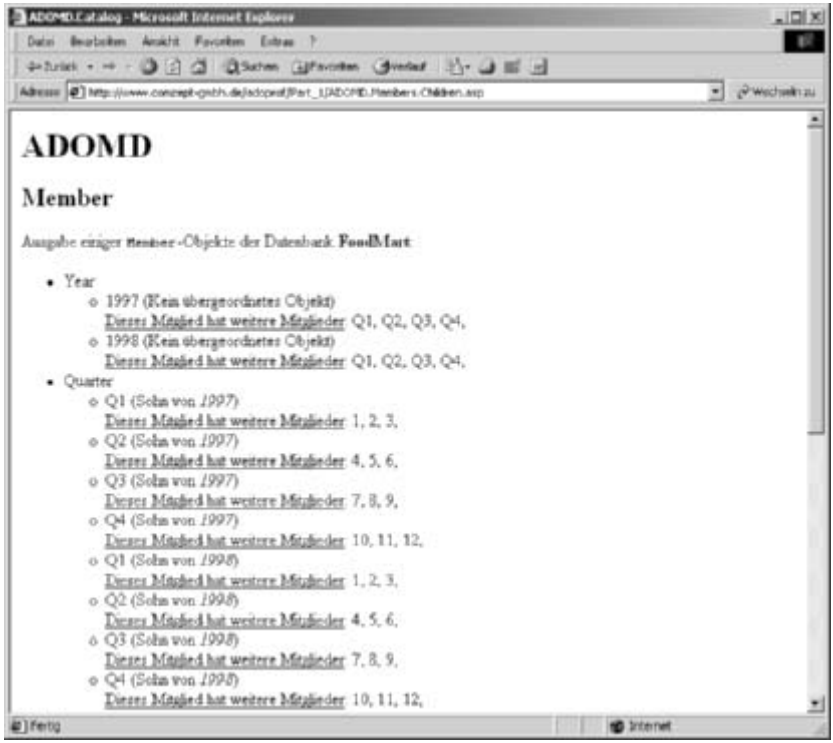
DrilledDown

Diese Boolesche Eigenschaft gibt `FALSE` zurück, wenn das Mitglied in der Hierarchie untergeordnete Objekte hat, andernfalls `TRUE`. Die Eigenschaft kann nur gelesen werden.

```
boolean blnHasChildren = objMember.DrilledDown
```

Mit dieser Eigenschaft lässt sich beim Durchlaufen der Hierarchie das Ende erkennen.

Abbildung 7.18: Ausgabe des kompletten Skripts aus Listing 7.8



LevelDepth

LevelDepth

Gibt die Tiefe der aktuellen Ebene zur Wurzel der Hierarchie an. Die Eigenschaft kann nur gelesen werden.

```
integer intLevel = objMember.LevelDepth
```

LevelName

LevelName

Gibt den Namen des Level-Objekts zurück, zu dem das Mitglied gehört.

```
string strName = objMember.LevelName
```

Name

Name

Gibt den Namen des Member-Objekts zurück.

```
string strName = objMember.Name
```

Parent

Diese Eigenschaft gibt das übergeordnete Objekt des Mitglieds zurück.

Parent

```
object objMemberParent = objMember.Parent
```

`Parent` kann nur für Mitglieder angewendet werden, die von einer `Levels`-Kollektion abstammen. Die Eigenschaft ist nur lesbar.

ParentSameAsPrev

Diese Eigenschaft ist `TRUE`, wenn dieses Mitglied und das vorhergehende dasselbe übergeordnete Element haben.

ParentSameAsPrev

```
boolean blnSame = objMember.ParentSameAsPrev
```

Type

Diese Eigenschaft ermittelt den aktuellen Typ des Mitglieds.

Type

```
integer intType = objMember.Type
```

intType kann der nachfolgenden Tabelle entnommen werden:

intType	Numerischer Wert	Beschreibung
adMemberAll	4	Dieses Objekt repräsentiert alle Mtiglieder der Ebene.
adMemberFormula	3	Zeigt an, dass das Mitglied auf Basis einer Formel berechnet wurde.
adMemberMeasure	2	Mitglied gehört zu einer Maß-Dimension und repräsentiert ein quantitatives Attribut.
adMemberRegular	1	Standardwert. Mitglied repräsentiert eine Instanz.
adMemberUnknow	0	Unbekannt

UniqueName

Gibt einen eindeutigen Namen für das Objekt zurück. Diese Eigenschaft kann nur gelesen werden.

UniqueName

```
string strUnique = objMember.UniqueName
```

7.6.12 Position

Positionen sind Objekte auf Achsen oder in Zellen. Im Gegensatz zu den Mitgliedern von Dimensionen können diese keine Hierarchien bilden. Positionen können aber selbst Mitglieder enthalten.

Methoden

Dieses Objekt kennt keine Methoden.

Eigenschaften

Ordinal

Ordinal

Diese Eigenschaft ergibt die fortlaufend gezählte Position im Zellsatz oder auf der Achse.

```
long lngPos = objPosition.Ordinal
```

Teil II
ADO.NET

8 Grundlagen ADO.NET

ADO.NET ist die evolutionäre Weiterentwicklung von ADO. Neu ist die Übertragung von Daten per XML und damit ein einfacher Datenaustausch zwischen den verschiedenen mit Visual Studio 7 erstellten Applikationen, auch wenn diese nicht über eine eigene ADO-Schnittstelle verfügen.

8.1 Einführung

Die Arbeit mit ADO.NET unterscheidet sich zwar in weiten Teilen von dem bereits zuvor beschriebenen ADO, entsprechende Kenntnisse im »klassischen« Umgang mit Datenbanken sind jedoch unbedingt notwendig. Vor allem offenbahrt sich der grundlegende Unterschied zur bisherigen Arbeitsweise nur, wenn man den Vergleich heranziehen kann. Sicher ist auch, dass ADO.NET nicht in allen Fällen zum Einsatz kommen wird. Vor allem bei sehr kleinen Projekten sind die Vorteile nicht erkennbar.

8.1.1 Programmierbarkeit

Programmierung »gegen« die Quelle

Der größte Unterschied bei der Programmierung mit ADO.NET ist die Art und Weise, wie Daten abgefragt werden. So programmieren Sie bei ADO direkt »gegen« die Datenquelle, also Datenbanken und Tabellen bzw. in letzter Konsequenz Spalten und Felder. Typisch sind solche Konstruktionen, wie in dem folgenden Pseudocode gezeigt:

```
IF NeuUmsatz > Table("Kunden").Column("Umsatz")
```

Hier wird die eigentliche Datenquelle (Tabelle *Kunden*) abgefragt und die Spalte *Umsatz* ausgewählt. Wenn man die Schreibweise einer objektorientierten Programmiersprache wählt, wird die Abfrage deutlich einfacher:

```
IF NeuUmsatz > Kunden.Umsatz
```

Dies ist nicht nur einfacher zu lesen, sondern auch einfacher zu schreiben. In Visual Studio 7 kann eine solche Notation auch von der automatischen Vervollständigung der Schlüsselwörter profitieren, wie es für alle registrierten Objekte möglich ist. Die Intellisense genannte Technologie verhindert durch das Anbieten einer Auswahl von Schlüsselwörtern Tippfehler. Sie können auch jederzeit feststellen, ob die Datenobjekte bereits die korrekten Verbindungen zur Datenquelle repräsentieren.

Automatische Vervollständigung

Die automatische Vervollständigung im Visual Studio Editor ist nun auch bei Datenobjekten möglich – zumindest theoretisch. Die PreView-Version hatte hiermit noch einige Probleme.

Bei der Ausführung ergibt sich aufgrund der direkten Angabe außerdem ein Geschwindigkeitsvorteil. Die klassische Schreibweise erzwingt immer das Durchsuchen eines Datensatzobjekts nach den praktisch erst zur Laufzeit analysierbaren Parametern. Diese Parameter können in der Praxis variabel sein. Mit der Objektschreibweise wird der Name aber schon zur Entwicklungszeit festgelegt. Nachteilig mag eine geringfügige Einschränkung der Flexibilität sein, was eine saubere, objektorientierte Programmierung aber immer erzwingt.

8.1.2 Der Datensatz

Datensatz als zentraler Punkt

Bei der Arbeit mit ADO.NET dreht sich alles um Datensätze. Das war bei ADO auch ein zentraler Bestandteil, allerdings nicht mit so absoluter Konsequenz. Ein Datensatz ist eine Kopie der Daten aus einer Datenquelle. Für den Programmierer ist es nicht nur eine andere Form der Darstellung der Daten, sondern eine vollkommene Abstraktionsschicht. Die Notwendigkeit offenbahrt sich, wenn man die möglichen Datenquellen betrachtet. Neben relationalen Datenbanken kommen auch XML-Dateien, Stream-Daten (Text) oder per SOAP initiierte Abfragen an andere Server im Netz in Frage. Alle Varianten zu berücksichtigen, würde den Programmieraufwand unzulässig in die Höhe treiben. Mit der Datensatz-Sicht vereinfacht sich die Programmierung signifikant.

Eigenschaften des Datensatzes

Der Datensatz hat keine direkte Verbindung mit der Datenquelle mehr. Er existiert normalerweise im Hauptspeicher des Computers, auf dem eine ADO.NET-Applikation ausgeführt wird. Diese Form der Datenverarbeitung unterstützt auch implizit Mehrschicht-Applikationen. So existieren die Daten zur Laufzeit praktisch in der mittleren Schicht einer Drei-Schicht-Anwendung – wo sie auch hingehören. Die untere Schicht bildet der physikalische Datenspeicher mit seinen Werkzeugen, die obere Schicht die Benutzerschnittstelle. Diese Trennung erlaubt den Austausch der Schichten auf sehr einfache Weise. So kann man die Datenquelle zwischen RDBMS und XML wechseln und den Benutzer mit einer Windows-Applikation ebenso leicht wie mit einem Web-Interface ansprechen.

Um den Datenaustausch zugleich auch zwischen Programmen zu ermöglichen, die im Netzwerk auf verschiedenen Maschinen laufen und deren Applikationen in unterschiedlichen Sprachen erstellt wurden, wird als Austauschsprache XML verwendet. Erfolgt also eine Abfrage der Daten aus der mittleren Schicht, dann wird ADO.NET das Resultat der Abfrage in XML darstellen und dann versenden.

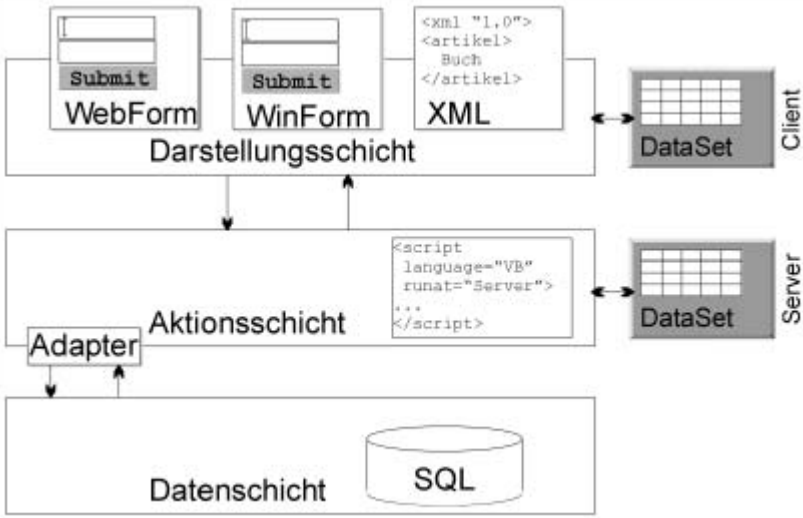


Abbildung 8.1: ADO.NET-Architektur unterstützt die Realisierung von Drei-Schicht-Modellen

Die Darstellung der Daten in XML bietet noch viele andere Vorteile, die später erläutert werden. Die Unterstützung in Visual Studio 7 ist konsequent. XML ist keine »Geheimsprache« in der Software, sondern wird offen präsentiert. Für Profis steht auch ein Editor zur Verfügung, mit dem Daten direkt in XML verändert werden können – mit allen typischen Eigenschaften übrigens wie Intellisense und farblicher Hervorhebung der Syntaxtypen.

XML-Darstellung

Typisch bei der Erstellung von Datenbankmodellen ist die grafische Sicht auf die Verknüpfungen von Tabellen. Diese Sicht wird in Visual Studio 7 auch für XML-Daten angeboten. Dieselbe Sicht erlaubt aber auch die Darstellung von Daten einer Oracle- oder SQL Server-Datenbank. Welche Technik dann konkret zur Laufzeit verwendet wird, muss beim Entwurf nicht berücksichtigt werden.

Daten-Entwurfswerkzeuge

Eine weitere Sichtweise erlaubt die schnelle Änderung der Daten während der Entwicklungsphase. Das betrifft sowohl die Datenstruktur als auch Verknüpfungen und Relationen, wie auch die Daten selbst, die eventuell zum Testen der Anwendung herangezogen werden.

Datenvorschau

Die Bindung an die Daten erfolgt also zur Entwurfszeit. So lässt sich bereits hier erkennen, wie sich reale Daten verhalten.

8.2 Eigenschaften von ADO.NET

Die angesprochene Technik der Datenübertragung zwischen Applikationen soll hier näher betrachtet werden. Tatsächlich werden die Daten nämlich nicht zwischen Applikationen, sondern zwischen Komponenten ausgetauscht. Allerdings bestehen viele sehr einfache Applikationen nur aus einer Komponente. Dem Anfänger mag der Unterschied deshalb nicht auffallen. Stellen Sie sich jedoch größere Programme vor, die über eine komplexe Logik verfügen, so wird auch die Bedeutung der Verteilung zwischen Komponenten offensichtlich.

8.2.1 Basisaussagen

Die wichtigste Eigenschaft einer Programmierumgebung für datenbankgestützte Applikationen ist ihr Verhalten für den Datenaustausch und die Manipulation von Daten. Dies ist sowohl zur Laufzeit als auch zur Entwurfszeit interessant – im Interesse einer schnellen Entwicklung.

Komponenten tauschen Daten aus

Der Datenaustausch der Komponenten erfolgt zur Laufzeit. Eine Komponente, die eine Datenbankabfrage ausführt und das Ergebnis verfügbar hat, kann diese Daten anderen Komponenten anbieten. Das Austauschformat ist XML. Das Ziel kann eine WebForm oder WinForm sein, je nachdem, ob es sich um ein Windows-Programm oder eine Webserver-Applikation handelt. Ebenso kommen aber auch andere Komponenten in Frage, welche die Daten weiterverarbeiten. Im Zusammenhang mit Active Server Pages, hier in der neuesten Entwicklung ASP.NET, handelt es sich natürlich immer im WebForms.

Um von vornherein Verständnisprobleme auszuräumen: XML existiert üblicherweise in Form von Dateien. Diese kann man öffnen, anschauen, ändern und löschen. Das ist bei ADO.NET nicht anders. Den einzigen Fall, wo das nicht so präsent ist, stellt das Protokoll SOAP dar. Hier werden die Daten im Körper einer HTTP-Nachricht versendet. Diese kann man sich aber auch gut als Datei vorstellen. SOAP steht für *Simple Object Access Protocol* und liegt derzeit in der Version 1.1 vor. Mit ASP.NET können sogenannte Webservices programmiert werden, die SOAP zum Datenaustausch verwenden. Mehr Informationen zu SOAP finden Sie unter der folgenden Adresse:

- *http://www.w3.org/TR/SOAP/*

Manipulation der Daten

Das Datensatzobjekt

Die Manipulation der Daten erfolgt in einem Programm mit Hilfe des Datensatzobjekts. Die Methoden sind weitestgehend konsistent zu denen, die beim relationalen Modell in ADO verwendet werden. Ein ADO.NET-Datensatz kennt Tabellen, Spalten und Reihen und Verknüpfungen zwischen ihnen. Die Darstellung erfolgt aber immer, auch auf der untersten

Ebene, als Objekt. Wenn eine Komponente Daten empfängt – im XML-Format – materialisiert sie diese Daten als Datensatzobjekt. Dieses Objekt ist also die letztendlich für den Programmierer sichtbare Erscheinungsform.

Da Daten in größeren Umgebungen zugleich von vielen Nutzern geändert werden können, erfolgt die Übertragung in den physikalischen Speicher sehr oft. Dadurch entsteht das Problem der Synchronisation zwischen den Datensatzobjekten und dem Datenspeicher. Wie schon bei ADO kann diese Verbindung gesteuert werden und es liegt in der Verantwortung des Programmierers, hier von vornherein die Konsistenz der Daten zu beachten. Allerdings müssen Sie sich nicht um die Technik der Übertragung der Daten kümmern. Wenn das Objekt »weiß«, dass es Daten übertragen muss, wird es dies auch tun.

Datensatz-Kommando-Objekt

Um die Verbindung mit den Datenquelle effizient zu gestalten, gibt es das Datensatz-Kommando-Objekt in ADO.NET. Dies liegt in zwei Versionen vor:

- *SQLDataSetCommand*-Objekt. Dieses Objekt unterstützt SQL Server 7 und SQL Server 2000.
- *ADODataSetCommand*-Objekt. Dieses Objekt setzt auf einen OLE DB-Provider auf. Diese Provider wurden bereits in den Kapiteln zu ADO ausführlich betrachtet.

8.2.2 ADO.NET im Vergleich zu ADO

ADO.NET wird sich sicher mittelfristig immer dem Vergleich mit ADO stellen müssen. Kaum ein Projekt dürfte von ADO nach ADO.NET umgestellt werden. Wenn Sie bei der laufenden Arbeit die Entscheidung treffen müssen, hilft Ihnen dieser Abschnitt.

Repräsentation der Daten

In ADO werden Daten in einem `RecordSet` dargestellt, in ADO.NET in einem `DataSet`. Das resultiert in grundlegenden Unterschieden:

Anzahl der Tabellen im lokalen Datenobjekt

- *Anzahl der Tabellen.* In ADO enthält ein `RecordSet`-Objekt immer nur Informationen einer einzigen Tabelle. Haben Sie verbundene Daten, müssen Sie die SQL-Abfrage um einen oder mehrere `JOIN` erweitern, um das Ergebnis in einer Tabelle darstellen zu können. Komplexe `JOIN`-Abfragen bereiten oft auch Profis Probleme.

  In ADO.NET repräsentiert ein `DataSet` eine oder auch mehrere Tabellen. Da dieses Objekt Tabellen als Subobjekte enthalten kann, ist eine Auflösung bei der Abfrage der Datenbank nicht zwingend notwendig. Zwangsläufig führt das `DataSet`-Objekt auch die Verknüpfungen zwischen den Tabellen mit. Diese arbeiten analog zu den `FOREIGN KEYS` der Datenbanken. Es stellt kein Problem dar, wenn eine Datenquelle solche Verknüpfungen nicht kennt (was bei XML der Fall ist) – diese Logik wird

in ADO.NET abgebildet. Verknüpfungen der werden durch ein `DataRelation`-Objekt dargestellt – auch diese Eigenschaften sind nun im direkten Zugriff des ADO-Programmierers.

Visualisierung der Daten

- *Datenvisualisierung*. Die Technik der Sichtbarmachung der Daten und auch der Änderung, des Lesens und Schreibens, ist in ADO.NET anders als bei ADO gelöst. In ADO läuft ein Programm sequenziell durch das RecordSet-Objekt. Das ist typisch und mag auf den ersten Blick als brauchbare Lösung erscheinen. Wollen Sie aber auf einen bestimmten Datensatz gezielt zugreifen, oder auch auf eine komplexe Auswahl, müssen Sie direkt die Datenbank ansprechen – hier ist `WHERE` das Stichwort. Stehen dabei Tabellen miteinander in einer Beziehung, wird es kompliziert.

  In ADO.NET können Sie ein Navigationsparadigma entwerfen und eine Tabelle abfragen, ohne die Beziehung zu berücksichtigen. Die verknüpften Daten werden dann automatisch mitgeführt, sodass die Darstellung der Daten sehr flexibel erfolgt.

Verbindung mit den Quelldaten

Grundsätzlich unterbricht ADO.NET immer die Verbindung zur Datenquelle. In ADO hatten Sie die Wahl zwischen einer stehenden Verbindung oder der Unterbrechung. Typischerweise wird die Verbindung nicht unterbrochen. Wenn Sie auf ADO.NET umsteigen, werden Sie damit vielleicht am meisten Probleme haben. Es gibt aber handfeste Vorteile der Methode in ADO.NET, die hier erläutert werden sollen.

In ADO sprechen Sie eine Datenquelle, beispielsweise über den OLE DB-Provider, direkt an. In ADO.NET dagegen kommuniziert Ihre Applikation nur mit dem `DataSetCommand`-Objekt. Es besteht also auch für die Aufnahme und Kontrolle eine Abstraktionsschicht, die es vorher nicht gab. Über die Art und Weise, wie die Verbindung gestaltet wird, müssen Sie sich also überhaupt nicht kümmern. Das `DataSetCommand`-Objekt führt die Aufrufe des OLE DB-Providers aus. Dies ist relativ effizient, da meist das API des Datenbankmanagement-Systems direkt angesprochen wird. Sie können nun das Verhalten des `DataSetCommand`-Objekts unmittelbar kontrollieren. Wie die Verbindung also tatsächlich ausgeführt wird, lässt sich mit einer einzigen Programmzeile steuern. Tatsächlich stehen mehr Optionen der Steuerung zur Verfügung als bei ADO.

Datenaustausch zwischen Schichten

Der Sinn von Mehrschicht-Applikationen muss hier sicher nicht diskutiert werden. Entscheidend sind die Erleichterungen, die ADO.NET mit sich bringt. Das Übertragen von Daten zwischen den Schichten einer Applikation in ADO war immer mit dem Transport des `RecordSet`-Objekts verbunden. Das war nie unproblematisch, denn dieses Objekt ist sehr sprachnah und wenig abstrahierend. Für den Austausch zwischen COM-Objekten wurde das so genannte COM-Marshaling eingesetzt. Diese Technik ist

schwer zu beherrschen und im Fehlerfall kaum zu debuggen. In ADO.NET erledigt diesen Austausch XML. XML-Dateien sind im Notfall sogar mit Notepad einzusehen.

Im Folgenden lesen Sie eine Zusammenfassung der Vorteile:

Mehr Datentypen

- *Datentypen*. COM-Marshaling kennt nur wenige elementare Datentypen. XML kennt keine Beschränkungen; Datentypen werden im Dokument beschrieben und können deshalb beliebig umfangreich ausfallen.

Höhere Leistung

- *Leistung*. Die Übertragung großer Datenmengen ist kritisch, wenn diese über ein Netzwerk ausgetauscht werden. Sowohl ADO als auch ADO.NET optimieren dies. ADO.NET kennt aber noch weitere Vorteile. So ist keine Konvertierung der Daten notwendig, dies beschleunigt die Auslieferung.

Vermeidung von Problemen mit FireWalls

- *Umgehung von Firewalls*. Typischerweise werden Firewalls so konfiguriert, dass sie HTML-Seiten durchlassen und Zugriffe auf anderen Wegen abfangen. COM-Marshaling arbeitet auf Systemebene und solche Zugriffe werden nicht durchgelassen (auf derselben Ebene arbeiten eben auch Hacker). Applikationen, die auf mehrere Server im Internet verteilt sind, sind mit ADO.NET leichter zu programmieren. XML arbeitet auf Dateiebene und wird wie eine HTML-Seite per HTTP übertragen.

8.3 ADO.NET in der Programmierumgebung .Net

Das .Net-Framework ist ein objektorientiertes System. ADO.NET kann über eine Referenz angesprochen werden. Es steht, wie auch ADO, in allen Sprachen zur Verfügung, die von Visual Studio 7 unterstützt werden. Intern ist auch ADO.NET eine DLL.

8.3.1 Der Namensraum System.Data

Der Namensraum

Die Referenz zu ADO.NET heißt `System.Data`. Hierunter können Sie sich zwischen `System.Data.ADO` (dies spricht den OLE DB-Provider an) oder `System.Data.SQL` (dies spricht SQL Server nativ an) entscheiden.

Visual Basic.NET

Durch der Bnennung fast aller neuen Microsoft-Produkte mit dem Suffix .NET wird die Positionierung deutlicher als zuvor. Auch die Programmiersprache Visual Basic ist davon betroffen. Visual Basic.NET wird es im Sprachgebrauch des Marketing nicht geben – es heißt nun Visual Basic.NET. Die interne Versionsnummer ist allerdings die 7. Was immer Sie zu Visual Basic.NET lesen, gilt für Visual Basic.NET.

VB.NET-Code In Visual Basic.NET wird der Namensraum wie folgt angesprochen:

```
Imports System.Data
Imports System.Data.ADO
Imports System.Data.SQL
```

Die Steuerung erfolgt also durch eine bewusste Integration ins Programm und damit unter Kontrolle des Programmierers.

VB.NET und ASP.NET

ASP.NET-Direktive Wenn Sie den Code wie bisher direkt in die ASP.NETSeite schreiben, sieht die Syntax etwas anders aus. Zumindest altgedienten ASP-Programmierern dürfte die folgende Form sympathischer sein:

```
<%@ Import Namespace="System.Data"  %>
<%@ Import Namespace="System.Data.ADO"  %>
<%@ Import Namespace="System.Data.SQL"  %>
```

Beachten Sie die Schreibweise des Befehls `Import`, einmal mit und einmal ohne »s«.

8.3.2 ASP.NET

Wenn Sie ADO.NET verwenden, steht zwangsläufig ASP.NET als Applikationssprache zur Verfügung. Hier empfiehlt sich ein kompletter Umstieg. ASP.NET bedeutet ebenso wie ADO.NET einen signifikanten Fortschritt. Zwar ist es zweifelhaft, ob sich eine Umstellung lohnt – don't touch a running system – aber für neue Projekte kommt ASP nicht mehr ernsthaft in Betracht. Das folgende Kapitel führt deshalb in einer sehr kompakten Form in ASP.NET ein. Die Beispiele zu ADO.NET in Kapitel 10 bauen dann direkt darauf auf. Sie können sich aber ADO.NET auch erst anschauen und dann entscheiden, ob Sie parallel den Einstieg in ASP.NET versuchen.

9 Einführung in ASP.NET

Die vorangegangenen Beispiele zeigten vor allem den Umgang mit den »inneren« Werten von ADO.NET. Die Ausgabe zum Browser ist die Sicht des Nutzers. Auch hier unterstützen spezielle Objekte die Arbeit des Programmierers.

9.1 Einführung in ASP.NET

.NET auch für ASP-Programmierer

Dieser Abschnitt bietet eine kompakte Einführung in ASP.NET, ohne einen Anspruch auf perfekte Struktur erheben zu wollen. Es geht hier darum, schnell die elementaren Techniken kennen zu lernen.

9.1.1 ASP.NET und ADO

Diese Überschrift mag widersinnig erscheinen – es ist aber tatsächlich nicht unbedingt notwendig, mit ASP.NET gleich auch auf ADO.NET zu wechseln. Vielleicht fallen Ihnen die ersten Schritte in ASP.NET leichter, wenn sich nicht alles ändert. ADO.NET ist, wie im letzten Kapitel gezeigt wurde, eine radikale Änderung.

ADO steht weiter zur Verfügung

ADO steht weiter zur Verfügung, lässt sich also auch problemlos verwenden. Aussagen über die Leistung sind hier noch nicht möglich, da das gesamte Buch mit Hilfe der PreView-Version von Visual Studio 7 entstand – praktisch also mit einer Alpha-Version. Wenn Sie die Final Release in den Händen halten, werden Sie im Buchhandel auch eine neue Auflage finden, die entsprechende Aussagen enthält.

Wesentliche Unterschiede

Unterschiede zwischen VB.NET und VBScript

Die eigentlichen und einzigen Unterschiede zwischen einer ASP.NET-Seite im klassischen ASP-Programmierstil und einer ASP-Seite mit ADO betrifft die Programmiersprache. Statt VBScript müssen Sie Visual Basic.NET (VB 7) einsetzen. Es mag Programmteile geben, die völlig unverändert sind – andere werden geringfügig abweichen. Da Sie die umfangreichen und vielseitigen neuen Features in Visual Basic.NET nicht verwenden müssen, können Sie wie mit VBScript arbeiten. Immerhin ist VBScript ein VB-Derivat.

Zwei Unterschiede sind jedoch gleich von Anfang wichtig:

Variablen

- Variablen müssen deklariert werden. Die zuvor mögliche »freiwillige« Deklaration mit `option explicit` ist nun immer notwendig. Sie sollten auch den Datentyp mit angeben. Dazu wird das Schlüsselwort `As` verwendet:

  ```
  Dim myVar As String
  ```

Schreibweisen

- Parameter müssen bei Methoden nun immer in Klammern eingeschlossen werden. In VBScript war dies möglich, aber nicht nötig. Folgender Ausdruck ist in VB7 ungültig:

```
Response.Write "Fehler:"
```

Richtig ist dagegen folgende Schreibweise:

```
Response.Write("Hinweis:")
```

Dies gilt selbstverständlich auch für alle ADO-Objekte, selbst wenn diese aus den alten Bibliotheken stammen – die Syntax wird durch die aufrufende Sprache bestimmt.

9.2 Web Controls

Web Controls

Unter dem Oberbegriff Web Controls[1] versteht man alle Anzeigeelemente, die in Formularen oder in HTML angewendet werden können.

9.2.1 Einteilung der Web Controls

Unterschieden werden zwei Gruppen:

- *HTML Controls*. Damit werden typische HTML-Tags erzeugt, wie Anchor, Table oder Text.
- *Web Controls*. Damit werden Formularelemente erzeugt. Diese Gruppe enthält drei Arten von Elementen:
  - *Web Form Controls*. Damit werden einfache Formularelemente erzeugt.
  - *Validation Controls*. Damit werden Elemente erzeugt, die Eingaben prüfen können. Als Prüfung können Grenzwerte, reguläre Ausdrücke oder Werte anderer Elemente herangezogen werden.
  - *Pagelet Controls.* Damit werden ausgelagerte Formulare bezeichnet, die wiederverwendet werden können. PageLets haben die Dateierweiterung ASPC.

Die Datenbindung, die im letzten Abschnitt besprochen wurde, kann in beiden Varianten zur Übergabe dynamischer Daten an die Elemente verwendet werden.

1 Hier und im Folgenden habe ich Übersetzungen entgegen der üblichen Praxis strikt vermieden, da die deutsche Dokumentation von Microsoft noch nicht vorliegt und die offizielle Notation normalerweise die einzige sinnvolle Basis für eine Übersetzung ist.

HTML- oder WebControls?

Wenn Sie die Möglichkeiten beider Gruppen betrachten, fallen Ihnen sicher viele Übereinstimmungen auf. Beide Gruppen enthalten z. B. Elemente zur Erzeugung von Eingabefeldern. In beiden Fällen wird im Browser das HTML-Tag `<input type="text">` erzeugt. Generell hängt es ein wenig von der Ausrichtung des Projekts ab.

HTML Controls

Wenn der Schwerpunkt auf dem HTML-Design liegt und Sie nur ein wenig Code schreiben, um dynamische Elemente einzubauen, sollten Sie die Struktur auch HTML-lastig belassen. Möglicherweise wird der Entwurf von einem Designer gemacht, der nicht allzuviel Neues lernen möchte, was ihm nichts wirklich Neues bringt.

Web Controls

Ist aber HTML nur Mittel zum Zweck, um eine komplexe Applikation herum, dann werden Sie WebControls als angenehmer empfinden. Hier erstellen Sie fast nur VB-Code und erzeugen damit ein wenig HTML. So haben Sie kompakten, konsistenten Code, der auch bei großem Funktionsumfang leicht zu warten ist. Hier werden beide Varianten vorgestellt, damit Sie die Möglichkeiten selbst vergleichen können.

9.2.2 HTML Controls

Aufgabe der HTML Controls

HTML Controls erzeugen HTML-Tags. Da stellt sich natürlich die Frage, warum HTML-Tags nicht direkt geschrieben werden sollen. HTML Controls sind immer dann sinnvoll, wenn eine Datenbindung erfolgt. Das muss nicht nur der Text aus einer Datenbank sein, sondern kann auch zur Personalisierung von Websites, zur Unterstützung mehrerer Sprachen oder zur einfacheren Integration eines Redaktionssystems dienen.

Übersicht

Die folgende Tabelle zeigt eine Übersicht der verfügbaren Elemente. Einige wichtige Elemente werden anschließend in Form von Beispielen vorgestellt.

Tabelle 9.1: HTML Controls

Name	HTML-Tag	Beschreibung
`HTMLForm`	`<form>`	Definiert ein Formular
`HTMLInputText`	`<input type=text>` `<input type=password>`	Texteingabefeld
`HTMLTextArea`	`<textarea>`	Texteingabefeld, mehrzeilig
`HTMLAnchor`	`<a href>`	Link, der das Formular absendet oder einfacher Hyperlink (parameterabhängig)
`HTMLButton`	`<button>`	Schaltfläche, HTML 4.0
`HTMLInputButton`	`<input type=button>`	Schaltfläche, HTML 3.2
`HTMLSelect`	`<select><option>...`	Liste von Elementen

Name	HTML-Tag	Beschreibung
HTMLImage	<img>	Bild
HTMLInputHidden	<input type=hidden>	Verstecktes Feld
HTMLInputCheckBox	<input type=checkbox>	Kontrollkästchen
HTMLInputRadioButton	<input type=radio>	Optionsfeld
HTMLTable	<table>	Tabelle
HTMLTableRow	<tr>	Tabellenreihe
HTMLTableCell	<td>	Tabellenzelle
HTMLInputFile	<input type=file>	Dateiupload-Element
HTMLGenericControl		Eigenes Element

HTMLForm

HTMLForm

Formulare können nur innerhalb einer HTML-Form existieren. Das Objektmodell, das den Formularen zugrunde liegt, ermöglicht eine einfache Übertragung der gesendeten Daten in die Applikation. Vergessen Sie solche Methoden wie `Request.Form` – die Daten stehen Ihnen in objektorientierter Form direkt zur Verfügung. Damit das funktioniert, muss das <form>-Tag in einer speziellen Weise ausgeführt werden. Und damit Sie sich darum nicht kümmern müssen, gibt es `HTMLForm`. Die Anwendung ist sehr einfach:

```
<form runat="server" id="formname">
```

Alle anderen Attribute werden bei Bedarf hinzugefügt. Das erledigen Sie natürlich – wie bisher –, indem diese in das Tag geschrieben werden. Besser und richtiger ist die Nutzung entsprechender Eigenschaften. Allein durch das Aufschreiben des Tags steht Ihnen nämlich nun ein Objekt *formname* zur Verfügung. Den Namen legen Sie mit dem Attribut `id` fest. Wollen Sie die Methode ändern, genügt folgende Zeile:

```
formname.Method = "Post"
```

Keine Frage, dass Sie hier alles dynamisch ändern können. Dabei sind niemals Eingriffe in die eigentliche HTML-Seite notwendig. Im Vorgriff auf ein paar ASP.NET-Techniken am Ende des Kapitels sei verraten, dass Sie Code und HTML auch physikalisch trennen können – der Code steht dann in einer eigenen Datei. Ihre Designer und Ihren HTML-Editor wird es freuen, reines HTML ohne komplizierten Code und lästige ASP-Tags zu sehen.

HTMLInputText

HTMLInputText

Dieses Element erzeugt <input type=text> in HTML. Die einfachste Form kann folgendermaßen aussehen:

```
<input type="text" runat="server" id="textcontrol">
```

Für Kennwörter schreiben Sie dagegen:

```
<input type="password" runat="server" id="textcontrol">
```

Alle anderen Modifikationen sind wieder über die entsprechende Klasse möglich. Die folgende Tabelle zeigt die wichtigsten Eigenschaften:

Eigenschaft	Beschreibung
Attributes	Enthält alle Attribute/Wert-Paare
DataBindings	Aktuelle Verknüpfung der Datenquelle
Disabled	Setzt das Attribut `disabled`, um das Feld zu sperren
MaxLength	Maximale Anzahl Zeichen, die eingegeben werden können
Name	`Name`-Attribut
Size	Breite der Box in Zeichen
Style	Style-Klasse
TagName	Name des Tags, hier: `INPUT`
Type	Das Attribut `Type` für das HTML-Tag
Value	Der Wert, der angezeigt werden soll oder eingegeben wurde
Visible	Entscheidet, ob das Element sichtbar ist

Tabelle 9.2: Einige wichtige Eigenschaften des HTMLInputText-Elements

9.2.3 Validation Controls

Validation Controls prüfen erfasste Daten

Die Validation Controls sind Elemente, die neben der Darstellung eines Formular-Elements auch gleich die Prüfung der Eingabe vornehmen. Dieser erste Check der Daten war in ASP relativ aufwändig. Auf dem Weg der Daten in die Datenbank sind solche Prüfungen aber unbedingt notwendig.

Validation Controls können aber noch mehr. Ebenso problematisch wie die Prüfung ist auch die Reaktion darauf. Wenn ein Nutzer Daten falsch eingegeben hat, sollte er gezielt auf diesen Fehler aufmerksam gemacht werden. Es ist aber nicht einfach, die Fehlermeldungen genau neben den Feldern zu platzieren, an zentraler Stelle allgemeine Tipps einzublenden und daneben noch bereits korrekte Daten zu erhalten. Mit Validation Controls brauchen Sie sich um all das nicht mehr zu kümmern.

Aufbau der Validation Controls

Es gibt zwei Arten von Validation Controls (VC), die Sie einsetzen können:

- *Serverseitige VC*. Diese senden das Formular zurück zum Server und dort läuft eine VB-Prozedur zur Prüfung ab.

- *Clientseitige VC*. Hier wird JavaScript-Code erzeugt, mit dem die Prüfung vorgenommen wird. Es wird kein ActiveX oder VBScript verwendet.

Die verfügbaren Elemente sind in der folgenden Tabelle zusammengefasst:

Tabelle 9.3: Elemente mit Prüffunktionen

Funktion	Element	Beschreibung
Erforderliche Felder	`RequiredFieldValidator`	Dieses Element prüft, ob erforderliche Felder wirklich ausgefüllt wurden.
Wertevergleich	`CompareValidator`	Dieses Element vergleicht den eingegebenen Wert mit einer Konstanten oder einem anderen Feld mit logischen Operatoren.
Bereichskontrolle	`RangeValidator`	Prüft, ob der eingegebene Wert zwischen einer unteren und einer oberen Grenze liegt. Zulässig sind Daten, Zeichen und numerische Werte.
Mustervergleich	`RegularExpression-Validator`	Prüft den Inhalt mit Hilfe eines regulären Ausdrucks.
Benutzerdefiniert	`CustomValidator`	Dieses Element verwendet eine benutzerdefinierte Funktion zur Inhaltsprüfung.

Neben diesen feldbezogenen Elementen gibt es noch ein abstraktes Element, das zur Generierung einer Zusammenfassung der Fehlerinformation dient: `ValidationSummary`.

Typische Probleme mit Formularen

Bei der Arbeit mit Formularen sollten Sie sicherstellen, dass fremde Server diese nicht senden können. Ansonsten sind Angriffe von außerhalb möglich. Das ist besonders kritisch, wenn die Auswertung bereits im Browser stattfindet. Nutzer könnten das Formular lokal ablegen, manipulieren und dann versenden. Dabei hilft auch nicht die Auswertung von Session-IDs oder anderen Techniken zur Nutzererkennung – denn diese Informationen stecken im Formular. Auf Grund der geringen Netzwerkbandbreite ist jedoch die Prüfung mit JavaScript weit verbreitet. Oft wird sie auch verwendet, weil es einfacher erscheint als mit Serverskripten. ASP.NET bringt hier einen signifikanten Fortschritt. Die Entscheidung zwischen clientseitiger und serverseitiger Prüfung erfolgt mit einer einzigen Anweisung der `Page`-Direktive. Als Downlevel werden Browser bezeichnet, die kein JavaScript unterstützen oder dieses nicht verwenden sollen. Schreiben Sie an den

Anfang des Skripts folgende Zeile, um die serverseitige Prüfung zu erzwingen:

```
<%@ Page ClientTarget="DownLevel" %>
```

Umgekehrt können Sie auch festlegen, dass vorzugsweise JavaScript verwendet wird. Erst wenn das Formular korrekt ausgefüllt wurde, wird es zum Server übertragen. Verwenden Sie folgende Direktive:

```
<%@ Page ClientTarget="UpLevel" %>
```

Anwendungsbeispiele für VCs

Es ist sinnvoll, Fehlerinformationen mit einer klaren Benutzerführung zu kombinieren. Nutzer fühlen sich angesichts von Fehlermeldungen irritiert. Vor allem Anfänger neigen dazu, den Vorgang schnell anzubrechen, wenn Probleme auftreten. Sie müssen hier Hilfestellung leisten. ASP.NET unterstützt das exzellent.

ValidationSummary

Zuerst überlegen Sie sich einen guten Platz, vorzugsweise am Anfang der Seite, um eine Zusammenfassung der Fehler anzuzeigen. So werden Nutzer auf einen Blick über *alle* Fehler informiert. Es ist sehr unprofessionell, wenn ein Nutzer aufgrund eines Hinweises im sichtbaren Teil des Formulars einen Fehler verbessert, einige Sekunden auf den Rücklauf wartet und dann erneut mit einer weiteren Fehlermeldung konfrontiert wird. Sie sollten gleich auf alle Probleme hinweisen und so die Chance erhöhen, dass der zweite Versuch erfolgreich ist. Das `ValidationSummary`-Control bauen Sie genau dort ein, wo die Meldungen erscheinen sollen:

```
<asp:ValidationSummary id="<val_id>" runat="server"/>
```

Sie können mit verschiedenen Attributen das Erscheinungsbild bestimmen:

- `ForeColor`: Farbe des Meldungstextes
- `BackColor`: Farbe des Hintergrunds
- `Font`: Font-Informationen, Größe, Schnitt usw.
- `BorderWidth`, `BorderColor` und `BorderStyle`: Umrandung
- `CSSStyle` und `CSSClass`: Styles, wie sie aus CSS bekannt sind.

Außerdem stehen Attribute zur Verfügung, mit denen Sie das Element aus dem VB-Code heraus kontrollieren können. Dazu gehen Sie folgendermaßen vor:

```
validationId.Property = "Value"
```

Wobei *validationId* der Wert des Parameters `id` ist. Einige wichtige Attribute sind:

- `ShowMessageBox`. `TRUE` oder `FALSE`: Wenn dieses Attribut gesetzt ist und `UpLevel` verwendet wird, erscheint die Meldung als JavaScript-Nachricht, d.h., in einer Pop-Up-Box.
- `HeaderText`: Text einer Überschrift
- `DisplayMode`: Art der Anzeige

RequiredField

Mit `RequiredField` können Sie eine Reaktion auf unausgefüllte Felder bestimmen. Oft werden bestimmte Felder benötigt. Es ist üblich, solche Felder mit einem Symbol, z.B. einem roten Sternchen, zu kennzeichnen. Werden sie dennoch nicht ausgefüllt, ist ein kurzer Hinweis der Art »Bitte füllen Sie dieses Feld aus« angebracht. Das folgende Element platzieren Sie an der Stelle, an der der Hinweis erscheinen soll:

```
<asp:RequiredFieldValidator id="valid" runat="server"/>
```

Sie können auch eine Nachricht bestimmen, die erscheint, wenn das Feld noch nicht ausgefüllt wurde. Folgende Attribute zeigen eine Auswahl der Möglichkeiten:

- `controlToValidate = "id"`: ID des Elements, das kontrolliert werden soll.
- `errorMessage = "Text"`: Text der Fehlerinformation
- `inititalValue = "Text"`: Text für das unausgefüllte Element.
- `display = "static"`. Dieses Attribut bestimmt, ob das Element auf der Seite auch dann Platz beansprucht, wenn es nichts anzeigt. Das ist sinnvoll, wenn Sie ein sehr empfindliches Layout haben, das zerfällt, wenn Text fehlt. Setzen Sie `"dynamic"` ein, wenn die Reservierung nicht notwendig ist.

CompareValidator

Mit diesem Element kontrollieren Sie den Inhalt mittels eines einfachen Vergleichs, also Gleich, Größer, Kleiner usw. Verglichen werden kann der Wert eines Feldes mit einer Konstanten, einem dynamischen Wert oder einem anderen Feld. Häufig wird dies verwendet, um zwei verdeckte Kennwort-Felder zu vergleichen. Der folgene Code zeigt diese Anwendung:

```
<input type="password" id="fldPW1" runat="server"/>
<input type="password" id="fldPW2" runat="server"/>
<asp:CompareValidator id="compid" runat="server"
     controlToValidate="fldPW1"
     controlToCompare="fldPW2"
     errorMessage="Beide Kennwortfelder müssen gleich sein"
     type="String"
     operator="Equal"
     display="static"
/>
```

Den zu überprüfenden Feldtyp zeigt die foldende Liste:

- `String`
- `Integer`
- `Double`
- `DateTime`
- `Currency`

Die zulässigen Operatoren werden mit folgenden Wörtern benannt:

- `Equal`: Gleich
- `NotEqual`: Ungleich
- `GreaterThan`: Größer als
- `GreaterThanEqual`: Größer als oder gleich
- `LessThan`: Kleiner als
- `LessThanEqual`: Kleiner als oder gleich
- `DateTypeCheck`: Datumsprüfung; prüft nur, ob es sich um ein gültiges Datumsformat handelt, ohne einen Kontrollwert einzubeziehen.

RangeValidator

Dieses Element prüft den Wert auf die Einhaltung einer unteren und einer oberen Grenze:

```
<asp:RangeValidator id="rangeid" runat="server"
     controlToValidate="FldAlter"
     errorMessage="Ihre Altersangabe ist unzulässig"
     minimumValue="18"
     maximumValue="90"
     display="static"
/>
```

RegularExpression Validator

Hiermit prüfen Sie den Inhalt mit Hilfe eines regulären Ausdrucks. Sie sollten bedenken, dass diese Prüfung auch im Client stattfinden kann. Dort steht möglicherweise nur JavaScript 1.0 zur Verügung. Die Variationsvielfalt regulärer Ausdrücke, wie sie z.B. Perl bietet, ist dort nicht gegeben. Trotzdem gibt es leistungsstarke Anwendungen. Das Beispiel am Ende des Abschnitts zeigt die Prüfung einer E-Mail-Adresse. Hier die Syntax des Elements:

```
<asp:RegularExpressionValidator id="regID" runat="Server"
     controlToValidate="Email"
     errorMessage="Ihre Altersangabe ist unzulässig"
     validationExpression="^(.)+@(.)+\.(.){2,3}"
     display="static"
/>
```

CustomValidator

Wenn Sie Kreditkartennummern oder Bankleitzahlen prüfen möchten, reichen reguläre Ausdrücke nicht aus. Berechnungen sind damit nicht möglich. Sie können aber auch eigene Funktionen definieren. Diese werden vom Element `CustomValidation` aufgerufen.

```
<asp:CustomValidator id = "custID" runat = "Server"
     controlToValidate = "BLZ"
     errorMessage = "Die Bankleitzahl ist falsch"
     onErrorValidationFunction = "CheckBLZ"
     display = "static"
/>
```

Kontrolle der Seitensteuerung

IsValid

Sie können noch zwei Zustände unterscheiden, die den Eintritt ins Formular erleichtern:

```
Page.IsValid
```

Damit wird bei Auftreten des `Page_Load`-Events ausgewertet, ob alle Felder gültig sind. Mit `Server.Transfer` können Sie, wenn noch keine Zeichen an den Browser gesendet wurden, die Verarbeitung an ein weiteres Skript weiterreichen. Weitergereicht werden auch die Inhalte der Felder:

```
Server.Transfer = "SkriptName.aspx"
```

IsPostBack

Wenn es nur darum geht zu erkennen, ob ein Formular zurückgesendet wurde, nutzen Sie folgende Eigenschaft:

```
Page.IsPostBack
```

Damit lässt sich feststellen, ob das Formular noch nie angezeigt wurde oder nicht. Sie vermeiden so, dass bei leeren Formularen gleich alle Felder mit Fehlern erscheinen.

9.2.4 WebForm Controls

Nachfolgend finden Sie eine Übersicht über alle WebForm Controls.

Tabelle 9.4: WebForm Controls

Element	Name	Beschreibung
Textanzeige (nur Anzeigen)	`Label`	Zeigt Text an, den Nutzer nicht verändern können
Textbearbeitung	`TextBox`	Textfeld zur Eingabe von Text durch den Nutzer oder durch die Applikation
Listenauswahl	`DropDownList`	Auswahl von Einträgen einer Liste. Die Liste ist ausklappbar.

Element	Name	Beschreibung
Listenauswahl	`ListBox`	Auswahl mehrere Einträge aus einer Liste. Die Liste ist mehrere Einträge lang.
Grafische Anzeige	`Image`	Zeigt ein Bild an
Bannerrotation	`AdRotator`	Zeigt eine Bildsequenz an, per Zufall oder durch Steuerinformationen
Kontrollkästchen	`CheckBox`	Zeigt ein Kontrollkästchen an, das aktiviert oder deaktiviert werden kann
Liste von Kontrollkästchen	`CheckBoxList`	Erzeugt eine Liste von Kontrollkästchen
Optionsfeld	`RadioButton`	Erzeugt ein Optionsfeld
Optionsfeldgruppe	`RadioButtonList`	Erzeugt eine Gruppe von Optionsfeldern
Datumsfeld	`Calendar`	Erzeugt einen grafischen Kalender zur Auswahl eines Datums
Schaltfläche	`Button`	Löst eine Aktion aus. Das Formular muss dazu zum Server gesendet werden
Hyperlink-Schaltfläche	`LinkButton`	Löst eine Aktion aus, hat aber das Aussehen eines Hyperlinks. Das Formular muss dazu zum Server gesendet werden.
Bild-Schaltfläche	`ImageButton`	Bild, das als Schaltfäche funktioniert. Das Formular muss dazu zum Server gesendet werden.
Hyperlink	`Hyperlink`	Ein Hyperlink
Tabelle	`Table`	Erzeugt eine Tabelle
Tabellenzelle	`TableCell`	Erzeugt eine Tabellenzelle innerhalb einer Reihe
Tabellenreihe	`TableRow`	Erzeugt eine Reihe innerhalb einer Tabelle; kann mit Tabellenzellen gefüllt werden
Gruppe	`CheckBoxList`	Erzeugt eine Liste von Kontrollkästchen
Gruppenfeld	`Panel`	Erzeugt eine Umrandung um mehrere andere Elemente, um eine logische Gruppe anzudeuten.

Element	Name	Beschreibung
Optionsgruppe	`RadioButtonList`	Erzeugt eine Gruppe zusammengehöriger Optionsfelder
Listenelement	`Repeater`	Erzeugt eine Liste mit Hilfe frei wählbarer HTML-Tags. Die Liste bezieht ihre Daten aus einer `DataSet`-Datenquelle.
Datenliste	`DataList`	Wie Repeater, aber mit der Möglichkeit der Bearbeitung der Daten durch den Nutzer
Datengitter	`DataGrid`	Zeigt Daten in Tabellenform an

Schaltflächen

Schaltflächen sind am einfachsten zu erzeugen. Das unterscheidet sich aber nur unwesentlich von HTML. Der Standardbefehl dazu sieht folgendermaßen aus:

```
<asp:Button id="ButtonID" runat="Server"/>
```

Alle anderen Attribute können entweder durch Aufruf der passenden Eigenschaften des Objekts oder durch Angabe im Tag erfolgen. Das folgende Skript zeigt die Einrichtung mehrerer Optionsfelder und die entsprechende Reaktion darauf:

```
<%@ Import NameSpace="System.Data" %>
<%@ Import NameSpace="System.Data.ADO" %>
<%@ Page ClientTarget="DownLevel" %>
<html>
<head>
<style>
#VotePanel { background-color:"#efefef"; margin:2px }
</style>
<script language="VB" runat="Server">
Sub Page_Load(o AS Object, e As EventArgs)
   dim note as Integer
   if Page.IsPostBack then
      AcknowLedge.Text = "Vielen Dank f&uuml;r die
                          Beurteilung.<p>"
      if Button1.Checked then note = 1
      if Button2.Checked then note = 2
      if Button3.Checked then note = 3
      if Button4.Checked then note = 4
      if Button5.Checked then note = 5
      Result.Text = "<b>Ihre Note war " & note & "</b>"
```

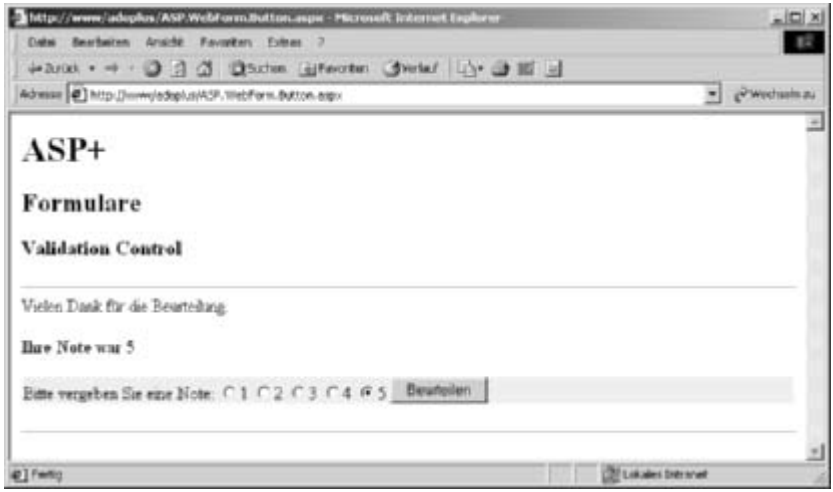
```
   end if
End Sub
```

```
</script>
</head>
<body>
<h1>ASP.NET</h1>
<h2>Formulare</h2>
<h3>Validation Control</h3>
<p/>
<hr noshade size="1">
<asp:Label id="Acknowledge" runat="Server"/>
<asp:Label id="Result" runat="Server"/>
<form runat="Server" id="NoteForm">
<asp:Panel id="VotePanel" runat="server">
Bitte vergeben Sie eine Note:
<asp:RadioButton id="Button1" text="1"
                 runat="Server" GroupName="Vote" />
<asp:RadioButton id="Button2" text="2"
                 runat="Server" GroupName="Vote" />
<asp:RadioButton id="Button3" text="3"
                 runat="Server" GroupName="Vote" />
<asp:RadioButton id="Button4" text="4"
                 runat="Server" GroupName="Vote" />
<asp:RadioButton id="Button5" text="5"
                 runat="Server" GroupName="Vote" />
<asp:Button text="Beurteilen" id="SendButton" runat="Server"/>
</asp:Panel>
</form>
<hr noshade size="1">
</body>
</html>
```

Listing 9.1: ASP.WebForm.Button.aspx: Verschiedene WebForm-Elemente in Aktion

Die Auswertung ist hier noch nicht besonders ausgereift, es soll aber nur das Grundprinzip verständlich gemacht werden. Letztendlich sind die wirklich notwendigen Elemente von Ihrer Applikation abhängig. Im folgenden Kapitel zu ADO.NET werden Sie weitere WebForm Controls kennen lernen, die nur zusammen mit einer Datenquelle angewendet werden können.

Abbildung 9.1: Listing 9.1 in Aktion



9.2.5 Anwendungsbeispiel

Das folgende Listing zeigt ein Formular, das die Felder so ausfüllt, wie es die Tabelle *Customers* der Datenbank Northwind erfordert.

```
<%@ Import NameSpace="System.Data" %>
<%@ Import NameSpace="System.Data.ADO" %>
<%@ Page ClientTarget="DownLevel" %>

<html>
<head>
<meta name="GENERATOR" content="Microsoft Visual Studio 7.0">
<meta name="CODE_LANGUAGE" content="Visual Basic.7.0">
<script language="VB" runat="Server">

Function ValidateZIP(o As Object, strZIP As String) As Boolean
   if Len(strZIP) <> 5 then
      ValidateZIP = FALSE
     exit Function
   elseif val(strZIP) < 10000 Or val(strZIP) >= 100000 then
      ValidateZIP = FALSE
     exit Function
   end if
   ValidateZIP = TRUE
End Function

Sub Page_Load(Source As Object, E As EventArgs)
   Call SetUpForm()
End Sub
```

```
Sub SetUpForm()
   Dim dicCountry As ArrayList
   dicCountry = New ArrayList
   dicCountry.Add ("Baden-W&uuml;rttemberg")
   dicCountry.Add ("Berlin")
   dicCountry.Add ("Brandenburg")
   dicCountry.Add ("Bayern")
   dicCountry.Add ("Bremen")
   dicCountry.Add ("Hamburg")
   dicCountry.Add ("Hessen")
   dicCountry.Add ("Mecklenburg-Vorpommern")
   dicCountry.Add ("Niedersachsen")
   dicCountry.Add ("Nordrhein-Westfalen")
   dicCountry.Add ("Rheinland-Pfalz")
   dicCountry.Add ("Saarland")
   dicCountry.Add ("Sachsen")
   dicCountry.Add ("Sachsen-Anhalt")
   dicCountry.Add ("Schleswig-Holstein")
   dicCountry.Add ("Th&uuml;ringen")

   with valCompanyName
      .controlToValidate = "CompanyName"
      .errorMessage = "Firmenname muss ausgefüllt werden"
      .display = 2
   end with
   with valContactName
      .controlToValidate = "ContactName"
      .errorMessage = "Name des Ansprechpartners muss
                       ausgef&uuml;llt werden"
      .display = 2
   end with
   with valAddress
      .controlToValidate = "Address"
      .errorMessage = "Adresse muss ausgef&uuml;llt werden"
      .display = 2
   end with
   with valCity
      .controlToValidate = "City"
      .errorMessage = "Stadt muss angegeben werden"
      .display = 2
   end with
   with valZIP
      .controlToValidate = "ZIP"
      .errorMessage = "Postleitzahl muss ausgefüllt werden"
```

```
      .display = 2
   end with
   with cValZIP
      .controlToValidate = "ZIP"
      .errorMessage = "Postleitzahl nicht korrekt"
      .display = 2
   End With
   with Country
      .Rows = 1
      .AutoPostBack = FALSE
      .DataSource = dicCountry
      .DataBind()
   End with
   with valCountry
      .controlToValidate = "Country"
      .errorMessage = "Land muss ausgew&aumlhlt werden"
      .display = 2
   end with
   with valeMail
      .controlToValidate = "eMail"
      .errorMessage = "E-Mail muss angegeben werden"
      .display = 2
   end with
   with regValeMail
      .controlToValidate = "eMail"
      .validationExpression = "^[_a-zA-ZO-9-]+(\.
       [_a-zA-ZO-9-]+)*@[_a-zA-ZO-9-]+\.([a-zA-Z]{2,3})$"
      .errorMessage = "E-Mail Adresse ist nicht g&uuml;ltig"
      .display = 2
   end with
   with SendButton
      .text = "Formular absenden"
   End With
End Sub

</script>
</head>
<body>
<h1>ASP.NET</h1>
<h2>Formulare</h2>
<h3>Validation Control</h3>

<p/>
<hr noshade size="1">
<form runat="Server">
```

```
<asp:ValidationSummary id="Summary" runat="Server"
     headerText="Es traten Fehler auf:"
     showSummary="TRUE"
     displayMode="BulletList"
/>
<table border=1>
   <tr>
      <td>CompanyName</td>
     <td><asp:TextBox id="CompanyName"
                          runat="Server"/></td>
     <td><asp:RequiredFieldValidator id="valCompanyName"
                  runat="Server" /></td>
   </tr>
   <tr>
      <td>ContactName</td>
     <td><asp:TextBox id="ContactName"
                          runat="Server"/></td>
     <td><asp:RequiredFieldValidator id="valContactName"
                  runat="Server"/></td>
   </tr>
   <tr>
      <td>ContactTitle</td>
     <td><asp:TextBox id="ContactTitle"
                  runat="Server"/></td>
     <td></td>
   </tr>
   <tr>
      <td>Address</td>
     <td><asp:TextBox id="Address" runat="Server"/></td>
     <td><asp:RequiredFieldValidator id="valAddress"
                  runat="Server"/></td>
   </tr>
   <tr>
      <td>City</td>
     <td><asp:TextBox id="City" runat="Server"/></td>
     <td><asp:RequiredFieldValidator id="valCity"
                  runat="Server"/></td>
   </tr>
   <tr>
      <td>ZIP</td>
     <td><asp:TextBox id="ZIP" runat="Server"/></td>
     <td>
        <asp:RequiredFieldValidator id="valZIP"
                 runat="Server"/>
       <asp:CustomValidator id="cvalZIP"
```

```
                    runat="Server"
                    OnServerValidationFunction="ValidateZIP"/>
      </td>
   </tr>
   <tr>
      <td>Country</td>
     <td><asp:ListBox id="Country" runat="Server"/></td>
     <td><asp:RequiredFieldValidator id="valCountry"
                 runat="Server"/></td>
   </tr>
   <tr>
      <td>Phone</td>
     <td><asp:TextBox id="Phone" runat="Server"/></td>
     <td></td>
   </tr>
   <tr>
      <td>eMail</td>
     <td><asp:TextBox id="eMail" runat="Server"/></td>
     <td>
        <asp:RequiredFieldValidator id="valeMail"
                 runat="Server"/>
       <asp:RegularExpressionValidator
                    id="regValeMail" runat="Server"/>
     </td>
   </tr>
</table>
<asp:Button id="SendButton" runat="Server"/>
</form>
<hr noshade size="1">
</body>
</html>
```

Listing 9.2: ASP.ValidationControls.Sendform.aspx: Komplexes Formular mit Prüfung für alle Felder und detaillierten Fehlernachrichten

Auch wenn das Listing nicht gerade kurz erscheint, ist diese Methode doch enorm leistungsfähig. Die Gestaltung des Formulars selbst kann außerdem mit jedem Editor vorgenommen werden, HTML-Profis werden hier keine Probleme haben. Die eigentlichen Funktionen stecken im Code der Seite.

Anmerkungen zur PreView-Version

Der PreView-Version vom Sommer 2000, mit der dieses Buch entstand, merkt man den frühen Charakter deutlich an. Längst nicht alle Funktionen reagieren erwartungsgemäß. So wird als Argument für `display` der Wert `"static"` oder `"dynamic"` erwartet, wenn die Angabe im Tag erfolgt. Schreiben Sie es dagegen als Eigenschaft im VB-Code, werden numerische Werte verlangt (1 für `static` und 2 für `dynamic`).

Abbildung 9.2: Das Formular mit sauber generierten Fehlermeldungen

In der endgültigen Version dürften solche Unterschiede beseitigt worden sein. Wenn Sie mit ASP.NET experimentieren, sollten Sie längere Versuchszeiten einplanen.

9.3 Datenbindung

Einer der zentralen Aspekte bei der Programmierung von datenbankgestützten Applikationen mit ASP.NET ist die Ausgabe von strukturierten Daten. Dazu wurden in Kapitel 9 bereits die entsprechenden ADO.NET-Objekte diskutiert. Für die Ausgabe mit ASP.NET stehen einige Elemente zur Verfügung, die insbesondere von ADO.NET aus angesprochen werden können.

9.3.1 Einführung

Das Stichwort für die Verknüpfung von ADO.NET-Ausgaben mit ASP.NET-Anzeigeelementen heißt Datenbindung (*Data Binding*). Dabei werden vor-

gefertigte Elemente bereitgestellt, die zur Laufzeit die Daten aus den ADO.NET-Objekten übernehmen und HTML-Code erzeugen. Diese Bindung läuft tatsächlich nur auf dem Server ab – im Gegensatz zu den RDO-Funktionen in ADO, die zwar ähnlich gedacht waren, aber entsprechende ActiveX-Steuerelemente im Browser voraussetzten.

Elementtypen für Datenanzeige

Es gibt zwei Elementtypen, die zur Ausgabe herangezogen werden können:

Wiederholende Elemente

- *Wiederholende Elemente*. Solche Elemente erzeugen Listen, geben also mehrere Datensätze aus.

Einfache Elemente

- *Einfache Elemente*. Das sind Eingabefelder, statische oder dynamische Textmarken usw.

Grundsätzliche Syntax

Alle Elemente sind als ASP.NET-Tags definiert. Dies ist typische XML-Syntax mit `ASP` als Namensraum. Im HTML-Template schreiben Sie an der Stelle, an der das Element erscheinen soll:

```
<asp:Element id="ElementName" parameter… >
```

An dieses Element werden mit der Methode `DataBind` Daten gebunden:

```
ElementName.DataSource = <ObjektDasDatenEnthält>
ElementName.DataBind()
```

Als Datenquelle kann eine Liste, ein Array, eine Eigenschaft, eine Kollektion oder das Ergebnis einer Methode dienen.

Bindung an Arrays

Eine einfache Form ist die Ausgabe des Inhalts eines Arrays in einer Liste vom Typ `Listbox`. Das folgende Listing zeigt den VB-Code und den HTML-Teil:

```
<html>
<head>
<script language="VB" runat="server">
Sub Page_Load(Source As Object, E As EventArgs)
  If IsPostBack then
     NameListAnswer.Text = "Sie haben <b>" &
                           NameList.SelectedItem.Text &
                           "</b> ausgewählt."
  else
     Dim dicNames As ArrayList
     dicNames = New ArrayList
     dicNames.Add ("M&uuml;ller, Helga")
```

```
    dicNames.Add ("Schultze, Olaf")
    dicNames.Add ("Marquardt, Bernd")
    NameList.DataSource = dicNames
    NameList.DataBind()
  end if
End Sub
</script>
</head>
<body>
<p/>
   <asp:Label id="NameListAnswer" runat="Server"/>
   <p/>
   <form runat="server">
   <asp:ListBox id="NameList"
                runat="Server"
                size="1"
                autopostback="TRUE"/>
   </form>
<hr noshade size="1">
</body>
</html>
```

Listing 9.3: ASP.DataBinding.ListBox.aspx: Ausgabe eines Arrays in einem Formular mit interaktiver Reaktion

Dieses Skript erzeugt eine Drop-Down-Listbox. Bei jeder Änderung wird das Formular abgesendet und das Ergebnis dargestellt. Als Erstes soll ein Blick in den HTML-Code im Browser klären, was erzeugt wird:

```
<p/>
<span id="NameListAnswer">Sie haben <b>Schultze, Olaf</b>
 ausgewählt.</span>
<p/>
<FORM name="ctrl3" method="post"
      action="ASP.DataBinding.ListBox.aspx" id="ctrl3">
<INPUT type="hidden" name="__EVENTTARGET" value="">
<INPUT type="hidden" name="__EVENTARGUMENT" value="">
<INPUT type="hidden" name="__VIEWSTATE" value="aOz-777898305_aOz_
hz5z4x_aOz_hz5z1x_aOzaOz_aOzaOzazM|ller, Helga_M|ller, Helgax_
azSchult\ze, Olaf_Schult\ze, Olafx_azMarquardt, Bernd_Marquardt,
Berndxxxx_xxx_5z2x_aOzaOzhzTe\xt_Sie haben <b>Schult\ze, Olaf</b>
ausgewählt.x_x_xxx_x\t5O_System.String">
<script language="javascript">
<!--
  function __doPostBack(eventTarget, eventArgument) {
     var theform = document.ctrl3
```

```
    theform.__EVENTTARGET.value = eventTarget
    theform.__EVENTARGUMENT.value = eventArgument
    theform.submit()
  }
// -->
</script>
<select name="NameList" id="NameList" size="1" size="5"
        onchange="javascript:__doPostBack('NameList', '')">
<option value="M|ller, Helga">M|ller, Helga</option>
<option selected value="Schultze, Olaf">
                              Schultze, Olaf</option>
<option value="Marquardt, Bernd">Marquardt, Bernd</option>
</select>
</FORM>
<hr noshade size="1">
</body>
</html>
```

Listing 9.4: Ausgabe des Skripts aus Hier kommt JavaScript ins Spiel

Die Liste wird, wie nicht anders zu erwarten war, mit `<select>` erzeugt. Beim Ändern des Eintrags wird mit `onchange` ein Ereignis ausgelöst und das Formular über die JavaScript-Funktion `__dopostback()` gesendet. Damit dieser JavaScript-Teil erzeugt wird, tragen Sie den Parameter `autopostback="TRUE"` ein. Entfällt dies, müssen Sie die Sendeschaltfläche selbst erzeugen.

Was passiert aber nun mit den zurückgesendeten Daten? In ASP.NET gibt es die Funktion `IsPostBack`. Wenn immer ein Formular zurückgesendet wurde, ist diese Funktion `TRUE`. Dann wird entsprechend verzweigt:

```
  If Page.IsPostBack then
     NameListAnswer.Text = "Sie haben <b>" &
                           NameList.SelectedItem.Text &
                           "</b> ausgewählt."
```

`NameListAnswer` ist auch ein Web Form-Element, es dient der Anzeige von Text. In HTML steht hier nur ein `<span>`-Element:

```
<span id="NameListAnswer">
  Sie haben <b>Schultze, Olaf</b> ausgewählt.
</span>
```

Auf die Auswahl der Liste wird wieder über `NameList` zugegriffen. Die Eigenschaft `SelectedItem` enthält das ausgewählte Element, `Text` gibt die Zeichenkette zurück. Das Ergebnis sehen Sie in der folgenden Abbildung:

Abbildung 9.3: Das Formular nach dem Absenden

In der Preview-Version hatte ASP.NET noch arge Probleme mit Umlauten. Hier sollten Sie keinen Aufwand verschwenden, dies mit Skripten lösen zu wollen. Ich bin sicher, dass in der endgültigen Version Umlaute problemlos verarbeitet werden.

Ohne JavaScript

Ohne JavaScript müssen Sie eine Sendeschaltfläche verwenden. Der Code im Formular sieht dann folgendermaßen aus:

```
<asp:ListBox id="NameList" runat="Server" size="1"/>
<input type="Submit" id="Sender"
       value="Absenden" runat="Server">
```

Dieses Element erzeugt lediglich das korrespondierende HTML-Tag. Der Parameter `runat="Server"` führt zur Verfügbarkeit des Inhalts im Code des Skripts. Sie sollten nie alte und neue Elemente mischen, sondern konsequent mit ASP.NET-WebControls arbeiten, auch wenn es – wie in diesem Fall – keine zusätzliche Funktionalität bietet.

Trennung von Optionen und Anzeige

In Listboxen ist es üblich, die übermittelten Werte nicht den angezeigten Elementen zu entnehmen. Sprachliche Änderungen könnten sonst Probleme mit dem auswertenden Code bereiten. Das WebControl `ListBox` kennt dazu zwei weitere Parameter:

```
<asp:ListBox id="NameList"
             DataTextField="Name"
             DataValueField="NameID"
             runat="Server"
             size="1"/>
```

Der Zugriff erfolgt über entsprechende Eigenschaften einer mit dem Objekt verbundenen Klasse. Um `ArrayList` mit Eigenschaften zu versehen, werden statt Zeichenketten Objekte gespeichert. Eine entsprechende Klassendefinition ist nicht neu, schon VBScript kannte ab Version 5 einfache Klassen, Visual Basic sowieso. Die Syntax hat sich etwas geändert, nehmen Sie hier erst einmal die Schreibweise als gegeben hin:

```
public class clsNames
  private intNameID as Integer
  pprivate strName   as String

  public sub New (ThisID As Integer, ThisName As String)
    MyBase.New
    intNameID = ThisID
    strName   = ThisName
  end sub

  ReadOnly Property Name() As String
    Get
      Name = strName
    End Get
  End Property

  ReadOnly Property NameID() As String
    Get
      NameID = intNameID.ToString
    End Get
  End Property
end class
```

Listing 9.5: Ausschnitt aus ASP.DataBinding.ListBox.3.aspx: Erzeugen einer Klasse, auf die Listbox-Parameter Zugriff haben

Definiert wird hier ein Konstruktor (durch den reservierten Namen `New()` der Prozedur). Dieser übernimmt zwei Werte und weist sie den internen Variablen *intNameID* und *strName* zu. Dann folgt die Definition zweier Eigenschaften, die nur gelesen werden können. Durch das Schlüsselwort `ReadOnly` kann die Angabe einer `Set`-Folge vermieden werden. Umgekehrt würde `WriteOnly` die `Get`-Folge sparen. Wenn nun das `ListBox`-Control Werte abruft, greift es auf die Eigenschaften `Name()` und `NameID()` zurück. Ein Ausschnitt aus dem erzeugten HTML-Code zeigt das Ergebnis:

```
<SELECT id=NameList size=1 name=NameList>
   <OPTION value=1 selected>M&uuml;ller, Helga</OPTION>
   <OPTION value=2>Schultze, Olaf</OPTION>
   <OPTION value=3>Marquardt, Bernd</OPTION>
</SELECT>
```

```
<INPUT id=Sender type=submit value=Absenden name=Sender>
```

Der Rest funktioniert unverändert. Wie die Werte zurückgelangen, spielt für den Programmierer keine Rolle. Die Methode `Text` sorgt dafür, dass weiterhin der Text angezeigt wird, in der Regel der Wert der Option. Alternativ können Sie mit Value auch auf den Wert zugreifen:

```
NameList.SelectedItem.Value
```

Fehlerbehandlung

Abfangen und Bedienen von Eingabefehlern

An dieser Stelle sollten Sie auch über eine einfache Fehlerbehandlung nachdenken. Wir reden hier von VB7 – nicht von VBScript. Vergessen Sie deshalb `on error resume next`. In VB7 gibt es endlich das elegantere `Try Catch Finally`-Konstrukt. Das folgende Listing zeigt, wie es verwendet wird:

```
If IsPostBack then
  Try
    NameListAnswer.Text = "Sie haben <b>" &
                          NameList.SelectedItem.Text &
                          "</b> ausgewählt."
  Catch myError As Exception
    NameListAnswer.Text = "Bitte wählen Sie einen Namen aus."
  End Try
else
... ' Entspricht dem vorhergehenden Listing
```

Listing 9.6: ASP.DataBinding.ListBox.4.aspx: Abfangen möglicher Fehler (Ausschnitt)

Der Fehler, der hier auftreten kann, besteht im Absenden des Formulars ohne Auswahl eines Eintrags der Listbox. Dann ist die Eigenschaft `SelectedItem` gleich `Null` und die Anwendung von Text misslingt. Der Fehler wird nun abgefangen – in der Variablen *myError*. Dies ist hier nur ein Dummy – in jedem Fall wird der Hinweistext im `Catch`-Zweig angezeigt.

Das ging mit ASP aber auch alles, werden Sie vielleicht denken. Sicher, aber die gezielte Zuordnung der Fehlermeldungen zu den Fehlerquellen bereitet doch einige Mühe. Hier ist dies mit einer einfachen Anweisungsfolge möglich. Und immer noch sind Code – hier in Form der `Page_Load`-Prozedur – und HTML streng getrennt.

Der Vorgang der Datenbindung kann sich also auf alle Arten von WebForm-Elementen beziehen. Es lohnt deshalb, einen näheren Blick auf die Möglichkeiten zu werfen, die WebForm-Elemente bieten.

Abbildung 9.4: Reaktion auf einen Fehler

10 ADO.NET im Detail

Um mit ADO.NET arbeiten zu können, werden Sie ein neues Datenmodell kennen lernen. Auf diesem Modell basieren auch die folgenden Beispiele.

Programmierumgebung

Auf ASP.NET und die Programmierumgebung Visual Studio 7 geht Kapitel 9 überblicksartig ein, da nicht vorausgesetzt werden kann, dass alle Leser dies bereits verinnerlicht haben. Außerdem ist das Literaturangebot derzeit (Anfang 2001) noch relativ überschaubar und damit nicht ausreichend hilfreich.

10.1 Das Datenmodell der Datenobjekte

ADO.NET besteht aus zwei elementaren Teilen – `DataSet` und `DataSetCommand`. Für die tägliche Arbeit werden Sie auf das `DataSet`-Objekt zurückgreifen, das zuerst vorgestellt wird. Beide Objekte haben ein eigenes Objektmodell, welches dass bereits vorgestellte elementare Objektmodell von ADO.NET ergänzt.

10.1.1 Das DataSet-Objektmodell

Das `DataSet`-Objektmodell zeigt den Aufbau des `DataSet`-Objekts. Zugriff auf Daten erhalten Sie über die abgeleiteten Objekte wie `DataTable`.

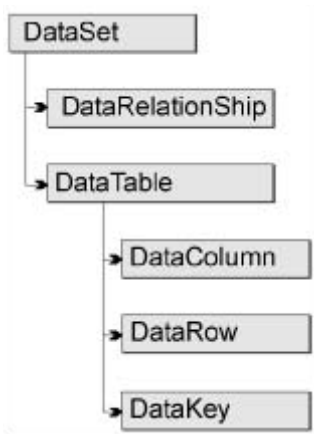


Abbildung 10.1: Das DataSet-Objektmodell

DataSet wurde entworfen, um mit aktuellen Daten aus der Datenquelle zu arbeiten – diese zu lesen, zu verändern und zu löschen. Jedes DataSet kann mehrere Tabellen und Relationen zwischen diesen enthalten. Stellen Sie sich vor, DataSet ist eine offline Kopie der Daten in der Datenbank – der gesamten Datenbank natürlich, nicht nur einer Abfrage wie dies ein RecordSet-Objekt bei ADO repräsentierte.

Der Vorteil wird beim Weiterreichen der Daten zwischen den Teilen der Applikation deutlich. Dazu später mehr.

10.1.2 DataView

Ein zweites wichtiges Objekt ist DataView. Es repräsentiert eine kundenspezifische Sicht auf eine Tabelle, ähnlich einer Sicht in SQL. DataView ist am ehesten mit dem Datensatz-Objekt in ADO vergleichbar. Die Daten werden allerdings generell in XML gespeichert – nicht nur als Option – und können so leicht zwischen Schichten und Applikationen ausgetauscht werden.

10.1.3 Die ersten Schritte

Verbindung zur Datenbank

Der erste Schritt besteht im Öffnen der Verbindung zur Datenbank. Wenn Sie das folgende Skript zum Laufen gebracht haben, bereiten alle folgenden Experimente wenig Probleme. Es gibt zwar noch einige Ungereimtheiten in der frühen Version des Visual Studio 7, aber grundlegende Aufgaben lassen sich in ADO.NET schon lösen.

```
<%@ Import NameSpace="System.Data" %>
<%@ Import NameSpace="System.Data.ADO" %>

<html>
  <head>
    <script language="VB" runat="server">
    Sub Check_DataBase()

      Dim objConn As ADOConnection
      Dim strConn As String
      Dim strSQL  As String
      strConn = "Provider=SQLOLEDB; Data Source=(local);
                 Initial Catalog=Northwind; User ID=sa"
      objConn = New ADOConnection
      objConn.ConnectionString = strConn
      objConn.Open()
      if objConn.State = 0 then
         Response.Write("Fehler")
      else
        Response.Write("Datenbank erfolgreich ge&ouml;ffnet")
      end if
```

```
      objConn.Close
      objConn = Nothing

    End Sub
    </script>

  </head>
  <body>
    <h1>ADO.NET</h1>
    <h2>ADOConnection</h2>
    Dieses Skript versucht eine Verbindung zu einem OLEDB-
    Provider herzustellen.
    <p/>
    <hr noshade size="1">
    <% Check_DataBase() %>
  </body>
</html>
```

Listing 10.1: ADO.OpenConnection.aspx: Öffnen der Verbindung zur Datenbank

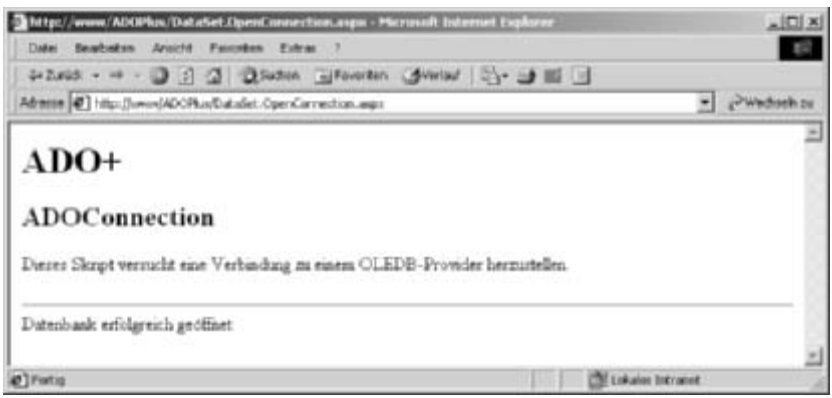


Abbildung 10.2: So sollte die Ausgabe aus Listing 10.1 aussehen

Wie es funktioniert

Im Prinzip gibt es hier – gegenüber ADO – noch nicht viel Neues zu sehen. Zuerst ein Grundprinzip: Alle Objekte werden über Klassenbibliotheken eingebunden, die in der gesamten .NET-Umgebung für alle Programmiersprachen identisch sind. In Visual Basic wird dazu die Anweisung `Import` verwendet. Für die Nutzung von ADO.NET werden zwei Bibliotheken verwendet: SYSTEM.DATA und SYSTEM.DATA.ADO. Die erste bindet die ADO.NET-Bibliothek an sich, die zweite die auf OLEDB abgestimmten Objekte. Daneben finden Sie auch System.Data.SQL, optimiert für das Zusammenspiel mit dem SQL Server (und nur mit diesem). Wenn Sie also auch Textdateien, XML und MS Access bearbeiten möchten, setzen Sie auf ADO. Wenn Sie nur mit dem SQL Server arbeiten, nutzen Sie SQL – die Verbindung ist schneller, weil native Treiber eingesetzt werden.

Im Beispiel wird eine Prozedur definiert, die wie üblich aufgerufen wird. Sie können den Code auch direkt einbinden, wie bisher. In Kapitel 10 werden Sie erfahren, dass ASP.NET hier ganz anders funktioniert als ADO, und diese Technik der konsequenten Trennung von Code und Layout schnell schätzen lernen.

Datentypen müssen in VB7 deklariert werden. Das Verbindungsobjekt und die Zeichenketten sind die ersten Deklarationen:

```
Dim objConn As ADOConnection
Dim strConn As String
```

Verbindungszeichenfolgen wie in ADO

Dann wird die Verbindungszeichenfolge verwendet, die den Provider anspricht. Da die ADO.NET-Bibliothek OLEDB verwendet, unterscheidet sich dieser Teil nicht von ADO.

```
strConn = "Provider=SQLOLEDB; Data Source=(local);
           Initial Catalog=Northwind; User ID=sa"
```

Dann wird ein neues Verbindungsobjekt erzeugt. Statt der Methode `CreateObject` kann nun das VB-Schlüsselwort `New` verwendet werden; Visual Basic-Programmierer werden das schon kennen:

```
      objConn = New ADOConnection
```

ConnectionString

Jetzt wird der Eigenschaft `ConnectionString` die Verbindungszeichenfolge übergeben und die Verbindung wird geöffnet:

```
      objConn.ConnectionString = strConn
      objConn.Open()
```

10.2 Arbeiten mit ADO.NET

Die ersten Schritte in ADO.NET fallen erfahrungsgemäß schwer, weil die alte Denkweise aus ADO nicht sofort abzulegen ist. Dieser Abschnitt zeigt die wichtigsten Techniken ohne jeden Anspruch auf Vollständigkeit. Versuchen Sie unbedingt, die Beispiele zum Laufen zu bekommen, alle anderen Methoden bauen auf diesen Informationen auf.

10.2.1 ADOCommand und ADODataSetCommand

`ADOCommand` dient der Erfassung und der Ausführung von Anweisungen »gegen« die Datenquelle. `ADODataSetCommand` erledigt dies ebenso, sieht aber gleich die Übergabe der Daten an `DataSet` vor. Das äquivalente Objekt in ADO hieß `Command`. Das Ergebnis der Abfrage ist – wenn Daten zurückgegeben wurden – ein autonomes `DataSet`-Objekt (Abschnitt 10.2.2 DataSet ab Seite 356). Komfortable Anzeigemöglichkeiten bietet dagegen `DataView`. (Abschnitt 10.2.3 Wie es weitergeht ab Seite 361).

ADOCommand

Das folgende Beispiel zeigt, wie eine Verbindung zur Datenbank aufgebaut wird und wie eine einfache SQL-Anweisung abgesendet wird.

```
<%@ Import NameSpace="System.Data" %>
<%@ Import NameSpace="System.Data.ADO" %>

<html>
<head>
<script language="VB" runat="server">
public objConn As ADOConnection
public blnConn As Boolean

Sub Page_Load(Source As Object, E As EventArgs)

  Dim strConn As String
  strConn = "Provider=SQLOLEDB; Data Source=(local);
             Initial Catalog=Northwind; User ID=sa"
  objConn = New ADOConnection
  objConn.ConnectionString = strConn
  objConn.Open()
  if objConn.State = 0 then
     blnConn = FALSE
  else
     blnConn = TRUE
  end if
End Sub

Sub Check_Command()
   Dim objC As ADOCommand
   Dim strSQL As String

   if blnConn then
      strSQL = "SELECT * FROM Customers"
      objC = New ADOCommand
      objC.ActiveConnection = objConn
      objC.CommandText = strSQL
      objC.Execute
      Response.Write("Verbindung: " &
                     objC.ActiveConnection.ToString())
      Response.Write("<br>")
      Response.Write("Abfrage: " &
                     objC.CommandText.ToString())
   else
      Response.Write("Es bestand keine Verbindung")
```

```
   end if
End Sub

</script>
</head>
<body>
<h1>ADO.NET</h1>
<h2>ADODataSetCommand</h2>
Dieses Skript versucht eine Verbindung zu einem OLEDB-Provider
herzustellen.
<p/>
<hr noshade size="1">
<% Check_Command() %>
</body>
</html>
```

Listing 10.2: ADOCommand.apsx: Ausführen von Anweisungen gegen die Datenquelle ohne Rückgabe von Daten

Abbildung 10.3: Ausgabe des Skripts in Listing 10.2

Wie es funktioniert

Zuerst fällt auf, dass die Struktur des Skripts etwas anders als in ASP ist. Die Prozedur `Page_Load` wird aufgerufen, wenn die Seite abgerufen wird. Der Name ist reserviert für diesen Zweck. Hier wird nun die Verbindung zur Datenbank hergestellt. Die Übergabe der Informationen erfolgt über zwei öffentliche Variablen:

```
public objConn As ADOConnection
public blnConn As Boolean
```

Die eigentliche Arbeit erledigt die Prozedur *`Check_Command`*. Zuerst wird eine SQL-Abfrage vorbereitet und den entsprechenden Eigenschaften des Objekts übergeben:

```
strSQL = "SELECT * FROM Customers"
objC = New ADOCommand
objC.ActiveConnection = objConn
objC.CommandText = strSQL
```

Dann wird das Kommando ausgeführt:

```
objC.Execute
```

Zur Kontrolle werden die Eigenschaften wieder gelesen. Beachten Sie die Methode `ToString()`, die solche Werte in eine Zeichenkette überführt – ohne diesen Aufruf funktioniert die Verkettung mit & nicht:

```
Response.Write("Verbindung: " &
               objC.ActiveConnection.ToString())
Response.Write("<br>")
Response.Write("Abfrage: " &
               objC.CommandText.ToString())
```

Nach der Ausführung dieses Schritts können Sie alle SQL-Anweisungen an die Datenbank senden.

ADODataSetCommand

Das folgende Skript entspricht dem in Listing 10.2 gezeigten, holt die Daten aber auch wirklich ab. Ab hier wird nur der relevante Teil abgedruckt, der übrige Aufbau des Skripts bleibt unverändert.

```
   Sub Check_Command()
      Dim objC As ADODataSetCommand
      Dim objDS As ADODataSet
      Dim strSQL As String

      if blnConn then
         strSQL = "SELECT * FROM Customers"
         objDS = New ADODataSet
         objC = New ADODataSetCommand(strSQL, objConn)
         objC.FillDataSet(objDS, "Kundenliste")
         Response.Write("Abfrage ausgef&uuml;hrt")
      else
         Response.Write("Es bestand keine Verbindung")
      end if
    End Sub
```

Listing 10.3: ADO.DataSetCommand.aspx: Ausführen einer Abfrage und Übergabe der Daten in ein DataSet-Objekt

Bis dahin können Sie nur sehen, dass das Skript abläuft, ohne Laufzeitfehler auszugeben – von den Daten gibt es keine Spur im Browser.

10.2.2 DataSet

Lokaler Datenspeicher: DataSet

Das `DataSet`-Objekt ist eine einfache Datenbank, die im Hauptspeicher des Computers gehalten wird. Dieses Objekt abstrahiert die Daten vom physikalischen Datenspeicher vollständig. Für die Repräsentation der Daten werden Kollektionen von Tabellen, Spalten, Reihen und Verknüpfungen sowie Einschränkungen angeboten.

Ein DataSet erzeugen

Das erste Beispiel erzeugt eine Verbindung, ein `DataSet`-Objekt und eine `DataView`, die Informationen darüber anzeigt.

```
<%@ Import NameSpace="System.Data" %>
<%@ Import NameSpace="System.Data.ADO" %>

<html>
  <head>
  <script language="VB" runat="server">
   public objConn As ADOConnection
   public blnConn As Boolean

   Sub Page_Load(Source As Object, E As EventArgs)

     Dim strConn As String
     strConn = "Provider=SQLOLEDB; Data Source=(local);
                Initial Catalog=Northwind; User ID=sa"
     objConn = New ADOConnection
     objConn.ConnectionString = strConn
     objConn.Open()
     if objConn.State = 0 then
        blnConn = FALSE
     else
        blnConn = TRUE
     end if

     Check_Command()
  End Sub

  Sub Check_Command()
     Dim objC As ADODataSetCommand
     Dim objDS As DataSet
     Dim strSQL As String
```

```
    if blnConn then
       objDS = New DataSet
       strSQL = "SELECT * FROM Customers"
       objC = New ADODataSetCommand(strSQL, objConn)
       strSQL = "SELECT * FROM Products"
       objC = New ADODataSetCommand(strSQL, objConn)
       objC.FillDataSet(objDS, "Customers")
       objC.FillDataSet(objDS, "Products")
       CustomerGrid.DataSource = objDS.Tables
       CustomerGrid.DataBind()
    else
       Response.Write("Es bestand keine Verbindung")
    end if
  End Sub

</script>
</head>
<body>
<hr noshade size="1">
  <asp:DataGrid id="CustomerGrid" runat="Server"/>
<hr noshade size="1">
</body>
</html>
```

Listing 10.4: ADO.DataSet.DataGrid.aspx: Anzeige von Tabelleninformationen

Wie es funktioniert

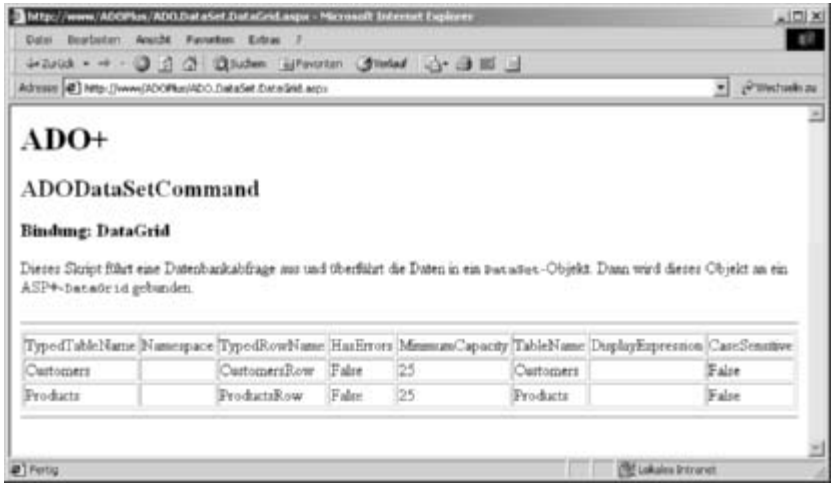
Wie funktioniert es nun genau? Zuerst ist auffällig, dass Sie die Datentypen benennen müssen. Die Applikationssprache ist nicht mehr VBScript, sondern Visual Basic.NET. Hier herrschen strengere Regeln. Wenn Sie im Umgang damit unsicher sind, konsultieren Sie Literatur zu Visual Basic 6, solange zu Visual Basic.NET nichts verfügbar ist.

Benötigt werden drei Objekte:

- `ADOConnection`: Stellt die Verbindung zur Datenbank her
- `ADODataSetCommand`: Führt SQL-Anweisung gegen die Quelle aus.
- `DataSet`: Dieses Objekt gestattet den Zugriff auf die Daten.

Vier Schritte führen zu dem folgenden Ergebnis:

Abbildung 10.4: Ergebnis von Listing 10.4



Zuerst wird die Verbindung hergestellt. Dies unterscheidet sich kaum von ADO:

```
strConn = "Provider=SQLOLEDB; Data Source=(local);
           Initial Catalog=Northwind; User ID=sa"
objConn = New ADOConnection(strConn)
```

Das Schlüsselwort `New` entspricht `Set` in VBScript, hiermit wird also eine Instanz eines Objekts erzeugt. *objConn* ist nun das Verbindungsobjekt.

Jetzt wird die Abfrage in das `ADODataSetCommand`-Objekt übertragen. Dies ist ähnlich dem `Command`-Objekt in ADO zu sehen. Übergeben wird natürlich auch die Verbindung:

```
objComm = New ADODataSetCommand(strSQL, objConn)
```

Das `DataSet`-Objekt wurde bereits bei der Deklaration vorbereitet – es ist schon existent, hat aber keinen Inhalt:

```
Dim objDS As New DataSet
```

FillDataSet

Die folgende Anweisung »befüllt« nun das `DataSet`-Objekt *objDS* mit Daten aus der Abfrage. Dabei wird nur die Tabelle angegeben, die betroffen ist, hier also *Customers*:

```
objComm.FillDataSet(objDS, "Customers")
```

DataGrid-Control

Im nächsten Schritt werden die Daten aller verbundenen Tabellen – es können hier mehr als nur eine Tabelle enthalten sein – mit dem `DataGrid`-Control verbunden. Dieses Element hat nichts direkt mit ADO.NET zu tun, sondern wird von ASP.NET bereitgestellt:

```
CustomerGrid.DataSource = objDS.Tables
CustomerGrid.DataBind()
```

In HTML sieht das dann folgendermaßen aus:

```
<asp:DataGrid id="CustomerGrid" Runat="server"/>
```

Das erzeugt übrigens keinen ActiveX- oder anderen browserseitigen Code, sondern eine einfache HTML-Tabelle. Der Name `DataGrid` führt zur gezeigten Anzeigeform Über den Parameter *`id`* erfolgt die Bindung an den VB-Code.

Die Übertragung der Daten von den ADO.NET-Objekten zu ASP.NET-Controls wird als *Data Binding* bezeichnet. Vom Ansatz her ist es eher eine Aufgabe, die ASP.NET zugeordnet werden kann.

Das letzte Skript zeigte nur Informationen über die Tabelle an. Die einzige Änderung ist bei der Übergabe der Daten nötig. Hier der Ausschnitt des modifizierten Skripts:

```
CustomerGrid.DataSource =
   objDS.Tables("Customers").DefaultView
```

Listing 10.5: ADO.DataSet.DataGrid.Data.aspx: Die Ausgabe der Daten benötigt nur eine minimale Modifikation (Code-Ausschnitt)

Hier wird nicht mehr auf eine Liste der Tabellen zugegriffen, sondern mit der Standardansicht auf die Tabelle *Customers*.

Das `DataGrid`-Steuerelement ist zwar offensichtlich zur Laufzeit gebunden, es erzeugt dennoch reinen HTML-Code. Das Skript läuft tatsächlich auch in NetScape- und Opera-Browsern und benötigt weder JavaScript noch irgendeine andere clientseitige Technologie. Es hat aber auch nichts mit ADO.NET zu tun, sondern stellt eines der spannendsten Features in ASP.NET dar. Mehr dazu und wie Sie die Tabelle ansprechend designen können, finden Sie in Kapitel 9.

DataGrid erzeugt reines HTML

Der HTML-Code in Abbildung 10.6 zeigt ohne jede spätere Änderung die Ausgabe des `DataGrid`-Elements: sauberes HTML 4.0 mit Style-Anweisungen. Das ist angesichts der bisherigen Microsoft-Politik, durchaus bemerkenswert.

Abbildung 10.5: Anzeige der Daten mit einer Standardansicht

ADO+

ADODataSetCommand

Bindung: DataGrid

Dieser Skript führt eine Datenbankabfrage aus und überführt die Daten in ein DataSet-Objekt. Dann wird dieses Objekt an ein ASP+-DataGrid gebunden.

CustomerID	CompanyName	ContactName	ContactTitle	Address	City	Region	PostalCode	Country	Phone	F
	ööö									
AAAAA										
ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representative	Obere Str. 57	Berlin		12209	Germany	030-0074321	0
ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Owner	Avda. de la Constitución 2222	México D.F.		05021	Mexico	(5) 555-4729	0
ANTON	Antonio Moreno Taquería	Antonio Moreno	Owner	Mataderos 2312	México D.F.		05023	Mexico	(5) 555-3932	
AROUT	Around the Horn	Thomas Hardy	Sales Representative	120 Hanover Sq.	London		WA1 1DP	UK	(171) 555-7788	0
BERGS	Berglunds snabbköp	Christina Berglund	Order Administrator	Berguvsvägen 8	Luleå		S-958 22	Sweden	0921-12 34 65	0
BLAUS	Blauer See Delikatessen	Hanna Moos	Sales Representative	Forsterstr. 57	Mannheim		68306	Germany	0621-08460	0
BLONP	Blondesddsl père et fils	Frédérique Citeaux	Marketing Manager	24, place Kléber	Strasbourg		67000	France	88.60.15.31	8
	Bööö									

Abbildung 10.6: Ausschnitt aus dem erzeugten HTML-Code

```
<TABLE
style="BORDER-RIGHT: 1px solid; BORDER-TOP: 1px solid; BORDER-LEFT: 1px solid; BORDER-BOTTOM: 1px solid"
rules=all border=1>
  <THEAD>
  <TR>
    <TD>CustomerID </TD>
    <TD>CompanyName </TD>
    <TD>ContactName </TD>
    <TD>ContactTitle </TD>
    <TD>Address </TD>
    <TD>City </TD>
    <TD>Region </TD>
    <TD>PostalCode </TD>
    <TD>Country </TD>
    <TD>Phone </TD>
    <TD>Fax </TD></TR>
  <TR>
    <TD>ANATR </TD>
    <TD>Ana Trujillo Emparedados y helados </TD>
    <TD>Ana Trujillo </TD>
    <TD>Owner </TD>
    <TD>Avda. de la Constitución 2222 </TD>
    <TD>México D.F. </TD>
    <TD>  </TD>
    <TD>05021 </TD>
    <TD>Mexico </TD>
    <TD>(5) 555-4729 </TD>
    <TD>(5) 555-3745 </TD></TR>
  <TR>
    <TD>ANTON </TD>
    <TD>Antonio Moreno Taquería </TD>
    <TD>Antonio Moreno </TD>
    <TD>Owner </TD>
    <TD>Mataderos 2312 </TD>
    <TD>México D.F. </TD>
    <TD>  </TD>
    <TD>05023 </TD>
    <TD>Mexico </TD>
    <TD>(5) 555-3932 </TD>
```

Eine Anmerkung noch zur Abfrage der Tabelle:

```
objDS.Tables("Customers").DefaultView
```

Hier wird auf das Objekt `DataTables` zugegriffen und dessen Methode `DefaultView` aufgerufen. `Tables` selbst liefert eine Kollektion – `DataSet` kann schließlich mehrere Tabellen enthalten. Es ist zu vermuten, dass die endgültigen Version hier noch Vereinfachungen bringt. Beispielsweise ist es sinnvoll, die Methode `DefaultView` als Standardmethode zu deklarieren und die Auswahl der Tabelle optional zu handhaben, wenn nur eine Tabelle existiert. In der Preview-Version funktionierte das aber nicht.

10.2.3 Wie es weitergeht

Mit `DataView` und `DataReader` können die Daten sehr flexibel transportiert werden. Leider war dieser Teil in der PreView-Version noch nicht so ausgereift, dass Skripte sinnvoll zum Laufen gebracht werden konnten. Schauen Sie in die Dokumentation der Beta-Version, um weitere Informationen zu erhalten. In künftigen Auflagen dieses Buchs werden Sie hier entsprechende Skripte finden.

DataView

Der erste Schritt besteht in einer einfachen ASPX-Seite, die Informationen über eine Tabelle enthält. In diesem und allen folgenden Beispielen wird wieder die Northwind-Datenbank verwendet, die schon unter ADO als Datenquelle diente. Auch der verwendete Provider ist – unverändert – der OLEDB-Treiber für den SQL Server 7.

Die Tabelle ausgeben: DataReader

Im nächsten Schritt interessieren natürlich die Daten in der Tabelle. Dafür gibt es den `DataReader`. Dieses Objekt erzeugt eine einfache, direkte Sicht auf die gewählte Tabelle – analog dem einfachsten `RecordSet`-Objekt mit Vorwärts-Zeiger in ADO.

10.2.4 Arbeit mit DataTable

DataTable erzeugt Tabellen

Das erste Objekt des `DataSet`-Objekts ist `DataTable`. Damit erstellen und verwalten Sie Tabellen. Von der Idee, Tabellen in SQL mit `CREATE TABLE` anzulegen, können Sie sich völlig lösen. Das `DataTable`-Objekt enthält eine Kollektion von Reihen, die so genannte `RowsCollection`. Diese Kollektion stellt den aktuellen Status der Daten bereit und ist für die Registrierung von Änderungen an den Daten verantwortlich.

Eine Tabelle erzeugen

Neue Tabellen erzeugen

`DataTable` kennt zwei Konstruktoren:

```
public DataTable()
public DataTable(string tableName)
```

Das folgende Beispiel zeigt, wie eine Tabellen *Kunden* angelegt wird:

```
Dim adoTable as DataTable
adoTable = New DataTable("Kunden")
adoTable.CaseSensitive = FALSE
adoTable.MinimumCapacity = 100
```

Diese Form der Übergabe des Namens der Tabelle an den Konstrukor ist nicht der einzige Weg. Es gibt eine Eigenschaft des Objekts, mit der sich der Name direkt erzeugen lässt:

```
adotable.TableName = "Kunden";
```

Zwei Eigenschaften wurden im Beispiel noch verwendet. `CaseSensitive` stellt sicher, wenn es auf `TRUE` gesetzt wurde, dass alle Zeichenkettenoperationen beim Sortieren, Filtern und Suchen unabhängig von Groß- und Kleinschreibung ablaufen.

Die zweite Eigenschaft, `MinimumCapacity`, reserviert eine Anzahl Speicherplätze und optimiert die Systemleistung. Die Angabe beinhaltet keine Beschränkung des Speicherplatzes,

Spalten der Tabelle hinzufügen

Spalten erzeugen

Das `DataTable`-Objekt enthält eine Kollektion von `DataColumn`-Objekten. Mit der Methode `Add` werden Spalten hinzugefügt. In VBScript sieht das folgendermaßen aus:

```
Dim workTable as DataTable
Dim workColumn as DataColumn
WorkTable = new DataTable("Customers")
workColumn = workTable.Columns.Add(
                        "CustID",
                        System.Type.GetType("System.Int32")
                        )
workColumn.AllowNull = FALSE
workColumn.Unique = TRUE
workColumn = workTable.Columns.Add(
                        "CustomerNameLast",
                        System.Type.GetType("System.String")
                        )
workColumn = workTable.Columns.Add(
                        "CustomerNameFirst",
                        System.Type.GetType("System.String")
```

```
                        )
workColumn = workTable.Columns.Add(
                        "Purchases",
                        System.Type.GetType("System.Double")
                        )
```

Im Beispiel wurden drei Spalten erzeugt. Die Methode `Add` benötigt den Namen der Spalte und den Datentyp. Der Zugriff erfolgt über eine Variable vom Typ `DataColumn`.

Dann wird eine Instanz der Tabelle erzeugt, welcher die Spalten zugeordnet werden sollen:

```
DataTable worktable = new DataTable("Customers");
```

Die Datentypen werden über eine weitere Systemfunktion erzeugt, hier am Beispiel des Datentyps 32-Bit-Integer:

```
System.Type.GetType("System.Int32");
```

Der Vorteil besteht in der Unabhängigkeit der Datentypen von der verwendeten Datenbank. Wenn Sie OLE DB-spezifische Typen verwenden, wird Ihre Applikation nicht transportierbar. `GetType` gibt null zurück, wenn der Datentyp nicht existiert.

Die korrekte Syntaxdarstellung der Methode `Add` ist:

```
Public DataColumn Add(String columnname, Type type)
Public DataColumn Add(String columnname)
```

Berechnungen mit Spalten

Aggregat-Funktionen

Aus SQL kennen Sie sicher die Aggregatfunktionen. Da ADO.NET den Datenbankzugriff abstrahiert, muss dafür ein Äquivalent geschaffen werden.

ADO.NET erlaubt berechnende Spalten für:

- Filtern
- Berechnen
- Zusammenstellen von Spalteninformationen (wie `COUNT` in SQL)

Um eine solche Spalte zu erstellen, wird die Eigenschaft `Expression` gesetzt:

```
Dim dc As DataColumn = New DataColumn
dc.DataType = System.Type.GetType("System.Currency")
dc.Expression = "total * 1.16"
```

Berechnende Spalten können Sie auch bei der Erstellung von Spalten definieren. Dazu wird die Syntax der `Add`-Methode erweitert:

```
Dim workColumn As DataColumn
workColumn = workTable.Columns.Add(
                        "Brutto",
                        System.Type.GetType("System.Double"),
                        "Netto * 1.16" )
```

Im letzten Beispiel wird eine neue Spalte *Brutto* erzeugt, die immer 16% mehr als die Spalte *Netto* enthält.

Spalten mit automatischen Werten

AutoIncrement

Typisch für SQL-Datenbanken sind Spalten, die eine eindeutige ID enthalten müssen. Damit das Programm mit der Erzeugung nicht belastet wird, übernimmt auch ADO.NET dies selbst. Dies erfolgt sehr einfach, indem die Eigenschaft `AutoIncrement` auf `TRUE` gesetzt wird. `AutoIncrementSeed` setzt den Startwert, `AutoIncrementStep` die Schrittweite. Beide Werte haben als Standardwert 1.

```
workColumn = workTable.Columns.Add _
   ("CustID", System.Type.GetType("System.Int32")
workColumn.AutoIncrement = TRUE
workColumn.AutoIncrementSeed = 10000
workColumn.AutoIncrementStep = 1
```

Nur-Lese-Spalten

Jede Spalte kann, wenn die Eigenschaft `ReadOnly` gesetzt wird, auch als Nur-Lese-Spalte deklariert werden:

```
workColumn.ReadOnly = TRUE;
```

Einen Primärschlüssel erstellen

Primärschlüssel

In jeder Tabelle muss es eine Spalte geben, die den betreffenden Datensatz eindeutig identifizieren kann. Üblich ist es, dazu die durch `AutoWert` geschaffene Spalte zu verwenden. Diese Spalte bekommt dann den Primärschlüssel zugeordnet. Der Spalte sollten Sie einen passenden Namen mit dem Suffix »ID« zu geben, beispielsweise *KundenID*. Dies erleichtert den Umgang mit den Namen im Code.

Für eine korrekte Definition muss außerdem sichergestellt werden, dass keine `Null`-Werte erlaubt sind:

```
workColumn.AllowNull = FALSE;
```

Im zweiten Schritt wird sichergestellt, dass die Werte eindeutig sind. Dies ist nur eine Regel und sagt nichts darüber, *wie* dies erfolgt. Sicherstellen können Sie das selbst oder mit `AutoIncrement`. Die Regel aktivieren Sie wie folgt:

```
workColumn.Unique = TRUE;
```

Jetzt kann die vorbereitete Spalte zum Primärschlüssel werden. Der folgende Code zeigt, wie das aussieht:

```
Dim myColArray As DataColumn()
myColArray = workTable.Columns("CustID")
workTable.PrimaryKey = myColArray
```

Ein Primärschlüssel lässt sich auch über mehrere Spalten gebildet werden. Die folgende Darstellung zeigt, wie dies erfolgt:

```
myColArray(0) = workTable.Columns("Col1")
myColArray(1) = workTable.Columns("Col2")
workTable.PrimaryKey = myColArray
```

Der Tabelle Daten hinzufügen

Daten in die Tabelle einfügen

Mit Spalten und einem Primärschlüssel können der Tabelle nun Daten hinzugefügt werden. Auch dieser Prozess ist abstrahiert und so gelangen `INSERT`-Anweisungen nicht mehr zum Einsatz.

```
Dim iCounter as integer
Dim workRow as DataRow
For i = 0 to 9
   workRow = worktable.NewRow()
   workRow("CustID") = iCounter
   workRow("CustomerNameLast") = "CustName" & i.ToString()
   worktable.Rows.Add(workRow)
Next
```

Hier wird ein neuer Typ, `DataRow`, eingeführt, um eine Reihe zu repräsentieren. Mit der Methode `NewRow` wird eine neue, leere Reihe erzeugt. Anschließend kann diese direkt mit Zuweisungsoperationen beschrieben werden. Der Zugriff auf Spalten erfolgt entweder mit den Spaltennamen (siehe das letzte VBScript-Beispiel) oder mit dem nummerischen Index. Selbstverständlich ist in beiden Sprachen auch der jeweils andere Weg möglich:

```
workRow("CustID") = "CustName" & i
workRow(1) = "CustName" & i
```

Wenn Ihnen der Weg über `NewRow` nicht praktikabel erscheint, können Sie auch hier die Methode `Add` anwenden:

```
workTable.Rows.Add(workRow)
```

Hinter der vollständigen Objektsyntax verbirgt sich eine weitere Schreibweise:

```
workTable.Rows.Add(new Object() {1, "CustName1"})
```

Status der Datenreihen

Status abfragen

Jedes `DataRow`-Objekt hat eine Eigenschaft `RowState`. Im letzten Beispiel wäre die Eigenschaft nach dem Erzeugen der Reihe mit `NewRow Detached`, nach dem Hinzufügen von Daten mit `Add` dagegen `New`.

Tabelle 10.1: Zustände der Eigenschaft RowState

RowState	Beschreibung
`Unchanged`	Seit dem letzten Aufruf von `AcceptChanges` sind keine Änderungen erfolgt.
`New`	Eine neue Reihe wurde hinzugefügt, aber `AcceptChanges` wurde noch nicht aufgerufen.
`Modified`	Ein oder mehrere Elemente der Datenreihe wurden verändert.
`Deleted`	Die Reihe wurde mit der Methode `Delete` gelöscht.
`Detached`	Entweder wurde die Reihe mit `Delete` gelöscht und mit `AcceptChanges` nicht aufgerufen oder die Reihe wurde erzeugt, der Tabelle aber noch nicht zugeordnet.

Die Abfrage der Eigenschaft erfolgt folgendermaßen:

```
Response.Write adoWorkRow.RowState
```

Daten löschen oder entfernen

Daten löschen

Wie schon beim Hinzufügen von Daten gezeigt, gibt es auch für das Löschen spezielle Methoden. Die Überschrift suggeriert übrigens zwei Formen von »Löschen«. Dies wird später noch diskutiert. Hier nur eine kurze Definition: »Entfernen« meint das physikalische Entfernen einer Reihe mit dem gesamten Inhalt. Beim »Löschen« wird die Reihe lediglich als gelöscht gekennzeichnet und erst durch einen zusätzlichen Befehl endgültig entfernt.

Die Methode `Remove` wird auf die Kollektion `RowsCollection` angewendet. Der folgende Code zeigt zwei äquivalente Varianten:

```
workTable.Rows.Remove(3)
workTable.Rows.Remove(workTable.Rows(3))
```

Ganz ähnlich sieht die Anwendung der Methode `Delete` aus:

```
workTable.Rows.Delete(3)
workTable.Rows.Delete(workTable.Rows(3))
```

Damit die so als gelöscht gekennzeichneten Reihen auch wirklich entfernt werden, muss zusätzlich die Methode `AcceptChanges` des Objekts `DataTable` aufgerufen werden. Dies ist möglicherweise effizienter, wenn Sie viele Löschvorgänge hintereinander ausführen müssen. Da der Löschvorgang erst später ausgeführt wird, haben Sie auch die Chance, diese Kennzeichnung mit `RejectChanges` wieder rückgängig zu machen. Diese Möglichkeit besteht bei `Remove` nicht. Ob eine Reihe als gelöscht gekennzeichnet wurde, können Sie mit `RowState` ermitteln. Die Eigenschaft gibt dann `"Deleted"` zurück.

Arbeiten mit Tabellendaten

Auslesen von Daten

Wenn Sie Daten in der Tabelle gespeichert haben, möchten Sie diese auch wieder auslesen. Das folgende Beispiel zeigt, wie dies erfolgt:

```
Dim CurrRows() As DataRow
     = workTable.Select(Nothing, Nothing,
                 System.Data.DataViewRowState.CurrentRows)
Dim list As String = System.String.Empty
Response.Write("<br>")
Response.Write("Ausgabe der aktuellen Datens&auml;tze "
                & workTable.Tablename)
Response.Write("<br>")
Dim RowNo As Integer
Dim ColNo As Integer
For RowNo = 0 to CurrRows.Count - 1
   For ColNo = 0 To workTable.Columns.Count - 1
      list  = ""
      list &= workTable.Columns(colNo).ColumnName & " = "
      list &= & CurrRows(RowNo)(ColNo).ToString
     Response.Write(list)
     Response.Write("<br>")
   Next
Next
If CurrRows.Count < 1 Then
   Response.Write("Keine aktuellen Datens&auml;tze gefunden")
End If
```

Im Beispiel wird die `Select`-Methode des Objekts `DataTable` verwendet. Diese Methode liest eine Anzahl Datensätze, die bestimmten Kriterien gehorchen. Zurückgegeben wird ein Array. Als Kriterium wird hier offensichtlich `CurrentRows` eingesetzt, die aktuellen Datensätze also. Aber was versteht man unter »aktuell«?

Aktuelle Datensätze und deren Zustandsformen

Das `DataTable`-Objekt kennt drei Versionen, die jede Datenreihe annehmen kann:

- *Original (original)*. Die originale Version entsteht, wenn die Daten der Tabelle das erste Mal hinzugefügt wurden. Immer wenn die Daten wieder diesen ursprünglichen Zustand annehmen, gilt die Reihe als original.
- *Aktuell (current)*. Diese Version ist der Zustand der Daten nach der letzten Änderung.
- *Vorgeschlagen (proposed)*. Diese Version existiert nur unter bestimmten Umständen für einen kurzen Zeitraum. Wird diese Version angezeigt, befindet sich das Objekt gerade im Prozess des Wechsels von *Original* zu *Aktuell*. Dies passiert, wenn die Methode `BeginEdit` aufgerufen wird, bis zum Abschluss des Editierens.

Das folgende Beispiel verdeutlicht diesen Vorgang:

```
workRow.BeginEdit
' Make some changes to the row.
If ValidateColValue(proposedValue)
   workRow.EndEdit
Else
   workRow.CancelEdit
End If
```

Während des Vorschlagszustands können Sie die einzutragenden Daten überprüfen und gegebenenfalls ablehnen. Dazu löst das `DataTable`-Objekt das `ColumnChange`-Ereignis aus. Der Programmteil, der das Ereignis bedient, kann dann entscheiden, ob `CancelEdit` aufgerufen wird und der Originalzustand wieder hergestellt wird. Wird dagegen `EndEdit` aufgerufen, wird der Vorschlagzustand beendet und die Daten befinden sich nun im aktualisierten Zustand.

10.2.5 Fehlerzustände bearbeiten

Eingabefehler abfangen

Wenn Nutzer Daten eingeben, können sie Fehler machen. Sie können die Prüflogik in ADO.NET implementieren, was unter Umständen einfacher ist als mit konventionellen Methoden. Denn nun müssen Sie Regeln für die Prüfung nur an einer einzigen Stelle erfassen und nicht bei jedem einzelnen Formular im gesamten Projekt. Dies ist mit ADO.NET besonders einfach, denn eine vernünftige Benutzerführung ist einfach zu implementieren. Normalerweise sollte ein Fehleingabe nicht zum Abbruch des Programms führen, sondern am Eingabefeld eine sinnvolle Fehlermeldung anzeigen.

Das folgende Beispiel zeigt die prinzipielle Vorgehensweise:

```
Public Sub AddErrorToRow(ByVal workTable As DataTable, _
                         ByVal Row As Integer, _
                         ByVal ErrorString As String)
```

```
   workTable.Rows(Row).RowError = ErrorString
End Sub
```

Fehler werden also, wenn sie auftreten, lediglich der Eigenschaft `RowError` zugewiesen.

```
Public Sub WriteTableErrors(ByVal workTable As DataTable)
   If workTable.HasErrors Then
      Dim ErrorRows() As DataRow = workTable.GetErrors()
      Response.Write("<br>")
      Response.Write("Tabelle " & workTable.TableName
                              & " hat ")
      Response.Write(ErrorRows.Count.ToString & " Fehler!")
      Dim i As Integer
      For i = 0 to Errors.Count - 1
         Response.Write("Fehler in Reihe " & _
                        ErrorRows(i)("CustID").ToString &  _
                        "Error Msg =" & _
                        ErrorRows(i).RowError)
      Next
   Else
   Response.Write("<br>")
   Response.Write("Tabelle " + workTable.TableName +
   " Hat keine Fehler ")
   End If
End Sub
```

Das zweite Beispiel gibt die zuvor registrierten Fehler aus.

Akzeptieren oder Zurückweisen von Änderungen

Änderungen zurückweisen

Auf die Methode `AcceptChanges` wurde bereits kurz eingegangen. Intern führt die Ausführung dazu, dass der Status der Datenreihe von *Vorgeschlagen* auf *Original* wechselt. Der neue Wert wird nun zum Originalwert. Das ist zwar schon ein Schritt zur Speicherung der Daten, alle bisher diskutierten Prozesse spielten sich aber nur in Bezug auf das im Hauptspeicher gehaltene `DataSet`-Objekt ab. Wichtiger wäre jedoch festzustellen, ob die Daten auch von der darunter liegenden Datenbank physikalisch gespeichert wurden.

Zuerst der Code zum Gültigmachen der endgültigen Daten im `DataTable`-Objekt, also für alle Änderungen an der assoziierten Tabelle:

```
workTable.AcceptChanges()
```

HasErrors

Etwas umfangreicher sieht die zweite Variante aus. Hier läuft eine Schleife durch alle geänderten Datensätze und wertet mit Hilfe der Eigenschaft

HasErrors aus, ob Fehler registriert wurden. Ist das der Fall, werden die Daten nicht gültig gemacht.

```
Public Function AcceptChanges(ByVal workTable As DataTable)
                As Integer
   Dim acceptedRows As Integer = 0
   Dim i As Integer
   For i = 0 To workTable.Rows.Count - 1
      If Not workTable.Rows(i).HasErrors Then
         acceptedRows += 1
         workTable.Rows(i).AcceptChanges()
      Else
         workTable.Rows(i).RejectChanges()
      End If
   Next
   AcceptChanges = acceptedRows
End Function
```

Die entscheidende Abfrage der Eigenschaft HasErrors auf einen Blick:

```
If Not workTable.Rows(i).HasErrors Then
```

10.2.6 Daten in einer Tabelle filtern und sortieren

Daten filtern

Die Select-Methode wurde bereits kurz vorgestellt. Sie erlaubt die Abfrage der Daten auf drei Wegen: mit einem Filter-Ausdruck, in einer bestimmten Sortierung oder mit DataViewRowState. Die drei Wege sind gleichzeitig anwendbar und werden durch drei Parameter spezifiziert. Wenn ein Weg nicht interessant ist, wird der Parameter auf null (in VBScript heißt das Nothing) gesetzt.

```
Dim CurrRows() As DataRow
        = workTable.Select(Nothing, Nothing,
        System.Data.DataViewRowState.CurrentRows)
```

Das folgende Beispiel sucht in der Kundentabelle alle Kunden mit dem Nachnamen Krause und sortiert die Ausgabe nach dem Vornamen:

```
Dim myNames() As DataRow = workTable.Select(
                           "CustomerLastName = 'Smith'"), _
                           "CustomerNameFirst",
                           System.Data.DataViewRowState.CurrentRows)
```

10.3 Arbeiten mit DataSet-Objekten

Der Umgang mit `DataSet`-Objekten ist in der Praxis weitaus komplexer als im Einführungsabschnitt beschrieben. Die wichtigsten Eigenschaften und Methoden sollen deshalb hier tiefergehend behandelt werden.

10.3.1 DataSet im Detail

Das `DataSet`-Objekt repräsentiert die Daten im Speicher. Es enthält alle Informationen über Tabellen und deren Struktur und Beziehungen. Das Objekt ist außerdem in der Lage, den Datenaustausch in beide Richtung über XML zu initiieren.

Prinzip der Datenanforderung vom Client

Eine typische Szene aus dem Inneren des zentralen ADO.NET-Objekts könnte folgendermaßen aussehen: Ein Client fordert per URL Daten von einer Applikation auf einem Webserver an. `DataSet` führt die Abfrage dann aus, holt die Daten aus der Datenquelle und erzeugt – »on the fly« – ein XML-Dokument mit der Beschreibung der Daten. Dieses Dokument wird zurückgesendet. Für die Darstellung ist der Client selbst verantwortlich. Üblicherweise nutzt man, wenn der Internet Explorer verwendet wird, ein ActiveX-Steuerelement, das die Darstellung übernimmt: `DataGrid`. Damit kann der Nutzer die Daten ansehen, löschen, ändern und wieder zurücksenden. Die erneute Konvertierung der Daten übernimmt das Steuerelement, denn das `DataSet`-Objekt erwartet die Daten wieder im XML-Format zurück. Die gesamte Kommunikation läuft per HTTP-Requests ab. `DataSet` übernimmt auch das korrekte Einsortieren der Daten in die Datenquelle. Wie im Fall von Konflikten verfahren wird, lässt sich leicht programmieren.

Mehrere Wege zum Ziel

Bei der Vielzahl von Aufgaben gibt es erwartungsgemäß mehrere Wege, zu einem funktionierenden `DataSet`-Objekt zu gelangen. Sie können zum einen einen programmatischen Weg nehmen und ein `DataSet`-Objekt erzeugen, darin `DataTable`-Kollektionen usw., wie es am Anfang des Kapitels bereits beschrieben wurde. Dieser Weg wird auch dann beschritten, wenn Sie das Objekt erst zur Laufzeit erzeugen können, beispielsweise weil die Struktur vorher nicht klar ist.

Der andere Weg besteht im Anlegen der Datenquelle. Dabei erzeugen Sie Tabellen mit allen benötigten Eigenschaften im SQL Server-Manager. Der Zugriff auf die fertigen Strukturen erfolgt dann mit `SQLDataSetAdapter`. Dieser Weg wird der häufiger benötigte sein und soll in den nächsten Abschnitten genauer dargestellt werden.

10.3.2 DataSet mit existierenden Datenstrukturen

Datenstrukturen aus SQL Server nutzen

Gehen Sie in der folgenden Reihenfolge vor, um `DataSet` mit existierenden Strukturen und Daten zu verwenden:

1. Erzeugen Sie Tabellen und holen Sie Daten mit `SQLDataSetCommand` oder `ADODataSetCommand`.

2. Bearbeiten Sie die Daten direkt in den `DataTable`-Objekten.
3. Führen Sie die Methode `GetChanges` aus, um ein zweites `DataSet`-Objekt zu erzeugen, das nur die Änderungen der Daten reflektiert. Das folgende Beispiel zeigt das:

```
Dim changedDataSet As DataSet
changedDataSet = ds.GetChanges(DataRowState.Modified)
```

4. Prüfen Sie nun im zweiten `DataSet`-Objekt, ob Fehler aufgetreten sind, indem die Eigenschaft `HasErrors` geprüft wird. Mit Hilfe der Methode `GetErrors` können außerdem alle Reihen mit Fehlern in ein Array exportiert werden.
5. Für jede Datenreihe mit Fehlern kann die Eigenschaft `RowError` ermittelt werden, in der eine nähere Beschreibung des Fehlers steht. Führen Sie notwendige Aktionen entsprechend den Fehlermeldungen aus.
6. Sind keine Fehler mehr zu finden, überführen Sie die gültigen Daten wieder zurück in das erste `DataSet`-Objekt:

```
ds.Merge(changedDataSet)
```

7. Rufen Sie nun die Methode `Update` auf, um die Daten, die das erste `DataSet`-Objekt enthält, an `SQLDataSetCommand` oder ADODataSetCommand zu übergeben. Als Parameter wird das `DataSet`-Objekt benutzt:

```
workDSCMD.Update(ds)
```

8. Machen Sie die Daten mit `AcceptChanges` gültig oder, falls es notwendig ist, machen Sie alle Änderungen mit `RejectChanges` rückgängig:

```
ds.AcceptChanges
```

10.3.3 Ein DataSet-Objekt zur Laufzeit erzeugen

Strukturen zur Laufzeit erzeugen

Das `DataSet`-Objekt kennt die zwei schon bekannten Konstruktoren, die auch hier wieder zum Einsatz gelangen:

```
public DataSet()
public DataSet(string DataSetName)
```

Instanzen der Objekte werden folgendermaßen angelegt:

```
Dim workDataSet as DataSet
```

Als leeres `DataSet` zum späteren Aufbau:

```
Set workDataSet As New DataSet()
```

Eine andere Varianten, der gleich eine Tabelle zugewiesen wird:

```
Set workDataSet As New DataSet("KundBestell")
```

Erzeugen der Tabellenstruktur

Erzeugen von Tabellen

Dieser Vorgang wurde bereits in Abschnitt 10.2.3 DataView ab Seite 350 diskutiert. Das folgende Beispiel zeigt dies anhand eines praktischen Projekts:

```
ds.Tables.Add(New DataTable("Bestell"))
ds.Tables("Bestell").Columns.Add("BestellID",
System.Type.GetType("System.Int32"))
ds.Tables("Bestell").Columns.Add("BestellMenge", _
            System.Type.GetType("System.Int32"))
ds.Tables("Bestell").Columns.Add("CustID", _
            System.Type.GetType("System.Int32"))
ds.Tables("Bestell").PrimaryKey =
           New DataColumn(ds.Tables.Columns("Bestell"),
             ds.Tables.Columns("BestellID"))
```

Ein alternativer Weg könnte folgendermaßen aussehen:

```
Dim workTable As DataTable = New DataTable("Bestell")
workTable.Columns.Add("BestellID",
  System.Type.GetType("System.Int32"))
workTable.Columns.Add("BestellMenge",
  System.Type.GetType("System.Int32"))
workTable.Columns.Add("KundenID",
  System.Type.GetType("System.Int32"))
ds.Tables("Bestell").PrimaryKey =
   New DataColumn(ds.Tables("Orders").Columns("BestellID"))
ds.Tables.Add(workTable)
```

Beziehungen zwischen Tabellen erzeugen

Da ein `DataSet`-Objekt mehrere Tabellen enthalten kann, können Sie auch Beziehungen zwischen den Tabellen definieren. Beziehungen werden benötigt, um normalisierte Tabellen miteinander zu verbinden.

DataRelation Relations

Die entsprechende Methode heißt folgerichtig `DataRelation`. Benötigt werden zwei Argumente: der Primärschlüssel der ersten Tabelle und der Fremdschlüssel (Foreign Key) der zweiten Tabelle. Da beide Schlüssel eindeutig sein müssen, sollten Sie nur solche Fremdschlüssel verwenden, die in der zweiten Tabelle selbst Primärschlüssel sind. Die Beziehung bekommt noch einen eindeutigen Namen, auf den später bei Abfragen Bezug genommen werden kann.

Beziehungen können auch zwischen mehreren Spalten einer Tabelle erzeugt werden. Beide Parameter erlauben deshalb als Argument auch ein Array aus `DataColumn`-Objekten.

Das folgende Beispiel zeigt, wie eine einfache Beziehung zwischen zwei Tabellen erstellt wird. Beide Tabellen besitzen eine Spalte *KundenID*, die als Primärschlüssel verwendet wird.

```
Dim ds As DataSet = New DataSet("KundBestell")
ds.Relations.Add("KundBestell",
                 ds.Tables("Kunden").Columns("KundenID"), _
                 ds.Tables("Bestell").Columns("KundenID"))
```

Diese Variante nutzt die Methode `Relations.Add` zum Hinzufügen der Beziehung. Kürzer aber eventuell schlechter zu lesen ist die folgende Version:

```
Dim dr As DataRelation = New DataRelation("KundBestell",
                    ds.Tables("Kunden").Columns("KundenID"),
                    ds.Tables("Bestell").Columns("KundenID"))
ds.Relations.Add(dr)
```

Beziehungen zwischen Tabellen anwenden

Anwendung der Beziehungs-Objekte

Nach dem Erstellen der Beziehungen ist zu klären, wie damit bei Abfragen der Daten umgegangen wird. In der Praxis geht es fast immer darum, bei einer Abfrage einer Tabelle auch die durch die Beziehung verbundenen Daten einer anderen Tabelle zu holen und gemeinsam auszugeben. Im E-Commerce ist ein häufiger Fall die Anzeige des Warenkorbs. Tatsächlich existiert nur eine Tabelle für alle Artikel aller Kunden, die derzeit in einem Warenkorb liegen. Wenn Sie die Kundendaten abrufen, wird Ihnen die Kundennummer bekannt sein. Dann können Sie die Daten des Kunden zum Erstellen der Rechnung zusammen mit dem Inhalt seines Warenkorbs abrufen. Das ist eine so genannte 1:n-Beziehung: Die *KundenID* ist in der Tabelle *Customers* eindeutig, in der Tabelle *Orders* jedoch nicht (weil ein Kunde ja mehrere Bestellungen getätigt haben kann).

Das folgende Beispiel zeigt dies: Zurückgegeben wird ein Array mit einem Element für die Kundendaten und mehreren Elementen für Bestellungen:

```
Dim x() As DataRow =
        ds.Tables("Kunden").ChildRelations("Kundbestell")_
        .GetChildRows(ds.Tables("Kunden").Rows(0))
Console.WriteLine("")
Console.WriteLine("Anzahl untergeordneter Einträge für
                  CustOrders Beziehungen = "
                  & x.Count.ToString)
Console.WriteLine("Anzahl Tabellen im DataSet = " &
                  ds.Tables.Count.ToString)
```

10.3.4 Einschränkungen definieren

Constraints: Einschränkungen in ADO.NET

In einer relationalen Datenbank ist es sehr wichtig, ständig über integre Daten zu verfügen. Auf einfache Weise lässt sich das erledigen, indem der gesamte Eingabedatenstrom kontrolliert wird. Möglicherweise wird das die Applikation aufblähen. Ändern sich die Bedingungen, werden vielleicht auch viele Eingriffe notwendig. Besser ist es, diese Aufgabe in der Datenbank selbst erledigen zu lassen. Auf die Möglichkeit, Fehler zu erzeugen und Reihen zuzuordnen, wurde bereits eingegangen. Dies ist ideal für die Kombination mit der Kontrolle der Daten. Diese Definitionen von Einschränkungen sind unter dem Namen »Constraints« bekannt. Zwei einfache Einschränkungen beziehen sich auf die bereits besprochenen Schlüssel. Sie sichern die korrekte Arbeitsweise der Beziehungen.

ForeignKey Constraint UniqueConstraint

ADO.NET kennt zwei Methoden für die Definition solcher Einschränkungen: `ForeignKeyConstraint` und `UniqueConstraint`.

ForeignKey-Constraint

Kontrolle der Fremdschlüssel

Wird ein Wert in einer Reihe gelöscht oder geändert und dieser Wert kommt in mehreren Tabellen vor, so ist die Integrität der Datenbank gefährdet, wenn diese Änderungen nicht in allen Tabellen erfolgen. Auch dies kann man programmtechnisch sicherstellen. Das ist aufwändig und riskant, da Sie viele Fälle bedenken müssen. Eine so genannte Fremdschlüssel-Einschränkung ist der bessere Weg. Hier definieren Sie, was passiert, wenn einer der Schlüssel entfernt wird:

- `Cascade`. Die Änderung wird auf die verbundenen Schlüssel übertragen. Wird ein Schlüssel gelöscht, werden auch die verbundenen entfernt usw.
- `SetNull`. Die Werte der verbundenen Reihen werden auf `Null` gesetzt.
- `SetDefault`. Die Werte der verbundenen Reihen werden auf die Standardwerte gesetzt, soweit dort welche definiert wurden.
- `None`. Keine Aktion, die verbundenen Reihen werden nicht bearbeitet.
- `Default`. Standardaktion, das ist normalerweise `Cascade`.

Um diese Einschränkung zu definieren, wird sie zuerst erzeugt und dann werden die Eigenschaften für die Vorgänge Löschen (`DeleteRule`) und Ändern (`UpdateRule`) gesetzt.

```
Dim fk As ForeignKeyConstraint
fk = New
ForeignKeyConstraint(ds.Tables("Kunden").Columns("KundenID"),
          ds.Tables("Bestellungen").Columns("KundenID"))
fk.DeleteRule = Cascade
fk.UpdateRule = SetDefault
' Jetzt wird die Einschränkung der Tabelle hinzugefügt
ds.Tables(0).Constraints("Customers").Constraints.Add(fk)
```

Unique-Constraint

Kontrolle auf eindeutige Datensätze

Auf die Bedeutung des Primärschlüssels wurde bereits eingegangen. Dieser Schlüssel muss eindeutig sein, er stellt das primäre Auswahlkriterium dar. Mit Hilfe einer Eindeutigkeits-Einschränkung teilen Sie dem Datenbankprovider mit, dass doppelte Werte nicht zulässig sind. Tritt dieser Fall ein, wird die Eintragung der Datenreihe verhindert. Die Einschränkung verhindert aber nicht, dass bei Löschvorgängen Lücken auftreten.

Sie können die Eigenschaft `Unique` einer oder mehreren Spalten zuordnen; die Namen mehrerer Spalten werden als Array übergeben.

```
Dim uc As UniqueConstraint
uc = New
   UniqueConstraint(ds.Tables("Kunden").Columns.("KundenID"))
' Jetzt wird die Einschränkung hinzugefügt:
ds.Tables("Kunden").Constraints.Add(uc)
```

10.3.5 Typsichere DataSet-Objekte

Kontrolle der Datentypen

Ein typisiertes `DataSet`-Objekt wird direkt von `DataSet` abgeleitet. Damit findet die Prüfung von Datentypen schon im Objekt statt und nicht erst in der Datenbank selbst. Auf der anderen Seite können Sie mit Variablen auf die Daten zugreifen, deren Datentyp Sie kennen und die mit benutzerfreundlichen Namen ausgestattet sind, unabhängig vom Namen in der Tabelle.

An dieser Stelle ist ein Vergleich mit ADO angebracht. Das folgende Beispiel zeigt, wie Sie mit ADO einen Datensatz selektieren und den Inhalt dann ändern:

```
AdoRecordSet.Find("KundenID = '12683'")
AdoRecordSet.Fields("KundenName").Value = "Joerg"
```

In ADO.NET schreiben Sie dieselbe Aufgabe direkter:

```
CustomerDataSet.Kunden("12683").KundenName = "Joerg"
```

Die Technik hat nicht nur den Vorteil, dass sie weniger Tipparbeit erfordert. Der Editor in Visual Studio 7 kann auch Zeilen automatisch komplettieren.

ADO.NET arbeitet in einer Umgebung, in der Quellcodes kompiliert werden. Es ist wichtig für den Compiler, die Prüfung der Datentypen bereits beim Übersetzen festzustellen. ADO konnte auf Typfehler erst zur Laufzeit reagieren. Dies erschwert auch den Textprozess. Wenn Sie in ADO.NET typisierte Variablen verwenden, wird der Fehler schon beim Übersetzen angezeigt. Umgekehrt ist ein fehlerfrei übersetztes Programm in Bezug auf die Verwendung der Datentypen laufzeitstabiler.

Erstellen typisierter Datenschemas

Datenschemata können Sie im XSD-Standard generieren, was die Deklaration erleichtert. XSD ist ein XML-Derivat, die so genannte »XML Schema Definition« Language. Damit werden Datenschemata für Datenbankmodelle beschrieben. Ausführliche Informationen finden Sie unter der folgenden Adresse:

- *http://www.w3.org/XML/Schema*

xsd.exe

Wenn Sie ein Datenmodell aufsetzen möchten und Sie haben eine XSD-Datei, ist die Erzeugung des `DataSet`-Objekts sehr einfach. Verwenden Sie das Werkzeug XSD.EXE dafür:

```
xsd.exe /d /l:C# {XSDSchemaFileName.xsd}
```

Die Parameter haben folgende Bedeutung:

- `/d`: Direktive zur Erzeugung des DataSet
- `/l:C#`: Erstellt die Anweisungen in C#-Syntax
- `/l:VB`: Erstellt die Anweisungen in VB-Syntax

Der erzeugte Code wird in einer Klasse verpackt, bei VB7 ist dies eine Datei mit der Endung .CLS und demselben Namen wie die xsd-Datei. Hier eine Musterdatei mit einer einfachen Definition:

```
Steuerung des Datenprov<?xml version="1.0"?>
<schema
 targetNamespace="http://example.com/stockquote/schemas"
 xmlns="http://www.w3.org/1999/XMLSchema">
 <element name="GetLastTradePrice">
   <complexType>
      <all>
         <element name="tickerSymbol" type="string"/>
      </all>
   </complexType>
 </element>
 <element name="GetLastTradePriceResult">
    <complexType>
       <all>
          <element name="result" type="float"/>
       </all>
    </complexType>
 </element>
</schema>
```

Listing 10.6: stock.xsd zum Erzeugen einer DataSet-Klasse

Das Ergebnis der Transformation ist eine umfangreiche Klassendatei mit mehreren Klassendefinitionen, die hier nur ausschnittsweise wiedergegeben werden kann (nur die Definition der Basisklasse stock):

```
Imports System
Imports System.Data
Imports System.Core

Namespace stock
   Public Class stock
      Inherits DataSet
      Private tableCount As Integer
      Private relationCount As Integer
      Private tableGetLastTradePrice As GetLastTradePrice
      Private tableGetLastTradePriceResult
              As GetLastTradePriceResult

      Public Sub New()
          MyBase.New
          Me.InitClass
      End Sub

      Public ReadOnly Property _
      <System.ComponentModel.PersistContentsAttribute(TRUE)> _
      GetLastTradePrice As GetLastTradePrice
        Get
          return Me.tableGetLastTradePrice
        End Get
      End Property
      Public ReadOnly Property _
      <System.ComponentModel.PersistContentsAttribute(TRUE)> _
      GetLastTradePriceResult As GetLastTradePriceResult
        Get
          return Me.tableGetLastTradePriceResult
        End Get
      End Property

      Private Sub InitClass()
          Me.DataSetName = "stock"
          Me.Namespace =
                      "http://example.com/stockquote/schemas"
          Me.tableCount = 2
          Me.relationCount = 0
          Me.tableGetLastTradePrice =
             New GetLastTradePrice("GetLastTradePrice")
          Me.Tables.Add(Me.tableGetLastTradePrice)
```

```
            Me.tableGetLastTradePriceResult =
        New GetLastTradePriceResult("GetLastTradePriceResult")
            Me.Tables.Add(Me.tableGetLastTradePriceResult)
        End Sub

        '... einige Hilfsfunktionen

        Public Overrides Sub ResetRelations()
            Dim i As Integer = 0
            Do While ((i) < (Me.Relations.Count))
                Me.Relations.Remove(Me.Relations(i))
                i = ((i) + (1))
            Loop
        End Sub

    End Class
    ' ... es folgen Klassendefinitionen für jedes Element
End Namespace
```

Listing 10.7: Kleiner Ausschnitt aus der erzeugten Klassen-Datei stock.cls.

10.4 Steuerung des Datenproviders

Zugriff auf den Provider

Die Objekte `Command`, `Connection` und `DataReader` sind Kernelemente des ADO.NET-Objektmodells. `Connection` kennt zwei Objekte: `SQLConnection` und `ADOConnection`. Ersteres dient der Verbindung mit dem SQL Server 7 oder SQL Server 2000, das andere für alle Datenbankmanagementsystems mit OLEDB-Treiber.

10.4.1 Verbindungsmanagement

Verbindungen

Mit `ADOConnection` kann eine Verbindung zu einem OLEDB-Provider aufgebaut werden, beispielsweise MS Access. Mit `SQLConnection` wird dagegen der SQL Server direkt angesprochen. Um die Verbindung zu initiieren, wird ein entsprechender Namensraum geöffnet:

- Der »SQL Managed Provider« mit `System.Data.SQL`
- Der »ADO Managed Provider« mit `System.Data.ADO`

SQLConnection

SQL Connection

Der folgende Code zeigt, wie Sie eine direkte Verbindung zu einem SQL Server initiieren:

```
Dim connectionString As String = _
            "server=localhost;uid=sa;pwd=;database=northwind"
```

```
Dim myConnection As SQLConnection = New
SQLConnection(connectionString)
myConnection.Open()
```

Als Zeichenkette wird eine Folge von Parametern mit Attributen eingesetzt. Die möglichen Varianten können Sie der folgenden Tabelle entnehmen:

Tabelle 10.2: Parameter der Verbindungszeichenfolge

Name	Synonym	Standard	Beschreibung
App	"Application Name"	Keiner	Name der Applikation
AttachDBFilename	"extended properties" "Initial File Name"	Keiner	Name der Datei einer angehängten Datenbank
Timeout	"Connection Timeout"	15	Wartezeit auf den Server
Database	"Initial Catalog"	Keiner	Datenbankname
Isolation Level	<none>	Read-Committed	Eines der folgenden Transaktionslevel: • ReadCommitted • ReadUncommitted • RepeatableRead • Serializable
Language	"Current Language"	Keiner	Sprache
Network	"Network Library", "Net"	dbmssocn	Verwendete Bibliothek: • dbmssocn • dbnmpntw • dbmsrpcn • dbmsvinn • dbmsadsn • dbmsspxn
Password	"Pwd"	Keiner	Kennwort
	"Data Source", "Address", "Addr", "Network Address"	Keiner	Name oder Netzwerkadresse des SQL Servers
Trusted_Connection	"Integrated Security"	no	Zulässige Werte sind yes oder no. Der Wert sspi entspricht yes.
Uid	"User id"	Keiner	SQL Server-Anmeldename

Name	Synonym	Standard	Beschreibung
Wsid	"Workstation Id"	Computer-name	Name der Workstation
PersistSecu-rityInfo	"Persist Security Info"	no	Wenn no, werden Kennwörter nicht wieder zurückgesetzt.

ADOConnection

Der folgende Code zeigt, wie Sie eine direkte Verbindung zu einem SQL Server vie OLEDB initiieren. Als Zeichenkette für den Provider können Sie die bei ADO in Abschnitt 2.2.2 Verbindungszeichenfolgen ab Seite 40 gezeigten Versionen einsetzen:

```
Dim connString As String =
         "Provider= SQLOLEDB.1; Data " &
         "Source=localhost; uid=sa; pwd=; " &
         "Initial Catalog=northwind;"
Dim myConn As ADOConnection = New ADOConnection(connString)
myConn.Open()
```

Die verwendbaren Methoden sind mit denen des ADO-Objekts Connection weitgehend identisch.

10.4.2 Kommandomanagement

Kommandos absetzen

Besteht die Verbindung zum Datenbankmanagementsystem, können Kommandos ausgeführt werden. Der direkteste Weg nutzt das schon aus ADO bekannte Command-Objekt. Die folgenden beiden Beispiele zeigen den Zugriff für ADO und SQL.

ADOCommand

```
Dim SQlStmt As String = "SELECT * FROM Customers"
Dim myCommand As ADOCommand = New
              ADOCommand(SQLStmt, myConnection)
```

SQLCommand

```
Dim SQlStmt As String = "SELECT * FROM Customers"
Dim myCommand As SQLCommand = New
                              SQLCommand(SQLStmt, myConnection)
```

Kommandos geben in vielen Fällen Daten zurück. Mit einem weiteren Objekt vom Typ ADODataReader werden diese Daten aufgenommen:

ADO

```
Dim myReader As ADODataReader = Nothing
myCommand.Execute(myReader)
```

SQL

```
Dim myReader As SQLDataReader = Nothing
myCommand.Execute(myReader)
```

Mit der Bedeutung des Objekts `DataReader` und dem Umgang damit befasst sich der nächste Abschnitt.

10.4.3 DataReader

Methoden zum Lesen großer Datenmengen

Wenn große Datenmengen gelesen werden, ist der zur Verfügung stehende Speicherplatz im Hauptspeicher ein entscheidender Faktor für die Gesamtleistung des Systems. Wenn Ihr Programm aus einer Tabelle 10 000 Reihen liest und dieses Programm auf einem Webserver von Hunderten Nutzern gleichzeitig aufgerufen wird, ist die Art und Weise der Verwendung von Speicherressourcen extrem wichtig. Dieses Verhalten ist aber für Objekte des Typs `DataTable` typisch. Es gibt viele Fälle, in denen die genaue Anzahl der gelesenen Datensätze nicht vorhergesagt werden kann. `DataTable` versucht immer, den gesamten Datenstrom zu lesen, egal wie groß er ist. Letztendlich wird der Hauptspeicher nicht ausreichend sein und Windows 2000 beginnt mit dem Auslagern auf die Festspeichermedien. Spätestens an dieser Stelle bricht die Systemleistung drastisch zusammen.

Leistungsvorteile durch DataReader

Das Objekt `DataReader` verhält sich anders. Es baut eine direkte Verbindung zur Datenbank auf und hält immer nur einen Datensatz im Speicher. Die Datenverbindung ist eine Nur-Lese-Verbindung und der Datenbankzeiger kann nur vorwärts laufen. Einen solchen »Firehose-Cursor« kannte bereits ADO. Wie die Anwendung in ADO.NET erfolgt, zeigt der folgende Code:

```
While myReader.Read
   ' Do something with the current row
End While
```

`DataReader` kennt mehrere »Get«-Methoden, um bestimmte Datentypen direkt zu lesen: `GetDataTime`, `GetDouble`, `GetGuid`, `GetInt32`, `GetStream` und andere.

10.4.4 Umgang mit gespeicherten Prozeduren

Abruf und Nutzung gespeicherter Prozeduren

Wenn Sie eine Abfrage so gestalten, dass mit Sicherheit nur eine Reihe zurückgegeben wird (oder im Extremfall nur ein skalarer Wert), sind gespeicherte Prozeduren besonders effektiv. Prozeduren können Sie nur mit SQL Server verwenden, nicht mit MS Access. Die Ansteuerung ähnelt der in ADO. Auch hier gibt es eine `Parameter`-Kollektion, um Parameter an die Prozedur zu übergeben. Die Zuordnung der Parameter erfolgt bei ADO.NET nicht in der Reihenfolge der Angabe, sondern durch Vergleich der Namen.

```
Dim myConnection As SQLConnection = New SQLConnection _
   ("server=delphi;uid=sa;pwd=;database=northwind")
Dim myCommand As SQLCOmmand = New SQLCommand _
("GetCustomerListbyState 'WA'", myConnection)
myCommand.CommandType = CommandType.StoredProcedure
Try
   myConnection.Open()
   myCommand.Execute(myReader)
   While myReader.Read
      Console.WriteLine(myReader("CompanyName").ToString))
   End While
   Catch e As Exception
      Console.WriteLine(e.ToString)
   Finally
      myReader.Close()
      myConnection.Close()
End Try
```

Im folgenden Beispiel wird eine gespeicherte Prozedur mit dem Namen *GetComanyName* aufgerufen. In SQL wurde sie folgendermaßen definiert:

```
CREATE PROCEDURE GetCompanyName
@CustomerID nvarchar(5),
@CompanyName nvarchar(40) output
as
SELECT @CompanyName = CompanyName FROM Customers
   WHERE CustomerID = @CustomerID
```

Die nächsten beiden Beispiele zeigen, wie dies mit den Namensräumen SQL und ADO realisiert wird:

SQL

```
Dim myConnection As SQLConnection = new _
SQLConnection("server=delphi;uid=sa;pwd=;database=northwind")
Dim myCommand As SQLCommand = New SQLCommand("GetCompanyName",
myConnection)
myCommand.CommandType = CommandType.StoredProcedure
Dim workPara As SQLParameter = Nothing
workParam = myCommand.Parameters.Add
 (New SQLParameter("@CustomerID", SQLDataType.NChar, 5))
workParam.Direction = ParameterDirection.Input
workParam.Value = "ALFKI"
workParam.myCommand.Parameters.Add
(New SQLParameter("@CompanyName", SQLDataType.NChar, 40))
workParam.Direction = ParameterDirection.Output
Try
   myConnection.Open()
```

```
   myConnection.Execute()
   Console.WriteLine("CompanyName = " & _
   myCommand.Parameters("@CompanyName").Value
Catch e As Exception
   Console.WriteLine(e.ToString)
Finally
   myConnection.Close()
End Try
```

ADO

```
Dim myCommand As ADOCommand = New ADOCommand("GetCompanyName",
myConnection)
myCommand.CommandType = CommandType.StoredProcedure
Dim workPara As ADOParameter = Nothing
workParam = myCommand.Parameters.Add("@CustomerID",
SQLDataType.NChar, 5)
workPara.Direction = ParameterDirection.Input
workParam.Value = "ALFKI"
workParam.myCommand.Parameters.Add("@CompanyName", SQLDataType.NChar,
40)
workParam.Direction = ParameterDirection.Output
Try
   myConnection.Open()
   myConnection.Execute()
   Console.WriteLine("CompanyName = " & _
   myCommand.Parameters("@CompanyName").Value
Catch e As Exception
   Console.WriteLine(e.ToString)
Finally
   myConnection.Close()
End Try
```

Jetzt wird der Name zugewiesen:

```
ADOCommand myCommand = new ADOCommand("GetCompanyName",
                                      myConnection);
myCommand.CommandType = CommandType.StoredProcedure;
```

Die anderen Optionen erzeugen weitere Parameter des Kommandos:

```
ADOCommand myCommand = new ADOCommand();
myCommand.ActiveConnection = myConnection;
myCommand.CommandText = "GetCompanyName";
myCommand.CommandType = CommandType.StoredProcedure;
```

Jetzt werden zwei Parameter des Objekts zur `Parameter`-Kollektion hinzugefügt:

SQL

```
SQLParameter workParam = null;
workParam = myCommand.Parameters.Add(new SQLParameter("@CustomerID",
SQLDataType.NChar, 5));
workParam.Direction = ParameterDirection.Input;
workParam.Value = "ALFKI";
workParam = myCommand.Parameters.Add(new
                 SQLParameter("@CompanyName",
                              SQLDataType.NChar, 40));
workParam.Direction = ParameterDirection.Output;
```

ADO

```
workParam = myCommand.Parameters.Add("@CustomerID", ADODBType.Char,
5);
workParam.Direction = ParameterDirection.Input;
workParam.Value = "ALFKI";
workParam = myCommand.Parameters.Add("@CompanyName", ADODBType.Char,
40);
workParam.Direction = ParameterDirection.Output;
```

Für die Parameterübergaben sind drei Schritte notwendig: Erzeugen und Hinzufügen der Parameter zur Kollektion, Einstellen der Richtung der Kollektion und Setzen der Werte für die Parameter.

Jetzt wird die Verbindung geöffnet und die Prozedur ausgeführt:

```
myConnection.Open()
myCommand.Execute()
Console.WriteLine("CompanyName = " & _
                  myCommand.Parameters("@CompanyName").Value)
```

Der letzte Block schließt die Verbindung, egal welchen Verlauf die Abfrage im `try-catch`-Block hatte:

```
Finally
   myConnection.Close
```

Das Schließen der Verbindung erfolgt hier nicht automatisch. Sie sollten die gezeigte Technik deshalb immer verwenden. Im Beispiel wurden keine Daten nach `DataReader` zurückgegeben. Die erfolgt mit `OUT`-Parametern. In diesem Fall ist auch das Schließen des `Command`-Objekts zwingend notwendig, denn die zurückgegebenen Daten werden erst zu diesem Zeitpunkt gültig und verfügbar.

10.4.5 DataSetCommand

Komfortable Kombination: DataSetCommand

Alle bisherigen Wege führten die Verarbeitung auf konventionelle Weise aus: verbindungsorientiert. In einer Mehrschicht-Umgebung, die möglicherweise auch im Netzwerk verteilt ist, ist dieses Verfahren weniger intelligent. Besser sind nachrichtenbasierte Systeme. Diese Systeme müssen sich aber zwischen Datenquelle und Netz schieben, denn die Datenquellen können Netzwerknachrichten nicht verarbeiten. Es gibt deshalb das Objekt `DataSetCommand` in ADO.NET. Dieses Objekt stellt Verbindungen zu `DataSet`-Objekten her und initiiert Kommandos gegen die Datenquelle, die naturgemäß aus üblichem SQL-Code bestehen.

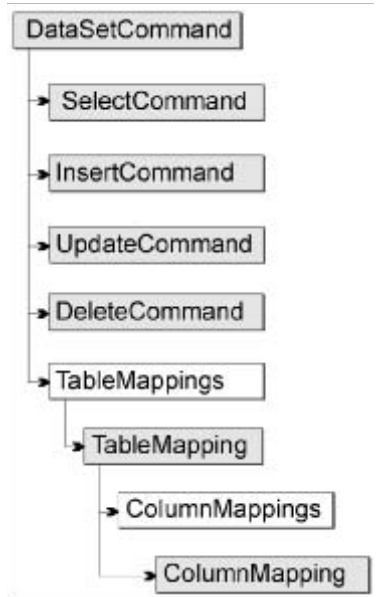


Abbildung 10.7: DataSetCommand-Objektmodell

Es stehen zwei Objekte zur Verfügung: `SQLDataSetCommand` und `ADODataSetCommand`. Beide kennen vier grundlegende Methoden – Erzeugen, Lesen, Ändern und Löschen –, die gegen die Datenbank ausgeführt werden können. Die Namen der Objekte können Sie dem Objektmodell in Abbildung 10.7 entnehmen.

Ein DataSetCommand-Objekt erzeugen

Die einfachste Funktion ist das Lesen von Daten aus einer Datenbank und das Speichern der empfangenen Daten in einem `DataTable`-Objekt, das seinerseits in einem `DataSet`-Objekt residiert. Benötigt werden dazu zwei Informationen: die Verbindungsinformation und das Auswahlkommando in SQL (`SELECT`).

Mit `ADODataSetCommand` wird dies für einen OLEDB-Provider folgendermaßen aussehen:

```
Dim workDSCMD As ADODataSetCommand
WorkDSCMD = New ADODataSetCommand _
  ("Select * from Customers", _
   "Provider=SQLOLEDB.1;Initial Catalog=Northwind;" & _
   "Data Source=MyServer;User ID=sa;")
```

Das Objekt `ADODataSetCommand` (korrekter gesagt: dessen Konstruktor) erwartet also zwei Parameter: die SQL-Anweisung und die Verbindungszeichenfolge.

Eine andere Variante löst dies auf die klassischen Verbindungs- und Kommandoobjekte auf:

```
Dim workDSCMD As ADODataSetCommand
Dim workCMD As ADOCommand
Dim workConn As ADOConnection
workConn = New ADOConnection _
         ("Provider=SQLOLEDB.1;Initial Catalog=Northwind;" & _
          "Data Source=MyServer;User ID=sa;")
workCMD = New ADOCommand("SELECT * FROM Customers", workConn)
workDSCMD = new ADODataSetCommand()
workDSCMD.SelectCommand = workCMD
```

Jetzt können die Daten abgerufen und in `DataTable` abgelegt werden. Dazu wird die Methode `FillDataSet` verwendet, die neben einer Instanz des Objekts `DataSet` auch den Namen einer Tabelle benötigt. Das folgende Beispiel zeigt die Vorgehensweise:

```
Dim workDS As DataSet = New DataSet("myDataSet")
Dim workDSCMD As ADODataSetCommand = New ADODataSetCommand _
            (SELECT * FROM Customers", _
            "Provider=SQLOLEDB.1;Initial Catalog=Northwind;" _
            "Data Source=MyServer;User ID=sa;")
workDSCMD.FillDataSet(workDS, "Customers")
DataSet workDS = new DataSet("mydataset");
ADODataSetCommand workDSCMD = new
          ADODataSetCommand("SELECT * FROM Customers",
          "Provider=SQLOLEDB.1;Initial Catalog=Northwind;
           Data Source=MyServer;User ID=sa;");
workDSCMD.FillDataSet(workDS, "Customers");
```

Nach der Ausführung von `FillDataSet` gibt es eine neue Tabelle *Customers* im `DataSet`-Objekt *workDS*. Es ist interessant zu wissen, wie die Tabelle dort angelegt wird, denn eine Deklaration ist zuvor offensichtlich nicht erfolgt.

Wann immer kein Schema existiert, werden die DataSetCommand-Objekte automatisch eines erzeugen. Wenn die Tabelle jedoch schon existiert, dann wird diese auch verwendet. Zwischen DataSet und DataSetCommand existiert keine physische Verbindung. Sie können deshalb eine beliebige Anzahl DataSet-Objekte erzeugen und mit Daten füllen.

DataSetCommand zum Ändern und Aktualisieren

Neben der Abfrage von Daten können Daten auch eingefügt, geändert oder gelöscht werden. Wenn die Daten in der DataSet-Instanz verändert wurden, müssen sie wieder zurück in die Datenbank geschrieben werden. Dazu wird die Methode Update verwendet. Das folgende Beispiel zeigt, wie dies angewendet wird.

Update

Intern wird die FillDataSet-Methode alle Änderungen an dem erzeugten DataSet-Objekt überwachen und speichern. Wenn die Methode Update aufgerufen wird, erzeugt die Methode die SQL-Anweisungen und sendet sie in der gespeicherten Reihenfolge an die Datenbank. Wenn Sie Reihen erst löschen und dann Änderungen durchführen, wird die Ausführung möglicherweise daran scheitern, dass die Datensätze nicht mehr existieren. Der Fehler tritt aber erst in der Datenbank auf, nicht schon vorher im Skript.

```
Dim workDS As DataSet = New DataSet("myDataSet")
Dim workDSCMD As ADODataSetCommand = New ADODataSetCommand _
        ("SELECT * FROM Customers", _
         "Provider=SQLOLEDB.1;Initial Catalog=Northwind;" & _
         "Data Source=MyServer;User ID=sa;")
workDSCMD.FillDataSet(workDS, "Customers")
' ... Hier erfolgen die Änderungen
workDSCMD.Update(workDS, "Customers")
```

An dieser Stelle ist auch ein Gedanke an die Unterschiede zwischen Entfernen und Löschen angebracht. Wenn Reihen als gelöscht gekennzeichnet wurden, wird beim Ausführen eine DELETE-Anweisung erzeugt. Wenn die Reihen dagegen zuvor bereits entfernt wurden, kann keine Anweisung mehr erfolgen. Dies wird sich auswirken, wenn Sie in der Datenbank beispielsweise DELETE-Trigger installiert haben.

10.4.6 Umgang mit Aliasen für Tabellen und Spalten

Ersatznamen verwenden (Mappings)

In allen bisherigen Beispielen wurden die originären Namen für Tabellen und Spalten verwendet. Manchmal haben Sie vielleicht keinen Einfluss auf die Gestaltung der Tabellen, wünschen sich aber umgänglichere Bezeichnungen für die Elemente. Dann können Sie Aliase definieren, die nur innerhalb von ADO.NET gelten. Dieser Vorgang wird in der Originalliteratur als »Mapping« bezeichnet, die beiden Varianten für Tabellen und Spalten heißen entsprechend »tablemapping« und »columnmapping«. Das folgende Beispiel zeigt, wie dies für Tabellen erfolgt:

```
workDSCMD.TableMappings.Add("MeineAutoren", "MyAuthors")
```

Dabei bezeichnet der erste Parameter den Namen der Tabelle im lokalen `DataSet`-Objekt, der zweite dagegen den Namen des Originals in der Datenquelle. Intern wird eine virtuelle Verbindung aufgebaut.

Für Spalten sieht dieser Vorgang ganz ähnlich aus:

```
With workDSCMD.TableMappings.Item(0).ColumnMappings
   .Add("au_id", "AuthorID")
   .Add("au_lname", "lastname")
   .Add("au_fname", "firstname")
   .Add("phone", "phone")
   .Add("address", "address")
   .Add("city", "city")
   .Add("state", "state")
   .Add("zip", "zip")
   .Add("contract", "contract")
End With
```

Eine besondere Arbeitsweise kennen die `Update`- und `FillDataSet`-Methoden. Diesen muss normalerweise der Name der Tabelle übermittelt werden. Sie können den Parameter auch weglassen. Dann wird als Name *Table* angenommen. Nun werden Sie die wichtigste Tabelle Ihrer Datenbank nicht *Table* nennen wollen. Mit dem Alias können Sie ihr aber intern den Namen *Table* geben und damit die Schreibung einer Vielzahl von Methodenaufrufen verkürzen.

```
Dim myDS As DataSet = New DataSet()
Dim myConnection As ADOConnection = New ADOConnection _
                    ("Provider=SQLOLEDB;Data Source=www;
                      Initial Catalog=Northwind;user id=sa")
Dim workDSCMD As ADODataSetCommand = New ADODataSetCommand _
                    ("SELECT * FROM Authors", myConnection)
workDSCMD.TableMappings.Add("Table", "Kunden")
With workDSCMD.TableMappings(0).ColumnMappings
   .Add("custom_id", "KundenID")
   .Add("lname", "Name")
   .Add("company", "Firma")
   .Add("phone", "Telefon")
   .Add("address", "Adresse")
   .Add("city", "Stadt")
   .Add("state", "Land")
   .Add("zip", "PLZ")
   .Add("contract", "Vertrag")
End With
workDSCMD.FillDataSet(myDS)
```

Die Tabelle *Kunden* im letzten Beispiel wird so zur Standardtabelle. Wird kein Name für eine Tabelle angegeben, bezieht sich die betreffende Aktion auf die Standardtabelle.

Das folgende Beispiel zeigt, wie die virtuelle Tabelle mit `FillDataSet` befüllt wird:

```
workDSCMD.TableMappings.Add("JK", "Kunden")
With workDSCMD.TableMappings(0).ColumnMappings
   .Add("custom_id", "KundenID")
   .Add("lname", "Name")
   .Add("company", "Firma")
   .Add("phone", "Telefon")
   .Add("address", "Adresse")
   .Add("city", "Stadt")
   .Add("state", "Land")
   .Add("zip", "PLZ")
   .Add("contract", "Vertrag")
End With
workDSCMD.FillDataSet(myDS, "JK")
```

Veränderung der Zuordnungen zur Laufzeit

Arbeitsweise der Mappings zur Laufzeit

Die Zuordnungen der Namen müssen nicht statisch sein. Sie können auch zur Laufzeit solche Zuordnungen ändern. Möglicherweise vereinfacht das die Programmierung, weil Sie universellere Klassen definieren können. Denken Sie beispielsweise an eine gespeicherte Prozedur, die folgende Anweisung ausführt:

```
SELECT @@IDENTITY
```

Dies wird nicht direkt funktionieren, denn das `DataSet`-Objekt benötigt einen Spaltennamen und mit dieser Anweisung wird kein Name zurückgegeben. Zu diesem Zweck wird ein `SQLSchemaMappingEvent` erzeugt, dessen korrespondierende Routine prüft, ob ein Spaltenname existiert. Ist das der Fall, wird dieser Name verwendet, wenn nicht, wird ein Standardname eingesetzt. Arbeiten Sie nun mit verschiedenen Tabellen, werden Sie den Standardnamen immer wieder neuen Namen zuordnen. Dies ist notwendig, weil Sie nur ein `SQLSchemaMappingEvent`-Ereignis im System gleichzeitig verwalten können.

```
Private Sub SchemaMappingEventHolder(ByVal sender As Object,
                                     ByVal e As _
                                     SQLSchemaMappingEvent)
  Dim column As DataColumn =
                           e.SchemaTable.Columns("DBCOLUMN_NAME")
  Dim row As DataRow
  For Each row in e.SchemaTable.Rows.All
```

```
    If row.RowState = DataRowState.Deleted Then
       row.RejectChanges()
    End If
    If row.IsNull(column) Or 0 = row(column).ToString Then
       column = e.SchemaTable.Columns("DataColumn")
       row(column) = New DataColumn("SomeUniqueName")
    End If
  Next
End Sub
```

Parameter und DataSetCommand

Verwendung von Parametern

Etwas komplexer sind parametrisierte Abfragen. Dabei werden die Abfragen vordefiniert und die Stellen, wo Parameter eingesetzt werden müssen, durch »?« gefüllt.

INSERT

Das folgende Beispiel zeigt das für `INSERT`:

```
Dim insertSQL As String
insertSQL = "INSERT INTO [Customers]
            ([CustomerID], [CompanyName],
             [ContactName], [ContactTitle], [Address],
             [City], [Region], [PostalCode], [Country],
             [Phone], [Fax])
             VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)";
```

UPDATE

Nur wenig anders sieht es für `UPDATE` aus:

```
Dim updateSQL As String
updateSQL = "UPDATE [Customers] SET [CustomerID] = ?,
                    [CompanyName] = ?,
                    [ContactName] = ?,
                    [ContactTitle] = ?,
                    [Address] = ?, [City] = ?,
                    [Region] = ?, [PostalCode] = ?,
                    [Country] = ?, [Phone] = ?,
                    [Fax] = ?
                    WHERE [CustomerID] = ? ";
```

DELETE

Besonders einfach wird eine parametrisierte `DELETE`-Anweisung erstellt:

```
Dim deleteSQL As String
deleteSQL = "DELETE FROM [Customers]
                    WHERE [CustomerID] = ? ";
```

Erstellung der Kollektion

Jetzt wird die `Parameter`-Kollektion erstellt, wie im nachfolgenden Beispiel gezeigt:

```
workParam = workCommand.Parameters.Add("CustomerID",
                                       ADODBType.WChar, 5);
workParam = workCommand.Parameters.Add("FldItemID",
                                       ADODBType.Integer, 4);
```

Die folgende Codezeile legt die Richtung der Parameter fest:

```
workParam.Direction = ParameterDirection.Input;
```

Eigenschaften

Außer `Input` können Sie folgende Eigenschaften einsetzen (in Klammern steht der numerische Wert, den sie repräsentieren):

- `Input` (1)
- `Output` (2)
- `InputOutput` (3)
- `ReturnValue` (6)

Jetzt wird noch die Spalte definiert, die als Datenquelle dient:

```
workParam.SourceColumn = "CustomerID";
```

Die Version stellt den Status der Quelle sicher:

```
workParam.SourceVersion = DataRowVersion.Default
```

Statt `Default` kann einer der folgenden Aufzählungstypen verwendet werden (in Klammern steht der numerische Wert, den sie repräsentieren):

- `Default` (0)
- `Original` (1)
- `Current` (2)
- `Proposed` (3)

Das folgende Beispiel zeigt die Anwendung. Die Unterschiede zu ADO sind nur marginal:

```
workParam.SourceVersion = DataRowVersion.Current;
workParam = workCommand.Parameters.Add("oldCustomerID",
                                       ADODBType.WChar, 5);
workParam.Direction = ParameterDirection.Input;
workParam.SourceColumn = "CustomerID";
workParam.SourceVersion = DataRowVersion.Original;
```

Input/Output-Parameter und Rückgabewerte

Parameter gespeicherter Prozeduren

Bei der Verwendung gespeicherter Prozeduren müssen Sie oft auch Rückgabewerte empfangen. ADO.NET betrachtet Rückgabewerte wie `OUTPUT`-Parameter. Das folgende Beispiel einer gespeicherten Prozedur benötigt einen IN-Parameter, gibt ein OUT-Parameter und zusätzlich einen Rückgabeparameter zurück.

```
CREATE PROCEDURE GetCompanyName
                @CustomerID nvarchar(5),
                @CompanyName nvarchar(40) output
AS
SELECT @CompanyName = CompanyName FROM Customers
   WHERE CustomerID = @CustomerIDGO
RETURN 99
```

Der folgende Code zeigt, wie die Prozedur *GetCompanyName* aufgerufen wird (als Visual Basic.NET-Modul):

```
Imports System
Imports System.Data
Imports System.Data.ADO
Namespace ConsoleApplication1
Module Module1
Shared Sub Main()
   Console.WriteLine("CustomerID benötigt")
   Console.WriteLine("example: OutParamsWithACommand ALFKI")
   ' Create a new Connection and DataSetCommand
   Dim myConnection As ADOConnection = New ADOConnection _
                     ("provider=SQLOLEDB;database=northwind;
                       data source=delphi;user id=sa;")
   Dim myCommand As ADOCommand = New
                    ADOCommand("GetCompanyName", myConnection)
   myCommand.CommandType = CommandType.StoredProcedure;
   Dim workParam As ADOParamater = Nothing
   workParam = myCommand.Parameters
               .Add("RETURN VALUE", ADODBType.Integer)
   workParam.Direction = ParameterDirection.ReturnValue
   workParam = myCommand.Parameters
               .Add("@CustomerID", ADODBType.Char, 5)
   workParam.Direction = ParameterDirection.Input
   workParam.Value = args(0)
   workParam = myCommand.Parameters
               .Add("@CompanyName", ADODBType.Char, 40)
   workParam.Direction = ParameterDirection.Output
   Try
      myConnection.Open()
```

```
      myCommand.Execute()
      Console.WriteLine("CompanyName= " & _
          myCommand.Parameters("RETURN_VALUE").Value)
      Console.WriteLine("CompanyName= " & _
          myCommand.Parameters("@CompanyName").Value)
      Catch e As Exception
      Console.WriteLine(e.ToString)
   Finally
      myConnection.Close
End Sub
End Module
End Namespace
```

Besonderheiten gibt es hier nicht: Der erste Parameter muss lediglich als `OUTPUT` gekennzeichnet werden, denn an dieser Stelle wird der Rückgabewert übertragen.

10.5 Verriegelungsverhalten (Concurrency)

Wenn mehrere Nutzer auf die Datenbank zugreifen und Daten verändern, kann es zu Konflikten kommen. Es gibt zwei Wege, damit umzugehen: Optimistic und Pessimistic Concurrency.

10.5.1 Pessimistic Concurrency

Pessimistic Concurrency

Pessimistic Concurrency ist ein Verriegelungsmechanismus, der Nutzer davor schützt, Daten so zu verändern, dass dies Daten anderer Nutzer beeinflusst. In einem pessimistischen Modell verriegelt ein Nutzer den Datensatz, der bearbeitet wird, sodass andere darauf nicht mehr schreibend zugreifen können. Es spielt dabei keine Rolle, ob solche Zugriffe tatsächlich zu erwarten sind. Dieses Modell eignet sich für Datenbanken mit vielen Daten, die verändert werden. In diesem Fall ist der prophylaktische Schutz effizienter als die immer wieder notwendigen Rückabwicklungen (Rollback) im Falle eines tatsächlich auftretenden Konflikts.

Arbeitsweise

Schon beim Lesen eines Datensatzes wird dieser gesperrt, wenn die theoretische Möglichkeit des Änderns besteht. Bis zum Ende des Prozesses bleibt die Verriegelung bestehen – kein anderer Nutzer kann diese Daten ändern. Selbstverständlich geschieht dies immer satzweise, niemals für die gesamte Tabelle.

10.5.2 Optimistic Concurrency

Optimistic Concurrency

Bei der optimistischen Verriegelung werden die Daten beim Lesen nicht prophylaktisch verriegelt. Man geht hier davon aus, dass Veränderungen der seltenere Fall sind. Es ist Aufgabe der Applikation, die Aktualität des Datensatzes sicherzustellen. Beispielsweise kann ein Nutzer einen Datensatz lesen. Dann liest ein anderer Nutzer denselben Datensatz und entscheidet sich, Daten zu verändern. Er ändert die Daten und aktualisiert die Datenbank. Davon bekommt der erste Nutzer nichts mit – er arbeitet fortan mit veralteten Daten. Noch kritischer ist es, wenn er die alten Daten später zurückschreibt, dann kann der Datenbestand inkonstistent werden.

Das Modell eignet sich für Datenbestände mit wenigen Daten. Im Falle eines Konflikts werden die Transaktionen zurückabgewickelt (Rollback). Vor allem wenn Lesevorgänge überwiegen, ist diese Methode effizienter.

Arbeitsweise

Im optimistischen Modell gibt es prinizipiell drei Wege, Veränderungen an Datensätzen zu erkennen. Zuerst kann einfach verglichen werden, ob sich die in der Datenbank befindlichen Daten von denen im lokalen Datensatzobjekt unterscheiden.

Das folgende Beispiel zeigt, wie hier Konflikte erkannt werden:

- 10:00 Uhr. Nutzer 1 liest einen Datensatz mit folgenden Daten:

```
CustID   LastName  FirstName
2605     Smith     Bob
```

  Die folgende Tabelle zeigt, welchen Zustand die Daten in der Datenbank (DB) und dem Datensatzobjekt (DO) haben:

Spalte	Original-DO	Aktuelles DO	Aktuelles DB
CustID	2605	2605	2605
NachName	Krause	Krause	Krause
VorName	Joerg	Joerg	Joerg

- 10:01 Uhr. Nutzer 2 liest dieselbe Zeile.
- 10:03 Uhr. Nutzer 2 ändert den Namen »Joerg« in »Jörg« und aktualisiert das Datensatz-Objekt:

Spalte	Original-DO	Aktuelles DO	Aktuelles DB
CustID	2605	2605	2605
NachName	Krause	Krause	Krause
VorName	Joerg	Jörg	Joerg

Da die Wert der aktuellen DB und des Original-DO übereinstimmen, wird der Prozess in der Datenbank ausgeführt.

- 10:05 Uhr. Nutzer 1 ändert den Wert »Joerg« in seiner lokalen Datensatzkopie in »Georg« und aktualisiert die Datenbank.

Spalte	Original-DO	Aktuelles DO	Aktuelles DB
CustID	2605	2605	2605
NachName	Krause	Krause	Krause
VorName	Joerg	Georg	Jörg

Da das vorhergehende Update ausgeführt wurde, stimmen die Wert des Originalen-DO und der aktuellen DB nicht überein. Das Update misslingt.

Der zweite und dritte Weg sind sich sehr ähnlich. Es gibt die Möglichkeit, eine Spalte mit Zeitstempeln oder mit Row-Daten zu verwenden. Bei jeder Veränderung eines Datensatzes wird die Spalte mit dem Zeitstempeln (oder der Row-ID) mit einem neuen, einmaligen und eindeutigen Wert belegt. So kann eine Änderung leicht festgestellt werden. Da sich oft mehrere Spalten ändern können, ist diese Version effizienter. Wann immer ein Nutzer einen Datensatz verändern will, wird der Zeitstempel im lokalen Datensatzobjekt mit dem in der Datenbank verglichen. Sind sie gleich, wird die Veränderung ausgeführt, andernfalls wird sie mit einem Fehler zurückgewiesen.

10.5.3 Praktische Umsetzung

In SQL sieht dieser Vorgang sehr einfach aus:

```
UPDATE Table1 set x = newx1, y = newy
    WHERE datetimestamp = originaldatetime
```

Besser – vor allem im Sinne der Systemleistung – ist die Verwendung einer gespeicherten Prozedur für die Behandlung von Updates. Die Prozedur wird von einem Trigger gestartet, der auf `UPDATE` reagiert.

Umsetzung in ADO.NET

Der Assistent für DataSet *(DataSetCommand Configuration Wizard,* DCW) bietet dafür die nötige Unterstützung. Er erstellt automatisch gespeicherte Prozeduren. Das folgende Beispiel zeigt, was der Assistent erzeugt hat. Die Prozedur enthält neben den aktuellen Parametern auch einen Satz von Kopien, bezeichnet mit *@Param1* bis *@Param9*. Dazu später mehr.

```
CREATE PROCEDURE AuthorUpdate
(
   @au_id id,
```

```
   @au_lname varchar(40),
   @au_fname varchar(20),
   @phone char(12),
   @address varchar(40),
   @city varchar(20),
   @state char(2),
   @zip char(5),
   @contract bit,
   @Param1 id,
   @Param2 varchar(40) /* Optimistic Concurrency Check */,
   @Param3 varchar(20) /* Optimistic Concurrency Check */,
   @Param4 varchar(40) /* Optimistic Concurrency Check */,
   @Param5 varchar(20) /* Optimistic Concurrency Check */,
   @Param6 bit /* Optimistic Concurrency Check */,
   @Param7 char(12) /* Optimistic Concurrency Check */,
   @Param8 char(2) /* Optimistic Concurrency Check */,
   @Param9 char(5) /* Optimistic Concurrency Check */
)
AS
  SET NOCOUNT OFF;
  UPDATE authors SET au_id = @au_id, au_lname = @au_lname,
                     au_fname = @au_fname, phone = @phone,
                     address = @address, city = @city,
                     state = @state, zip = @zip,
                     contract = @contract
  WHERE (au_id = @Param1) AND (address = @Param2)
    AND (au_fname = @Param3) AND (au_lname = @Param4)
    AND (city = @Param5) AND (contract = @Param6)
    AND (phone = @Param7) AND (state = @Param8)
    AND (zip = @Param9);
```

Dieser zweite Satz Parameter enthält die Originalwerte der Felder, wie sie aus der Datenbank geladen wurden. Sie werden verwendet, um die `WHERE`-Bedingung zu steuern. Dies ist also die erste Version der optimistischen Verriegelung.

Wenn Sie mit dieser Prozedur arbeiten, fragen Sie am Ende des `Update`-Befehls die Anzahl der veränderten Reihen ab. Ist diese Zahl 0, ist der Befehl wegen eines Konflikts misslungen. In der Regel ist dann ein erneutes Lesen der – nunmehr aktualisierten – Daten erforderlich.

10.6 Transaktionen

Transaktionen werden verwendet, um die Gültigkeit von Daten zu bestätigen. Wenn Sie an Kontobewegungen denken, so ist es unbedingt notwendig, dass die Belastung eines Kontos mit der Gutschrift des Betrags auf einem anderen Konto eine Einheit bildet. Computer können in einem solchen Prozess gestört werden, durch Stromausfall, Hard- oder Softwarefehler oder Bedienerprobleme. In solchen Fällen dürfen Datenbestände nicht inkonsistent werden. Wenn die Transaktion nicht bis zu Ende ausgeführt werden kann, muss zumindest der alte Zustand sicher erhalten bleiben.

10.6.1 Transaktionskommandos

Es gibt drei Kommandos, die für die Steuerung von Transaktionen verantwortlich sind: `BEGIN`, `COMMIT` und `ROLLBACK`. `BEGIN` markiert den Anfang der Transaktion; alles Folgende, bis zum `COMMIT` (Bestätigen) oder `ROLLBACK` (Verwerfen), wird als eine Einheit betrachtet. Misslingt nur eine der in diesem Block befindlichen Aktionen, werden alle anderen Vorgänge ebenfalls ungültig und verworfen.

```
BEGIN TRANSACTION

INSERT INTO account (account,amount,debitcredit)
    VALUES (100,100,'d')
INSERT INTO account (account,amount,debitcredit)
    VALUES (300,100,'c')
IF (@@ERROR > 0)
   ROLLBACK
ELSE
   COMMIT
```

10.6.2 Transaktionen in ADO.NET

Hier bietet ADO.NET nicht viel Neues. Ebenso wie in ADO muss die Datenbank Transaktionen unterstützen. Dann werden Transaktionen auch von ADO.NET angeboten. Kann die Datenbank dies nicht, hilft ADO.NET auch nicht weiter. Sie müssen dann die nötigen Sicherheitsmaßnahmen selbst im Programm kodieren.

Nutzung der Transaktionen in ADO.NET

In ADO.NET werden Transaktion durch das `Connection`-Objekt kontrolliert, ähnlich wie auch in ADO. Wird ADOConnection verwendet, so basiert dies auf der Implementierung des zugrunde liegenden OLEDB-Transaktionsmodells. Wenn ADO funktioniert hat, wird es mit ADO.NET keine Probleme geben.

Das Commit-Modell des DataSet-Objekts

Das `DataSet`-Objekt hat auch Methoden zur Steuerung von Transaktionen (`AcceptChanges` und `RejectChanges`). Dies hat nichts mit den Vorgängen in der Datenbank zu tun und beeinflusst nur das Verhalten des lokalen Cache des `DataSet`-Objekts.

Teil III
Konstanten und Referenzen

A Referenz ADO 2.6

Diese Kurzreferenz wiederholt nicht die bereits im Buch dargestellten Objekte, Methoden und Eigenschaften. Wenn Sie schnell nach bestimmten Namen suchen möchten, nutzen Sie den Index, der ausnahmslos alle im Buch beschriebenen Objekte und deren Bestandteile aufführt und auch extra kennzeichnet.

A.1 Properties-Kollektion

Die Kollektion `Properties` erlaubt sehr viele Einstellungen. Die komplette Liste für die einzelnen Objekte finden Sie nachfolgend. Der Auszug wurde mit dem OLEDB-Provider für den SQL Server erstellt. ODBC und Jet haben teilweise abweichende Eigenschaften.

Die Bestandteile der Tabellenüberschriften haben folgende Bedeutung:

- *Name*: Name der Eigenschaft
- *P*: Muss vor dem Öffnen definiert werden
- *O*: Die Angabe ist optional
- *R*: Die Eigenschaft ist lesbar
- *W*: Die Eigenschaft ist schreibbar
- *Type*: Datentyp des Parameters (Int=Integer, Bool=Boolean, BSTR=String usw.)
- *Beispiel*: Diesen Wert enthielt die Eigenschaft bei meinem SQL Server 7

Connection

Name	P	O	R	W	Type	Beispiel	Erklärung
`Active Sessions`	X	-	X	-	Int	0	Anzahl gleichzeitiger Verbindungen. 0 bedeutet: keine Begrenzung
`Allow Native Variant`	X	-	X	X	Bool	Falsch	Variant wird unterstützt

Name	P	O	R	W	Type	Beispiel	Erklärung
Alter Column Support	X	-	X	-	Int	501	Teil der Spalte, der geändert werden kann, DBCOLUMDESCFLAG: • _TYPENAME (1): Name • _ITYPEINFO (2): Typ • _PROPERTIES (4): Eigenschaften • _CLSID (8): Class-ID • _COLSIZE (16): Spaltenbreite • _DBCID (32): DBCID • _WTYPE (64): Datentyp • _PRECISION (128): Genauigkeit • _SCALE (256): Nachkommastellen
Application Name	X	-	X	X	BSTR		Name der Applikation
Asynchable Abort	X	-	X	-	Bool	Falsch	Asnynchroner Abbruch von Transaktionen
Asynchable Commit	X	-	X	-	Bool	Falsch	Asynchrone Bestätigung von Transaktionen
Auto Translate	X	-	X	X	Bool	Wahr	OEM/ANSI-Konvertierung aktiv
Autocommit Isolation Levels	X	-	X	X	Int	4096	Art der Transaktionsisolation, DBPROPVAL_OS: • _ENABLEALL (-1): Alle Dienste • _RESOURCEPOOLING (1): Ressourcen werden zwischengespeichert • _TXNENLISTMENT (2): MTS-Sessions werden aufgelistet
Catalog Location	X	-	X	-	Int	1	Position des Katalogs: • 1. Anfangs des Namens • 2. Ende des Namens 2 tritt bei Oracle auf, wo admin@catalog stehen kann. Sonst meist 1.
Catalog Term	X	-	X	-	BSTR	database	Datenquelle, z.B. »catalog«, »database« etc.
Catalog Usage	X	-	X	-	Int	15	Verwendung von Katalognamen in Kommandos, DBPROPVAL_CU: • STATEMENTS (1): Alle DML-Anweisungen • _TABLE_DEFINITION (2): Alle Table-Create-Anweisungen • _INDEX_DEFINITION (4): Alle Index-Create-Anweisungen • _PRIVILEGE_DEFINITION (8): Alle Privilege-Create-Anweisungen

Name	P	O	R	W	Type	Beispiel	Erklärung
Character Set Name	X	-	X	-	BSTR	iso_1	Verwendeter Zeichensatz
Column Definition	X	-	X	-	Int	1	Behandlung von Nullspalten, DBPROPVAL_CD: • _NOTNULL (1): Null ist erlaubt
Column Level Collation Support	X	-	X	-	Bool	Falsch	Sortierung auf Spaltenniveau
Connect Timeout	X	-	X	X	Int	15	Timeout in Sekunden
Connection Status	X	-	X	-	Int	1	Aktueller Status, DBPROPVAL_CS: • _UNINITIALIZED (0): Verbunden, aber nicht bereit • _INITIALIZED (1): Verbunden und bereit • _COMMUNICATIONFAILURE (2): Verbindungsfehler
Current Catalog	X	-	X	X	BSTR	Northwind	Aktuelle Datenbank
Current Collation Name	X	-	X	-	BSTR		Aktuelle Sortierung
Current Language	X	-	X	X	BSTR		Sprache für Systemmeldungen
Data Source	X	-	X	X	BSTR	WWW	Datenquelle; Servername, der verwendet werden soll
Data Source Name	X	-	X	-	BSTR	WWW	Tatsächliche Quelle
Data Source Object Threading Model	X	-	X	-	Int	1	Threading-Modell, DBPROPVAL_RT: • _FREETHREAD (1): Freier Thread • _APTMTTHREAD (2): Apartment • _SINGLETHREAD (4): Single
DBMS Name	X	-	X	-	BSTR	Microsoft SQL Server	Name des Datenbanksystems
DBMS Version	X	-	X	-	BSTR	07.00.0623	Version des Servers
Enable Fastload	X	-	X	X	Bool	Falsch	Bulk-Operationen sind erlaubt
Extended Properties	X	-	X	X	BSTR	Initital Catalog = Northwind	Weitere Optionen, die dem Provider übermittelt wurden
General Timeout	X	-	X	X	Int	0	Zeitüberschreitung in Sekunden für Anfragen

Name	P	O	R	W	Type	Beispiel	Erklärung
GROUP BY Support	X	-	X	-	Int	4	GROUP BY-Unterstützung, DBPROPVAL_BG: • _NOTSUPPORTED (1): nicht unterstützt • _EQUALS_SELECT (2): Alle Spalten in der SELECT-Liste müssen verwendet werden, andere sind nicht erlaubt. • _CONTAINS_SELECT (4): Alle Spalten in der SELECT-Liste müssen verwendet werden, andere sind zusätzlich erlaubt. • _NO_RELATION (8): SELECT-Liste und GROUP BY müssen nicht übereinstimmen. • _COLLATE (16): COLLATE wird unterstützt.
Heterogeneous Table Support	X	-	X	-	Int	3	JOIN über mehrere Kataloge wird unterstützt, DBPROPVAL_HT: • _DIFFERENT_CATALOGS (1): über mehrere Kataloge/Datenbanken • _DIFFERENT_PROVIDERS (2): über mehrere Provider
Identifier Case Sensitivity	X	-	X	-	Int	8	Bezeichner (Namen) sind Abhängig von Groß- und Kleinschreibung (DBPROPVAL_IC): • _UPPER (1): egal, speichert groß • _LOWER (2): egal, speichert klein • _SENSITIVE (4): wird unterschieden • _MIXED (8): egal, speichert Original
Initial Catalog	X	-	X	X	BSTR		Standardkatalog, wenn keine Angabe
Initial File Name	X	-	X	X	BSTR		Dateiname
Integrated Security	X	-	X	X	BSTR		Name des Authentifizierungsdienstes

Name	P	O	R	W	Type	Beispiel	Erklärung
Isolation Levels	X	-	X	-	Int	1118464	Transaktionsisolierung, DBPROPVAL_TI: • _CHAOS (16): Standardwert. Änderungen können in höherwertigen Ebenen nicht überschreiben. • _BROWSE(256): Änderungen sind sichtbar, bevor sie bestätigt werden. • _CURSORSTABILITY (4 096): Änderungen sind nicht sichtbar, bevor sie bestätigt wurden. • _REPEATABLEREAD (65 535): Änderungen anderen Transaktionen sind nun sichtbar • _ISOLATED (1 048 576): Konkurrierende Transaktionen werden so ausgeführt, als ob sie nacheinander ausgeführt würden.
Isolation Retention	X	-	X	-	Int	0	Transaktions-Isolations-Zurückhaltung, DBPROPVAL_TR: • _COMMIT_DC (1): Behält Isolation oder gibt frei, je nach vorherigem Zustand • _COMMIT (2): Behält Isolation • _COMMIT_NO (4): Gibt Isolation frei • _ABORT_DC (8): Behält Isolation oder gibt frei, je nach vorherigem Zustand • _ABORT (16): Behält Isolation • _ABORT_NO (32): Gibt Isolation frei • _DONTCARE (64): Standardwert. Je nach Zustand sowohl COMMIT als auch ABORT. • _BOTH (128): Behält beide • _NONE (256): Behält keines • _OPTIMISTIC (512): »Optimistic Concurrency« ist zu verwenden.
Locale Identifier	X	-	X	X	Int	1031	Locale ID (Sprache), 1031 = Deutsch
Maximum Index Size	X	-	X	-	Int	900	Maximale Breite einer indizierten Spalte in Bytes
Maximum Open Chapters	X	-	X	-	Int	0	Anzahl offener Chapter
Maximum Row Size	X	-	X	-	Int	8060	Maximale Breite einer Reihe, 0 = keine Begrenzung

Name	P	O	R	W	Type	Beispiel	Erklärung
Maximum Row Size Includes BLOB	X	-	X	-	Bool	Falsch	Begrenzung von BLOB-Spalten
Maximum Tables in SELECT	X	-	X	-	Int	256	Maximale Anzahl Tabellen in SELECT
Multiple Connections	X	-	X	X	Bool	Wahr	Mehrfache Verbindungen werden automatisch erstellt.
Multiple Parameter Sets	X	-	X	-	Bool	Wahr	Unterstützt mehrfache Parameter
Multiple Results	X	-	X	-	Int	1	Unterstützt mehrfache Ergebnislisten
Multiple Storage Objects	X	-	X	-	Bool	Falsch	Unterstützt mehrere Speicherobjekte
Multi-Table Update	X	-	X	-	Bool	Falsch	Unterstützt mehrfache Aktualisierungen
Network Address	X	-	X	X	BSTR		Netzwerkadresse
Network Library	X	-	X	X	BSTR		Netzwerkbibliothek
NULL Collation Order	X	-	X	-	Int	4	Sortierung von NULL, DBPROPVAL_NC: • _END (1): Ans Ende • _HIGH (2): An das obere Ende • _LOW (4): An das tiefere Ende • _START (8): An den Anfang
NULL Concatenation Behavior	X	-	X	-	Int	1	Verknüpfung von NULL-Werten mit anderen Spalten (DBPRPOVAL_CB): • _NULL (1): Ergebnis ist NULL • _NON_NULL (2): NULL wird ignoriert
OLE DB Version	X	-	X	-	BSTR	02.60	Version des Providers
OLE Object Support	X	-	X	-	Int	33	Unterstützung, DBPROPVAL_OO: • _BLOB (1): BLOBS werden als Objekte angesprochen • _IPERSIST (2): OLE • _ROWOBJECT (4): Row-Objekte • _SCOPED (8): Scope_Operationen (z. B. Index-Server) • _DIRECTBIND (16): Direkte Bindung an BLOB • _SINGLETON (32): Singleton-Operationen

Name	P	O	R	W	Type	Beispiel	Erklärung
Open Rowset Support	X	-	X	-	Int	0	Unterstützung des Providers für offene Datensätze, DBPROPVAL_ORS: • _TABLE (1): Tabellen • _INDEX (2): Indizes • _STOREDPROC (4): Gesp. Prozeduren • _INTEGRATEDINDEX (16): Tabelle und Index
ORDER BY Columns in Select List	X	-	X	-	Bool	Falsch	Wahr, wenn ORDER BY-Spalten in der SELECT-Liste stehen müssen
Output Parameter Availability	X	-	X	-	Int	4	Zeit, wann Ausgabeparameter gültig werden, DBPROPVAL_OA: • _NOTSUPPORTED (1): Nicht unterstützt • _ATEXECUTE (2): Unmittelbar nach der Ausführung • _ATROWRELEASE (4): Bei Freigabe der Reihe, also nach Close oder nach dem Umschalten auf die nächste Reihe
Packet Size	X	-	X	X	Int	4096	Netzwerkpaketgröße (Optimierungsparameter)
Pass By Ref Accessors	X	-	X	-	Bool	Wahr	Unterstützung für Accessoren
Password	X	-	X	X	BSTR		Kennwort
Persist Security Info	X	-	X	X	Bool		Sicherheitsinformationen werden gespeichert
Persistent ID Type	X	-	X	-	Int	1	Persistenz-Typ, DBPROPVAL_PT: • _GUID_NAME (1): GUID-Name • _GUID_PROPID (2): GUID-Prop-ID • _NAME (4): Name • _GUID (8): GUID • _PROPID (16): Property-ID • _PGUID_NAME (32): Property-Name • _PGUID_PROPID (64): Property-GUID

Name	P	O	R	W	Type	Beispiel	Erklärung
Prepare Abort Behavior	X	-	X	-	Int	2	Beide Optionen bestimmen, wie vorbereitete (Prepared) Kommandos in Transaktionen behandelt werden (CBPROPVAL_TB): • _DELETE (1): Abbruch löscht Kommando • _PRESERVE (2): Abbruch erhält Kommando
Prepare Commit Behavior	X	-	X	-	Int	2	
Procedure Term	X	-	X	-	BSTR	stored procedure	Bezeichnung von gespeicherten Prozeduren
Prompt	X	-	X	X	SInt	4	Art des Nutzereingriffs während der Initialisierung, DBPROMPT: • _PROMPT (1): Immer • _COMPLETE (2): Nur, wenn Informationen fehlen • _COMPLETEREQUIRED (3): Nur, wenn Informationen fehlen, optionale Daten können nicht eingegeben werden. • _NOPROMPT (4): Keine Aufforderung
Provider Friendly Name	X	-	X	-	BSTR	Microsoft OLE DB Provider for SQL Server	Name des Providers für Nutzerinfo
Provider Name	X	-	X	-	BSTR	sqloledb.dll	DLL des Providers
Provider Version	X	-	X	-	BSTR	08.00.0100	Version des Providers
Quoted Catalog Names	X	-	X	X	Bool	Falsch	Anführungszeichen in Katalognamen erlaubt
Quoted Identifier Sensitivity	X	-	X	-	Int	8	Behandlung von Groß- und Kleinschreibung, DBPROPVAL_IC: • _UPPER (1): egal, speichert groß • _LOWER (2): egal, speichert klein • _SENSITIVE (4): wird unterschieden • _MIXED (8): egal, speichert Original
Read-Only Data Source	X	-	X	-	Bool	Falsch	Quelle kann nur gelesen werden

Name	P	O	R	W	Type	Beispiel	Erklärung
Replication server name connect option	X	-	X	X	BSTR		Optionen für den Replikationsserver, wenn vorhanden
Reset Datasource	X	-	X	X	Int		Setzt Datenquelle zurück, DBPROPVAL_RD: • _RESETALL (-1): Ja, alles
Rowset Conversions on Command	X	-	X	-	Bool	Wahr	Kommandos können die Konvertierung bestimmen
Schema Term	X	-	X	-	BSTR	owner	Name des Schemas, »schema« oder »owner«
Schema Usage	X	-	X	-	Int	15	Verwendung von Schema-Namen in Kommandos, DBPROPVAL_SU: • _DML_STATEMENTS (1): In allen DML-Anweisungen • _TABLE_DEFINITION (2): In Tabellendefinitionen • _INDEX_DEFINITION (4): In Indexdefinitionen • _PRIVILEGE_DEFINITION (8): In Privilegien-Definitionen
Server Name	X	-	X	-	BSTR	WWW	Server-Name
Sort Order Name	X	-	X	-	BSTR	nocase_iso	Name der Sortiermethode
SQL Support	X	-	X	-	Int	283	SQL-Unterstützung
Sqlxmlx.dll progid	X	-	X	X	BSTR		ID der XML-DLL
Structured Storage	X	-	X	-	Int	1	Provider unterstützt Speicherobjekte, DBPROPVAL_SS: • _ISEQUENTIALSTREAM (1): ISequentialStream • _ISTREAM (2): IStream • _ISTORAGE (4): Istorage • _ILOCKBYTES (8): ILocktBytes
Subquery Support	X	-	X	-	Int	31	Arten verschachtelter Abfragen, DBPROPVAL_SQ: • _CORRELATEDSUBQUERIES (1): Alle korrellierenden • _COMPARISON (2): Alle vergleichenden • _EXISTS (4): Mit EXISTS • _IN (8): Mit IN-Schlüsselwort • _QUANTIFIED (16): Quantifizierte

Name	P	O	R	W	Type	Beispiel	Erklärung
Table Statistics Support	X	-	X	-	Int	0	Unterstützt statistische Angaben
Table Term	X	-	X	-	BSTR	table	Name für Tabellenart, z. B.: »table« oder »file«
Tag with column collation when possible	X	-	X	X	Bool	Falsch	Spaltensortierung voranstellen
Transaction DDL	X	-	X	-	Int	8	DDL-Kommandos innerhalb von Transaktionen, DBPROPVAL_TC: • _DML (1): Nur DML, DLL löst Fehler aus • _DDL_COMMIT (2): Nur DML, DDL löst COMMIT aus • _DDL_IGNORE (4): DDL wird ignoriert • _ALL (8): DDL und DML • _NONE (0): Transaktionen werden nicht unterstützt
Unicode Comparision Style	X	-	X	-	Int	196609	Sortieroptionen für Unicode-Daten
Unicode Locale Id	X	-	X	-	Int	1033	Locale ID für Unicode
Use Encryption for Data	X	-	X	X	Bool	Falsch	Verschlüsselung
Use Procedure for Prepare	X	-	X	X	Int	1	Temporäre gespeicherte Prozeduren sollen für Prepared-Kommandos verwendet werden
User ID	X	-	X	X	BSTR	sa	User ID
User Name	X	-	X	-	BSTR	dbo	User Name
Window Handle	X	-	X	X	Int		Handle des Fensters (nicht in ASP)
Workstation ID	X	-	X	X	BSTR	WWW	ID der Arbeitsstation

RecordSet

Name	P	O	R	W	Type	Beispiel	Erläuterung
Access Order	X	-	X	X	Int	2	Reihenfolge der Spalten, DBPROPVAL_AO: • _SEQUENTIALSTORAGEOBJECTS (1): Nur in der Reihenfolge, wie sie gespeichert wurden (i.d.R. nur bei XML) • _RANDOM (2): Egal • _SEQUENTIAL (0): Nach Ordnungsnummer
Blocking Storage Objects	X	-	X	-	Bool	Wahr	Verhindert andere Zugriffsmethoden
Bookmark Information	X	-	X	-	Int	0	Informationen über Lesezeichen, DBPROPVAL_BI: • _CROSSROWSET (1): Über alle Datensätze der Abfrage
Bookmark Type	X	-	X	-	Int	1	Art des Lesezeichens, DBPROPVAL_BMK: • _NUMERIC (1): Numerisch • _KEY (2): Schlüssel
Bookmarkable	X	-	X	X	Bool	Falsch	Lesezeichen werden unterstützt.
Change Inserted Rows	X	-	X	X	Bool	Falsch	Neue Zeilen können geändert werden.
Column Privileges	X	-	X	-	Bool	Wahr	Es bestehen Zugriffsrestriktionen.
Column Set Notification	X	-	X	-	Int	3	Änderungen führen zu einem Ereignis, DBPROPVAL_NP. Ereignisse werden in ASP nicht unterstützt.
Command Time Out	X	-	X	X	Int	30	Zeitüberschreitungswert in Sekunden, 0 = unendlich
Cursor Auto Fetch	X	-	X	X	Bool	Falsch	Zeiger holt automatisch neuen Wert.
Defer Column	X	-	X	X	Bool	Falsch	Holt Daten nur nach Anforderung.
Defer Prepare	X	-	X	X	Bool	Wahr	Holt vorbereitete Daten nur nach Anforderung.

Name	P	O	R	W	Type	Beispiel	Erläuterung
Delay Storage Object Updates	X	-	X	-	Bool	Falsch	Legt fest, ob auch Speicherobjekte im verzögerten Mode bedient werden.
Fastload Options	X	-	X	X	BSTR		Fastload-Optionen
Fetch Backwards	X	-	X	X	Bool	Falsch	Datensatz kann rückwärts durchlaufen werden.
Hold Rows	X	-	X	X	Bool	Falsch	Kann weitere Reihen lesen, ohne aktuelle Änderungen gültig zu machen.
Immobile Rows	X	-	X	X	Bool	Wahr	Eingefügte oder geänderte Reihen werden neu sortiert.
Keep Identity	X	-	X	X	Bool	Falsch	Legt fest, ob die Werte von IDENTITY-Spalten überschrieben werden dürfen.
Keep Nulls	X	-	X	X	Bool	Falsch	Legt fest, ob Spalten mit DEFAULT-Werten mit NULL überschrieben werden dürfen.
Literal Bookmarks	X	-	X	X	Bool	Falsch	Lesezeichen werden literal behandelt.
Literal Row Identity	X	-	X	-	Bool	Wahr	Binärvergleiche erfolgen literal.
Lock Mode	X	-	X	X	Int	1	Art der Satzverriegelung, DBPROPVAL_LM: • _NONE (1): Nicht erforderlich • _READ (2): Nur beim Lesen • _INTENT (4): Immer
Maximum BLOB Length	X	-	X	X	Int	0	Maximale Größe eines BLOB-Felds.
Maximum Open Rows	X	-	X	-	Int	0	Maximale Anzahl geöffneter Reihen.
Maximum Pending Rows	X	-	X	-	Int	0	Maximale Anzahl von Reihen mit unbestätigten Änderungen.
Maximum Rows	X	-	X	X	Int	0	Maximale Anzahl Reihen überhaupt.

Name	P	O	R	W	Type	Beispiel	Erläuterung
`Notification Granularity`	X	-	X	-	Int	1	Benachrichtigung bei Operationen, die mehrere Reihen betreffen, `DBPROPVAL_NT`. Hinweis: Ereignisse werden in ASP nicht unterstützt.
`Notification Phases`	X	-	X	-	Int	31	Art der Benachrichtigung, Ereignisse werden in ASP nicht unterstützt.
`Objects Transacted`	X	-	X	-	Bool	Falsch	Jedes Objekt ist in einer Transaktion.
`Others' Changes Visible`	X	-	X	X	Bool	Falsch	Änderungen anderer Nutzer sind sichtbar.
`Others' Inserts Visible`	X	-	X	X	Bool	Falsch	Einfügungen anderer Nutzer sind sichtbar.
`Own Changes Visible`	X	-	X	X	Bool	Falsch	Eigene Änderungen sind sichtbar.
`Own Inserts Visible`	X	-	X	X	Bool	Falsch	Eigene Einfügungen sind sichtbar.
`Preserve on Abort`	X	-	X	X	Bool	Falsch	Nach `ABORT` einer Transaktion bleibt der Datensatz aktiv und offen.
`Preserve on Commit`	X	-	X	X	Bool	Falsch	Nach `COMMIT` einer Transaktion bleibt der Datensatz aktiv und offen.
`Quick Restart`	X	-	X	X	Bool	Falsch	`RestartPositon` ist schnell.
`Reentrant Events`	X	-	X	-	Bool	Wahr	Wiedereintritt nach erneutem Aufruf.
`Remove Deleted Rows`	X	-	X	X	Bool	Falsch	Provider entfernt gelöschte Reihen.
`Report Multiple Changes`	X	-	X	-	Bool	Falsch	Erkennt Änderungen an mehrere Reihen
`Return Pending Inserts`	X	-	X	-	Bool	Falsch	Ausstehende (unausgeführte) Einfügungen können beim Lesen schon zurückgegeben werden.
`Row Delete Notification`	X	-	X	-	Int	3	Änderungen führen zu einem Ereignis, `DBPROPVAL_NP`. Ereignisse werden in ASP nicht unterstützt.
`Row First Change Notification`	X	-	X	-	Int	3	Änderungen führen zu einem Ereignis, `DBPROPVAL_NP`. Ereignisse werden in ASP nicht unterstützt.

Name	P	O	R	W	Type	Beispiel	Erläuterung
Row Insert Notification	X	-	X	-	Int	3	Änderungen führen zu einem Ereignis, DBPROPVAL_NP. Ereignisse werden in ASP nicht unterstützt.
Row Privileges	X	-	X	-	Bool	Wahr	Restriktionen auf Satzebene
Row Resynchronization Notification	X	-	X	-	Int	3	Änderungen führen zu einem Ereignis, DBPROPVAL_NP. Ereignisse werden in ASP nicht unterstützt.
Row Threading Model	X	-	X	-	Int	1	Threading-Modell, DBPROPVAL_RT: • _FREETHREAD (1): Freier Thread • _APTMTTHREAD (2): Apartment _SINGLETHREAD (4): Single
Row Undo Change Notification	X	-	X	-	Int	3	Änderungen führen zu einem Ereignis, DBPROPVAL_NP. Ereignisse werden in ASP nicht unterstützt.
Row Undo Delete Notification	X	-	X	-	Int	3	Änderungen führen zu einem Ereignis, DBPROPVAL_NP. Ereignisse werden in ASP nicht unterstützt.
Row Undo Insert Notification	X	-	X	-	Int	3	Änderungen führen zu einem Ereignis, DBPROPVAL_NP. Ereignisse werden in ASP nicht unterstützt.
Row Update Notification	X	-	X	-	Int	3	Änderungen führen zu einem Ereignis, DBPROPVAL_NP. Ereignisse werden in ASP nicht unterstützt.
Rowset Fetch Position Change Notification	X	-	X	-	Int	3	Änderungen führen zu einem Ereignis, DBPROPVAL_NP. Ereignisse werden in ASP nicht unterstützt.
Rowset Release Notification	X	-	X	-	Int	3	Änderungen führen zu einem Ereignis, DBPROPVAL_NP. Ereignisse werden in ASP nicht unterstützt.
Scroll Backwards	X	-	X	X	Bool	Falsch	Datensatz kann rückwärts durchlaufen werden.
Server Cursor	X	-	X	X	Bool	Falsch	Zeiger ist serverseitig.

Name	P	O	R	W	Type	Beispiel	Erläuterung
`Server Data on Insert`	X	-	X	X	Bool	Falsch	Provider holt Daten zum Auffrischen des Cache beim Ausführen von `INSERT`.
`Skip Deleted Bookmarks`	X	-	X	-	Bool	Falsch	Überspringt gelöschte Lesezeichen.
`Strong Row Identity`	X	-	X	-	Bool	Wahr	Reihen-Identität wird geprüft bei `INSERT`
`Unique Rows`	X	-	X	X	Bool	Falsch	Jede Reihe wird eindeutig durch die Spaltenwerte erkannt.
`Updatability`	X	-	X	X	Int	0	Methode für `UPDATE`, `DBPROPVAL_UP`: • `_CHANGE` (1): SetRows • `_DELETE` (2): DeleteRows • `_INSERT` (4): InsertRows
`Use Bookmarks`	X	-	X	X	Bool	Falsch	Verwendet Lesezeichen

Field

Name	P	O	R	W	Type	Beispiel	Erklärung
`BASECATALOGNAME`	X	-	-	-	Varchar		Name des Katalogs
`BASECOLUMNNAME`	X	-	-	-	Varchar		Name der Spalte
`BASESCHEMANAME`	X	-	-	-	Varchar		Name des Schemas
`BASETABLENAME`	X	-	-	-	Varchar		Name der Tabelle
`COLLATINGSEQUENCE`	X	-	-	-	Int		ID der Sortiersequenz
`COMPUTEMODE`	X	-	-	-	Int		Art der Berechnung berechneter Felder, `DBCOMPUTEMODE`: • `_COMPUTED` (1): Ist berechnet • `_DYNAMIC` (2): Ist berechnet und berechnet bei jeder Abfrage neu • `_NOTCOMPUTED` (3): Keine berechnete Spalte

Name	P	O	R	W	Type	Beispiel	Erklärung
DATETIMEPRECISION	X	-	-	-	USInt		Anzahl der Stellen des Nachkommateils von Zeitwerten (Sekundenbruchteile)
ISAUTOINCREMENT	X	-	-	-	Bool	Falsch	Spalte ist AUTO_INCREMENT
ISCASESENSITIVE	X	-	-	-	Bool	Falsch	Inhalt ist abhängig von Groß- und Kleinschreibung.
ISSEARCHABLE	X	-	-	-	USInt	4	Inhalt ist durchsuchbar, DB: • _UNSEARCHABLE (1): Kann nicht in WHERE verwendet werden • _LIKE_ONLY (2): Nur mit LIKE • _ALL_EXCEPT_LIKE (3): Alles außer LIKE. • _SEARCHABLE (4): Uneingeschränkt in WHERE verwendbar
OCTETLENGTH	X	-	-	-	USInt	10	Maximale Breite in Bytes
KEYCOLUMN	X	-	-	-	Bool	Falsch	Schlüsselspalte

Command

Name	P	O	R	W	Type	Example	Erklärung
Access Order	X	-	X	X	Int	2	Zugriffsreihenfolge
Base path	X	-	X	X	BSTR		Stammpfad
Blocking Storage Objects	X	-	X	-	Bool	Wahr	Verhindert andere Zugriffsmethoden
Bookmark Information	X	-	X	-	Int	0	Informationen über Lesezeichen, DBPROPVAL_BI: • _CROSSROWSET (1): Über alle Datensätze der Abfrage
Bookmark Type	X	-	X	-	Int	1	Art des Lesezeichens, DBPROPVAL_BMK: • _NUMERIC (1): Numerisch • _KEY (2): Schlüssel
Bookmarkable	X	-	X	X	Bool	Falsch	Lesezeichen werden unterstützt.
Change Inserted Rows	X	-	X	X	Bool	Falsch	Neue Zeilen können geändert werden.

Name	P	O	R	W	Type	Example	Erklärung
Column Privileges	X	-	X	-	Bool	Falsch	Es bestehen Zugriffsrestriktionen.
Column Set Notification	X	-	X	-	Int	3	Änderungen führen zu einem Ereignis, DBPROPVAL_NP. Ereignisse werden in ASP nicht unterstützt.
Command Time Out	X	-	X	X	Int	30	Zeitüberschreitungswert in Sekunden, 0 = unendlich
Command type	X	-	X	X	Int	21	Art des Kommandos
Cursor Auto Fetch	X	-	X	X	Bool	Falsch	Zeiger holt automatisch neuen Wert.
Defer Column	X	-	X	X	Bool	Falsch	Holt Daten nur nach Anforderung.
Defer Prepare	X	-	X	X	Bool	Wahr	Holt vorbereitete Daten nur nach Anforderung.
Delay Storage Object Updates	X	-	X	-	Bool	Falsch	Legt fest, ob auch Speicherobjekte im verzögerten Mode bedient werden
Fastload Options	X	-	X	X	BSTR		Fastload-Optionen.
Fetch Backwards	X	-	X	X	Bool	Falsch	Datensatz kann rückwärts durchlaufen werden.
Hold Rows	X	-	X	X	Bool	Falsch	Kann weitere Reihen lesen, ohne aktuelle Änderungen gültig zu machen.
Immobile Rows	X	-	X	X	Bool	Wahr	Eingefügte oder geänderte Reihen werden neu sortiert.
Keep Identity	X	-	X	X	Bool	Falsch	Legt fest, ob die Werte von IDENTITY-Spalten überschrieben werden dürfen.
Keep Nulls	X	-	X	X	Bool	Falsch	Legt fest, ob Spalten mit DEFAULT-Werten mit NULL überschrieben werden dürfen.
Literal Bookmarks	X	-	X	X	Bool	Falsch	Lesezeichen werden literal behandelt.
Literal Row Identity	X	-	X	-	Bool	Wahr	Binärvergleiche erfolgen literal.

Name	P	O	R	W	Type	Example	Erklärung
Lock Mode	X	-	X	X	Int	1	Art der Satzverriegelung, DBPROPVAL_LM: • _NONE (1): Nicht erforderlich • _READ (2): Nur beim Lesen • _INTENT (4): Immer
Mapping schema	X	-	X	X	BSTR		Verbundenes Schema
Maximum BLOB Length	X	-	X	X	Int	0	Maximale Größe eines BLOB-Felds.
Maximum Open Rows	X	-	X	-	Int	0	Maximale Anzahl geöffneter Reihen.
Maximum Pending Rows	X	-	X	-	Int	0	Maximale Anzahl von Reihen mit unbestätigten Änderungen.
Maximum Rows	X	-	X	X	Int	0	Maximale Anzahl Reihen überhaupt.
Notification Granularity	X	-	X	-	Int	1	Benachrichtigung bei Operationen, die mehrere Reihen betreffen, DBPROPVAL_NT. Hinweis: Ereignisse werden in ASP nicht unterstützt.
Notification Phases	X	-	X	-	Int	31	Art der Benachrichtigung, Ereignisse werden in ASP nicht unterstützt.
Objects Transacted	X	-	X	-	Bool	Falsch	Jedes Objekt ist in einer Transaktion.
Others' Changes Visible	X	-	X	X	Bool	Falsch	Änderungen anderer Nutzer sind sichtbar.
Others' Inserts Visible	X	-	X	X	Bool	Falsch	Einfügungen anderer Nutzer sind sichtbar.
Output encoding	X	-	X	X	BSTR	UTF-8	Kodierung der Ausgabe.
Output stream	X	-	X	X	Unknown		Ausgabe-Stream
Own Changes Visible	X	-	X	X	Bool	Falsch	Eigene Änderungen sind sichtbar.
Own Inserts Visible	X	-	X	X	Bool	Falsch	Eigene Einfügungen sind sichtbar.
Preserve on Abort	X	-	X	X	Bool	Falsch	Nach ABORT einer Transaktion bleibt der Datensatz aktiv und offen.

Name	P	O	R	W	Type	Example	Erklärung
Preserve on Commit	X	-	X	X	Bool	Falsch	Nach COMMIT einer Transaktion bleibt der Datensatz aktiv und offen.
Quick Restart	X	-	X	X	Bool	Falsch	RestartPosition ist schnell.
Reentrant Events	X	-	X	-	Bool	Wahr	Wiedereintritt nach erneutem Aufruf.
Remove Deleted Rows	X	-	X	X	Bool	Falsch	Provider entfernt gelöschte Reihen.
Report Multiple Changes	X	-	X	-	Bool	Falsch	Erkennt Änderungen an mehreren Reihen.
Return Pending Inserts	X	-	X	-	Bool	Falsch	Ausstehende (unausgeführte) Einfügungen können beim Lesen schon zurückgegeben werden.
Row Delete Notification	X	-	X	-	Int	3	Änderungen führen zu einem Ereignis, DBPROPVAL_NP. Ereignisse werden in ASP nicht unterstützt.
Row First Change Notification	X	-	X	-	Int	3	Änderungen führen zu einem Ereignis, DBPROPVAL_NP. Ereignisse werden in ASP nicht unterstützt.
Row Insert Notification	X	-	X	-	Int	3	Änderungen führen zu einem Ereignis, DBPROPVAL_NP. Ereignisse werden in ASP nicht unterstützt.
Row Privileges	X	-	X	-	Bool	Wahr	Restriktionen auf Satzebene
Row Resynchronization Notification	X	-	X	-	Int	3	Änderungen führen zu einem Ereignis, DBPROPVAL_NP. Ereignisse werden in ASP nicht unterstützt.
Row Threading Model	X	-	X	-	Int	1	Threading-Modell, DBPROPVAL_RT: • _FREETHREAD (1): Freier Thread • _APTMTTHREAD (2): Apartment _SINGLETHREAD (4): Single
Row Undo Change Notification	X	-	X	-	Int	3	Änderungen führen zu einem Ereignis, DBPROPVAL_NP. Ereignisse werden in ASP nicht unterstützt.

Name	P	O	R	W	Type	Example	Erklärung
Row Undo Delete Notification	X	-	X	-	Int	3	Änderungen führen zu einem Ereignis, DBPROPVAL_NP. Ereignisse werden in ASP nicht unterstützt.
Row Undo Insert Notification	X	-	X	-	Int	3	Änderungen führen zu einem Ereignis, DBPROPVAL_NP. Ereignisse werden in ASP nicht unterstützt.
Row Update Notification	X	-	X	-	Int	3	Änderungen führen zu einem Ereignis, DBPROPVAL_NP. Ereignisse werden in ASP nicht unterstützt.
Rowset Fetch Position Change Notification	X	-	X	-	Int	3	Änderungen führen zu einem Ereignis, DBPROPVAL_NP. Ereignisse werden in ASP nicht unterstützt.
Rowset Release Notification	X	-	X	-	Int	3	Änderungen führen zu einem Ereignis, DBPROPVAL_NP. Ereignisse werden in ASP nicht unterstützt.
Scroll Backwards	X	-	X	X	Bool	Falsch	Datensatz kann rückwärts durchlaufen werden
Server Cursor	X	-	X	X	Bool	Falsch	Zeiger ist serverseitig
Server Data on Insert	X	-	X	X	Bool	Falsch	Provider holt Daten zum Auffrischen des Cache beim Ausführen von INSERT
Skip Deleted Bookmarks	X	-	X	-	Bool	Falsch	Überspringt gelöschte Lesezeichen
ss stream flags	X	-	X	X	Int	0	Stream Flags
Strong Row Identity	X	-	X	-	Bool	Falsch	Reihen-Identität wird geprüft bei INSERT
Unique Rows	X	-	X	X	Bool	Falsch	Jede Reihe wird eindeutig durch die Spaltenwerte erkannt
Updatability	X	-	X	X	Int	0	Methode für UPDATE, DBPROPVAL_UP: • _CHANGE (1): SetRows • _DELETE (2): DeleteRows • _INSERT (4): InsertRows
Use Bookmarks	X	-	X	X	Bool	Falsch	Verwendet Lesezeichen

Name	P	O	R	W	Type	Example	Erklärung
xml root	X	-	X	X	BSTR		Basis des XML-DOM
xsl	X	-	X	X	BSTR		XSL-Sheet

A.2 Schemas

Schema	N	Beschreibung	
adSchema-Asserts	0	Grundeinstellungen der Datenbank	
		CONSTRAINT_CATALOG	Name der Datenbank (string)
		CONSTRAINT_SCHEMA	Name des Schemas (string)
		CONSTRAINT_NAME	Name der Einschränkung (string)
		IS_DEFERRABLE	TRUE, wenn aufschiebbar (boolean)
		INITIALLY_DEFERRED	TRUE, wenn bei der Initialisierung aufschiebbar (boolean)
		DESCRIPTION	Beschreibung (string)
adSchema-Catalogs	1	Informationen über die Datenbank	
		CATALOG_NAME	Name der Datenbank (string)
		DESCRIPTION	Beschreibung (string)
adSchemaCha-racterSets	2	Informationen über die zur Verfügung stehenden Zeichensätze	
		CHARACTER_SET_CATA-LOG	Name der Datenbank (string)
		CHARACTER_SET_ SCHEMA	Name des Schemas (string)
		CHARACTER_SET_NAME	Name des Zeichensatzes (string)
		FORM_OF_USE	Form der Nutzung (string)
		NUMBER_OF_CHARAC-TERS	Anzahl der Zeichen (int)
		DEFAULT_COLLATE_CA-TALOG	Name der Datenbank mit der Standardsortierung (string)
		DEFAULT_COLLATE_ SCHEMA	Name des Schemas mit der Standardsortierung (string)
		DEFAULT_COLLATE_ NAME	Name der Standardsortierung (string)
adSchemaCheck-Constraints	5	Einschränkungen der Datenbank	
		CONSTRAINT_CATALOG	Name der Datenbank (string)

Schema	N	Beschreibung	
		CONSTRAINT_SCHEMA	Name des Schemas (string)
		CONSTRAINT_NAME	Name der Einschränkung (string)
		CHECK_CLAUSE	Die WHERE-Bedingung der Einschränkung (string)
		DESCRIPTION	Beschreibung (string)
adSchemaCollations	3	Sortierverhalten der Datenbank	
		COLLATION_CATALOG	Name der Datenbank
		COLLATION_SCHEMA	Name des Schemas
		COLLATION_NAME	Name der Sortierbedingung
		CHARACTER_SET_CATALOG	Name der Datenbank
		CHARACTER_SET_SCHEMA	Name des Schemas
		CHARACTER_SET_NAME	Name des Zeichensatzes
		PAD_ATTRIBUTE	Dieses Attribut entscheidet, ob Spalten mit variabler Länge zum Sortieren mit Leerzeichen aufgefüllt werden. Kann folgende Werte zurückgeben: • NO PAD. Keine Auffüllung • PAD SPACE. Wird aufgefüllt
adSchemaColumnPrivileges	13	Zugriff auf die Rechte der Nutzer auf Spalten und Tabellen, die verfügbar oder vergeben sind. Sinnvolle Auswahl: TABLE_NAME	
		GRANTOR	User, der das Recht vergeben hat
		GRANTEE	User, der das Recht besitzt
		TABLE_CATALOG	Datenbank, zu der die Tabelle gehört (string)
		TABLE_SCHEMA	Schema, zu dem die Tabelle gehört (string)
		TABLE_NAME	Tabellenname (string)
		COLUMN_NAME	Spaltenname (string)
		COLUMN_GUID	GUID der Spalte (GUID)
		COLUMN_PROPID	Property-ID der Spalte (long)
		PRIVILEGE_TYPE	Art des Rechts (string): • SELECT • DELETE • INSERT • UPDATE • REFERENCES

Schema	N	Beschreibung	
		IS_GRANTABLE	TRUE, wenn das Recht zugewiesen werden kann (boolean)
adSchemaColumns	4	Enthält Informationen über Spalten der Tabellen und Sichten	
		TABLE_CATALOG	Name der Datenbank (string)
		TABLE_SCHEMA	Name des Schemas (string)
		TABLE_NAME	Name der Tabelle (string)
		COLUMN_NAME	Spaltenname (string)
		COLUMN_GUID	GUID der Spalte (string)
		COLUMN_PROPID	Property-ID der Spalte (string)
		ORDINAL_POSITION	Numerischer Index der Spalte (Spaltennummer) (integer)
		COLUMN_HASDEFAULT	TRUE, wenn Spalte einen Standardwert hat (boolean)
		COLUMN_DEFAULT	Standardwert der Spalte
		COLUMN_FLAGS	DBCOLUMNFLAGS Bitmaske entsprechend der folgenden Liste: • _MAYDEFER (2): Kann abgeleitet werden • _WRITE (4): Kann beschrieben werden • _WRITEUNKNOWN (8): Nicht bekannt, ob geschrieben werden kann • _ISFIXEDLENGTH (16): Spalte hat feste Breite • _ISNULLABLE (32): Kann NULL werden • _MAYBENULL (64): Kann NULL enthalten • _ISLONG (128): Ist eine BLOB-Spalte • _ISROWID (256): Ist eine RowID-Spalte • _ISROWVER (512): TimeStamp oder andere Row-Versionsverwaltung • _CHACHEDEFERRED (4 096): Abgeleitete Spalte wird zwischengespeichert • _ISCHAPTER (8 192): Ist ein Chapter
		IS_NULLABLE	TRUE, wenn Spalte NULL werden darf (boolean)

Schema	N	Beschreibung	
		DATA_TYPE	Datentyp, entspricht den DataTypeEnum-Konstanten (integer)
		TYPE_GUID	GUID des Datentyps
		CHARACTER_MAXIMUM_LENGTH	Maximale Anzahl Stellen der Spalte (long)
		CHARACTER_OCTET_LENGTH	Maximale Anzahl Bytes bei Zeichenketten oder Binärspalten (long)
		NUMERIC_PRECISION	Genauigkeit; Anzahl Stellen vor dem Komma
		NUMERIC_SCALE	Genauigkeit; Anzahl Stellen nach dem Komma
		DATETIME_PRECISION	Genauigkeit einer Zeitangabe; Anzahl der Stellen nach dem Komma der Sekunden
		CHARACTER_SET_CATALOG	Datenbank, in der der Zeichensatz definiert ist
		CHARACTER_SET_SCHEMA	Schema des Zeichensatzes
		CHARACTER_SET_NAME	Name des Zeichensatzes
		COLLATION_CATALOG	Datenbank mit der Sortiervorschrift
		COLLATION_SCHEMA	Schema der Datenbank der Sortiervorschrift
		COLLATION_NAME	Name der Sortiervorschrift
		DOMAIN_CATALOG	Datenbank, in der die Domain definiert ist
		DOMAIN_SCHEMA	Schema der Domain
		DOMAIN_NAME	Name der Domain
		DESCRIPTION	Beschreibung der Spalte
adSchemaColumnsDomainUsage	11	Gibt die Spalten an, die zu einer bestimmten Domäne gehören, wenn solche Domänen definiert wurden.	
		DOMAIN_CATALOG	Standard-Katalog der Domäne
		DOMAIN_SCHEMA	Schema der Domäne
		DOMAIN_NAME	Name der Domäne
		TABLE_CATALOG	Datenbank, zu der die Tabelle gehört (string)
		TABLE_SCHEMA	Schema, zu dem die Tabelle gehört (string)
		COLUMN_NAME	Spaltenname (string)

Schema	N	Beschreibung	
		COLUMN_GUID	GUID der Spalte
		COLUMN_PROPID	Property-ID der Spalte
adSchemaConstraintColumn-Usage	6	Auf Spalten bezogene Einschränkungen	
		TABLE_CATALOG	Datenbank, zu der die Tabelle gehört (string)
		TABLE_SCHEMA	Schema, zu dem die Tabelle gehört (string)
		TABLE_NAME	Tabellenname (string)
		COLUMN_NAME	Spaltenname (string)
		COLUMN_GUID	GUID der Spalte
		COLUMN_PROPID	Property-ID der Spalte
		CONSTRAINT_CATALOG	Name der Datenbank (string)
		CONSTRAINT_SCHEMA	Name des Schemas (string)
		CONSTRAINT_NAME	Name der Einschränkung (string)
adSchemaConstraintTableUsage	7	Tabellen, zu denen Einschränkungen definiert wurden	
		TABLE_CATALOG	Datenbank, zu der die Tabelle gehört (string)
		TABLE_SCHEMA	Schema, zu dem die Tabelle gehört (string)
		TABLE_NAME	Tabellenname (string)
		CONSTRAINT_CATALOG	Name der Datenbank (string)
		CONSTRAINT_SCHEMA	Name des Schemas (string)
		CONSTRAINT_NAME	Name der Einschränkung (string)
adSchemaCubes	32	Informationen über Cubes einer OLAP-Datenbank	
		CATALOG_NAME	Name der Datenbank
		SCHEMA_NAME	Name des Schemas
		CUBE_NAME	Name des Cube
		CUBE_TYPE	Typ des Cube (string): • CUBE. Regulärer Cube • VIRTUAL CUBE. Virtueller Cube
		CUBE_GUID	GUID des Cube
		CREATED_ON	Datum der Erzeugung
		LAST_SCHEMA_UPDATE	Letzte Änderung am Schema
		SCHEMA_UPDATED_BY	User, der das Schema geändert hat
		LAST_DATE_UPDATE	Letzte Änderung an der Datenbank

Schema	N	Beschreibung	
		DATA_UPDATED_BY	User, der Daten zuletzt geändert hat
		DESCRIPTION	Beschreibung (string)
adSchemaDB-InfoKeywords	30	Gibt eine Liste Provider-spezifischer Schlüsselwörter zurück. Dazu gehören nicht die SQL-Standardanweisungen.	
adSchemaDB-InfoLiterals	31	Gibt eine Liste Provider-spezifischer Literale zurück, die in Bezeichnern verwendet werden können. Das Datensatzobjekt enthält für den SQL Server 7 folgende Felder: • LiteralName. Name des Literals, z.B. COLUMN_NAME • LiteralValue. Zulässiger Literal, z.B. der Punkt als CATALOG_SEPARATOR • InvalidChars. Im Namen unzulässige Zeichen • IcnvalidStartingChars. Für Namen und zulässiges erstes Zeichen • Literal. Interne Nummer des Literals • Supported. Unterstützung durch den Provider (TRUE oder FALSE) • Maxlen. Maximale Länge des Names, z.B. 128 für COLUMN_NAME	
adSchemaDimensions	33	Gibt Informationen über die Dimension eines Cube zurück.	
		CATALOG_NAME	Name der Datenbank
		SCHEMA_NAME	Name des Schemas
		CUBE_NAME	Name des Cube
		DIMENSION_NAME	Name der Dimension
		DIMENSION_UNIQUE_NAME	Eindeutiger (interner) Name der Dimension
		DIMENSION_GUID	GUID der Dimension
		DIMENSION_CAPTION	Überschrift
		DIMENSION_ORDINAL	Numerischer Index der Dimension in ihrer Kollektion
		DIMENSION_TYPE	Typ der Dimension (string): • MD_DIMTYPE_MEASURE. Maß-Dimension • MD_DIMTYPE_TIME. Zeit-Dimension • MD_DIMTYPE_OTHER. Andere • MD_DIMTYPE_UNKNOWN. Unbekannt
		DIMENSION_CARDINALITY	Anzahl der Mitglieder der Dimension (long)
		DEFAULT_HIERARCHY	Name der Standard-Hierarchie
		DESCRIPTION	Beschreibung (string)

Schema	N	Beschreibung	
		IS_VIRTUAL	TRUE, wenn Dimension virtuell ist
adSchemaForeignKeys	27	Gibt Informationen über Fremdschlüssel und den zugehörigen Primärschlüssel zurück	
		PK_TABLE_CATALOG	Datenbank, die die Tabelle mit dem Primärschlüssel enthält
		PK_TABLE_SCHEMA	Schema, in dem die Tabelle mit dem Primärschlüssel ist
		PK_TABLE_NAME	Tabelle mit Primärschlüssel
		PK_COLUMN_NAME	Spalte, die den Primärschlüssel hat
		PK_COLUMN_GUID	GUID der Spalte des Primärschlüssels
		PK_COLUMN_PROPID	Property-ID der Spalte des Primärschlüssels
		FK_TABLE_CATALOG	Datenbank, in der der Fremdschlüssel definiert ist
		FK_TABLE_SCHEMA	Schema, das die Tabelle enthält, die den Fremdschlüssel besitzt
		FK_TABLE_NAME	Tabelle, die den Fremdschlüssel enthält
		FK_COLUMN_NAME	Spalte, auf die der Fremdschlüssel weist
		FK_COLUMN_GUID	GUID der Spalte
		FK_COLUMN_PROPID	Property-ID der Spalte
		ORDINAL	Numerischer Index für mehrere Fremdschlüssel. Wenn mehrere Fremdschlüssel definiert wurden, können diese mit diesem Wert unterschieden werden.
		UPDATE_RULE	Die Aktion, die ausgeführt wird, wenn eine UPDATE-Regel implementiert wurde. • NO ACTION. Wird zurückgegeben, wenn nichts definiert wurde. • CASCADE. Übergeordneter Wert • SET NULL. Wird auf NULL gesetzt. • SET DEFAULT. Wird auf Standardwert gesetzt.

Schema	N	Beschreibung	
		DELETE_RULE	Die Aktion, die ausgeführt wird, wenn eine DELETE-Regel implementiert wurde. • NO ACTION. Wird zurückgegeben, wenn nichts definiert wurde. • CASCADE. Übergeordneter Wert • SET NULL. Wird auf NULL gesetzt. • SET DEFAULT. Wird auf Standardwert gesetzt.
		PK_NAME	Name des Primärschlüssels
		FK_NAME	Name des Fremdschlüssels
		DEFERRABILITY	Ableitbarkeit des Fremdschlüssels. Kann folgende numerische Werte zurückgeben: • DBPROPVAL_DF_INITIALLY_DEFERRED (1). Bei der Initialisierung ableiten • DBPROPVAL_DF_INITIALLY_IMMEDIATE (2). Sofort nach dem Erzeugen ableiten • DBPROPVAL_DF_NOT_DEFERRABLE (3). Nicht ableitbar
adSchema-Hierarchies	34	Informationen über die Hierarchie einer mehrdimensionalen Datenbank	
		CATALOG_NAME	Name der Datenbank
		SCHEMA_NAME	Name des Schemas
		CUBE_NAME	Name des Cube
		DIMENSION_UNIQUE_NAME	Eindeutiger (interner) Name der Dimension
		HIERARCHY_NAME	Name der Hierarchie
		HIERARCHY_UNIQUE_NAME	Eindeutiger (interner) Name der Hierarchie
		HIERARCHY_GUID	GUID der Hierarchie
		HIERARCHY_CAPTION	Überschrift

Schema	N	Beschreibung	
		DIMENSION_TYPE	Typ der Dimension (string): • MD_DIMTYPE_MEASURE. Maß-Dimension • MD_DIMTYPE_TIME. Zeit-Dimension • MD_DIMTYPE_OTHER. Andere • MD_DIMTYPE_UNKNOWN. Unbekannt
		HIERARCHY_CARDINALITY	Anzahl der Mitglieder der Hierarchie (long)
		DEFAULT_MEMBER	Name des Standard-Mitglieds
		ALL_MEMBER	Name des Standard-Mitglieds, wenn die erste Ebene ALL ist
		DESCRIPTION	Beschreibung (string)
adSchemaIndexes	12	Gibt die Indizes der Datenbank an	
		TABLE_CATALOG	Name der Datenbank der Tabelle (string)
		TABLE_SCHEMA	Name des Schemas der Tabelle (string)
		TABLE_NAME	Name der Tabelle (string)
		INDEX_CATALOG	Name der Datenbank (string)
		INDEX_SCHEMA	Name des Schemas (string)
		INDEX_NAME	Name des Index (string)
		PRIMARY_KEY	TRUE, wenn die Indexspalte auch den Primärschlüssel enthält (boolean)
		UNIQUE	TRUE, wenn die Spalte UNIQUE ist (boolean)
		CLUSTERED	TRUE, wenn Index geclustert ist (boolean)
		TYPE	Typ des Index. Der Wert kann sein: • DBPROPVAL_IT_BTREE (1). B+-Baum • DBPROPVAL_IT_HASH (2). Hash-Datei • DBPROPVAL_IT_CONTENT (3). Content • DBPROPVAL_IT_OTHER (4). Anderer Index
		FILL_FACTOR	Speicherverbrauch bei der Indizierung eines B+-Baumes

Schema	N	Beschreibung	
		INITIAL_SIZE	Totaler Speicherverbrauch in Byte zum Zeitpunkt der Erzeugung
		NULLS	Zeigt an, ob NULL-Werte erlaubt sind: • DBPROPVAL_IN_DISALLOWNULL (1). Index erlaubt keine Felder, die NULL enthalten. • DBPROPVAL_IN_IGNORENULL (2). Der Index bezieht Einträge, deren Indexfeld NULL ist, nicht mit ein • DBPROPVAL_IN_IGNOREANYNULL (4). Der Index fügt Felder, die NULL sind, nicht ein.
		SORT_BOOKMARKS	Regelt, wie der Index sich wiederholende Schlüssel behandelt. TRUE, wenn der Index solche Schlüssel nach Lesezeichen sortiert.
		AUTO_UPDATE	TRUE, wenn der Index sich selbst aktualisiert
		NULL_COLLATION	Regelt, wie NULL einsortiert wird. Kann sein: • DBPROPVAL_NC_END (1). Ans Ende der Liste • DBPROPVAL_NC_HIGH (2). Über den größten Wert • DBPROPVAL_NC_LOW (4). Unter den kleinsten Wert • DBPROPVAL_NC_START (8). An den Anfang der Liste
		ORDINAL_POSITION	Numerische Position der Spalte, beginnend mit 1
		COLUMN_NAME	Name der Spalte
		COLUMN_GUID	GUID der Spalte
		COLUMN_PROPID	Property-ID der Spalte
		COLLATION	Sortierreihenfolge: • DB_COLLATION_ASC (1). Aufsteigend • DB_COLLATION_ASC (2). Absteigend
		CARDINALITY	Anzahl eindeutiger Werte im Index
		PAGES	Anzahl Speicherseiten
		FILTER_CONDITION	Teil hinter WHERE der Filter-Bedingung

Schema	N	Beschreibung	
		INTEGRATED	TRUE, wenn integriert
adSchemaKeyColumnUsage	8	Spalten, die durch Schlüssel eingeschränkt sind	
		CONSTRAINT_CATALOG	Name der Datenbank
		CONSTRAINT_SCHEMA	Name des Schemas
		CONSTRAINT_NAME	Name der Einschränkung
		TABLE_CATALOG	Name der Datenbank der Tabelle (string)
		TABLE_SCHEMA	Name des Schemas der Tabelle (string)
		TABLE_NAME	Name der Tabelle (string)
		COLUMN_NAME	Spaltennamen
		COLUMN_GUID	GUID der Spalte
		COLUMN_PROPID	Property-ID der Spalte
adSchemaLevels	35	Informationen über die Ebenen einer Dimension	
		CATALOG_NAME	Name der Datenbank
		SCHEMA_NAME	Name des Schemas
		CUBE_NAME	Name des Cube
		DIMENSION_UNIQUE_NAME	Eindeutiger (interner) Name der Dimension
		HIERARCHY_UNIQUE_NAME	Eindeutiger (interner) Name der Hierarchie
		LEVEL_NAME	Name der Ebene
		LEVEL_UNIQUE_NAME	Eindeutiger (interner) Name der Ebene
		LEVEL_GUID	GUID der Ebene
		LEVEL_CAPTION	Überschrift der Ebene
		LEVEL_NUMBER	Index
		LEVEL_CARDINALITY	Mächtigkeit, Anzahl der Mitglieder

Schema	N	Beschreibung	
		LEVEL_TYPE	Typ, Bitwert: • MDLEVEL_TYPE_REGULAR (0). Normale Ebene • MDLEVEL_TYPE_ALL (1). Oberste Ebene der Hierarchie für alle Ebenen • MDLEVEL_TYPE_CALCULATED (2). Berechnet • MDLEVEL_TYPE_TIME (4). Time-Ebene • MDLEVEL_TYPE_TIME_YEARS (20). Time-Ebene, basiert auf Jahren • MDLEVEL_TYPE_HALF_YEAR (36). Time-Ebene, basiert auf Halbjahren • MDLEVEL_TYPE_QUARTERS (68). Time-Ebene, basiert auf Quartalen • MDLEVEL_TYPE_MONTH (132). Time-Ebene, basiert auf Monaten • MDLEVEL_TYPE_WEEKS (260). Time-Ebene, basiert auf Wochen • MDLEVEL_TYPE_DAYS (516). Time-Ebene, basiert auf Tagen • MDLEVEL_TYPE_HOURS (772). Time-Ebene, basiert auf Stunden • MDLEVEL_TYPE_MINUTES (1028). Time-Ebene, basiert auf Minuten • MDLEVEL_TYPE_SECONDS (2052). Time-Ebene, basiert auf Sekunden • MDLEVEL_TYPE_TIME_UNDEFINED (4100). Nicht definiert • MDLEVEL_TYPE_UNKNOWN (0). Nicht definiert
		DESCRIPTION	Beschreibung
adSchemaMeasures	36	Informationen über Maßeinheiten	
		CATALOG_NAME	Name der Datenbank
		SCHEMA_NAME	Name des Schemas
		CUBE_NAME	Name des Cube

Schema	N	Beschreibung	
		MEASURE_NAME	Name der Maßeinheit
		MEASURE_UNIQUE_NAME	Eindeutiger (interner) Name der Maßeinheit
		MEASURE_CAPTION	Überschrift
		MEASURE_GUID	GUID
		MEASURE_AGGREGATOR	Typ der Aggregation
		DATA_TYPE	Datentyp
		NUMERIC_PRECISION	Stellen vor dem Komma
		NUMERIC_SCALE	Stellen nach dem Komma
		MEASURE_UNITS	Maßeinheit
		DESCRIPTION	Beschreibung
		EXPRESSION	Ausdruck, der der Kalkulation zugrunde liegt
adSchemaMembers	38	Informationen über Mitglieder	
		CATALOG_NAME	Name der Datenbank
		SCHEMA_NAME	Name des Schemas
		CUBE_NAME	Name des Cube
		DIMENSION_UNIQUE_NAME	Eindeutiger (interner) Name der Dimension
		HIERARCHY_UNIQUE_NAME	Eindeutiger (interner) Name der Hierarchy
		LEVEL_UNIQUE_NAME	Eindeutiger (interner) Name der Ebene
		LEVEL_NUMBER	Nummer der Ebene
		MEMBER_ORDINAL	Ordnungsnummer des Mitglieds
		MEMBER_NAME	Name des Mitglieds
		MEMBER_UNIQUE_NAME	Eindeutiger (interner) Name des Mitglieds
		MEMBER_TYPE	Typ: • MDMEMBER_TYPE
		MEMBER_GUID	GUID des Mitglieds
		MEMBER_CAPTION	Überschrift
		CHILDREN_CARDINALITY	Anzahl der untergeordneten Mitglieder. Dies ist ein nicht unbedingt exakter Wert.
		PARENT_LEVEL	Position oder Nummer des übergeordneten Mitglieds

Schema	N	Beschreibung	
		PARENT_UNIQUE_NAME	Name des übergeordneten Mitglieds
		PARENT_COUNT	Anzahl der übergeordneten Elemente
		DESCRIPTION	Beschreibung
		<property>	Eine weitere Spalte für jede Eigenschaft eines Mitglieds
adSchemaPrimaryKeys	28	Primärschlüsseldefinitionen	
		TABLE_CATALOG	Datenbank, die die Tabelle mit dem Primärschlüssel enthält
		TABLE_SCHEMA	Schema, in dem die Tabelle mit dem Primärschlüssel ist
		TABLE_NAME	Tabelle mit Primärschlüssel
		COLUMN_NAME	Spalte, die den Primärschlüssel hat
		COLUMN_GUID	GUID der Spalte des Primärschlüssels
		COLUMN_PROPID	Property-ID der Spalte des Primärschlüssels
		ORDINAL	Reihenfolge der Spalten
		PK_NAME	Name des Primärschlüssels
adSchemaProcedureColumns	29	Informationen über die Spalten, die von Prozeduren erzeugt werden	
		PROCEDURE_CATALOG	Name der Datenbank (string)
		PROCEDURE_SCHEMA	Name des Schemas (string)
		PROCEDURE_NAME	Name der Prozedur (string)
		COLUMN_NAME	Name der Spalte, die die Prozedur zurück gibt (string)
		COLUMN_GUID	GUID der Spalte (GUID)
		COLUMN_PROPID	Property-ID der Spalte (string)
		ROWSET_NUMBER	Fortlaufende Datensatznummer, wenn die Prozedur mehrere Datensätze erzeugt
		ORDINAL_POSITION	Ordnungsnummer der Spalte (long)
		IS_NULLABLE	TRUE, wenn die Spalte NULL sein darf (boolean)
		DATA_TYPE	Datentyp (DataType-Konstante) (integer)
		TYPE_GUID	GUID des Datentyps (GUID)

Schema	N	Beschreibung	
		CHARACTER_MAXIMUM_LENGTH	Maximale Breite der Spalte (long)
		CHARACTER_OCTET_LENGTH	Maximale Breite der Spalte in Bytes bei Text- oder Binärspalten (long)
		NUMERIC_PRECISION	Anzahl der Stellen vor dem Komma (integer)
		NUMERIC_SCALE	Anzahl der Stellen nach dem Komma (integer)
		DESCRIPTION	Beschreibung
adSchemaProcedureParameters	26	Informationen über die Parameter, die von Prozeduren benötigt werden	
		PROCEDURE_CATALOG	Name der Datenbank (string)
		PROCEDURE_SCHEMA	Name des Schemas (string)
		PROCEDURE_NAME	Name der Prozedur (string)
		PARAMETER_NAME	Name des Parameters, den die Prozedur erwartet (string)
		ORDINAL_POSITION	Ordnungsnummer des Parameters in der Parameterliste, beginnend mit 1 (integer)
		PARAMETER_TYPE	Typ des Parameters, kann Folgendes sein (integer): • DBPARAMTYPE_INPUT (1): Input • DBPARAMTYPE_INPUTOUTPUT (2): Input und Output • DBPARAMTYPE_OUTPUT (3): Nur Output • DBPARAMTYPE_RETURNVALUE (4): Rückgabewert
		PARAMETER_HASDEFAULT	TRUE, wenn Parameter einen Standardwert hat (boolean)
		PARAMETER_DEFAULT	Standardwert, der angenommen wird (string)
		IS_NULLABLE	TRUE, wenn die Spalte NULL sein darf (boolean)
		DATA_TYPE	Datentyp (DataType-Konstante) (integer)
		CHARACTER_MAXIMUM_LENGTH	Maximale Breite der Spalte (long)
		CHARACTER_OCTET_LENGTH	Maximale Breite der Spalte in Bytes bei Text- oder Binärspalten (long)
		NUMERIC_PRECISION	Anzahl der Stellen vor dem Komma (integer)

Schema	N	Beschreibung	
		NUMERIC_SCALE	Anzahl der Stellen nach dem Komma (integer)
		DESCRIPTION	Beschreibung
		TYPE_NAME	Providerabhängiger Datentyp
		LOCAL_TYPE_NAME	Providerabhängiger Datentyp in der lokalisierten (landesspezifischen) Version
adSchemaProcedures	16	Gibt Informationen über gespeicherte Prozeduren an	
		PROCEDURE_CATALOG	Name der Datenbank (string)
		PROCEDURE_SCHEMA	Name des Schemas (string)
		PROCEDURE_NAME	Name der Prozedur (string)
		PROCEDURE_TYPE	Typ der Prozedur bzgl. des Rückgabewerts
		PROCEDURE_DEFINITION	Definition der Prozedur
		DESCRIPTION	Beschreibung
		DATE_CREATED	Datum der Erzeugung
		DATE_MODIFIED	Datum der letzten Änderung
adSchemaProperties	37	Eigenschaften für jede Ebene einer Dimension	
		CATALOG_NAME	Name der Datenbank
		SCHEMA_NAME	Name des Schemas
		CUBE_NAME	Name des Cube
		DIMENSION_UNIQUE_NAME	Eindeutiger (interner) Name der Dimension
		HIERARCHY_UNIQUE_NAME	Name der Hierarchie
		LEVEL_UNIQUE_NAME	Eindeutiger (interner) Name der Hierarchie
		MEMBER_UNIQUE_NAME	Eindeutiger (interner) Name des Mitglieds
		PROPERTY_TYPE	Typ der Eigenschaft
		PROPERTY_NAME	Name der Eigenschaft
adSchemaProviderSpecific	-1	Providerabhängige Informationen	
adSchemaProviderTypes	22	Informationen über die vom Provider unterstützten Datentypen	
		TYPE_NAME	Name des Datentyps (vom Provider)

Schema	N	Beschreibung	
		DATA_TYPE	ADO-Typnummer
		COLUMN_SIZE	Spaltenbreite
		LITERAL_PREFIX	Literal, das in Zeichenketten zur Erkennung des Anfangs verwendet wird
		LITERAL_SUFFIX	Literal, das in Zeichenketten zur Erkennung des Endes verwendet wird
		CREATE_PARAMS	Zeichenkette mit der Angabe der zusätzlich benötigen Parameter. Für DECIMAL wird z.B. precision, scale angezeigt; es müssen also zusätzlich die Genauigkeit und Nachkommastellen angegeben werden.
		IS_NULLABLE	TRUE, wenn der Datentyp NULL werden kann
		CASE_SENSITIVE	TRUE, wenn es ein Zeichenkettentyp ist, der Groß- und Kleinbuchstaben unterscheiden kann.
		SEARCHABLE	TRUE, wenn Spalten dieses Typs durchsuchbar sind.
		UNSIGNED ATTRIBUTE	TRUE, wenn es sich um einen Typ ohne Vorzeichen handelt.
		FIXED_PREC_SCALE	TRUE, wenn die Genauigkeit feststeht.
		AUTO_UNIQUE_VALUE	TRUE, wenn der Typ in einer Spalte mit automatischer Erhöhung genutzt werden darf.
		LOCAL_TYPE_NAME	Lokalisierte Version des Typnamens
		MINIMUM_SCALE	Minimale Anzahl Stellen nach dem Komma
		MAXIMUM_SCALE	Maximale Anzahl Stellen nach dem Komma
		GUID	GUID des Typs
		TYPELIB	Bibliothek, in der eine Beschreibung steht
		VERSION	Version, providerabhängig, nur selten verfügbar
		IS_LONG	TRUE bei BLOB-Typen
		BEST_MATCH	TRUE, wenn dieser Typ die beste Übereinstimmung mit dem OLEDB-Datentyp im Feld DATA_TYPE ist

Schema	N	Beschreibung	
		IS_FIXEDLENGTH	TRUE, wenn der Typ eine feste Breite hat
adSchemaReferentialConstraints	9	Informationen über referentielle Einschränkungen	
		CONSTRAINT_CATALOG	Name der Datenbank
		CONSTRAINT_SCHEMA	Name des Schemas
		CONSTRAINT_NAME	Name der Einschränkung
		UNIQUE_CONSTRAINT_CATALOG	Eindeutiger (interner) Name der Datenbank
		UNIQUE_CONSTRAINT_SCHEMA	Eindeutiger (interner) Name des Schemas
		UNIQUE_CONSTRAINT_NAME	Eindeutiger (interner) Name der Einschränkung
		MATCH_OPTION	Bedingung, kann folgende Werte annehmen: • NONE. Keine Bedingung • PARTIAL. Teilweise Übereinstimmung • FULL. Vollständige Übereinstimmung
		UPDATE_RULE	Aktion für UPDATE oder »NO ACTION«, kann außerdem einen der folgenden Werte haben: • CASCADE. Übernahme des Vorwerts • SET NULL. Auf NULL setzen • SET DEFAULT. Auf Standardwert setzen
		DELETE_RULE	Aktion für DELETE oder »NO ACTION«, kann außerdem einen der folgenden Werte haben: • CASCADE. Übernahme des Vorwerts • SET NULL. Auf NULL setzen • SET DEFAULT. Auf Standardwert setzen
		DESCRIPTION	Beschreibung
adSchemaSchemata	17	Liste aller Datenbankobjekte	
		CATALOG_NAME	Name der Datenbank
		SCHEMA_NAME	Name des Schemas
		SCHEMA_OWNER	Besitzer

Schema	N	Beschreibung	
		DEFAULT_CHARCTER_SET_CATALOG	Datenbank mit dem Standardzeichensatz
		DEFAULT_CHARCTER_SET_SCHEMA	Schema mit dem Standardzeichensatz
		DEFAULT_CHARCTER_SET_NAME	Name des Standardzeichensatzes
adSchemaSQLLanguages	18	Angaben zum SQL-Dialekt des Providers	
		SQL_LANGUAGE_SOURCE	Standard, meist »ISO 9075«
		SQL_LANGUAGE_YEAR	Jahr des ANSI-Standards, z.B.: »1992« für ANSI-SQL-92
		SQL_LANGUAGE_CONFORMANCE	SQL-Niveau: • ENTRY • INTERMEDIATE • FULL
		SQL_LANGUAGE_INTEGRITY	YES, wenn Integritätsfunktionen vorhanden sind. Solche Funktionen prüfen die Integrität der Datenbank selbstständig.
		SQL_LANGUAGE_IMPLEMENTATION	NULL, wenn ISO 9075 kompatibel
		SQL_LANGUAGE_BINDING_STYLE	DIRECT
		SQL_LANGUAGE_PROGRAMMING_LANGUAGE	NULL
adSchemaStatistics	19	Statistische Informationen	
		TABLE_CATALOG	Name der Datenbank (string)
		TABLE_SCHEMA	Name des Schemas (string)
		TABLE_NAME	Name einer Tabelle (string)
		CARDINALITY	Anzahl der Reihen dieser Tabelle (bigint)
adSchemaTableConstraints	10	Einschränkungen der Tabelle	
		CONSTRAINT_CATALOG	Name der Datenbank (string)
		CONSTRAINT_SCHEMA	Name des Schemas (string)
		CONSTRAINT_NAME	Name der Einschränkung (string)
		TABLE_CATALOG	Datenbank, in der die Tabelle definiert ist
		TABLE_SCHEMA	Schema, wo die Tabelle definiert ist

Schema	N	Beschreibung	
		TABLE_NAME	Name der Tabelle
		CONSTRAINT_TYPE	Typ der Einschränkung
		IS_DEFERRABLE	TRUE, wenn ableitbar
		INITIALLY_DEFFERED	TRUE, wenn zur Initalisierung ableitbar
		DESCRIPTION	Beschreibung
adSchemaTable-Privileges	14	Rechte in Bezug auf bestimmte Tabellen	
		GRANTOR	User, der das Recht vergeben hat
		GRANTEE	User, der das Recht bekommen hat
		TABLE_CATALOG	Name der Datenbank (string)
		TABLE_SCHEMA	Name des Schemas (string)
		TABLE_NAME	Name einer Tabelle (string)
		PRIVILEGE_TYPE	Typ des Rechts: • SELECT • DELETE • UPDATE • INSERT • REFERENCES
		IS_GRANTABLE	FALSE, wenn das Recht mit der Option WITH GRANT OPTION gesetzt wurde. Ist TRUE, wenn der Nutzer das Recht weiterreichen darf.
adSchemaTables	20	Für den Nutzer zugängliche Tabellen und Sichten	
		TABLE_CATALOG	Name der Datenbank (string)
		TABLE_SCHEMA	Name des Schemas (string)
		TABLE_NAME	Name einer Tabelle (string)
		TABLE_TYPE	Typ der Tabelle (string): • ALIAS. Alias • TABLE. Normale Tabelle • SYNONYM. Synonym • SYSTEM TABLE. Systemtabelle • VIEW. Sicht • GLOBAL TEMPORARY. Temporär, global verfügbar • LOCAL TEMPORARY. Temporär, lokal
		TABLE_GUID	GUID der Tabelle
		DESCRIPTION	Beschreibung
		TABLE_PROPID	Property-ID

Schema	N	Beschreibung	
		DATE_CREATED	Datum der Erzeugung
		DATE_MODIFIED	Datum der letzten Änderung
adSchemaTranslations	21	Informationen über Übersetzungen	
		TRANSLATION_CATALOG	Name der Datenbank
		TRANSLATION_SCHEMA	Name des Schemas
		TRANSLATION_NAME	Name der Übersetzung
		SOURCE_CHARACTER_SET_CATALOG	Zeichensatz der Quelldatenbank
		SOURCE_CHARACTER_SET_SCHEMA	Zeichensatz des Quelldatenschemas
		SOURCE_CHARACTER_SET_NAME	Name des Zeichensatzes der Quelle
		TARGET_CHARACTER_SET_CATALOG	Zeichensatz der Zieldatenbank
		TARGET_CHARACTER_SET_SCHEMA	Zeichensatz der Zielschemas
		TARGET_CHARACTER_SET_NAME	Name des Zeichensatzes des Ziels
adSchema-Trustees	39	Reserviert	
adSchema-UsagePrivileges	15	Informationen über Rechte	
		GRANTOR	User, der das Recht vergeben hat
		GRANTEE	User, der das Recht bekommen hat
		OBJECT_CATALOG	Name der Datenbank (string)
		OBJECT_SCHEMA	Name des Schemas (string)
		OBJECT_NAME	Name eines Objekts (string)
		OBJECT_TYPE	Objekt-Typ, kann einer der folgenden Werte sein: • DOMAIN. Domäne • CHARACTER_SET. Zeichensatz • COLLATION. Sortierbestimmung • TRANSLATION. Übersetzung
		PRIVILEGE_TYPE	Typ des Rechts: • SELECT • DELETE • UPDATE • INSERT • REFERENCES

Schema	N	Beschreibung	
		IS_GRANTABLE	FALSE, wenn das Recht mit der Option WITH GRANT OPTION gesetzt wurde. Ist TRUE, wenn der Nutzer das Recht weiterreichen darf.
adSchemaView-ColumnUsage	24	Zeigt Spalten an, von denen Sichten abhängen	
		VIEW_CATALOG	Name der Datenbank, in der die Sicht definiert ist (string)
		VIEW_SCHEMA	Name des Schemas (string)
		VIEW_NAME	Name der Sicht (string)
		TABLE_CATALOG	Name der Datenbank der zugrunde liegenden Tabelle (string)
		TABLE_SCHEMA	Name des Schemas der zugrunde liegenden Tabelle (string)
		TABLE_NAME	Name einer Tabelle der zugrunde liegenden Tabelle (string)
		COLUMN_NAME	Name der Spalte
		COLUMN_GUID	GUID der Spalte
		COLUMN_PROPID	Property-ID der Spalte
adSchemaViews	23	Zeigt alle Sichten an	
		TABLE_CATALOG	Name der Datenbank der zugrunde liegenden Tabelle (string)
		TABLE_SCHEMA	Name des Schemas der zugrunde liegenden Tabelle (string)
		TABLE_NAME	Name einer Tabelle der zugrunde liegenden Tabelle (string)
		VIEW_DEFINITION	Definition der Sicht (Abfragezeichenkette)
		CHECK_OPTION	TRUE, wenn nur lokale Updates geprüft werden (entspricht der Angabe von CHECK OPTION)
		IS_UPDATABLE	TRUE, wenn die Sicht auch Update akzeptiert
		DESCRIPTION	Beschreibung
		DATE_CREATED	Datum der Erstellung
		DATE_MODIFIED	Datum der letzten Änderung
adSchemaView-TableUsage	25	Zeigt Informationen über Tabellen an, von denen Sichten abhängen	
		VIEW_CATALOG	Name der Datenbank, in der die Sicht definiert ist (string)
		VIEW_SCHEMA	Name des Schemas (string)

Schema	N	Beschreibung	
		VIEW_NAME	Name der Sicht (string)
		TABLE_CATALOG	Name der Datenbank der zugrunde liegenden Tabelle (string)
		TABLE_SCHEMA	Name des Schemas der zugrunde liegenden Tabelle (string)
		TABLE_NAME	Name einer Tabelle der zugrunde liegenden Tabelle (string)

A.3 Datentypen

Numerischer Wert	Konstante
0	Empty
16	TinyInt
2	SmallInt
3	Integer
20	BigInt
17	UnsignedTinyInt
18	UnsignedSmallInt
19	UnsignedInt
21	UnsignedBigInt
4	Single
5	Double
6	Currency
14	Decimal
131	Numeric
11	Boolean
10	Error
132	UserDefined
12	Variant
9	IDispatch
13	IUnknown
72	GUID
7	Date
133	DBDate
134	DBTime
135	DBTimeStamp

Numerischer Wert	Konstante
8	BSTR
129	Char
200	VarChar
201	LongVarChar
130	WChar
202	VarWChar
203	LongVarWChar
128	Binary
204	VarBinary
205	LongVarBinary

A.4 Nummerische Werte der Konstanten

Konstante	Wert	Beschreibung
CursorTypeEnum		
adOpenForwardOnly	0	Standardwert. Öffnet einen schnellen Vorwärts-Zeiger.
adOpenKeyset	1	Öffnet einen KeySet-Zeiger.
adOpenDynamic	2	Öffnet einen dynamischen Zeiger.
adOpenStatic	3	Öffnet einen statischen Zeiger.
CursorOptionEnum		
adHoldRecords	&H0000 0100 = 256	Weitere Datensätze können gelesen werden, ohne ausstehende Änderungen bestätigen zu müssen.
adMovePrevious	&H0000 0200 = 512	MovePrevious, MoveFirst und Move können verwendet werden
adAddNew	&H0100 0400 = 16 778 240	AddNew ist erlaubt
adDelete	&H0100 0800 = 16 779 264	Delete ist erlaubt
adUpdate	&H0100 8000 = 16 809 984	Update ist erlaubt
adBookmark	&H0000 2000 = 8 192	Lesezeichen werden unterstützt
adApproxPosition	&H0000 4000 = 16 384	AbsolutePosition und AbsolutePage werden unterstützt
adUpdateBatch	&H0001 0000 = 65 535	UpdateBatch und CancelBatch werden unterstützt

Konstante	Wert	Beschreibung
`adResync`	&H0002 0000 = 131 072	`Resync` wird unterstützt.
`adNotify`	&H0004 0000 = 262 144	Benachrichtigungen werden unterstützt.
`adFind`	&H0008 0000 = 524 288	`Find` kann verwendet werden.
`adSeek`	&H0040 0000 = 4 194 304	`Seek` kann verwendet werden.
`adIndex`	&H0080 0000 = 8 388 608	`Index`-Eigenschaft kann verwendet werden.
LockTypeEnum		
`adLockReadOnly`	1	Standardwert. Nur Lesen
`adLockPessimistic`	2	Pessimistisch, der Provider verriegelt sofort.
`adLockOptimistic`	3	Optimistisch, der Provider verriegelt nur, wenn `Update` verwendet wird.
`adLockBatchOptimistic`	4	Optimistisch, im Batch-Mode.
ExecuteOptionEnum		
`adAsyncExecute`	&H0000 0010 = 16	Ausführung asynchron
`adAsyncFetch`	&H0000 0020 = 32	Abrufen der Datensätze asynchron
`adAsyncFetchNonBlocking`	&H0000 0040 = 64	Abrufen der Datensätze asynchron, untergeordnete Operationen werden nicht blokkiert.
`adExecuteNoRecords`	&H0000 0080 = 128	Kennzeichnet, dass das Kommando keine Datensätze zurückgibt.
ConnectOptionEnum		
`adAsyncConnect`	&H0000 0010 = 16	Öffnet die Verbindung asynchron.
ObjectStateEnum		
`adStateClosed`	&H0000 0000 = 0	Standardwert. Objekt ist geschlossen.
`adStateOpen`	&H0000 0001 = 1	Objekt ist offen.
`adStateConnecting`	&H0000 0002 = 2	Objekt ist verbunden.
`adStateExecuting`	&H0000 0004 = 4	Objekt führt gerade ein Kommando aus.
`adStateFetching`	&H0000 0008 = 8	Datensätze werden gerade geholt.

Konstante	Wert	Beschreibung
CursorLocationEnum		
adUseServer	2	Standardwert. Zeiger serverseitig
adUseClient	3	Zeiger im Client
DataTypeEnum		
adEmpty	0	Leer
adTinyInt	16	1 Byte Integer mit Vorzeichen
adSmallInt	2	2 Byte Integer mit Vorzeichen
adInteger	3	4 Byte Integer mit Vorzeichen
adBigInt	20	8 Byte Integer mit Vorzeichen
adUnsignedTinyInt	17	1 Byte Integer ohne Vorzeichen
adUnsignedSmallInt	18	2 Byte Integer ohne Vorzeichen
adUnsignedInt	19	4 Byte Integer ohne Vorzeichen
adUnsignedBigInt	21	8 Byte Integer ohne Vorzeichen
adSingle	4	Gleitkomma, einfache Genauigkeit
adDouble	5	Gleitkomma, doppelte Genauigkeit
adCurrency	6	Währung
adDecimal	14	Exakter numerischer Wert
adNumeric	131	Exakter numerischer Wert
adBoolean	11	Boolescher Wert
adError	10	Fehler
adUserDefined	132	Benutzerdefiniert
adVariant	12	Variabel
adIDispatch	9	IDispatch (OLE)
adIUnknown	13	Unbekannt
adGUID	72	GUID
adDate	7	Datum, eine Gleitkommanzahl, bei der der ganzzahlige Teil die Tage seit dem 30.12.1899 angibt; der Bruchteil ist der Teil des Tages.
adDBDate	133	Datumswert im Format YYYYMMTT
adDBTime	134	Zeit im Format HHMMSS
adDBTimeStamp	135	Zeitstempel YYYYMMTTHHMMSS,milliardstel
adBSTR	8	Zeichenkette, mit NULL begrenzt
adChar	129	Zeichen
adVarChar	200	Zeichenkette (typisch <= 255)

Konstante	Wert	Beschreibung
adLongVarChar	201	Zeichenkette (typisch <= 2 GByte)
adWChar	130	Zeichen, 16 Bit
adVarWChar	202	Zeichenkette (typisch <= 255) , 16 Bit
adLongVarWChar	203	Zeichenkette (typisch <= 2 GByte) , 16 Bit
adBinary	128	Binärwert, Byte
adVarBinary	204	Binärwert, variabel <= 255 Byte
adLongVarBinary	205	Binärwert, variabel <= 2 Gbyte
adChapter	136	Chapter
adFileTime	64	Datum einer Datei
adPropVariant	138	Varianter Wert, der nicht automatisch konvertiert werden kann
adVarNumeric	139	Exakte numerische Zahl mit variabler Länge
adArray	&H2000	Array (Datenfeld)
FieldAttributeEnum		
adFldMayDefer	&H0000 0002	Werte werden nur dann geholt, wenn der Datensatz benötigt wird.
adFldUpdatable	&H0000 0004	Feld ist beschreibbar.
adFldUnknownUpdatable	&H0000 0008	Es ist nicht bekannt, ob das Feld beschreibbar ist.
adFldFixed	&H0000 0010	Feld hat feste Breite.
adFldIsNullable	&H0000 0020	Feld kann NULL werden.
adFldMayBeNull	&H0000 0040	Feld kann möglicherweise NULL enthalten.
adFldLong	&H0000 0080	Feld ist ein BLOB-Feld.
adFldRowID	&H0000 0100	Feld enthält eine ID.
adFldRowVersion	&H0000 0200	Feld enthält Versionsinformationen für Row-ID.
adFldCacheDeferred	&H0000 1000	Provider wird nachfolgende Abrufe aus dem Cache laden.
adFldIsChapter	&H0000 2000	Feld ist Chapter.
adFldNegativeScale	&H0000 4000	Feld hat negativen Wert.
adFldKeyColumn	&H0000 8000	Feld ist ein Schlüssel.
adFldIsRowURL	&H0001 0000	Feld ist eine URL.
adFldIsDefaultStream	&H0002 0000	Feld ist der Standard-Stream
adFldIsCollection	&H0004 0000	Feld ist eine Kollektion

Konstante	Wert	Beschreibung
EditModeEnum		
adEditNone	&H0000 = 0	Datensatz wird nicht bearbeitet
adEditInProgress	&H0001 = 1	Datensatz wird gerade bearbeitet
adEditAdd	&H0002 = 2	Datensatz wird gerade mit AddNew erzeugt
adEditDelete	&H0004 = 4	Datensatz wird gerade mit Delete gelöscht
RecordStatusEnum		
adRecOK	&H0000 0000 = 0	Aktualisierung war erfolgreich.
adRecNew	&H0000 0001 = 1	Datensatz ist neu.
adRecModified	&H0000 0002 = 2	Datensatz wurde geändert.
adRecDeleted	&H0000 0004 = 4	Datensatz wurde gelöscht.
adRecUnmodified	&H0000 0008 = 8	Datensatz unverändert.
adRecInvalid	&H0000 0010 = 16	Datensatz nicht gespeichert, weil Lesezeichen nicht stimmt.
adRecMultipleChanges	&H0000 0040 = 64	Datensatz nicht gespeichert, weil Änderungen mehrere Datensätze betroffen hätten.
adRecPendingChanges	&H0000 0080 = 128	Datensatz nicht gespeichert, weil er zu einer noch ausstehenden Insert-Anweisung gehört.
adRecCanceled	&H0000 0100 = 256	Datensatz nicht gespeichert, weil Operation abgebrochen wurde.
adRecCantRelease	&H0000 0400 = 1 024	Datensatz nicht gespeichert, weil eine Verriegelung existiert.
adRecConcurrencyViolation	&H0000 0800 = 2 048	Datensatz nicht gespeichert, weil optimistische Concurrency aktiv war.
adRecIntegrityViolation	&H0000 1000 = 4 096	Datensatz nicht gespeichert, weil Integritätsregeln verletzt wurden.
adRecMaxChangesExceeded	&H0000 2000 = 8 192	Datensatz nicht gespeichert, weil zu viele offene Änderungen vorhanden waren.
adRecObjectOpen	&H0000 4000 = 16 384	Datensatz nicht gespeichert, weil er in Konflikt mit einem anderen offenen Objekt stand.
adRecOutOfMemory	&H0000 8000 = 32 768	Datensatz nicht gespeichert, weil nicht genug Speicher zur Verfügung stand.
adRecPermissionDenied	&H0001 0000 = 65 535	Datensatz nicht gespeichert, weil Rechte fehlten.

Konstante	Wert	Beschreibung
adRecSchemaViolation	&H0002 0000 = 131 072	Datensatz nicht gespeichert, weil die Struktur der zugrundeliegenden Datenbank geändert wurde.
adRecDBDeleted	&H0004 0000 = 262 144	Datensatz wurde bereits gelöscht.
GetRowsOptionEnum		
adGetRowsRest	-1	Auch die übrigen Reihen des Datensatzobjekts werden gelesen.
PositionEnum		
adPosUnknown	-1	Die Position konnte nicht ermittelt werden.
adPosBOF	-2	Der Zeiger steht auf BOF.
adPosEOF	-3	Der Zeiger steht auf EOF.
BookmarkEnum		
adBookmarkCurrent	0	Aktueller Datensatz
adBookmarkFirst	1	Erster Datensatz
adBookmarkLast	2	Letzter Datensatz
MarshalOptionsEnum		
adMarshalAll	0	Alle Reihen werden zum Server zurückgesendet.
adMarshalModifiedOnly	1	Nur die gänderten Reihen werden gesendet.
AffectEnum		
adAffectCurrent	1	Operation betrifft den aktuellen Datensatz.
adAffectGroup	2	Operation betrifft die gefilterte Gruppe.
adAffectAll	3	Operation betrifft alle Datensätze.
adAffectAllChapters	4	Operation betrifft alle untergeordneten Datensätze.
ResyncEnum		
adResyncUnderlyingValues	1	Daten werden nicht überschrieben und ausstehende Änderungen werden nicht abgebrochen.
adResyncAllValues	2	Daten werden überschrieben und ausstehende Änderungen werden abgebrochen.
CompareEnum		
adCompareLessThan	0	Das erste Lesezeichen liegt hinter dem zweiten.

Konstante	Wert	Beschreibung
adCompareEqual	1	Die Lesezeichen sind gleich.
adCompareGreaterThan	2	Das erste Lesezeichen liegt nach dem zweiten.
adCompareNotEqual	3	Die Lesezeichen sind nicht gleich und nicht sortiert.
adCompareNotComparable	4	Die Lesezeichen können nicht verglichen werden.
FilterGroupEnum		
adFilterNone	0	Kein Filter, Filter entfernen.
adFilterPendingRecords	1	Zeigt alle Reihen, die geändert, aber noch nicht zum Server gesendet wurden.
adFilterAffectedRecords	2	Zeigt die Reihen, die bei der letzten Operation mit Delete, Resnch, UpdateBatch oder CancelBatch geändert wurden.
adFilterFetchedRecords	3	Zeigt alle Reihen im aktuellen Cache.
adFilterConflictingRecords	5	Zeigt alle Reihen, die auf Grund von Konlikten beim letzten Update nicht geschrieben wurden.
SearchDirectionEnum		
adSearchForward	1	Vom aktuellen Datensatz vorwärts suchen
adSearchBackward	-1	Vom aktuellen Datensatz rückwärts suchen
PersistFormatEnum		
adPersistADTG	0	Speichert Datensatz im internen ADTG-Format
adPersistXML	1	Speichert Datensatz als XML inkl. Schema
StringFormatEnum		
adClipString	2	Formatierung der Ausgabe von GetString
ConnectPromptEnum		
adPromptAlways	1	Immer Verbindungsinformationen abfragen
adPromptComplete	2	Nur abfragen, wenn Informationen fehlen

Konstante	Wert	Beschreibung
adPromptCompleteRequired	3	Nur abfragen, wenn Informationen fehlen. Dabei werden nur die möglichen Optionen angezeigt.
adPromptNever	4	Nie fragen. Dies ist in ASP der einzig mögliche Wert, da Dialogfelder zur Abfrage der Daten nicht erlaubt sind.
ConnectModeEnum		
adModeUnknown	0	Zugriffsrechte wurden noch nicht gesetzt oder sind unbekannt.
adModeRead	1	Nur-Lesen
adModeWrite	2	Nur-Schreiben
adModeReadWrite	3	Lesen und Schreiben
adModeShareDenyRead	4	Verhindert, dass andere die Verbindung lesend öffnen
adModeShareDenyWrite	8	Verhindert, dass andere die Verbindung schreibend öffnen
adModeShareExclusive	12	Verhindert, dass andere die Verbindung öffnen
adModeShareDenyNone	16	Verhindert, dass andere die Verbindung mit irgendwelchen Rechten öffnen
adModeRecursive	32	Weitere Restriktionen in Verbindung mit den Deny-Optionen
RecordCreateOptionsEnum		
adCreateNonCollection	&H00000000 = 0	Erzeugt einen neuen Datensatz zu der URL
adCreateCollection	&H00002000 = 8 192	Erzeugt eine neue Struktur unter der URL
adOpenIfExists	&H02000000 = 33 554 432	Öffnet ein Dokument unter der URL, wenn es existiert
adCreateOverwrite	&H04000000 = 67 108 864	Überschreibt ein vorhandenes Dokument unter der URL
adCreateStructDoc	&H80000000 = -2 147 483 648	Erzeugt ein neues strukturiertes Dokument unter der URL
adFailIfNotExists	-1	Fehler, wenn die URL nicht existiert
RecordOpenOptionsEnum		
adOpenRecordUnspecified	-1	Keine Angabe

Konstante	Wert	Beschreibung
adOpenAsync	&H00001000 = 4 069	Öffnet den Datensatz asynchron
adDelayFetchStream	&H00004000 = 16 384	Verzögert die Übertragung des Streams bis er angefordert wird
adDelayFetchFields	&H00008000 = 32 768	Verzögert die Übertragung der Felder, bis sie angefordert werden
adOpenSource	&H00800000 = 8 388 608	Öffnet das Dokument unter der URL, anstatt es auszuführen
IsolationLevelEnum		
adXactUnspecified	&HFFFF FFFF = -1	Der Provider verwendet einen anderen Wert als angegeben, dieser konnte aber nicht erkannt werden.
adXactChaos	&H0000 0010 = 16	Ausstehende Änderungen können von höheren Niveaus nicht überschrieben werden.
adXactReadUncommitted	&H0000 0100 = 256	Unbestätigte Änderungen einer Transaktion sind in einer anderen sichtbar.
adXactBrowse	&H0000 0100 = 256	
adXactCursorStability	&H0000 1000 = 4 096	Änderungen einer Transaktion sind erst dann für andere sichtbar, wenn diese bestätigt wurden.
adXactReadCommitted	&H0000 1000 = 4 096	
adXactRepeatableRead	&H0001 0000 = 65 536	Änderungen anderer Transaktionen sind nicht sichtbar, aber Requery zeigt neue Datensätze an.
adXactSerializable	&H0010 0000 = 1 048 576	Alle Transaktion sind voneinander isoliert.
adXactIsolated	&H0010 0000 = 1 048 576	
XactAttributeEnum		
adXactCommitRetaining	&H0002 0000 = 131 072	Nach der Bestätigung einer Transaktion wird automatisch eine neue gestartet.
adXactAbortRetaining	&H0004 0000 = 262 144	Nach der Ablehnung einer Transaktion wird automatisch eine neue gestartet.
adXactAsynchPhaseOne	&H0008 0000 = 524 288	Asynchrone Bestätigungen
adXactSynchPhaseOne	&H0010 0000 = 1 048 576	Synchrone Bestätigungen

Konstante	Wert	Beschreibung
PropertyAttributesEnum		
adPropNotSupported	&H0000 = 0	Eigenschaft wird nicht unterstützt.
adPropRequired	&H0001 = 1	Eigenschaft muss gesetzt werden, bevor die Datenquelle initialisiert wird.
adPropOptional	&H0002 = 2	Eigenschaft ist optional.
adPropRead	&H0200 = 512	Eigenschaft ist lesbar.
adPropWrite	&H0400 = 1 024	Eigenschaft kann gesetzt werden.
ErrorValueEnum		
adErrProviderFailed	&HBB8 = 3 000	Provider konnte Aktion nicht ausführen.
adErrInvalidArgument	&HBB9 = 3 001	Argumente waren falsch (Anzahl, Datentyp, …).
adErrOpeningFile	&HBBA = 3 002	Fehler beim Öffnen einer Datei
adErrReadFile	&HBBB = 3 003	Fehler beim Lesen aus einer Datei
adErrWriteFile	&HBBC = 3 004	Fehler beim Schreiben in eine Datei
adErrNoCurrentRecord	&HBCD = 3 021	EOF oder BOF sind TRUE. Die Operation benötigt aber einen gültigen Datensatz.
adErrIllegal-Operation	&HC93 = 3 219	Die Operation ist hier nicht erlaubt.
adErrCantChange-Provider	&HC94 = 3 220	Der Provider kann während der Operation nicht gewechselt werden.
adErrInTransaction	&HCAE = 3 246	Während einer Transaktion kann das Verbindungsobjekt nicht geschlossen werden.
adErrFeatureNotAvai-lable	&HCB3 = 3 251	Der Provider unterstützt diese Aktion nicht.
adErrItemNotFound	&HCC1 = 3 265	ADO kann das Objekt in der Kollektion nicht finden.
adErrObjectIn-Collection	&HD27 = 3 367	Objekt kann der Kollektion nicht hinzugefügt werden, weil es bereits existiert.
adErrObjectNotSet	&HD5C = 3 420	Die Referenz in der Kollektion zeigt nicht mehr auf ein gültiges Objekt.
adErrDataConversion	&HD5D = 3 421	Falscher Datentyp in diesem Kontext

Konstante	Wert	Beschreibung
adErrObjectClosed	&HE78 = 3 704	Operation ist nicht erlaubt, wenn das Objekt geschlossen ist.
adErrObjectOpen	&HE79 = 3 705	Operation ist nicht erlaubt, wenn das Objekt offen ist.
adErrProviderNot-Found	&HE7A = 3 706	ADO kann den Provider nicht finden.
adErrBoundToCommand	&HE7B = 3 707	ActiveConnection kann nicht mit einem Command-Objekt belegt werden
adErrInvalidParam-Info	&HE7C = 3 708	Falsch definiertes Parameter-Objekt
adErrInvalid-Connection	&HE7D = 3 709	Die Operation wurde mit einem geschlossenen oder ungültigen Verbindungsobjekt ausgeführt
adErrNotReentrant	&HE7E = 3 710	Während eines Ereignisses kann die Operation nicht erneut ausgeführt werden.
adErrStillExecuting	&HE7F = 3 711	Die Operation kann nicht während einer anderen asynchronen Operation ausgeführt werden.
adErrOperation-Cancelled	&HE80 = 3 712	Die Operation wurde vom Nutzer abgebrochen.
adErrStillConnecting	&HE81 = 3 713	Die Operation kann nicht während einer anderen asynchronen Operation ausgeführt werden.
adErrInvalid-Transaction	&HE82 = 3 714	Die Transaktion ist ungültig.
adErrNotExecuting	&HE83 = 3 715	Die Operation wird nicht ausgeführt.
adErrUnsafeOperation	&HE84 = 3 716	Die Operation ist nicht sicher.
adwrnSecurityDialog	&HE85 = 3 717	Dialog (in ASP nicht möglich): Daten werden aus einer anderen Domain geholt. Möchten Sie das erlauben?
adwrnSecurityDialog-Header	&HE86 = 3 718	Dialog (in ASP nicht möglich): Daten werden aus einer anderen Domain geholt. Möchten Sie das erlauben?
adErrIntegrity-Violation	&HE87 = 3 719	Aktion fehlgeschlagen, wegen einer Verletzung der Datenintegrität.
adErrPermission-Denied	&HE88 = 3 720	Aktion fehlgeschlagen, wegen mangelnder Zugriffsrechte

Konstante	Wert	Beschreibung
adErrDataOverflow	&HE89 = 3 721	Daten waren zu groß für das Feld.
adErrSchemaViolation	&HE8A = 3 722	Daten standen in Konflikt mit Einschränkungen.
adErrSignMismatch	&HE8B = 3 723	Daten konvertieren wegen Vorzeichen fehlgeschlagen.
adErrCantConvertvalue	&HE8C = 3 724	Daten konvertieren wegen eines anderen Fehlers fehlgeschlagen.
adErrCantCreate	&HE8D = 3 725	Datentyp des Felds ist unbekannt, deshalb konnte der Wert nicht gelesen oder geschrieben werden.
adErrColumnNotOnThisRow	&HE8E = 3 726	Das Feld ist in der Reihe nicht enthalten.
adErrURLDoesNotExist	&HE8F = 3 727	Die URL existiert nicht.
adErrTreePermissionDenied	&HE90 = 3 728	Keine ausreichenden Rechte zum Zugriff auf das Verzeichnis oder Unterverzeichnis.
adErrInvalidURL	&HE91 = 3 729	URL enthält ungültige Zeichen.
adErrResourceLocked	&HE92 = 3 730	URL von anderem Prozess blokkiert.
adErrResourceExists	&HE93 = 3 731	Ressource unter der URL existiert bereits.
adErrCannotComplete	&HE94 = 3 732	Aktion kann nicht ausgeführt werden
adErrVolumeNotFound	&HE95 = 3 733	Speichergerät nicht gefunden
adErrOutOfSpace	&HE96 = 3 734	Kein freier Speicherplatz mehr
adErrResourceOutOfScope	&HE97 = 3 735	URL ist außerhalb des Sichtbereiches des Datensatzes.
adErrUnavailable	&HE98 = 3 736	Operation wurde nicht ausgeführt und ein Status ist nicht verfügbar
adErrURLNamedRowDoesNotExist	&HE99 = 3 737	Die URL im benannten Record existiert nicht
adErrDelResOutOfScope	&HE9A = 3 738	Die URL kann nicht gelöscht werden, weil die Ressource außerhalb des Sichtbereichs liegt.
adErrPropInvalidColumn	&HE9B = 3 739	Die Eigenschaft ist für dieses Feld nicht gültig
adErrPropInvalidOption	&HE9C = 3 740	Das Attribut der Eigenschaft ist nicht gültig
adErrPropInvalidValue	&HE9D = 3 741	Der Eigenschaftswert ist nicht gültig

Konstante	Wert	Beschreibung
adErrPropConflicting	&HE9E = 3 742	Die Eigenschaft steht in Konflikt mit einer anderen.
adErrPropNot-AllSettable	&HE9F = 3 743	Die Eigenschaft kann nicht gesetzt werden (Nur-Lese-Eigenschaft)
adErrPropNotSet	&HEA0 = 3 744	Der optionale Wert wurde nicht gesetzt
adErrPropNotSettable	&HEA1 = 3 745	Die Eigenschaft kann nur gelesen werden und der Wert ist nicht gesetzt.
adErrPropNot-Supported	&HEA2 = 3 746	Die Eigenschaft wird vom Provider nicht unterstützt.
adErrCatalogNotSet	&HEA3 = 3 747	Aktion konnte nicht abgeschlossen werden, weil ParentCatalog nicht gesetzt wurde.
adErrCantChange-Connection	&HEA4 = 3 748	Die Verbindung kann nicht geändert werden.
adErrFieldsUpdate-Failed	&HEA5 = 3 749	Update ist für die Kollektion fehlgeschlagen.
adErrDenyNot-Supported	&HEA6 = 3 750	Der Provider unterstützt keine verteilten Restriktionen.
adErrDenyTypeNot-Supported	&HEA7 = 3 751	Der Provider unterstützt nicht diese Art von verteilten Restriktionen.
ParameterAttributesEnum		
adParamSigned	&H0010 = 16	Der Parameter akzeptiert Werte mit Vorzeichen.
adParamNullable	&H0040 = 64	Der Parameter akzeptiert NULL.
adParamLong	&H0080 = 128	Der Parameter akzeptiert große Binärwerte.
ParameterDirectionEnum		
adParamUnknown	&H0000 = 0	Parameterrichtung ist nicht bekannt.
adParamInput	&H0001 = 1	Eingabeparameter (Input f. gesp. Prozeduren)
adParamOutput	&H0002 = 2	Ausgabeparameter (Output f. gesp. Prozeduren)
adParamInputOutput	&H0003 = 3	Ein- und Ausgabeparameter (Input/Output)
adParamReturnValue	&H0004 = 4	Rückgabewert (Return f. gesp. Prozeduren)

Konstante	Wert	Beschreibung
CommandTypeEnum		
adCmdUnknown	&H0008 = 8	Inhalt des Kommandos ist unbekannt
adCmdText	&H0001 = 1	Inhalt des Kommandos ist Text
adCmdTable	&H0002 = 2	Inhalt des Kommandos ist Tabelle
adCmdStoredProc	&H0004 = 4	Inhalt des Kommandos ist eine gespeicherte Prozedur
adCmdFile	&H0100 = 256	Inhalt des Kommandos ist ein Dateiname
adCmdTableDirect	&H0200 = 512	Inhalt des Kommandos ist eine direkte Abfrage
EventStatusEnum (Hinweis: Ereignisse werden in ASP nicht unterstützt)		
adStatusOK	&H0000001	Operation wurde erfolgreich ausgeführt.
adStatusErrors-Occurred	&H0000002	Will hat die Ausführung abgebrochen.
adStatusCantDeny	&H0000003	Will konnte nicht abbrechen.
adStatusCancel	&H0000004	Operation wurde abgebrochen.
adStatusUnwante-dEvent	&H0000005	Ereignis wurde nicht mehr erwartet.
EventReasonEnum (Hinweis: Ereignisse werden in ASP nicht unterstützt)		
adRsnAddNew	1	Ein neuer Datensatz wurde hinzugefügt.
adRsnDelete	2	Datensatz wurde gelöscht.
adRsnUpdate	3	Datensatz wurde geändert.
adRsnUndoUpdate	4	Änderung wurde abgebrochen.
adRsnUndoAddNew	5	Hinzufügen wurde abgebrochen.
adRsnUndoDelete	6	Löschen wurde abgebrochen.
adRsnRequery	7	Requery wurde ausgeführt.
adRsnResynch	8	Resynch wurde ausgeführt.
adRsnClose	9	Objekt wurde geschlossen.
adRsnMove	10	Move wurde ausgeführt.
adRsnFirstChange	11	Der Datensatz wurde das erste Mal verändert.
adRsnMoveFirst	12	MoveFirst wurde ausgeführt.
adRsnMoveNext	13	MoveNext wurde ausgeführt.
adRsnMovePrevious	14	MovePrevious wurde ausgeführt.
adRsnMoveLast	15	MoveLast wurde ausgeführt.

Konstante	Wert	Beschreibung
SchemaEnum		
adSchemaProvider-Specific	-1	Providerspezifische Informationen
adSchemaAsserts	0	Assert-Informationen
adSchemaCatalogs	1	Katalog-Informationen
adSchemaCharacter-Sets	2	Informationen über den Zeichensatz
adSchemaCollations	3	Informationen über Sortier-bedingungen
adSchemaColumns	4	Spalten-Informationen
adSchemaCheck-Constraints	5	Informationen über Einschränkungen
adSchemaConstraint-ColumnUsage	6	Informationen über die Verwen-dung von Einschränkungen in Spalten
adSchemaConstraint-TableUsage	7	Informationen über die Verwen-dung von Einschränkungen in Tabellen
adSchemaKeyColumn-Usage	8	Informationen über Schlüsselspalten
adSchemaReferential-Constraints	9	Informationen über referenzielle Einschränkungen
adSchemaTable-Constraints	10	Informationen über Tabelleneinschränkungen
adSchemaColumns-DomainUsage	11	Informationen über Verwendung von Spalten innerhalb von Domänen
adSchemaIndexes	12	Informationen über Indizes
adSchemaColumn-Privileges	13	Informationen über Spalten-Rechte
adSchemaTable-Privileges	14	Informationen über Tabellen-Rechte
adSchemaUsage-Privileges	15	Informationen über Verwendungs-Rechte
adSchemaProcedures	16	Informationen über Prozeduren
adSchemaSchemata	17	Informationen über Schemas
adSchemaSQLLanguages	18	Informationen über die SQL-Sprache
adSchemaStatistics	19	Statistische Informationen
adSchemaTables	20	Informationen über Tabellen

Konstante	Wert	Beschreibung
adSchemaTranslations	21	Informationen über Zeichensatzübersetzungen
adSchemaProviderTypes	22	Informationen über Providertypen
adSchemaViews	23	Informationen über Sichten
adSchemaViewColumnUsage	24	Informationen über die Verwendung von Spalten in Sichten
adSchemaViewTableUsage	25	Informationen über die Verwendung von Tabellen in Sichten
adSchemaProcedureParameters	26	Informationen über die Parameter gespeicherter Prozeduren
adSchemaForeignKeys	27	Informationen über Fremdschlüssel
adSchemaPrimaryKeys	28	Informationen über Primärschlüssel
adSchemaProcedureColumns	29	Informationen über Spalten in Prozeduren
adSchemaDBInfoKeywords	30	Liste der Schlüsselwörter des Providers
adSchemaDBInfoLiterals	31	Liste der Literale des Providers
adSchemaCubes	32	Informationen über Cubes einer MD-Datenbank
adSchemaDimensions	33	Informationen über Dimensions einer MD-Datenbank
adSchemaHierarchies	34	Informationen über Hierarchies einer MD-Datenbank
adSchemaLevels	35	Informationen über Levels einer MD-Datenbank
adSchemaMeasures	36	Informationen über Measures einer MD-Datenbank
adSchemaProperties	37	Informationen über Properties einer MD-Datenbank
adSchemaMembers	38	Informationen über Members einer MD-Datenbank
adSchemaTrustees	39	Informationen über Vertrauensstellungen
FieldStatusEnum		
adFieldOK	0	Erfolgreich gelöscht oder hinzugefügt
adFieldCantConvertValue	2	Feld konnte Wert nicht konvertieren.

Konstante	Wert	Beschreibung
adFieldIsNull	3	Feld ist NULL.
adFieldTruncated	4	Feld wurde abgeschnitten.
adFieldSignMismatch	5	Vorzeichenfehler, einem vorzeichenlosen Feld wurde ein Wert mit Vorzeichen zugewiesen.
adFieldDataOverflow	6	Überlauf, Daten sind zu groß für das Feld.
adFieldCantCreate	7	Feld konnte nicht erzeugt werden, weil der Provider die Größe nicht mehr unterstützt.
adFieldUnavailable	8	Provider konnte den Wert nicht finden.
adFieldPermission-Denied	9	Unzureichende Zugriffrechte.
adFieldIntegrity-Violation	10	Feld konnte nicht aktualisiert werden, weil es ein berechnetes Feld ist.
adFieldSchema-Violation	11	Wegen einer Verletzung des Zugriffsschemas konnte das Feld nicht aktualisiert werden.
adFieldBadStatus	12	Der Statuswert von der Datenbank konnte vom Provider nicht erkannt werden.
adFieldDefault	13	Der Standardwert des Felds wurde verwendet.
adFieldIgnore	15	Es wurde kein Wert gesetzt, das Feld ist unverändert.
adFieldDoesNotExist	16	Das Feld existiert nicht.
adFieldInvalidURL	17	Die URL war ungültig oder enthielt ungültige Zeichen.
adFieldResource-Lokked	18	Die Ressource ist von einem anderen Prozess belegt.
adFieldResource-Exists	19	Die Ressource existiert bereits.
adFieldCannot-Complete	20	Die Aktion konnte nicht vollständig ausgeführt werden.
adFieldVolumeNot-Found	21	Das Laufwerk, auf das die URL der Ressource zeigt, wurde nicht gefunden.
adFieldOutOfSpace	22	Das Laufwerk hat keinen Speicherplatz mehr.

Konstante	Wert	Beschreibung
adFieldCannotDeleteSource	23	Die Ressource konnte nicht gelöscht werden.
adFieldReadOnly	24	Das Feld kann nur gelesen werden.
adFieldResourceOutOfScope	25	Das Feld ist außerhalb des aktuellen Sichtbereiches.
adFieldAlreadyExists	26	Das Feld existiert bereits.
adFieldPendingInsert	&H1 0000 = 65 535	Das Feld wurde in die Kollektion eingefügt, aber der Provider hat `Update` noch nicht ausgeführt.
adFieldPendingDelete	&H2 0000 = 131 072	Das Feld wurde aus der Kollektion gelöscht, aber der Provider hat `Update` noch nicht ausgeführt.
adFieldPendingChange	&H4 0000 = 262 144	Das Feld wurde geändert, aber der Provider hat `Update` noch nicht ausgeführt.
adFieldPendingUnknown	&H8 0000 = 524 288	Es wurde eine Aktion ausgeführt und Update noch nicht beendet, die Aktion aber ist unbekannt.
adFieldPendingUnknownDelete	&H10 0000 = 1 048 576	Der Provider kann den Status der Operation nicht feststellen; das Feld wird aus der Kollektion gelöscht.
SeekEnum		
adSeekFirstEQ	&H1	Suche den ersten passenden Schlüssel
adSeekLastEQ	&H2	Suche den letzten passenden Schlüssel
adSeekAfterEQ	&H4	Suche den Schlüssel nach dem ersten passenden
adSeekAfter	&H8	Suche den Schlüssel nach dem ersten
adSeekBeforeEQ	&H10	Suche den Schlüssel vor dem ersten passenden
adSeekBefore	&H20	Suche den Schlüssel vor dem ersten
MoveRecordOptionsEnum		
adMoveUnspecified	-1	Keine Angabe
adMoveOverWrite	1	Überschreibe das Ziel, wenn es existiert
adMoveDontUpdateLinks	2	Keine Aktualisierung von Hyperlinks

Konstante	Wert	Beschreibung
adMoveAllowEmulation	4	Wenn das Verschieben der Ressource misslingt, versuche eine Kombination aus Upload-, Download-, Lösch- und Schreiboperationen.
CopyRecordOptionsEnum		
adCopyUnspecified	-1	Keine Angabe
adCopyOverWrite	1	Existierende Dateien oder Verzeichnisse werden überschrieben
adCopyAllowEmulation	4	Wenn das Verschieben der Ressource misslingt, versuche eine Kombination aus Upload-, Download-, Lösch- und Schreiboperationen
adCopyNonRecursive	2	Kopiere das aktuelle Verzeichnis, aber nicht das Unterverzeichnis
StreamTypeEnum		
adTypeBinary	1	Stream enthält binäre Daten
adTypeText	2	Stream enthält ASCII-Daten
LineSeparatorEnum		
adLF	10	Zeilentrenner LineFeed (chr(10))
adCR	13	Zeilentrenner Carriage Return (chr(13))
adCRLF	-1	Zeilentrenner LF und CR (chr(10)+chr(13))
StreamOpenOptionsEnum		
adOpenStream-Unspecified	-1	Keine Angabe
adOpenStreamAsync	1	Öffnet den Stream asynchron.
adOpenStreamFrom-Record	4	Öffnet den Stream aus einem Record-Objekt.
StreamWriteEnum		
adWriteChar	0	Schreibt in den Stream.
adWriteLine	1	Schreibt in den Stream und hängt ein LF an.
SaveOptionsEnum		
adSaveCreateNotExist	1	Erzeugt eine neue Datei, wenn sie noch nicht existiert.
adSaveCreateOver-Write	2	Überschreibt die Datei, wenn sie existiert.

Konstante	Wert	Beschreibung
FieldEnum		
adDefaultStream	-1	Gibt den Standard-Stream des Record-Objekts aus.
adRecordURL	-2	Gibt die absolute URL des Record-Objekts aus.
StreamReadEnum		
adReadAll	-1	Liest alle Bytes aus dem Stream von der aktuellen Position an.
adReadLine	-2	Liest die nächste Zeile vom Stream. Das Zeilenende wird mit Hilfe der Eigenschaft LineSeperator erkannt.
RecordTypeEnum		
adSimpleRecord	0	Das Record-Objekt ist eine Kollektion
adCollectionRecord	1	Das Record-Objekt ist eine Datei
adStructDoc	2	Das Record-Objekt ist ein strukturiertes Dokument

A.5 Fehlercodes in ADO

A.5.1 ADO-Fehlercodes

Fehlercode	Fehlertext
-2147483647	Not implemented
-2147483646	Ran out of memory
-2147483645	One or more arguments are invalid
-2147483644	No such interface supported
-2147483643	Invalid pointer
-2147483642	Invalid handle
-2147483641	Operation aborted
-2147483640	Unspecified error
-2147483639	General access denied error
-2147483638	The data necessary to complete this operation is not yet available.
-2147467263	Not implemented
-2147467262	No such interface supported
-2147467261	Invalid pointer

Fehlercode	Fehlertext
-2147467260	Operation aborted
-2147467259	Unspecified error
-2147467258	Thread local storage failure
-2147467257	Get shared memory allocator failure
-2147467256	Get memory allocator failure
-2147467255	Unable to initialize class cache
-2147467254	Unable to initialize RPC services
-2147467253	Cannot set thread local storage channel control
-2147467252	Could not allocate thread local storage channel control
-2147467251	The user supplied memory allocator is unacceptable
-2147467250	The OLE service mutex already exists
-2147467249	The OLE service file mapping already exists
-2147467248	Unable to map view of file for OLE service
-2147467247	Failure attempting to launch OLE service
-2147467246	There was an attempt to call CoInitialize a second time while single threaded
-2147467245	A Remote activation was necessary but was not allowed
-2147467244	A Remote activation was necessary but the server name provided was invalid
-2147467243	The class is configured to run as a security id different from the caller
-2147467242	Use of Ole-services requiring DDE windows isdisabled
-2147467241	A RunAs specification must be <domain name>\<user name> or simply <user name>
-2147467240	The server process could not be started. The pathname may be incorrect.
-2147467239	The server process could not be started as the configured identity. The pathname may be incorrect or unavailable.
-2147467238	The server process could not be started because the configured identity is incorrect. Check the username and password.
-2147467237	The client is not allowed to launch thisserver.
-2147467236	The service providing this server could not be started.
-2147467235	This computer was unable to communicate withthe computer providing the server.
-2147467234	The server did not respond after being launched.
-2147467233	The registration information for this serveris inconsistent or incomplete.

Fehlercode	Fehlertext
-2147467232	The registration information for this interface is inconsistent or incomplete.
-2147467231	The operation attempted is not supported.
-2147418113	Catastrophic failure
-2147024891	General access denied error
-2147024890	Invalid handle
-2147024882	Ran out of memory
-2147024809	One or more arguments are invalid
-2147217920	Invalid accessor
-2147217919	Creating another row would have exceeded the total number of active rows supported by the rowset
-2147217918	Unable to write with a read-only accessor
-2147217917	Given values violate the database schema
-2147217916	Invalid row handle
-2147217915	An object was open
-2147217914	Invalid chapter
-2147217913	A literal value in the command could not be converted to the correct type due to a reason other than data overflow
-2147217912	Invalid binding info
-2147217911	Permission denied
-2147217910	Specified column does not contain bookmarks or chapters
-2147217909	Some cost limits were rejected
-2147217908	No command has been set for the command object
-2147217907	Unable to find a query plan within the given cost limit
-2147217906	Invalid bookmark
-2147217905	Invalid lock mode
-2147217904	No value given for one or more required parameters
-2147217903	Invalid column ID
-2147217902	Invalid ratio
-2147217901	Invalid value
-2147217900	The command contained one or more errors
-2147217899	The executing command cannot be canceled
-2147217898	The provider does not support the specified dialect
-2147217897	A data source with the specified name already exists
-2147217896	The rowset was built over a live data feed and cannot be restarted

Fehlercode	Fehlertext
-2147217895	No key matching the described characteristics could be found within the current range
-2147217894	Ownership of this tree has been given to theprovider
-2147217893	The provider is unable to determine identityfor newly inserted rows
-2147217892	No nonzero weights specified for any goals supported, so goal was rejected; current goal was not changed
-2147217891	Requested conversion is not supported
-2147217890	lRowsOffset would position you past either end of the rowset, regardless of the cRows value specified; cRowsObtained is 0
-2147217889	Information was requested for a query, and the query was not set
-2147217888	Provider called a method from IRowsetNotify in the consumer and NT
-2147217887	Errors occurred
-2147217886	A non-NULL controlling IUnknown was specified and the object being created does not support aggregation
-2147217885	A given HROW referred to a hard- or soft-deleted row
-2147217884	The rowset does not support fetching backwards
-2147217883	All HROWs must be released before new ones can be obtained
-2147217882	One of the specified storage flags was not supported
-2147217880	The specified status flag was neither DBCOLUMNSTATUS_OK nor DBCOLUMNSTATUS_ISNULL
-2147217879	The rowset cannot scroll backwards
-2147217878	Invalid region handle
-2147217877	The specified set of rows was not contiguous to or overlapping the rows in the specified watch region
-2147217876	A transition from ALL* to MOVE* or EXTEND* was specified
-2147217875	The specified region is not a proper subregion of the region identified by the given watch region handle
-2147217874	The provider does not support multi-statement commands
-2147217873	A specified value violated the integrity constraints for a column or table
-2147217872	The given type name was unrecognized
-2147217871	Execution aborted because a resource limit has been reached; no results have been returned

Fehlercode	Fehlertext
-2147217870	Cannot clone a command object whose command tree contains a rowset or rowsets
-2147217869	Cannot represent the current tree as text
-2147217868	The specified index already exists
-2147217867	The specified index does not exist
-2147217866	The specified index was in use
-2147217865	The specified table does not exist
-2147217864	The rowset was using optimistic concurrency and the value of a column has been changed since it was last read
-2147217863	Errors were detected during the copy
-2147217862	A specified precision was invalid
-2147217861	A specified scale was invalid
-2147217860	Invalid table ID
-2147217859	A specified type was invalid
-2147217858	A column ID was occurred more than once in the specification
-2147217857	The specified table already exists
-2147217856	The specified table was in use
-2147217855	The specified locale ID was not supported
-2147217854	The specified record number is invalid
-2147217853	Although the bookmark was validly formed, no row could be found to match it
-2147217852	The value of a property was invalid
-2147217851	The rowset was not chaptered
-2147217850	Invalid accessor
-2147217849	Invalid storage flags
-2147217848	By-ref accessors are not supported by this provider
-2147217847	Null accessors are not supported by thisprovider
-2147217846	The command was not prepared
-2147217845	The specified accessor was not a parameter accessor
-2147217844	The given accessor was write-only
-2147217843	Authentication failed
-2147217842	The change was canceled during notification; no columns are changed
-2147217841	The rowset was single-chaptered and the chapter was not released
-2147217840	Invalid source handle

Fehlercode	Fehlertext
-2147217839	The provider cannot derive parameter info and SetParameterInfo has not been called
-2147217838	The data source object is already initialized
-2147217837	The provider does not support this method
-2147217836	The number of rows with pending changes has exceeded the set limit
-2147217835	The specified column did not exist
-2147217834	There are pending changes on a row with a reference count of zero
-2147217833	A literal value in the command overflowed the range of the type of the associated column
-2147217832	The supplied HRESULT was invalid
-2147217831	The supplied LookupID was invalid
-2147217830	The supplied DynamicErrorID was invalid
-2147217829	Unable to get visible data for a newly-inserted row that has not yet been updated
-2147217828	Invalid conversion flag
-2147217827	The given parameter name was unrecognized
-2147217826	Multiple storage objects can not be opensimultaneously
ASP Fehlcodes	Fehlertext
265920	Fetching requested number of rows would have exceeded total number of active rows supported by the rowset
265921	One or more column types are incompatible; conversion errors will occur during copying
265922	Parameter type information has been overridden by caller
265923	Skipped bookmark for deleted or non-member row
265924	Errors found in validating tree
265925	There are no more rowsets
265926	Reached start or end of rowset or chapter
265927	The provider re-executed the command
265928	Variable data buffer full
265929	There are no more results
265930	Server cannot release or downgrade a lock until the end of the transaction
265931	Specified weight was not supported or exceeded the supported limit and was set to 0 or the supported limit
265933	Input dialect was ignored and text was returned in different dialect

Fehlercode	Fehlertext
265934	Consumer is uninterested in receiving further notification calls for this phase
265935	Consumer is uninterested in receiving further notification calls for this reason
265937	In order to reposition to the start of the rowset, the provider had to reexecute the query; either the order of the columns changed or columns were added to or removed from the rowset
265938	The method had some errors; errors have been returned in the error array
265939	Invalid row handle
265940	A given HROW referred to a hard-deleted row
265941	The provider was unable to keep track of allthe changes; the client must refetch the data associated with the watch region using another method
265942	Execution stopped because a resource limit has been reached; results obtained so far have been returned but execution cannot be resumed
265944	A lock was upgraded from the value specified
265945	One or more properties were changed as allowed by provider
265946	Errors occurred
265947	A specified parameter was invalid
265948	Updating this row caused more than one row to be updated in the data source

A.5.2 SQL-Fehlercodes

Fehlercode	Fehlertext des SQL Servers
01000	General warning
01001	Cursor operation conflict, during following operation: ExecDirect
01002	Disconnect error, during following operation: Disconnect
01003	NULL value eliminated in set function, during following operation: ExecDirect
01004	String data, right-truncated, during following operation: BrowseConnect
01006	Privilege not revoked, during following operation: ExecDirect
01007	Privilege not granted, during following operation: ExecDirect
01S00	Invalid connection string attribute, during following operation: BrowseConnect

Fehlercode	Fehlertext des SQL Servers
01S01	Error in row, during following operation: BulkOperations
01S02	Option value changed, during following operation: BrowseConnect
01S06	Attempt to fetch before the result set returned the first rowset
01S07	Fractional truncation, during following operation: BulkOperations
01S08	Error saving File DSN, during following operation: DriverConnect
01S09	Invalid keyword, during following operation: DriverConnect
07001	Wrong number of parameters, during following operation: ExecDirect
07002	COUNT field incorrect, during following operation: ExecDirect
07005	Prepared statement not a cursor-specification, during following operation: ColAttribute
07006	Restricted data type attribute violation, during following operation: BindCol
07009	Invalid descriptor index, during following operation: BindCol
07S01	Invalid use of default parameter, during following operation: ExecDirect
08001	Client unable to establish connection, during following operation: BrowseConnect
08002	Connection name in use, during following operation: BrowseConnect
08003	Connection does not exist, during following operation: AllocHandle
08004	Server rejected the connection, during following operation: BrowseConnect
08007	Connection failure during transaction, during following operation: EndTran
08S01	Communication link failure, during following operation: BrowseConnect
21S01	Insert value list does not match column list, during following operation: ExecDirect
21S02	Degree of derived table does not match column list, during following operation: BulkOperations
22001	String data, right-truncated, during following operation: BulkOperations
22002	Indicator variable required but not supplied, during following operation: ExecDirect
22003	Numeric value out of range, during following operation: BulkOperations
22007	Invalid datetime format, during following operation: BulkOperations

Fehlercode	Fehlertext des SQL Servers
22008	Datetime field overflow, during following operation: BulkOperations
22012	Division by zero, during following operation: ExecDirect
22015	Interval field overflow, during following operation: BulkOperations
22018	Invalid character value for cast specification, during following operation: BulkOperations
22019	Invalid escape character, during following operation: ExecDirect
22025	Invalid escape sequence, during following operation: ExecDirect
22026	String data, length mismatch, during following operation: ParamData
23000	Integrity constraint violation, during following operation: BulkOperations
24000	Invalid cursor state, during following operation: BulkOperations
25000	Invalid transaction state, during following operation: Disconnect
25S01	Transaction state, during following operation: EndTran
25S02	Transaction is still active, during following operation: EndTran
25S03	Transaction is rolled back, during following operation: EndTran
28000	Invalid authorization specification, during following operation: BrowseConnect
34000	Invalid cursor name, during following operation: ExecDirect
3C000	Duplicate cursor name, during following operation: SetCursorName
3D000	Invalid catalog name, during following operation: ExecDirect
3F000	Invalid schema name, during following operation: ExecDirect
40001	Serialization failure, during following operation: BulkOperations
40002	Integrity constraint violation, during following operation: EndTran
40003	Statement completion unknown, during following operation: BulkOperations
42000	Syntax error or access violation, during following operation: BulkOperations
42S01	Base table or view already exists, during following operation: ExecDirect
42S02	Base table or view not found, during following operation: ExecDirect
42S11	Index already exists, during following operation: ExecDirect
42S12	Index not found, during following operation: ExecDirect
42S21	Column already exists, during following operation: ExecDirect
42S22	Column not found, during following operation: ExecDirect
44000	WITH CHECK OPTION violation, during following operation: BulkOperations

Fehlercode	Fehlertext des SQL Servers
HY000	General error All ODBC functions except:
HY001	Memory allocation error All ODBC functions except:
HY003	Invalid application buffer type, during following operation: BindCol
HY004	Invalid, during following operation: data type, during following operation: BindParameter
HY007	Associated statement is not prepared, during following operation: CopyDesc
HY008	Operation canceled All ODBC functions that can be processed asynchronously:
HY009	Invalid use of null pointer, during following operation: AllocHandle
HY010	Function sequence error, during following operation: AllocHandle
HY011	Attribute cannot be set now, during following operation: BulkOperations
HY012	Invalid transaction operation code, during following operation: EndTran
HY013	Memory management error All ODBC functions except:
HY014	Limit on the number of handles exceeded, during following operation: AllocHandle
HY015	No cursor name available, during following operation: GetCursorName
HY016	Cannot modify an implementation row descriptor, during following operation: CopyDesc
HY017	Invalid use of an automatically allocated descriptor handle, during following operation: FreeHandle
HY018	Server declined cancel request, during following operation: Cancel
HY019	Non-character and non-binary data sent in pieces, during following operation: PutData
HY020	Attempt to concatenate a null value, during following operation: PutData
HY021	Inconsistent descriptor information, during following operation: BindParameter
HY024	Invalid attribute value, during following operation: SetConnectAttr
HY090	Invalid string or buffer length, during following operation: BindCol
HY091	Invalid descriptor field identifier, during following operation: ColAttribute
HY092	Invalid attribute/option identifier, during following operation: AllocHandle
HY095	Function type out of range, during following operation: GetFunctions
HY096	Invalid information type, during following operation: GetInfo

Fehlercode	Fehlertext des SQL Servers
HY097	Column type out of range, during following operation: SpecialColumns
HY098	Scope type out of range, during following operation: SpecialColumns
HY099	Nullable type out of range, during following operation: SpecialColumns
HY100	Uniqueness option type out of range, during following operation: Statistics
HY101	Accuracy option type out of range, during following operation: Statistics
HY103	Invalid retrieval code, during following operation: DataSources
HY104	Invalid precision or scale value, during following operation: BindParameter
HY105	Invalid parameter type, during following operation: BindParameter
HY106	Fetch type out of range, during following operation: ExtendedFetch
HY107	Row value out of range, during following operation: ExtendedFetch
HY109	Invalid cursor position, during following operation: ExecDirect
HY110	Invalid driver completion, during following operation: DriverConnect
HY111	Invalid bookmark value, during following operation: ExtendedFetch
HYC00	Optional feature not implemented, during following operation: BindCol
HYT00	Timeout expired, during following operation: BrowseConnect
HYT01	Connection timeout expired All ODBC functions except:
IM001	Driver does not support this function All ODBC functions except:
IM002	Data source name not found and no default driver specified, during following operation: BrowseConnect
IM003	Specified driver could not be loaded, during following operation: BrowseConnect
IM004	Driver's SQLAllocHandle SQL_HANDLE_ENV failed, during following operation: BrowseConnect
IM005	Driver's SQLAllocHandle SQL_HANDLE_DBC failed, during following operation: BrowseConnect
IM006	Driver's SQLSetConnectAttr failed SQL: BrowseConnect
IM007	No data source or driver specified; dialog prohibited, during following operation: DriverConnect
IM008	Dialog failed, during following operation: DriverConnect
IM009	Unable to load translation DLL, during following operation: BrowseConnect

Fehlercode	Fehlertext des SQL Servers
IM010	Data source name too long, during following operation: BrowseConnect
IM011	Driver name too long, during following operation: BrowseConnect
IM012	DRIVER keyword syntax error, during following operation: BrowseConnect
IM013	Trace file error All ODBC functions
IM014	Invalid name of File DSN, during following operation: DriverConnect
IM015	Corrupt file data source, during following operation: DriverConnect

B Index

B.1 Erläuterungen zum Index

Der Index enthält zwei Arten von Seitenzahlen:

- Normale Seitenzahlen verweisen auf Fundstellen im Fließtext.
- **fett geschrieben** Zahlen kennzeichnen Syntaxdiagramme.

Außerdem sind Abkürzungen immer mit einem Verweis auf die ausgeschriebene Version versehen, sodass Sie manchmal nur im Index nachschlagen müssen, um sich ein unbekanntes Akronym zu erklären.

C An den Autor

Für Fragen, Anregungen, aber auch Kritik und Hinweise steht Ihnen der Autor gern zur Verfügung. Dieses Buch soll eine solide Basis für alle sein, die professionelle ASP und ADO Programmierer werden wollen und ein solides, deutschsprachiges Arbeitsmittel benötigen. Insofern sind Verbesserungsvorschläge und konstruktive Anmerkungen immer willkommen und werden in künftigen Auflagen sicher berücksichtigt.

Aktuelle Informationen finden Sie im Internet

Alle Skripte, Bugfixes und Korrekturen finden Sie im Internet unter der folgenden Adresse:

- *http://www.asp.comzept.de/dotnet*

Den Autor selbst können Sie auf seiner *Homepage* näher kennen lernen:

- *http://www.joerg.krause.net*

Informationen über professionelle Unterstützung

Hilfe finden Sie beim Autor, wenn es um eines der folgenden »Probleme« geht:

- *Entwicklung professioneller Websites jeder Größenordnung*
- *Projektmanagement und Programmierung in ASP, ASP.NET, PHP, JavaScript, HTML usw.*
- *Schulungen, Seminare, Programmierkurse, Workshops*
- *Fachliche Unterstützung für Start-Ups, Venture Capitalists und Old Economy ;-)*

Anfragen senden Sie bitte direkt per E-Mail an:

- *joerg@krause.net*

Leider schaffe ich es nicht immer, jede E-Mail sofort zu beantworten. Sie können aber sicher sein, dass jede E-Mail gelesen wird. Insofern können Sie dieses Medium jederzeit nutzen, um irgendetwas loszuwerden – was auch immer Sie gern mitteilen möchten.