Datenbanking

Mit Windows Script Host auf InterBase zugreifen

von Karsten Strobel

Auf der Suche nach einer Definition für Scripting stößt man auf den Begriff Glue Code (Programmcode), der hauptsächlich dazu dient, fertige Komponenten – wie Zement zwischen Ziegelsteinen – aneinander zu kleben. Im Gegensatz zu Zement trocknet Spriptcode aber nicht ein, sondern bleibt flexibel. Am Beispiel eines Anbaus an eine (Daten-)Bank möchte ich zeigen, dass die flexible Bauweise gegenüber der Betonarchitektur durchaus ihre Vorzüge hat

Der Umfang der Befehle, die unter Windows 2000 und XP in der immer noch sogenannten DOS-Box zur Verfügung stehen, hat sich zwar seit Vor-Windows-Zeiten hier und da weiterentwickelt, zu einer Revolution ist es aber – zumindest in den Augen eingefleischter .BAT-Programmierer – nicht gekommen. Oder doch? Schon seit 1996 bietet Microsoft mit dem inzwischen in der Version 5.6 verfügbaren Windows Script Host (WSH) eine erhebliche Erweiterung der Möglichkeiten an und das auch noch kostenlos. Auf neueren Windows-Varianten ist diese Software schon vorinstalliert.

Vielleicht wäre der Bekanntheitsgrad dieser Innovation unter Batch-Jongleuren größer, wenn man WSH tatsächlich als Erweiterung des DOS-Befehlssatzes verstehen könnte. Aber Berührungspunkte mit der Vorläufertechnik sind auf den ersten Blick kaum vorhanden, schon weil WSH sich nicht als klassischer Kommandointerpreter mit Prompt und blinkendem Cursor präsentiert. Auch der Befehlsvorrat ist (zum Glück) ein völlig anderer, denn man hat es nun mit respektablen Programmiersprachen und nicht mit einem Sammelsurium schlecht konzipierter Konsolenbefehle zu tun. WSH kann Programmcode interpretieren, der wahlweise in Visual Basic-Script oder in JavaScript verfasst werden kann und der zwar noch interpretierend verarbeitet wird, aber in Punkto Funktionsumfang und Komfort einen erheblichen Schritt in Richtung „echte" Programmiersprachen vollzieht.

Ich möchte aber mit diesem Artikel keine Einführung in WSH-Programmierung geben, sondern nur eine spezielle Anwendungsmöglichkeit vorstellen und zwar den Zugriff auf InterBase/Firebird-Datenbanken aus solchen Batch-Programmen. Die Flexibilität des WSH liegt nicht allein in dem Befehlsumfang der unterstützten Programmiersprachen, sondern vor allem in der Fähigkeit, automationsfähige COM-Objekte einzubinden und anzuwenden. Damit erlangen die an sich schlichten Scriptsprachen Zugriff auf einen riesigen Vorrat an Funktionalität. Von zwei Vertretern solcher COM-Objekte möchte ich in diesem Artikel Gebrauch machen: ADO (ActiveX Data Objects) für den Datenbankzugriff und FSO (File System Objects) für das Erstellen einfacher Textdateien. Diese Kombination verrät auch schon fast das Thema meines Anwendungsbeispiels.

Grenzgänger

In der Entwicklung von Datenbankanwendungen stellt sich immer wieder die Aufgabe, Informationen aus der jeweils verwendeten Datenbank und ein individuelles Textdateiformat zu exportieren, um sie dann mit anderen Programmen wieder einzulesen und weiterzuverarbeiten. Auch wenn diese Vorgehensweise vielleicht nicht zu den modernsten Techniken zählt, ist sie trotzdem Alltag bei der Integration in eine vorhandene EDV-Landschaft. Und nicht selten greifen Entwickler zur Lösung solcher eigentlich trivialer Probleme auf ihren gewohnte Hochsprachencompiler zurück und schreiben schnell ein Programm. Dateischnittstellen werden aber oft und gerne geändert und erweitert und außerdem handelt es sich bei solchen Schnittstellen auch fast immer um einen Grenzübergang in die Welt eines anderen Entwicklers, der vielleicht irgendwann selbst gerne das eine oder andere Detail an dem Datenformat verändern möchte. Deswegen ist eine offene Scriptinglösung eine schöne Alternative zur in-Stein-gemeißelten Exportanwendung. Der Quellcode ist offen lesbar, einigermaßen leicht verständlich und kann ohne Entwicklungsumgebung (außer vielleicht Notepad) geändert werden. So könnte die Vorschrift für einfache Dateischnittstelle aussehen, mit der Lieferpositionen aus einem Warenwirtschaftssystem an ein fremdes Buchhaltungssystem übergeben werden:

```
Datum:    jjjj-mm-tt
Identnummer: 10 Zeichen
Warenwert:  0000000.00
Kundennummer: 10 Zeichen
Kundenname:  30 Zeichen
```

Hinzu kommen einige Regeln, etwa dass jeder Datensatz eine feste Länge haben und mit den Zeilenendzeichen *<cr>* und *<lf>* abgeschlossen werden soll, dass ein Punkt als Dezimalkomma zu verwenden ist usw. Tatsächlich sind auch im Zeitalter von XML und Web Services solche banalen Dateischnittstellen noch immer die am häufigsten verwendete Art der Datenweitergabe.

Der Einfachheit halber können wir die zu exportierenden Daten aus der Demo-

datenbank *EMPLOYEE.GDB* beziehen, die Teil jeder Standardinstallation des InterBase-Servers ist. Das Verzeichnis, in dem diese Datenbankdatei zu finden ist, variiert von Version zu Version. Am einfachsten ist es, den Dateinamen unterhalb des InterBase-Hauptverzeichnisses (i.d.R. *C:\Programme\Borland\Interbase*) zu suchen. Das Datenmodell umfasst nur zehn Tabellen und ist sehr schlicht gehalten. Aber für unsere Zwecke reicht es völlig aus. Das folgende *SELECT*-Kommando liefert rund 20 Datensätze, die mit etwas gutem Willen als Testdaten für die obige Datensatzbeschreibung herhalten können:

```
select s.ship_date, s.po_number, s.total_value, s.cust_no,
                                        c.customer from sales s
inner join customer c on (c.cust_no=s.cust_no)
where s.order_status = 'shipped'
```

Wahl des Wirts

Wie kommt nun das Script an die Daten? Zunächst noch ein paar Vorbemerkungen zum Windows Script Host (WSH). Hinter dieser Bezeichnung verbergen sich genau genommen zwei Host-Anwendungen, nämlich *WScript.exe* und *CScript.exe*, die beide die Ausführung von Programmen erlauben, die in VBScript (Dateiendung *.vb*) oder in JScript (Microsofts Implementations von JavaScript, Dateiendung *.js*) geschrieben sind. *WScript* führt das Script im Rahmen einer Windowsanwendung aus, wobei alle Ausgaben als Dialogboxen dargestellt werden. CScript ist ein klassisches Konsolenprogramm. Der Leistungsumfang dieser beiden Hosts ist fast identisch, der Sprachumfang ist definitiv gleich, weil sich beide Programme der selben Script Engine bedienen. Geben Sie an der Kommandozeile einmal *cscript* ein.

```
C:> csript
Microsoft (R) Windows Script Host, Version 5.6
...
```

Bei allen neueren Windows-Varianten ist WSH bereits installiert. Die aktuelle Version ist 5.6. Wenn *cscript* bei Ihnen eine ältere Versionsnummer meldet, sollten Sie diese Komponenten updaten [1]. Die Wahl der Scriptsprache (VBScript oder JScript) ist Geschmackssache. Ich verwende lieber JScript, weil mir die C-ähnlich Syntax besser geläufig ist und ich von der Webseiten-Programmierung hier etwas mehr Erfahrung habe. Die Verwendung von JScript in Webseiten – als Programmcode, der in HTML-Seiten eingebettet und vom Browser ausgeführt wird – ist übrigens sehr populär. Hier kommt die selbe Script Engine zum Einsatz, aber der Script Host ist z.B. der Internet Explorer. Wundern Sie sich also nicht, wenn Sie an einigen Stellen der Dokumentation Beispiele finden, die mit der Programmierung von Homepages zu tun haben. Microsoft bietet ein sehr nützliches Helpfile zum Download [1] an, in dem sowohl die Sprachen als auch die speziellen WSH-Funktionen sehr gut dokumentiert sind.

Der WSH (*CScript* bzw. *WScript*) stellt den Scriptprogrammen ein Objekt namens *WScript* zur Verfügung, das den Zugang zu verschiedensten Funktionen anbietet, mit denen u.a. Administrationsaufgaben (Drucker einrichten, Netzwerkverbindungen herstellen etc.) ausgeführt werden können. Die simpelste aller Funktionen heißt *WScript.Echo()*, die einen Text an der Konsole bzw. in einer MessageBox ausgibt. Mit JScript (und auch *VBScript*) kann man selbstverständlich eigene Funktionen und sogar Objekte programmieren, jedoch gibt es keinen spezifischen Einsprungpunkt. Wenn Sie ein Scriptprogramm starten, dann wird sämtlicher Code unmittelbar ausgeführt, der nicht in Funktionen oder Objekte verpackt ist. Das klassischste aller Beispiele ist daher tatsächlich ein Einzeiler, den Sie z.B. als *test.js* speichern und mit *cscript test.js* ausführen können:

```
WScript.Echo(„Hallo Welt“);
```

Treiber heißt jetzt Dienstanbieter

Da der WSH es uns ermöglicht, automationsfähige ActiveX-Objekte einzubinden, kann man auch ohne weiteres *ADO* (ActiveX Data Objects) von JScript aus verwenden. ADO ist die anwenderfreundliche Objektarchitektur, die als Unterbau einen sogenannten OLE DB Provider verwendet, um zum Beispiel auf eine relationale Datenbank zuzugreifen. Um mit ADO-Objekten InterBase erreichen zu können, braucht man also einen passenden OLE DB Provider. ADO und ein Standardvorrat an OLE DB Providern werden mit dem von Microsoft kostenlos angebotenen Setupprogramm *MDAC_TYP.EXE* [2] installiert. Die aktuelle Version von MDAC (Microsoft Data Access Components) ist 2.7 SP1. Um festzustellen, welche Version bei Ihnen installiert ist, können Sie entweder den MDAC Component Checker [2] verwenden, oder einfach gleich das neueste *MDAC_TYP.EXE* ausführen.

Allerdings bietet Microsoft keinen OLE DB Provider für InterBase an, was auch kaum überraschen dürfte. Vor einigen Monaten habe ich vier verschiedene Provider für InterBase bzw. Firebird vorgestellt [3]. Nur einer der Kandidaten, nämlich das Produkt mit dem schlichten Namen *IB OLEDB* von Ralph Curry [4], ist als Freeware verfügbar. Zwar hatte gerade dieser Treiber bei dem letzten Vergleichstest ziemlich schlecht abgeschnitten, aber inzwischen ist die Versionsnummer von 1.1 auf 1.6 geklettert und der Entwickler (bzw. sein Produkt) hat offenbar deutliche Fortschritte in Punkto Zuverlässigkeit gemacht. Inzwischen ist sogar der Quellcode als Open Source verfügbar [5]. Erfahrungen bezüglich der Langzeitstabilität dieses Treibers liegen zwar noch nicht vor, sind im Batch-Betrieb aber auch nicht von so großer Bedeutung. Zumindest für unseren ziemlich unkritischen Anwendungsfall dürfte dieser OLE DB Provider völlig ausreichen.

Die Installation ist herzlich einfach. Die Software besteht nur aus der Datei *IbOleDb.dll*, die Sie in ein beliebiges Verzeichnis (empfohlen wird *\Programme\Gemeinsame Dateien\System\Ole DB*) herunterladen und per Kommandozeile mit

```
regsvr23 iboledb.dll
```

registrieren. Anschließend können Sie einen ganz einfachen Test machen: Legen Sie eine leere Datei mit der Endung *.UDL* an und öffnen Sie diese im Windows Explorer per Doppelklick. Dadurch wird ein Eigenschaftsdialog (Abb. 1) geöffnet, in dem Sie auf dem ersten Tabenblatt den vorher registrierten Provider *IbOleDb* auswählen, und auf dem zweiten Tabenblatt die Verbindungsdaten einstellen. Mit

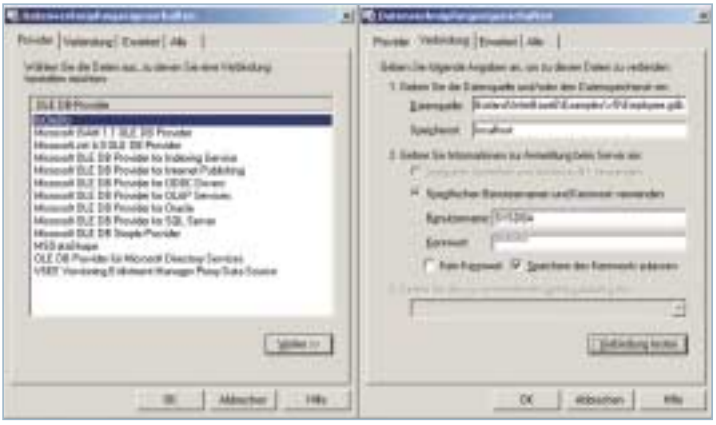

Abb. 1: Provider testen

der Schaltfläche *Verbindung testen* können Sie überprüfen, ob die Verbindung zu Stande kommt. Die *.UDL*-Datei können Sie danach wieder löschen.

Um nun das Script und den OLE DB Provider zusammenzubringen, verwenden wir ADO als Überbrückungshilfe. Aus der umfangreichen Auswahl von ADO-Objekten greifen wir uns nur zwei heraus, nämlich das *Connection* und das *RecordSet* Objekt. Wie die Namen schon andeuten, ist das erste für die Definition der Datenbankverbindung, das zweite für die Darstellung einer Datenmenge zuständig. Die Beschreibung der ADO Objekte ist Teil der Doku zum Microsoft Platform SDK, die Sie als Teil diverser Entwicklerprodukte, im MSDN-Abo oder im Internet [6] vorfinden. Mit den folgenden zwei Anweisungen erzeugen Sie mit JScript jeweils eine Instanz dieser Objekte:

```
var conn = new ActiveXObject("ADODB.Connection");
var rs = new ActiveXObject("ADODB.Recordset");
```

Um die Verbindung zur Datenbank herzustellen, rufen Sie die *Open()*-Methode des Connection Objektes auf und geben als Parameter den passenden Connection-String an:

```
conn.Open("Provider=IbOleDb;Location=localhost;Data
Source=C:\\Programme\\Borland\\InterBase6\\Exam-
ples\\v5\\Employee.gdb;User ID=SYSDBA;Password=mas-
terkey;Extended Properties='Character Set=ISO8859_1'");
```

Der Aufbau des Connection-Strings kann bei verschiedenen OLE DB Providern etwas variieren. Am besten konsultiert man die jeweilige Dokumentation, um die Details herauszufinden. In unserem Fall ist eine Besonderheit, dass der vom Datenbankclient verwendete Zeichensatz in den „Extended Properties" angegeben werden muss. Der von mir gewählte ISO8859_1 Zeichensatz wird bei InterBase für westeuropäische Sprachen am häufigsten verwendet. Beachten Sie auch, dass die Backslashes in Pfadangaben doppelt angegeben werden müssen, weil JScript sonst jeden Backslash als Einleitung eines Sonderzeichencodes interpretiert. Die Open-Methode gibt kein Ergebnis zurück, löst aber eine Exception aus, wenn die Verbindung nicht hergestellt werden kann, etwa weil eine Pfadangabe nicht stimmt. Keine Nachricht ist also eine gute Nachricht. Am Ende des Scriptprogramms wird die Verbindung mit *conn.Close()* wieder geschlossen.

ADO auf einfachste Art

Nachdem die Verbindung aufgebaut ist, können wir das *RecordSet*-Objekt verwenden, um eine Datenmenge abzurufen, was mit den folgenden Codezeilen geschieht:

```
rs.CursorLocation = 3; //adUseClient
var sql = "select s.ship_date, s.po_number, s.total_value,
s.cust_no, c.customer from sales s inner join customer c on
(c.cust_no=s.cust_no) where s.order_status = 'shipped'";

rs.Open(sql, conn);
```

Leider kann man für ADO-Aufzählungstypen nicht die symbolischen Bezeichner verwenden, sondern muss die Wertigkeit der gewünschten Konstante der ADO-Dokumentation entnehmen und einsetzen. Es empfiehlt sich sehr, den Konstantenbezeichner als Kommentar anzugeben. Die Eigenschaft CursorLocation des Recordsets muss vor dem Öffnen der Datenmenge gesetzt werden. Die Einstellung 3 (symbolisch: *adUseClient*) fordert ADO auf, die Ergebnismenge clientseitig zwischenzuspeichern. Die beiden Parameter von *rs.Open()* geben den SQL-Code und den Bezug auf die Datenbankverbindung (*conn*) an. Für die Open-Methode sind weitere optionale Parameter möglich, die wir aber nicht zwingend brauchen und die ich der Einfachheit halber weglasse. Auch sonst streife ich die Möglichkeiten von ADO hier nur ganz leicht. Wenn Sie mehr erfahren möchten, empfehle ich Ihnen die offizielle ADO-Dokumentation oder das Buch „ADO und Delphi" [7].

Um mich zu vergewissern, dass ich wirklich die erwarteten Daten erhalte, gebe ich den ersten Feldwert (*ship_date*) einfach an der Konsole aus:

```
while (!rs.EOF) {
 WScript.Echo(rs(0).Value);
 rs.MoveNext();
}
```

rs(0) liefert ein Field-Objekt, das die erste Ergebnisspalte des aktuellen Datensatzes repräsentiert. Die Value-Eigenschaft enthält den Feldwert als varianten Typ. In Listing 1 ist der bisherige Stand des Scripts abgedruckt.

Reine Formsache

Nun ist es an der Zeit, sich um die Formatierung der Ausgabe Gedanken zu machen. Wenn Sie das bis jetzt entstandene Script laufen lassen, dann sehen Sie, dass das Datum aus dem Feld *ship_date* zum Beispiel als *31.05.1993* erscheint, was – je nach Einstellung des Gebietsschema – auch variieren kann. Benötigt wird aber das Format *jjjj-mm-tt*. JScript bietet leider nur eine recht spartanische Ausstattung mit Formatierungsfunktionen an. Daher muss man solche Probleme selbst mit ein paar Zeile Code lösen. Hier bietet die prototype Eigenschaft der JScript-Objekte einen interessanten Lösungsansatz. JScript arbeitet sehr weitgehend objektorientiert. Auch JScript-Datentypen wie *String*, *Date* oder *Number* sind Objekte, die jeweils typspezifische Eigenschaften und Methoden aufweisen. Ungewöhnlich ist dabei, dass man auch für vordefinierte Objekttypen neue Methoden einführen kann, indem man eine

einfache Funktion implementiert und diese der Objekteigenschaft *prototype* unter einem frei wählbaren, neuen Methodennamen zuweist:

```
function str_ralign(size, fill_ch) {
 var s = this;
 while (s.length < size) s = fill_ch + s;
 return s;
}
String.prototype.ralign = str_ralign;

function date_myFormat() {
 var s = this.getFullYear() + "-";
 s += (this.getMonth() + 1).toString().ralign(2, "0") + "-";
 s += this.getDate().toString().ralign(2, "0");
 return s;
}
Date.prototype.myFormat = date_myFormat;
```

Hier wird für das *String*-Objekt eine *ralign*-Methode, für das *Date*-Objekt die *myFormat*-Methode „hinzuerfunden". Wie man sieht, müssen Methodenname und Funktionsname nicht unbedingt übereinstimmen. Nach der Anmeldung der neuen Methoden kann diese sofort für solche Objektinstanzen verwendet werden. Wenn Sie die Ausgabeschleife dann wie folgt abändern:

```
while (!rs.EOF) {
 var d = new Date(rs(0).Value);
 WScript.Echo(d.myFormat());
 rs.MoveNext();
}
```

wird das Datum im gewünschten Format (z.B. „1993-05-31") ausgegeben. Ebenso kann man bei der Formatierung anderer Felder verfahren. Listing 2 zeigt den bisher erreichten Stand.

Spätestens jetzt sollte ich auf die Debugging-Möglichkeiten hinweisen. Wenn Sie *cscript.exe* mit der Option *//X* starten, wird das Script in einem Debugger ausgeführt, sofern ein kompatibler Debugger installiert ist. Dies kann Visual Studio sein oder aber auch der etwas schlichtere Script Debugger, den Microsoft zum freien Download anbietet [1].

Szenenwechsel: Es ist jetzt ersichtlich, wie wir an die Daten herankommen und wie bei der Formatierung vorzugehen ist. Es fehlt nur noch ein Weg, die Daten in eine einfache Textdatei auszugeben. JScript bietet für Dateizugriffe keine eigene Funktionalität, aber mit der Script Engine wird ein ActiveX-Objekt namens *FileSystemObject* (kurz: FSO) installiert, das diese Aufgabe übernehmen kann. Analog zur Erzeugung der ADO-Objekte kann das JScript-Programm von diesem Objekt eine Instanz kreieren und anwenden, z.B.:

```
var fso = new ActiveXObject("Scripting.FileSystemObject");
var ts = fso.CreateTextFile("C:\\Temp\\Test.txt", true);
ts.WriteLine("Eine Zeile Text");
ts.Close();
```

CreateTextFile() liefert ein Objekt vom Typ *TextStream* zurück, das wiederum u.a. eine Methode *WriteLine()* anbietet, die eine Textzeile in die Datei schreibt. Damit wäre für alle Probleme ein Lösungsansatz vorhanden. Listing 3 zeigt das fertige Script, das den gewünschten Datenexport ausführt. Dokumentiert ist das FileSystemObject übrigens in der selben Help-Datei (Script56.CHM), in der JScript und WSH beschrieben sind.

In Listing 3 sind noch ein paar Neuerungen hinzugekommen: Ich habe die Formatierungsmethoden ausgebaut, um auch die Strings und den Währungswert in das geforderte Format bringen zu können. Außerdem ist der Datenbankzugriff noch in eine explizite Transaktion (*BeginTrans*, *CommitTrans*) gekapselt, begleitet von einem *try..catch*-Block, der im Falle einer Exception die Datenbankänderungen mit RollbackTrans zurücknimmt und auch die evtl. schon begonnene Exportdatei wieder löscht. Da Bewegungsdaten wie Lieferpositionen i.d.R. nicht immer wieder sondern nur einmalig exportiert werden sollen, wollte ich die Datensätze eigentlich mit einem Update-Statement als bereits exportiert kennzeichnen. Leider bietet sich in der Sales Tabelle dafür kein Feld an, deshalb habe ich mich einfach dazu entschlossen, jeden exportierten Datensatz mit einer Delete-Anweisung zu löschen. Zuverlässiger kann man ein mehrfaches Exportieren zwar kaum verhindern, aber so ganz praxisge-

Listing 1

```
 var conn = new ActiveXObject("ADODB.Connection");
var rs = new ActiveXObject("ADODB.Recordset");

conn.Open("Provider=IbOleDb;Location=localhost;
    Data Source=C:\\Programme\\Borland\\InterBase6\\
        Examples\\v5\\Employee.gdb;User ID=SYSDBA;
Password=masterkey;Extended Properties='Character Set=
                                        ISO8859_1'");

rs.CursorLocation = 3; //adUseClient
var sql = "select s.ship_date, s.po_number, s.total_value,
                                    s.cust_no, c.customer " +
    "from sales s inner join customer c on
(c.cust_no=s.cust_no) " +
    "where s.order_status = 'shipped'";
rs.Open(sql, conn);

while (!rs.EOF) {
 WScript.Echo(rs(0).Value);
 rs.MoveNext();
}
rs.Close();
conn.Close();
```

Listing 2

```
 function str_ralign(size, fill_ch) {
 var s = this;
 while (s.length < size) s = fill_ch + s;
 return s;
}
String.prototype.ralign = str_ralign;

function date_myFormat() {
 var s = this.getFullYear() + "-";
 s += (this.getMonth() + 1).toString().ralign(2, "0") + "-";
 s += this.getDate().toString().ralign(2, "0");
 return s;
}
Date.prototype.myFormat = date_myFormat;

var conn = new ActiveXObject("ADODB.Connection");
var rs = new ActiveXObject("ADODB.Recordset");

conn.Open("Provider=IbOleDb;Location=localhost;
    Data Source=C:\\Programme\\Borland\\InterBase6\\
        Examples\\v5\\Employee.gdb;User ID=SYSDBA;
Password=masterkey;Extended Properties='Character Set=
ISO8859_1'");

rs.CursorLocation = 3; //adUseClient
var sql = "select s.ship_date, s.po_number, s.total_value,
                                    s.cust_no, c.customer " +
    "from sales s inner join customer c on (c.cust_no=
                                        s.cust_no) " +
    "where s.order_status = 'shipped'";
rs.Open(sql, conn);

while (!rs.EOF) {
 var d = new Date(rs(0).Value);
 WScript.Echo(d.myFormat());
 rs.MoveNext();
}
rs.Close();
conn.Close();
```

recht ist dieses Vorgehen natürlich nicht. Mir kam es aber darauf an, ein möglichst einfaches Beispiel zu liefern, daher hoffe ich, dass Sie diesen rüden Umgang mit den Demodaten verzeihen werden. Um zu verhindern, dass die Daten tatsächlich gelöscht werden, können Sie die *CommitTrans* Anweisung einfach in RollbackTrans abändern. Zum Löschen der exportierten Datensätze reicht es in diesem Beispiel übrigens nicht aus, die *Delete*-Methode des Recordsets aufzurufen, denn die Grundlage des Recordsets bildet ein *Select*, der sich per JOIN-Syntax auf mehrere Tabellen bezieht. Daher ist für ADO nicht feststellbar, welche Datensätze aus welchen Tabellen zu löschen sind, und deswegen muss das Löschen in diesem Fall „von Hand" vorgenommen werden.

Ein Makel an dieser Lösung bleibt, dass das Datenbankpasswort im *ConnectString* als Klartext sichtbar ist. In dem vorliegenden Beispiel habe ich sogar gegen zwei goldene Regeln verstoßen, nämlich das *SYSDBA*-Kennwort auf der Voreinstellung „masterkey" zu belassen, und diesen Super-Benutzer überhaupt für Anwendungszwecke zu verwenden. Sinnvoll wäre es natürlich, für den Datenexport einen anderen Datenbankbenutzer einzurichten, und diesem nur die erforderlichen Zugriffsrechte zu geben. Darüber hinaus sollte der Script-Quellcode nur für autorisierte Anwender einsehbar sein.

Fazit

Die vorgestellte Technik für den Zugriff auf Datenbanken kann weder in Punkto Komfort noch Performance mit den Möglichkeiten konkurrieren, die man mit Produkten wie Delphi, CBuilder o.ä. zur Verfügung hat. Der Charme dieses Ansatzes liegt in seiner Schlichtheit und in der Tatsache, dass ein Scriptprogramm auf jedem Anwendungsrechner modifizieren kann, auch wenn keine Entwicklungsumgebung installiert ist.

Links & Literatur

[1] www.microsoft.com/germany/scripting
[2] www.microsoft.com/data
[3] *Der Entwickler*: OLE DB-Provider im Überblick, 6.2002
[4] www.oledb.net
[5] sourceforge.net/project/showfiles.php?group_id=72859
[6] msdn.microsoft.com
[7] Andreas Kosch: ADO und Delphi, Software und Support Verlag

Listing 3

```
function str_lalign(size, fill_ch) {
 var s = this;
 while (s.length < size) s += fill_ch;
 return s;
}
String.prototype.lalign = str_lalign;

function str_ralign(size, fill_ch) {
 var s = this;
 while (s.length < size) s = fill_ch + s;
 return s;
}
String.prototype.ralign = str_ralign;

function str_quoted( ) {
 var s = "";
 for (i=0; i<this.length; i++) {
  s += this.charAt(i);
  if (this.charAt(i) == "'") s += "'";
 }
 return "'" + s + "'";
}
String.prototype.quoted = str_quoted;

function num_myFormat(size) {
 var s;
 if (this >= 0) s = this.toString();
 else {
  size—;
  s = (-this).toString();
 }
 var i = s.indexOf(".");
 if (i < 0) { s = s + „.00" } else while (s.length - i < 3) s = s +
"0";
 s = s.ralign(size, "0");
 if (this < 0) s = "-" + s;
 return s;
}
Number.prototype.myFormat = num_myFormat;

function date_myFormat() {
 var s = this.getFullYear() + "-";
 s += (this.getMonth() + 1).toString().ralign(2, "0") + "-";
 s += this.getDate().toString().ralign(2, "0");
 return s;
}
Date.prototype.myFormat = date_myFormat;

//-------------------- Hauptprogramm----------------------

var conn = new ActiveXObject("ADODB.Connection");
var rs = new ActiveXObject("ADODB.Recordset");

conn.Open("Provider=IbOleDb;Location=localhost;Data
Source=C:\\Programme\\Borland\\InterBase6\\Examples\\v5\\Employee.gdb;User ID=SYSDBA;Password=masterkey;Extended Properties='Character Set=ISO8859_1'");

var ExportFileName = "c:\\temp\\export.dat";
var fso = new ActiveXObject("Scripting.FileSystemObject");
var ts = fso.CreateTextFile(ExportFileName, true);

conn.BeginTrans();
try {
 rs.CursorLocation = 3; //adUseClient
 var sql = "select s.po_number, s.ship_date, s.total_value,
s.cust_no, c.customer" +
      "from sales s inner join customer c on
(c.cust_no=s.cust_no) „ +
      "where s.order_status = 'shipped'";
 rs.Open(sql, conn);

 while (!rs.EOF) {
  s = rs(0).Value.toString().lalign(10, " ");
  ts.Write(s);

  d = new Date(rs(1).Value);
  ts.Write(d.myFormat());

  n = new Number(rs(2).Value);
  ts.Write(n.myFormat(10));

  s = rs(3).Value.toString().lalign(10, " ");
  ts.Write(s);

  s = rs(4).Value.toString().lalign(30, " ");
  ts.WriteLine(s);

  s = rs(0).Value.toString();
  conn.Execute("delete from sales where po_number=" +
s.quoted());

  rs.MoveNext;
 }
 WScript.Echo(rs.RecordCount.toString() + " Datensätze
wurden exportiert");
 conn.CommitTrans();
 rs.Close();
}
catch(e) {
 conn.RollbackTrans();
 WScript.Echo("Rollback wegen Exception!");
 ts.Close();
 fso.DeleteFile(ExportFileName);
 throw e;
}

conn.Close();
ts.Close();
```