Verwaltet verwalten

Sammeln von Testdaten leicht gemacht, Teil 3

von Urs Gehrig

Mit SQL-DMO das generierte Skript einer Tabelle aus einem SQL Server auszulesen, ist eine Leichtigkeit. Das Bestimmen des Datentypes einer Tabellenspalte geht auch noch so – auch wenn die Typenbeschreibung nicht zu den ADO.NET-Typen passt. Mit einem *SELECT*-Statement dagegen Daten aus einer Tabelle abzufragen und diese anschließend in einem *DataGrid* darzustellen, ist dann endgültig nicht mehr lustig. Das *QueryResult* von SQL-DMO hat mit einem *DataSet* überhaupt nichts zu tun und an ein *Data Binding* ist nicht zu denken. Gut Ding will eben Weile haben: Zusammen mit SQL Server 2005 stellt Microsoft mit den *SQL Management Objects* (SMO) ein neues, 100 Prozent pures .NET-API für den SQL Server zur Verfügung.

Wenn Sie Teil 2 dieser Artikelserie [1] gelesen haben, wissen Sie schon, dass sich mit wenig Aufwand und einem gezielten Einsatz von SQL-DMO intelligente Management-Tools für den SQL Server schreiben lassen – so z. B. *CloneDB*. Das Zusammenspiel von SQL-DMO und .NET lässt aber schwer zu wünschen übrig: Inkompatible Datentypen, ungewohntes Iterieren durch Enumerationen und ein unübersichtlicher Namespace sind nur ein paar Beispiele. Mit den SMO bringt Microsoft einen Nachfolger von SQL-DMO ins Spiel, der nicht nur perfekt in die .NET-Welt passt, sondern zusätzlich noch neue nützliche Features enthält. Dieser Artikel zeigt, wie Sie *CloneDB* auf SMO migrieren, und stellt ein paar Tricks vor, mit denen Sie hinter die Kulissen der SMO schauen können. Dieser Beitrag basiert noch auf einer Vorabversion (Beta 2 CTP June) des SQL Server 2005.

Das kleine Einmaleins der SMO

SQL Management Objects (SMO) ist das Managementobjektmodell für den SQL Server 2005 und Microsofts erste Adresse für die Entwicklung von Datenbankmanagementapplikationen. *SQL Server Management Studio* selbst basiert auf SMO – jede administrative Aufgabe, die sich damit durchführen lässt, kann mit SMO auch in eine eigene Applikation verpackt werden. Im Zusammenspiel mit WMI ersetzt SMO die schon etwas in die Jahre gekommene SQL-DMO. SQL-DMO lässt sich zwar auch mit dem SQL Server 2005 nutzen, dessen spezifische Features werden aber nicht unterstützt. SMO selbst läuft ab SQL Server 7.0. Das Objektmodell der SMO ist eine logische Fortführung der Entwicklung von SQL-DMO und funktionskompatibel mit diesem. Um ein Maximum der *Data-Definition-Language*- (DDL) und administrativen Erweiterungen des SQL Server 2005 abzudecken, sind in den SMO mehr als 150 neue Klassen hinzugekommen. Die wichtigsten Vorteile der SMO gegenüber SQL-DMO sind wohl deren Realisierung in 100 Prozent purem .NET sowie ihre Performance und Skalierbarkeit. SMO hat ein Objekt-Cache-Modell, welches erlaubt, mehrere Attribute eines Objektes zu ändern, bevor diese auf dem Server abgebildet werden. Als Folge daraus unternimmt SMO weniger *Round Trips* zum Server.

kurz & bündig

Inhalt

Mit SQL-DMO war es bereits ein leichtes, Schemas und Daten aus einer Datenbank zu skripten – mit dem Nachfolger SMO geht dies noch viel eleganter

Zusammenfassung

SMO als 100% pures .NET-API für den SQL Server integriert bestens in Ihre .NET-Applikation

Quellcode auf CD

Erste Schritte mit den SMO

Wer *CloneDB* auf seinem SQL Server 2005 ausprobiert, wird schon rasch seine erste Überraschung erleben – kaum gestartet erscheint die erste Fehlermeldung:

```
[Microsoft][ODBC SQL Server Driver][SQL Server]To connect to this server you must use SQL Server Management Studio or SQL Server Management Objects (SMO).
```

Wenngleich auch Microsoft die SQL-DMO funktional nicht erweitert hat, mussten die Entwickler dennoch ein kleines SQL-DMO-Update für den SQL Server 2005 schreiben [5]. Nach der Installation des Updates läuft *CloneDB* aber ohne Beanstandung und das erste Versprechen von Microsoft – SQL Server 2005 lässt sich auch mit SQL-DMO ansprechen – ist bereits eingelöst. Los geht der kurze Überblick über das Migrieren von *CloneDB*. Die wichtigsten Klassen der SMO befinden sich in den drei Namespaces:

```
using Microsoft.SqlServer.Management.Smo;
using Microsoft.SqlServer.Management.Common;
using Microsoft.SqlServer.MessageBox;
```

Das Durchsuchen des Netzwerkes nach vorhanden SQL Servern und das Verbinden zu einem spezifischen Server (Listing 1) gestaltet sich mit den SMO nicht viel anders als bei SQL-DMO. Einzig das Verbinden zum Server ist etwas eleganter, bietet das *ConnectionContext*-Objekt jetzt doch ein *IsOpen*-Property. Schon richtig cool wird es aber beim *Data Binding* eines SMO *QueryResult* (Listing 2). Das ist wirklich einfach und jeder, der ADO .NET kennt, kann dies auch mit den SMO machen. Den Vergleich zu SQL-DMO hier abzubilden, möchte der Autor dem Leser ersparen, wen es interessiert, der

kann die Lösung gerne im beigefügten Quellcode zum zweiten Teil dieser Serie nachlesen [1]. Nur so viel: Die Lösung mit SQL-DMO benötigt für dieselbe Aufgabe ca. 100 Zeilen Code, weil das *QueryResult* von SQL-DMO erst in ein ADO.NET-kompatibles *DataTable*-Objekt konvertiert werden muss. Auch das Generieren eines T-SQL-Skripts für ein beliebiges Datenbankobjekt ist kinderleicht – jedes Objekt besitzt hierfür eine *Script*-Methode. Über die spezielle *Scripter*-Klasse lässt sich gar das Skripten ganzer Collections von Datenbankobjekten gezielt steuern (Listing 3).

Listing 1

C#-Codesnippet zum Auflisten aller verfügbaren SQL Server

```
DataTable myServers = SmoApplication.
                        EnumAvailableSqlServers(false);

foreach(DataRow nextServer in myServers.Rows)
{
  // SMO Server Objekt erstellen
  ServerConnection myServerConnection =
                                  new ServerConnection();
  myServerConnection.LoginSecure = true;
  myServerConnection.ServerInstance = nextServer
                                  ["Name"].ToString();
  myServerConnection.ApplicationName = "CloneDB";
  Server myServer = new Server(myServerConnection);

  // Verbindung zum Server aufbauen
  if (!myServer.ConnectionContext.IsOpen)
  {
    myServer.ConnectionContext.Connect();
  }

  // SMO Server Objekt irgendwo beiseite stellen

}
```

Listing 2

C#-Codesnippet zum Data Binding eines SMO-Query-Ergebnisses

```
Database myDatabase;
DataGrid myGrid;

// SMO Database Objekt initialisieren
...

// Query an Database absetzen
myGrid.DataSource = myDatabase.ExecuteWithResults
                        ("SELECT * FROM Address");
myGrid.DataMember = "Table";
```

Von den SMO lernen

Wo lässt sich am Meisten über die Innereien des SQL Server lernen? Richtig, bei den Profis des *Microsoft SQL Server Teams*. Lernen Sie von Ihnen, wie man komplexe Aufgaben in T-SQL lösen kann. Damit Sie den weiten Weg nach Redmond aber nur virtuell antreten müssen, können Sie sich einem neuen Feature der SMO bedienen – dem *Capture Mode* (Listing 4). Der Capture Mode erlaubt es, alle T-SQL Befehle, welche über die *SMO Connection* zum Server geschickt werden, aufzuzeichnen. Alle T-SQL-Befehle heißt auch wirklich alle – jeder SMO-Befehl endet letztlich in einem oder mehreren T-SQL-Befehlen und all diese werden aufgezeichnet.

Von den SMO können Sie aber noch mehr lernen – registrieren Sie einige SMO Event Handler (Listing 5 auf der Heft-CD) und beobachten Sie, wann welche Events ausgelöst werden und welche Informationen sich in deren Event-Argumenten verstecken. Die *ExceptionMessageBox* der SMO hilft Ihnen dabei, diese Informationen augenfällig zu visualisieren.

Zusammengefasst

Wer bereits mit SQL-DMO vertraut ist, wird sich über das einfachere und leistungsfähigere Objektmodell der SMO freuen. Neulinge werden von den SMO ermuntert, ihre Applikationen um SQL-Server-Managementaufgaben zu erweitern. Und wenn Sie sich doch im Dschungel der SMO und des SQL Server verlaufen, denken Sie an den *Capture Mode* und die diversen nützlichen Event Handler – sie werden Ihnen helfen, Licht ins Dunkel zu bringen. ●

Urs Gehrig ist Head of System Services der bbv Software Services AG in der Schweiz. Er selbst ist seit 1989 als Entwickler von Windows-Applikationen und EDV-Berater tätig. Seine Steckenpferde sind Datenbanken und Security. Sie erreichen Ihn unter urs.gehrig@bbv.ch.

Listing 3

C#-Codesnippet zum Skripten einer Tabelle

```
Server      myServer;
Table       myTable;
Scripter    myScrptr;
SqlSmoObject[]  smoObjects;
StringCollection strColl;

// SMO Server & Table Objekt initialisieren
...

// SMO Scripter Objekt erstellen und initialisieren
myScrptr = new Scripter(myServer);
myScrptr.Options.ScriptDrops = true;
myScrptr.Options.WithDependencies = true;

// myTable scripten (inkl. allen Abhängigkeiten und
deren Drop Befehlen)
smoObjects = new SqlSmoObject[1];
smoObjects[0] = myTable;
strColl = scrptr.Script(smoObjects);

// T-SQL Script ausgeben
foreach (string s in strColl)
{
        Console.WriteLine(s);
}
```

Listing 4

C#-Codesnippet zum Aufzeichnen aller T-SQL-Befehle

```
Server myServer;

// SMO Server Objekt initialisieren
...

// SMO in Capture Mode versetzen
myServer.ConnectionContext.SqlExecutionModes =
        SqlExecutionModes.ExecuteAndCaptureSql;

// die gewünschte Arbeit vollbringen
...

// SMO wieder in Normal Mode versetzen
myServer.ConnectionContext.SqlExecutionModes =
        SqlExecutionModes.ExecuteSql;

// T-SQL Script ausgeben
foreach (string s in myServer.ConnectionContext.
                                CapturedSql.Text)
{
        Console.WriteLine(s);
}
```

● Links & Literatur

[1] Urs Gehrig: Aller Anfang ist schwer; Sammeln von Testdaten leicht gemacht, Teil 1 & 2, in: *dot.net magazin* 9/10.2005

[2] Getting Started with SMO in SQL 2005: www.sqldbatips.com/showarticle.asp?ID=34.

[3] blogs.msdn.com/mwories/archive/2005/05/07/basic_scripting.aspx

[4] Urs Gehrig: Optimale Performance mit SQL Server 2000, in: *dot.net magazin* 3/4.2004

[5] www.microsoft.com/downloads/ und Suche nach „SQL Server"