Aller Anfang ist schwer

Sammeln von Testdaten leicht gemacht (Teil 1)

von Urs Gehrig

Was hat eine Testdatenbank mit einem Humpen Bier gemeinsam? Richtig – beide sind immer leer. Und wo liegt der große Unterschied zwischen diesen beiden? Wieder richtig – der Humpen füllt sich leicht, die Testdatenbank bleibt leer. Dieser Zweiteiler zeigt Ihnen einen Weg, wie Sie zu relevanten Testdaten kommen. Damit dies nicht in Schwerstarbeit ausartet und Ihnen genügend Zeit für ein kühles Bier bleibt, bekommen Sie gleich noch ein Scripting-Tool zum Weiterentwickeln mit auf den Weg.

Es ist schon fast ein Naturgesetz – Testdatenbanken sind immer leer. Im besten Fall sind ein paar wenige, triviale Fälle darin abgelegt. Gründe hierfür mag es viele geben – Sie reichen von *keine Zeit* über *mangelnde Kenntnisse der Geschäftsabläufe* bis hin zu einer menschlichen Eigenschaft, die mit dem Attribut *Faulheit* vielleicht nur unzureichend umschrieben wäre. Das Resultat aber ist immer dasselbe: In der Entwicklung läuft die Applikation fehlerfrei und beim Kunden häufen sich die Probleme. Wäre es nicht toll, Sie als Entwickler und Tester hätten ein Werkzeug zur Hand, welches Ihnen Testfälle aus der produktiven Datenbank herauslöst und in Ihre Testdatenbank schreibt? Ein Werkzeug, das auch dann funktioniert, wenn Sie vom Kundensystem aus keinen Zugriff auf Ihr Testsystem haben? Genau ein solches Werkzeug für den SQL Server sollen Sie bekommen – inklusive Quelltext zum Weiterentwickeln und Anpassen an Ihre Wünsche. Der Name des Werkzeugs ist *CloneDB*.

kurz & bündig

Inhalt

Das Skripten von produktiven Datenbanken ist mit vielen Stolpersteinen versehen

Zusammenfassung

Die besten Testdaten kommen noch immer aus realen Kundeninstallationen – wie diese am effizientesten in die Testdatenbank kommen und warum die vertrauten Tools wenig taugen, ist Inhalt dieses Artikels

Doch ist es denn wirklich notwendig, ein solches Werkzeug selbst zu schreiben? Schließlich gibt Microsoft dem SQL Server bereits etliche Werkzeuge mit – ganz zu schweigen von der großen Anzahl an Produkten von Drittanbietern auf dem Markt. Der Autor hat jedenfalls keines gefunden; alle besitzen den einen oder anderen Mangel und wenn es nur die horrenden Lizenzpreise sind. Die Mängel der Microsoft-Tools wie *Enterprise Manager* oder *Query Analyzer* werden Sie nun auf den nächsten Seiten kennen lernen. Jetzt aber ran an die Arbeit – schliesslich soll Ihr kühles Bier ja nicht warm werden.

CloneDB generiert Skriptdateien

Die wichtigste Anforderung an CloneDB? Seine Fähigkeit zu funktionieren, auch wenn Produktiv- und Testsystem nicht gleichzeitig zur Verfügung stehen (Mangel 1). Am elegantesten lässt sich dies mit einer Skriptdatei lösen – CloneDB erstellt für die gewünschten Testdaten eine Skriptdatei mit T-SQL-Befehlen. Auf dem Zielsystem ausgeführt, füllt die Skriptdatei die Daten in die Testdatenbank ein. Ein solches Vorgehen hat noch weitere Vorteile: Für spezielle Bedürfnisse lässt sich die Skriptdatei vor dem Anwenden auf die Testdatenbank manuell anpassen (Mangel 2). Schreiben Sie um einzelne *INSERT*-Befehle eine *FOR*-Schlaufe und schon haben Sie eine große Testdatenbank mit Tausenden von Daten – ideal für Performanztests. Eine Skriptdatei lässt sich auch leicht per E-Mail einem Freund senden oder auszugsweise Ihrem Hilferuf in einer Newsgroup beifügen (Mangel 3) – so erhöhen Sie sich die Chance, dass Ihnen jemand bei Ihrem Problem konstruktiv zur Seite steht. Für die Tabelle *ProductCategory* aus der Datenbank *AdventureWorks-2000* könnte eine derartige Skriptdatei in etwa so wie in Listing 1 aussehen. Seit den *Reporting Services* ist *AdventureWorks-2000* die Lieblingstestdatenbank von Microsoft und löst die etwas triviale *Northwind* ab. *AdventureWorks2000* werden Sie auch wieder im SQL Server 2005 finden – Zeit also, sich allmählich daran zu gewöhnen. Alle Beispiele in dieser Reihe drehen sich um die Tabellen rund um *Product* – den entsprechenden Ausschnitt aus dem *Entity Relationship Diagram* (ERD) sehen Sie in Abbildung 1.

CloneDB berücksichtigt Abhängigkeiten

Beim Einfügen von Daten ist deren Abhängigkeit zu weiteren Datenbankobjekten zu berücksichtigen. Es gilt sowohl die *Referentielle Integrität* zu wahren (Mangel 4) wie auch *Check Constraints* (Mangel 5) nicht zu verletzen. Soll zusätzlich zu den Daten auch das Schema mit ins Scriptfile aufgenommen werden, gibt es noch weitere Abhängigkeiten zu berücksichtigen, z.B. *User Defined Data Types* oder *User Defined Functions* für das Bestim-

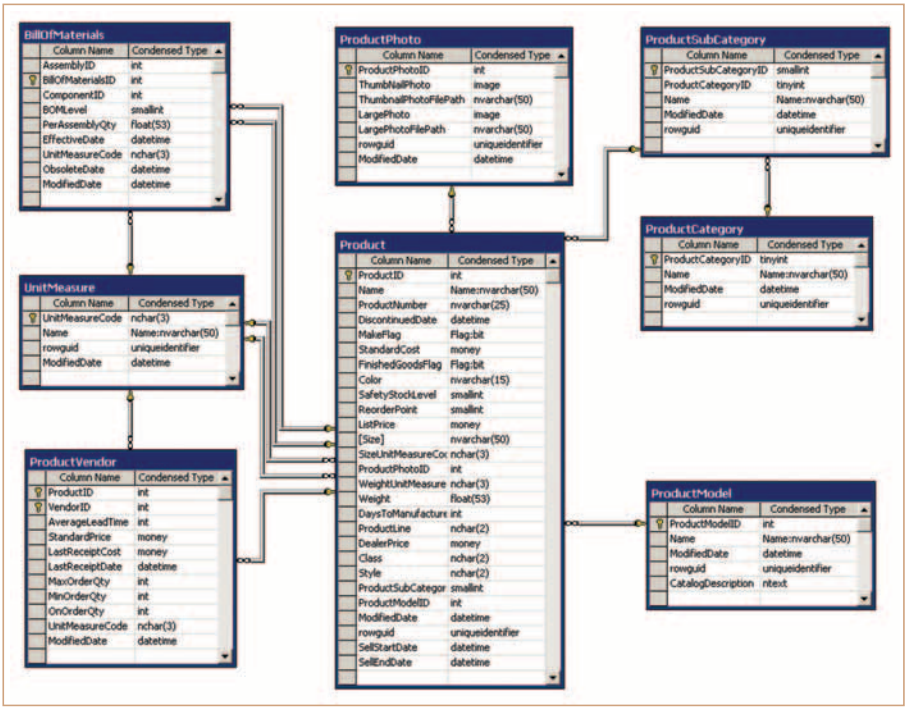

Abb. 1: Ausschnitt aus dem AdventureWorks2000 ERD

men von *Default*-Werten oder zum Überprüfen der *Check Constraints*. Ein Beispiel: Wollen Sie in Ihre Testdatenbank neue Einträge für die Tabelle *Product* aufnehmen, müssen Sie zuerst sicherstellen, dass die notwendigen Eintragungen in den Tabellen *UnitMeasure*, *ProductModel*, *ProductPhoto* sowie *ProductSubCategory* vorhanden sind. *Product* unterhält nämlich zu all diesen Tabellen eine Fremdschlüsselbeziehung. Es geht gar noch weiter: Um in *ProductSubCategory* einen Eintrag vornehmen zu können, müssen Sie zuerst noch *ProductCategory* überprüfen. Und wenn Sie in die Skriptdatei noch alle *CREATE-TABLE*-Befehle aufnehmen, müssen Sie sicher stellen, dass die beiden Datentypen *Flag* und *Name* definiert sind, andernfalls werden Sie die Spalten *Product.Name*, *Product.MakeFlag* und *Product.FinishGoodsFlag* nicht anlegen können.

Sie sehen also, bevor Sie auch nur ein einziges Produkt neu in Ihre Datenbank aufnehmen können, müssen Sie eine ganze Menge gewährleisten. Wie finden Sie nun heraus, von welchen Daten Ihr Produkt abhängt? Der erste Schritt besteht im Bestimmen der Abhängigkeiten der Tabelle *Product* und der zweite Schritt im Herausfiltern der notwenigen Datensätze aus den entsprechenden Tabellen. Wie lösen Sie den ersten Schritt am effizientesten? Natürlich, Sie können sich die notwendigen Informationen aus den verschiedensten Systemtabellen und Schema-Views herausholen – aber ist dies wirklich so einfach und effizient? Haben Sie sich schon einmal gefragt, wie dies der *Enterprise Manager* macht? Leider (oder vielleicht zum Glück) ist der Quellcode des Enterprise Manager kein Allgemeingut – Sie müssen sich daher einer List bedienen, um hinter seine Geheimnisse zu kommen. Beobachten Sie mit Hilfe des *SQL Server Profiler* welche Aufrufe der Enterprise Manager macht, wenn Sie sich die Abhängigkeiten der Tabelle *Product* anzeigen lassen. Sie werden einen Aufruf der folgenden Art finden:

```
exec sp_MSdependencies N'[dbo].[Product]', null, 1315327
```

Alleine schon der Name der Stored Procedure klingt verheißungsvoll. In den SQL Server Books Online (BOL) werden Sie allerdings vergeblich nach einer Beschreibung suchen – *sp_MSdependencies* ist nämlich eine von vielen undokumentierten Stored Procedures des SQL Server. Greifen Sie nochmals in die Trickkiste und führen Sie den folgenden Batch aus:

```
use master
exec sp_helptext N'sp_MSdependencies'
```

Jetzt sehen Sie den vollständigen Quellcode samt seinen Kommentaren für *sp_MSdependencies* (siehe Kasten „Bestimmen von Abhängigkeiten mithilfe von *sp_MSdependencies*“). Jetzt, da Sie wissen von welchen Tabellen *Product* abhängt, ist es ein Leichtes die notwendigen Datensätze

Listing 1

Skriptdatei für die Tabelle *ProductCategory* aus AdventureWorks2000

```
— Data for [dbo].[ProductCategory]
USE [AdventureWorks2000]
GO
SET NOCOUNT ON;
SET IDENTITY_INSERT [dbo].[ProductCategory] ON;
INSERT [dbo].[ProductCategory]
        ([ProductCategoryID], [Name],
                [ModifiedDate], [rowguid])
VALUES (1, 'Bike', 'May 28 2002 3:30PM',
'72AAA0D2-CA67-4930-BBB3-AAFAE2D3C8BC');
INSERT [dbo].[ProductCategory]
        ([ProductCategoryID], [Name],
                [ModifiedDate], [rowguid])
VALUES (2, 'Component', 'May 28 2002 3:30PM',
'3723C37B-599A-47FD-8754-F8161F9ACA71');
INSERT [dbo].[ProductCategory]
        ([ProductCategoryID], [Name],
                [ModifiedDate], [rowguid])
VALUES (3, 'Clothing', 'May 28 2002 3:30PM',
'25D81275-31FD-4926-A729-964FFB9818E6');
INSERT [dbo].[ProductCategory]
([ProductCategoryID], [Name], [ModifiedDate],
                [rowguid])
VALUES (4, 'Accessory', 'May 28 2002 3:30PM',
'14A200BD-CCBE-4FE2-92D5-C3BDFB68AF7E');
INSERT [dbo].[ProductCategory]
([ProductCategoryID], [Name], [ModifiedDate],
                [rowguid])
VALUES (5, 'Service', 'May 28 2002 3:30PM',
'C3FEE8DB-D0A7-421A-94F9-D06AB6E233C9');
SET IDENTITY_INSERT [dbo].[ProductCategory] OFF;
SET NOCOUNT OFF;
```

Listing 2

```
SELECT PC.*
FROM Product P LEFT JOIN ProductSubCategory PSC
        ON P.ProductSubCategoryID =
                PSC.ProductSubCategoryID
    LEFT JOIN ProductCategory PC
        ON PSC.ProductCategoryID =
                PC.ProductCategoryID
WHERE P.ProductID = 50
```

aus den einzelnen Tabellen rauszuholen. Verbinden Sie einfach alle Ihre Tabellen mit *JOIN* und wenden Sie die korrekte *WHERE*-Clause für das Filtern an.

Die Abfrage aus Listing 2 gibt Ihnen den notwendigen Datensatz aus *ProductCategory* für das Produkt 50. Bleibt nur noch die Frage, wie Sie am schnellsten auf die *Search Condition* der einzelnen *JOINS* kommen? Die Stored Procedure *sp_fkeys* liefert Ihnen hierzu alle notwendigen Informationen. Kurz zusammengefasst: Mithilfe von *sp_MSdependencies*, *sp_fkeys* und einem *SELECT-JOIN*-Statement bilden Sie alle *INSERT*-Statements in der korrekten Reihenfolge.

Bestimmt haben Sie sich schon mal gewundert, warum der *Enterprise Manager* die Abhängigkeiten für Ihre Views nicht korrekt anzeigt. Dies tritt immer dann auf, wenn Sie innerhalb von verschachtelten Views eine oder mehrere ändern. Der SQL Server führt in diesem Falle die Metadaten in den Systemtabellen nicht korrekt nach. Diese bringen Sie aber leicht wieder ins Lot: Benutzen Sie *sp_refreshview*. Wenn Sie dies vor dem Aufruf von *sp_MSdependencies* machen, kann nichts mehr schief gehen (Mangel 6).

CloneDB berücksichtigt auch spezielle Spalten

Nicht jede Tabellenspalte lässt sich via *INSERT*-Befehl ohne Weiteres setzen. Am Auffälligsten ist dies z.B. bei der Spalte *Product.ProductID*, welche als *Identity* gekennzeichnet ist. Identities werden in der Regel von der Datenbank selber vergeben. Möchten Sie diesen Wert doch einmal selber vorgeben, setzen Sie vor dem *INSERT*-Befehl folgendes Kommando ab:

```
SET IDENTITY_INSERT [dbo].[Product] ON;
```

Aber Achtung – zwei, drei Dinge sollten Sie dabei schon beachten:

1. Pro Datenbank kann jeweils nur eine Tabelle im Modus *IDENTITY_INSERT ON* sein. Vergessen Sie daher nach den *INSERT*-Befehlen nie das entsprechende *OFF*-Kommando (Listing 1):

```
Server: Msg 8107, Level 16, State 1, Line 1
IDENTITY_INSERT is already ON for table
'AdventureWorks2000.dbo.Contact'.
Cannot perform SET operation for table 'Product'.
```

2. Wenden Sie dieses Kommando nie auf eine Tabelle an, welche keine *IDENTITY*-Spalte besitzt – sonst erhalten Sie eine Fehlermeldung:

```
Server: Msg 8106, Level 16, State 1, Line 1
Table 'Support' does not have the identity property.
Cannot perform SET operation.
```

3. In der Regel werden Sie Identities für Primärschlüsselfelder verwenden. In diesem Falle existiert ein *UNIQUE constraint*. Auch mit *IDENTITY_INSERT ON* dürfen Sie gegen diese Bedingung nicht verstoßen; andernfalls werden Sie wieder eine Fehlermeldung erhalten:

```
Server: Msg 2627, Level 14, State 1, Line 1
Violation of PRIMARY KEY constraint 'PK_Product_
ProductID'.
Cannot insert duplicate key in object 'Product'.
```

Gerade der dritte Punkt kann besonders viel Ärger bereiten. Oft werden Sie Daten

Bestimmen von Abhängigkeiten mit Hilfe von *sp_MSdependencies*

Mithilfe der undokumentierten Stored Procedure *sp_MSdependencies* bestimmen Sie, von welchen Datenbankobjekten ein bestimmtes Objekt abhängig ist, aber auch was von diesem abhängt. Mit *sp_MSdependencies N'?'* erhalten Sie Microsofts Original-Aufrufdokumentation:

```
sp_MSdependencies name = NULL, type = NULL, flags = 0x01fd
 name: name or null (all objects of type)
 type: type number (see below) or null
            if both null, get all objects in database
 flags is a bitmask of the following values:
    0x10000 = return multiple parent/child rows per object
    0x20000 = descending return order
    0x40000 = return children instead of parents
    0x80000 = Include input object in output result set
    0x100000 = return only firstlevel (immediate) parents/children
    0x200000 = return only DRI dependencies
    power(2, object type number(s)) to return in results set:
       0 (1 - 0x0001)   - UDF
       1 (2 - 0x0002)   - system tables or MS-internal objects
       2 (4 - 0x0004)   - view
       3 (8 - 0x0008)   - user table
       4 (16 - 0x0010)  - procedure
       5 (32 - 0x0020)  - log
       6 (64 - 0x0040)  - default
       7 (128 - 0x0080) - rule
       8 (256 - 0x0100) - trigger
       12 (1024 - 0x0400)  - uddt
    shortcuts:
       29 (0x011c) - trig, view, user table, procedure
       448 (0x00c1) - rule, default, datatype
       4606 (0x11fd) - all but systables/objects
       4607 (0x11ff) - all
```

Beispiel

Sie wollen wissen, was und in welcher Reihenfolge Sie in der DB anlegen müssen, damit Sie erfolgreich die Tabelle *Products* aus der AdventureWorks2000-Datenbank kreieren können:

```
sp_MSdependencies N'Product', null, 528895

oType   oUDDTName          oOwner    oSequence
————————————————————————————————————————————-
4096    Flag               dbo       1
4096    Name               dbo       1

oType   oObjName           oOwner    oSequence
————————————————————————————————————————————-
8       ProductCategory    dbo       2
8       ProductModel       dbo       2
8       ProductPhoto       dbo       2
8       UnitMeasure        dbo       2
8       ProductSubCategory dbo       3
8       Product            dbo       4
```

Auf einen Blick sehen Sie, dass Sie erst die beiden User Defined Data Types *Flag* und *Name* anlegen müssen, bevor Sie der Reihe nach von der Tabelle *ProductCategory* bis *Product* gehen können.

auch aktiv in die Testdatenbank einfügen wollen, d.h. mittels der zu testenden Applikation. So kann es schon mal vorkommen, dass ein *Identity*-Wert bereits in der Testdatenbank vorhanden ist, wenn Sie Ihre Skriptdatei applizieren wollen. Dieses Problem lösen Sie einfach: Setzen Sie in Ihrer Testdatenbank den *Seed* für Identity sehr hoch – so hoch, wie es Ihr Produktivsystem im Normalfall nie bringen wird. So gewährleisten Sie, dass sich die beiden Systeme nie in die Quere kommen. Schlimmer wird es, wenn Sie Testfälle von verschiedenen Produktivsystemen sammeln. Kollisionen in *Identity*-Werten sind vorprogrammiert – Sie werden kaum die Möglichkeit haben, jedem Produktivsystem seinen eigenen Nummernkreis zu geben. Hier bleibt Ihnen dann nichts anderes übrig, als beim Einfügen der Testfälle in die Testdatenbank, die *Identity*-Werte zu überprüfen und gegebenenfalls anzupassen. Dass dies nicht ganz problemlos ist, lässt sich leicht erahnen: Sie müssen den Wert nämlich auch bei allen Fremdschlüsseln anpassen. So richtig schlimm wird es, wenn Sie Testfälle sukzessive von Zeit zu Zeit hinzufügen – was auch der Normalfall sein wird. Die Chance ist groß, dass Sie dieselben Stammdaten mehrmals einfügen wollen. Wenn aber die *Identity*-Werte in der Testdatenbank nicht mehr mit denjenigen aus dem Produktivsystem übereinstimmen, werden Sie es kaum bemerken, dass die Daten bereits vorhanden sind und diese ein zweites Mal einfügen. So werden Sie bald dasselbe Produkt mehrmals in Ihrer Testdatenbank vorfinden – jedes Mal mit einem anderen Primärschlüssel. Dies ist einer der Gründe, warum der Autor persönlich *GUIDs* (*uniqueidentifier*) als Primärschlüsselwert den *Identities* vorzieht.

Identities sind aber nicht die einzigen Werte, die beim Einfügen in die Datenbank einer Sonderbehandlung bedürfen – auch *Calculated Columns* müssen separat behandelt werden. Eine solche finden Sie zum Beispiel in der Tabelle *Customer* der *AdventureWorks2000*-Datenbank:

```
Customer.AccountNumber = [CustomerID]
```

Dieser Fall ist simpel – *Calculated Columns* lassen Sie in Ihrem INSERT- und UPDATE-Skript einfach weg. Genauso verhält es sich mit Spalten vom Typ *timestamp*. Entgegen dem SQL-92-Standard beschreibt eine Spalte vom Typ *timestamp* nicht eine Uhrzeit, sondern entspricht beim SQL Server viel mehr einer eindeutigen *Identity*, welche bei jedem INSERT- und UPDATE-Befehl automatisch angepasst wird. *Timestamp* ist also eine Art Versionsstempel über alle Datensätze in einer kompletten Datenbank. Mit SQL Server 2000 wurde daher auch das *timestamp*-Synonym *rowversion* eingeführt, welches mit Vorzug zu benutzen ist. Kein Wunder, dass Sie für diesen Datentyp keinen expliziten Wert angeben dürfen. In Ihren *INSERT*- und *UPDATE*-Befehlen lassen Sie diese Spalten einfach weg oder weisen ihnen den Wert *null* zu.

Sie wissen jetzt, dass Sie sowohl Identities als auch *Calculated Columns* und *timestamps* separat behandeln müssen. Wie aber ermitteln Sie am effizientesten, welche Spalten in Ihrer Tabelle von diesen Ausnahmen betroffen sind? Auch hierfür gibt es wiederum eine undokumentierte Stored Procedure:

```
exec sp_MShelpcolumns N'[dbo].[Product]'
```

Jede Spalte der Tabelle *Product* wird in einer Zeile beschrieben. Unter anderem existieren die Spalten *col_identity*, *col_iscomputed* und *col_basetypename*. Hier haben Sie alle Informationen, die Sie für die Sonderbehandlung einzelner Spalten aus Ihrer Testtabelle benötigen.

Ausblick

Wenn Sie beim Skripten Ihrer Testdatenbank alle diskutierten Punkte berücksichtigen, werden Sie bereits recht weit kommen. Natürlich gibt es noch eine Menge weiterer Spezialfälle zu beachten. Da Sie aber den Weg über eine Skriptdatei gewählt haben, ist nichts verloren. Passen Sie Ihre Skriptdatei mit einem Texteditor vor dem Anwenden auf die Testdatenbank Ihren Anforderungen an.

Wie war das gleich noch mit dem kühlen Bier? Richtig – Sie wollen diese Skripts ja nicht von Hand erstellen, sondern mit einem einfach zu erweiternden Tool generieren lassen und die so eingesparte Zeit lieber unter Freunden bei einem kühlen Bier verbringen. Dieses Tool finden Sie im Teil 2 in der nächsten Ausgabe des *dot.net magazin*. Dort werden Sie sehen, wie Sie mit dem Einsatz der richtigen Code Library sowie dem Know-how aus diesem Teil ein solches Tool effizient in C# selber entwickeln. Den aufmerksamen Lesern unter Ihnen ist bestimmt aufgefallen, dass Sie bis jetzt noch keine Lösung für das Erstellen der einzelnen *CREATE-TABLE*-Skripts gesehen haben. Dabei ist doch genau dies eine besonders komplexe Arbeit mit vielen Varianten. Keine Angst, in der kommenden Folge werden Sie auch hierfür eine einfache Lösung bekommen. Wagen Sie sich doch schon mal selber an dieses interessante Problem ran. Brauchen Sie eine kleine Starthilfe? Schreiben Sie mir ein E-Mail und ich sende Ihnen das Zauberwort. Viel Spaß und auf bald.

Urs Gehrig ist Head of System Services der bbv Software Services AG. Dieser Service hat sich auf die kundenorientierte Softwareentwicklung mit .NET spezialisiert. 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.

Links & Literatur

[1] SQL Server 2000 Books Online: www.microsoft.com/sql/techinfo/productdoc/2000/books.asp

[2] SQL Server Community: www.sqljunkies.com

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