HANSER


Leseprobe

Thorsten Kansy

Datenbankprogrammierung mit .NET 4.5

Mit Visual Studio 2012 und SQL Server 2012

Herausgegeben von Dr. Holger Schwichtenberg

ISBN (Buch): 978-3-446-43492-9

Weitere Informationen oder Bestellungen unter

http://www.hanser-fachbuch.de/978-3-446-43492-9

sowie im Buchhandel.

# 1 SQL Server 2012 – die Neuerungen

Microsoft bietet mit der neuen Version SQL Server 2012 eine ganze Reihe von Erweiterungen und Verbesserungen für den Entwickler. Darunter befinden sich komplett neue Funktionalitäten wie das FileTable-Feature, aber auch einige neue T-SQL-Befehle, die vieles einfacher machen. Dieses Kapitel gibt Ihnen einen tiefen Einblick in die Neuerungen. Um den Umfang dieses Buches nicht zu sprengen, musste jedoch auch eine Auswahl getroffen werden, sodass leider nicht alle Themen in voller Tiefe behandelt werden können. Andere, wenngleich zwar interessant, aber nicht im Fokus dieses Buches, können leider nur angekratzt werden oder müssen sogar komplett unerwähnt bleiben. Allerdings sei dem geneigten Leser versichert: Die spannendsten Neuerungen für Entwickler sind in diesem Kapitel zu finden.

Doch zunächst beginnen wir mit der wohl augenfälligsten Neuerung nach der Installation und dem ersten Start des SQL Server Management Studios.

## 1.1 SQL Server Management Studio

Mit dem wohlklingenden Codenamen „Juneau“ ist die wohl offensichtlichste Neuerung im SQL Server Management Studio 2012 angekündigt worden: SQL Server Management Studio (SSMS) verwendet nun die gleichen GUI-Komponenten wie Visual Studio 2010.

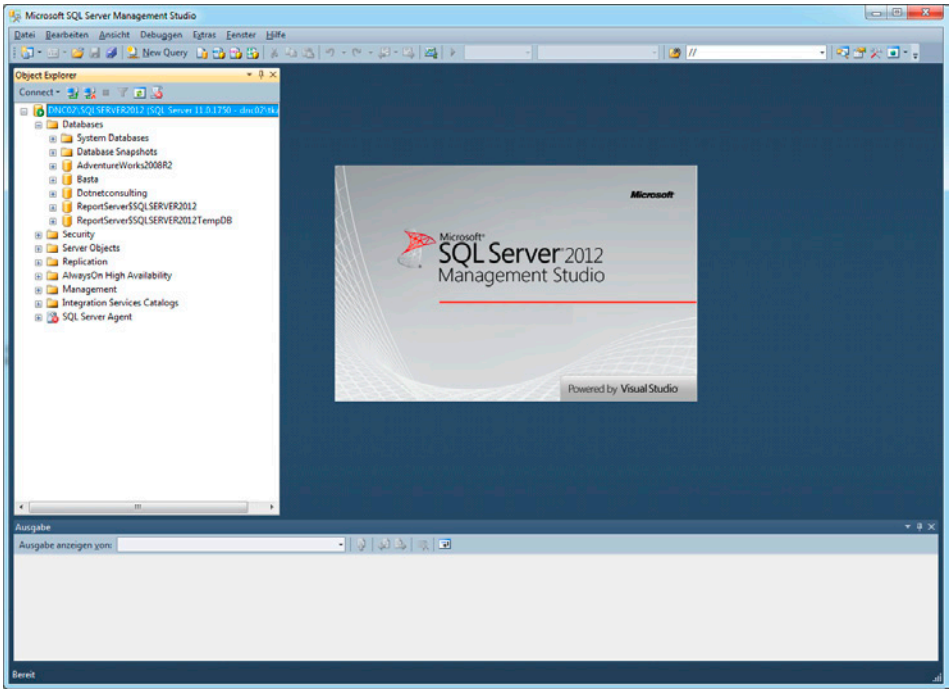



**Bild 1.1** Das SQL Server Management Studio im neuen Gewand

Auf diesem Weg sollen die vom Entwickler verwendeten Tools vereinheitlicht werden.

## 1.2 FileTable

Die Einführung von FileStream mit SQL Server 2008 war bereits ein Schritt in diese Richtung: Große, binäre Inhalte (auch Blobs genannt), wie z. B. Office-Dokumente, Audio- und Videodateien etc., konnten der Performance zuliebe im Dateisystem gespeichert werden. Inhalte wurden aufseiten der Tabellen in Spalten vom Typ `VARBINARY(MAX)` abgespeichert. Ein Zugriff war über T-SQL (ADO.NET) und eine entsprechende API möglich. Hier liegt allerdings auch der größte Nachteil: Eine Anwendung, die nicht über T-SQL auf die Daten zugreifen wollte, musste dies über eine spezielle API und in Zusammenarbeit mit dem SQL Server tun. Ein direkter Zugriff war nicht vorgesehen (obwohl theoretisch auch möglich) und jegliche Änderung an den Dateien führte zu einer Datenbankinkonsistenz. Das neue FileTable-Feature von SQL Server 2012 macht dies nun möglich: Ordner und Dateien werden als Tabelle abgebildet, können jedoch auch parallel von jeder Anwendung aus dem Dateisystem geöffnet und bearbeitet werden. Hier kommen die gewohnte Win32 IO-API oder die .NET-Klassen aus dem `System.IO`-Namensraum zum Einsatz.

Das FileStream-Feature wird in Abschnitt 2.1 näher beschrieben.

FileTables sind eine neue Art von Tabellen, die auch im SQL Server Management Studio getrennt von den bisherigen Tabellen mit relationalen Daten angezeigt werden.

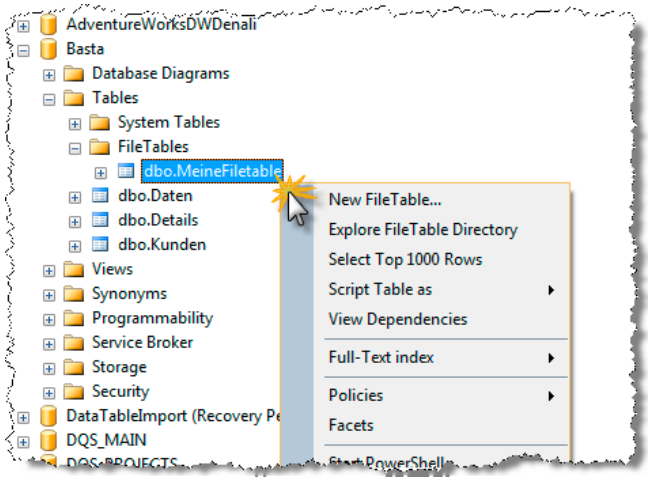



**Bild 1.2** FileTables im SQL Server Management Studio

Der Inhalt einer Tabelle sieht auszugsweise wie in Bild 1.3 aus.

Results | Messages

| | stream_id | file_stream | name | path_locator |
|---|---|---|---|---|
| 1 | ECDE8D38-7760-E111-BC1C-0026B90C6196 | NULL | Analysis Services | 0xFF92644853883B8 |
| 2 | A2E220E2-4961-E111-B1A1-0026B90C6196 | 0x48616C6C6F2057656C74 | HalloWelt.txt | 0xFF92644853883B8 |
| 3 | EA78EC7B-4F61-E111-B507-0026B90C6196 | 0x0D0A | Neues Textdokument (2).txt | 0xFF92644853883B8 |
| 4 | 7387F166-4F61-E111-B507-0026B90C6196 | 0x3C736476736473646673646673646 | Neues Textdokument.txt | 0xFF92644853883B8 |
| 5 | 948DFE8F-4F61-E111-B507-0026B90C6196 | 0x7361646E6A616B736464861736B6A646861736B6A646861... | Neues Textdokument (3).txt | 0xFF92644853883B8 |
| 6 | DEDC46A5-4F61-E111-B507-0026B90C6196 | 0x504B03041400060008000000210009248782810100008E0... | Microsoft Word-Dokument (neu).docx | 0xFF92644853883B8 |

**Bild 1.3** Der Inhalt einer FileTable

Die gleichen Daten werden übrigens in Bild 1.6 im Windows Explorer abgedruckt.

Geben Sie FileTables nach Möglichkeit nicht mittels `NSELECT *`-Anweisung aus, da dies konsequenterweise die Spalte `file_stream` mit einschließt. In dieser Spalte wird der komplette Inhalt der Datei gespeichert und damit ausgegeben – eine unnötige Belastung der Ressourcen von Server und Client.

FileTables werden von allen SQL Server-Editionen von Express bis hin zur größten Variante unterstützt.

Die Express Edition verfügt über ein Limit von maximal 10 GB Größe pro Datenbank. Daten und Dateien, die per FileStream oder FileTable gespeichert werden, fallen nicht unter dieses Limit. Hier ist theoretisch kein Limit gegeben und es kann so viel gespeichert werden, wie Platz auf der Partition vorhanden ist.

Die Daten werden in der Tat physikalisch auf der lokalen Festplatte (NTFS-Laufwerk) gespeichert. Auf den entsprechenden Ordner hat jedoch standardmäßig nicht einmal der Windows-Administrator per NTFS-Berechtigung Zugriff. Zwar kann der Administrator sich diesen Zugriff erzwingen, doch sollte davon (besonders schreibend) Abstand genommen werden. Zum einen sind die Informationen ohnehin gut verborgen und außerhalb des SQL Server kaum zu verwerten, zum anderen sorgen Änderungen dafür, dass die Datenbank unrettbar inkonsistent wird.

Dass die Dateien nicht einfach 1:1 im Ordner zu finden sind, liegt daran, dass zusätzlich auch eine ganze Reihe von Metadaten benötigt wird, um z. B. Sicherungen, Replikationen etc. zu ermöglichen.

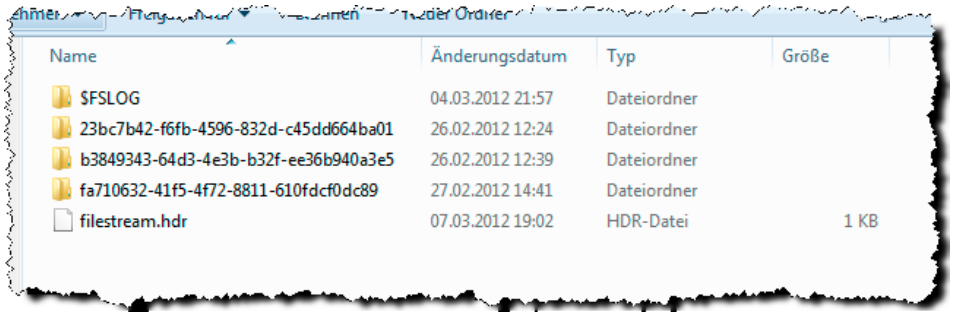



**Bild 1.4** Ein Blick in das FileStream-Verzeichnis

Der tatsächliche Zugriff auf die Dateien kann wie schon beschrieben via T-SQL (ADO.NET) oder Win32 IO-API erfolgen. Für Letzteres wird ein Pfad benötigt und dies ist im Falle des FileTable-Features ein UNC-Pfad. Dieses kann auch ohne Weiteres mit dem `NET SHARE`-Befehl in der Eingabeaufforderung aufgelistet werden.

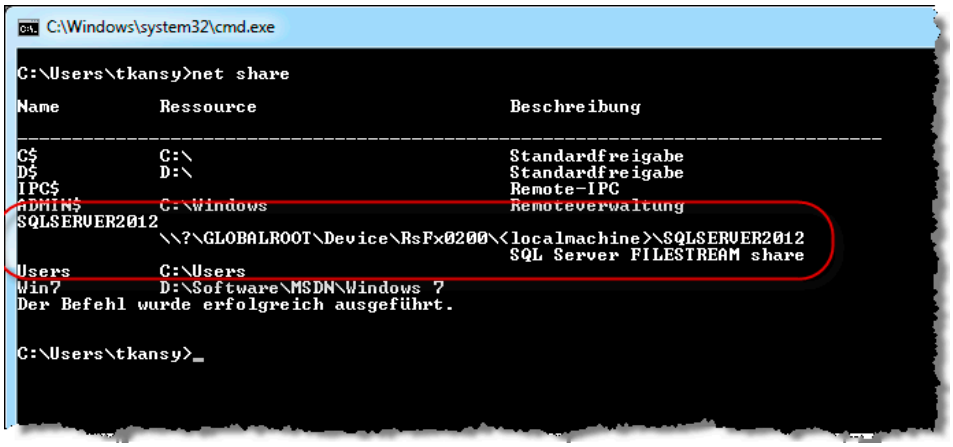



**Bild 1.5** Der Name der Freigabe mit dem kryptischen Verweis auf den SQL Server als Ressource

Wie in Bild 1.5 zu sehen ist, lautet der gewählte Name der Freigabe „SQLSERVER2012“ (dieser kann bei der Einrichtung bestimmt werden, siehe dazu Abschnitt 1.2.1, „Die Installation“). Über die Freigabe können Sie dann auf die Dateien zugreifen.

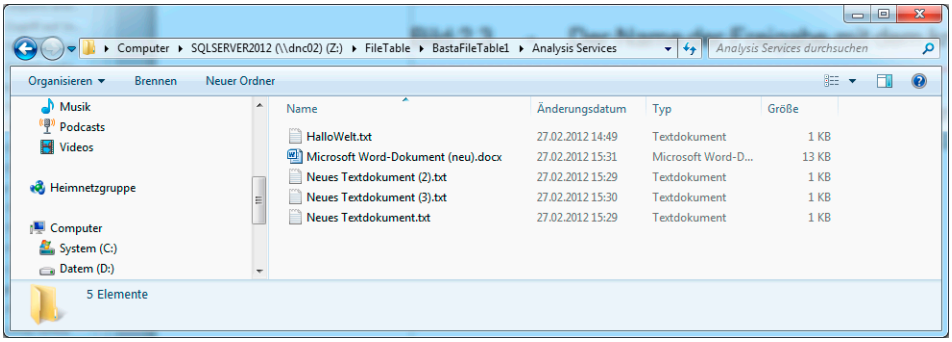



**Bild 1.6** Die Dateien im FileTable, zu sehen über den Windows Explorer

Selbstredend können die Dateien hier direkt mit der entsprechenden Anwendung wie z. B. Microsoft Word oder einfach nur Notepad dargestellt werden.

Um es nochmals ganz deutlich zu sagen: Es findet keine Synchronisation oder Ähnliches zwischen den Daten in der Tabelle und den Dateien in der Freigabe statt – es sind die gleichen Daten, die vom SQL Server auf der einen Seite als Tabelle und auf der anderen als Dateien bereitgestellt werden. Es gibt also keine Latenz oder Ähnliches zwischen beiden!

Nach diesen grundlegenden Erläuterungen geht es nun um die Einrichtung von FileTables.

### 1.2.1 Die Installation

Da FileTable technisch auf dem FileStream-Feature aufbaut, muss dieses ebenfalls aktiviert und konfiguriert werden. Dies geschieht entweder direkt während der Installation oder nachträglich über den SQL Server Configuration Manager (SQL Server 2012 > Configuration Tools > SQL Server Configuration Manager).

In Abschnitt 2.11 wird die Konfiguration des FileStream-Features beschrieben, sodass an dieser Stelle nur auf dieses Kapitel verwiesen sei.

Die nächste wichtige Einstellung befindet sich auf der Datenbankebene. Hier muss festgelegt werden, dass Zugriffe auf das FileStream-Feature ohne Transaktion möglich sind. Dies ist eine notwendige Voraussetzung für FileTable. In Abschnitt 1.2.4, „Transaktionen“, wird dieser wichtige Aspekt vertieft.

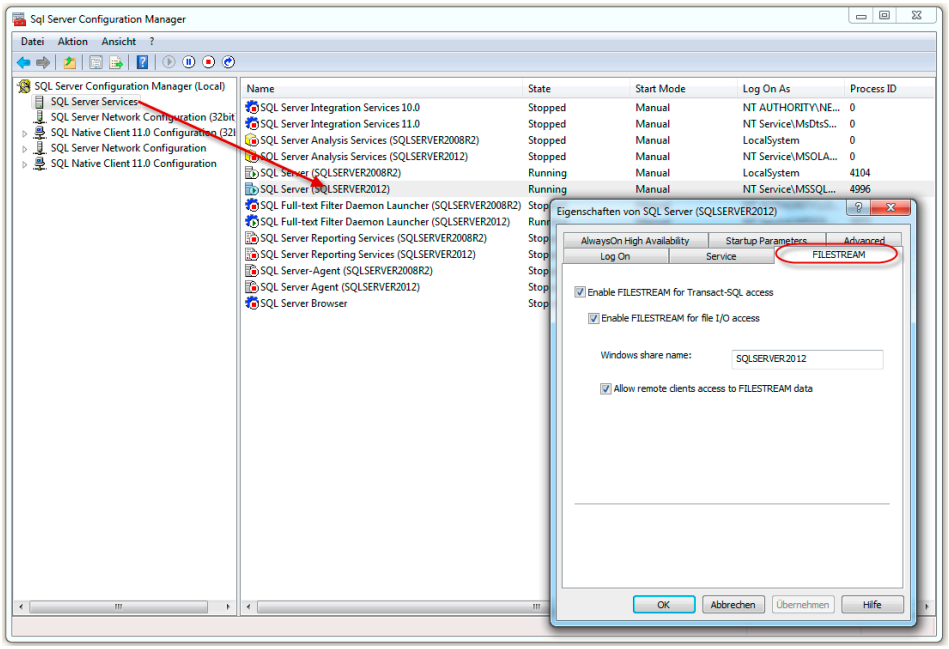



**Bild 1.7** Der SQL Server Configuration Manager mit den FileStream-Einstellungen

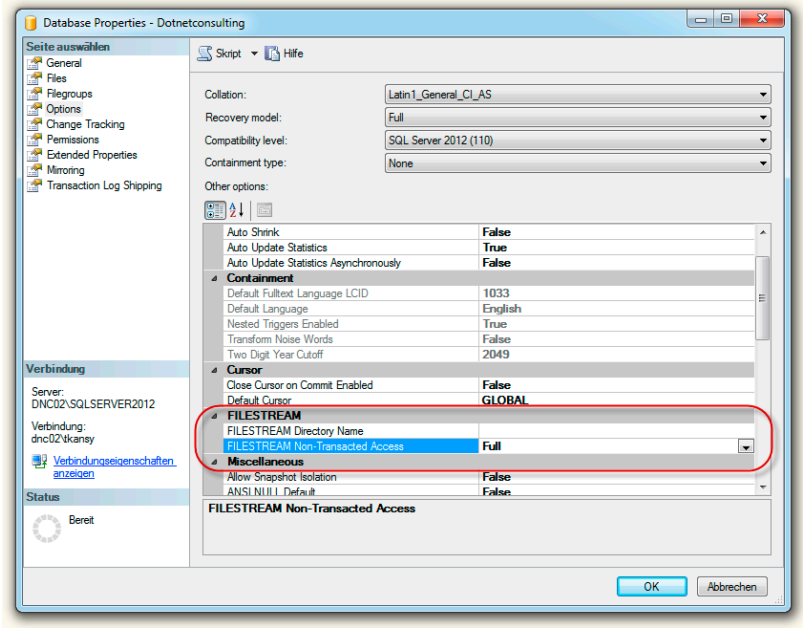


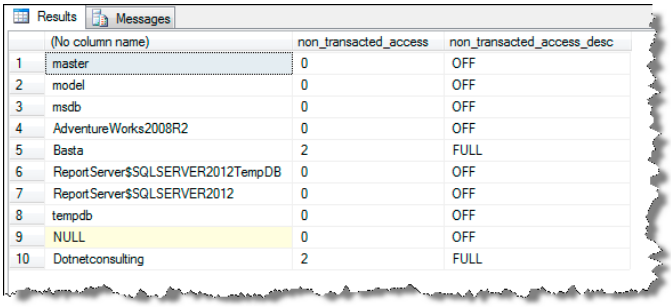

**Bild 1.8** Einstellungen für das FileTable-Feature auf Datenbankebene

Ob und welche Datenbank in diesem Bezug richtig konfiguriert ist, kann einfach mit dieser Abfrage ermittelt werden.

**Listing 1.1** Übersicht über die FileStream-Einstellung aller Datenbanken

```
SELECT DB_NAME(database_id),
       non_transacted_access, non_transacted_access_desc FROM sys.database_
filestream_options;
```

Das Ergebnis kann wie in Bild 1.9 aussehen.



Results | Messages

| | (No column name) | non_transacted_access | non_transacted_access_desc |
|---|---|---|---|
| 1 | master | 0 | OFF |
| 2 | model | 0 | OFF |
| 3 | msdb | 0 | OFF |
| 4 | AdventureWorks2008R2 | 0 | OFF |
| 5 | Basta | 2 | FULL |
| 6 | ReportServer$SQLSERVER2012TempDB | 0 | OFF |
| 7 | ReportServer$SQLSERVER2012 | 0 | OFF |
| 8 | tempdb | 0 | OFF |
| 9 | NULL | 0 | OFF |
| 10 | Dotnetconsulting | 2 | FULL |

**Bild 1.9** Übersicht über alle Datenbanken inklusive deren Transaktionseinstellungen

Die Datenbank ohne Namen (siehe Bild 1.9, Zeile 9) ist die interne Ressourcen-Datenbank des SQL Servers. Dies ist erkennbar, wenn der Wert der database_id-Spalte ausgegeben wird. Er ist bei besagter Ressourcen-Datenbank immer 32767. Weitere Details dazu finden Sie in der MSDN[1].

Bevor nun ein FileTable in der Datenbank erstellt werden kann, muss nun sichergestellt sein, dass eine FileStream-Dateigruppe mit mindestens einer Datenbankdatei vom Typ „FileStream Data“ existiert. Dies gehört eigentlich zur Konfiguration, sodass an dieser Stelle nur ein kurzes Skript gedruckt folgt, das beides via T-SQL anlegt.

**Listing 1.2** Anlegen einer Dateigruppe und Datei vom Typ „FileStream Data“

```
USE [master];
GO
ALTER DATABASE [Dotnetconsulting] ADD FILEGROUP [FileStreamGroup] CONTAINS
FILESTREAM;
GO
ALTER DATABASE [Dotnetconsulting] ADD FILE
( NAME = N'Dotnetconsulting_Blobs',
  FILENAME = N'D:\...\Dotnetconsulting_Blobs' ) TO FILEGROUP [FileStreamGroup];
```

Damit ist die Einrichtung so weit abgeschlossen und es kann damit begonnen werden, FileTable in der gewünschten Datenbank anzulegen. Abschnitt 1.2.2 zeigt, wie es geht.

[1] *http://msdn.microsoft.com/en-us/library/ms190940.aspx*

### 1.2.2 FileTable anlegen

Aktuell können FileTables nur direkt via T-SQL angelegt werden. Selbst wenn ein entsprechender Befehl im SQL Server Management Studio existiert, öffnet dieser nur ein Abfragefenster mit einer Vorlage, die noch von Hand bearbeitet werden muss, damit sie lauffähig wird.

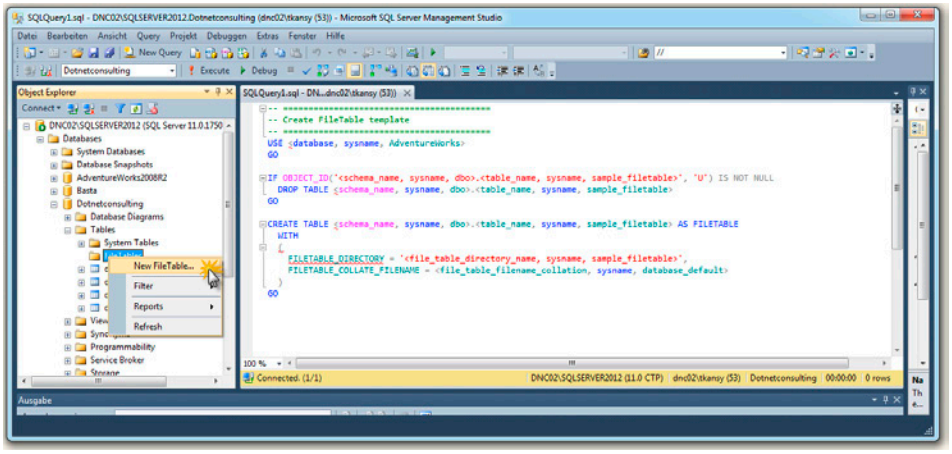



**Bild 1.10** Eine neue FileTable via SSMS erstellen

Die einfachste Anweisung, um ein FileTable anzulegen, ist die folgende. Das Wichtige und Neue bei dieser Anweisung ist der `AS FILETABLE`-Zusatz.

**Listing 1.3** Erstellung einer FileTable I

```
CREATE TABLE dbo.MeinFileTable AS FILETABLE;
```

Die Anweisung setzt allerdings voraus, dass ein „FILESTREAM Directory Name“, also ein Standardverzeichnis für die FileTables dieser Datenbank, existiert. Dies geschieht wie in Bild 1.8 gezeigt über die entsprechende Eigenschaft.

Ist diese Eigenschaft nicht gesetzt und kann auch nicht gesetzt werden, so muss ein Verzeichnis bei `CREATE TABLE` angegeben werden.

**Listing 1.4** Erstellung einer FileTable II

```
CREATE TABLE dbo.MeinFileTable2 AS FILETABLE
WITH
(
    FILETABLE_DIRECTORY = 'D:\Temp'
);
```

Somit verfügt die Datenbank über eine weitere, leere FileTable.

### 1.2.3 Berechtigungen

Berechtigungen, d. h. welcher Benutzer wie auf die Daten/Dateien zugreifen darf, wird einzig und alleine durch die SQL Server-Berechtigungen auf die FileTable bestimmt. Sie können also nicht über die Freigabe oder eine der so veröffentlichten Dateien/Ordner festgelegt werden.

Das hat einige tiefreichende Konsequenzen, die beim sicheren Einsatz von FileTable berücksichtigt werden müssen. Zunächst einmal kennt SQL Server nur ein Sicherheitskonzept, das (in diesem Fall) pro Tabelle gilt. Es legt fest, welche Rechte ein Benutzer innerhalb der Tabelle besitzt. So wird bestimmt, wer Dateien anlegen, lesen, verändern und löschen darf – analog zu dem, was sonst mit Zeilen in einer gewöhnlichen Tabelle geschehen kann. Im Gegensatz zu Berechtigungen auf NTFS-Ebene, die pro Datei festgelegt werden können, gilt das hier Gesagte allerdings nur für die gesamte FileTable.

Wird Windows-Authentifizierung in Form von Windows-Logins verwendet, so ist dies kein Problem. Anhand der Windows-Anmeldung kann damit sowohl beim Zugriff auf die Tabelle via T-SQL (ADO.NET) als auch beim Zugriff durch die Freigabe via Win32 IO-API bestimmt werden, ob der Zugriff erlaubt sein soll.

Die Windows-Authentifizierung ist bei SQL Server immer aktiv. Lediglich die SQL Server-Authentifizierung muss aktiviert werden (im gemischten Sicherheitsmodus).

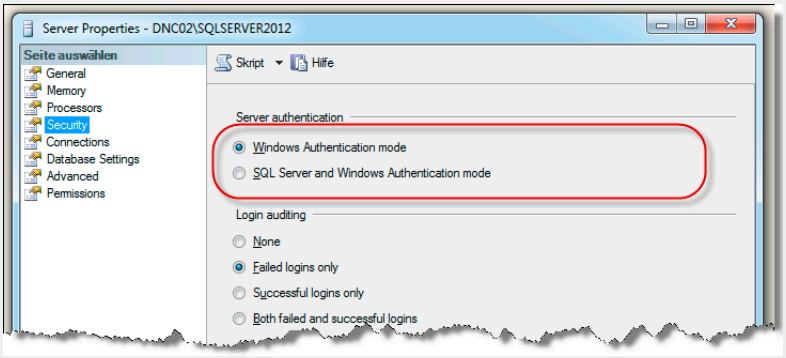



**Bild 1.11** Authentifizierungsmodus in den SQL Server-Eigenschaften

Für Details sei auf Abschnitt 2.5.2 verwiesen. Dort werden beide Sicherheitsmodi beschrieben.

Wird allerdings die SQL Server-Authentifizierung verwendet, bei der der SQL Server die Berechtigungen autrak bestimmen, so ist dies leider nicht so einfach. Aufseiten der Tabelle ändert sich zwar nichts, aber beim Zugriff mittels Win32 IO-API entsteht ein Problem. Der Anwender, der mit seiner SQL Server-Anmeldung zugreifen darf, erhält mit seinem Windows-Login eventuell keinen gleichartigen Zugriff.

Außer der Möglichkeit, doch (z. B. über einen Webserver) mit der Windows-Authentifizierung zu arbeiten, gibt es eigentlich nur die Lösung, dass jeder den gleichen Zugriff auf die Dateien hat[1]. In diesem Fall kann das Gast-Konto des SQL Servers für die betreffende Datenbank aktiviert und diesem das entsprechende Recht auf der FileTable zugewiesen werden.

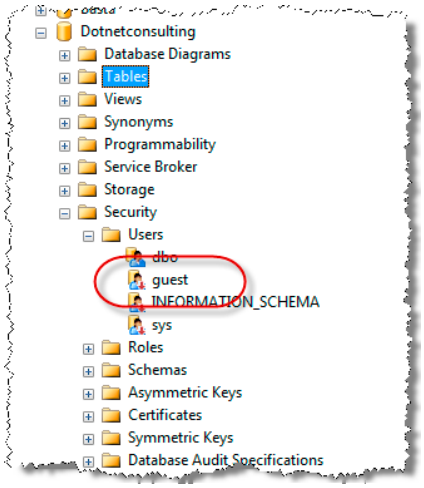



**Bild 1.12** Der Gast in der SQL Server-Datenbank

Damit der Anwender nicht einfach die Freigabe mittels Windows Explorer öffnen und sich so anschauen kann, welche Dateien und Ordner existieren, kann dem Gast das Recht *Definition anzeigen* entzogen werden. So müssen zumindest der Name und der Pfad der Datei für einen Zugriff bekannt sein.

Via SQL Server Management Studio geht dies wie folgt:

1. Aus dem Kontextmenü öffnen Sie die Eigenschaften der FileTable.
2. Dort klicken Sie *Berechtigungen* (*Permissions*) an.
3. Über „Suchen“ (*Search*) wählen Sie das Gast-Konto aus.
4. Im unteren Teil des Dialogs setzen Sie den entsprechenden Haken.
5. Den Dialog bestätigen Sie mit einem Klick auf die OK-Schaltfläche.

Alternativ geht dies per T-SQL wie folgt.

**Listing 1.5** Entzug des Rechts „Definition anzeigen“ für den Gast via T-SQL

```
DENY VIEW DEFINITION ON [dbo].[MeinFileTable] TO Guest;
```

Damit ist der Zugriff auch für einen dem SQL Server unbekannten Anwender realisierbar.

Wie der interessierte Leser bereits erkannt haben wird, sind FileTable- und SQL Server-Authentifizierung nicht wirklich gut kompatibel in der Praxis. Daher empfiehlt es sich, wenn möglich, die Windows-Authentifizierung zu verwenden.

[1] Natürlich nur, wenn es sich nicht um sensible Daten handelt.

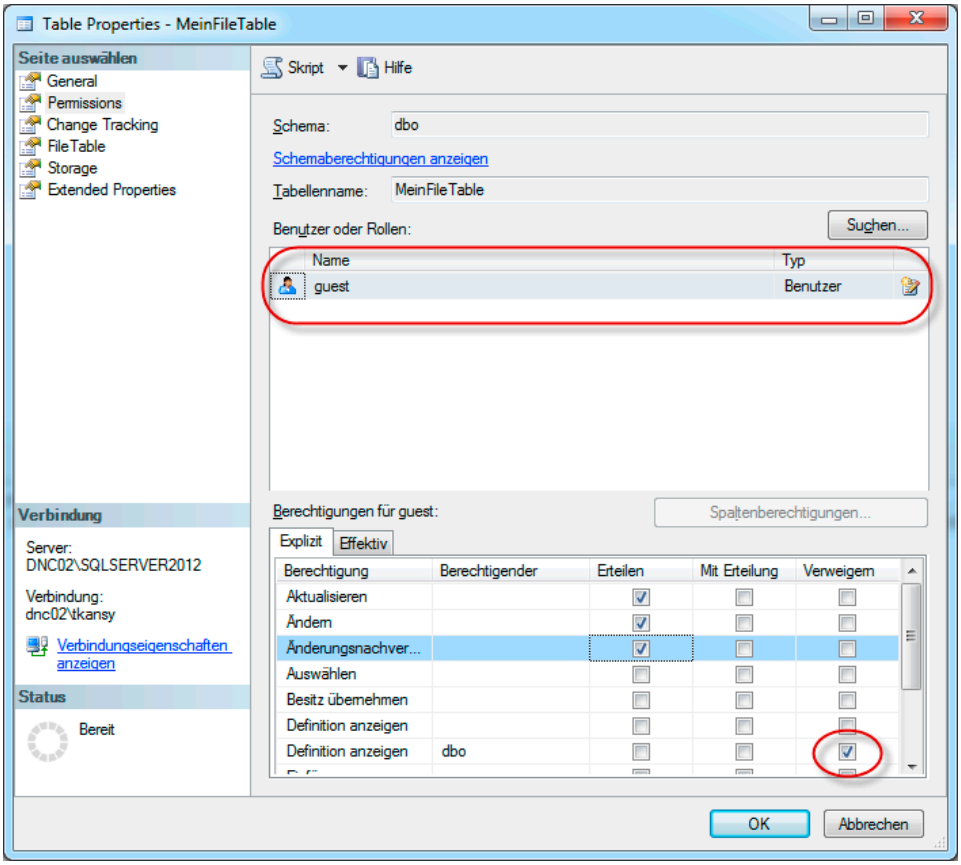



**Bild 1.13** Berechtigung via SQL Server Management Studio setzen

### 1.2.4 Transaktionen

Zugriffe (insbesondere Änderungen) auf FileTable-Tabellen unterliegen, wie schon erwähnt, keinen Transaktionen. Dies betrifft sowohl Zugriffe via Win32 IO-API als auch Zugriffe via T-SQL und sowohl implizite als auch explizite Transaktionen. Im Detail bedeutet dies:

- Umfangreiches Löschen von Daten/Dateien kann dazu führen, dass bei einem Abbruch nur ein Teil der gewünschten Daten/Dateien gelöscht wurde.
- In einer expliziten Transaktion (`BEGIN TRAN...COMMIT TRAN`) werden bei einem Zurückrollen der Änderungen (`ROLLBACK TRAN`) mögliche Änderungen an anderen Daten/Dateien (nicht FileTable) rückgängig gemacht.
- Das Transaktionsprotokoll verzeichnet Änderungen an FileTable nur rudimentär – eine Maßnahme, um die Größe nicht ins Unermessliche steigen zu lassen.
- Auch wenn keine Transaktionen greifen, Daten/Dateien in einer FileTable sind immer entweder gelöscht oder vorhanden. Hier ist nicht zu befürchten, dass eine Datei nur „halb“ vorhanden ist.

Werden diese Punkte bedacht und im Konzept einer Anwendung berücksichtigt, sind Zugriffe auf Daten/Dateien in einer FileTable wahrlich kein Hexenwerk.

Wie in Abschnitt 1.2.1, „Die Installation“, gezeigt, muss dieser nicht transaktionelle Zugriff für die Datenbank aktiviert werden. Geschieht dies nicht, sind die Daten/Dateien in einer FileTable entweder gar nicht oder nur lesend verfügbar.

## 1.2.5 Zugriff

Über den Zugriff auf die Dateien mittels Win32 API (`System.IO`) muss wohl nicht sehr viel geschrieben werden, da dieser jederzeit z. B. mittels Windows und seinem Explorer oder jeder anderen Anwendung möglich ist. Bei entsprechender Konfiguration stellt die SQL Server-Instanz eine passende Freigabe bereit, damit ein Zugriff von anderen Maschinen aus möglich ist. Der dabei benötigte (absolute) Pfad wird mit der `GetFileNamespacePath()`-Funktion ermittelt, auf die später noch eingegangen wird.

Um das Durchsuchen noch mehr zu vereinfachen, bietet das SQL Server Management Studio einen entsprechenden Ordner mit einem Befehl zum Öffnen des Pfades im Kontextmenü an.

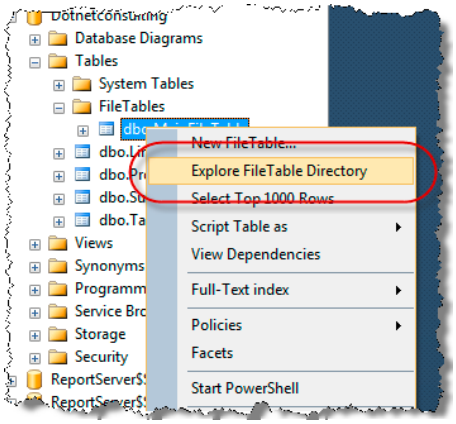



**Bild 1.14** Das Verzeichnis einer FileTable kann direkt im Windows Explorer geöffnet werden.

Auf der Seite von T-SQL (ADO.NET) stellt sich der Ordner als eine Tabelle mit festem Schema dar, die entsprechend der Berechtigungen alle Zugriffe vom Lesen über Änderung und Anlegen bis hin zum Löschen über die entsprechenden DML-Anweisungen (`SELECT`, `INSERT`, `UPDATE`, `DELETE` und `MERGE`) ermöglicht. Wichtig ist jedoch die Tatsache, dass Änderungen an FileTable-Tabellen keinerlei Transaktionen unterliegen. Somit hat z. B. ein `ROLLBACK TRANSACTION` schlicht keine Wirkung.

Tabelle 1.1 listet die vorhandenen Spalten und deren Zweck einer FileTable-Tabelle auf.

**Tabelle 1.1** Spalten einer FILETABLE-Tabelle und ihr Zweck

| Spaltenname | Zweck |
|---|---|
| `stream_id` | Eindeutige ID dieser Datei in der entsprechenden `FileTable`-Tabelle |
| `file_stream` | Diese Spalte spiegelt den binären Inhalt der Datei wider. Bei einfachen Inhalten wie z. B. .TXT kann dieser mit einer solchen Abfrage `SELECT [stream_id], CONVERT(VARCHAR(MAX), [file_stream]) FROM [Basta].[dbo].[MeineFiletable];` testweise ausgelesen werden. |
| `Name` | Name der Datei/des Verzeichnisses |
| `path_locator` | Diese Spalte beschreibt mit einem Wert vom Typ `HIERARCHYID` die Position der Datei/des Verzeichnisses in der Hierarchie der Verzeichnisstruktur. |
| `parent_path_locator` | Ebenfalls ein Wert vom Typ `HIERARCHYID`, der allerdings die Position des übergeordneten Verzeichnisses beschreibt oder mit `NULL` anzeigt, dass dieses Element im Basisverzeichnis liegt |
| `file_type` | Dateierweiterung der Datei (theoretisch auch Verzeichnis) |
| `cached_file_size` | Größe der Datei in Bytes. Dieser Wert ist aus Gründen der Performance zwischengespeichert und muss daher nicht immer stimmen. |
| `creation_time` | Zeitpunkt der Erstellung |
| `last_write_time` | Zeitpunkt der letzten Änderungen |
| `last_access_time` | Zeitpunkt des letzten Zugriffs |
| `is_directory` | Gibt an, ob dies ein Verzeichnis ist. Andernfalls ist es logischerweise eine Datei. |
| `is_offline` | Attribut „Offline“ der Datei oder des Verzeichnisses |
| `is_hidden` | Attribut „Hidden“ der Datei oder des Verzeichnisses |
| `is_readonly` | Attribut „Read Only“ der Datei oder des Verzeichnisses |
| `is_archive` | Attribut „Archive“ der Datei oder des Verzeichnisses |
| `is_system` | Attribut „System“ der Datei oder des Verzeichnisses |
| `is_temporary` | Attribut „Temporary“ der Datei oder des Verzeichnisses |

Jede FileTable-Tabelle ist eine Tabelle mit festem, d. h. unveränderlichem Aufbau.

Um weitere Informationen („Metadaten“) an Dateien/Verzeichnisse zu binden, bietet es sich an, eine zweite Tabelle mit den benötigten Informationen zu erstellen und über eine 1:1-Beziehung mit der FileTable-Tabelle zu verbinden.

### 1.2.6 Rekursive Zugriffe

Da FileTables Verzeichnisse abbilden, die auch rekursiven Inhalt beherbergen können (ein Verzeichnis beinhaltet ein weiteres), müssen auch Abfragen, welche diese Verschachtelung berücksichtigen, rekursiv ausgelegt sein. Zu diesem Zweck besitzt T-SQL allgemeine Tabellenausdrücke (Common Table Expressions, CTE).

Allgemeine Tabellenausdrücke können nicht nur mit FileTables verwendet werden. Vielmehr stellen sie eine sehr elegante Möglichkeit dar, rekursive Abfragen für den SQL Server zu schreiben. In Abschnitt 3.5.6 erfahren Sie mehr darüber.

Kern sind hierbei die Spalten `path_locator` und `parent_path_locator`, die Auskunft über die eigene Lage in der Hierarchie und die des übergeordneten Ordners geben (falls vorhanden, sonst `NULL`).

Die folgende Abfrage zeigt, wie ein solcher allgemeiner Tabellenausdruck aussehen kann, der alle Verzeichnisse nebst ihren Inhalten (weitere Verzeichnisse und Dateien) ausgibt.

**Listing 1.6** Rekursive Abfrage einer FileTable-Tabelle

```
-- Wie soll der Pfad ausgegeben werden?
DECLARE @AbsolutePath INT = 1;
-- oder
DECLARE @RelativePath INT = 0;

WITH FileTableCTE (name, path_locator, parent_path_locator, cached_file_size,
[level])
AS
(
    -- Anker
    SELECT file_stream.GetFileNamespacePath(@AbsolutePath),
           path_locator, parent_path_locator,cached_file_size, 0
        FROM dbo.MeinFileTable WHERE parent_path_locator IS NULL

    UNION ALL

    -- Rekursion
    SELECT file_stream.GetFileNamespacePath(@AbsolutePath),
            ft.path_locator,
            ft.parent_path_locator,
            ft.cached_file_size,
             [level] + 1
            FROM dbo.MeinFileTable ft INNER JOIN FileTableCTE cte
    ON ft.parent_path_locator = cte.path_locator
)
SELECT * from FileTableCTE ORDER BY Level;
```

Die Abfrage verwendet die `GetFileNamespacePath()`-Funktion, die in Abschnitt 1.2.8, „GetFileNamespacePath()“, genauer beschrieben wird, um auf den Pfad für den Zugriff via Win32 IO-API zuzugreifen.

Sowohl `path_locator` als auch `parent_path_locator` sind beides Spalten vom Typ `HIERARCHYID`, auch wenn dies für das rekursive Durchlaufen von FilteTable direkt keine große Rolle spielt. Nähere Details zu diesem Datentyp finden Sie in Abschnitt 2.12.

Ähnlich sieht es aus, wenn Daten gelöscht werden müssen. Beim Löschen ist zudem wichtig, dass in der richtigen Reihenfolge gelöscht wird: zuerst die Dateien/Order, die am tiefsten verschachtelt sind, anschließend die der nächsten Ebene usw. Schon die Abfragen in Listing 1.6 geben die Verschachtelungstiefe als `Level`-Spalte zurück. Diese wird in der nächsten Abfrage zum geordneten Löschen (siehe Listing 1.7) verwendet.

**Listing 1.7** Löschen des komplettes Inhaltes einer FileTable-Tabelle

```
-- Inhalte auflisten. "Tiefste" Elemente zuerst
WITH FileTableCTE (stream_id, path_locator, [level])
AS
(
    -- Anker
    SELECT stream_id, path_locator, 0
        FROM dbo.MeinFileTable WHERE parent_path_locator IS NULL

    UNION ALL

    -- Rekursion
    SELECT ft.stream_id,
            ft.path_locator,
            [level] + 1
            FROM dbo.MeinFileTable ft INNER JOIN FileTableCTE cte
    ON ft.parent_path_locator = cte.path_locator
)
SELECT stream_id INTO #Temp FROM FileTableCTE ORDER BY Level ASC;

-- Löschen der Inhalte
DELETE dbo.MeinFileTable WHERE stream_id IN (SELECT * FROM #Temp);
```

Auf diese Weise wird systematisch der komplette Inhalt gelöscht – Filter sind natürlich möglich.

Soll der komplette Inhalt einer Tabelle gelöscht werden, kann es unter bestimmten Umständen sinnvoll sein, dies mittels der `TRUNCATE TABLE`-Anweisung zu tun. Voraussetzungen sind, dass es keine Fremdschlüssel gibt und selbstverständlich dass die notwendige Berechtigung vorhanden ist. Details finden Sie in Abschnitt 3.5.3 dieses Buches.

### 1.2.7 FileTableRootPath()

Die `FileTableRootPath()`-Funktion liefert das Basisverzeichnis einer FileTable-Tabelle in UNC-Notation. Der Parameter der Funktion ist dabei der Name der Tabelle. Die Abfrage `SELECT FILETABLERootPath('dbo.MeineFILETABLE');` kann also ein solches Ergebnis liefern: `\\DNCX99\SQLSERVER2012\FileGroup\MeineFileTableVerzeichnis`. Ein optionaler zweiter Parameter erlaubt es, genau zu bestimmen, in welchem Format die Rückgabe stattfinden soll (z. B. ob der Name des Servers vollqualifiziert werden soll).

Die erlaubten Werte für den zweiten, optionalen Parameter sind in Tabelle 1.2 zu sehen.

**Tabelle 1.2** Werte zum Steuern der Rückgabe der FileTableRootPath()-Funktion

| Wert | Effekt |
|---|---|
| 0 | Liefert den Servernamen im NetBIOS-Format (Großbuchstaben etc.):<br>`\\DNCX99\SQLSERVER2012\FileGroup\MyFileTableDirectory`<br>Dies ist der Standardwert, der auch angenommen wird, wenn kein zweiter Parameter übergeben wird. |
| 1 | Liefert den Servernamen ohne irgendwelche Formatierung:<br>`\\DNCX99\SQLServer2012\FileGroup\MyFileTableDirectory` |
| 2 | Liefert den vollqualifizierten Servernamen inklusive seiner Domäne:<br>`\\DNCX99.dotnetconsulting.eu\SQLServer2012\FileGroup\MyFileTableDirectory` |

### 1.2.8 GetFileNamespacePath()

Diese Funktion ist wahrscheinlich die wichtigste, um auf Dateien mittels Win32 IO-API (`System.IO`) zuzugreifen. Sie liefert den kompletten Pfad der Datei, der dabei auf Wunsch absolut oder relativ zum Basisverzeichnis der FileTable-Tabelle sein kann. Da diese Funktion bei absoluten Pfaden mit der zuvor beschriebenen `FileTableRootPath()`-Funktion arbeitet, kann wieder das Format der Rückgabe über einen optionalen Parameter beeinflusst werden. Etwas speziell ist der Aufruf dieser Funktion, der als Methode der `file_stream`-Spalte geschieht. Dies sieht wie folgt aus.

**Listing 1.8** Absolute und relative Pfade für den Zugriff via Win32 IO-API

```
SELECT file_stream.GetFileNamespacePath(1) AS Absolut,
       file_stream.GetFileNamespacePath(0) AS Relativ
FROM dbo.MeineFileTable;
```

Durch ein Prädikat (`WHERE`) kann selbstverständlich eine Einschränkung durchgeführt werden. Die Rückgabe erfolgt jeweils einmal aus dem absoluten und einmal aus dem relativen Pfad der entsprechenden Datei. Ein Programm hat damit alle benötigten Informationen, um auf eine Datei oder ein Verzeichnis zuzugreifen.

### 1.2.9 FileTable-Tabellen in der Datei ermitteln

Um zu bestimmen, welche FileTable-Tabellen in einer Datenbank existieren, gibt es mehrere Wege. Zum einen ist eine Abfrage auf den Datenbankkatalog via `SELECT * FROM sys.FILETABLES;` möglich. Zum anderen kann eine Abfrage auf die altbewährte `sys.tables`-Methode durchgeführt werden, bei der auf die neue `is_FILETABLE`-Spalte gefiltert werden kann, um keine herkömmlichen Tabellen zu liefern.

**Listing 1.9** Ausgeben, welche FileTables in der aktuellen Datenbank existieren

```
SELECT * FROM sys.tables WHERE is_FILETABLE = 1;
```

Als Drittes wäre auch eine Abfrage auf die ANSI-kompatible Sicht `INFORMATION_SCHEMA.Tables` möglich, doch diese lässt keinen Rückschluss darauf zu, ob es sich bei einer Tabelle um eine FileTable-Tabelle handelt oder nicht – beide Arten von Tabellen werden zusammen zurückgegeben.

### 1.2.10 ServerProperty

Für das FileStream- und damit auch das FileTable-Feature gib es drei Servereigenschaften, die wie folgt abgefragt werden können.

**Listing 1.10** Abfragen einer Servereigenschaft mittels SERVERPROPERTY

```
SELECT SERVERPROPERTY('FilestreamShareName');
```

Die Ausgabe im SQL Server Management Studio sieht wie in Bild 1.15 aus.

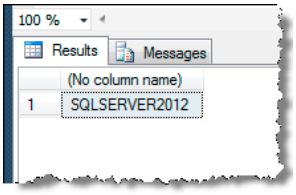



**Bild 1.15** Die Ausgabe der SERVERPROPERTY()-Funktion

Auf diesem Weg lassen sich auch aus einer .NET-Anwendung heraus gezielt gewünschte Werte abfragen.

**Tabelle 1.3** Für FileStream/FileTable interessante Eigenschaften

| Eigenschaft | Rückgabe |
|---|---|
| `FilestreamShareName` | Der Name der von FileStream verwendeten Freigabe |
| `FilestreamConfiguredLevel` | Die konfigurierte FileStream-Zugriffsebene |
| `FilestreamEffectiveLevel` | Die effektive FileStream-Zugriffsebene. Dieser Wert kann von `FilestreamConfiguredLevel` abweichen, wenn die Ebene geändert wurde und ein Neustart der Instanz oder des Computers aussteht. |

Weitere Details über die SERVERPROPERTY()-Funktion und viele weitere Servereigenschaften finden Sie in der MSDN unter folgendem Link: *http://msdn.microsoft.com/de-de/library/ms174396.aspx*.

## 1.3 Sequenzen

Sequenzen (*engl.* sequences) sind, vereinfacht gesagt, die geordnete Abfolge numerischer Werte, deren aktueller Wert für die gesamte Datenbank gilt und deren nächster Wert automatisch ermittelt wird, wenn dessen aktueller Wert abgefragt wird. Der große Unterschied zu Identitätswerten (*engl.* identity values) ist, dass dies für eine gesamte Datenbank Gültigkeit hat und nicht nur für eine bestimmte Tabelle. Außerdem wird, unabhängig davon, von wo (Tabelle, Prozedur, Funktion etc.) der aktuelle Wert einer Sequenz abgerufen wird, vom SQL Server sichergestellt, dass dieser Wert nur einmal vergeben wird, bevor der nächste ermittelt wird. Mögliche Ausnahme sind natürlich zirkuläre Sequenzen, die wir in einem der folgenden Abschnitte besprechen werden. Dass keine „Lücken" entstehen, z. B. durch Transaktionen, die zurückgerollt werden, ist jedoch nicht garantiert. Später erfahren Sie mehr zu diesem Aspekt.

Sequenzen verwalten immer numerische Werte und können linear sein (also von einem Start- bis zu einem Endwert verlaufen) oder zirkulieren – je nach Wunsch.

### 1.3.1 Sequenz anlegen (linear)

Beginnen wir zunächst mit der Erstellung einer Sequenz. Dies geschieht alternativ mit der folgenden T-SQL-Anweisung.

**Listing 1.11** Eine Sequenz per T-SQL anlegen

```
CREATE SEQUENCE MeineSequenz
    AS INT
    MINVALUE 1
    MAXVALUE 1000
    START WITH 1;
```

Eine Sequenz verfügt über einen (datenbankweit) eindeutigen Namen, einen numerischen Datentyp (erlaubt sind hier TINYINT, SMALLINT, INT, BIGINT, DECIMAL und NUMERIC), ein Minimum, ein optionales Maximum und einen Startwert, der beim ersten Zugriff zurückgeliefert wird, bevor der nächste Wert ermittelt wird. Wie der nächste Wert zu ermitteln ist, kann mit INCREMENT BY bestimmt werden, sodass die Sequenz alternativ auch folgendermaßen angelegt werden könnte.

**Listing 1.12** Eine Sequenz mit INCEREMENT BY per T-SQL anlegen

```
CREATE SEQUENCE MeineSequenz
    AS INT
    MINVALUE 1
    NO MAXVALUE
    INCREMENT BY 3
    START WITH 1;
```

Hier sehen Sie auch, dass es kein direktes Maximum geben muss. Lediglich der größte Wert des zugrunde liegenden Datentyps ist hier das tatsächliche Maximum. Wird darüber hinaus versucht, weitere Werte abzurufen, kommt es zu einem Fehler.

Eine Sequenz lässt sich aber auch über die neue Version des SQL Server Management Studio (SSMS) verwalten. Zu diesem Zweck gibt es im Objekt Explorer nun in jeder Datenbank unter *Programmierbarkeit/Sequenzen* (oder *Programmability/Sequences*) einen entsprechenden Container.

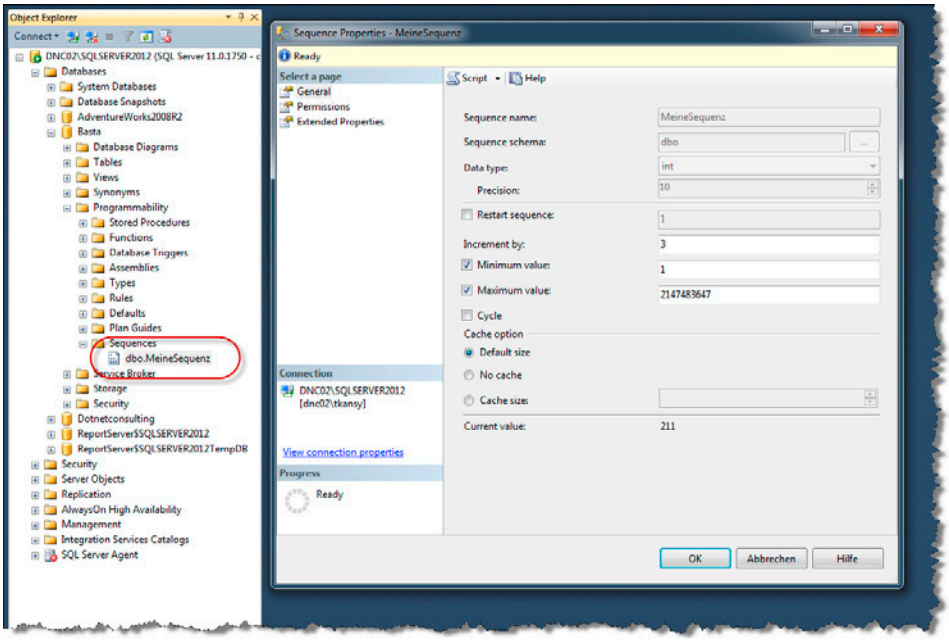



**Bild 1.16** Sequenzen lassen sich auch mit dem SSMS verwalten.

Welche Sequenzen zurzeit in der Datenbank mit welchen Eigenschaften existieren, lässt sich zusätzlich per T-SQL durch die folgende Abfrage ermitteln.

```
SELECT * FROM sys.sequences;
```

### 1.3.2 Sequenz anlegen (zirkulär)

Ein besonders interessantes Detail besteht darin, dass die Sequenzen zirkuläre Werte vergeben können. Das folgende Bespiel liefert die Werte von 1 bis 50 und beginnt anschließend wieder am Anfang mit dem Wert 1.

**Listing 1.13** Eine Sequenz mit CYCLE per T-SQL anlegen

```
CREATE SEQUENCE CyclingSequence
    AS INT
    MINVALUE 1
    MAXVALUE 50
    CYCLE;
```

Selbstredend müssen `MINVALUE` und `MAXVALUE` angegeben werden und mit `sys.sp_sequence_get_range` darf nicht versucht werden, mehr Werte abzurufen, als überhaupt möglich sind. Ansonsten ist die Entnahme von Werten vom Vorgehen her so, wie es zuvor bei nicht zirkulären Sequenzen gezeigt wurde.

### 1.3.3 Werte aus Sequenzen abrufen

Um auf den nächsten Wert einer Sequenz zuzugreifen, existiert die ebenfalls neue `NEXT VALUE FOR`-Anweisung, die wie folgt verwendet werden kann.

**Listing 1.14** Abfrage eines einzelnen Sequenzwertes

```
DECLARE @id INT = NEXT VALUE FOR MeineSequenz;
```

Die Variable enthält nach der Ausführung dieser Zeile den zu der Zeit aktuellen Wert der Sequenz und der nächste Wert wurde ermittelt.

Wird zu einem Zweck mehr als nur ein Wert benötigt, so wäre ein mehrfacher Aufruf von `NEXT VALUE FOR` in einer Schleife sicherlich recht ineffizient. Daher existiert nun die `sys.sp_sequence_get_range`-Prozedur, die es erlaubt, gleich eine gewünschte Anzahl von Werten abzufragen.

**Listing 1.15** Abfrage mehrerer Sequenzwerte

```
DECLARE @firstValue SQL_VARIANT,
        @lastValue SQL_VARIANT;

EXEC sys.sp_sequence_get_range
    @sequence_name = 'MeineSequenz',
    @range_size = 12,
    @range_first_value = @firstValue OUTPUT,
    @range_last_value = @lastValue OUTPUT;

SELECT FirstValue = CONVERT(INT, @firstValue),
       LastVlaue = CONVERT(INT, @lastValue);
```

Die beiden Output-Parameter `@firstValue` und `@lastValue` enthalten nach dem Aufruf den ersten und letzten Wert einer Folge von lückenlosen Sequenzwerten, deren Anzahl durch den `@range_size`-Parameter bestimmt wurde. Diese Werte können dann benutzt werden, ohne dass `NEXT VALUE FOR` verwendet werden muss.

### 1.3.4 Transaktionen

Sequenzen arbeiten unbeachtet von Transaktionen, d. h., der nächste Wert einer Sequenz wird nicht davon beeinflusst, ob eine begonnene Transaktion zurückgerollt wird oder nicht. Ähnlich verhalten sich auch Identitätswerte. Hier entstehen Lücken in den vergebenen Werten, wenn Transaktionen zurückgerollt werden.

**Listing 1.16** Transaktionen sind für Sequenzen transparent.

```
BEGIN TRANSACTION
-- In der Transaktion einen Wert aus der Sequenz abrufen
DECLARE @id INT = NEXT VALUE FOR MeineSequenz;
SELECT @id
GO
ROLLBACK TRANSACTION

-- Ungeachtet des ROLLBACK wird der nächste Wert aus der Sequenz abgerufen
DECLARE @id INT = NEXT VALUE FOR MeineSequenz;
SELECT @id;
```

Dieses Verhalten ist „by Design“ und auch nicht anders realisierbar, da ein einmal abgefragter Wert nicht wieder „zurückgezogen“ werden kann – schließlich wird er an anderer Stelle ja bereits verwendet.

Mehrere Werte, die mit `NEXT VALUE FOR` einer Sequenz entnommen wurden, müssen außerdem auch nicht lückenlos sein (Ausnahme `sys.sp_sequence_get_range`).

### 1.3.5 Ändern und Löschen

Bestehende Sequenzen können selbstverständlich auch geändert und gelöscht werden. Besonders um eine Sequenz „von vorne beginnen zu lassen“, ist dies recht praktisch. Dies ist mit dem SQL Server Management Studio oder per T-SQL möglich.

**Listing 1.17** ALTER SEQUENCE zum Zurücksetzen einer Sequenz

```
ALTER SEQUENCE MeineSequenz
RESTART WITH 10
INCREMENT BY 10
NO MAXVALUE;
```

Wird eine Sequenz letztendlich nicht mehr benötigt, so reicht eine einfache `DROP SEQUENCE MeineSequenz`-Anweisung, um sie aus der Datenbank zu entfernen.