SQL Server & T-SQL

Uwe Ziegenhagen

29. Mai 2010

Inhaltsverzeichnis

Tabellenverzeichnis

1 T-SQL 'Good Practices'

Kansy, 2007 listet eine Reihe von 'good practises' für T-SQL, hier ein verkürzter Überblick:

Gezielter Umbruch Durch gezielten Umbruch kann T-SQL deutlich lesbarer gestaltet werden.

Schlüsselwörter groß schreiben Alle T-SQL-Schlüsselwörter groß schreiben.

Keine Schlüsselwörter, Leerzeichen oder Sonderzeichen verwenden Der SQL Server lässt es zwar oft zu, aber Schlüsselwörter, Leerzeichen oder Umlaute sollten nicht als Spaltennamen verwendet werden.

Kein SELECT * Die Verwendung von SELECT * sollte aus Effizienzgründen möglich vermieden werden.

IdentitiyCol Wenn auf die Identitätsspalte einer Tabelle zugegriffen wird, sollte nicht deren Name angegeben werden, sondern der Alias `IdentityCol`.

Ordnung muss sein Wird bei der Weiterverarbeitung eine bestimmte Reihenfolge benötigt, sollte auf jeden Fall ein `ORDER BY` angegeben werden.

Alias Insbesondere bei der Arbeit mit mehreren Tabellen sollte ein `Alias` angegeben werden, da dies Mehrdeutigkeiten verringert und die Lesbarkeit erhöht.

Groß- und Kleinschreibung bei Filter mittels = und LIKE Standardmäßig führt der SQL Server = und `LIKE` Vergleiche ohne Berücksichtigung von Groß- und Kleinschreibung durch. Will man explizit ein Suchverhalten festlegen, kann man die Einstellung gemäß Listing 1 definieren.

```
1 // Berücksichtige Groß- und Kleinschreibung
2 SELECT * FROM Personen WHERE Nachname =
3 'Schmidt' COLLATE SQL_Latin1_General_CP1_CS_AS
4
5 // Berücksichtige Groß- und Kleinschreibung nicht
6 SELECT * FROM Personen WHERE Nachname =
7 'Schmidt' COLLATE Latin1_General_CI_AS
```

Listing 1: Groß- und Kleinschreibung bei Suchanfragen

Benötigte Größe und Genauigkeit festlegen Dezimalstellen und Genauigkeit von Dezimaldatentypen sollte man immer festlegen, da sonst Standardeinstellungen mit nicht genau festgelegten Werten für Genauigkeit und Ganzzahlen genutzt werden. Bei Variablenzuweisungen und Wertvergleichen kann mit `CONVERT` sichergestellt werden, dass eine einheitliche Genauigkeit genutzt wird.

Wird kein Unicode benötigt, sollten Unicode-Spaltentypen wie `NCHAR` oder `NVARCHAR` nicht genutzt werden, da diese den im Vergleich zu `CHAR` oder `VARCHAR` doppelten Speicherplatz brauchen.

Systemkomponenten Zugriffe auf Systemkomponenten sollten unterbleiben, da es oftmals auch 'legale' Wege des Zugriffs gibt. Beispiel:

```
1 // SCHLECHT
2 SELECT [name] FROM sysobjects WHERE xtype='U'
3
4 // GUT
5 SELECT table_name from INFORMATION_SCHEMA.TABLES
```

Listing 2: Zugriffe auf Metainformationen der Datenbank

Siehe dazu auch Microsoft, 2007 für einen Überblick, wie man an Metainformationen der Datenbank kommt.

Vergleiche mit NULL Vergleiche und Rechenoperationen von einem Wert und `NULL` sollten unterbleiben. Stattdessen muss `NULL`, `IS NULL` oder `ISNULL()` verwandt werden, um keine seltsamen Resultate zu erhalten.

Fehler! In `@@ERROR` wird die letzte Fehlermeldung gespeichert. Dieser Wert wird aber mit jedem neuen fehlerlosen SQL Statement überschrieben, wenn sie nicht direkt nach der den Fehler erzeugenden SQL Anweisung zwischengespeichert wird. Also: lokale Variable deklarieren, nach einer fehlerträchtigen Anweisung zwischenspeichern, danach auswerten.

SET NOCOUNT ON/OFF Die Anzahl der betroffenen Zeilen ist oftmals unwichtig, daher sollte bei Stored Procedures am Anfang ein `SET NOCOUNT ON` stehen, am Ende dann ein `SET NOCOUNT OFF`.

Gespeicherte Prozeduren Gespeicherte Prozeduren sollten nicht mit `SP_` beginnen, da sonst zuerst in der Master-Datenbank und in den Systemtabellen der aktuellen Datenbank gesucht wird. Einzelne Rückgabewerte sollten nicht per `SELECT` sondern per `OUTPUT` oder `RETURN` zurückgegeben werden.

Temporäre Tabellen Wenn temporäre Tabellen nicht vermieden werden können (z.B. durch TABLE Variablen), gilt: Erstellung am Anfang, drop am Ende.

Kommentare Kommentare, Kommentare, Kommentare!

Vorlagen Vorlagen sind hilfreich, um ein einheitliches Erscheinungsbild zu gewährleisten. Im Vorlagen-Explorer (Strg+Alt+T) kann man Vorlagen verwalten und neu erstellen.

2 T-SQL Schlüsselwörter

Die folgenden Wörter sind T-SQL Schlüsselwörter und sollten nicht als Variablennamen genutzt werden.

Tabelle 1: T-SQL Schlüsselwörter Teil 1

ADD	EXISTS	PRECISION
ALL	EXIT	PRIMARY
ALTER	EXTERNAL	PRINT
AND	FETCH	PROC
ANY	FILE	PROCEDURE
AS	FILLFACTOR	PUBLIC
ASC	FOR	RAISERROR
AUTHORIZATION	FOREIGN	READ
BACKUP	FREETEXT	READTEXT
BEGIN	FREETEXTTABLE	RECONFIGURE
BETWEEN	FROM	REFERENCES
BREAK	FULL	REPLICATION
BROWSE	FUNCTION	RESTORE
BULK	GOTO	RESTRICT
BY	GRANT	RETURN
CASCADE	GROUP	REVERT
CASE	HAVING	REVOKE
CHECK	HOLDLOCK	RECHTS
CHECKPOINT	IDENTITY	ROLLBACK
CLOSE	IDENTITY_INSERT	ROWCOUNT
CLUSTERED	IDENTITYCOL	ROWGUIDCOL
COALESCE	IF	RULE
COLLATE	IN	SAVE
COLUMN	INDEX	SCHEMA
COMMIT	INNER	SECURITYAUDIT

Tabelle 2: T-SQL Schlüsselwörter Teil 2

COMPUTE	INSERT	SELECT
CONSTRAINT	INTERSECT	SESSION_USER
CONTAINS	INTO	SET
CONTAINSTABLE	IS	SETUSER
CONTINUE	JOIN	SHUTDOWN
CONVERT	KEY	SOME
CREATE	KILL	STATISTICS
CROSS	LEFT	SYSTEM_USER
CURRENT	LIKE	TABLE
CURRENT_DATE	LINENO	TABLESAMPLE
CURRENT_TIME	LOAD	TEXTSIZE
CURRENT_TIMESTAMP	MERGE	THEN
CURRENT_USER	NATIONAL	TO
CURSOR	NOCHECK	TOP
DATABASE	NONCLUSTERED	TRAN
DBCC	NOT	TRANSACTION
DEALLOCATE	NULL	TRIGGER
DECLARE	NULLIF	TRUNCATE
DEFAULT	OF	TSEQUAL
DELETE	OFF	UNION
DENY	OFFSETS	UNIQUE
DESC	ON	UNPIVOT
DISK	OPEN	UPDATE
DISTINCT	OPENDATASOURCE	UPDATETEXT
DISTRIBUTED	OPENQUERY	USE
DOUBLE	OPENROWSET	USER

Tabelle 3: T-SQL Schlüsselwörter Teil 3

DROP	OPENXML	VALUES
DUMP	OPTION	VARYING
ELSE	OR	VIEW
END	ORDER	WAITFOR
ERRLVL	OUTER	WHEN
ESCAPE	OVER	WHERE
EXCEPT	PERCENT	WHILE
EXEC	PIVOT	WITH
EXECUTE	PLAN	WRITETEXT

3 Datenbankwartung

Eine SQL-Server Datenbank besteht aus mindestens zwei Dateien, der MDF-Datei für die Datenbankinhalte und der LDF-Datei, die zur Protokollierung dient.

```
1 CREATE DATABASE name
```

Listing 3: einfachste `CREATE DATABASE` Abfrage

```
CREATE DATABASE BookStoreArchive
ON PRIMARY
(
 NAME = 'Buchladen',
 FILENAME = 'F:\MSSQL\DATA\Buchladen.mdf',
 SIZE = 5MB ,
 MAXSIZE = UNLIMITED,
 FILEGROWTH = 10MB),
(
 NAME = 'Buchladen2',
 FILENAME = 'G:\MSSQL\DATA\Buchladen2.ndf',
 SIZE = 5MB ,
 MAXSIZE = 50MB,
 FILEGROWTH = 5%)
 LOG ON (
   NAME = 'BookStoreArchive_log',
   FILENAME = 'H:\MSSQL\TLOG\Buchladen_log.LDF',
   SIZE = 500KB ,
   MAXSIZE = 100MB ,
   FILEGROWTH = 5%)
GO
```

Listing 4: `CREATE DATABASE` mit kompletter Angabe 

Listing 4 erstellt die Datenbank `Buchladen` in der primären Dateigruppe. Die Datenbank selbst wird in zwei Dateien gespeichert, `Buchladen2.ndf` dient nur der sekundären Speicherung. Benutzerdefinierte Dateigruppen nutzt man in 'very large databases' (VLDB), für normale Anforderungen ist `PRIMARY`.

```
EXEC sp_helpdb 'dbname'
GO
```

Listing 5: Informationen zur Datenbank 'dbname' abfragen

```
EXEC sp_helptable 'tname'
GO
```

Listing 6: Informationen zur Tabelle 'tname' abfragen

```
EXEC sp_columns 'tname'
GO
```

Listing 7: Informationen zu den Spalten der Tabelle 'tname' abfragen

```
sp_executesql
```

Listing 8: `sp_executesql` Statement

```
name  db_size   owner  dbid  created status        compatibility_level
------ ------------ ---------- ------ ----------- ---------------------------------
      ---------------------
test  4.00 MB CORE\Uwe 5   Okt 31 2008 Status=ONLINE,    90
              Updateability=READ_WRITE,
              UserAccess=MULTI_USER,
              Recovery=SIMPLE, Version=611,
              Collation=Latin1_General_CI_AS,
              SQLSortOrder=0,
              IsAutoCreateStatistics,
              IsAutoUpdateStatistics,
              IsFullTextEnabled

name   fileid filename   filegroup size  maxsize growth  usage
---------- ------ ------------------------- ---------- --------- -------------- -------- ---------
test   1   G:\_SQLserver\test.mdf PRIMARY 3072 KB Unlimited  1024 KB data only
test_log  2   G:\_SQLserver\test_log.ldf NULL  1024 KB  2147483648 KB 10%  log only
```

Listing 9: Ergebnis von Listing 5

```
EXEC sp_dboption 'test'
GO
```

Listing 10: Informationen zur Datenbank 'name' abfragen

```
The following options are set:
-----------------------------------
trunc. log on chkpt.
auto create statistics
auto update statistics
```

Listing 11: Ergebnis von Listing 10

```
ALTER DATABASE name
```

Listing 12: `ALTER DATABASE` Abfrage

```
DELETE DATABASE name
```

Listing 13: `DELETE DATABASE` Abfrage

Datenbanken sollten nur über T-SQL oder den Manager gelöscht werden, nicht über das Dateisystem.

4 Tabellen anlegen

```
IF EXISTS(SELECT name FROM sys.tables
  WHERE name = 'T')
  BEGIN
  PRINT 'T already.'
  DROP TABLE T_old
  EXEC sp_rename 'T', 'T_old'
  END
ELSE PRINT 'No T already.'
  CREATE TABLE T (
  c1 bigint,
  c2 nvarchar(max)
)
DROP TABLE T
GO
```

Listing 14: CREATE TABLE mit EXISTS Abfrage 

5 Constraints

5.1 PRIMARY KEY

```
ALTER TABLE Kunden
ADD CONSTRAINT PK_KUNDEN
PRIMARY KEY (KundenID)
GO
```

Listing 15: ALTER TABLE Abfrage um einen PRIMARY KEY anzulegen

5.2 FOREIGN KEY

```
ALTER TABLE Bestellung
ADD CONSTRAINT FK_BESTELLTBEI
FOREIGN KEY (BestellerID)
REFERENCES Kunden(KundenID)
GO
```

Listing 16: ALTER TABLE Abfrage, um einen FOREIGN KEY anzulegen

5.3 IDENTITY

5.4 UNIQUE

5.5 CHECK

5.6 NOT NULL

5.7 DEFAULT

6 INSERT und DELETE

7 Einfache Abfragen

```
1 SELECT * FROM kunden;
```

Listing 17: einfache SELECT Abfrage

8 Abfragen aus mehreren Tabellen

8.1 INNER JOINS

8.1.1 Implizite Schreibweise

```
1 SELECT RechnungsNr, KundenNr, Betrag, Rechnungen_Oktober.
     kartennummer, Firma, Inhaber, Ablaufdatum
2 FROM Kreditkarte, Rechnungen_Oktober
3 WHERE Kreditkarte.Kartennummer = Rechnungen_Oktober.
     Kartennummer
```

Listing 18: Schreibweise eines Implicit Join 

8.1.2 Explizite Schreibweise

```
1 SELECT
2  RechnungsNr, KundenNr, Betrag,
3  Rechnungen_Oktober.Kartennummer, Firma,
4  Inhaber, Ablaufdatum
5 FROM Kreditkarte
6 INNER JOIN Rechnungen_Oktober
7 ON Kreditkarte.Kartennummer = Rechnungen_Oktober.
     Kartennummer
```

Listing 19: Explizite Schreibweise eines INNER JOIN 

Der INNER JOIN führt Datensätze aus der linken und rechten Tabelle genau dann zusammen, wenn die angegebenen Kriterien alle erfüllt sind. Ist mindestens eins der Kriterien nicht erfüllt, so entsteht kein Datensatz in der Ergebnismenge. Durch den Einsatz dieses `JOIN` reduziert sich das Ergebnis des kartesischen Produktes auf ein Minimum (vergleiche auch nachfolgende Join-Varianten).

8.2 LEFT JOINS

Die Logik lautet für den LEFT JOIN: Ein Datensatz aus der linken Tabelle kommt in jedem Fall in das Ergebnis. Wenn ein Datensatz der rechten Tabelle dem ON-Kriterium entspricht, so wird er entsprechend in den Spalten eingetragen, ansonsten bleiben die Spalten leer (null). Der RIGHT JOIN arbeitet genau entgegengesetzt.
Gesucht werden alle Rechnungen vom Oktober. Falls sie per Kreditkarte bezahlt wurden, so sollen die Kartendaten ebenfalls ausgegeben werden.

```
1 SELECT
2  RechnungsNr, KundenNr, Betrag,
3  Rechnungen_Oktober.Kartennummer, Firma,
4  Inhaber, Ablaufdatum
5 FROM Rechnungen_Oktober
6 LEFT JOIN Kreditkarte
7 ON Kreditkarte.Kartennummer = Rechnungen_Oktober.
     Kartennummer
```

Listing 20: Beispiel für einen `LEFT JOIN`

8.3 RIGHT JOIN

Gesucht werden alle Karteninformationen. Falls mit der entsprechenden Kreditkarte im Oktober etwas bestellt wurde, sollen die Rechnungsinformationen beigefügt werden.

```
1 SELECT
2  RechnungsNr, KundenNr, Betrag,
3  Kreditkarte.Kartennummer, Firma,
4  Inhaber, Ablaufdatum
5 FROM rechnungen_oktober
6 RIGHT JOIN Kreditkarte
7 ON Kreditkarte.Kartennummer = Rechnungen_Oktober.
     Kartennummer
```

Listing 21: Beispiel für einen `RIGHT JOIN`

8.4 FULL OUTER JOIN

Der `FULL OUTER JOIN` kommt dem ursprünglichen Kreuzprodukt von allen Joins am nächsten. Er ist gewissermaßen die Kombination aus `LEFT` und `RIGHT JOIN`.

```
1 SELECT
2  RechnungsNr, KundenNr, Betrag,
3  Rechnungen_Oktober.Kartennummer,
4  Firma, Inhaber, Ablaufdatum
5 FROM Rechnungen_Oktober
6 OUTER JOIN Kreditkarte
7 ON Kreditkarte.Kartennummer = Rechnungen_Oktober.
     Kartennummer
```

Listing 22: Beispiel für einen `OUTER JOIN`

Gesucht werden sowohl alle Karteninformationen als auch alle Rechnungen. Sofern möglich sollen dabei Rechnungen und Karten kombiniert werden.

9 SQL Funktionen

9.1 Aggregatfunktionen

9.1.1 AVG()

Berechnet das arithmetische Mittel einer Spalte.

9.1.2 MIN()

Gibt das Minimum einer Spalte aus.

9.1.3 CHECKSUM_AGG()

Kann benutzt werden, um Änderungen an einer Tabelle festzustellen. Siehe dazu auch http://www.mssqltips.com/tip.asp?tip=1023.

9.1.4 SUM()

Summiert eine Spalte. Für das Produkt einer Spalte siehe Abschnitt 17.5.

9.1.5 COUNT()

Zählt die Einträge einer Spalte, die nicht NULL sind.

9.1.6 STDEV()

Errechnet die Standardabweichung einer Spalte für eine Stichprobe.

9.1.7 COUNT_BIG()

Wie `COUNT`, gibt aber eine Zahl vom Typ `BIG` zurück.

9.1.8 STDEVP()

Errechnet die Standardabweichung einer Population.

9.1.9 GROUPING()

Eine Aggregatfunktion die zusammen mit `CUBE` und `ROLLUP` Operatoren genutzt wird.

9.1.10 VAR()

Berechnet die Varianz einer Stichprobe.

9.1.11 MAX()

Gibt das Maximum einer Spalte zurück.

9.1.12 VARP()

Errechnet die Varianz einer Grundgesamtheit.

9.2 Datumsfunktionen

9.2.1 DATEADD(datepart, number, date)

Gibt ein neues Datum zurück, das auf dem Hinzufügen eines Zeitintervalls zum angegebenen Datum basiert. `datepart` ist einer der Parameter aus Tabelle 4, `number` eine ganze Zahl, `date` das Basisdatum.

Listing 23: Beispiel für `DATEADD`

9.2.2 DATENAME()

9.2.3 DATEDIFF()

```
1 DATEDIFF(datumsteil,anfang,ende)
```

Listing 24: Beispiel für `DATEDIFF`

`datumsteil` kann folgende Werte annehmen:

Tabelle 4: Mögliche Werte für `DATEDIFF()`

Einheit	SQL-Parameter	Abkürzung
Jahr	year	yy, yyyy
Quartal	quarter	qq, q
Monat	month	mm, m
Tag des Jahres	dayofyear	dy, y
Tag	day	dd, d
Woche	week	wk, ww
Stunde	hour	hh
Minute	minute	mi, n
Sekunde	second	ss, s
Millisekunden	millisecond	ms

```
1 Select DATEDIFF(yy,Geburtstag,GETDATE()) from personen
```

Listing 25: Beispiel `DATEDIFF`

9.2.4 GETDATE()

gibt das aktuelle Datum im `datetime` Format zurück.

9.3 RANK Funktionen

9.3.1 RANK()

9.3.2 DENSE_RANK()

9.3.3 NTILE()

9.3.4 ROW_NUMBER()

9.4 Mathefunktionen

9.4.1 ABS(n)

Gibt den absoluten Wert (ohne Vorzeichen) des Ausdrucks n zurück.

9.4.2 ACOS(n)

Berechnet den Arcus Cosinus von n.

9.4.3 ASIN(n)

Berechnet den Arcus Sinus von n.

9.4.4 ATAN(n)

Berechnet den Arcus Tangens von n.

9.4.5 ATN2(n,m)

Berechnet den Arcus Tangens von n/m.

9.4.6 CEILING(n)

Berechnet den kleinsten ganzzahligen Wert, der größer oder gleich n ist.

9.4.7 COS(n)

Berechnet den Kosinus von n.

9.4.8 COT(n)

COT(n)
Berechnet den Kotangens von n.

9.4.9 DEGREES(n)

Konvertiert Radian in Grad.

9.4.10 EXP(n)

Berechnet den Wert e^n.

9.4.11 FLOOR(n)

Berechnet den größten ganzzahligen Wert, der kleiner gleich der Zahl n ist

9.4.12 LOG(n)

Berechnet den natürlichen Logarithmus der Zahl n.

9.4.13 LOG10(n)

Berechnet den dekadischen Logarithmus von n.

9.4.14 PI()

Gibt den Wert von Pi zurück.

9.4.15 POWER(x,y)

Berechnet x^y.

9.4.16 RADIANS(n)

Konvertiert Grad nach Radian.

9.4.17 RAND

Gibt eine zufällige Zahl aus [0,1] zurück.

9.4.18 ROUND(n, p,[t])

Rundet den Wert der Zahl n mit der Präzision p. Positive Werte von p runden rechts vom Dezimalpunkt, negative Werte links vom Dezimalpunkt. Der Parameter t ist optional und bewirkt ein Abschneiden der Zahl nach t Stellen.

9.4.19 ROWCOUNT_BIG

Gibt die Anzahl der Zeilen zurück, die vom letzten T-SQL Kommando betroffen waren.

9.4.20 SIGN(n)

Gibt das Vorzeichen von n zurück: $+1$ für positive Werte, -1 für negative Werte und 0 für 0.

9.4.21 SIN(n)

Berechnet den Sinus von n.

9.4.22 SQRT(n)

Berechnet die Quadratwurzel von n.

9.4.23 SQUARE(n)

Berechnet das Quadrat von n.

9.4.24 TAN(n)

Berechnet den Tangens von n.

9.5 Metadatenfunktionen

9.5.1 DB_NAME()

9.5.2 DB_ID()

(Seite 155 im AW Buch)

9.6 Sicherheitsfunktionen

9.6.1 USER_NAME()

9.6.2 SUSER_NAME()

9.6.3 IS_MEMBER()

9.7 String-Funktionen

ab Seite 156

9.7.1 ASCII(<char>)

Gibt den ASCII-Zahlenwert für das Zeichen <char> zurück

9.7.2 CHAR(<Zahl>)

Gibt das ASCII-Zeichen für <Zahl> aus.

9.7.3 LEFT(<String>,<Zahl>)

Gibt die linken <Zahl> Zeichen von <String> zurück.

9.7.4 RIGHT(<String>,<Zahl>)

rechts
Gibt die rechten <Zahl> Zeichen von <String> zurück.

9.7.5 CHARINDEX(<String1>,<String2>)

Gibt die Position von <String1> in <String2> zurück. Wird <String1> nicht gefunden, wird 0 ausgegeben.

9.7.6 LEN(<String>)

Gibt die Länge von String <String> zurück.

9.7.7 LOWER(<String>)

Wandelt den übergebenen String in Kleinbuchstaben um.

9.7.8 LTRIM(<String>)

Entfernt im String <String> eventuell vorhandene Leerzeichen links.

9.7.9 REPLICATE(<String>,<Zahl>)

Wiederholt den Ausdruck <String> <Zahl>-mal.

9.7.10 RTRIM(<String>)

Entfernt im String <String> eventuell rechts vorhandene Leerzeichen.

9.7.11 SOUNDEX(<String>)

Gibt einen phonetischen Wert für den Klang bzw. die Aussprache eines Ausdrucks, der nützlich sein kann, um ähnliche klingende Wörter zu finden. Alle vier Aufrufe in Listing 26 geben M600 zurück.

```
1 SELECT SOUNDEX('meyer');
2 SELECT SOUNDEX('meier');
3 SELECT SOUNDEX('maier');
4 SELECT SOUNDEX('mayer');
```

Listing 26: SOUNDEX Beispiel

9.7.12 SPACE(im String <Zahl>)

Gibt im String <Zahl> Leerzeichen zurück.

9.7.13 STR(<String>)

wandelt <String> in eine Zahl um. Bei nicht konvertierbaren Strings wird eine Fehlermeldung zurückgegeben.

9.7.14 SUBSTRING(<String>,<Zahl1>,<Zahl2>)

Gibt aus <String> den Teilstring <Zahl1> bis <Zahl2> zurück.

9.7.15 UPPER(<String>)

Text in Großbuchstaben
Wandelt den übergebenen String in Großbuchstaben um.

9.8 Systemfunktionen

9.8.1 CONVERT()

Konvertiert zwischen verschiedenen Typen:

```
SELECT
 CONVERT(varchar, GETDATE(), 100) AS 'Format 100',
 CONVERT(varchar, GETDATE(), 101) AS 'Format 101',
 CONVERT(varchar, GETDATE(), 102) AS 'Format 102',
 CONVERT(varchar, GETDATE(), 103) AS 'Format 103',
 CONVERT(varchar, GETDATE(), 104) AS 'Format 104',
 CONVERT(varchar, GETDATE(), 105) AS 'Format 105',
 CONVERT(varchar, GETDATE(), 106) AS 'Format 106',
 CONVERT(varchar, GETDATE(), 107) AS 'Format 107',
 CONVERT(varchar, GETDATE(), 108) AS 'Format 108',
 CONVERT(varchar, GETDATE(), 109) AS 'Format 109',
 CONVERT(varchar, GETDATE(), 110) AS 'Format 110',
 CONVERT(varchar, GETDATE(), 111) AS 'Format 111',
 CONVERT(varchar, GETDATE(), 112) AS 'Format 112',
 CONVERT(varchar, GETDATE(), 113) AS 'Format 113',
 CONVERT(varchar, GETDATE(), 114) AS 'Format 114',
 CONVERT(varchar, GETDATE(), 120) AS 'Format 120',
 CONVERT(varchar, GETDATE(), 121) AS 'Format 121',
 CONVERT(varchar, GETDATE(), 126) AS 'Format 126',
 CONVERT(varchar, GETDATE(), 130) AS 'Format 130',
 CONVERT(varchar, GETDATE(), 131) AS 'Format 131'
```

Listing 27: CONVERT Beispiel

9.8.2 CAST()

10 Views

11 Temporäre Tabellen und TABLE Variablen

Temporäre Tabellen werden mit vorangestelltem # angelegt und sind nur in der aktuellen Session des SQL Servers sichtbar. Sobald die Session beendet wird oder ein explizites Drop ausgeführt wird, werden die temporären Tabellen gelöscht. Es gibt auch global verfügbare temporäre Tabellen, diese werden mit führendem ## angelegt.
Der wesentliche Unterschied zu normalen Tabellen ist, dass keine FOREIGN KEY Constraints auf einer temporären Tabelle angelegt werden können.
Wenn unterschiedliche Nutzer die gleiche temporäre Tabelle anlegen, erhält jeder Nutzer seine eigene Tabelle. Temporäre Tabellen, die innerhalb einer Stored Procedure angelegt werden, werden am Ende der Prozedurausführung automatisch gelöscht.

Wenn die Stored Procedure A eine temporäre Tabelle anlegt und die SP B aufruft, kann B die temporäre Tabelle benutzen. Als 'good practice' gilt, erstellte temporäre Tabellen explizit zu löschen.

```
1 CREATE TABLE #vornamen (
2 ID int,
3 vorname char(30))
4
5 select name
6 from tempdb..sysobjects
7 where name like '#vornamen%'
8
9 drop table #vornamen
```

Listing 28: Nutzung einer temporären Tabelle

Ab SQL Server 2000 gibt es den Variablentyp `TABLE`. Variablen dieses Typs sind temporären Tabellen ähnlich, sind aber flexibler und und werden ausschließlich im RAM gespeichert.

Für die Wahl, ob temporäre Tabelle oder `TABLE` Variable gilt nach Graziano, 2001 und Allen, 2005 folgendes:

- Bei weniger als 100 Zeilen ist eine TABLE Variable normalerweise effizienter, da der SQL Server für TABLE Variablen keine Statistik anlegt.
- TABLE Variablen
- Sobald ein Index benötigt wird, muss eine temporäre Tabelle genutzt werden.
- Für temporäre Tabellen sind Indizes empfehlenswert, da sie die Anzahl der notwendigen Rekompilierungen verringern. TABLE Variablen innerhalb von Stored Procedures können weniger notwendige Rekompilierungen benötigen als temporäre Tabellen.

```
1 DECLARE @vornamen TABLE (
2 ID int,
3 vorname char(30))
4
5 INSERT INTO @vornamen (ID, vorname)
6 SELECT  ID, vorname
7 FROM  dbo.namen
8 WHERE  nachname = 'Schmidt'
```

Listing 29: Nutzung einer `TABLE` Variablen

```
DECLARE @MyTable TABLE(
  ProductID int,
  Price money CHECK(Price < 10.0))

INSERT INTO @MyTable VALUES(1,1);
INSERT INTO @MyTable VALUES(2,2);
INSERT INTO @MyTable VALUES(3,3);
INSERT INTO @MyTable VALUES(4,5);
SELECT * FROM @MyTable;
```

Listing 30: Nutzung einer TABLE Variablen

12 Cursors

```
DECLARE @temp char(10)

DECLARE testcursor CURSOR FOR
  Select name from tabelle where id = '12345' and BDATE>'
     30.09.2000'

OPEN testcursor

FETCH next FROM test INTO @temp
  WHILE @@Fetch_Status = 0 BEGIN
  print 'Hello'
  FETCH next FROM test INTO @temp
END

CLOSE testcursor
DEALLOCATE testcursor
```

Listing 31: Einfaches Beispiel für einen CURSOR

13 Transaktionen

Allen, 2005

```
DECLARE @ProductTotals TABLE(
  ProductID int ,
  Revenue money
)

BEGIN TRANSACTION
  INSERT INTO @ProductTotals (ProductID, Revenue)
  SELECT ProductID, SUM(UnitPrice * Quantity)
  FROM [Order Details]
  GROUP BY ProductID
ROLLBACK TRANSACTION

SELECT COUNT(*) FROM @ProductTotals
```

Listing 32: Beispiel für eine Transaktion

14 Stored Procedures

Die Wikipedia Wikipedia, 2009 schreibt zu Stored Procedures:

> Der Begriff Gespeicherte Prozedur (GP) oder englisch Stored Procedure (SP) bezeichnet eine Funktion bestimmter Datenbankmanagementsysteme. In einer Stored Procedure können ganze Abläufe von Anweisungen unter einem Namen gespeichert werden, die dann auf dem Datenbankserver zur Verfügung stehen und ausgeführt werden können. Eine SP ist somit ein eigenständiger Befehl, der eine Abfolge von gespeicherten Befehlen ausführt.
>
> Mittels Stored Procedures können häufiger verwendete Abläufe, die sonst durch viele einzelne Befehle vom Client ausgeführt werden würden, auf den Server verlagert werden, und durch einen einzigen Aufruf ausgeführt werden (siehe auch Client-Server-System). Mitunter wird dadurch die Leistung gesteigert, da weniger Daten zwischen Client und Datenbankserver ausgetauscht werden müssen und das Datenbankmanagementsystem häufig auf leistungsfähigeren Servern läuft.

Zum Vergleich von Stored Procedures mit Funktionen siehe Modi, 2007.

15 Variablen

SQL Server unterscheidet benutzerdefinierte und globale Variablen. Globale Variablen beginnen mit @@ und können nur ausgelesen, nicht jedoch verändert werden.

Tabelle 5: Globale Variablen

Variable	Erläuterung
@@LANGUAGE	Sprachversion des Servers
@@NESTLEVEL	Schachtelungstiefe (maximal 32)
@@SERVERNAME	Namen des Servers
@@VERSION	Programmversion des Servers
@@ERROR	Wert der letzten Fehlermeldung
@@FETCHSTATUS	bei Cursors genutzt. solange Wert = 0 kann noch ein weiterer Wert abgefragt werden.

15.1 @@ERROR

Wenn der SQL Server ein statement erfolgreich ausführt, wird die globale Variable `@@ERROR` auf 0 gesetzt.
Da @@ERROR nach jeder SQL Anweisung geleert und neu gesetzt wird, sollte sie unmittelbar nach fehlerträchtigen Anweisungen ausgewertet werden bzw. in einer lokalen Variablen gespeichert werden.
Listing 33 zeigt ein Beispiel für ein `UPDATE` statement, das versucht einen negativen Wert in eine Spalte einzutragen.

```
1 CREATE TABLE #temp
2 CONSTRAINT
3
4 INSERT INTO
```

Listing 33: Beispiel für `@@ERROR`

15.2 Deklaration von Variablen

```
1 DECLARE @var AS int
2
3 SET @var = 1;
4 SELECT @var = (SELECT COUNT(*) FROM mitarbeiter);
5
6 SELECT @var;
```

Listing 34: Beispiel für benutzerdefinierte Variablen

Bei der Zuweisung mehrerer Werte an Variablen ist es günstiger, SELECT zu nutzen:

```
1 SELECT
2  @nachname = nachname,
3  @vorname = vorname
4 FROM personen
5 WHERE persID = 123;
```

Listing 35: Beispiel für benutzerdefinierte Variablen

```
1  DECLARE @abc int
2  SET @abc = 1;
3
4  IF @abc=1
5    PRINT 'Hallo'
6  ELSE
7    PRINT 'WELT'
8
9  SET @abc = 0;
10 IF @abc=1
11   PRINT 'Hallo'
12 ELSE
13   PRINT 'WELT'
```

Listing 36: Beispiel für benutzerdefinierte Variablen

15.3 Variablentypen

- Character
  - `CHAR(n)`

    Minimum: 1

    Maximum: 8000

    ben. Platz: 1 Byte pro Zeichen
  - `VARCHAR(n)`

    Minimum: 1

    Maximum: 8000

    ben. Platz: 1 Byte pro Zeichen plus 2 Byte
  - `NCHAR(n)`

    Minimum: 1

    Maximum: 4000

    ben. Platz: 2 Byte für jedes Zeichen.
  - `VARCHAR(MAX)`

    Minimum: 1

    Maximum: 2,147,483,647

    ben. Platz:

    Ohne explizite Angabe der Länge werden 30 Zeichen genommen.
  - `NVARCHAR(n)`

    Minimum:

    Maximum:

    ben. Platz:

  - `NVARCHAR(MAX)`

    Minimum: 1

    Maximum: 2,147,483,647

    ben. Platz: 2 Byte pro Zeichen plus 2 Byte

- Datum/Uhrzeit
  - `DATETIME`

    Minimum: 1. Januar 1753

    Maximum: 31. Dezember 9999

    ben. Platz: Genauigkeit: 3,33 Millisekunden, 8 Byte (zwei 4-Byte integer Werte). Die ersten 4 Byte repräsentieren die Anzahl der Tage vor oder nach dem 1. Januar 1900. Die zweiten 4 Byte speichern die Tageszeit in Schritten von $1/3000$ Sekunden nach 0:00:00 Uhr.

  - `SMALLDATETIME`

    Minimum: 1. Januar 1900

    Maximum: 6. Juni 2079

    ben. Platz: Genauigkeit: 1 Minute, 4 Byte (ein integer). Die ersten 2 Byte enthalten die Anzahl der Tage nach dem 1. Januar 1900, die zweiten 2 Byte speichern die Tageszeit in Minuten nach 0:00:00 Uhr.

- Zahlen
  - `DECIMAL(Genauigkeit, Dezimalstellen)`

    Genauigkeit = totale Anzahl an Stellen, links und rechts vom Dezimalzeichen.
    Dezimalstellen = Anzahl der Stellen rechts vom Dezimalstellen.
    - Präzision 1–9: 5 Byte
    - Präzision 10–19: 9 Byte
    - Präzision 20–28: 13 Byte
    - Präzision 29–38: 17 Byte

    Die minimale Genauigkeit beträgt 1, die maximale Genauigkeit beträgt 38. Hinweis: Decimal entspricht Numeric.
  - `FLOAT(n)`

    Minimum:

    Maximum:

    ben. Platz:

  - `REAL`

    Minimum: -3.40E + 38 to -1.18E - 38, 0

    Maximum: 1.18E - 38 to 3.40E + 38

    ben. Platz: 4 Byte

    Hinweis: Real ist äquivalent zu FLOAT(24).
  - `BIGINT`

    Minimum: -9,223,372,036,854,775,808

    Maximum: 9,223,372,036,854,775,807

ben. Platz: 8 Byte

- `INT`

  Minimum: -2,147,483,648

  Maximum: 2,147,483,647

  ben. Platz: 4 Byte

- `SMALLINT`

  Minimum: -32,768

  Maximum: 32,767

  ben. Platz: 2 Byte

- `TINYINT`

  Minimum: 0

  Maximum: 255

  ben. Platz: 1 Byte

- Währung
  - `MONEY`

    Minimum: -922,337,203,685,477.5808

    Maximum: 922,337,203,685,477.5807

    ben. Platz: 8 Byte

  - `SMALLMONEY`

    Minimum: -214,748.3648

    Maximum: 214,748.3647

    ben. Platz: 4 Byte

- Boolean
  - `BIT`

    Minimum: 0

    Maximum: 1

    ben. Platz: Bis zu 8 Bit Spalten werden zusammen in einer 1 Byte Spalte gespeichert, 9–16 in einer 2 Byt Spalte, etc.

- Text und Bilder
  - `TEXT` Veraltet.
  - `NTEXT` Veraltet.
  - `IMAGE` Veraltet.
- Binär
  - `BINARY(n)`

    Minimum: 1

    Maximum: 8000

    ben. Platz: 1 Byte pro Byte

  - `VARBINARY(n)`

Minimum: 1
Maximum: 8000
ben. Platz: 1 Byte pro Byte plus 2 Byte.

– `VARBINARY(MAX)`
Minimum: 1
Maximum: 2,147,483,647
ben. Platz: 1 Byte pro Byte plus 2 Byte.

- XML
  - `XML`
- Variante
  - `SQL_VARIANT`

Vorteile von Stored Procedures:
- zentralisieren den T-SQL Code - reduzieren den Netzwerk-Traffic - fördern die Wiederverwendbarkeit von Code-Schnipseln - haben einen stabilisiereden Einfluss auf Antwortzeiten - sind förderlich für die Systemsicherheit, da der direkte Zugriff auf Tabellen ein Sicherheitsrisiko darstellen kann
grundlegende Erstellung
CREATE PROCEDURE name AS sql-statement GO
und aufgerufen durch
EXEC name
Hinweis zur Namensvergabe: Eigene Stored Procedures sollten nicht mit sp_beginnen, da sonst der SQL Server erst in den Systemtabellen nach der Prozedur sucht.
Parametrisierte Stored Procedures
Parametrisierte Stored Procedures werden wie folgt erstellt:
CREATE PROCEDURE name (@param1 typ [=default] @param2 typ [=default] @param3 typ [=default]) AS sql-statement GO
und aufgerufen durch
EXEC name param1, param2, param3
Ausgabe von Stored Procedures
Mittels OUTPUT Parameter kann eine SP Werte an ihren Aufrufer geben, eine ad hoc Anfrage oder eine andere Stored Procedure.
Beispiel?
Ändern von existierenden Stored Procedures
ALTER PROCEDURE name (@param1 typ [=default] @param2 typ [=default] @param3 typ [=default]) AS sql-statement GO
Löschen von Stored Procedures
DROP PROCEDURE name
Automatische Ausführung beim Serverstart
Über die SP sp_procoption in der MASTER Datenbank des SQL Server 2005 lassen sich Gespeicherte Prozeduren ablegen, die beim Serverstart automatisch ausgeführt werden sollen.
Aktivieren:
EXE sp_procoption @ProcName = 'name', @OptionName = 'startup', @OptionValue ='true'
Deaktivieren über:
EXE sp_procoption @ProcName = 'name', @OptionName = 'startup', @OptionValue ='off'

16 Trigger

Ein Trigger ist eine gespeicherte Prozedur, die bei einer bestimmten Art der Änderungen (z. B. INSERT, UPDATE, DELETE) von Daten aufgerufen wird, das diese Änderung erlaubt, verhindert und/oder weitere Tätigkeiten vornimmt.

```
1 CREATE TRIGGER personenTrigger
2 ON personen
3 FOR INSERT
4 AS
5   DECLARE @anzahl char(50);
6   SET @anzahl = (SELECT COUNT(*) from personen);
7   Print @anzahl + ' sind in der Datenbank.';
```

Listing 37: Trigger, der nach dem INSERT die Anzahl der Zeilen in der Tabelle ausgibt.

17 Tipps, Tricks und Schnipsel

17.1 IF und ELSE

```
1 DECLARE @a float
2 DECLARE @b float
3
4 set @a = 0.1
5 set @b = 0.05
6
7 IF (SELECT @a) < (SELECT @b) BEGIN SELECT @a END ELSE
    BEGIN SELECT @b END
```

Listing 38: Kleines IF-ELSE Beispiel

17.2 Auf Großschreibweise prüfen

Das Problem, alle Namen finden zu wollen die komplett in Großbuchstaben geschrieben waren, lässt sich oft dahingehend vereinfachen, dass man alle Namen sucht, bei denen der zweite Buchstabe groß geschrieben wurde.

```
1 SELECT [Name]
2 FROM [DATABASE].[namestable]
3 WHERE ASCII(SUBSTRING([NAME],2,1)) BETWEEN 65 AND 90
```

Listing 39: Nach Großbuchstaben suchen

17.3 Summe von NULL-Werten bilden

Mit NULL Werten läßt sich schlecht rechnen da, wenn ein Operand NULL ist, die Berechnung auch NULL ergibt. Listing 40 zeigt, wie COALESCE genutzt werden kann, um eine Summe von zwei Werten zu bilden, die NULL sein können. Sind beide Werte ungleich NULL, wird die Summe ausgegeben. Ist mindestens einer NULL, so wird versucht, nur @abc auszugeben. Ist @abc NULL, wird versucht @def auszugeben. Wenn @def ebenfalls NULL ist, wird 0 zurückgegeben.

```
1 DECLARE @abc int
2 DECLARE @def int
3 DECLARE @ghi int
4
5 SET @abc = 4;
6 SET @def = null;
7 SET @ghi = 2
8
9 SELECT COALESCE(@abc+@def,@abc,@def,@ghi,0)
```

Listing 40: COALESCE Beispiel

Ein äquivalente Umsetzung mittels CASE würde

```
1  DECLARE @abc int
2  DECLARE @def int
3  DECLARE @ghi int
4
5  SET @abc = 4;
6  SET @def = null;
7  SET @ghi = 2
8
9  SELECT CASE
10   WHEN (@abc IS NOT NULL) THEN @abc
11   WHEN (@def IS NOT NULL) THEN @def
12   WHEN (@ghi IS NOT NULL) THEN @ghi
13   ELSE 0
14 END
```

Listing 41: COALESCE Beispiel mittels CASE

17.4 Datum umwandeln

```
1 SELECT GETDATE() AS GETDATE,
2 CONVERT(varchar,GETDATE(),4)
3 AS '2-stellig',
4 CONVERT(varchar,GETDATE(),104) AS '4-stellig'
```

17.5 Produkt eines Resultsets

SQL kennt keine PRODUCT Funktion, über Logarithmen kann diese jedoch nachgebildet werden. Beispiel: Das Produkt der Zahlen 1 bis 10 kann dargestellt werden als:

$$10^{\left(\sum_{i=1}^{10}(\log(i))\right)}$$

Beispiel Das Produkt der Zahlen 1 bis 5 ist: $1 \times 2 \times 3 \times 4 \times 5 = 120$. Da wir das in SQL so direkt nicht ausrechnen können, bilden wir die Logarithmen:

```
log(1:5) #Logarithmus der Zahlen 1 bis 5
[1] 0.0000000 0.6931472 1.0986123 1.3862944 1.6094379

sum(log(1:5)) # Aufsummieren
[1] 4.787492

exp(sum(log(1:5))) # Summe in den Exponenten heben
[1] 120
```

Listing 42: Rechenbeispiel in R (http://r-project.org)

17.6 Ergebniszeilen beschränken

Das LIMIT(Zeilen,Anfang) von MySQL gibt es in T-SQL nicht, es läßt sich aber nachbauen.

```
SELECT TOP 10 *
FROM (SELECT TOP 14 * FROM tabelle ORDER BY spalte) AS
    result
ORDER BY spalte DESC
```

Listing 43: SELECT Abfrage um die Ergebniszeilen 5 bis 10 eines Resultsets zu erhalten

17.7 Die letzten n Zeilen ausgeben

Listing 43 läßt sich auch leicht abwandeln, um die letzten n Zeilen einer Tabelle auszugeben.

```
SELECT TOP 100 *
FROM (SELECT TOP (SELECT COUNT(*) FROM Tabelle) * FROM
    Tabelle ORDER BY 1)
AS result
ORDER BY 1 DESC
```

Listing 44: SELECT Abfrage um die letzten 100 Ergebniszeilen eines Resultsets zu erhalten

17.8 Ergebnisspalten zusammenfassen

```
SELECT RTRIM(name) + ' ' + RTRIM(name) AS name,
 DATEDIFF(yy,geburtstag,GETDATE()) AS alter_in_jahren
FROM personen
```

Listing 45: Zusammenfassen von Spalten 

17.9 Temporäre Tabellen auf Existenz prüfen 1

Folgendes Skript löscht die temporäre Tabelle `#vornamen` falls diese existiert und legt die Tabelle neu an. Damit verhindert man Fehlermeldungen von nicht ausführbaren `DROP TABLE`.

```
IF EXISTS (
  SELECT * FROM tempdb..sysobjects WHERE name LIKE '#
      vornamen%' AND type in ('U')
)
 drop table #vornamen;
ELSE
 BEGIN
  CREATE TABLE #vornamen (
   ID int,
   vorname char(30)
  );
END
```

Listing 46: Auf Existenz einer temporären Tabelle prüfen 

17.10 Temporäre Tabellen auf Existenz prüfen 2

Folgendes Skript schaut, ob eine `OBJECT_ID` für `tempdb..#temptabelle` existiert. Falls diese existiert, wird sie gelöscht.

```
IF OBJECT_ID (N'tempdb..#temptabelle') IS NOT NULL
DROP TABLE #temptabelle
```

Listing 47: Auf Existenz einer temporären Tabelle prüfen 

17.11 Datumsformate

Das folgende SQL Skript gibt eine Liste der vom SQL Server 2005 unterstützten Datumsformate.

```
1  SELECT
2   CONVERT(varchar, GETDATE(), 100) AS 'Format 100',
3   CONVERT(varchar, GETDATE(), 101) AS 'Format 101',
4   CONVERT(varchar, GETDATE(), 102) AS 'Format 102',
5   CONVERT(varchar, GETDATE(), 103) AS 'Format 103',
6   CONVERT(varchar, GETDATE(), 104) AS 'Format 104',
7   CONVERT(varchar, GETDATE(), 105) AS 'Format 105',
8   CONVERT(varchar, GETDATE(), 106) AS 'Format 106',
9   CONVERT(varchar, GETDATE(), 107) AS 'Format 107',
10  CONVERT(varchar, GETDATE(), 108) AS 'Format 108',
11  CONVERT(varchar, GETDATE(), 109) AS 'Format 109',
12  CONVERT(varchar, GETDATE(), 110) AS 'Format 110',
13  CONVERT(varchar, GETDATE(), 111) AS 'Format 111',
14  CONVERT(varchar, GETDATE(), 112) AS 'Format 112',
15  CONVERT(varchar, GETDATE(), 113) AS 'Format 113',
16  CONVERT(varchar, GETDATE(), 114) AS 'Format 114',
17  CONVERT(varchar, GETDATE(), 120) AS 'Format 120',
18  CONVERT(varchar, GETDATE(), 121) AS 'Format 121',
19  CONVERT(varchar, GETDATE(), 126) AS 'Format 126',
20  CONVERT(varchar, GETDATE(), 130) AS 'Format 130',
21  CONVERT(varchar, GETDATE(), 131) AS 'Format 131'
```

Listing 48: CONVERT Beispiel 

Tabelle 6: Datumsformate

Format	Ausgabe
100	Mär 11 2009 8:22PM
101	03/11/2009
102	2009.03.11
103	11/03/2009
104	11.03.2009
105	11-03-2009
106	11 Mär 2009
107	Mär 11, 2009
108	20:22:48
109	Mär 11 2009 8:22:48:670PM
110	03-11-2009
111	2009/03/11
112	20090311
113	11 Mär 2009 20:22:48:670
114	20:22:48:670
120	2009-03-11 20:22:48
121	2009-03-11 20:22:48.670
126	2009-03-11T20:22:48.670
130	15 ???? ????? 1430 8:22:48:67
131	15/03/1430 8:22:48:670PM

17.12 Behandlung von UNICODE

Der SQL Server besitzt drei Spaltentypen, um UNICODE-kodierte Strings zu verarbeiten:

- NCHAR
- NVARCHAR
- NTEXT

Wenn eine Spalte als UNICODE spezifiziert wurde, muss man dem SQL Server bei **jedem** Kommando sagen, dass nachfolgende Strings UNICODE-kodiert sind. Dies geschieht über `N`.
Beispiel: Die zu `OBJECT_ID` gehörige Spalte ist UNICODE-kodiert, daher müssen wir dem String ein `N` voranstellen.

```
1 IF OBJECT_ID (N'tempdb..#temptabelle') IS NOT NULL
2 DROP TABLE #temptabelle
```

Listing 49: UNICODE Behandlung

17.13 SQL Statements generieren

Zum Debuggen ist es manchmal hilfreich, eine Stored Procedure für jede Zeile eines Resultsets getrennt aufzurufen. Die notwendigen SQL statements lassen sich mit einem ordentlichen Editor (http://www.ultraedit.com) generieren, es geht aber auch in T-SQL.

In Listing 50 gehen wir von einem Feld `PersonenID` aus, das wir als Parameter zusammen mit den weiteren Parametern 1 und `Juni` an die Stored Procedure `BerechneGehalt` übergeben.

```
1 SELECT 'exec BerechneGehalt '''+CAST(PersonenID as
     VARCHAR(3))+''', 0, ''Juni'''
2 FROM [dbo].[GEHALT]
```

Listing 50: exec String generieren

17.14 Zeilen zu Spalte

```
1  SET NOCOUNT ON
2
3  CREATE TABLE #Beverages
4  (
5      Beverage VARCHAR(32)
6  )
7  GO
8
9  INSERT #Beverages SELECT 'Coffee'
10 INSERT #Beverages SELECT 'Whine'
11 INSERT #Beverages SELECT 'Beer'
12 INSERT #Beverages SELECT 'Tea'
13 GO
14
15 DECLARE @beverages VARCHAR(1024)
16
17 SELECT @beverages = COALESCE(@beverages + ',', '') +
       Beverage
18 FROM   #Beverages
19
20 SELECT #Beverages = @beverages
21 GO
22
23 DROP TABLE #Beverages
24 GO
```

Listing 51: Zeilen zu Spalte 

17.15 Loggen eines Update-Prozesses

Ausgehend von der Aufgabenstellung, nur Zeilen zu aktualisieren, die NULL sind und existierende Werte zu behalten, hier ein cleveres Beispiel aus der `microsoft.public.de.sqlserver` Newsgroup.

Es nutzt die Tatsache, dass OUTPUT Informationen zu INSERT-, UPDATE-, DELETE- oder MERGE-Anweisungen zurückgeben kann oder in einer LOG-Tabelle speichern kann. Wie funktioniert es:

1. Es wird eine Tabellenvariable deklariert, die zwei Variablen (ID und Jahr) aufnimmt.
2. Es werden sechs Datensätze eingefügt, drei davon NULL.
3. Die Datensätze werden mit 2010 aktualisiert, wenn sie NULL sind, ansonsten mit dem existierenden Wert.
4. Jetzt kommt die OUTPUT-Routine:
   a) Wenn der gelöschte (überschriebene) Wert dem eingefügten Wert entspricht, wird die „Jahr enthielt. . . “ Zeile ausgegeben
   b) ansonsten die „Jahr auf. . . “ Zeile

```
1  create table #t(id int identity(1,1)
2  primary key not NULL, jahr char(4) NULL)
3
4  insert into #t
5  select NULL union all
6  select NULL union all
7  select NULL union all
8  select '2001' union all
9  select '2002' union all
10 select '2003'
11
12 update #t set
13    jahr = case
14           when jahr is NULL then '2010'
15           else jahr
16       end
17 output
18    case when deleted.jahr = inserted.jahr
19         then 'id ' + cast(inserted.id as varchar(12)) + '
                jahr enthielt Wert ' + deleted.jahr
20       else 'id ' + cast(inserted.id as varchar(12)) + '
              jahr auf Wert ' + inserted.jahr + ' gesetzt'
21    end
22
23 drop table #t
```

17.16 Datensatz filtern

Bei Datensätzen, die in einem Merkmal identisch sind und von denen man nur eine Zeile benötigt, lässt sich mit folgendem Code die Anzahl der doppelten Zeilen ausgeben (und gegebenenfalls filtern).

```
1  CREATE TABLE #t(id int IDENTITY(1, 1)
2  PRIMARY KEY NOT NULL, text CHAR(10) NULL)
3
4  INSERT INTO #t
5  SELECT 'abc' UNION ALL
6  SELECT 'def' UNION ALL
7  SELECT 'ghi' UNION ALL
8  SELECT 'def' UNION ALL
9  SELECT 'ghi' UNION ALL
10 SELECT 'jkl' UNION ALL
11 SELECT 'mno'
12
13 SELECT * FROM #t
14
15 SELECT ROW_NUMBER() OVER(PARTITION BY text
16 ORDER BY text) AS 'Row Number', text
17 FROM #t
18
19 DROP TABLE #t
```

Listing 52: ROWNUMBER/PARTITION Beispiel 

17.17 Kumulative Summen berechnen

Kumulative Summen lassen sich einfach mit dem folgenden Skript berechnen. Zu jedem Wert der amount-Spalte (t1) wird die Summe aller amount-Zeilen (t2)ausgegeben, deren ID kleiner als t1.

```
CREATE TABLE #t(id int IDENTITY(1, 1)
PRIMARY KEY NOT NULL, amount INT NULL)

INSERT INTO #t
SELECT 10 UNION ALL
SELECT 20 UNION ALL
SELECT 30 UNION ALL
SELECT 40

SELECT t1.Amount,
(
  SELECT SUM(t2.Amount)
  FROM #t AS t2
  WHERE t2.id <= t1.id
) AS Total
FROM #t AS t1

DROP TABLE #t
```

Listing 53: Kumulative Summen berechnen 

18 Neuerungen im SQL Server 2008

Aus dem i'x Artikel Völkl, 2008 die wesentlichsten Neuerungen im SQL Server 2008:

18.1 Überblick

- Datenbankmodul
  - bessere Kommunikation bei Spiegelung, automatische Seitenreparatur bei Spiegelung
  - Verwaltung: SQL Audit (Überwachung von Datenbankereignissen), Komprimierung der Sicherung, automatisches Ablegen geänderter Daten in relationaler Form, zentraler Verwaltungsserver möglich, Hinzufügen von CPUs im laufenden Betrieb, verbesserte Kontrolle der Ressourcenzuteilung, richtlinienorientierte Verwaltung, T-SQL Debugger und Intellisense im Management Server
  - Programmierbarkeit: komprimierte Speicherung von Tabellen und Indizes, Speichern von Daten in Dateien (Filestream), bessere Performance bei Spalten mit vielen NULL-Einträgen, Datums- und Uhrzeitdatentypen, Datentyp für hierarchische Daten, Geodaten, GROUP-BY-Optionen mit ROLLUP und CUBE, MERGE
  - Sicherheit: transparente Datenverschlüsselung, externe Schlüsselverwaltung
- Integration Services
  - VSTA: neue Skriptumgebung
  - Datenprofilerstellungs-Task und Datenprofil-Viewer
- Analysis Services

  - Vereinfachung im Design von Aggregaten, Cubes und Dimensionen
  - Personalisierungserweiterung
  - Partitionierung von Trainings- und Testdaten
  - Verbesserung des Time-Series-Algorithmus
- Reporting Services
  - Berichterstellung: erweiterter Grafikdatenbereich, Messgeräte- und Tablixdatenbereich
  - Rendering: neu als Word-Dokument, Excel mit Unterberichten, CSV, einheitliche Paginierung
  - Abhängigkeit vom IIS beseitigt, neues Tool für Berichtsserverkonfiguration

18.2 Neue Datentypen

- date
- time
- datetime2 (speichert jetzt Nanosekunden)
- datetimeoffset
- geometry für 2D-Koordinatensysteme (UDT)
- geography für geografische Daten (UDT)
- hierarchyid für hierarchische Daten (UDT)

Index

Literatur

Allen, Scott (2005). *Table Variables In T-SQL*. http://www.odetocode.com/articles/365.aspx.

Graziano, Bill (2001). *Temporary Tables*. http://www.sqlteam.com/article/temporary-tables.

Kansy, Thorsten (2007). *Aus dem aktuellen dot.net magazin - Der T-SQL-Knigge*. http://it-republik.de/dotnet/artikel/Aus-dem-aktuellen-dot.net-magazin----Der-T-SQL-Knigge-1334.html.

Microsoft, Hrsg. (2007). *Katalogsichten (Transact-SQL)*. http://msdn.microsoft.com/de-de/library/ms174365(SQL.90).aspx.

Modi, Vijay (2007). *MS SQL Server: Stored Procedures vs. Functions*. http://vijaymodi.wordpress.com/2007/04/04/stored-procedures-vs-functions/.

Völkl, Gerhard (2008). »Von Stockwerk zu Stockwerk - Microsoft SQL Server 2008«. In: *i'x* 12, S. 82–84.

Wikipedia, Hrsg. (2009). *Stored Procedure*. http://de.wikipedia.org/wiki/Stored_Procedure.