FB 4 – Wirtschaftsinformatik
Prof. Dr. Peter Zschockelt

Hochschule für Technik
und Wirtschaft Berlin
University of Applied Sciences

Kurzreferenz SQL-Server 2008 (2)

Dieser Teil der Kurzreferenz SQL-Server 2008 enthält die wesentlichsten Teile der programmierbaren Funktionalität des Datenbankmanagementsystems, nämlich die Erstellung von gespeicherten Prozeduren (Stored Procedures), von nutzerindividuellen Funktionen (nur Skalarwertfunktionen) und von Triggern (nur tabellengebundene Trigger).

Gespeicherte Prozeduren (Stored Procedures) dienen zur Speicherung funktionaler Operationen in der Datenbank. Stored Procedures werden zu einem beliebigen Zeitpunkt definiert (CREATE PROCEDURE ...) und sind dann für einen beliebigen Aufruf verfügbar. Der Aufruf seitens des Nutzers erfolgt über eine EXECUTE-Anweisung.

Benutzerdefinierte Funktionen sind den gespeicherten Prozeduren bezüglich Verwaltung und Syntax sehr ähnlich. Der wesentlichste Unterschied besteht in der Nutzung. Während gespeicherte Prozeduren explizit mit einer SQL-Anweisung aufgerufen werden, können benutzerdefinierte Funktionen innerhalb einer SQL-Anweisungen wie systeminterne Funktionen benutzt werden.

Trigger sind ihrem Wesen nach ebenfalls gespeicherte Prozeduren. Sie werden jedoch nicht explizit aufgerufen, sondern automatisch als Folge einer Datenbankmanipulation ausgeführt, an die ein Trigger gebunden ist.

Stored Procedures (SP), Funktionen und Trigger werden innerhalb der Datenbank verwaltet und sind deshalb in der Regel außerordentlich effektiv. Auf Grund der engen technologischen Bildung an das DBMS sind häufig nur proprietäre Frontends in der Lage, SP und Trigger zu kreieren und zu ändern. Die Ausführung der SP ist aber über beliebige SQL-fähige Clients möglich.

Da sowohl Stored Procedures und Funktionen als auch Trigger proprietär in den DBMS unterschiedlich formuliert werden, wird nachfolgend nur beispielhaft auf wenige ausgewählte T-SQL Statements eingegangen. Für die vollständige Dokumentation nutzen Sie bitte die T-SQL-Hilfe.

1. Notation und generelles Sprachformat

Als **Notationsform** wird eine Mischung zwischen SQL-Quelltext-Kommentaren und der für die Beschreibung von Programmiersprachen üblichen Symbolik verwendet. Im Einzelnen bedeuten:

- Zeichenfolgen nach **--** (2 Minuszeichen) — SQL-Kommentare,
- **[** Variable/SQL-Worte in eckigen Klammern **]** — wahlfreie Angaben,
  (Bitte verwechseln Sie dies nicht mit der für SQL Server generell möglichen Angabe von Namen in eckigen Klammern !)
- { Auswahltex t in / mehreren Zeilen } — alternative Angaben (**eine** muss gewählt werden)
- Worte in Großbuchstaben — SQL-Schlüsselworte,
- Worte in Kleinbuchstaben — Variable.

Die allgemeine **Darstellungsform** von SQL ist formatfrei. Die einzelnen SQL-Schlüsselworte und/oder Variablen müssen jedoch immer vollständig in einer Zeile stehen und dürfen nicht abgeteilt werden.

Als **Trennzeichen** zwischen

- einzelnen Bezeichnern (SQL-Schlüsselworte, Variable, Literale) stehen 1 bis n Leerzeichen (Steht eine runde Klammer, ist kein Trennzeichen erforderlich.);
- einzelnen SQL-Anweisungen steht bei SQL-Server 2008 das Wort **GO** oder ein Semikolon;
- den Elementen einer Liste (z. B. Feldliste) stehen Kommas (**,**).

Datenbank-Objekt-Namen werden in Standard-SQL mit Buchstaben und Ziffern dargestellt (z. B. Create procedure *NeuSP*). Sie können in SQL-Server aber auch durch eckige Klammern eingeschlossen werden. Diese T-SQL-Erweiterung ist zum Beispiel nützlich, wenn Leer- oder Sonderzeichen in Spaltennamen enthalten sein sollen (z. B. Create procedure *[Neue Proc]*)

Generell können Datenbankobjekte mit dem Datenbanknamen, dem Schemanamen, Tabellennamen und ggf. Spaltennamen bezeichnet werden. Sofern Eindeutigkeit gegeben ist, können jedoch alle Angaben bis zum Tabellen- bzw. Spaltennamen entfallen. Bei Verwendung nutzerdefinierter Funktionen ist der Schemaname anzugeben.

2. Gespeicherte Prozeduren (Stored Procedures)

```
CREATE PROCEDURE procedurename
parameterdefinition, …
AS
variablendefinition
…
anweisung
…
GO
```

erzeugt eine gespeicherte Prozedur (SP).

Bemerkungen:

- Der **`procedurename`** kann in eckige Klammern eingeschlossen werden. In diesem Fall dürfen auch Leer- und Sonderzeichen im Procedurnamen stehen
  z. B. CREATE PROCEDURE [Referenz Angebot]
- eine **`parameterdefinition`** beginnt immer mit dem Zeichen @ und besteht aus dem Parameternamen und Datentyp. Stored Procedures ohne Parameter sind ebenso möglich wie Procedures mit eine Parameterliste,
  z. B. @bnr char (5), @anr char (6)
- eine **`variablendefinition`** beginnt mit dem Schlüsselwort DECLARE. Dem folgen der Variablenname (beginnt immer mit dem Zeichen @) und der Datentyp. Beachten Sie, dass Variablendefinitionen nicht durch Komma getrennt werden.
  z. B. declare @zaehler integer declare @regnr char (6)
- **`anweisung`** kann eine SQL-Anweisung oder eine SPL-Anweisung sein. SPL (=Stored Procedure Language) beinhaltet prozedurale Elemente und ist Bestandteil von T-SQL.

  Einige beispielhaft ausgewählte wesentliche SPL-Befehle sind:

SPL-Befehl	Beschreibung und Beispiele
DECLARE	Definition von Variablen *DECLARE @n INT* *DECALRE @ort CHAR (20)*
SET *oder* **SELECT**	Wertzuweisung an Variable *SET @n = 0* *SELECT @x = (SELECT a FROM tab WHERE y=1)*
IF ...[ELSE]	Verzweigung mit einem IF-Zweig, und wahlweise ELSE-Zweig *IF (@a > @b)* *@resultat = 1* *ELSE* *@resultat = 0*
GOTO marke	Gehe zu einer Marke

SPL-Befehl	Beschreibung und Beispiele
IF ...[ELSE] (Fortsetzung)	*Sollen mehrere Anweisungen hinter IF bzw. ELSE stehen, werden diese in einem BLOCK zusammengefasst, z. B.* *if (Select count(*) from Bkopf where regnr='12345') > 0* *BEGIN* *set @x = 1* *set @n = @n +1* *END*
WHILE bedingung	*Bedingungsschleife für eine Anweisung oder einen Anweisungsblock* *WHILE @i < 10* *insert into temp values (@i, getdate())* *…* *WHILE (Select anr from angebot) = @anr* *update angebot set preis = preis + 5 where anr = @anr*
BREAK	Beenden einer Schleife
RAISERROR	Benutzerdefinierte Ad-hoc-Fehlermeldung mit den Parametern msg_str, severity, state; wobei gilt - msg_str = Benutzerdefinierter Fehlertext, - severity = Fehlerschweregrad zwischen 0 bis 18 festlegbar, - state = Aufrufstatus des Fehler zwischen 1 bis 127 festlegbar. Die standardmäßig erzeugte Fehler-ID ist immer 50000 Beispiel: If (@nr=0) goto E_OAError CleanUp: return E_OAError: Select @ftext = 'Kein gültiges Angebot für Bestellnr. ' + @bnr + ' und Artikel ' + @anr raiserror(@ftext,16,1) GOTO CleanUp
RETURN	Sofortiger Rücksprung aus einer Prozedur, wahlweise kann eine nachfolgende ganze Zahl als Rückgabecode angegeben werden. *RETURN 1.*

Beispiel einer kompletten Stored Procedures:

```
-- SP Angebotskontrolle zum Datenmodell EINKAUF
CREATE PROCEDURE [Referenz Angebot]
@bnr char (5), @anr char(6)
 AS
declare @nr integer declare @regnr char(6)
declare @ftext char(61)
select @regnr = (select regnr   from bkopf where bnr=@bnr)
Set @nr = (select Count(anr) from angebot where anr=@anr and regnr=@regnr)
If (@nr=0)
 goto E_OAError
CleanUp:
   return
E_OAError:
Select  @ftext = 'Kein gültiges Angebot für Bestellnr. ' + @bnr + ' und Artikel ' + @anr
  raiserror(@ftext,16,1)
  GOTO CleanUp
go
```

```
EXECUTE procedurename parameter
```

führt eine Prozedur aus, ggf. mit Parametern (als Kürzel ist auch EXEC erlaubt).
z. B. Exec [Referenz Angebot] '12345' '666666'

```
DROP PROCEDURE procedurename
```

löscht eine Prozedur.

3. Benutzerdefinierte Funktionen

Benutzerdefinierte Funktionen werden beispielhaft nur für skalare Werte beschrieben. Für alle weitergehenden Aufgaben informieren Sie sich bitte in der T-SQL-Hilfe.

```
CREATE FUNCTION [ owner_name. ] function_name
  ( [  @parameter_name [AS] scalar_parameter_data_type  [ ,...n ] ] )
RETURNS scalar_return_data_type
 [ AS ]

BEGIN
    function_body
    RETURN scalar_expression
END
```

erzeugt eine Funktion. Dabei gilt:

- Der **function_name** kann in eckige Klammern eingeschlossen werden. In diesem Fall dürfen auch Leer- und Sonderzeichen im Namen stehen.
- Nach **RETURNS** ist der Datentyp des Rückgabewertes anzugeben.
- Im Funktionskörper (**function_body**) können im Prinzip gleiche T-SQL-Anweisungen verwendet werden wie für SP, jedoch darf das Ergebnis aller Operationen immer nur ein skalarer Wert sein.
- Mit **RETURN** wird der berechnete Wert zurück gegeben.

Beispiel einer komletten Funktion:

```
-- Funktion Währungsformatierung
CREATE FUNCTION formatmoney (@mon money)
RETURNS numeric(15,2)
AS
BEGIN
DECLARE @Money numeric (15,2)
SET @Money = cast((@mon)as numeric(15,2))
Return(@Money)
END
```

Die Nutzung einer Funktion erfolgt durch Angabe des Funktionsnamens und der Parameter in einer SQL-Anweisung. Dabei ist zu beachten, dass in der Regel der Name des Funktionseigentümers mit angegeben werden muss.

Beispiel:

```
-- Anwendung (Owner der Funktion muss im Namen angegeben werden)
Select anr, preis*1.16 as [Brutto unformatiert],
     dbo.formatmoney(preis*1.16) as [Brutto formatiert] from Angebot
go
```

4. Trigger

Für MS SQL Server 2008 sind umfangreiche Varianten der Triggergestaltung möglich. Nachfolgend soll lediglich eine gängige Variante beispielhaft erklärt werden. Für weiterführende Erklärungen sei wiederum auf die T-SQL-Hilfe verwiesen.

```
CREATE TRIGGER triggername
trigger_Event
AS
Anweisung
…
GO
```

erzeugt einen Trigger.

Bemerkungen:

- **trigger_Event** bezeichnet die Operation, an welche die Aktivierung des Triggers geknüpft wird. Möglich sind unter anderem
  - ON tabellenname for INSERT,
  - ON tabellenname for DELETE,
  - ON tabellenname for UPDATE.

  Eine weitere Einschränkung der Wirkungsweise eines Triggers auf bestimmte Spalten einer Tabelle ist innerhalb der Triggeranweisungen möglich. Fernerhin kann an Stelle von tabellenname auch der Name einer änderbaren View stehen. Informieren Sie sich hierzu in der SQL-Online-Hilfe.
- **AS** kennzeichnet den Beginn der bei Triggeraktivierung auszuführenden Anweisungen. Im Prinzip sind wiederum die gleichen Anweisungen wie für SP nutzbar. Es stehen jedoch eine Reihe zusätzlicher Objekte zur Verfügung, um eine effektive Nutzung der Trigger zu ermöglichen.
  Von besonderer Bedeutung ist dabei die **Bezugnahme auf die den Trigger auslösende Zeile(n)** einer Tabelle. Sie wird durch folgende zwei vordefinierte Objekte realisiert:
  - eine virtuelle Tabelle **inserted** stellt eine Kopie der triggerauslösenden Zeile nach Abarbeitung der triggerauslösenden SQL-Anweisung und
  - eine virtuelle Tabelle **deleted** stellt eine Kopie der triggerauslösenden Zeile vor Abarbeitung der triggerauslösenden SQL-Anweisung

  jeweils für die Dauer der Ausführung des Triggers bereit. Die Verwendung wird an nachfolgendem Beispiel erklärt.
- Die Anweisung **ROLLBACK TRANSACTION** bedeutet, dass die gesamte Transaktion, in der ein Trigger aktiviert wurde, abgebrochen wird. Sinnvoll ist diese Anweisung damit nur als Bestandteil einer IF-Anweisung, die einen Fehlerfall abprüft.

Beispiel:

```
-- Trigger
CREATE TRIGGER insangebot
ON bart
FOR INSERT
AS
declare @pbnr char (5)
declare @panr char (6)
declare @fnr  integer

Select @pbnr = (Select inserted.bnr from inserted)
Select @panr = (Select inserted.anr from inserted)
execute [Referenz Angebot] @pbnr, @panr
Select @fnr = @@ERROR
if @fnr=50000 ROLLBACK TRANSACTION
GO
```

In diesem Beipiel wird der Trigger bei Einfügen einer Zeile in die Tabelle BART ausgelöst (ON bart FOR INSERT). In den beiden Select-Anweisungen zur Ermittlung der Variableninhalte Bestellnummer (@pbnr) und Artikelnummer (@panr), die als Parameter für den Aufruf der SP [Referenz Angebot] benötigt werden, wird auf die Tabellenzeile von BART zurückgegriffen, die durch die INSERT-Anweisung hinzugefügt wurde. Dafür wird der virtuelle Tabellenname INSERTED benutzt.
Sofern die SP [Referenz Angebot] einen Fehlerwert 50000 zurückliefert (Standardfehlerwert für nutzerindividuelle Fehlermeldungen) wird mit ROLLBACK TRANSACTION die gesamte INSERT-Operation zurück gesetzt.

```
DROP TRIGGER triggername
```

löscht einen Trigger.