Transact-SQL – Variablen und Kontrollstrukturen

Stephan Karrer

Transact-SQL: Prozeduraler Anteil

ist eine Micosoft-spezifische (Sybase) Erweiterung, um auf die Datenbank zuzugreifen

erweitert SQL um die Elemente einer klassischen prozeduralen Programmiersprache:

- Explizite Cursor
- Variablen
- Bedingungen und Schleifen
- Prozeduren und Funktionen
- Ausnahmebehandlung

Vor- und Nachteile?

- gut integriert mit SQL
- ideal für die Erstellung und Wiederverwendung von gespeichertem Code auf dem Datenbank-Server
- läuft überall auf Plattform SQL Server
- Performanz

- proprietäre Sprache
- Interaktion mit dem Anwender wird wenig unterstützt

Syntaxregeln

Innerhalb des Stapels/Blocks werden die einzelnen Anweisungen mittels Semikolon abgeschlossen (wie bei SQL)

Als Trenner (Whitespace) dienen Leerzeichen, Tabulatoren und Zeilenumbrüche

Schlüsselwörter dürfen nicht getrennt werden

Groß/Kleinschreibung von Bezeichnern und Schlüsselworten ist irrelevant (wie in SQL)

Als Zeichensatz für das Programm steht der Zeichensatz des DBMS zur Verfügung (Zeichenketten als Daten können sehr wohl nationalen Konventionen folgen)

Es existieren Zeilen- und Block-Kommentare

Variablen: Deklaration und Wertzuweisung

```
USE HR;
GO
DECLARE @Job varchar(50), @Salary decimal(8,2) = 2000;
SET @Job = 'IT_PROG';
SELECT @Salary = 5000;
SET NOCOUNT OFF;
SELECT first_name, last_name, salary
FROM employees
WHERE job_id = @Job and salary >= @Salary;
```

- Variablennamen beginnen mit „@“
- Zuweisung erfolgt mit SET bzw. SELECT, Initialwert ist NULL
- Deklaration und Initialisierung kann auch in einem Schritt erfolgen
- Können direkt in SQL-Anweisungen verwendet werden

Variablen: Arithmetik

```
/* Example one */
DECLARE  @NewBalance  int=10 ;
SET  @NewBalance  =  @NewBalance  *  10;
SELECT  @NewBalance;

GO

/* Example Two */
DECLARE @NewBalance int = 10;
SET @NewBalance *= 10;
SET @NewBalance = SQUARE(@NewBalance);
SELECT @NewBalance;
```

- Mit den Variablen sind auch die für den Datentyp vorhandenen Operationen und Funktionen möglich

Gültigkeitsbereich von Variablen

```
DECLARE @MyVariable int;
SET @MyVariable = 100;
-- Terminate the batch by using the GO keyword.
GO
-- @MyVariable has gone out of scope and no longer exists.

-- This SELECT statement generates a syntax error because it is
-- no longer legal to reference @MyVariable.
SELECT employee_id, last_name
FROM employees
WHERE employee_id = @MyVariable;
```

- Der Gültigkeitsbereich erstreckt sich ab der Deklaration bis zum Ende des Stapels

Wertzuweisung via Unterabfrage

```
DECLARE @rows int;
SET @rows = (SELECT COUNT(*) FROM jobs);
SELECT @rows;
```

- Selbstverständlich darf die Abfrage höchstens einen Wert zurückliefern (NULL ist erlaubt)

Wertzuweisung innerhalb SELECT

```
DECLARE @EmpIDVariable int;

SELECT @EmpIDVariable = employee_id
FROM employees
ORDER BY employee_id DESC;

SELECT @EmpIDVariable;
```

- Vorsicht:
  Falls die Abfrage viele Werte liefert, wird die Variable auf den letzten Wert gesetzt! (impliziter Cursor)

Wertzuweisung innerhalb SELECT: Nutzung des impliziten Cursor

```
-- Die Namen aller Angestellten durch Semikolon
-- getrennt aneinanderhängen
declare @Liste varchar(max)
select @Liste = isnull(@Liste + '; ','') + last_name
   from employees
   order by last_name

-- Ergebnis im Meldungsbereich ausgeben
print @Liste
```

Dynamisches SQL mit Variablen

```
-- Eine SELECT-Anweisung in einer Variablen
-- speichern und ausführen
declare @cmd varchar(max) set @cmd =
   'select first_name, last_name'
          + ' from employees where employee_id = 100'
exec(@cmd)

-- Die gespeicherte Prozedur sp_configure als
-- dynamisches SQL ausführen
set @cmd = 'sp_configure'
exec @cmd
```

- Mit Hilfe von Variablen für den Anweisungstext lässt sich dynamisches SQL (wird zur Laufzeit interpretiert) realisieren

Dynamisches SQL mit Prozedur sp_executesql

```
declare @cmd nvarchar(max) ;
set @cmd = N'select first_name, last_name
                from employees where employee_id = @eid';
exec sp_executesql @cmd, N'@eid int', @eid=100 ;
exec sp_executesql @cmd, N'@eid int', @eid=110 ;
```

- Bequemer geht dynamisches SQL mit der Systemprozedur sp_executesql

Blockstruktur

```
BEGIN TRANSACTION;

IF @@TRANCOUNT = 0
 BEGIN
    BEGIN TRANSACTION
    SELECT first_name, last_name
    FROM employees
    WHERE last_name = 'Higgins';
    ROLLBACK TRANSACTION;
 END;

ROLLBACK TRANSACTION;
PRINT N'Rolled back the transaction.';
```

- Bei Bedarf können mehrere Anweisungen in einem Block zusammengefasst werden (im Bsp. würde es ansonsten eine Fehlermeldung geben)

- Blöcke dürfen geschachtelt werden, aber Variablen können nicht verdeckt werden!

Bedingte Ausführung mit IF - THEN

```
DECLARE
  @sales decimal(8,2) = 10100,
  @quota decimal(8,2) = 10000,
  @bonus decimal(6,2),
  @emp_id int = 120;
  IF @sales > (@quota + 200)
    BEGIN
        SET @bonus = (@sales - @quota)/4;
        UPDATE employees
            SET salary = salary + @bonus
            WHERE employee_id = @emp_id;
    END ;
```

- Der THEN-Zweig kommt ohne Schlüsselwort aus
- Mehrere Anweisungen im Zweig müssen mit BEGIN – END geklammert werden

Bedingte Ausführung mit IF – THEN - ELSE

```
DECLARE
  @sales decimal(8,2) = 10100,
  @quota decimal(8,2) = 10000,
  @bonus decimal(6,2),
  @emp_id int = 120;
  IF @sales > (@quota + 200)
    BEGIN
        SET @bonus = (@sales - @quota)/4;
        UPDATE employees
            SET salary = salary + @bonus
            WHERE employee_id = @emp_id;
    END
  ELSE
    PRINT 'kein Bonus';
```

- Der ELSE-Zweig ist optional
- Mehrere Anweisungen im Zweig müssen mit BEGIN – END geklammert werden
- Selbstverständlich kann geschachtelt werden

Bedingte Ausführung mit IF EXISTS

```
IF EXISTS (SELECT * FROM employees WHERE
employee_id=100)
    BEGIN
        PRINT 'Datensatz existiert';
    END
  ELSE
    BEGIN
        PRINT 'keinen Datensatz gefunden';
    END;
```

- In der IF-Bedingung kann die EXISTS-Klausel verwendet werden

Mehrfachauswahl mit CASE

```
DECLARE @anzahl int;
SELECT @anzahl = count(*) from employees
     WHERE department_id = 50;
SELECT CASE
           WHEN @anzahl < 5
               THEN 'Wenig Mitarbeiter'
           WHEN @anzahl between 5 and 20
               THEN 'Einige Mitarbeiter'
           ELSE 'Viele Mitarbeiter'
       END
 AS "Anzahl Mitarbeiter"
```

- Mit der CASE-Anweisung steht eine weitere Verzweigung zur Verfügung

CASE als Expression

```
DECLARE @anzahl int, @result varchar(30);
SELECT @anzahl = count(*) from employees
     WHERE department_id = 50;
SET @result =
       CASE
            WHEN @anzahl < 5
                THEN 'Wenig Mitarbeiter'
            WHEN @anzahl between 5 and 20
                THEN 'Einige Mitarbeiter'
            ELSE 'Viele Mitarbeiter'
       END ;
 SELECT @result;
```

- Das Ergebnis der CASE-Anweisung kann wiederum zugewiesen werden

While-Schleife

```
DECLARE @counter int = 0;
WHILE @counter < 10
    BEGIN
       PRINT 'Zähler ist: ' + CAST(@counter AS varchar);
       SET @counter = @counter + 1;
       IF @counter > 7
            CONTINUE;
       DECLARE @innercounter int = 0;
       WHILE @innercounter < @counter
        BEGIN
            PRINT 'Innerer Zähler ist: '
                    + CAST(@innercounter AS varchar);
            SET @innercounter = @innercounter + 1;
            IF @innercounter = 6
                BREAK;
        END;
    END;
```

- Schleifen können geschachtelt werden
- BREAK und CONTINUE stehen ohne Sprungmarken zur Verfügung

Sprünge mit GOTO

```
DECLARE @Counter int;
SET @Counter = 1;
WHILE @Counter < 10
BEGIN
    SELECT @Counter
    SET @Counter = @Counter + 1
    IF @Counter = 4 GOTO Branch_One --Jumps to the first branch.
    IF @Counter = 5 GOTO Branch_Two  --Will never execute.
END
Branch_One:
    SELECT 'Jumping To Branch One.'
    GOTO Branch_Three; --Prevent Branch_Two from executing.
Branch_Two:
    SELECT 'Jumping To Branch Two.'
Branch_Three:
    SELECT 'Jumping To Branch Three.'
```

- Das sollte heutzutage ein NoGo sein!!

Zeitgesteuerte Ausführung mit WAITFOR

```
WAITFOR DELAY '00:00:02';
SELECT employee_id FROM employees;

GO

BEGIN
    WAITFOR TIME '22:00';
    DBCC CHECKALLOC;
END;
GO
```