Transact-SQL: Prozeduren und Funktionen

Stephan Karrer

Prozeduren und Funktionen

- Sind faktisch benannte Blöcke, die parametrisiert sein können
- Werden üblicherweise in kompilierter Form (Bytecode) in der Datenbank gespeichert, deshalb auch „gespeicherte“ Prozedur/Funktion
- Ein Prozeduraufruf steht für eine Anweisung in Transact-SQL
- Ein Funktionsaufruf steht für einen Ausdruck, der einen Wert zurückgibt (analog den vorhandenen SQL-Funktionen)

Funktionen und Prozeduren: Zentrale Speicherung auf dem Server

```
CREATE OR ALTER FUNCTION half_of_square (@param real)
   RETURNS real
AS
BEGIN
RETURN (@param * @param)/2 + (@param * 4);
END ;
```

Verwaltet via Data Dictionary

```
SELECT name AS "Name"
  ,SCHEMA_NAME(schema_id) AS schema_name
  ,type_desc
  ,create_date
  ,modify_date
FROM sys.objects
WHERE type_desc LIKE '%PROCEDURE%' OR
      type_desc LIKE '%FUNCTION%';
```

Beispiel für eine Prozedur ohne Parameter

```
CREATE PROCEDURE GetAllEmployees
AS
    SET NOCOUNT ON;
    SELECT last_name, first_name, salary
        FROM employees;
    SELECT * FROM jobs;
GO
```

```
-- Aufruf

EXECUTE GetAllEmployees;
GO
-- Or
EXEC GetAllEmployees;
GO
-- Or, if this procedure is the first statement
-- within a batch:
GetAllEmployees;
```

- Ausgaben erfolgen wie üblich

Beispiel für eine Prozedur mit Parameter

```
CREATE OR ALTER PROCEDURE GetEmployees
    @LastName VARCHAR(50),
    @FirstName VARCHAR(50)
AS
    SET NOCOUNT ON;
    SELECT first_name, last_name, salary
    FROM employees
    WHERE first_name = @FirstName AND
          last_name = @LastName;
GO
```

```
-- Aufruf

EXEC GetEmployees 'King', 'Steven';

-- Or
EXEC GetEmployees @LastName = 'King', @FirstName = 'Steven';

-- Or
EXEC GetEmployees @FirstName = 'Steven', @LastName = 'King';
```

Beispiel für eine Prozedur mit OUT-Parameter

```
CREATE OR ALTER PROCEDURE InOut
    @in1 int = 1,  --default
    @in2 int,
    @res int OUT
AS
    SET @in1 = @in2 + @in1;
    SET @res = @in1 + @res;

GO
```

- Default-Werte für Parameter sind möglich
- Die Input- und Output-Parameter sind schreibbar

```
-- Aufruf
DECLARE @result int;
EXEC InOut 5, 2, @result OUT;
SELECT @result;

-- Or
EXEC InOut default, 2, @result OUT;
SELECT @result;

-- Or
EXEC InOut @in1 = default, @in2 = 2,
           @res = @result OUT;
SELECT @result;
```

- Der Output-Parameter ist eine Variable, die nach dem Aufruf überschrieben ist
- Der Output-Parameter muss als solcher auch beim Aufruf übergeben werden

Vergleich der Parameter-Modi

- IN-Modus:
  - kann nicht explizit angegeben werden (default)
  - kann in der Prozedur verändert werden
  - wird per Value übergeben
  - Default-Wert kann benutzt werden
- OUT-Modus:
  - muss explizit angeben werden
  - aktueller Parameter muss Variable sein, kein Default-Wert möglich
  - wird in der Regel in der Prozedur gesetzt für Wert-Rückgabe
  - kann im Sub-Programm gelesen und geschrieben werden
  - wird bei Rückkehr kopiert, sofern kein Fehler auftritt

Keine Schachtelung mit Funktionsaufrufen

```
CREATE OR ALTER PROCEDURE InOut
    @in1 int = 1,  --default
    @in2 int,
    @res int OUT
AS
    SET @in1 = @in2 + @in1;
    SET @res = @in1 + @res;

GO
```

```
-- Aufruf
DECLARE @result int =20, @myin int = 5, @f int = FLOOR(2.3);
EXEC InOut @myin, @f, @result OUT;
SELECT @result, @myin;
```

- Leider ist kein Aufruf mit einem Funktionsaufruf als Parameter möglich!

Prozeduraufruf innerhalb einer Transaktion

```
CREATE PROCEDURE UpdateSalary
@sal decimal(8,2), @Rowcount INT OUTPUT
AS
SET NOCOUNT ON;
UPDATE employees
    SET salary = @sal
    WHERE employee_id = 100;
SET @Rowcount = @@rowcount;

GO

BEGIN TRANSACTION
DECLARE @Rowcount INT
EXEC UpdateSalary 30000, @Rowcount OUTPUT
PRINT @Rowcount;
COMMIT;
```

- DML-Anweisungen sollten natürlich in einer Transaktion ausgeführt werden
- Sofern in der Prozedur Transaktionsteuerung gemacht wird, sollte dies nicht mit der Transaktionsteuerung des Aufrufers kollidieren

Transaktion mit Fehler-Handling innerhalb der Prozedur

```
CREATE PROCEDURE DeleteDepartment ( @depid decimal(4,0) )
AS
SET NOCOUNT ON;
BEGIN TRY
  BEGIN TRANSACTION
    DELETE FROM Departments
    WHERE department_id = @depid;
   COMMIT
END TRY

BEGIN CATCH
  -- Determine if an error occurred.
  IF @@TRANCOUNT > 0
    ROLLBACK

  -- Return the error information.
  DECLARE @ErrorMessage NVARCHAR(4000), @ErrorSeverity INT;
  SELECT @ErrorMessage = ERROR_MESSAGE(),@ErrorSeverity =
ERROR_SEVERITY();
  RAISERROR(@ErrorMessage, @ErrorSeverity, 1);
END CATCH;

GO
EXEC DeleteDepartment 20;
GO
```

Berechtigung beim Ausführen

```
CREATE OR ALTER PROCEDURE UpdateSalary
@sal decimal(8,2), @Rowcount INT OUTPUT
WITH EXECUTE AS OWNER
AS
SET NOCOUNT ON;
UPDATE employees
    SET salary = @sal
    WHERE employee_id = 100;
SET @Rowcount = @@rowcount;

GO
```

- Via WITH EXECUTE AS kann die Berechtigung sowohl bei Prozeduren als auch Funktionen gesetzt werden

Beispiel für eine Funktion

```
CREATE OR ALTER Function subtract( @in1 int = 1, @in2 int )
    RETURNS int
AS
BEGIN
    SET @in1 = @in1 - @in2;
    RETURN @in1;
END;
GO

-- Aufruf
SELECT dbo.subtract(4, 5);    SELECT dbo.subtract(default, 5);
```

- Bei Funktionen erfolgt die Parameterangabe in runden Klammern
- Es gibt keine OUT-Parameter, sondern einen Return-Wert
- Der Rumpf muss mit BEGIN … END geklammert werden
- Aufruf kann wie bei herkömmlichen SQL-Funktionen erfolgen, wobei „default“ angegeben werden muss

Beispiel für eine Funktion mit SQL-Anweisungen

```
CREATE OR ALTER FUNCTION get_salary
 (@empid decimal(6,0)) RETURNS decimal(8,2)
AS
BEGIN
  DECLARE @sal decimal(8,2) = 0;
  SELECT  @sal = salary
    FROM   employees
    WHERE  employee_id = @empid;
  RETURN @sal;
END;
GO

DECLARE @sal decimal(8,2);
EXEC @sal = get_salary 100;
SELECT @sal;
```

- Selbstverständlich können Funktionen SQL-Anweisungen einsetzen
- Allerdings erfolgt keine Standardausgabe der Anweisungen
- Aufruf via EXEC ist möglich

Verwenden von Funktionen in SQL

Analog zu den integrierten SQL-Funktionen können benutzerdefinierte skalare Funktionen verwendet werden in:

- SELECT-Liste von Abfragen
- WHERE- und HAVING-Klauseln bei Abfragen und DML-Anweisungen
- ORDER BY-, GROUP BY-Klauseln bei Abfragen
- VALUES-Klausel bei INSERT-Anweisungen
- SET-Klausel bei UPDATE-Anweisungen

Table-Valued Funktion

```
CREATE FUNCTION EmpDeps (@depid decimal(4,0))
RETURNS TABLE
AS
RETURN (
    SELECT e.employee_id, e.last_name, d.department_name
    FROM employees e
    INNER JOIN departments d  ON e.department_id=
    d.department_id
    WHERE d.department_id = @depid
);
GO

SELECT * FROM dbo.EmpDeps(50);
```

- Funktionen können auch tabellenwertige Rückgaben haben
- Dann darf der Rumpf nur aus einer eventuell komplexen SELECT-Anweisung bestehen (auch ein CTE ist möglich)
- Kann wie ein View mit Parametern verwendet werden