SQL Server Data Manipulation (DML)

Stephan Karrer

Datenmanipulation (DML)

- Data Manipulation Language Statements:
  - INSERT,
  - UPDATE,
  - DELETE
  - MERGE

Einfügen einzelner Zeilen

```
INSERT  INTO departments (department_id, manager_id,
                          location_id, department_name)
        VALUES ( 70, 100, 1700, 'Public Relations' );

- Sofern die Spaltenreihenfolge bekannt ist:
INSERT  INTO departments
        VALUES ( 70, 'Public Relations', NULL, NULL);
```

- Das Schlüsselwort INTO darf weggelassen werden
- Sofern die Spaltenliste angeben wird, folgen die Werte dieser Reihenfolge
- Die Werte müssen einen kompatiblen Datentyp haben und bzgl. eventueller Beschränkungen passen

Spezialfälle

```
CREATE TABLE T  ( c1 int IDENTITY,
                  c2 varchar(20),
                  c3 AS 'Computed' + c2,
                  c4 int DEFAULT 1,
                  c5 int NULL );

INSERT INTO T (c2) VALUES ('otto');
INSERT INTO T (c2, c4, c5) VALUES ('hugo', 3, 4);
INSERT INTO T (c2, c4, c5) VALUES ('hugo', DEFAULT, 4);
INSERT INTO T DEFAULT VALUES;

SET IDENTITY_INSERT T ON;
INSERT INTO T (c1, c2) VALUES (99, 'otto');
```

- Spalten mit DEFAULT-Werten oder Null-fähige Spalten müssen nicht angegeben werden, Schlüsselwort DEFAULT kann verwendet werden
- Berechnete oder Identity-Spalten können normalerweise nicht gesetzt werden

Einfügen mehrerer Zeilen

```
-- Test-Tabelle
CREATE TABLE my_departments ( id      DECIMAL(4,0),
                              name    VARCHAR(30) );


INSERT  INTO my_departments ( id, name)
        VALUES  ( 1 , 'Abteilung1'),
                ( 2 , 'Abteilung2'),
                ( 3 , 'Abteilung3');


INSERT  INTO my_departments (id, name)
        SELECT department_id, department_name
                FROM departments
                WHERE manager_id = 100;
```

Erzeugen einer Ziel-Tabelle anhand einer Anfrage

```
SELECT employee_id, last_name, salary
    INTO my_employees1
    FROM employees WHERE department_id = 50;


SELECT employee_id, last_name, salary
    INTO my_employees2
    FROM employees
    WHERE 1 = 0;    -- verhindert die Werteübernahme
```

- Es werden Spaltentyp, -reihenfolge, -name anhand der Abfrage übernommen
- Nullable und Identity-Eigenschaft wird übernommen, alle anderen Einschränkungen nicht.

Erzeugen einer Ziel-Tabelle anhand einer Anfrage

```
SELECT employee_id, last_name, salary
    INTO my_employees1
    FROM employees WHERE department_id = 50;


SELECT employee_id, last_name, salary
    INTO my_employees2
    FROM employees
    WHERE 1 = 0;    -- verhindert die Werteübernahme
```

- Es werden Spaltentyp, -reihenfolge, -name anhand der Abfrage übernommen
- Nullable und Identity-Eigenschaft wird übernommen, alle anderen Einschränkungen nicht.

Vorhandene Zeilen ändern

```
UPDATE  my_employees
        SET last_name = 'Karrer',
            salary = 5000
        WHERE employee_id = 137 ;

UPDATE  my_employees
        SET salary = 7000 ;

UPDATE my_employees
        SET salary = DEFAULT  - sofern Default gesetzt
       WHERE employee_id = 120;
```

- Vorsicht: Ohne Filter werden alle Zeilen aktualisiert !

Spalten mit Unterabfragen aktualisieren

```
UPDATE  employees
        SET  job_id = (SELECT job_id FROM employees
                               WHERE employee_id = 207),
             salary = (SELECT salary FROM employees
                               WHERE employee_id = 134)
        WHERE employee_id = 67;
```

Zeilen löschen

```
DELETE FROM employees  WHERE employee_id = 67;

DELETE FROM my_employee;

DELETE FROM employees
       WHERE department_id = (SELECT department_id
                              FROM departments
                              WHERE department_name =
                                   'Public Relations' );
```

- Vorsicht: Ohne Filter werden alle Zeilen aktualisiert !

Veränderte Werte zurückgeben

```
CREATE TABLE Test( c1 int IDENTITY,
                  c2 varchar(20)   );

INSERT INTO Test (c2) OUTPUT INSERTED.c1 VALUES ('hugo');
INSERT INTO Test (c2) OUTPUT INSERTED.*  VALUES ('hugo');

UPDATE Test SET c2 = 'Test' OUTPUT INSERTED.c2 ;

DELETE FROM Test OUTPUT DELETED.*  WHERE c2 = 'Test';
```

- DML-Anweisungen (auch MERGE) haben eine OUTPUT-Klausel über die die veränderten bzw. gelöschten Werte zurückgegeben werden.

MERGE: Beispiel

```
MERGE INTO bonuses D     -- Ziel
  USING (SELECT employee_id, salary, department_id
          FROM employees
          WHERE department_id = 80)  S  -- Quelle
  ON (D.employee_id = S.employee_id)     -- Bedingung
  WHEN MATCHED THEN                      -- Aktion
     UPDATE SET D.bonus = D.bonus + S.salary*.01
  WHEN NOT MATCHED BY TARGET THEN        -- Aktion
     INSERT (employee_id, bonus)
     VALUES (S.employee_id, S.salary*0.1)
  WHEN NOT MATCHED BY SOURCE THEN DELETE  -- Aktion
OUTPUT  $ACTION, INSERTED.*, DELETED.*;   -- OUTPUT-Klausel
```

- Mit MERGE kann in einem Schritt eingefügt, verändert und gelöscht werden

Transaktionen

Mehrere Anweisungen sollen zu einer unteilbaren Aktion zusammengefasst werden.

Anforderung (ACID-Prinzip)

- Atomar (Atomicity)
- Entweder ist die gsamte Transaktion erfolgreich oder sie hat keinerlei Auswirkungen
- Konsistent (Consistency)
  Führt einen konsistenten Datenzustand wieder in einen konsistenten Zustand über (logische Anforderung)
- Isoliert (Isolation)
  Andere Transaktionen/Zugriffe sehen nur konsistente Datenzustände
- Persistent (Persistence)
  Bei Erfolgt wird der konstente Zustand dauerhaft gespeichert (DBMS-Anforderung)

Explizite Transaktionssteuerung

```
BEGIN TRANSACTION;
   -- Aktionen (DML oder DDL)
ROLLBACK;
-- Aktionen (AutoCommit-Modus)
BEGIN TRAN t1;
   -- Aktionen (DML oder DDL)
COMMIT t1;
```

- SQLServer verwendet standardmäßig AutoCommit-Modus:
  Jede SQL-Anweisung ist eine Transaktion, COMMIT erfolgt automatisch bei Erfolg
- Sollen mehrere Anweisungen in einer Transaktion zusammengefasst werden, kann man explizite Transaktionen verwenden
- Es gibt auch einen einstellbaren impliziten Transaktionsmodus, in dem DML- und DDL-Anweisungen automatisch eine neue Transaktion starten
  (Entspricht dem Standard von Oracle bzw. DB2)

SavePoints

```
BEGIN TRANSACTION;
   -- Aktionen (DML oder DDL)
   SAVE TRANSACTION P1
        -- Aktionen
   ROLLBACK TRANSACTION P1
   -- Aktionen
COMMIT;
```

- Innerhalb einer Transaktion können Zwischenpunkte gesetzt werden, auf deren Zustand ein lokaler Rollback erfolgen kann, ohne die gesamte Transaktion zurückzusetzen
- Entscheidend ist allerdings, wie die umfassende Transaktion beendet wird (COMMIT oder ROLLBACK). Ein Commit bzgl. SAVEPOINT hat keine Auswirkungen.

Isolations-Stufe

```
SET TRANSACTION ISOLATION LEVEL
       REPEATABLE READ;

BEGIN TRANSACTION;

   -- Aktionen (DML oder DDL)

COMMIT;

SET TRANSACTION ISOLATION LEVEL
       READ COMMITTED;
```

- Üblicherweise wird die Isolation der Transaktionen untereinander aufgeweicht, um die parallele Nutzung der Datenbank zu erhöhen
- ANSI definiert hier 4 Isolations-Stufen:
  Üblicherweise (so auch bei SQL Server) ist der Default „READ COMMITED“
- Das lässt sich allerdings für die jeweilige Transaktion auch anders einstellen