SQL – Data Manipulation Language

Stephan Karrer

Datenmanipulation (DML)

- Zeilen hinzufügen, Werte verändern, Zeilen löschen
- Data Manipulation Language Statements:
  - INSERT,
  - UPDATE,
  - DELETE,
  - MERGE

INSERT: Neue Zeilen einfügen

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


INSERT  INTO departments       --Spaltenreihenfolge!
        VALUES ( 70, 'Public Relations', NULL, NULL);
```

- Sofern die Spaltenliste angegeben wird, werden die Spalten gemäß dieser Reihenfolge befüllt.
- Selbstverständlich müssen die Daten zu den Datentypen und Constraints der Tabelle passen.

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;
```

- Selbstverständlich müssen auch hier die einzufüllenden Daten zu den Datentypen und Constraints der Tabelle passen.

Spezialfälle

```
CREATE TABLE T ( c1 int GENERATED ALWAYS AS IDENTITY,
                 c2 int GENERATED BY DEFAULT AS IDENTITY,
                 c3 varchar(20),
                 c4 varchar(30) GENERATED ALWAYS AS ('Computed'||c3),
                 c5 int DEFAULT 1,
                 c6 int NULL );

INSERT INTO T (c3) VALUES ('otto');
INSERT INTO T (c3, c5, c6) VALUES ('hugo', 3, 4);
INSERT INTO T (c3, c5, c6) VALUES ('hugo', DEFAULT, 4);
INSERT INTO T DEFAULT VALUES;
INSERT INTO T (c2) VALUES (100);
INSERT INTO T (c1) OVERRIDING SYSTEM VALUE VALUES (100);
```

- 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

UPDATE: Vorhandene Zeilen ändern

```
UPDATE  employees
        SET department_id = 70,
            manager_id = 100
        WHERE employee_id = 137 ;

UPDATE  my_employees
        SET department_id = 70 ;
```

- Selbstverständlich müssen auch hier die einzufüllenden Daten zu den Datentypen und Constraints der Tabelle passen.
- Vorsicht: Ohne Filter werden alle Zeilen aktualisiert !
- Das Schlüsselwort DEFAULT kann auch hier für den neuen WERT benutzt werden.

UPDATE: Spalten mit Unterabfragen aktualisieren

```
UPDATE  employees
        SET  job_id = (SELECT job_id FROM employees
                               WHERE employee_id = 207),
             department_id = (SELECT department_id
                               FROM departments
                               WHERE department_name = 'IT')
        WHERE employee_id = 67;
```

UPDATE: Spaltentupel aktualisieren

```
UPDATE employees
   SET  (job_id, department_id) =
               (SELECT job_id, department_id
                       FROM employees
                       WHERE employee_id = 104)
   WHERE employee_id = 101;
```

ANSI: das können neben PostgreSQL auch Oracle, DB2, aber nicht SQL Server

DELETE: 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 gelöscht !

MERGE: Beispiel

```
CREATE TABLE Bonuses AS SELECT employee_id, salary AS bonus
           FROM employees WHERE FALSE;

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 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
```

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

Veränderte Werte zurückgeben

```
BEGIN TRANSACTION;
UPDATE employees
   SET  (job_id, department_id) =
                (SELECT job_id, department_id
                        FROM employees
                        WHERE employee_id = 104)
   WHERE employee_id = 101
        RETURNING old.job_id AS old_jobid,
                  old.department_id AS old_depid,
                  new.job_id AS new_jobid,
                  new.department_id AS new_depid;
ROLLBACK;
```

- DML-Anweisungen (auch MERGE) haben eine RETURNING-Klausel über die die veränderten bzw. gelöschten Werte zurückgegeben werden.
- Insbesondere nützlich bei automatisch vergebenen Werten.

Transaktionskonzept bei DBMS

- Eine Transaktion umfasst eine oder mehrere Anweisungen für die gilt:

Atomic	Entweder alle Anweisungen sind erfolgreich oder keine
Consistent	Eine erfolgreiche Transaktion führt den Datenbestand in einen konsistenten Zustand (semantischer Begriff!).
Isolated	Die Zwischenzustände des Datenbestands während einer Transaktion sind für parallel laufende Zugriffe nicht sichtbar.
Durable	Die Ergebnisse einer erfolgreichen Transaktion werden in der Datenbank persistiert.

- ANSI fordert: Alle schreibenden Zugriffe müssen innerhalb einer Transaktion erfolgen.
  - ➢ dies betrifft auf jeden Fall alle DML-Anweisungen.

- Bei allen Systemen gilt:
  Eine einzelne SQL-Anweisung ist auf jeden Fall transaktional.

Umsetzung des Transaktionskonzepts

- Da die Anweisungsfolge innerhalb einer Transaktion anforderungsspezifisch ist:

  COMMIT für die explizite erfolgreiche Beendigung

  ROLLBACK für den Abbruch (und damit rückgängig machen aller Änderungen)

- Einige Systeme, z.B. Oracle, DB2 benutzen implizite Transaktionssteuerung:

  Eine neue Transaktion startet automatisch, wenn die vorherige Transaktion explizit oder implizit durch das System beendet wird und umfasst jetzt alle folgenden Anweisungen.

- Andere Systeme nutzen „AutoCommit“-Modus:

  Standardmäßig ist nur eine einzelne Anweisung eine Transaktion. Sollen mehrere Anweisungen in einer Transaktionsklammer ausgeführt werden, so muss diese explizit gestartet werden:

  - START TRANSACTION (ANSI) , herstellerspezifische Anweisungen sind auch üblich!

Transaktionsteuerung am Bsp. PostgreSQL

```
-- Ende der letzten Transaktion
BEGIN;
INSERT INTO departments
       VALUES (280, 'Recreation', 110, 1700);
UPDATE employees SET salary = 10 WHERE employee_id = 100;
SELECT * FROM employees WHERE employee_id =100;
ROLLBACK;
SELECT * FROM employees WHERE employee_id =100;

-- jetzt sind wir wieder im AutoCommit-Modus
-- nächste Transaktion beginnt
BEGIN;
UPDATE employees SET salary = 48000 WHERE employee_id = 100;
SAVEPOINT punkt1;
UPDATE employees SET salary = 200 WHERE employee_id = 100;
SELECT * FROM employees WHERE employee_id =100;
ROLLBACK TO SAVEPOINT punkt1;
SELECT * FROM employees WHERE employee_id =100;
COMMIT; -- comm = 100
SELECT * FROM employees WHERE employee_id =100;
-- jetzt sind wir wieder im AutoCommit-Modus
UPDATE employees SET salary = 24000 WHERE employee_id = 100;
```

Isolation-Level setzen

```
-- Ende der letzten Transaktion
BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ
INSERT INTO departments
       VALUES (280, 'Recreation', 110, 1700);
UPDATE employees SET salary = 10 WHERE employee_id =
100;
SELECT * FROM employees WHERE employee_id =100;
COMMIT;
```

- Der Isolation-Level lässt sich je Transaktion setzen oder global für die Session

Table 13.1. Transaction Isolation Levels

Isolation Level	Dirty Read	Nonrepeatable Read	Phantom Read	Serialization Anomaly
Read uncommitted	Allowed, but not in PG	Possible	Possible	Possible
Read committed	Not possible	Possible	Possible	Possible
Repeatable read	Not possible	Not possible	Allowed, but not in PG	Possible
Serializable	Not possible	Not possible	Not possible	Not possible