



# **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);
```

## INSERT: Zeilen aus vorhandenen Tabellen kopieren

```
INSERT INTO my_employees  
      SELECT * FROM employees;
```

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

## UPDATE: Vorhandene Zeilen ändern

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

```
UPDATE my_employees
    SET department_id = 70 ;
```

## 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 z.B. Oracle, DB2, PostgreSQL

## INSERT und UPDATE: Verwendung von DEFAULT-Werten

```
INSERT INTO departments
VALUES (280, 'Recreation', DEFAULT, 1700);

UPDATE departments
SET manager_id = DEFAULT
WHERE department_id = 20;
```



## 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' );
```

## 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;
```

## Transaktionsteuerung am Bsp. Oracle

```
-- Ende der letzten Transaktion  
-- implizit bei Oracle durch jede DDL- oder DCL-Anweisung  
-- egal ob erfolgreich oder nicht!
```

```
INSERT INTO departments
```

```
        VALUES (280, 'Recreation', DEFAULT, 1700);
```

```
UPDATE emp SET sal = 10;
```

```
ROLLBACK;
```

```
-- nächste Transaktion beginnt
```

```
UPDATE emp SET comm = 100;
```

```
SAVEPOINT punkt1;
```

```
UPDATE emp SET sal = sal * 1.1;
```

```
ROLLBACK TO SAVEPOINT punkt1;
```

```
COMMIT; -- comm = 100
```