

# Datenbank-Programmierung

---

## Kapitel 3: Updates in SQL

Prof. Dr. Stefan Brass

Martin-Luther-Universität Halle-Wittenberg

Sommersemester 2018/19

<http://www.informatik.uni-halle.de/~brass/dbp19/>

## Lernziele

Nach diesem Kapitel sollten Sie Folgendes können:

- die SQL-Kommandos `INSERT`, `UPDATE`, `DELETE`, sowie `COMMIT` und `ROLLBACK` verwenden.
- das Transaktions-Konzept erklären.

Typisches Beispiel nennen, ACID-Eigenschaften erklären.

# Inhalt

① Update-Kommandos in SQL

② Transaktionen

# Updates in SQL: Übersicht

- **SQL-Befehle zur Änderung des DB-Zustands:**
  - **INSERT:** Einfügung neuer Zeilen in eine Tabelle.
  - **DELETE:** Löschung von Zeilen aus einer Tabelle.
  - **UPDATE:** Änderung von Tabelleneinträgen (Werten in existierenden Zeilen).
- **SQL-Befehle zur Beendigung von Transaktionen:**
  - **COMMIT:** Erfolgreiches Ende der Transaktion, Änderungen des DB-Zustands werden dauerhaft.
  - **ROLLBACK:** Transaktion fehlgeschlagen, alle Änderungen rückgängig machen.

# Beispiel-Datenbank

## STUDENTEN

<u>SID</u>	VORNAME	NACHNAME	EMAIL
101	Lisa	Weiss	...
102	Michael	Grau	NULL
103	Daniel	Sommer	...
104	Iris	Winter	...

## BEWERTUNGEN

<u>SID</u>	<u>ATYP</u>	<u>ANR</u>	PUNKTE
101	H	1	10
101	H	2	8
101	Z	1	12
102	H	1	9
102	H	2	9
102	Z	1	10
103	H	1	5
103	Z	1	7

## AUFGABEN

<u>ATYP</u>	<u>ANR</u>	THEMA	MAXPT
H	1	ER	10
H	2	SQL	10
Z	1	SQL	14

# INSERT: Übersicht

- Der **INSERT**-Befehl hat zwei Formen:
  - Einfügung einer einzelnen Zeile mit neuen Daten.
  - Einfügung des Ergebnisses einer Anfrage.
- Die zweite Form kann z.B. benutzt werden, um eine Tabelle zu kopieren.

Die neue Tabelle (Kopie) muss allerdings vorher mit "CREATE TABLE" erstellt werden. Oracle erlaubt "CREATE TABLE ... AS SELECT ...".

- In SQL-92 gibt es nur einen allgemeinen **INSERT**-Befehl: "VALUES" (erste Form) und "SELECT" (zweite Form) sind beides Tabellenausdrücke.

## INSERT: Neue Werte (1)

- Beispiel:

```
INSERT INTO STUDENTEN  
VALUES (105, 'Nina', 'Brass', NULL);
```

- Mögliche Werte für die VALUES-Klausel sind:

- Terme, also insbesondere Konstanten, aber auch z.B. "100+5", "SYSDATE".

Die Terme können natürlich keine Attributreferenzen enthalten, da hier keine Tupelvariablen deklariert sind.

- Schlüsselworte "NULL", "DEFAULT".

Nach dem SQL-Standard ist "NULL" kein Term, deswegen muss es hier getrennt aufgeführt werden. "DEFAULT" meint den in der "CREATE TABLE"-Anweisung deklarierten Defaultwert für die Spalte.

## INSERT: Neue Werte (2)

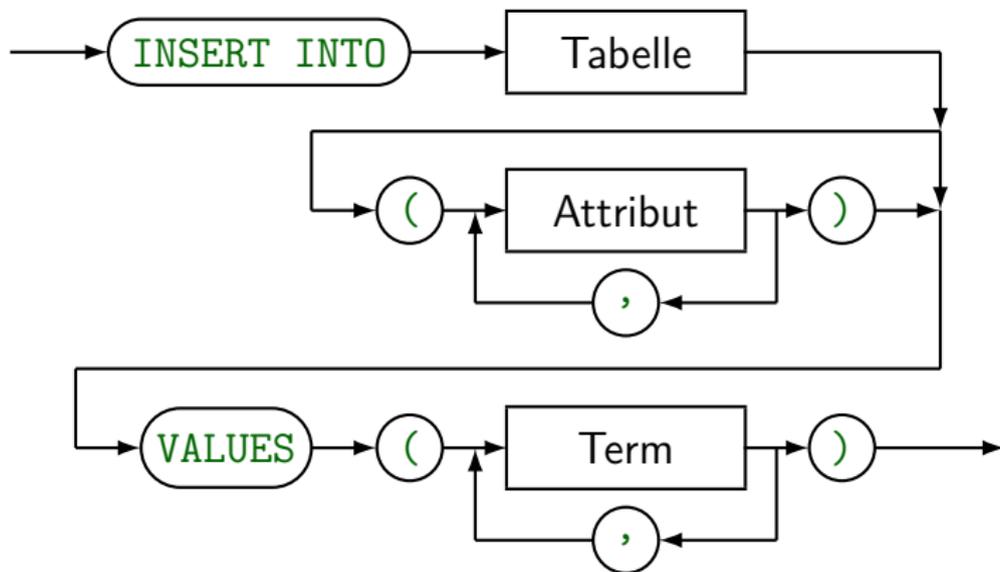
- Man kann Werte auch nur für eine Teilmenge der Spalten angeben:

```
INSERT INTO STUDENTEN(SID, VORNAME, NACHNAME)  
VALUES (105, 'Nina', 'Brass')
```

- In die übrigen Spalten (hier "EMAIL") wird der jeweilige Default-Wert eingetragen (hier Null).
- Die Spalten müssen nicht in der gleichen Reihenfolge wie in der Tabelle angegeben werden.

Daher ist diese Syntax auch bequem, wenn man zwar für alle Spalten einen Wert hat, aber sich nicht an die Reihenfolge der Spalten in der Tabelle erinnert. In einem Programm sollte man diese Syntax wählen, um Probleme durch später hinzugefügte Spalten zu vermeiden.

## INSERT: Neue Werte (3)



# INSERT: Mit Anfrage (1)

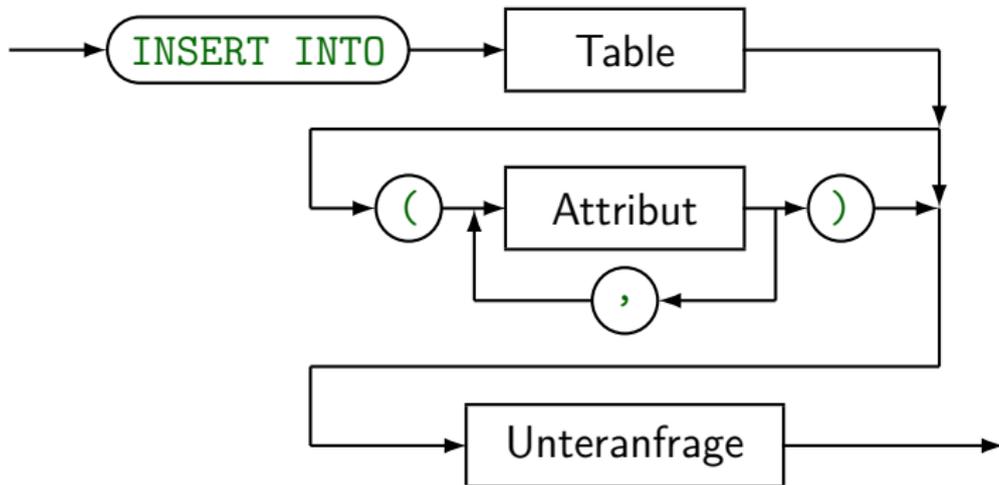
- Beispiel:

```
INSERT INTO KLAUSUREN(BEZ, ANR, THEMA, PROZENT)
SELECT 'DB-2005-E', A.ANO, A.THEMA,
      AVG(B.PUNKTE/A.MAXPT)*100
FROM   AUFGABEN A, BEWERTUNGEN B
WHERE  A.ATYP='E' AND B.ATYP='E' AND A.ANO=B.ANO
GROUP BY A.ANO, A.THEMA
```

- Die Unteranfrage wird vollständig ausgewertet bevor die Ergebnistupel eingefügt werden.

Daher gibt es auch dann ein definiertes Ergebnis (und niemals Endlosschleifen), wenn die Tabelle, in die eingefügt wird, in der Unteranfrage selbst verwendet wird.

## INSERT: Mit Anfrage (2)



Beachte: Hier steht die Unteranfrage ausnahmsweise nicht in (...).

# DELETE (1)

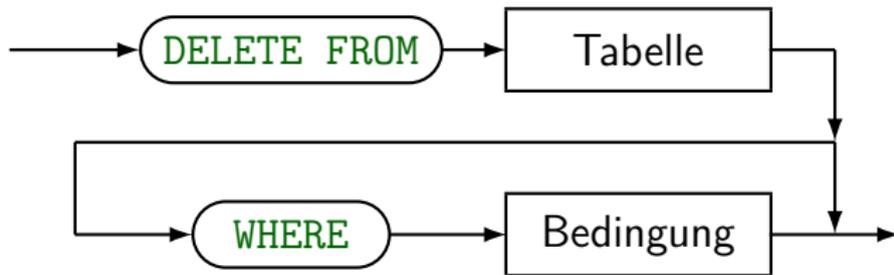
- Beispiel: Lösche alle Bewertungen für Lisa Weiss:

```
DELETE FROM BEWERTUNGEN
WHERE SID IN (SELECT SID
              FROM STUDENTEN
              WHERE VORNAME = 'Lisa'
              AND NACHNAME = 'Weiss')
```

- **Achtung:** Wenn man die WHERE-Bedingung weglässt, werden alle Tupel gelöscht!

Es ist eventuell möglich, "ROLLBACK" zu verwenden, wenn etwas schief gelaufen ist. Dafür muss man den Fehler aber bemerken, bevor die Transaktion beendet wird. Man sollte sich die Tabelle also nochmal anschauen. Manche SQL-Schnittstellen bestätigen jede Änderung sofort (`autocommit`), dann gibt es keine Möglichkeit mehr für ein Undo.

## DELETE (2)



# TRUNCATE (1)

- Oracle, SQL Server, und MySQL (aber nicht DB2 und Access) haben ein Kommando

```
TRUNCATE TABLE <Tabellenname>
```

Dies löscht alle Zeilen aus der Tabelle und gibt den von der Tabelle belegten Speicherplatz frei.

- Es ist ähnlich zu einem `DROP TABLE`, aber die Tabellendefinition (Schemainformation) bleibt erhalten, nur die Tabellendaten werden gelöscht.

Daher sind auch Verweise auf die Tabelle in Zugriffsrechten, Sichten, Triggern, gespeicherten Prozeduren, etc. nicht betroffen.

## TRUNCATE (2)

- Im Gegensatz zu DELETE, kann TRUNCATE nicht mit ROLLBACK zurückgenommen werden.
- Dafür ist es viel schneller.

Zumindest in Oracle würde DELETE auch nicht wirklich Speicherplatz freigeben, er bleibt für die Tabelle reserviert.

Ein anderes Problem ist, dass wenn man alle Zeilen einer großen Tabelle mit DELETE löschen will, das Rollback Segment (Speicherplatz für Undo-Information in Oracle) eventuell zu klein ist. Dann liefert "DELETE FROM <Table Name>" eine Fehlermeldung und nichts wird gelöscht. Bei TRUNCATE kann das nicht passieren.

- TRUNCATE gehört nicht zum SQL-92 Standard.

Aber es tritt in der Oracle Zertifizierungs-Prüfung auf.

# UPDATE (1)

- Das **UPDATE**-Kommando dient zur Änderung von Attributwerten ausgewählter Tupel.
- Z.B. soll ein Zusatzpunkt für alle Lösungen von Aufgabe 1 in der Zwischenklausur vergeben werden:

```
UPDATE BEWERTUNGEN
SET     PUNKTE = PUNKTE + 1
WHERE  ATYP = 'Z' AND ANO = 1
```

- Die rechte Seite der Zuweisung kann die alten Werte aller Attribute des aktuellen Tupels verwenden.

Die **WHERE**-Bedingung und die Terme auf der rechten Seite der Zuweisung werden ausgewertet, bevor ein Update wirklich durchgeführt wird. Als neuer Attributwert ist auch **NULL** erlaubt.

## UPDATE (2)

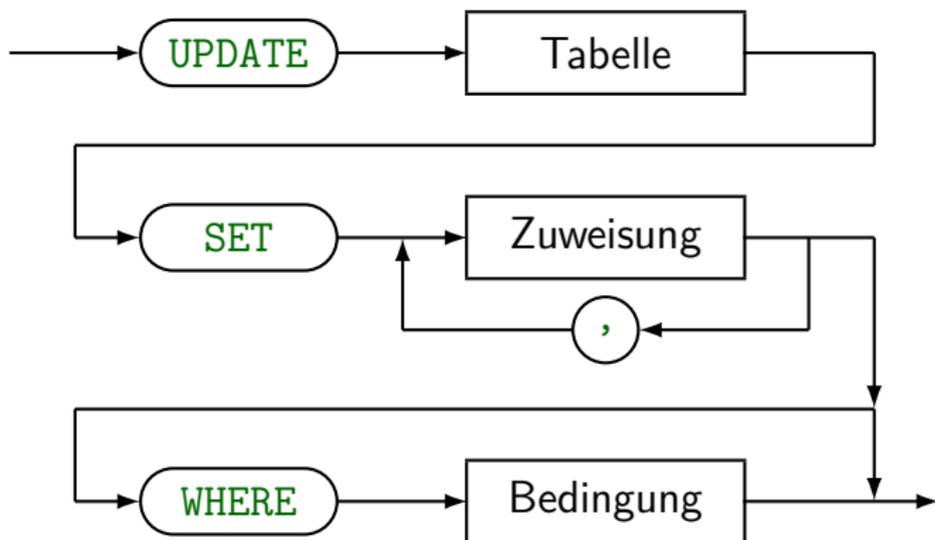
- In SQL-92, Oracle, DB2, SQL Server (aber nicht in SQL-86, MySQL, Access), kann der neue Attributwert mit einer Unteranfrage berechnet werden.

Die Unteranfrage darf nicht mehr als eine Zeile liefern (mit einer Spalte).  
Falls sie keine Zeile liefert, wird ein Nullwert verwendet.

- Man kann in einer UPDATE-Anweisung auch mehrere Attribute ändern:

```
UPDATE AUFGABEN
SET   THEMA = 'Einfaches SQL',
      MAXPT = 8
WHERE ATYP = 'H' AND ANO = 1
```

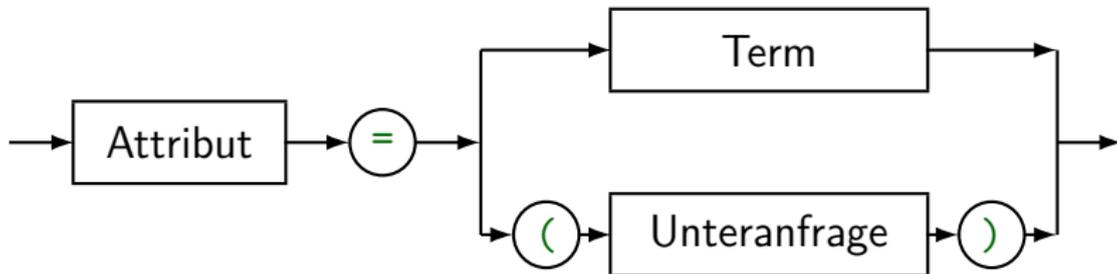
# UPDATE (3)



- In MySQL werden Zuweisungen in der gegebenen Reihenfolge abgearbeitet: Neue Werte weiter links stehender Zuweisungen sind schon sichtbar.

# UPDATE (4)

Zuweisung:



- SQL-86, MySQL, und Access erlauben keine Unteranfragen auf der rechten Seite. MySQL erlaubt mehrere Tabellen nach UPDATE (auch einen Join), im SQL-2003 Standard ist das nicht vorgesehen.
- In SQL-92, DB2 und SQL-Server kann eine Unteranfrage ohnehin als Term genutzt werden, daher ist der zweite Fall eigentlich ein Spezialfall des ersten. Nur für Oracle 8 muss die Unteranfrage explizit genannt werden.

# MERGE (1)

- Im SQL-2003 Standard wurde eine neue Anweisung **MERGE** eingeführt, die **INSERT** und **UPDATE** kombiniert.

Es gibt die MERGE-Anweisung z.B. in Oracle 10g, DB2 Ver. 9, SQL Server 2008. Jedes System hat andere Erweiterungen zum Standard.

- Sie wird meistens verwendet, wenn man eine Menge von Änderungen zu einer Tabelle in einer anderen Tabelle gesammelt hat.
- Damit kann man z.B. eine Tabelle mit den Daten in einer anderen Tabelle synchronisieren, wie es besonders im Data Warehouse Bereich benötigt wird.

## MERGE (2)

- **Beispiel:** Ein Tutor trägt seine Bewertungen für Hausaufgaben in folgende Tabelle ein:

`GRUPPE1 (ANR, SID, PUNKTE)`

- Von Zeit zu Zeit müssen die Punkte in die Haupt-Tabelle `BEWERTUNGEN` übernommen werden.
- Es ist auch möglich, dass der Tutor in seiner Tabelle Punkte nachträglich verändert hat (nachdem die Punkte bereits einmal übernommen wurden).

Wenn Sie eine Bewertung nicht verstehen, oder für falsch halten, fragen Sie nach. Auch Tutoren, Mitarbeiter und Professoren machen gelegentlich Fehler.

## MERGE (3)

- Befehl zur Übernahme der Daten:

```
MERGE INTO BEWERTUNGEN B
      USING GRUPPE1 G
      ON (B.SID = G.SID AND
          B.ANR = G.ANR AND B.ATYP = 'H')
      WHEN MATCHED THEN
          UPDATE SET B.PUNKTE = G.PUNKTE
      WHEN NOT MATCHED THEN
          INSERT(SID, ATYP, ANR, PUNKTE)
          VALUES(G.SID, 'H', G.ANR, G.PUNKTE)
```

Eine Zeile der Zieltabelle (BEWERTUNGEN) kann höchstens einen Join-Partner in der Quelltable (GRUPPE1) haben. Sonst wäre der Update nicht eindeutig definiert und das MERGE scheitert.

## MERGE (4)

- Das entsprechende klassische INSERT wäre einfach, aber der UPDATE-Teil wäre umständlich:

```
UPDATE BEWERTUNGEN
SET     PUNKTE = (SELECT PUNKTE
                  FROM   GRUPPE1 G
                  WHERE  G.SID = BEWERTUNGEN.SID
                  AND    G.ANR = BEWERTUNGEN.ANR
                  AND    G.ATYP = 'H')
WHERE  EXISTS (SELECT *
              FROM  GRUPPE1 G
              WHERE G.SID = BEWERTUNGEN.SID
              AND   G.ANR = BEWERTUNGEN.ANR
              AND   G.ATYP = 'H')
```

# Inhalt

① Update-Kommandos in SQL

② Transaktionen

## Transaktionen (1)

- **Transaktion: Folge von DB-Kommandos, insbesondere Updates, die das DBMS als Einheit behandelt.**
- Z.B. besteht eine Überweisung von 50 Euro von Konto 11 auf Konto 23 aus folgenden Schritten:
  - Prüfung von Kontostand und Kreditrahmen von Konto 11,
  - Reduktion des Stands von Konto 11 um 50 Euro,
  - Erhöhung des Stands von Konto 23 um 50 Euro,
  - Schreiben von Einträgen in die Kontoauszüge beider Konten (plus ggf. Protokoll des Arbeitsplatzes).

## Transaktionen (2)

### ACID-Merkregel für Eigenschaften von Transaktionen:

- Atomarität (“Atomicity”)

Eine Transaktion wird “ganz oder garnicht” ausgeführt.

- Konsistenz (“Consistency”)

Eine Transaktion führt von einem konsistenten in einen konsistenten DB-Zustand.

- Isolation (“Isolation”)

Transaktionen paralleler Benutzer stören sich nicht gegenseitig.

- Dauerhaftigkeit (“Durability”)

Wenn eine Transaktion erfolgreich mit COMMIT abgeschlossen wurde, sind ihre Daten sicher gespeichert.

## Transaktionen (3)

### Atomarität:

- Moderne DBMS garantieren, dass eine Transaktion
  - entweder vollständig ausgeführt wird,
  - oder keinerlei Spuren hinterlässt (“alles oder nichts”-Prinzip).
- Kann eine Transaktion nicht zu Ende ausgeführt werden (z.B. wegen Stromausfall), so wird der Zustand vor Beginn der Transaktion beim nächsten Hochfahren des Systems wieder hergestellt.

## Transaktionen (4)

- **Atomarität gibt eine Undo-Möglichkeit:**
  - Solange die Transaktion nicht als vollständig deklariert wurde (mit `COMMIT`), können alle Änderungen zurückgenommen werden (mit `ROLLBACK`).
  - In den meisten DBMS kann man aber nur die ganze Transaktion zurücknehmen (nicht nur das letzte Kommando).

Allerdings kann man z.B. in Oracle und SQL Server "savepoints" innerhalb einer Transaktion setzen, und bei Bedarf auf den so benannten Zustand zurücksetzen.
  - Nach dem `COMMIT` ist kein Undo mehr möglich.

# Transaktionen (5)

## Dauerhaftigkeit:

- Wenn das DBMS das erfolgreiche Ende einer Transaktion bestätigt, sind die Änderungen dauerhaft.
- Die Daten sind dann auf einer Platte gespeichert — sie sind nicht verloren, selbst wenn eine Sekunde später der Strom ausfällt.

Bei Betriebssystemen weiß man dagegen oft nicht genau ob die Daten schon auf der Platte oder noch in einem Puffer sind.

- Mächtige Backup&Recovery-Mechanismen: selbst wenn eine Platte ausfällt, sind keine Daten verloren.

Auf Betriebssystem-Ebene dagegen nur ein Backup pro Tag normal.

## Transaktionen (6)

- Atomarität und Dauerhaftigkeit zusammen bedeuten, dass es **einen Zeitpunkt** gibt, an dem **alle Änderungen schlagartig dauerhaft werden**.

Stürzt das System vor diesem Zeitpunkt ab, erhält man den alten DB-Zustand (vor der Transaktion). Stürzt es danach ab, erhält man den neuen Zustand (mit allen Änderungen der Transaktion).

- Der Zeitpunkt liegt zwischen
  - dem Abschicken des COMMIT-Kommandos an das DBMS (Benutzer: "Transaktion fertig")
  - und der Mitteilung des Systems, dass das COMMIT erfolgreich ausgeführt wurde.

# Transaktionen (7)

## Isolation:

- Alle größeren DBMS erlauben gleichzeitige Zugriffe mehrerer Benutzer.
- Ohne Kontrolle könnte dies zum Verlust von Daten führen, und zur Zerstörung der Konsistenz der DB.
- Das DBMS versucht aber die Transaktionen von einander zu isolieren: Jeder Benutzer soll den Eindruck haben, dass seine Transaktion exklusiven Zugriff auf die ganze Datenbank hat.

Die meisten DBMS verwalten dazu automatisch (intern) Sperren auf DB-Objekten (z.B. Tabellen, Tupeln): s.u.

# Transaktionen (8)

## Konsistenz:

- Benutzer und System können sicher sein, dass der aktuelle Zustand das Ergebnis einer Folge von vollständig ausgeführten Transaktionen ist.
- Der Benutzer muss sicherstellen, dass jede Transaktion, wenn sie vollständig und isoliert (einzeln) auf einen konsistenten Zustand angewendet wird, auch wieder einen konsistenten Zustand produziert.

Ein Zustand heißt konsistent, wenn er alle Integritätsbedingungen erfüllt. Moderne DBMS bieten Unterstützung dafür an: Schlüssel, Fremdschlüssel, NOT NULL und CHECK-Bedingungen können deklarativ spezifiziert werden. Für komplexere Bedingungen gibt es Trigger.

## Transaktionen (9)

- Die Konsistenz ist zum Teil eine Folge der anderen drei Eigenschaften und zum Teil etwas, was der Benutzer garantieren muss.
- Konsistenz ist besonders auch für komplexe/redundante Datenstrukturen wichtig.

Wenn Benutzer redundante Daten speichern, müssen sie diese Daten in derselben Transaktion aktualisieren, die auch die Originaldaten modifiziert. Dann stellt aber das System sicher, dass selbst bei einem Stromausfall zwischen den Befehlen die beiden Kopien niemals auseinander laufen. Dies betrifft auch die internen Datenstrukturen des DBMS, z.B. Indexe (redundante Datenstrukturen, um Zeilen mit gegebenen Attributwerten schnell zu finden). Würden manche Zeilen im Index fehlen, wäre das Systemverhalten unvorhersehbar.

# Transaktions-Verwaltung (1)

- SQL hat kein Kommando, um den Beginn einer Transaktion zu markieren.

Eine Transaktion beginnt automatisch, wenn man sich beim DBMS anmeldet, und jedes Mal, nachdem eine Transaktion beendet wurde.

- Eine Transaktion wird beendet mit
  - `COMMIT [WORK]`: Macht Änderungen dauerhaft.
  - `ROLLBACK [WORK]`: Nimmt Änderungen zurück.
- Manche Kommandos, wie etwa `DROP TABLE`, lösen zumindest in Oracle automatisch ein `COMMIT` aus.

Solche Kommandos können daher nicht zurückgenommen werden. Dies betrifft auch vorangegangene, noch nicht bestätigte Updates.

## Transaktions-Verwaltung (2)

- Manche Systeme haben einen “Autocommit Modus”:  
Dann wird ein COMMIT automatisch nach jedem Update durchgeführt (dann gibt es keine Undo-Möglichkeit mehr!).

In Oracle SQL\*Plus kann man diesen Modus mit “`set autocommit on`” auswählen (defaultmäßig ist der Autocommit Modus ausgeschaltet).

SQL Server läuft normalerweise im Autocommit Modus, aber das Kommando “`BEGIN TRANSACTION`” schaltet diesen Modus aus.

In DB2 funktionieren COMMIT und ROLLBACK normal.

MySQL hat nur den Autocommit Modus, außer wenn man einen speziellen Tabellentyp verwendet, der Transaktionen unterstützt.

Access bestätigt ebenfalls alle Änderungen automatisch und versteht die Kommandos COMMIT und ROLLBACK nicht.

## Transaktions-Verwaltung (3)

- Wenn man SQL\*Plus (SQL-Interpreter von Oracle) normal verlässt (mit `QUIT` oder `EXIT`), findet automatisch ein `COMMIT` statt.
- Wenn man dagegen einfach das Fenster schließt, oder sich beim Betriebssystem abmeldet, findet ein `ROLLBACK` statt.

Obwohl man die geänderten Daten vorher gesehen hat, sind sie bei der nächsten Sitzung verschwunden.

- Es ist also besser, explizit `COMMIT` einzugeben.

Das gilt auch für SQL-Skripte.

## Transaktions-Verwaltung (4)

- Wenn man mit der Datenbank für eine längere Zeit arbeitet, sollte man die Änderungen von Zeit zu Zeit mit COMMIT bestätigen.

Falls es zu einem Stromausfall etc. kommen sollte, sind so nur die Änderungen nach dem letzten COMMIT verloren. Außerdem sperrt das DBMS typischerweise von der Transaktion veränderte Zeilen, eventuell auch ganze Blöcke auf der Platte. Diese Sperren bleiben bis zum Ende der Transaktion erhalten. Lange Transaktionen können dann andere Benutzer behindern. Schließlich muss das DBMS für die Dauer der Transaktion Undo-Information aufbewahren. Wenn es die Speicherbereiche zyklisch neu verwendet, kann das auch zu Problemen führen. Klassische Datenbanksysteme sind nicht für lange Transaktionen gedacht.

# Literatur/Quellen

- Elmasri/Navathe: Fundamentals of Database Systems, 3rd Edition, 1999. Chap. 8, "SQL — The Relational Database Standard"
- Kemper/Eickler: Datenbanksysteme (in German), 4th Ed., Oldenbourg, 1997. Chapter 4: Relationale Anfragesprachen (Relational Query Languages).
- Lipeck: Skript zur Vorlesung Datenbanksysteme (in German), Univ. Hannover, 1996.
- Date/Darwen: A Guide to the SQL Standard, Fourth Edition, Addison-Wesley, 1997.
- van der Lans: SQL, Der ISO-Standard (in German), Hanser, 1990.
- Sunderraman: Oracle Programming, A Primer. Addison-Wesley, 1999.
- Oracle8 SQL Reference, Oracle Corporation, 1997, Part No. A58225-01.
- Oracle8 Concepts, Release 8.0, Oracle Corporation, 1997, Part No. A58227-01.
- Chamberlin: A Complete Guide to DB2 Universal Database. Morgan Kaufmann, 1998.
- Microsoft SQL Server Books Online: Accessing and Changing Data.
- H. Berenson, P. Bernstein, J. Gray, J. Melton, E. O'Neil, P. O'Neil: A critique of ANSI SQL isolation levels. In Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, 1–10, 1995.