Einführung in Datenbanken

Kapitel 20: Änderungs-Operationen (Updates) in SQL

Prof. Dr. Stefan Brass

Martin-Luther-Universität Halle-Wittenberg

Wintersemester 2020/21

http://www.informatik.uni-halle.de/~brass/db20/

Lernziele

Nach diesem Kapitel sollten Sie Folgendes können:

- die SQL-Kommandos `INSERT`, `UPDATE`, `DELETE`, sowie `COMMIT` und `ROLLBACK` verwenden.
- das Transaktions-Konzept (inklusive der ACID-Eigenschaften) erklären und bei der Nutzung eines DBMS berücksichtigen.

  Sie sollten auch ein typisches Beispiel für Transaktionen nennen können.
- den Autocommit-Modus vieler Datenbanken erklären, und `BEGIN TRANSACTION` in PostgreSQL verwenden.

Beispiel-Datenbank

STUDENTEN

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

AUFGABEN

ATYP	ANR	THEMA	MAXPT
H	1	ER	10
H	2	SQL	10
Z	1	SQL	14

BEWERTUNGEN

SID	ATYP	ANR	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

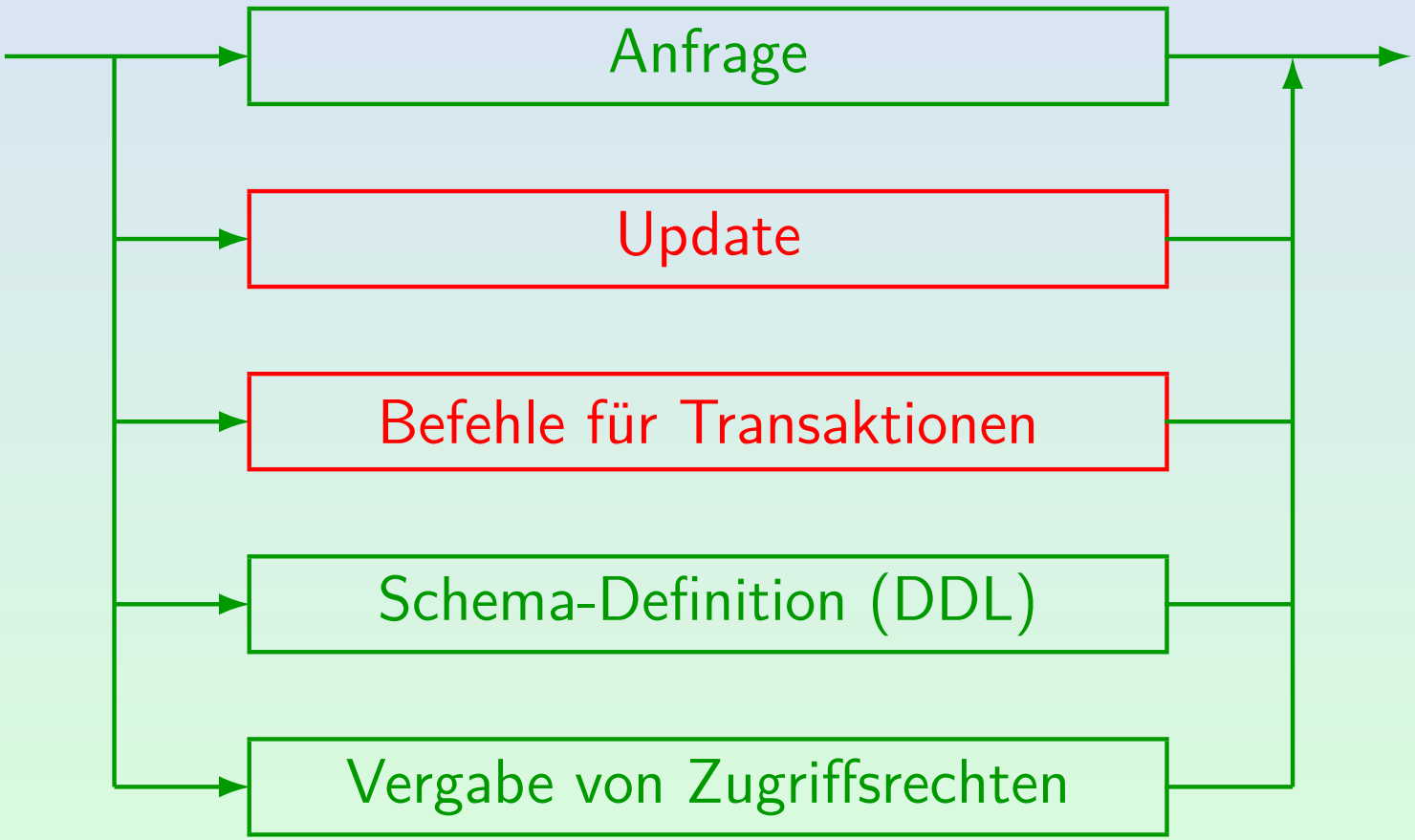
Inhalt

Updates in SQL: Übersicht (1)

- 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.

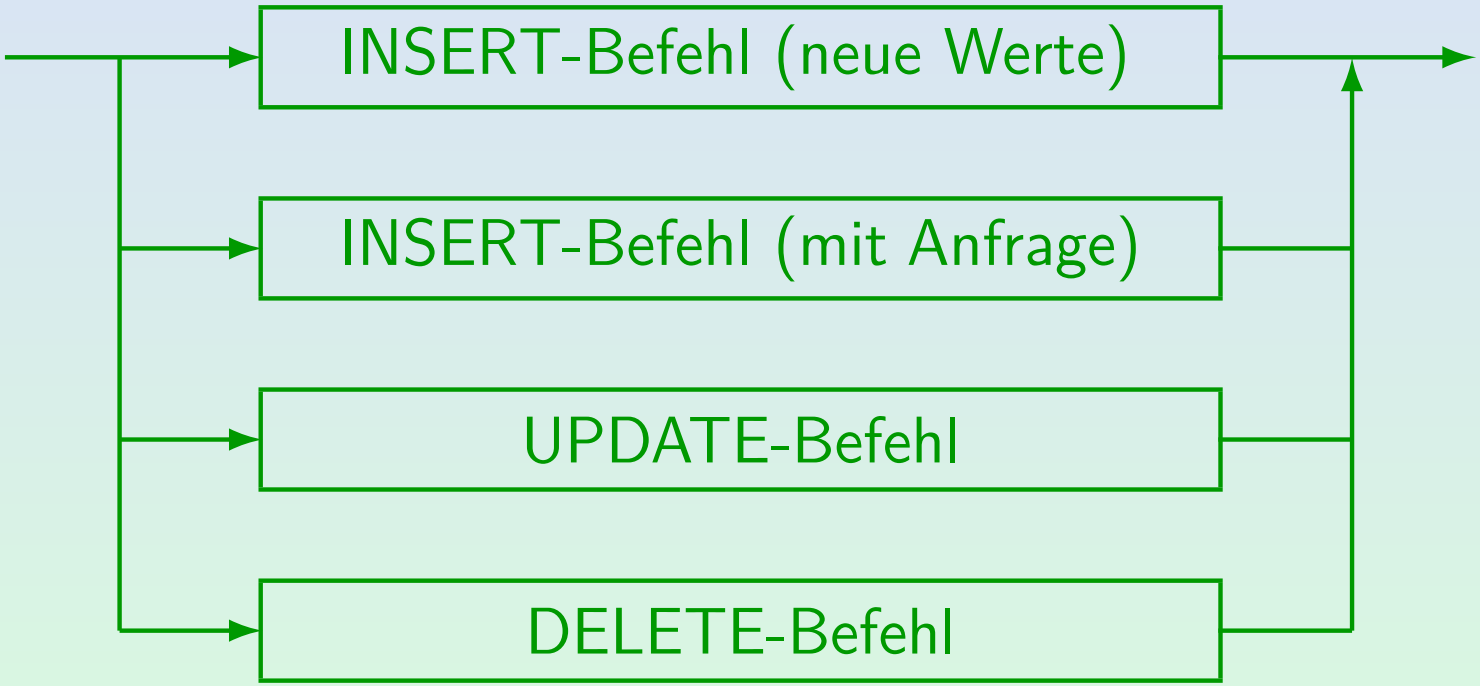
Updates in SQL: Übersicht (2)

SQL-Befehl (Auswahl):



Updates in SQL: Übersicht (3)

Update:



- In der Fortsetzungs-Vorlesung im Sommersemester werden noch `TRUNCATE`, `MERGE` und der PostgreSQL-spezifische Befehl `COPY` besprochen.

Inhalt

INSERT: Übersicht

- Der `INSERT`-Befehl hat zwei Formen:
  - Einfügung einer 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. Einige DBMS (z.B. Oracle, PostgreSQL) erlauben „`CREATE TABLE ... AS SELECT ...`“.
- In SQL-92 gibt es nur einen allgemeinen `INSERT`-Befehl: „`VALUES`“ (erste Form) und „`SELECT`“ (zweite Form) sind beides eine „query expression“ (u.a. in Unteranfragen).

  In PostgreSQL, DB2, HSQLDB geht: `SELECT * FROM (VALUES (1,2)) X`

  In Oracle, MySQL, SQLite3 nicht. In MS SQL Server sind Spaltennamen Pflicht.

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, CURRENT_DATE, CURRENT_TIMESTAMP, CURRENT_USER.

    Die Terme können natürlich keine Attributreferenzen enthalten, da hier keine Tupelvariablen deklariert sind. In Oracle heißt es SYSDATE statt CURRENT_TIMESTAMP und USER statt CURRENT_USER.
  - Schlüsselworte NULL und DEFAULT.

    Nach dem SQL-Standard ist „NULL“ kein Term, deswegen muss dieses Schlüsselwort hier getrennt aufgeführt werden. „DEFAULT“ meint den im „CREATE TABLE“ deklarierten Defaultwert für die Spalte.

INSERT: Neue Werte (2)

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

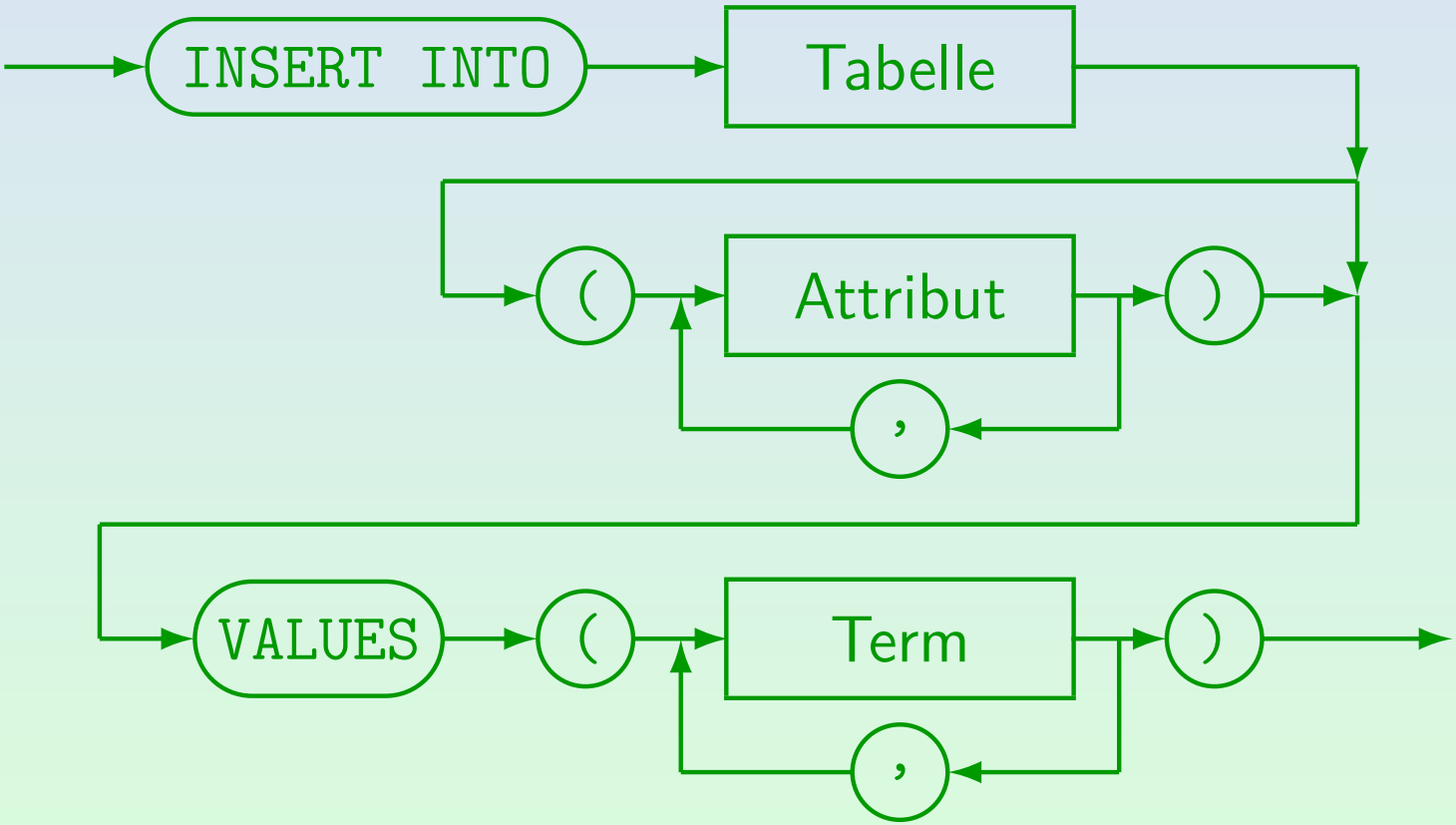
```
INSERT INTO STUDENTEN(SID, VORNAME, NACHNAME)
VALUES (105, 'Nina', 'Brass')
```

- 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-Befehl (neue Werte):



INSERT: Neue Werte (4)

- Der SQL-Standard (seit SQL-92) und viele Systeme (z.B. PostgreSQL) erlauben, dass beim `INSERT` mehrere Tupel angegeben werden:

```
INSERT INTO STUDENTEN VALUES
   (101, 'Lisa',    'Weiss',  'weiss@acm.org'),
   (102, 'Michael', 'Grau',   NULL),
   (103, 'Daniel',  'Sommer', 'daniel@gmx.de'),
   (104, 'Iris',    'Winter', 'irisw@gmail.com')
```

- In Oracle ist dies ein Syntax-Fehler.

  Wenn man also SQL-Skripte zum Anlegen von Datenbanken verbreiten will, ist es portabler, jeweils ein „`INSERT INTO ... VALUES ...`“ pro Tabellenzeile zu schreiben. Allerdings verstehen MySQL (MariaDB), MS SQL Server, DB2, SQLite3 und HSQLDB die obige Syntax. Oracle ist also eine Ausnahme.

INSERT: Mit Anfrage (1)

- Beispiel:

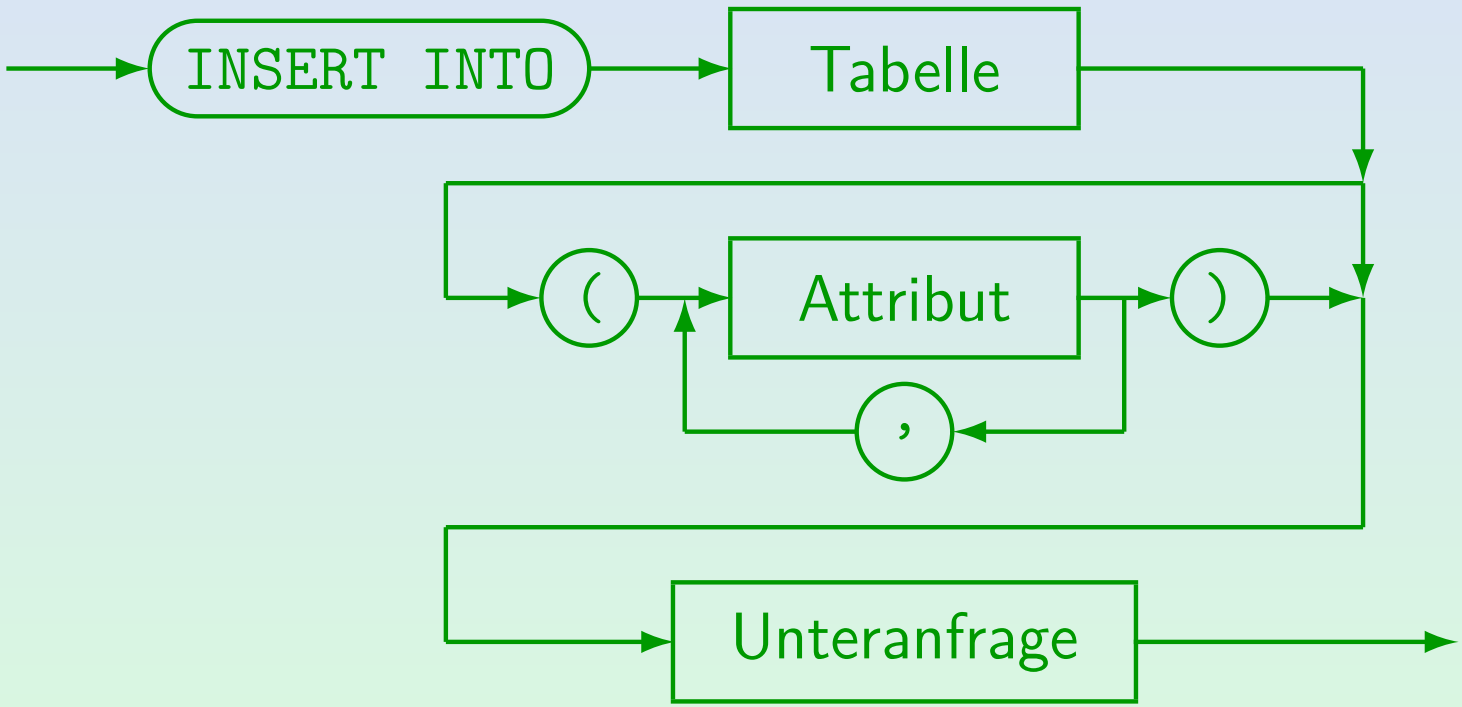
```
INSERT INTO KLAUSUREN(BEZ, ANR, THEMA, PROZENT)
SELECT 'DB-2005-E', A.ANO, A.THEMA,
       AVG(B.PUNKTE/A.MAXPT)*100
FROM   AUFGABEN A, BEWERTUNGEN B
WHERE  A.ATYP='E' AND B.ATYP='E' AND A.ANO=B.ANO
GROUP  BY A.ANO, A.THEMA
```

- 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)

INSERT-Befehl (mit Anfrage):



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

Inhalt

DELETE (1)

- Beispiel: Lösche alle Bewertungen für Lisa Weiss:

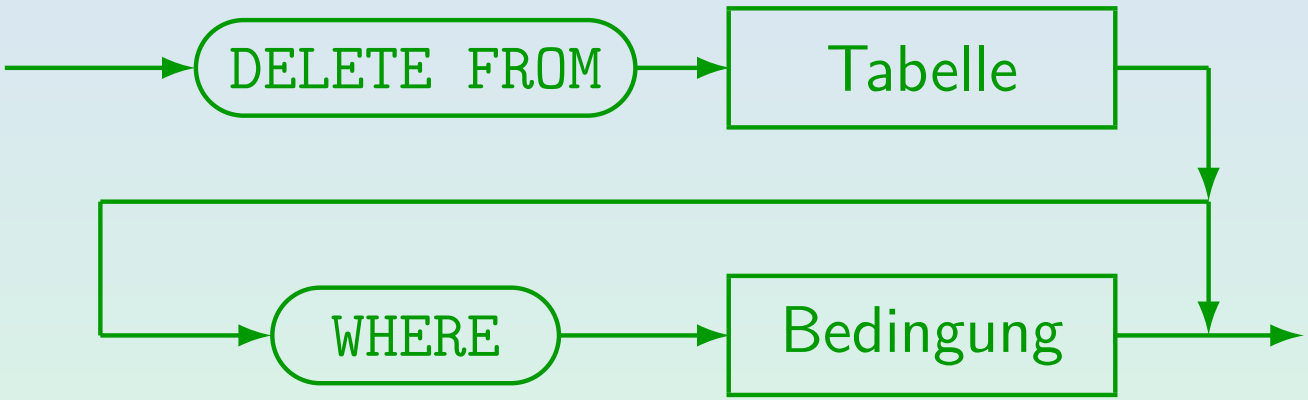
```
DELETE FROM BEWERTUNGEN
WHERE  SID IN (SELECT SID
               FROM   STUDENTEN
               WHERE  VORNAME  = 'Lisa'
               AND    NACHNAME = 'Weiss')
```

- 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)

DELETE-Befehl:



Inhalt

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

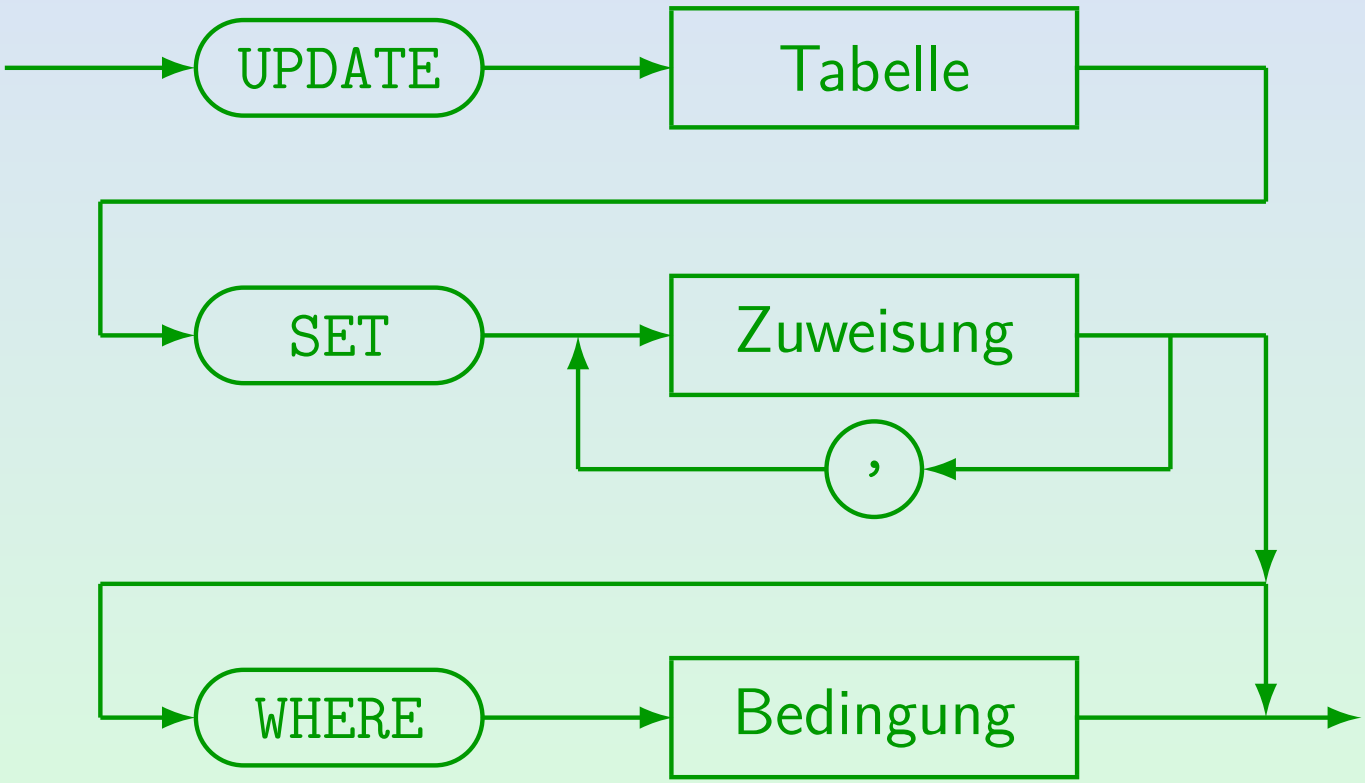
  Die Unteranfrage darf nicht mehr als eine Zeile liefern (mit einer Spalte). Falls sie keine Zeile liefert, wird ein Nullwert verwendet. In der Unteranfrage kann man auf den Tabellennamen außen als Tupelvariable zugreifen — das Ergebnis der Unteranfrage kann also davon abhängen, welche Zeile aktualisiert wird.

- 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)

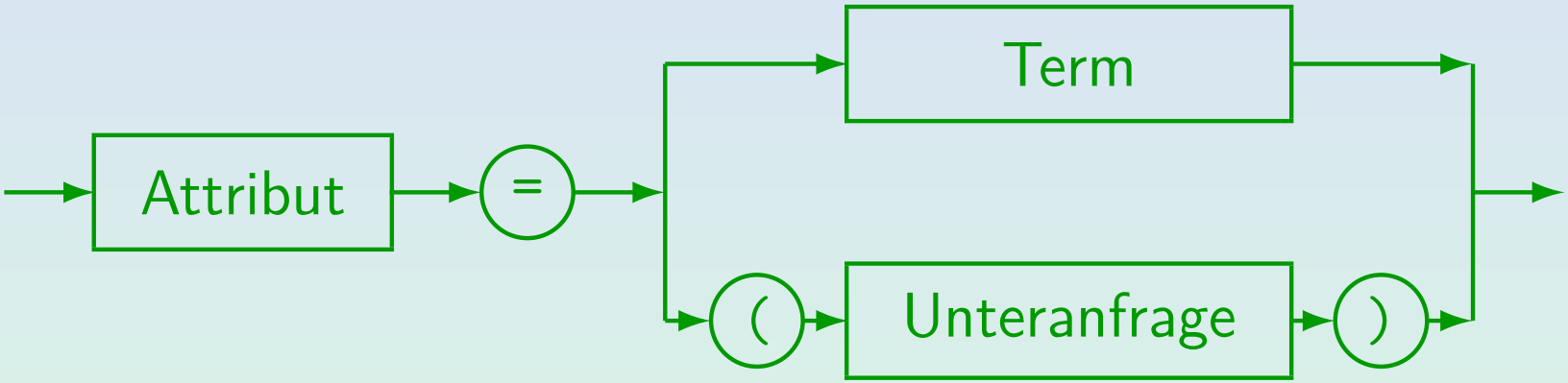
UPDATE-Befehl:



- 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 musste die Unteranfrage explizit genannt werden.

Inhalt

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 Tranaktion 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.
- Datenbank-Managementsysteme bieten meist mächtige Backup&Recovery-Mechanismen: selbst wenn eine Platte ausfällt, sind keine Daten verloren.

  Auf Betriebssystem-Ebene ist 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).

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.

Inhalt

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.

- In PostgreSQL schreibt man „`START TRANSACTION`“, um den Autocommit-Modus auszuschalten.

  Im Standard seit SQL:2009. PostgreSQL versteht auch „`BEGIN TRANSACTION`“.

Transaktions-Verwaltung (3)

- Wenn es kein Autocommit gibt, und man beendet das Programm (z.B. `psql` bei PostgreSQL) ohne `COMMIT`, findet automatisch ein `ROLLBACK` statt.

  Obwohl man die geänderten Daten vorher gesehen hat, sind sie bei der nächsten Sitzung verschwunden. Das ist insofern ok, weil man ja bewusst eine Transaktion geöffnet hat. Dann muss man sie auch explizit schließen.

- Wenn man dagegen bei Oracle SQL*Plus (entspricht `psql`) normal verlässt (mit `QUIT` oder `EXIT`), findet automatisch ein `COMMIT` statt.

  Dort öffnet man die Transaktion nicht explizit. Wenn man dagegen einfach das Fenster schließt, findet ein `ROLLBACK` statt. Konsequenz: s.o.

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

  Das gilt auch für SQL-Skripte.

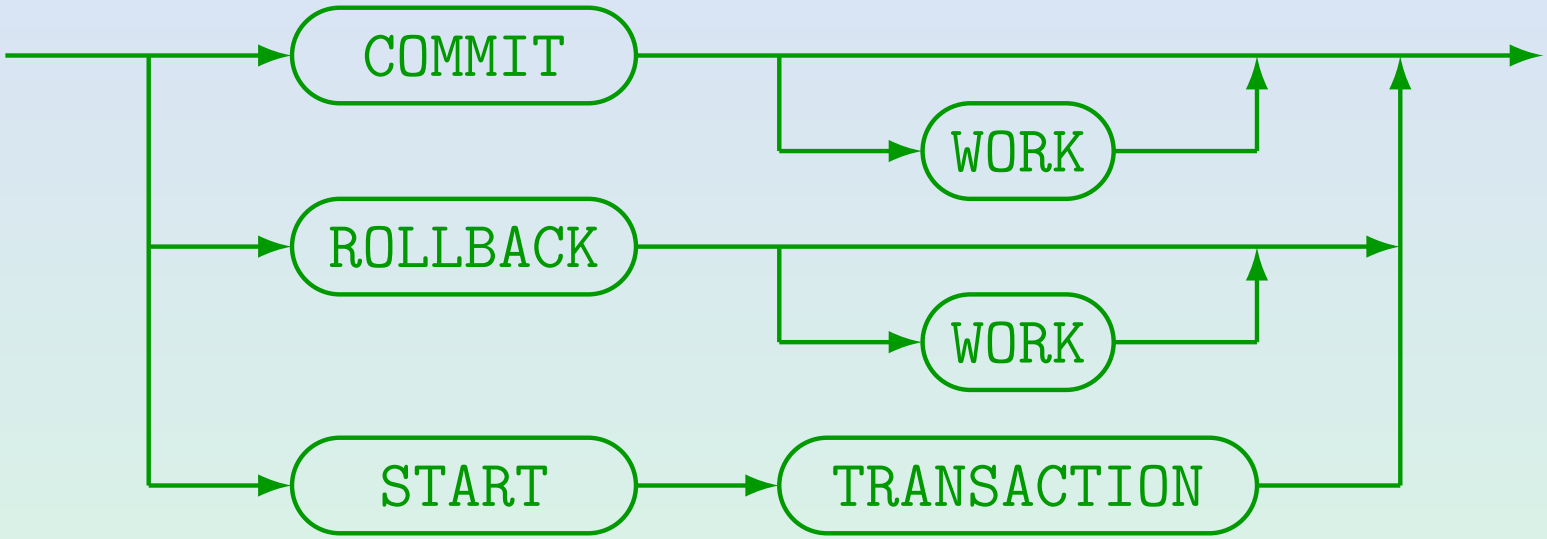
Transaktions-Verwaltung (4)

- 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.

Transaktions-Verwaltung (5)

Befehle für Transaktionen:



- `START TRANSACTION` ist wenig portabel, es wird von PostgreSQL und HSQLSB verstanden (und MySQL/MariaDB, dort scheint es aber keine Wirkung zu haben). Microsoft SQL Server und SQLite erwarten stattdessen „`BEGIN TRANSACTION`“.
- Oracle und DB2 folgen dem klassischen Modell, dass der Anfang einer Transaktion nicht speziell markiert werden muss. Sie verstehen weder `START TRANSACTION` noch `BEGIN TRANSACTION`. Bei DB2 ist der Default der Autocommit-Modus, Lösung: `UPDATE COMMAND OPTIONS USING c OFF` (oder +c beim Aufruf).

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.