Datenbank-Programmierung

Kapitel 3: Updates in SQL

Prof. Dr. Stefan Brass

Martin-Luther-Universität Halle-Wittenberg

Sommersemester 2022

http://www.informatik.uni-halle.de/~brass/dbp22/

Lernziele

Nach diesem Kapitel sollten Sie Folgendes können:

- die SQL-Kommandos `INSERT`, `UPDATE`, `DELETE`, sowie `COMMIT` und `ROLLBACK` verwenden.
- beschreiben, wofür `TRUNCATE` und `MERGE` zu verwenden sind.
- erklären, wie man eine CSV-Datei in PostgreSQL laden kann.

  Und was überhaupt eine CSV-Datei ist.
- das Transaktions-Konzept erklären.

  Typisches Beispiel nennen, ACID-Eigenschaften erklären.

Beispiel-Datenbank

STUDENTEN

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

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

AUFGABEN

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

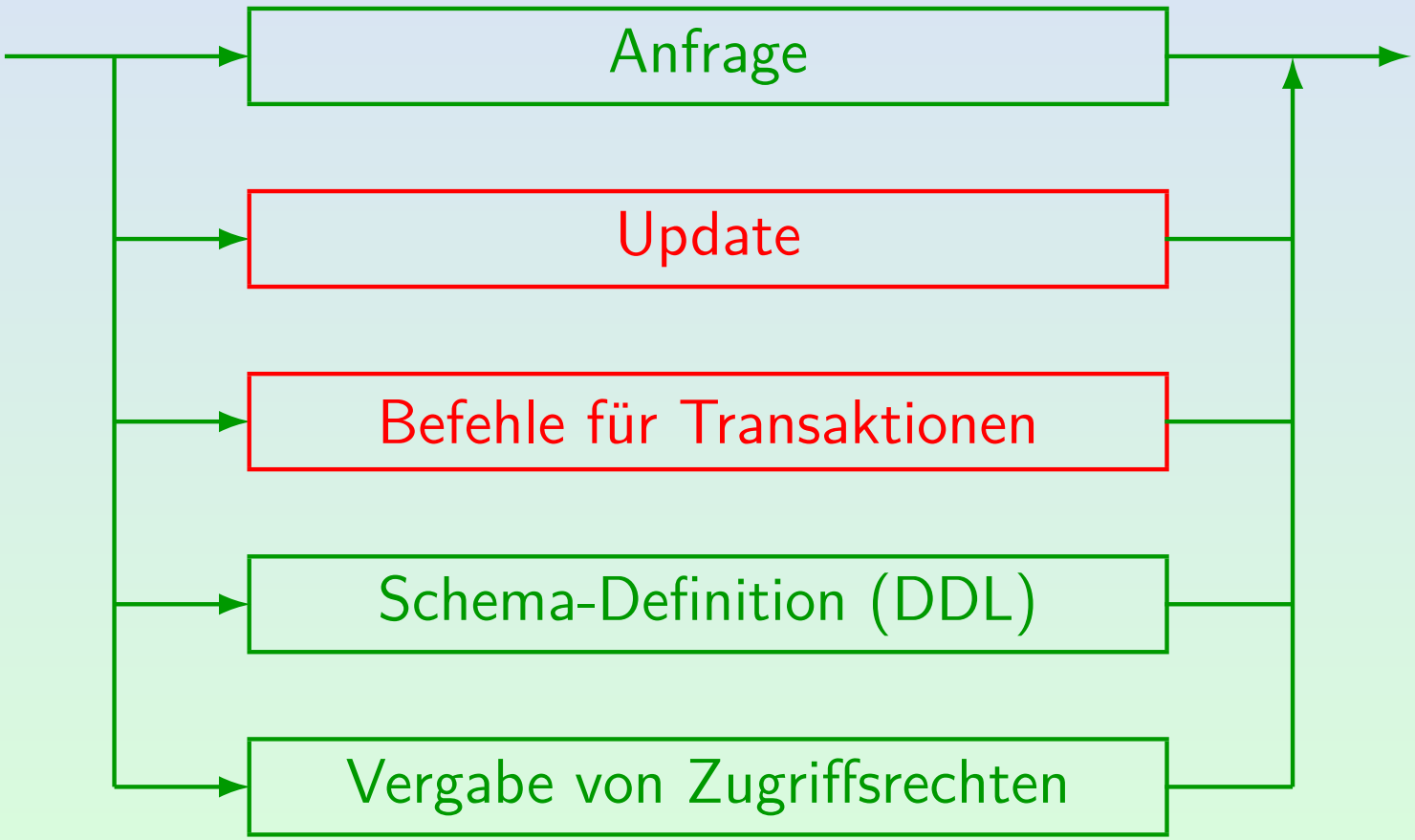
Inhalt

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.

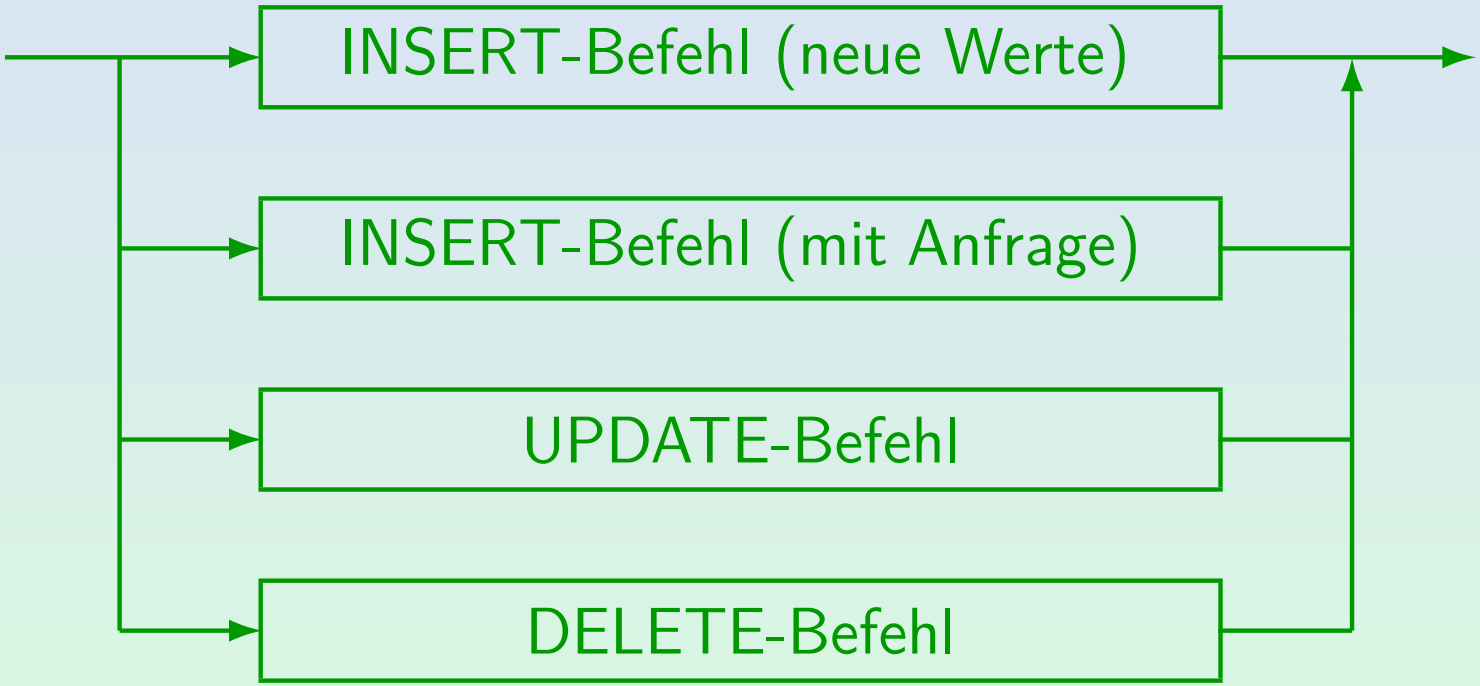
Updates in SQL: Übersicht (2)

SQL-Befehl (Auswahl):



Updates in SQL: Übersicht (3)

Update:



- In dieser Vorlesung werden außerdem noch `TRUNCATE`, `MERGE` und der PostgreSQL-spezifische Befehl `COPY` besprochen.

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. 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 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 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:

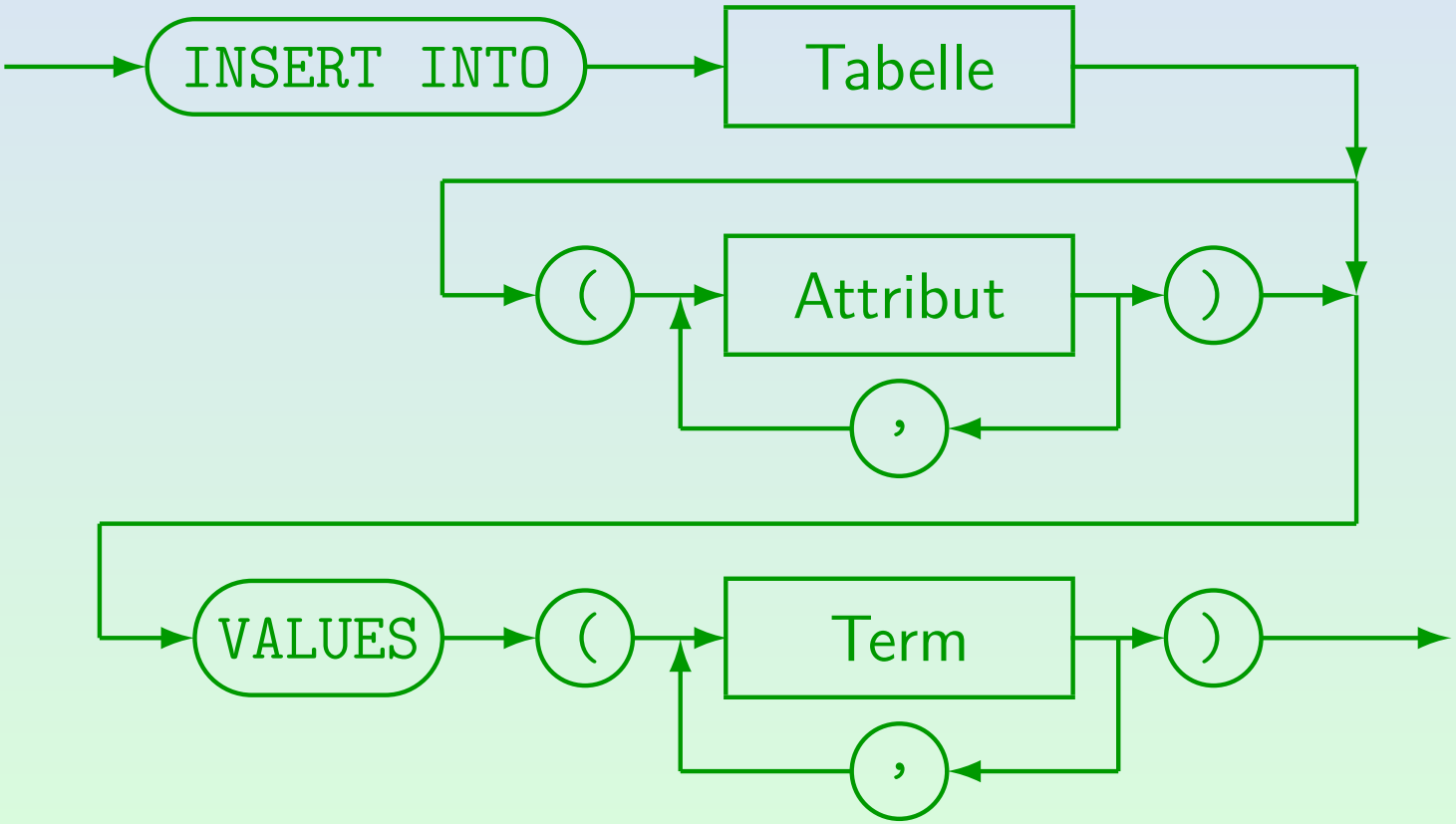
```
INSERT INTO STUDENTEN(SID, VORNAME, NACHNAME)
VALUES (105, 'Nina', 'Brass')
```

- 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-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:

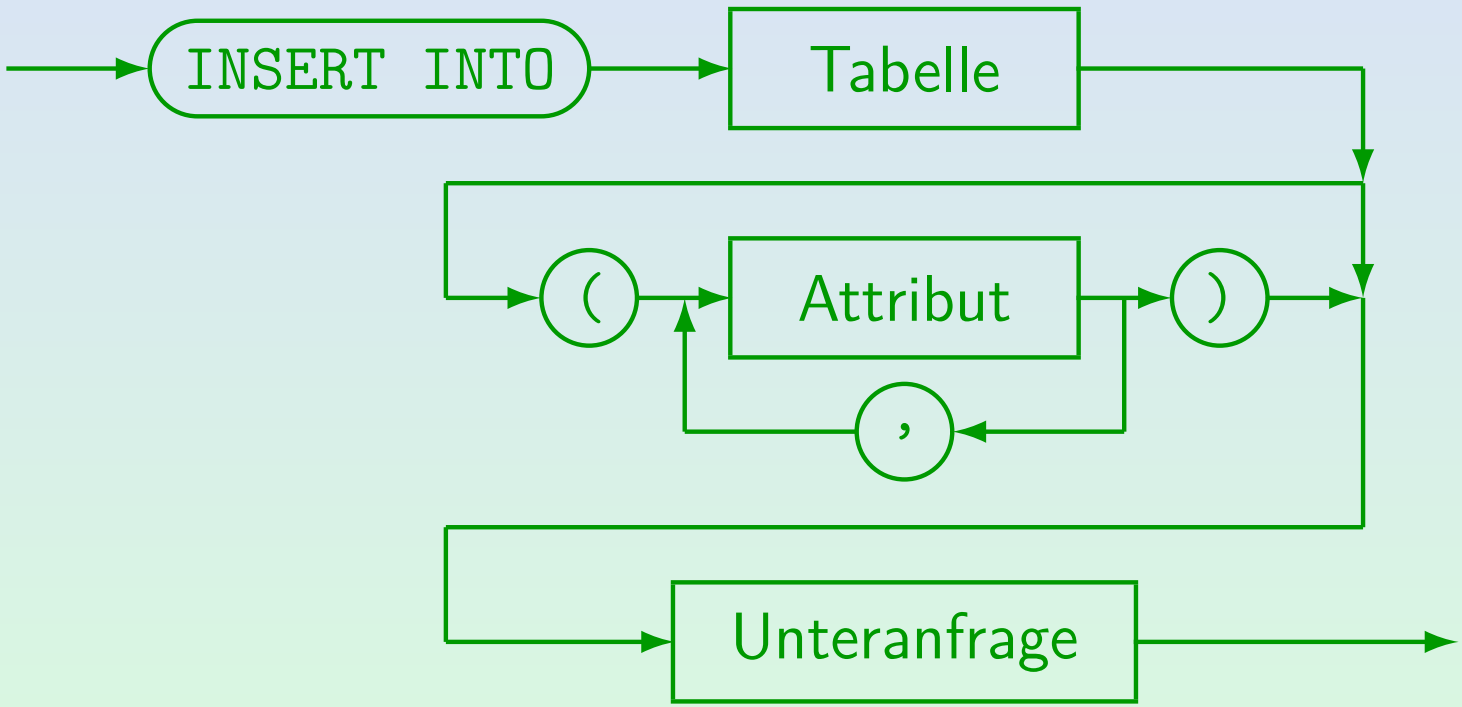
```
INSERT INTO KLAUSUREN(BEZ, ANR, THEMA, PROZENT)
SELECT 'DB-2005-Z', A.ANR, A.THEMA,
       AVG(B.PUNKTE/A.MAXPT)*100
FROM   AUFGABEN A, BEWERTUNGEN B
WHERE  A.ATYP='Z' AND B.ATYP='Z' AND A.ANR=B.ANR
GROUP  BY A.ANR, A.THEMA
```

- 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 `(...)`.

DELETE (1)

- Beispiel: Lösche alle Bewertungen für Lisa Weiss:

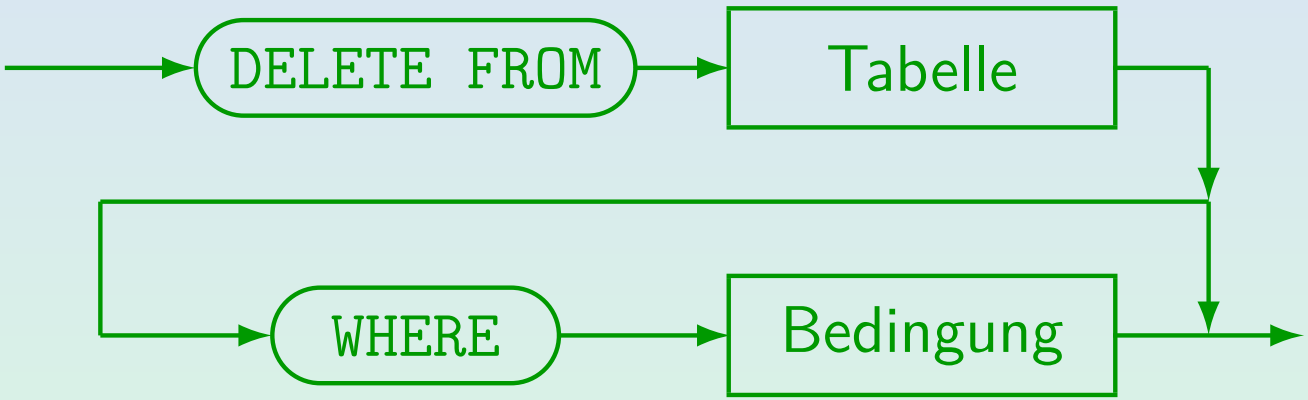
```
DELETE FROM BEWERTUNGEN
WHERE  SID IN (SELECT SID
               FROM   STUDENTEN
               WHERE  VORNAME  = 'Lisa'
               AND    NACHNAME = 'Weiss')
```

- 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:



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 ANR = 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).

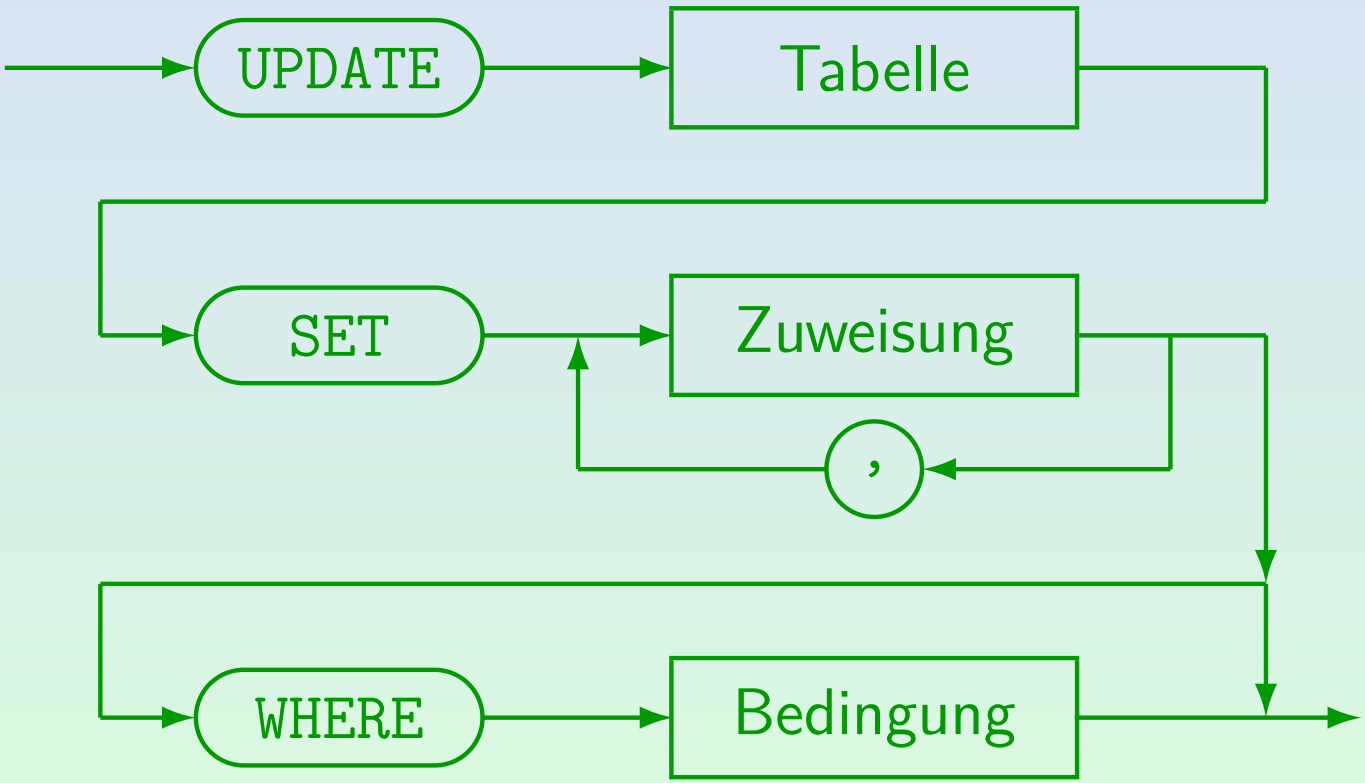
  Falls sie keine Zeile liefert, wird ein Nullwert verwendet.

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

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

UPDATE (3)

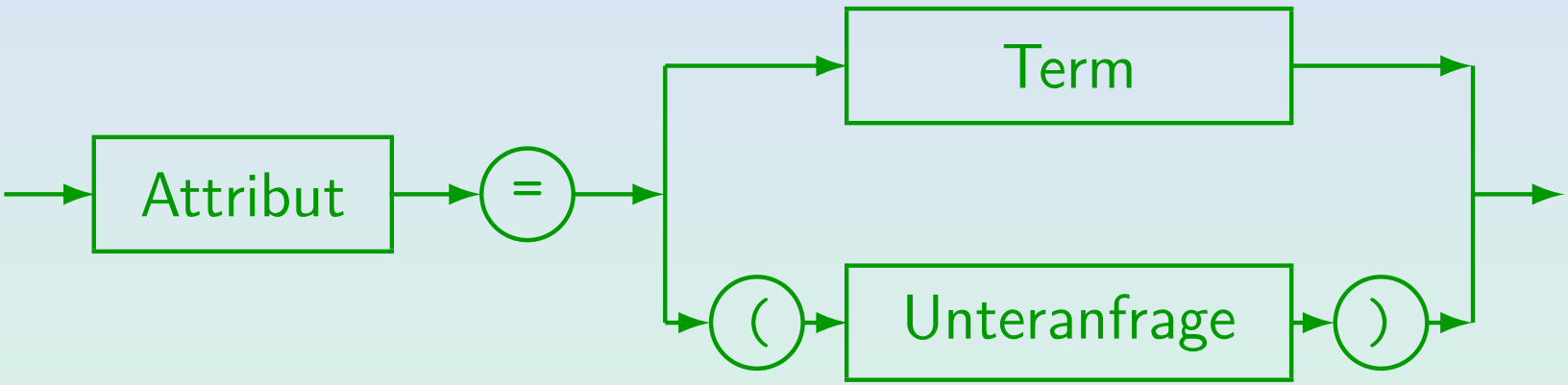
UPDATE-Befehl:



- 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 muss die Unteranfrage explizit genannt werden.

Inhalt

TRUNCATE (1)

- PostgreSQL, 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.

  [https://www.postgresql.org/docs/9.2/sql-truncate.html]

- 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 (in Oracle).

  In PostgreSQL geht es dagegen.

- Dafür ist es viel schneller.

  Zumindest in Oracle würde `DELETE` auch nicht wirklich Speicherplatz frei geben, 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 erst seit SQL:2008 zum SQL Standard.

  Es kam aber z.B. schon 1999 in der Oracle Zertifizierungs-Prüfung vor.

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.

- PostgreSQL unterstützt nicht das `MERGE`-Statement.

  PostgreSQL hat eine `ON CONFLICT` Klausel beim `INSERT`, die einem ähnlichen Zweck dient (aber kein Standard-SQL ist). Siehe unten (Folie 28).

- 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.

- Die Anweisung ist eine Kombination von `UPDATE` und `INSERT` (solche Anweisungen werden manchmal „`UPSERT`“ genannt).

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 Quelltabelle (`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 (und parallele Zugriffe anderer Nutzer zwischen beiden Befehlen problematisch):

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

Bei mehreren `UPDATE`-Spalten braucht man jeweils eine Unteranfrage.

Erweiterungen in PostgreSQL (1)

- PostgreSQL (ab Version 9.5) hat statt `MERGE` „`ON CONFLICT`“ beim `INSERT` (nicht im Standard):

```
INSERT INTO BEWERTUNGEN(SID,ATYP,ANR,PUNKTE)
SELECT SID, 'H', ANR, PUNKTE
FROM   GRUPPE1
ON CONFLICT (SID, ATYP, ANR)
DO UPDATE   SET PUNKTE = EXCLUDED.PUNKTE
```

Die Spalten nach „`ON CONFLICT`“ dienen dazu, einen Schlüssel auszuwählen. Man kann alternativ auch „`ON CONFLICT ON CONSTRAINT BEW_PK`“ sagen, wenn man den Primärschlüssel von `BEWERTUNGEN` mit „`CONSTRAINT BEW_PK`“ benannt hat. Alternativ zu „`DO UPDATE`“ kann man auch „`DO NOTHING`“ wählen, dann werden problematische Zeilen einfach ignoriert.
Bei „`DO UPDATE`“ gibt es die spezielle Tupelvariable „`EXCLUDED`“ für das Tupel, das nicht eingefügt werden kann.

Erweiterungen in PostgreSQL (2)

- In PostgreSQL ist es möglich, bei `INSERT`, `UPDATE` und `DELETE` Werte der betroffenen Zeilen wie bei einer Anfrage zu bekommen (nicht standard-konform):

```
UPDATE    BEWERTUNGEN
SET       PUNKTE = PUNKTE + 1
WHERE     ATYP = 'Z' AND ANR = 1
RETURNING SID, PUNKTE
```

- Dies würde die SIDs der Studierenden liefern, deren Punktzahl erhöht wurde (und die neue Punktzahl).
- Man kann so auch automatisch generierte IDs bekommen.

  Das ist wahrscheinlich die Haupt-Anwendung dieses Konstrukts (beim `INSERT` den generierten `DEFAULT`-Wert abfragen). Es ist aber auch nützlich, dass sich keine parallelen Nutzer zwischen Update und Anfrage schieben können.

Inhalt

Laden großer Datenmengen

- Die meisten DBMS haben Spezialbefehle oder Zusatz-Werkzeuge zum Laden großer Datenmengen aus Dateien mit verschiedenen Formaten (z.B. csv).

  „CSV“ bedeutet „Comma-Separated Values“, siehe nächste Folie.

- Die Befehle für solche „Bulk Loads“ sind sehr systemspezifisch (in jedem DBMS anders).

- Das Laden großer Datenmengen geht mit den Spezialbefehlen signifikant schneller als mit einzelnen `INSERT`-Befehlen.

  Dafür werden eventuell Transaktions-Garantien abgeschwächt.

  Möglicherweise kann man auch die Integritäts-Prüfung abschalten.

- Ein Vorteil ist auch, dass man für die unterstützten Dateiformate nicht selbst etwas programmieren muss.

CSV-Format (1)

- Die Daten einer einzelnen Tabelle kann man gut im „Comma-Separated Values“-Format austauschen.

  Definiert in RFC 4180: [https://tools.ietf.org/html/rfc4180]

  Siehe auch: [https://de.wikipedia.org/wiki/CSV_(Dateiformat)]

- Die Datenwerte der Spalten werden durch Kommata getrennt, die Zeilen durch einen Zeilen-Umbruch.

  Der RFC verlangt den DOS/Windows-Zeilenumbruch aus CR und LF. Außerdem soll jede Zeile gleich viele Felder (Datenwerte) enthalten.

- Datenwerte können in "..." eingeschlossen werden.

  In diesem Fall können sie auch Kommata und Zeilenumbrüche enthalten, außerdem auch ", das muss allerdings verdoppelt werden: "". Wenn der Datenwert nicht in "..." steht, sind Komma und " verboten, außerdem alle Steuerzeichen (mit Code < 32). Bei Einschluss in "..." sind die einzigen erlaubten Steuerzeichen CR und LF.

CSV-Format (2)

- Beispiel (teils mit, teils ohne Einschluss in "..."):

```
"101","Lisa","Weiss","weiss@acm.org"
102,Michael,Grau,
103,Daniel,Sommer,daniel@gmx.de
104,Iris,Winter,irisw@gmail.com
```

Es gibt im CSV-Format keine spezielle Codierung für Nullwerte.

Die EMail-Adresse von Michael Grau könnte als leerer String importiert werden (ggf. mit `UPDATE` korrigieren). Für PostgreSQL siehe Folie 40.

STUDENTEN

SID	VORNAME	NACHNAME	EMAIL
101	Lisa	Weiss	weiss@acm.org
102	Michael	Grau	NULL
103	Daniel	Sommer	daniel@gmx.de
104	Iris	Winter	irisw@gmail.com

CSV-Format (3)

- Oft werden die Spaltennamen in der ersten Zeile übermittelt:

```
SID,VORNAME,NACHNAME,EMAIL
101,Lisa,Weiss,weiss@acm.org
...
```

- Leider kann man am Inhalt der Datei nicht unbedingt erkennen, wie die erste Zeile zu interpretieren ist (Spaltenamen oder schon Daten).

  Der MIME-Typ `text/csv` hat einen optionalen Parameter `header` mit den Werten `present` und `absent`. Außerdem gibt es einen Parameter `charset` für die Zeichencodierung. Ein möglicher Typ (z.B. als `Content-Type` in HTTP) wäre: `text/csv;charset=iso-8859-1;header=present`

- Wenn man das CSV-Format zum Austausch von Daten verwendet, müssen solche Details dokumentiert werden.

  Ebenso wie das Format von Datenwerten, z.B. Datums-Angaben.

CSV-Format (4)

- Oft erlauben Import-Funktionen die Wahl von
  - Zeichensatz,
  - Trennzeichen (z.B. „;“ statt „,“),
  - Anführungzeichen („Feldbegrenzerzeichen“), und
  - Interpretation der ersten Zeile.
- Das CSV-Format kann auch von Tabellenkalkulationen wie Microsoft Excel gelesen und geschrieben werden.

  Leider ist Excel zu intelligent und interpretiert Folgen von Ziffern als Zahlwerte, weswegen dann führende Nullen z.B. in Postleitzahlen fehlen. CSV macht keine Aussagen über Datentypen. Man kann die Datei z.B. mit der Extension „`.txt`“ benennen, und kommt dann beim Öffnen in den „Text-Import Wizard“, wo man für die entsprechende Spalte den Typ „Text“ wählen kann. (Oder Endung lassen und statt Doppelklick explizit importieren.)

CSV-Format (5)

- Es gibt auch eine Variante „TSV“: „Tab-Separated Values“, die statt des Kommas ein Tabulator-Zeichen verwendet.

  Wenn man das Trennzeichen frei wählen kann, ist das einfach ein Spezialfall von CSV (bzw. „delimiter-separated values“).
  [https://en.wikipedia.org/wiki/Tab-separated_values]

- Wenn sich die Länge der Datenwerte nicht zu stark unterscheidet, lässt sich das gut in einem Texteditor lesen:

```
101    Lisa    Weiss  weiss@acm.org
102    Michael Grau
103    Daniel  Sommer daniel@gmx.de
104    Iris    Winter irisw@gmail.com
```

  Wenn ein Name natürlich länger als 7 Zeichen ist (bei einer Tabulator-Breite von 8 Zeichen) verrutscht der Rest dieser Zeile.

Daten-Import/Export in PostgreSQL (1)

- PostgreSQL-Befehl zum Laden großer Datenmengen: `COPY`.

  [https://www.postgresql.org/docs/12/sql-copy.html]

- Der Befehl heißt so, weil man damit auch umgekehrt Tabellen in Dateien exportieren kann:

  - `COPY STUDENTEN FROM '/home/sb/studenten.csv' FORMAT CSV`

    Dieser Befehl läd die Daten von der Datei in die Tabelle (Daten-Import). Weil der Befehl auf dem Server-Rechner ausgeführt wird, muss die Datei dort stehen und mit den Rechten des Nutzers „`postgres`“, unter dem der Server läuft, zugreifbar sein. Sicherheit: Sie nächste Folie.

  - `COPY STUDENTEN TO '/home/sb/studenten.csv' FORMAT CSV`

    Dieser Befehl exportiert Daten aus der Tabelle in die Datei. Statt einer Tabelle kann man auch eine `SELECT`-Anfrage in Klammern schreiben.

Daten-Import/Export in PostgreSQL (2)

- Weil der obige `COPY`-Befehl Dateien auf dem Server liest oder schreibt, kann er nur mit Administrator-Rechten (für das DBMS) benutzt werden.

  D.h. der „`CREATE USER`“ Befehl enthielt die Option „`SUPERUSER`“.
  Alternativ reicht auch die Rolle (d.h. das Recht) „`pg_read_server_files`“.
  Statt einer Datei kann auch `PROGRAM '...'` angegeben werden, wenn z.B. die Eingabedaten von PostgreSQL Ausgabe eines Programms sind (etwa Dekomprimierung einer Datei). Es können damit also auch Programme auf dem Server ausgeführt werden (mit den Rechten des „`postgres`“ Nutzers).

- Nur Varianten mit `STDIN` bzw. `STDOUT` statt eines Dateinamens sind für normale Nutzer möglich.

  Dies bedeutet, dass die Eingabe vom Client kommt, bzw. die Ausgabe zum Client geschickt wird. Diese Variante wird von der Kommandoschnittstelle `psql` benutzt, um die Funktionalität normalen Nutzern anzubieten, aber mit Dateien auf dem Client.

Daten-Import/Export in PostgreSQL (3)

- In der Kommandoschnittstelle `psql` geht der Import so:

  ```
  \COPY STUDENTEN FROM 'studenten.csv' FORMAT CSV
  ```

  Hier machen relative Dateinamen Sinn (sie beziehen sich auf das aktuelle Verzeichnis, in dem `psql` gestartet wurde).

- In beiden Varianten (`COPY` auf dem Server und `\COPY` auf dem Client) gibt es eine ganze Reihe von Optionen, um das Datenformat festzulegen.

  [https://www.postgresql.org/docs/12/app-psql.html] (Client-Anleitung)
  Ausführlichere Informationen finden sich aber beim serverseitigen Kommando, das auch der Client implizit benutzt:
  [https://www.postgresql.org/docs/12/sql-copy.html]
  Z.B. `DELIMITER 'c'`, `QUOTE 'c'`, `ESCAPE 'c'` zur Wahl der Spezial-Zeichen und `FORCE_QUOTE`, um alle Datenwerte in `"..."` zu schreiben.
  Alle Optionen werden nur durch Leerzeichen getrennt (ohne Komma).

Daten-Import/Export in PostgreSQL (4)

- PostgreSQL behandelt leere Einträge als Nullwerte.

  Zur Unterscheidung wird ein leerer String mit Anführungszeichen geschrieben: `""`. Wenn man nicht will, das leere Einträge als Nullwerte gelesen werden, muss man `FORCE_NOT_NULL` angeben. Dies geht auch für spezielle Spalten. Mit der Option `NULL '...'` kann man eine Codierung für den Nullwert wählen (normalerweise der leere String im CSV-Format).

- `FORMAT CSV HEADER`: CSV-Format mit Spaltennamen in erster Zeile.

  Bei der Eingabe wird die erste Zeile der Datei einfach ignoriert.

- `ENCODING 'UTF8'`: Wählt Zeichencodierung UTF-8.

  Das ist vermutlich der Default. Alternativ z.B. „`LATIN1`“. In `psql` kann man mit „`show server_encoding;`“ und „`show client_encoding;`“ die aktuellen Codierungen abfragen. Alle unterstützten Codierungen stehen hier: [https://www.postgresql.org/docs/12/multibyte.html]

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).
  - Seit SQL-99 gibt es aber „savepoints“. Wenn man innerhalb einer Transaktion einen „savepoint“ gesetzt hat, kann man bei Bedarf auf den so benannten Zustand zurücksetzen.

    Natürlich nur innerhalb der Transaktion. Savepoints gibt es z.B. in Oracle, PostgreSQL, MS SQL Server.

    [https://www.postgresql.org/docs/current/sql-savepoint.html]
  - 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.
- 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.

- 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.

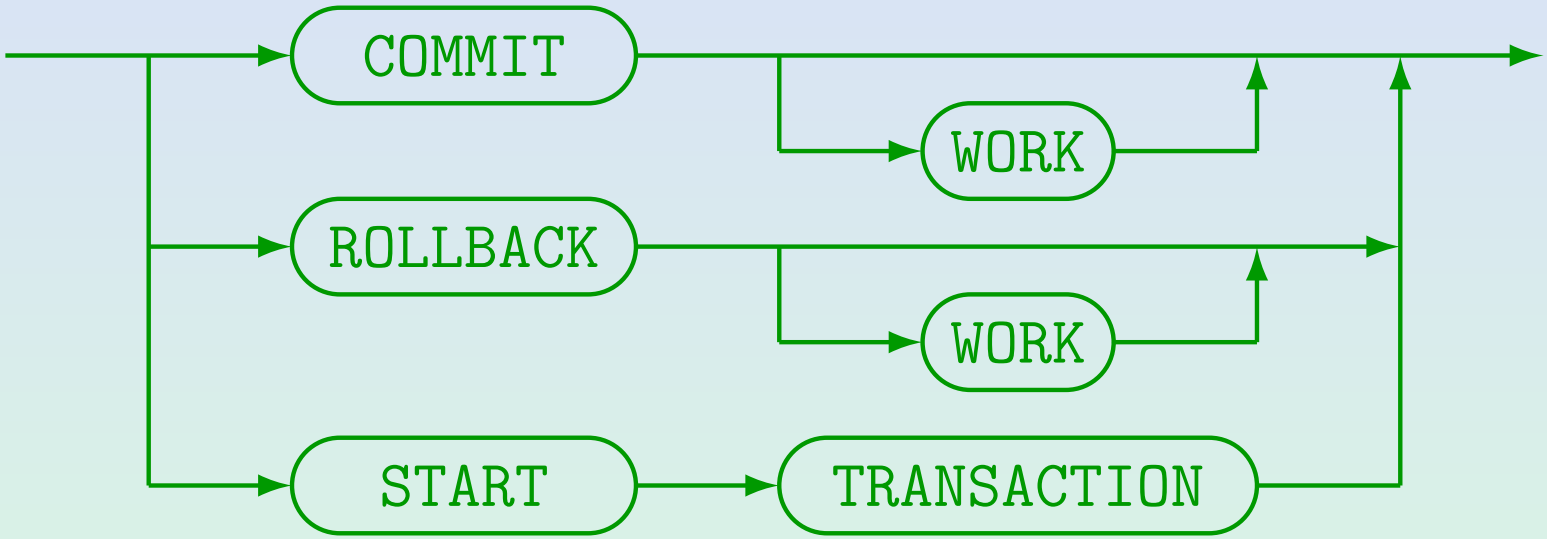
Transaktions-Verwaltung (4)

- 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.