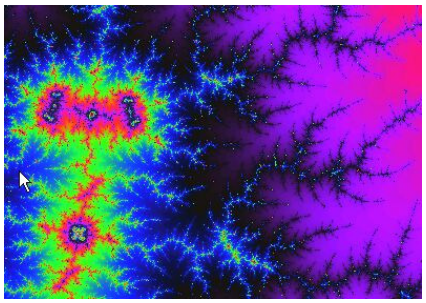
Datenbanken

3. Komplexe SQL Abfragen



Fachhochschule Darmstadt
Fachbereich Informatik

Dr. Peter Nevermann

Rückblick

```
SELECT [DISTINCT]<*,attribute, funktionen>
FROM <tabelle>
[WHERE <bedingung>]
[GROUP BY <gruppierungs-attribute>]
[HAVING <gruppen-bedingung>]
[ORDER BY <attribut-list>];
```

- **Arithmetik:** `+, -, *, /` **in** `SELECT`, `WHERE`, **...**
- **Funktionen:** `CONCAT()`, `YEAR()`, `MONTH()`, `EXP()`, `LN()`, **...**
- **Vergleiche:** `=, <, <=, >, >=, <>` **und** `LIKE` **in** `WHERE` **und** `HAVING`
- **Logisch:** `NOT`, `AND`, `OR`, **(**`BETWEEN`**) in** `WHERE` **und** `HAVING`
- **Sortierung: aufsteigend (**`ASC`**), absteigend (**`DESC`**)**
- **Mengenoperationen:** `UNION`, `INTERSECT`, `EXCEPT`
- **NULL Werte:** `IS NULL`, `IS NOT NULL`
- **Aggregatfunktionen:**
  `COUNT([DISTINCT])`, `SUM()`, `MAX()`, `AVG()`, `MIN()`

Datenbanken I | 2 | SS 05, 29.04.2005 | FH Darmstadt, FB Informatik, Dr. P. Nevermann

An dieser Stelle möchte ich einige Funktionssignaturen nachliefern (MySQL-konform), welche im Skript der vorigen Woche gefehlt haben.

String Funktionen

•|| **Operator:** Der standard String-Konkatenationsoperator || wird in MySQL als logischer Operator verwendet und ist äquivalent mit OR. Daher gibt es in MySQL die CONCAT Funktion.

•**CONCAT(s1, s2, ...)**, wobei s1, s2, ... string-wertige Attribute oder sonstige Strings sein können. Numerische Argumente werden auch akzeptiert und dessen String-Darstellung verwendet. Ist ein Argument NULL, so ist das Ergebnis ebenfalls NULL.

•**LENGTH(str)** gibt die Länge des Strings str zurück

•**SUBSTRING(str, pos, len)** gibt den Teilstring ab Position pos (Achtung: beginnend mit 1, nicht mit 0) in der Länge len zurück
LEFT(str, len) äquivalent zu SUBSTRING(str, 1, len)
RIGHT(str, len) äquivalent SUBSTRING(str, LENGTH(str) – len + 1, len)

•**INSTR(str, substr)** gibt die Position in str (beginnend mit 1) des ersten Vorkommens des Strings substr an, bzw. 0 falls substr nich in str vorkommt
Synonym:
LOCATE(substr, str)
POSITION(substr IN str)

•**REPLACE(str, from_str, to_str)** ersetzt in str alle Vorkommnisse von from_str durch to_str

•**LOWER(str)** verwandelt den String str komplett in Kleinschreibung
UPPER(str) analog in Großschreibung

Date/Time-Funktionen

•**CURRENT_DATE()** gibt das aktuelle Datum als 'YYYY-MM-DD' oder YYYYMMDD zurück, je nach dem ob die Funktion in einem String- oder in einem numerischen Kontext verwendet wird.

•**CURRENT_TIME()** gibt die aktuelle Zeit als 'HH:MM:SS' oder HHMMSS zurück

•**NOW()** gibt den aktuellen Zeitstempel als 'YYYY-MM-DD HH:MM:SS' oder YYYYMMDDHHMMSS zurück
Synonym
CURRENT_TIMESTAMP()

•**DATE(expr)** extrahiert den Datumsanteil aus einem Zeitspempel-Ausdruck
TIME(expr) analog mit dem Zeitanteil

•**YEAR(date)** extrahiert die Jahreszahl aus einem Datum

•**MONTH(date)** extrahiert den Monat aus einem Datum
MONTHNAME(datum) liefert den Monatsnamen

•**DAY(date)** extrahiert den Tag aus dem Datum
DAYNAME(date) liefert den Tagesnamen

•**HOUR(time)** extrahiert die Stunde
MINUTE(time) extrahiert die Minute
SECOND(time) extrahiert die Sekunde

Unterabfragen

- **Welche weiblichen Mitarbeiter verdienen mehr als der bestverdienende männliche Kollege?**

```
SELECT nachname, gehalt
FROM mitarbeiter
WHERE geschlecht = 'W'
AND gehalt > ALL (
  SELECT gehalt
  FROM mitarbeiter
  WHERE geschlecht = 'M'
);
```

nachname	gehalt
Grenz	65000
Noll	70000

Datenbanken I | 4 | SS 05, 29.04.2005 | FH Darmstadt, FB Informatik, Dr. P. Nevermann

Vergleiche mit Multi-Mengen in WHERE (ALL, SOME, ANY, IN)

Im folgenden bezeichnet "V" eine *Multi-Menge* von *Werten*, welche Duplikate enthalten kann**.**

•a = (V), a > (V), a <= (V), ... [setzt voraus, daß V nur aus einem Wert besteht]

•a > **ALL** (V) ⇔ a > **SOME** (V) ⇔ **NOT** a <= **ANY** (V)

•a < **ANY** (V) ⇔ **NOT** a >= **ALL(V**)

•a = **ANY** (V) ⇔ a = **SOME** (V) ⇔ a **IN** (V)

"V" kann eine einfache Auflistung sein, z.B.

•Wer verdient 50000, 60000 oder 70000?
SELECT nachname, gehalt
FROM mitarbeiter
WHERE gehalt IN (50000, 60000, 70000);

Unterabfragen in WHERE

"V" kann aber auch das Ergebnis einer *Unterabfrage* mit nur einem Attribut in SELECT sein, wie das Beispiel auf der Folie zeigt.

Weitere Beispiele:

•Wer verdient am meisten?

```
SELECT nachname, gehalt
FROM mitarbeiter
WHERE gehalt = (
  SELECT max(gehalt) FROM mitarbeiter);
```

•Wer verdient überdurchschnittlich?

```
SELECT nachname, gehalt
FROM mitarbeiter
WHERE gehalt >= (
  SELECT avg(gehalt) FROM mitarbeiter);
```

Vorsicht ist wieder mit NULL-Werten geboten:

•Alle Mitarbeiter die Herrn Mayer (pnr=1111) oder Herrn Bach (pnr=7777) als Vorgesetzten haben?

```
SELECT pnr, vpnr
FROM mitarbeiter
WHERE vpnr IN (1111, 7777);
```

Ergebnis:
(2222, 1111), (4444, 1111), (5555, 7777), (6666, 7777) ... soweit alles OK, aber:

•Alle Mitarbeiter die **weder** Herrn Mayer (pnr=1111) **noch** Herrn Bach (pnr=7777) als Vorgesetzten haben?

```
SELECT pnr, vpnr
FROM mitarbeiter
WHERE vpnr NOT IN (1111, 7777);
```

Ergebnis:
(1111, 3333), (3333, 8888), (7777, 3333) ... aber leider fehlt (8888, **NULL**) im Ergebnis!!

Wie muß die 2. Abfrage also korrekt lauten?

Unterabfragen

- **Was ist das Durchschnittsgehalt der *männlichen Mitarbeiter pro Abteilung mit mehr als zwei Mitarbeitern*?**

```
SELECT abtnr, AVG(gehalt)
FROM mitarbeiter
WHERE geschlecht = 'M'
GROUP BY abtnr
HAVING count(*) > 2;
```

abtnt	AVG(gehalt)
2	53333.3333

```
SELECT abtnr, AVG(gehalt)
FROM mitarbeiter
WHERE geschlecht = 'M' AND
abtnr IN
  (SELECT abtnr FROM mitarbeiter
   GROUP BY abtnr
   HAVING count(*) > 2)
GROUP BY abtnr;
```

abtnt	AVG(gehalt)
2	53333.3333
3	55000.0000

Datenbanken I | 6 | SS 05, 29.04.2005 | FH Darmstadt, FB Informatik, Dr. P. Nevermann

Zurück zur Aufgabe von voriger Woche: mit Unterabfragen geht's also, die zweite Abfrage auf der Folie liefert jetzt das gewünschte Ergebnis!

"Korrelierte" Unterabfragen

- **In welchen Abteilungen arbeiten nur Frauen?**

```
SELECT anr, abtname
FROM abteilung
WHERE NOT EXISTS (
  SELECT *
  FROM mitarbeiter
  WHERE abtnr = anr AND
        geschlecht = 'M'
);
```

anr	abtname
1	Headquarters

Datenbanken I | 7 | SS 05, 29.04.2005 | FH Darmstadt, FB Informatik, Dr. P. Nevermann

Eine Unterabfrage in WHERE kann sich auch auf Attribute der umfassenden Abfrage beziehen (*correlated subquery*). Korrelierte Unterabfragen werden mehrfach, abhängig von Ergebnissen der äußeren Abfrage ausgeführt.

Bei korrelierten Unterabfragen kann es zu Konflikten mit den Attributnamen kommen. Im Beispiel auf der Folie wäre das der Fall gewesen, wenn die Abteilungs-Nr in beiden Tabellen (also *abteilung* und *mitarbeiter*) denselben Namen (z.B. *abtnr*) gehabt hätten. In einem solchen Fall werden die Attributnamen mit dem Tabellennamen *qualifiziert*, d.h. in der Form <tabellen-name>.<spalten-name> geschrieben. Mit qualifizierten Attributnamen sieht obiges Beispiel dann folgendermaßen aus:

```
SELECT anr, abtname
FROM abteilung
WHERE NOT EXISTS (
  SELECT * FROM mitarbeiter
  WHERE mitarbeiter.abtnr = abteilung.anr AND
  mitarbeiter.geschlecht = 'M';
```

Regel:

Unqualifizierte Attributnamen beziehen sich immer auf die Tabelle der Unterabfrage selbst bzw. der nächst-äusseren (Unter-)Abfrage.

Operatoren:

•**EXISTS** – prüft, daß das Ergebnis einer korrelierten Unterabfrage nicht leer ist

•**UNIQUE** - prüft, daß das Ergebnis einer korrelierten Unterabfrage keine Duplikate hat

Der UNIQUE Operator ist weder in Access noch in MySQL implementiert.

Verbünde

- **Wer arbeitet in der Verwaltung?**

```
SELECT abteilung.abtname, mitarbeiter.nachname
FROM abteilung, mitarbeiter
WHERE abteilung.abtname = 'Verwaltung' AND
      mitarbeiter.abtnr = abteilung.anr;
```

abtname	nachname
Verwaltung	Schneider
Verwaltung	Schumann
Verwaltung	Bach

Datenbanken I | 8 | SS 05, 29.04.2005 | FH Darmstadt, FB Informatik, Dr. P. Nevermann

Abfragen auf mehreren Tabellen

Bisher haben wir uns auf Abfragen beschränkt, welche in FROM nur eine Tabelle ansprechen. Normalerweise sind aber die Daten einer Datenbank über mehrere Tabellen verteilt.

Warum überhaupt viele Tabellen verwenden? Kann man nicht alle Informationen in einer einzigen Tabelle speichern?

Nunja, man schreibt ja zuhause auch keine Kochrezepte in das Adressenbuch ☺.

Ein weiterer Schwerpunkt dieses Kurses (neben SQL) wird es übrigens sein, die optimale Anzahl und Struktur von Tabellen für eine bestimmte Datenbank zu entwerfen.

Verbünde:

Im Beispiel der Folie muß zusätzlich zur *mitarbeiter* Tabelle noch die *abteilung* Tabelle herangezogen werden, weil in der *mitarbeiter* Tabelle nur die Abteilungsnummer, nicht aber der Abteilungsname steht, welcher wiederum in der *abteilung* Tabelle zu finden ist. In dem Beispiel wird einen *Verbund* (oder *Verküpfung*, engl: *join*) zwischen den beiden Tabellen hergestellt. Dabei werden nur Tupel der beiden Tabellen kombiniert, welche der *Join-Bedingung* (mitarbeiter.abtnr = abteilung.anr) genügen. Fehlt die Join-Bedingung, so werden alle Tupel (= Kreuzprodukt) gebildet!

Verbünde können natürlich auch über 3 oder mehr Tabellen hergestellt werden.

Sind mehrere Tabellen in einer Abfrage im Spiel, ist es angebracht alle Attribute zu *qualifizieren* (wie im Beispiel geschehen). Um die Schreibweise zu verkürzen, kann man den beteiligten Tabellen auch Kurznamen (oder *Alias-Namen*) zuordnen, welche man dann zum Qualifizieren der Attribute verwendet. Alias-Namen für Tabellen werden in der FROM-Klausel in der Form *tabellen-name alias-name* oder *tabellen-name* **AS** *alias-name* vergeben.

Beispiel:

•Wer arbeitet in der Verwaltung?

```
SELECT a.abtname, m.nachname
FROM abteilung AS a, mitarbeiter AS m
WHERE a.abtname = 'Verwaltung' AND
m.abtnr = a.anr;
```

Bildet man einen Verbund einer Tabelle mit sich selbst, dann ist das *Aliasing* sogar unumgänglich, wie folgendes Beispiel zeigt:

•Liste der Mitarbeiter jeweils mit ihren Vorgesetzten?

```
SELECT m1.nachname, m2.nachname
FROM mitarbeiter m1, mitarbeiter m2
WHERE m1.vpnr = m2.pnr;
```

Verbünde in SQL2 (JOIN)

- **Wer arbeitet in der Verwaltung?**

```
SELECT a.abtname, m.nachname
FROM (abteilung a JOIN mitarbeiter m
      ON m.abtnr = a.anr)
WHERE a.abtname = 'Verwaltung';
```

abtname	nachname
Verwaltung	Schneider
Verwaltung	Schumann
Verwaltung	Bach

Die "traditionelle" Schreibweise des Joins (wie auf der vorigen Folie zu sehen) kann unübersichtlich werden, da die *Join-Bedingungen* mit den übrigen Bedingungen in WHERE vermischt werden.

SQL2 führt eine günstigere Schreibweise ein, die es erlaubt verknüpfte Tabellen als eine einzige Tabelle (*joined-table*) in FROM erscheinen zu lassen.

JOIN Operatoren:

•(t1 **JOIN** t2 **ON** t1.a = t2.b) ⇔ (t1 **INNER JOIN** t2 **ON** t1.a = t2.b)

•(t1 **NATURAL JOIN** t2) ⇔ (t1 **JOIN** t2 **ON** t1.a1 = t2.a1 AND t1.a2 = t2.a2 ...) [wobei a1, a2, ... genau die Attributnamen sind, die in beiden Tabellen vorkommen]

•(t1 **LEFT OUTER JOIN** t2 **ON** t1.a = t2.b) ⇔ (t1 **LEFT JOIN** t2 **ON** t1.a = t2.b)

•(t1 **RIGHT OUTER JOIN** t2 **ON** t1.a = t2.b) ⇔ (t1 **RIGHT JOIN** t2 **ON** t1.a = t2.b)

Das Beispiel auf der Folie zeigt einen *inneren* Verbund (*inner join*). Das Schlüsselwort INNER kann weggelassen werden.

Der *natürlichen* Verbund (*natural join*) geht automatisch über alle Attributnamen die in beiden Tabellen den selben Namen haben. Beim natürlichen Verbund ist also die ON-Klausel nicht nötig.

Beim *äußeren* Verbund (*outer join*) werden zu einer der beteiligten Tabellen (links oder rechts) alle Tupel selektiert, d.h. auch solche ohne Übereinstimmung in der gegenüberliegenden Tabelle. In der Ergebnistabelle werden dann Tupel ohne Gegenpart in den Spalten der Gegenüberliegenden Tabelle mit NULL aufgefüllt. Das Schlüsselwort OUTER kann weggelassen werden.

Normalerweise wird die Join-Bedingung mit dem '=' Operator ausgedrückt (*equi-join*). In Ausnahmefällen kann es sinnvoll sein einen anderen Operator ('>', '<=', ...) für die Join-Bedingung zu benutzen (z.B. für Berechnungen in einem Routennetz). Dann spricht man vom *non-equi-join*.

Über die JOIN Operatoren können durch Schachtelung auch mehr als zwei Tabellen verknüpft werden: ((t1 **JOIN** t2 **ON** t1.a = t2.b) **JOIN** t3 **ON** t1.c = t3.c).

Mehr Beispiele:

•Namen der Mitarbeiter der Abteilung 1 jeweils mit Namen des Vorgesetzten?
SELECT m1.nachname, m2.nachname
FROM mitarbeiter m1 JOIN mitarbeiter m2
ON m1.vpnr = m2.pnr
WHERE m1.abtnr = 1;
Ergebnis:
(Grenz, Noll)

•Namen der Mitarbeiter der Abteilung 1 jeweils mit Namen des Vorgesetzten? (Diesmal mit Frau Noll bitte!!)
SELECT m1.nachname, m2.nachname
FROM mitarbeiter m1 **LEFT** JOIN mitarbeiter m2
ON m1.vpnr = m2.pnr
WHERE m1.abtnr = 1;
Ergebnis:
(Grenz, Noll), (Noll, NULL)

Im Ergebnis der obigen Beispiele erscheint "nachname" jeweils als Spaltenüberschrift für den Mitarbeiter- und Vorgesetzten-Namen. In solchen Fällen kann man sich wieder Alias-Namen – diesmal für Attribute – helfen. Alias-Namen für Attribute werden in der SELECT-Klausel in der Form *attribut-name alias-name* oder *attribut-name* **AS** *alias-name* vergeben.

Beispiel

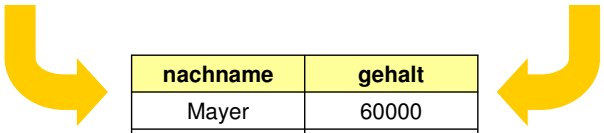
•Namen der Mitarbeiter der Abteilung 1 jeweils mit Namen des Vorgesetzten? (Diesmal mit Frau Noll und vernünftigen Spaltenüberschriften bitte!!)
SELECT m1.nachname **AS Mitarbeiter**, m2.nachname **AS Vorgesetzter**
FROM mitarbeiter m1 LEFT JOIN mitarbeiter m2
ON m1.vpnr = m2.pnr
WHERE m1.abtnr = 1;

Verbünde und Unterabfragen

- **Nachname und Gehalt der *männlichen Mitarbeiter in der Forschungsabteilung*?**

```
SELECT nachname, gehalt
FROM mitarbeiter
WHERE geschlecht = 'M'
AND abtnr IN (
    SELECT anr
    FROM abteilung
    WHERE abtname =
          'Forschung');
```

```
SELECT nachname, gehalt
FROM (
  mitarbeiter JOIN abteilung
  ON abtnr = anr)
WHERE geschlecht = 'M'
AND abtname = 'Forschung';
```



nachname	gehalt
Mayer	60000
Wegner	55000
Clausen	45000

Datenbanken I | 12 | SS 05, 29.04.2005 | FH Darmstadt, FB Informatik, Dr. P. Nevermann

Wie das Beispiel auf der Folie zeigt, läßt sich eine Unterabfrage mit IN häufig auch als Join formulieren.

Frage: Geht das auch mit folgender Unterabfragen?

•Welche weiblichen Mitarbeiter verdienen mehr als bestverdienende männliche Kollege?

```
SELECT nachname, gehalt
FROM mitarbeiter
WHERE geschlecht = 'W'
AND gehalt > ALL (
  SELECT gehalt
  FROM mitarbeiter
  WHERE geschlecht = 'M');
```

•Welche weiblichen Mitarbeiter verdienen mehr als irgendein männlicher Kollege?

```
SELECT nachname, gehalt
FROM mitarbeiter
WHERE geschlecht = 'W'
AND gehalt > ANY (
  SELECT gehalt
  FROM mitarbeiter
  WHERE geschlecht = 'M');
```

Unterabfragen in FROM (*derived tables*)

- **Nachnamen der Mitarbeiter in Abteilungen mit genau einem Standort?**

```
SELECT m.nachname
FROM mitarbeiter m,
   (SELECT s.abtnummer FROM standort s
   GROUP BY s.abtnummer
   HAVING COUNT(*) = 1) AS a1
WHERE m.abtnr = a1.abtnummer;
```

nachname
Noll
Grenz
Bach
Schumann
Schneider

Datenbanken I | 13 | SS 05, 29.04.2005 | FH Darmstadt, FB Informatik, Dr. P. Nevermann

Bei einer Unterabfrage in FROM wird die Unterabfrage (besser gesagt: das Ergebnis der Unterabfrage) wieder als Tabelle in einer neuen Abfrage verwendet. Die Vergabe eines Aliasnamens (im Beispiel "a1") ist zwingend, da jede Tabelle in FROM einen Namen haben muß!

Unterabfragen in FROM werden auch "abgeleitete Tabellen" (*derived tables*) genannt.

Unterabfragen in FROM kann man auch umgehen, wenn man *temporäre* Tabellen verwendet.

Für die Abfrage auf der Folie sähe das folgendermaßen aus:

•CREATE TABLE temp as (
  SELECT abtnummer FROM standort
  GROUP BY abtnummer
  HAVING COUNT(*) = 1);

•SELECT m.nachname
FROM mitarbeiter m, temp a1
WHERE m.abtnr = a1.abtnummer;

•DROP TABLE temp;

Zusammenfassung

- **SELECT Abfrage**
  - besteht aus bis zu 6 Abschnitten
  - bis auf SELECT und FROM sind alle Abschnitte optional
  - die Reihenfolge muß eingehalten werden
  - Aggregatfunktionen (AVG, SUM, MAX, MIN, COUNT) können mit GROUP BY und HAVING verwendet werden

```
SELECT <attribut-und-funktionen-liste>
FROM <tabellen-liste oder join-ausdruck>
[WHERE <bedingung>]
[GROUP BY <gruppierungs-attribute>]
[HAVING <gruppen-bedingung>]
[ORDER BY <attribut-list>];
```

Datenbanken I | 14 | SS 05, 29.04.2005 | FH Darmstadt, FB Informatik, Dr. P. Nevermann

Konzeptuell gesprochen, werden SQL SELECT Abfragen in folgender Reihenfolge abgearbeitet:

1. FROM: Bestimmung der beteiligten Tabellen (auch temporärer Natur wie z.B. Joins, Unterabfragen)
2. WHERE: Selektieren der Ergebnis-Tupel
3. GROUP-BY und HAVING: Gruppenbildung und Selektieren der Ergebnis-Gruppen
4. ORDER BY: Sortierung des Endergebnisses

In der Regel gibt es mehrere alternative Formulierungen für ein und dieselbe Abfrage (z.B. "traditionelle" Verbund Schreibweise ⇔ Verbund mit JOIN in FROM (joined tables) ⇔ Unterabfrage in WHERE mit IN-Operator statt Verbund). Theoretisch sollte es egal sein, welche Schreibweise verwendet wird. In der Realität hängt aber leider oft viel davon ab, weil das DBMS nicht alle Formen mit gleicher Effizienz verarbeitet.

SQL ist eine sehr mächtige Abfragesprache, mit hohem Abdeckungsgrad in der Praxis, die sich seit ca. 30 Jahren gehalten hat. Die meisten DBMS implementieren heute raffinierte Optimierungsstrategien um SELECT Abfragen möglichst effizient beantworten zu können.

Trotzdem gibt es auch Dinge, die man mit SQL nicht so gut machen kann:

- Rekursive Abfragen beliebiger Tiefe sind nicht möglich (z.B. kann man nicht mit einer einzigen SQL-Abfrage zu einem Mitarbeiter die Liste aller Vorgesetzten und Vor-Vorgesetzten bis hin zum Vorstand bekommen, wenn man vorher nicht weiß, wieviele Hierarchie-Ebenen es oberhalb des Mitarbeiters gibt).
- Unbequem zu formulieren sind Abfragen der Art: Wer verdient am zehnt-besten?

SQL INSERT

- **Frau Anna Voss ist neu eingestellt worden. Sie ist am 2.3.1979 geboren und wohnt im Ida-Weg 11 in Mainz. Sie arbeitet in der Abteilung "Forschung" und berichtet an Herrn Mayer. Das Einstiegsgehalt ist € 45.000.**

```
INSERT INTO mitarbeiter
VALUES ('Voss', 'Anna', 9999, '1979-03-02',
   'Ida-Weg 11, Mainz', 'W', 45000, 1111, 2);
```

Datenbanken I | 15 | SS 05, 29.04.2005 | FH Darmstadt, FB Informatik, Dr. P. Nevermann

Mit SQL SELECT haben wir bisher nur lesend auf die Datenbank zugegriffen. Um schreibend auf Tabellen zuzugreifen kennt SQL drei Anweisungen: INSERT (Einfügen), UPDATE (Ändern) und DELETE (Löschen).

INSERT:

•In seiner einfachsten Form wird ein einzelnes Tupel in die Tabelle eingefügt (siehe Beispiele auf der Folie). Attributwerte für alle Spalten müssen in der definierten Reihenfolge angegeben werden.

•Eine zweite Form erlaubt Attributwerte nur für ausgewählte Spalten anzugeben:
Beispiel:
Auch Herr Elmar Quantz wird eingestellt, allerdings sind Geburtsdatum, Adresse, Gehalt, Abteilung und Vorgesetzter noch unbekannt.
INSERT INTO mitarbeiter(pnr, nachname, vorname, geschlecht)
VALUES (1234, 'Quantz', 'Elmar', 'M');

Die Reihenfolge der Attibute kann jetzt selbst bestimmt werden. Attribute für die kein Wert angegeben wurde, werden auf NULL (bzw. auf einen definierten Initialwert) gesetzt, was natürlich nur dann gut geht, wenn die Tabellendefinition NULL-Werte für die Spalte erlaubt (bzw. Initialwerte vorgibt).

Auf VALUES können auch mehrere, durch ',' getrennte Tupel folgen.

Beispiel:

•INSERT INTO mitarbeiter(pnr, nachname, vorname, geschlecht)
VALUES (1234, 'Quantz', 'Elmar', 'M'),
(1235, 'Quentz', 'Alma', 'W'),
(1236, 'Quintz', 'Wilma', 'W');

Eine weitere Form von INSERT erlaubt Tupel einzufügen, die man als Ergebnis einer Abfrage bekommen hat.

Beispiel:

•Zuerst wird eine temporäre Tabelle geschlecht erzeugt, die dann mit allen unterschiedlichen Werten des geschlecht Attributs von mitarbeiter gefüllt wird:

```
CREATE TABLE geschlecht (
   sex char(1)
);
INSERT INTO geschlecht
   SELECT DISTINCT geschlecht FROM mitarbeiter;
```

SQL UPDATE

- **Jubel!!!**
  Alle Mitarbeiter (Führungskräfte ausgenommen!) bekommen eine Gehaltserhöhung von 15%.

```
UPDATE mitarbeiter AS m1
SET m1.gehalt = m1.gehalt * 1.15
WHERE NOT EXISTS
    (SELECT * FROM mitarbeiter AS m2
     WHERE m1.pnr = m2.vpnr);
```


UPDATE:

•Dient zum Ändern von Attributwerten eines oder mehrerer Tupel in einer Tabelle

•In der SET-Klausel werden die Attribute mit ihren neuen Werten angegeben (SET a1 = v1, a2 = v2, ...)

•Wie das Beispiel der Folie zeigt, kann man sich in der SET-Klausel auch auf ursprüngliche Attributwerte beziehen (z.B. das aktuelle Gehalt)

SQL DELETE

- **Leider mußte Frau Voss frühzeitig das Unternehmen wieder verlassen, da ihr anderweitig ein besseres Angebot gemacht wurde.**

```
DELETE FROM mitarbeiter
WHERE nachname = 'Voss' AND vorname = 'Anna';
```


DELETE:

•Es werden alle durch die WHERE-Bedingung selektierten Tupel gelöscht

•In WHERE stehen alle aus SELECT bekannten Mittel zur Verfügung

•*Explizit* können mit einer DELETE Anweisung nur Tupel aus einer Tabelle gelöscht werden. [Später werden wir sehen, daß das Löschen eines Tupels aus einer Tabelle das *implizite* Löschen von anderen Tupeln (möglicherweise anderer Tabellen) nach sich ziehen kann ➔ *referentielle Integrität*]

•Fehlt die WHERE-Klausel, so werden alle Tupel aus der Tabelle gelöscht!
Beispiel:
Leider ist die Firma pleite gegangen und alle Mitarbeiter mußten entlassen werden ☹
DELETE FROM mitarbeiter;

Die Tabelle selbst existiert weiterhin, ist aber jetzt leer.

Ausblick

- **Wie entwirft man eine Datenbank für eine vorgegebene Miniwelt?**
  - Semantische Datenmodellierung mit dem Entity-Relationship Modell