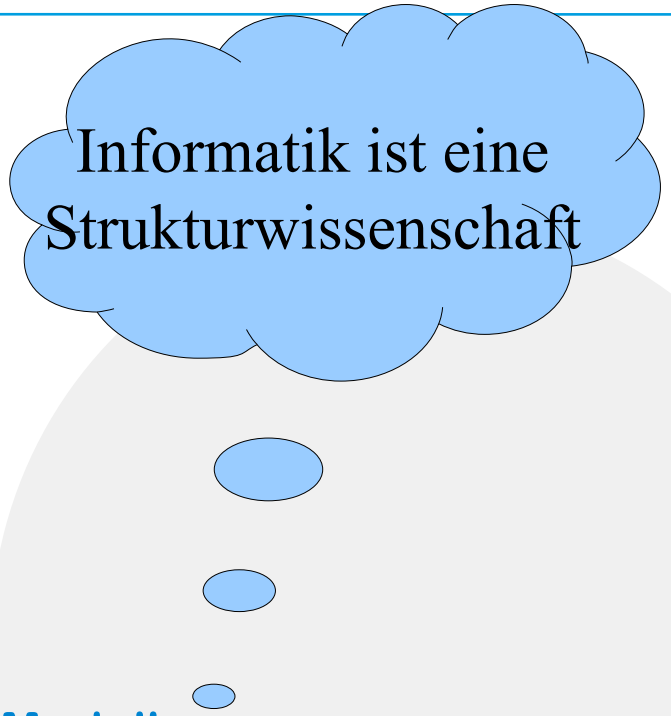
Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences



Teil II

Von Daten und ihren Modellen

Robert Hartmann (SoSe 2024)

basierend auf Folien von
Prof. Dr. Harm Knolle

Fachbereich Informatik
Hochschule Bonn-Rhein-Sieg

Theta-Join, Equi-Join (Natürlicher Verbund) versus NATURAL JOIN

```
Anfrage :=
  SELECT   <Tupel>
  FROM     <Tabellen>              [
  WHERE    <Prädikat>]             [
  ORDER BY <Sortierung>            ]
```

Equi-Join: ⋈

Natural-Join: ⊗

Mit Theta-Join als Equi-Join:

```
Tabellen :=
          {   <Tabelle> [[AS] <Alias>]
[INNER] JOIN <Tabelle> [[AS] <Alias>]
           ON <PRÄDIKAT> }
```

`Prädikat` nutzt den Vergleich auf Gleichheit mit Operator =

Mit Equi-Join (zu Deutsch Gleichheitsverbund oder Natürlicher Verbund) :

```
Tabellen :=
          {   <Tabelle> [[AS] <Alias>]
[INNER] JOIN <Tabelle> [[AS] <Alias>]
        USING <TUPEL> }
```

Mit Natural Join:

```
Tabellen :=
          {      <Tabelle> [[AS] <Alias>]
   NATURAL JOIN <Tabelle> [[AS] <Alias>]}
```

Verbund von Tabellen

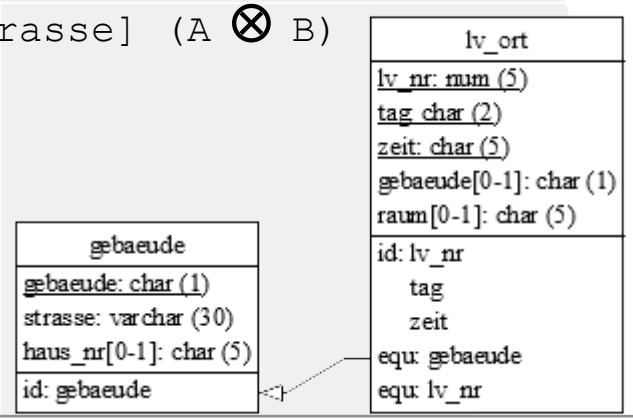
Theta-Join, Equi-Join (Natürlicher Verbund) versus NATURAL JOIN

Gesucht: In welchen Straßen finden welche Lehrveranstaltungen statt? (Gewünscht sind Straßennamen und Lehrveranstaltungsnummern.)

Equi-Join: ⋈ , Natural-Join: ⊗

A = gebaeude
B = lv_ort

1. π[lv_nr,strasse] (A $\bowtie_{(A.gebaeude=B.gebaeude)}$ B)
2. π[lv_nr,strasse] (A $\bowtie_{(gebaeude)}$ B)
3. π[lv_nr,strasse] (A $\otimes$ B)



1.
Mit Theta-Join als Equi-Join [Vergleichsoperator ist =]

```
SELECT o.lv_nr , g.strasse
FROM lv_ort o INNER JOIN gebaeude g
ON o.gebaeude = g.gebaeude;
```

2.
Equi-Join mit „syntactic sugar“ :

```
SELECT o.lv_nr , g.strasse
FROM lv_ort o INNER JOIN gebaeude g
USING (gebaeude);
```

3.
Mit Natural Join:

```
SELECT lv_nr , strasse
FROM lv_ort NATURAL JOIN gebaeude ;
```

Der Equi-Join wird in deutschen Quellen bezeichnet als „Gleichheitsverbund“ oder „Natürlicher Verbund“

- Verdichtung von Daten -

Inhalt

- Einführung
- Theoretische Grundlagen relationaler Sprachen
- Relationenalgebra
- Datenbanksprache SQL
  - Allgemeines zu SQL
  - SQL DML - Datenanfrage
    - Projektion
    - Formatierung
    - Selektion
    - Verbund von Tabellen
    - Verdichtung von Daten
    - Unterabfragen
  - SQL DML - Datenmanipulation

Überblick

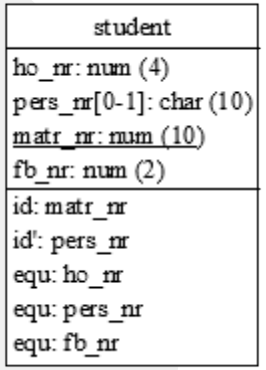
- Aggregatfunktionen
- Gruppenbildung

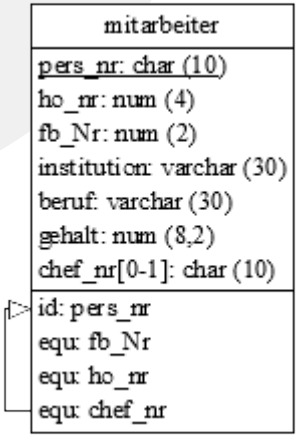
Verdichtung von Daten

- Aggregatfunktionen -

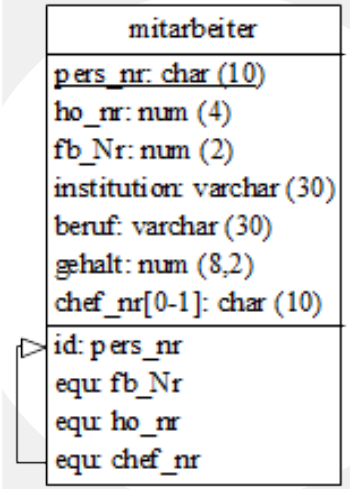
```
Anfrage :=
  SELECT   <Tupel>
  FROM     <Tabellen>                [
  WHERE    <Prädikat>]               [
  ORDER BY <Sortierung>              ]

Tupel :=
  {[DISTINCT] <Spalten>              |
  [COUNT ([DISTINCT] <Spalten>)      }

Spalten :=
  {*                                 |
  {<Ausdruck>                        |
  {COUNT | MIN | MAX | AVG | SUM }
  (<Ausdruck>)                       }
  [AS <neue Spalte>] [, <Spalten>]

Ausdruck :=
  [{<Tabelle>. | <Alias>.}]
  {<Spalte> | <Wert>} [⊗ <Ausdruck>]
```

```
SELECT   COUNT(*)
FROM     dbs_tab_student
WHERE    fb_nr = 2 ;

SELECT   COUNT(pers_nr)
FROM     dbs_tab_student
WHERE    fb_nr = 2 ;

SELECT   COUNT(DISTINCT fb_nr)
FROM     dbs_tab_student ;

SELECT   MAX(gehalt),
         MIN(gehalt)
FROM     dbs_tab_mitarbeiter ;

SELECT   AVG(gehalt),
         SUM(gehalt)
FROM     dbs_tab_mitarbeiter ;

SELECT   COUNT(MIN(gehalt))
FROM     dbs_tab_mitarbeiter ;

-- Verschachtelung
-- funktioniert nicht
```





- Aggregatfunktionen -



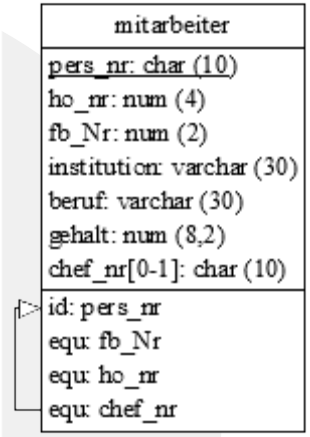

```
SELECT MAX(gehalt), Count(gehalt)
FROM mitarbeiter
```

Verdichtung von Daten

- Gruppenbildung -

```
Anfrage :=
  SELECT    <Tupel>
  FROM      <Tabellen>                  [
  WHERE     <Prädikat> ]                [
  GROUP BY <Ausdruck> [
  HAVING    <Prädikat> ]]               [
  ORDER BY <Sortierung>                 ]

Ausdruck :=
  [{<Tabelle>. | <Alias>.}]
  {<Spalte> | <Wert>} [⊗ <Ausdruck>]

Prädikat:=
  {<Bedingung>                          |
   NOT (<Prädikat>)                     |
   <Bedingung> AND <Prädikat>           |
   <Bedingung> OR  <Prädikat>           }

Bedingung :=
  {<Ausdruck> [ θ <Ausdruck>]           |
  . . .                                 }
```

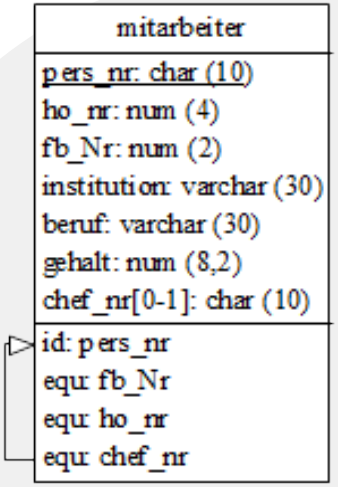
```
SELECT    fb_nr,
          AVG(gehalt)
FROM      dbs_tab_mitarbeiter
GROUP BY  fb_nr ;

SELECT    fb_nr,
          COUNT(pers_nr)
FROM      dbs_tab_mitarbeiter
GROUP BY  fb_nr ;
```




```
SELECT    fb_nr, COUNT(pers_nr) AS Anzahl
FROM      dbs_tab_mitarbeiter
GROUP BY  fb_nr
HAVING    COUNT(pers_nr) < 9
ORDER BY  fb_nr DESC;

SELECT    fb_nr, COUNT(pers_nr) AS Anzahl
FROM      dbs_tab_mitarbeiter
GROUP BY  fb_nr
HAVING    Anzahl < 9
ORDER BY  fb_nr DESC ;
-- funktioniert unter Oracle nicht
```

- Gruppenbildung -

```
SELECT COUNT(pers_nr) AS Anzahl
FROM mitarbeiter
GROUP BY fb_nr
HAVING COUNT(pers_nr) < 9 ;

SELECT fb_nr, COUNT(pers_nr) AS Anzahl
FROM mitarbeiter
GROUP BY fb_nr
HAVING COUNT(pers_nr) < 9 ;
```




```
SELECT fb_nr, institution, COUNT(pers_nr) AS Anzahl
FROM mitarbeiter
GROUP BY fb_nr
HAVING COUNT(pers_nr) < 9 ;

SELECT fb_nr, institution, COUNT(pers_nr) AS Anzahl
FROM mitarbeiter
GROUP BY fb_nr, institution
HAVING COUNT(pers_nr) < 9 ;
```

Standard versus Implementation

SQL window functions, which where standarized in SQL:2003
and covered by SQLite3 with version 3.25.0 dated to 2018-09-15.

But, because of some bugs inside SQLite3,
you cannot use that first version supporting SQL window functions, too.
Please use SQLite3 Version 3.38.5 from 2022-05-06.

A window function performs a calculation across a set of table rows that are somehow related to the current row.
This is comparable to the type of calculation that can be done with an aggregate function.
but unlike regular aggregate functions,
use of a window function does not cause rows to become grouped into a single output row
— the rows retain their separate identities.

Behind the scenes, the window function is able to access more than just the current row of the query result.

- https://mode.com/sql-tutorial/sql-window-functions/
- https://www.postgresql.org/docs/current/tutorial-window.html
- https://docs.oracle.com/cd/E17952_01/mysql-8.4-en/window-functions-usage.html
- https://docs.oracle.com/en/database/oracle/oracle-database/19/sqlrf/Analytic-Functions.html

The most practical example of this is a running total:

```
SELECT start_terminal
        , duration_seconds
        , SUM(duration_seconds) OVER ( PARTITION BY start_terminal  ORDER BY start_time) AS running_total
  FROM tournament ;
```

- Unterabfragen -

Inhalt

- Einführung
- Theoretische Grundlagen relationaler Sprachen
- Relationenalgebra
- Datenbanksprache SQL
  - Allgemeines zu SQL
  - SQL DML - Datenanfrage
    - Projektion
    - Formatierung
    - Selektion
    - Verbund von Tabellen
    - Verdichtung von Daten
    - Unterabfragen
  - SQL DML - Datenmanipulation

Überblick

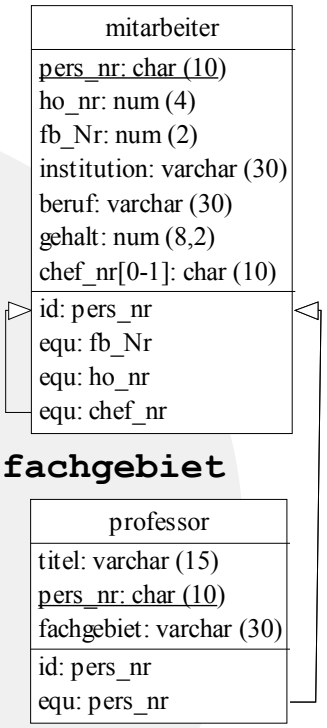
- 'IN'-Operator
- Vergleichs-Operator
- 'EXISTS'-Operator
- Outer-Join realisiert mit Existenzbedingungen

Unterabfragen

- 'IN'-Operator -

```
Anfrage :=
  SELECT    <Tupel>
  FROM      <Tabellen>                [
  WHERE     <Prädikat>]               [
  . . .                               ]

Prädikat:=
  {<neue Bedingung>                   |
   NOT (<neue Bedingung>)             |
   <neue Bedingung> AND <Prädikat>    |
   <neue Bedingung> OR  <Prädikat>    }

neue Bedingung :=
  {<Bedingung>                        |
  <Ausdruck> IN (<Anfrage>)           }

Ausdruck :=
  [{<Tabelle>. | <Alias>.}]
  {<Spalte> | <Wert>} [⊗ <Ausdruck>]
```

```
SELECT *
FROM   dbs_tab_professor
WHERE  pers_nr IN (
  SELECT pers_nr
  FROM   dbs_tab_mitarbeiter
  WHERE  gehalt > 5300 );

SELECT p.titel, p.pers_nr, p.fachgebiet
FROM   dbs_tab_professor p,
       dbs_tab_mitarbeiter m
WHERE  p.pers_nr = m.pers_nr
AND    gehalt > 5300 ;

SELECT *
FROM   dbs_tab_professor
WHERE  pers_nr NOT IN (
  SELECT pers_nr
  FROM   dbs_tab_mitarbeiter
  WHERE  gehalt > 5300 ) ;

SELECT p.titel, p.pers_nr, p.fachgebiet
FROM   dbs_tab_professor p,
       dbs_tab_mitarbeiter m
WHERE  p.pers_nr = m.pers_nr
AND    gehalt <= 5300 ;
```

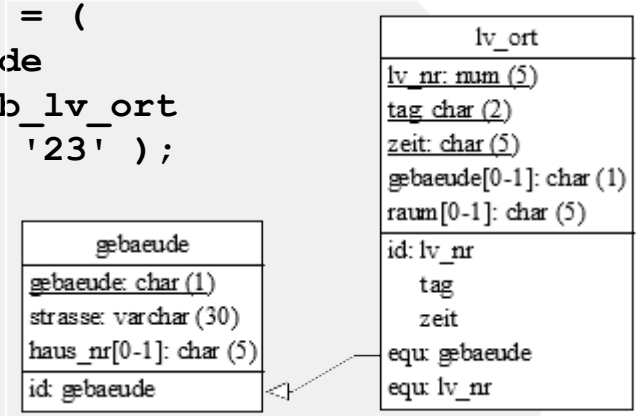


Unterabfragen

- Vergleichs-Operatoren „=“, ALL, ANY -

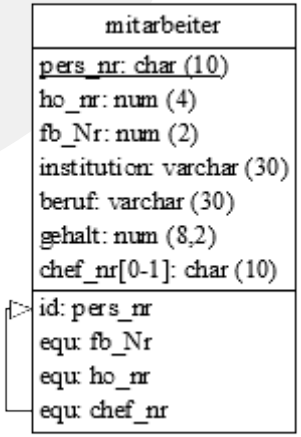
```
Anfrage :=
  SELECT    <Tupel>
  FROM      <Tabellen>                  [
  WHERE     <Prädikat>]                 [
  . . .                                 ]

Prädikat:=
  {<neue Bedingung>                     |
   NOT (<neue Bedingung>)               |
   <neue Bedingung> AND <Prädikat>      |
   <neue Bedingung> OR  <Prädikat>      }

neue Bedingung :=
  {<Bedingung>                          |
  <Ausdruck>
  θ [ALL | ANY] (<Anfrage>)             }

Ausdruck :=
  [{<Tabelle>. | <Alias>.}]
  {<Spalte> | <Wert>} [⊗ <Ausdruck>]
```

```
SELECT *
FROM   dbs_tab_gebaeude
WHERE  gebaeude = (
  SELECT gebaeude
  FROM   dbs_tab_lv_ort
  WHERE  raum = '23' );
```



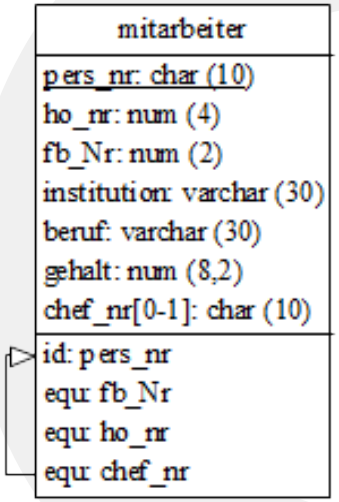

```
SELECT *
FROM   dbs_tab_mitarbeiter
WHERE  gehalt >= ALL (
  SELECT gehalt
  FROM   dbs_tab_mitarbeiter);
```

```
SELECT *
FROM   dbs_tab_mitarbeiter
WHERE  gehalt > ANY (
  SELECT gehalt
  FROM   dbs_tab_mitarbeiter);
```



- Vergleichs-Operator -

Welches Ergebnis wird diese Anfrage liefern?




```
SELECT *
FROM mitarbeiter
WHERE gehalt > ALL ( SELECT gehalt
                       FROM mitarbeiter
                   );
```

Unterabfragen

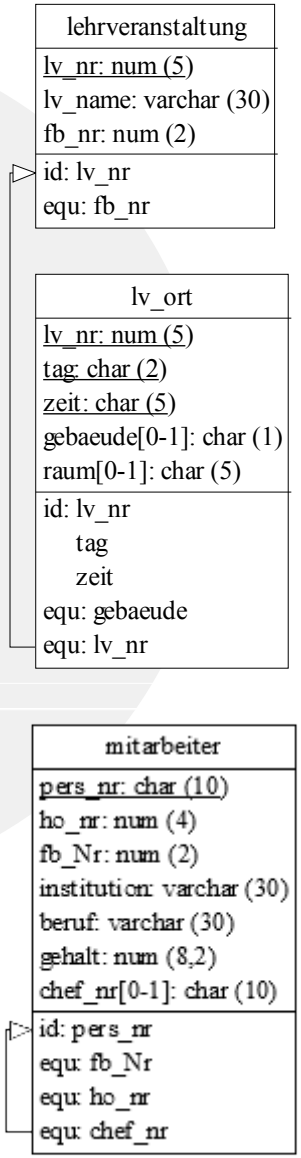
- 'EXISTS'-Operator (Existenzquator ∃)-

```
Anfrage :=
  SELECT    <Tupel>
  FROM      <Tabellen>                  [
  WHERE     <Prädikat>]                 [
  . . .                                 ]

Prädikat:=
  {<neue Bedingung>                     |
   NOT (<neue Bedingung>)               |
   <neue Bedingung> AND <Prädikat>      |
   <neue Bedingung> OR  <Prädikat>      }

neue Bedingung :=
  {<Bedingung>                          |
  EXISTS (<Anfrage>)                    }
```

Korrelierende Unterabfrage

- Prädikat der inneren Abfrage ist abhängig von der aktuellen Ergebniszeile der äußeren Abfrage

```
SELECT *
FROM   dbs_tab_lehrveranstaltung lv
WHERE  NOT EXISTS (
   SELECT *
   FROM   dbs_tab_lv_ort o
   WHERE  o.lv_nr = lv.lv_nr );

SELECT pers_nr
FROM   dbs_tab_mitarbeiter chef
WHERE  EXISTS (
   SELECT *
   FROM   dbs_tab_mitarbeiter m
   WHERE  chef.pers_nr = m.chef_nr
   AND    m.gehalt > chef.gehalt );

SELECT DISTINCT chef.pers_nr
FROM   dbs_tab_mitarbeiter chef,
       dbs_tab_mitarbeiter m
WHERE  chef.pers_nr = m.chef_nr
AND    m.gehalt > chef.gehalt;
```



- ALLquantor ∀ durch 'EXISTS'-Operator -

```
Anfrage :=
  SELECT    <Tupel>
  FROM      <Tabellen>                  [
  WHERE     <Prädikat>]                 [
  . . .                                 ]

Prädikat:=
  {<neue Bedingung>                     |
   NOT (<neue Bedingung>)               |
   <neue Bedingung> AND <Prädikat>      |
   <neue Bedingung> OR  <Prädikat>      }

neue Bedingung :=
  {<Bedingung>                          |
  EXISTS (<Anfrage>)                    }
```

Korrelierende Unterabfrage

- Prädikat der inneren Abfrage ist abhängig von der aktuellen Ergebniszeile der äußeren Abfrage

SQL unterstützt keinen Allquantor im Sinne der Prädikatenlogik.
Das SQL-Schlüsselwort ALL entspricht leider nicht den Anforderungen an einen Allquantor der Prädikatenlogik.

Der SQL-Operator ALL vergleicht nur einen Wert mit einer Menge, während der Allquantor Bedingungen für alle Elemente einer Menge festlegt.

Daher: Bekannte Äquivalenz nutzen

$$\forall x\,(\theta(x)) \;=\; \neg\exists x(\neg\theta(x))$$

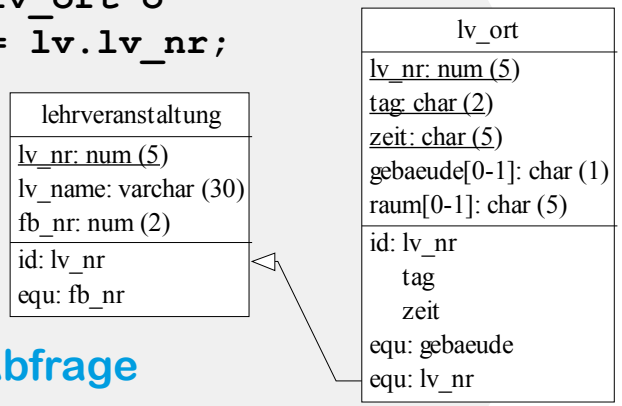
„Für alle x gilt θ(x).“
= „Es gibt kein x, für das nicht θ(x) gilt.“

```
SELECT ...
FROM   Tab...
WHERE  NOT EXISTS( SELECT *
                   FROM Tab...
                   WHERE NOT BEDINGUNG(x)
                 );
```

- Outer-Join realisiert mit Existenzbedingungen -

```
Anfrage :=
  SELECT    <Tupel>
  FROM      <Tabellen>                  [
  WHERE     <Prädikat>]                 [
  . . .                                 ]

Prädikat:=
  {<neue Bedingung>                     |
   NOT (<neue Bedingung>)               |
   <neue Bedingung> AND <Prädikat>      |
   <neue Bedingung> OR  <Prädikat>      }

neue Bedingung :=
  {<Bedingung>                          |
  EXISTS (<Anfrage>)                    }
```

```
SELECT lv.lv_nr, lv.lv_name,
       o.tag, o.zeit
FROM   dbs_tab_lehrveranstaltung lv
LEFT OUTER JOIN
       dbs_tab_lv_ort o
ON     o.lv_nr = lv.lv_nr;
```



Als Union einer Abfrage mit Unterabfrage

```
  SELECT lv.lv_nr, lv_name, o.tag, o.zeit
  FROM   dbs_tab_lehrveranstaltung lv,
         dbs_tab_lv_ort o
  WHERE  o.lv_nr = lv.lv_nr
UNION
  SELECT lv.lv_nr, lv_name, NULL, NULL
  FROM   dbs_tab_lehrveranstaltung lv
  WHERE  lv.lv_nr NOT IN (
    SELECT o.lv_nr
    FROM   dbs_tab_lv_ort o );
```

- SQL DML - Datenmanipulation -

Inhalt

- Einführung
- Theoretische Grundlagen relationaler Sprachen
- Relationenalgebra
- Datenbanksprache SQL
  - SQL - Structured Query Language
  - SQL DML - Datenanfrage
  - SQL DML - Datenmanipulation

Überblick

- Veränderung von Relationen
- Einfügen
- Löschen
- Modifikation

SQL DML - Datenmanipulation

- Veränderung von Relationen -

Operationen auf einem Tupel

- z.B. Immatrikulation eines neuen Studenten
- z.B. Adressänderung eines Studenten

Operationen auf mehreren Tupeln

- z.B. Erhöhung des Gehalts aller Mitarbeiter um einen bestimmten Prozentsatz
- z.B. Exmatrikulation eines Studenten im laufenden Semester führt zu Veränderungen im Belegungsplan

Einfügen von Tupel

- `INSERT`

Löschen von Tupel

- `DELETE`

Verändern von Tupel

- `UPDATE`

Vgl auch Kapitel 6 Teil 1 Folie 27

- Veränderung von Relationen -

Operationen auf einem Tupel

INSERT (= Einfügen eines Tupels in einer bestehenden Relationsinstanz)

Student

Name	SSN
Maier	12344

INSERT <„Karl“, 46464> into Student

INSERT <NULL, 46562> into Student

→

Student

Name	SSN
Maier	12344
Karl	46464
NULL	46562

UPDATE (= Modifikation eines Tupels)

Student

Name	SSN
Maier	12344
Schmitz	46562

UPDATE Name = „Franz“ of Student tuple with SSN = 12344

→

Student

Name	SSN
Franz	12344
Schmitz	46562

DELETE (= Löschen eines Tupels)

Student

Name	SSN
Maier	12344
Schmitz	46562

DELETE Student Tupel with SSN = 12344

→

Student

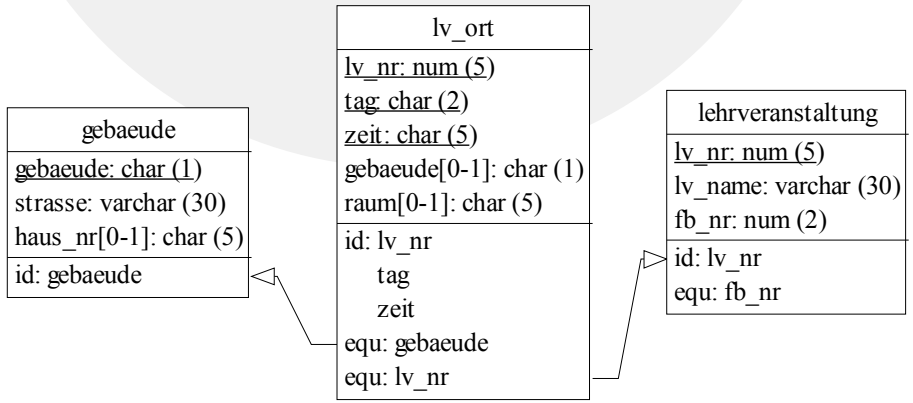
Name	SSN
Schmitz	46562

SQL DML - Datenmanipulation

- Einfügen -

```
Einfügung :=
  INSERT
  INTO     <Tabelle> [<Spalten>]
  <neue Tupel>

neue Tupel :=
  VALUES {<Ausdrücke>}
  | <Anfrage>

Ausdrücke :=
  (<Ausdruck> [, Ausdrücke])

Spalten :=
  (<Spalte> [, <Spalten>])

Ausdruck :=
  [{<Tabelle>. | <Alias>.}]
  {<Spalte> | <Wert>} [⊗ <Ausdruck>]
```

```
INSERT
INTO   dbs_tab_lehrveranstaltung
VALUES (1156,'Marketing 2' ,1);

INSERT
INTO   dbs_tab_gebaeude
VALUES ('G', 'Grantham-Allee','20');

INSERT
INTO   dbs_tab_lv_ort
   (lv_nr, tag, zeit, gebaeude, raum)
SELECT lv_nr, 'Mo', '1600', 'C', '177'
FROM   dbs_tab_lehrveranstaltung
WHERE  lv_name = 'Marketing';
```

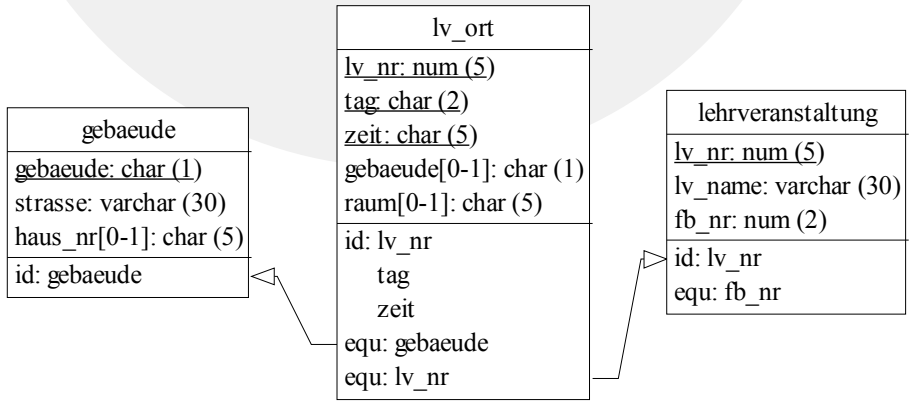


SQL DML - Datenmanipulation

- Einfügen (II) -

```
Einfügung :=
  INSERT
  INTO     <Tabelle> [<Spalten>]
  <neue Tupel>

neue Tupel :=
  VALUES {<Ausdrücke>}
  | <Anfrage>

Ausdrücke :=
  (<Ausdruck> [, Ausdrücke])

Spalten :=
  (<Spalte> [, <Spalten>])

Ausdruck :=
  [{<Tabelle>. | <Alias>.}]
  {<Spalte> | <Wert>} [⊗ <Ausdruck>]
```

```
INSERT
INTO   dbs_tab_lv_ort
VALUES ( ( SELECT lv_nr
           FROM dbs_tab_Lehrveranstaltung
           WHERE lv_name = 'Datenbanksysteme')
       , 'Do'
       , '1600'
       , ( SELECT DISTINCT gebaeude
           FROM dbs_tab_gebaeude
           WHERE strasse = 'Grantham-Allee')
       ,  '177'
       );

-- Achtung: pro Unterabfrage
            maximal ein Tupel!
```



SQL DML - Datenmanipulation

- Löschen -

```
Löschung :=
  DELETE
  FROM      <Tabelle>                [
  WHERE     <Prädikat>               ]
```

```
DELETE
FROM   dbs_tab_student;

DELETE
FROM   dbs_tab_student
WHERE  matr_nr = 807252;

DELETE
FROM   dbs_tab_prof_haelt_lv
WHERE  pers_nr = (
  SELECT pers_nr
  FROM   dbs_tab_professor
  WHERE  fachgebiet = 'Datenbanksysteme');

DELETE
FROM   dbs_tab_prof_haelt_lv
WHERE  pers_nr IN (
  SELECT pers_nr
  FROM   dbs_tab_professor
  WHERE  fachgebiet LIKE 'Daten%');
```

student
ho_nr: num (4) pers_nr[0-1]: char (10) matr_nr: num (10) fb_nr: num (2)
id: matr_nr id': pers_nr equ: ho_nr equ: pers_nr equ: fb_nr

prof_haelt_lv
pers_nr: char (10) lv_nr: num (5) tag: char (2) zeit: char (5)
id: pers_nr lv_nr tag zeit equ: pers_nr equ: lv_nr tag zeit

SQL DML - Datenmanipulation

- Modifikation (I) -

```
Modifikation :=
  UPDATE    <Tabelle>
  SET       <Zuweisung>                [
  WHERE     <Prädikat>                 ]

Zuweisung :=
  <Einzelzuweisung> [, <Zuweisung>]

Einzelzuweisung :=
  {<Spalte> = {<Ausdruck> |
  (<Anfrage>)                }         |
  <Spalten> = (<Anfrage>)               }

Spalten :=
  (<Spalte> [, <Spalten>])

Ausdruck :=
  [{<Tabelle>. | <Alias>.}]
  {<Spalte> | <Wert>} [⊗ <Ausdruck>]
```

```
UPDATE    dbs_tab_lv_ort
SET       Gebaeude = 'C',
          Raum = 231
WHERE     lv_Nr = 1150;


UPDATE    dbs_tab_lv_ort
SET       (gebaeude, raum) = ('C',231)
WHERE     lv_Nr = 1150;

-- funktioniert unter Oracle
-- nicht
```

lv_ort
lv_nr: num (5)
tag: char (2)
zeit: char (5)
gebaeude[0-1]: char (1)
raum[0-1]: char (5)
id: lv_nr tag zeit
equ: gebaeude
equ: lv_nr

SQL DML - Datenmanipulation

- Modifikation (II) -

```
Modifikation :=
  UPDATE    <Tabelle>
  SET       <Zuweisung>                [
  WHERE     <Prädikat>                 ]

Zuweisung :=
  <Einzelzuweisung> [, <Zuweisung>]

Einzelzuweisung :=
  {<Spalte> = {<Ausdruck> |
  (<Anfrage>)                 }        |
  <Spalten> = (<Anfrage>)              }

Spalten :=
  (<Spalte> [, <Spalten>])

Ausdruck :=
  [{<Tabelle>. | <Alias>.}]
  {<Spalte> | <Wert>} [⊗ <Ausdruck>]
```

```
UPDATE    dbs_tab_prof_haelt_lv
SET       pers_nr = (
             SELECT pers_nr
             FROM   dbs_tab_professor
             WHERE  fachgebiet =
                       'Mathematik' )
WHERE     lv_Nr = 1150;

UPDATE    dbs_tab_mitarbeiter
SET       gehalt = gehalt * 1.1;
```