

Informatik ist eine  
Strukturwissenschaft

## 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



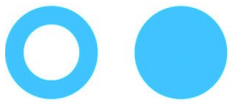
## - SQL - Structured Query Language -

### Inhalt

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

### Überblick

- ♦ Einführung (Whlg.)
- ♦ Anfragen und Sichten (Views)
- ♦ Anfragekonzepte von SQL-DML



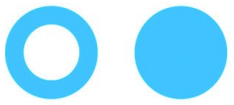
## - Einführung (Whlg.) -

### Sprachliche Fassung des relationalen Datenmodells

- ◆ Konzepte zur Implementierung eines relationalen logischen Modells
- ◆ Sprachgrundlage relationale Algebra
  - DDL - Data Definition Language
  - DML - Data Manipulation Language (dieses Kapitel)
- ◆ Konzepte interne Ebene (Kapitel 8)
  - Datenstrukturdefinition
- ◆ Konzepte konzeptuelle und externe Ebene (Kapitel 5)
  - Tabellen, anwendungsspezifische Sichten (virtuelle Tabellen)
- ◆ Konzepte für Datenbankbetrieb (Vertiefung im Hauptstudium)
  - Transaktion. Mehrbenutzerkontrolle
  - Datenschutz, Datensicherheit
- ◆ zahlreiche Dialekte kommerzieller Systeme

### Normierung

- ◆ SEQUEL: 70er Jahre - System R , IBM
- ◆ SQL-86: 1986 - ANSI/ISO-Standard
  - ISO: International Organisation for Standardisation
  - ANSI: American National Standards Institute
- ◆ SQL-89: 1989 - Erweiterungen zu SQL-86
- ◆ SQL-92 (SQL 2): 1992
  - „derzeitiger“ Standard (fast) aller kommerzieller SQL-Systeme
- ◆ SQL-99: objekt-relationale Grundlagen
- ◆ SQL-2003: objekt-relationaler Standard, XML
- ◆ SQL-2006: Integration von XML
- ◆ SQL-2008: u.a. Optimierung Trigger
- ◆ SQL-2011: u.a. zeitbezogene Daten
- ◆ SQL-2016/19: aktueller (theoretischer) Standard
- ◆ SQL-2023: Unterstützung JSON als Datentyp



## - SQL DML – Datenanfrage -

### Inhalt

- ♦ Einführung
- ♦ Theoretische Grundlagen relationaler Sprachen
- ♦ Relationenalgebra
- ♦ Datenbanksprache SQL
  - Allgemeines zu SQL
  - **SQL DML - Datenanfrage**
  - SQL DML - Datenmanipulation

### Überblick

- ♦ Projektion
- ♦ Formatierung
- ♦ Selektion
- ♦ Verbund von Tabellen
- ♦ Verdichtung von Daten
- ♦ Unterabfragen

## - Anfragen und Sichten (Views) -

### Alle Sichten werden als Anfragen definiert

- ♦ siehe die folgenden Beispiele
- ♦ Anfragen auf Sichten sind somit Anfragen auf Anfragen

### Jede Anfrage lässt sich auch als Sicht definieren

- ♦ siehe Beispiele in Kapitel 5

```
CREATE VIEW <Sicht>  
AS          <Anfrage>
```

Sichtendefinition siehe Kapitel 5

### Alle Anfragen können sich beziehen auf

- ♦ Tabellen der konzeptuellen Ebene und / oder
- ♦ Sichten (Views) der externen Ebene

### Im Folgenden ist der Begriff Tabelle ein Synonym für SQL-Tabelle oder SQL-View !!!

```
Tabelle :=  
{<Name einer Tabelle>      |  
 <Name einer Sicht>       }
```

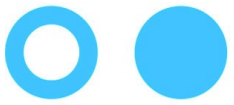
```
CREATE TABLE <Tabelle>  
  <Spaltenbeschreibung>  
Spaltenbeschreibung :=  
  ( <Spalte> <Wertebereich>  
    [, Spaltenbeschreibung] )
```

Tabellendefinition siehe Kapitel 5

```
CREATE TABLE TabLehrveranstaltung  
(   LV_Nr      NUMERIC(4)  
  , LV_Name    VARCHAR(30)  
  , Gebaeude   CHAR(1)  
  , Raum       VARCHAR(5)  
) ;
```

```
INSERT  
INTO   TabLehrveranstaltung  
VALUES (2024, 'Datenbanken', 'C', '116')
```

Weiteres zu Einfügen, Löschen und Modifikation  
folgt später in Kapitel 6 Teil 3.



## - Anfragekonzepte von SQL-DML -

### Projektion

- ♦ Angabe von Spalten

### Formatierung

- ♦ Umbenennung von Spalten
- ♦ künstliche Spalten
- ♦ Sortierung von Zeilen
- ♦ Entfernen von doppelten Zeilen (Duplikate)

### Selektion

- ♦ einfache Bedingung
- ♦ logische Verknüpfung von Bedingungen
- ♦ Vergleich mit Mengen
- ♦ Vergleich mit Suchmuster
- ♦ Vergleich mit 'NULL'-Werten

### Verbund von Tabellen

- ♦ Equi-Join
- ♦ Rekursiver Equi-Join
- ♦ Equi-Join mit 'NULL'-Werten (Outer Join)
- ♦ Equi-Join mit mehr als zwei Tabellen
- ♦ Vereinigung / Durchschnitt / Differenz

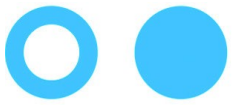
===== in Kapitel 6 Teil 3 : =====

### Verdichtung von Daten

- ♦ Aggregatfunktionen
- ♦ Gruppenbildung

### Geschachtelte Anfragen (Unterabfragen)

- ♦ 'IN'-Operator
- ♦ Vergleichs-Operatoren
- ♦ 'EXISTS'-Operator
- ♦ Outer-Join realisiert m. Existenzbedingungen



## - Projektion -

```
Anfrage :=  
  SELECT  <Tupel>  
  FROM    <Tabelle>
```

```
Tupel :=  
  { *                |  
    {<Spalte>} [, <Tupel>] }
```

**SELECT entspricht  $\pi$**

### Alle Spalten einer Tabelle

```
SELECT  *  
FROM    dbs_tab_student
```

### Bestimmte Spalten einer Tabelle

```
SELECT  matr_nr  
FROM    dbs_tab_student
```

```
SELECT  matr_nr, fb_nr  
FROM    dbs_tab_student
```

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



## - Formatierung -

### 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

- ♦ Spalten
- ♦ Zeilen

## - Spalten -

```
Anfrage :=
    SELECT    <Tupel>
    FROM      <Tabelle>

Tupel :=
    { *
      {<Ausdruck>} [AS <neue Spalte>]
    [, <Tupel>]
    }

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

AS entspricht β

### Umbenennung von Spalten

```
SELECT    fb_nr AS Fachbereich
FROM      dbs_tab_student
```

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

### Künstliche Spalten

```
SELECT    'Text' AS TEXT
FROM      dbs_tab_student
```

```
SELECT    'Matrikelnummer:', matr_nr,
          'Fachbereich:', fb_nr
FROM      dbs_tab_student
```

### Berechnen von Spalten

```
SELECT    pers_nr,
          Beruf,
          Gehalt*13
          AS Jahresgehalt
FROM      dbs_tab_mitarbeiter
```

mitarbeiter
pers_nr: char (10)
ho_nr: num (4)
fb_Nr: num (2)
institution: varchar (30)
beruf: varchar (30)
gehalt: num (8,2)
chef_nr[0-1]: char (10)
id: pers_nr
equ: fb_Nr
equ: ho_nr
equ: chef_nr

## - Zeilen -

### Anfrage :=

```
SELECT  <Tupel>
FROM    <Tabelle>
ORDER BY <Sortierung>      ]
```

### Tupel :=

```
[DISTINCT] { *
{<Ausdruck>} [AS <neue Spalte>]
[, <Tupel>] }
```

### Ausdruck :=

```
{<Spalte> | <Wert>} [⊗ <Ausdruck>]}
```

### Sortierung :=

```
{<Spalte> | <Spalten-Nr>}
{ASC | DESC} [, Sortierung]}
```

### Ausblenden identischer Zeilen

```
SELECT  DISTINCT fb_nr
FROM    dbs_tab_student
```

### Sortierung der Zeilen

```
SELECT  DISTINCT fb_nr
FROM    dbs_tab_student
ORDER BY fb_nr DESC
```

```
SELECT  pers_nr, matr_nr, fb_nr
FROM    dbs_tab_student
ORDER BY fb_nr DESC, matr_nr ASC
```

```
SELECT  pers_nr, matr_nr, fb_nr
FROM    dbs_tab_student
ORDER BY 3 DESC, 2 ASC
```

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



## - Selektion -

### 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

- ♦ Einfache Bedingung
- ♦ Mehrfache Bedingung
- ♦ 'IN'-Operator
- ♦ Vergleich mit einem Muster
- ♦ Vergleich mit 'NULL'-Werten

## - Einfache Bedingung -

```
Anfrage :=
  SELECT   <Tupel>
  FROM     <Tabelle>
  WHERE    <Prädikat>
  ORDER BY <Sortierung>

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

Bedingung :=
  <Ausdruck> [ θ <Ausdruck>]

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

WHERE entspricht  $\sigma$

$\theta \in \{=, \neq, <, >, \leq, \geq\}$

### Tupel, die einer Bedingungen genügen

```
SELECT matr_nr, fb_nr
FROM   dbs_tab_student
WHERE  fb_nr = 2
```

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

```
SELECT matr_nr, fb_nr
FROM   dbs_tab_student
WHERE  NOT (fb_nr = 2)
```

```
SELECT matr_nr, fb_nr
FROM   dbs_tab_student
WHERE  fb_nr != 2
```

mitarbeiter
<u>pers_nr: char (10)</u>
ho_nr: num (4)
fb_Nr: num (2)
institution: varchar (30)
beruf: varchar (30)
gehalt: num (8,2)
chef_nr[0-1]: char (10)
id: pers_nr
equ: fb_Nr
equ: ho_nr
equ: chef_nr

```
SELECT pers_nr
FROM   dbs_tab_mitarbeiter
WHERE  gehalt * 13
      >= 100000
```

## - Mehrfache Bedingung -

```
Anfrage :=
  SELECT   <Tupel>
  FROM     <Tabelle>
  WHERE    <Prädikat>
  ORDER BY <Sortierung>

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

Bedingung :=
  <Ausdruck> [ θ <Ausdruck>]

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

### Tupel, die mehreren Bedingungen genügen

```
SELECT *
FROM   dbs_tab_mitarbeiter
WHERE  gehalt >= 2000
AND    fb_nr   = 2
```

```
SELECT *
FROM   dbs_tab_mitarbeiter
WHERE  Gehalt >= 2000
OR     fb_nr   = 2
```

```
SELECT *
FROM   dbs_tab_mitarbeiter
WHERE  NOT (      Gehalt >= 2000
              AND fb_nr   = 2 )
```

```
SELECT *
FROM   dbs_tab_mitarbeiter
WHERE  gehalt * 13 >= 50000
AND    fb_nr      = 2
```

mitarbeiter	
pers_nr:	char (10)
ho_nr:	num (4)
fb_Nr:	num (2)
institution:	varchar (30)
beruf:	varchar (30)
gehalt:	num (8,2)
chef_nr[0-1]:	char (10)
id:	pers_nr
equ:	fb_Nr
equ:	ho_nr
equ:	chef_nr

## - 'IN'-Operator -

```
Anfrage :=
  SELECT   <Tupel>
  FROM     <Tabelle>
  WHERE    <Prädikat>
  ORDER BY <Sortierung>

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

neue Bedingung :=
  {<Bedingung>
  <Ausdruck> IN <Ausdruck Liste> }

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

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

Tupel, die mehreren Bedingungen genügen

```
SELECT  matr_nr, fb_nr
FROM    dbs_tab_student
WHERE   fb_nr IN (1,3)
```

```
SELECT  matr_nr, fb_nr
FROM    dbs_tab_student
WHERE   NOT ( fb_nr IN (1,3)
              OR matr_nr > 806000 )
```

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

## - Vergleich mit einem Muster -

```
Anfrage :=
  SELECT   <Tupel>
  FROM     <Tabelle>
  WHERE    <Prädikat>
  ORDER BY <Sortierung>
```

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

```
neue Bedingung :=
  {<Bedingung>
  <Ausdruck> LIKE <Muster> }
```

```
Muster :=
  { '_' | '%' } <Text> [ <Muster> ] }
```

'\_' := Platzhalter für ein Zeichen

'%' := Platzhalter für Zeichenkette

### Zeichenkette als Platzhalter

```
SELECT   ho_nr, ho_name
FROM     dbs_tab_hochschulangehoeriger
WHERE    ho_name LIKE 'SCH%'
```

```
SELECT   ho_nr, ho_name
FROM     dbs_tab_hochschulangehoeriger
WHERE    ho_name LIKE '%sch%'
```

### Einzelne Zeichen als Platzhalter

```
SELECT   ho_nr, ho_name
FROM     dbs_tab_hochschulangehoeriger
WHERE    ho_name LIKE 'M_er'
```

hochschulangehoeriger
ho_nr: num (4)
ho_name: varchar (30)
id: ho_nr

## - Vergleich mit 'NULL'-Werten -

```
Anfrage :=
  SELECT  <Tupel>
  FROM    <Tabelle>
  WHERE   <Prädikat>
  ORDER BY <Sortierung>
```

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

```
neue Bedingung :=
  {<Bedingung>
  <Ausdruck> IS [NOT] NULL }
```

```
NULL AND  NULL      = FALSE
NULL AND  <Ausdruck> = FALSE
NULL ⊗    <Ausdruck> = FALSE
```

### Suche nach unvollständigen Tupeln

```
SELECT *
FROM   dbs_tab_lv_ort
WHERE  gebaeude IS NULL
```

```
SELECT *
FROM   dbs_tab_lv_ort
WHERE  gebaeude != 'A'
```

```
SELECT *
FROM   dbs_tab_lv_ort
WHERE  gebaeude <> 'A'
OR
      gebaeude IS NULL
```

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

## - Verbund von Tabellen -

### 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

- ♦ Equi-Join (natürlicher Verbund) - vor SQL92
- ♦ Equi-Join (natürlicher Verbund) - ab SQL92
- ♦ Rekursiver Equi-Join - vor SQL92
- ♦ Rekursiver Equi-Join - ab SQL92
- ♦ Equi-Join mit mehr als zwei Tabellen - vor SQL92
- ♦ Equi-Join mit mehr als zwei Tabellen - ab SQL92
- ♦ Equi-Join mit 'NULL'-Werten (Outer Join) - vor SQL92
- ♦ Equi-Join mit 'NULL'-Werten (Outer Join) - ab SQL92
- ♦ Vereinigung / Durchschnitt / Differenz

## - Equi-Join (natürlicher Verbund) - vor SQL92 -

```

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

Tupel :=
  [DISTINCT] {*                |
  {<Ausdruck>} [AS <neue Spalte>]
  [, <Tupel>]                  }

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

Tabellen :=
  {<Tabelle> [<Alias>]}
  [, <Tabellen>]
  
```

```

SELECT  dbs_tab_lv_ort.gebaeude,
        dbs_tab_lv_ort.raum,
        dbs_tab_gebaeude.strasse
FROM    dbs_tab_gebaeude,
        dbs_tab_lv_ort
WHERE   dbs_tab_lv_ort.gebaeude =
        dbs_tab_gebaeude.gebaeude

SELECT  o.gebaeude, o.raum, g.strasse
FROM    dbs_tab_gebaeude g,
        dbs_tab_lv_ort o
WHERE   o.gebaeude = g.gebaeude

SELECT  o.gebaeude, raum, strasse
FROM    dbs_tab_gebaeude g,
        dbs_tab_lv_ort o
WHERE   o.gebaeude
        = g.gebaeude
AND     strasse
        <> 'Grantham-Allee'
  
```

gebaeude
gebaeude: char (1)
strasse: varchar (30)
haus_nr[0-1]: char (5)
id: gebaeude

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

Equi-Join: ⋈ , Natural-Join: ⋉

## - Equi-Join (natürlicher Verbund) - ab SQL92 -

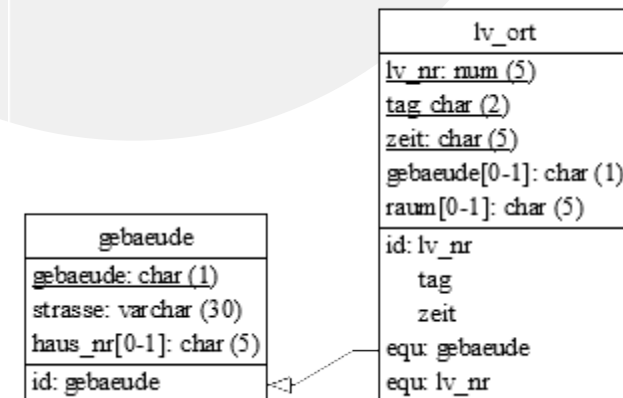
```
Anfrage :=
  SELECT    <Tupel>
  FROM      <Tabellen>          [
  WHERE     <Prädikat>]        [
  ORDER BY  <Sortierung>       ]

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

```
SELECT      dbs_tab_lv_ort.gebaeude,
            dbs_tab_lv_ort.raum,
            dbs_tab_gebaeude.strasse
FROM        dbs_tab_gebaeude
INNER JOIN  dbs_tab_lv_ort
ON          dbs_tab_lv_ort.gebaeude =
            dbs_tab_gebaeude.gebaeude

SELECT      o.gebaeude, o.raum, g.strasse
FROM        dbs_tab_gebaeude g
INNER JOIN  dbs_tab_lv_ort o
ON          o.gebaeude = g.gebaeude
WHERE       strasse <> 'Grantham-Allee'
```

Equi-Join: ⋈ , Natural-Join: ⋈



## - Rekursiver Equi-Join - vor SQL92 -

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

```
Tupel :=
  [DISTINCT] {*
  {<Ausdruck> [AS <neue Spalte>]
  [, <Tupel>]
```

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

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

```
SELECT  m.pers_nr,
        m.gehalt,
        c.pers_nr AS Chef,
        c.gehalt AS Chef_Gehalt
FROM    dbs_tab_mitarbeiter m,
        dbs_tab_mitarbeiter c
WHERE   m.chef_nr = c.Pers_Nr
```

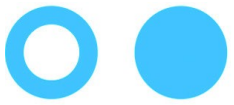
mitarbeiter	
<u>pers_nr</u> :	char (10)
ho_nr:	num (4)
fb_Nr:	num (2)
institution:	varchar (30)
beruf:	varchar (30)
gehalt:	num (8,2)
chef_nr[0-1]:	char (10)
id: pers_nr	
equ: fb_Nr	
equ: ho_nr	
equ: chef_nr	

## - Rekursiver Equi-Join - ab SQL92 -

```
Anfrage :=  
  SELECT  <Tupel>  
  FROM    <Tabellen>          [  
  WHERE   <Prädikat>]        [  
  ORDER BY <Sortierung>      ]  
  
Tabellen :=  
  {<Tabelle> [[AS] <Alias>]  
    [INNER] JOIN <Tabelle> [[AS] <Alias>]  
    ON <PRÄDIKAT> }
```

```
SELECT      m.pers_nr,  
            m.gehalt,  
            c.pers_nr AS Chef,  
            c.gehalt AS Chef_Gehalt  
FROM        dbs_tab_mitarbeiter m  
INNER JOIN  dbs_tab_mitarbeiter c  
ON          m.chef_nr = c.Pers_Nr
```

mitarbeiter	
<u>pers_nr</u> : char (10)	
ho_nr: num (4)	
fb_Nr: num (2)	
institution: varchar (30)	
beruf: varchar (30)	
gehalt: num (8,2)	
chef_nr[0-1]: char (10)	
id: pers_nr	
equ: fb_Nr	
equ: ho_nr	
equ: chef_nr	



$\beta, \times$

<u>Nr</u>	Name	Chef
1	A	2
2	B	NULL
3	C	2

<u>Nr_M</u>	Name_M	Chef_M	<u>Nr_C</u>	Name_C	Chef_C
1	A	2	1	A	2
2	B	NULL	1	A	2
3	C	2	1	A	2
1	A	2	2	B	NULL
2	B	NULL	2	B	NULL
3	C	2	2	B	NULL
1	A	2	3	C	2
2	B	NULL	3	C	2
3	C	2	3	C	2

wird übergangen

wird übergangen

wird übergangen

$\sigma (\text{Chef\_nr} = \text{Nr\_C})$

<u>Nr_M</u>	Name_M	Chef_M	<u>Nr_C</u>	Name_C	Chef_C
1	A	2	2	B	NULL
3	C	2	2	B	NULL

$\pi (\text{Name\_M}, \text{Name\_C})$

Name_M	Name_C
A	B
C	B

## - Equi-Join mit mehr als zwei Tabellen - vor SQL92 (I) -

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

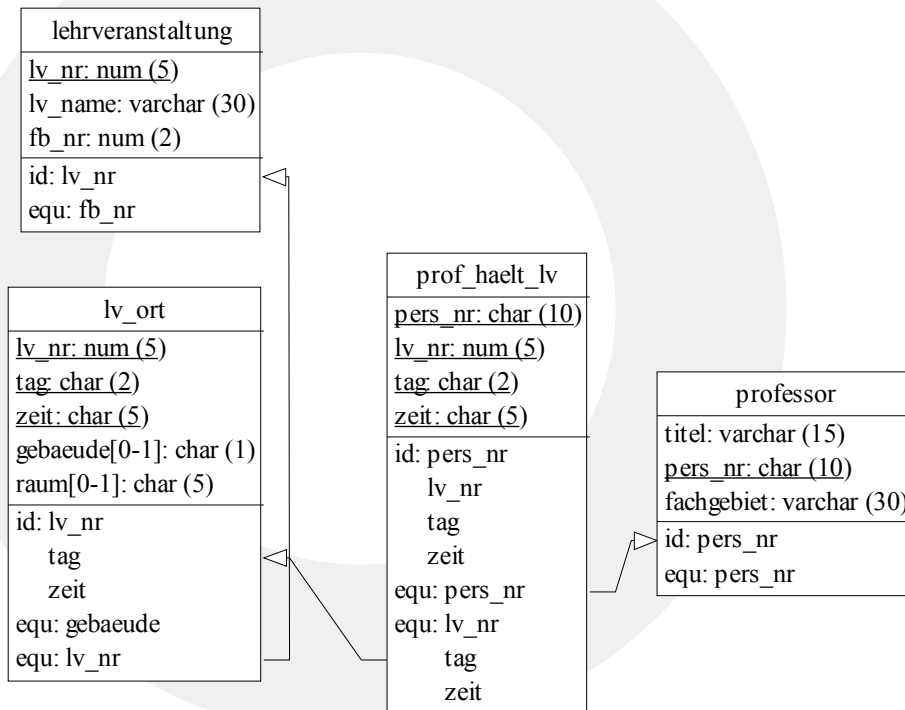
```
Tupel :=  
  [DISTINCT] {*           |  
  {<Ausdruck>} [AS <neue Spalte>]  
  [, <Tupel>]             }
```

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

```
Tabellen :=  
  {<Tabelle> [<Alias>]}  
  [, <Tabellen>]
```

```
SELECT  dbs_tab_professor.fachgebiet,  
        dbs_tab_lehrveranstaltung.lv_name,  
        dbs_tab_lv_ort.gebaeude,  
        dbs_tab_lv_ort.tag,  
        dbs_tab_lv_ort.zeit  
FROM    dbs_tab_professor,  
        dbs_tab_prof_haelt_lv,  
        dbs_tab_lehrveranstaltung,  
        dbs_tab_lv_ort  
WHERE   dbs_tab_professor.pers_nr =  
        dbs_tab_prof_haelt_lv.pers_nr  
AND     dbs_tab_lehrveranstaltung.lv_nr =  
        dbs_tab_lv_ort.lv_nr  
AND     dbs_tab_prof_haelt_lv.tag =  
        dbs_tab_lv_ort.tag  
AND     dbs_tab_prof_haelt_lv.zeit =  
        dbs_tab_lv_ort.zeit  
AND     dbs_tab_prof_haelt_lv.lv_nr =  
        dbs_tab_lv_ort.lv_nr
```

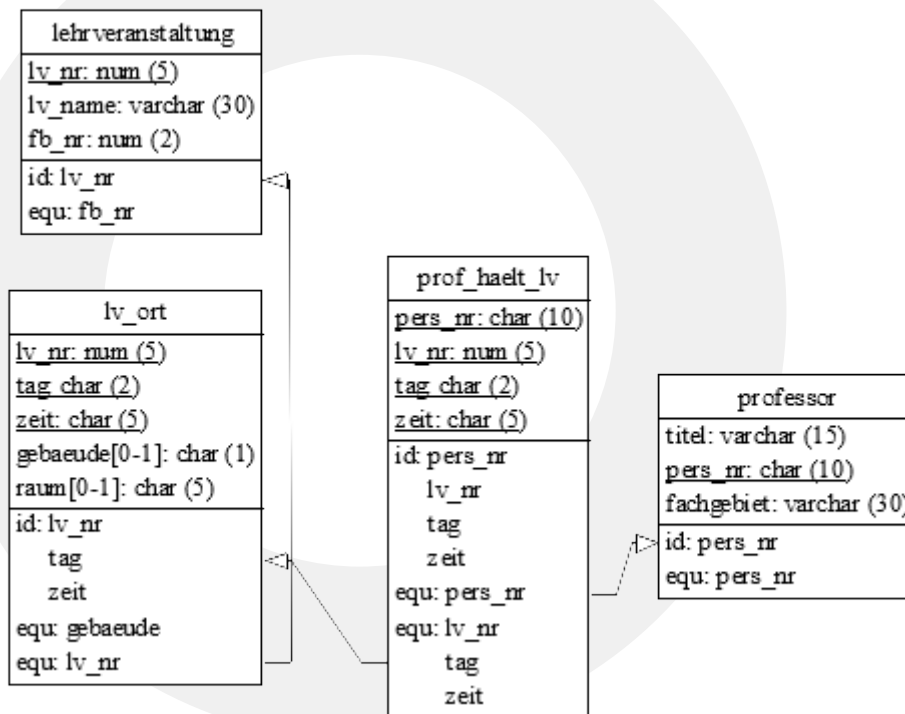
## - Equi-Join mit mehr als zwei Tabellen - vor SQL92 (II) -



```

SELECT    dbs_tab_professor.fachgebiet,
          dbs_tab_lehrveranstaltung.lv_name,
          dbs_tab_lv_ort.gebaeude,
          dbs_tab_lv_ort.tag,
          dbs_tab_lv_ort.zeit
FROM      dbs_tab_professor,
          dbs_tab_prof_haelt_lv,
          dbs_tab_lehrveranstaltung,
          dbs_tab_lv_ort
WHERE     dbs_tab_professor.pers_nr =
          dbs_tab_prof_haelt_lv.pers_nr
AND       dbs_tab_lehrveranstaltung.lv_nr =
          dbs_tab_lv_ort.lv_nr
AND       dbs_tab_prof_haelt_lv.tag =
          dbs_tab_lv_ort.tag
AND       dbs_tab_prof_haelt_lv.zeit =
          dbs_tab_lv_ort.zeit
AND       dbs_tab_prof_haelt_lv.lv_nr =
          dbs_tab_lv_ort.lv_nr
AND       dbs_tab_lehrveranstaltung.lv_name
          LIKE 'Daten%'
AND       dbs_tab_lv_ort.gebaeude = 'C'
    
```

## - Equi-Join mit mehr als zwei Tabellen - vor SQL92 (III) -



```

SELECT  p.fachgebiet,
        lv.lv_name,
        lvo.gebaeude,
        lvo.tag,
        lvo.zeit

FROM    dbs_tab_professor p,
        dbs_tab_prof_haelt_lv plv,
        dbs_tab_lehrveranstaltung lv,
        dbs_tab_lv_ort lvo

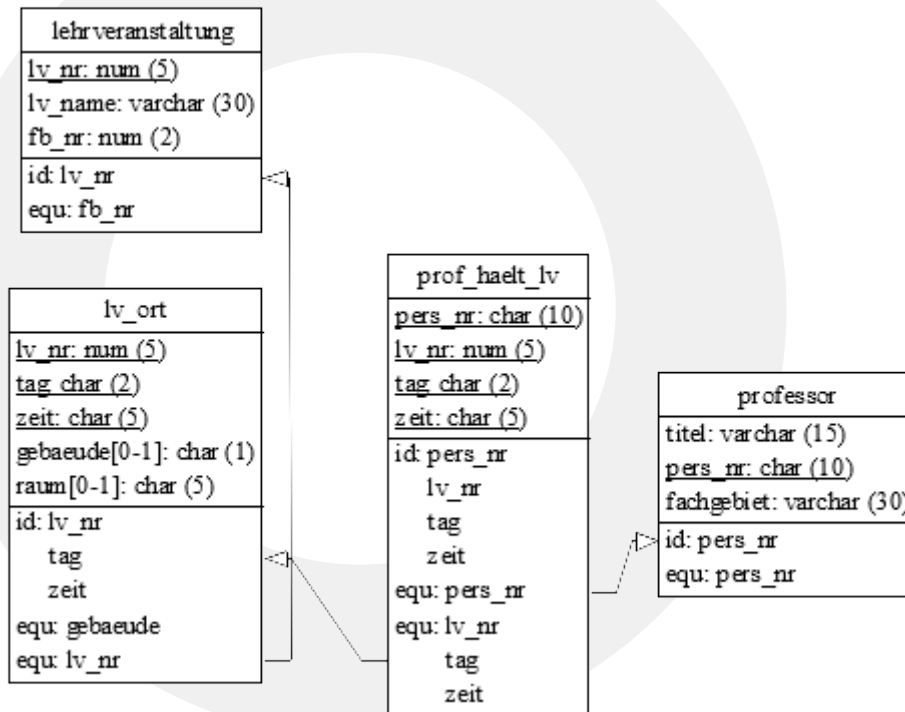
WHERE   P.pers_nr = plv.pers_nr
AND     lv.lv_nr  = lvo.lv_nr
AND     plv.lv_nr = lvo.lv_nr
AND     plv.tag   = lvo.tag
AND     plv.zeit  = lvo.zeit
AND     lv.lv_name LIKE 'Daten%'
AND     lvo.gebaeude = 'C'
  
```

## - Equi-Join mit mehr als zwei Tabellen - ab SQL92 (I) -

```
Anfrage :=  
  SELECT    <Tupel>  
  FROM      <Tabellen>          [  
  WHERE     <Prädikat>]         [  
  ORDER BY  <Sortierung>        ]  
  
Tabellen :=  
  { ({Tabellen | Tabelle [AS <Alias>] }  
    [INNER] JOIN  
    <Tabelle> [[AS] <Alias>]  
    ON <PRÄDIKAT>  
  ) }
```

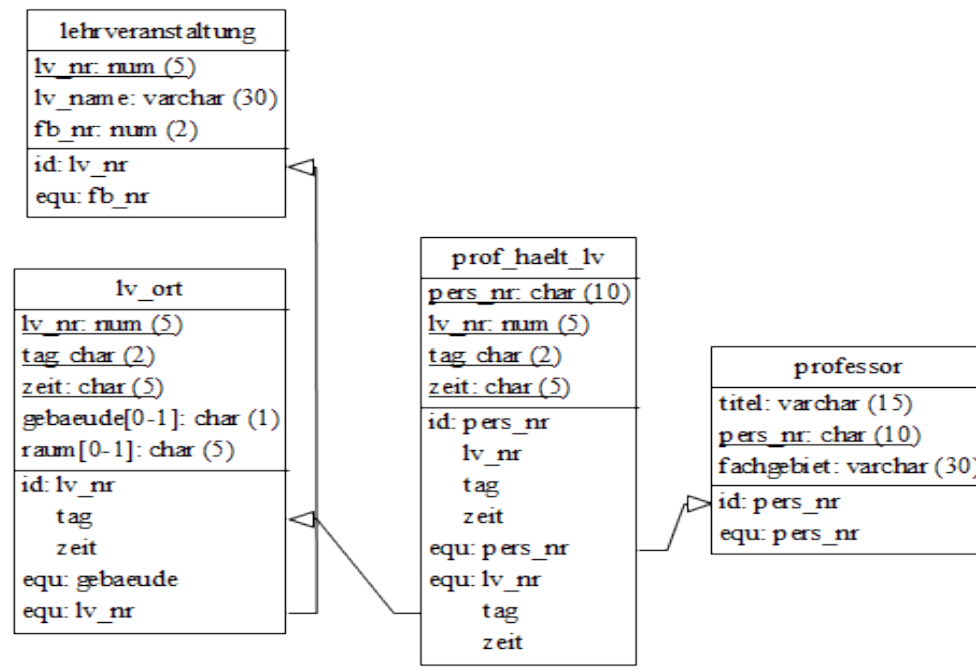
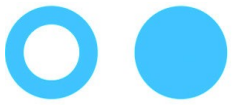
```
SELECT p.fachgebiet,  
       lv.lv_name,  
       lvo.gebaeude,  
       lvo.tag,  
       lvo.zeit  
FROM   ( ( dbs_tab_professor P  
           INNER JOIN dbs_tab_prof_haelt_lv plv  
             ON P.pers_nr = plv.pers_nr  
         )  
  
        INNER JOIN dbs_tab_lv_ort lvo  
          ON lvo.lv_nr = plv.lv_nr  
         AND lvo.tag   = plv.tag  
         AND lvo.zeit  = plv.zeit  
      )  
  INNER JOIN dbs_tab_lehrveranstaltung lv  
    ON lvo.lv_nr = lv.lv_nr  
WHERE lv.lv_name LIKE 'Daten%'  
AND   lvo.gebaeude = 'C'
```

## - Equi-Join mit mehr als zwei Tabellen - ab SQL92 (II) -



```

SELECT p.fachgebiet,
       lv.lv_name,
       lvo.gebaeude,
       lvo.tag,
       lvo.zeit
FROM   ( ( dbs_tab_professor P
           INNER JOIN dbs_tab_prof_haelt_lv plv
             ON P.pers_nr = plv.pers_nr
         )
        INNER JOIN dbs_tab_lv_ort lvo
          ON lvo.lv_nr = plv.lv_nr
          AND lvo.tag   = plv.tag
          AND lvo.zeit  = plv.zeit
        )
INNER JOIN dbs_tab_lehrveranstaltung lv
  ON lvo.lv_nr = lv.lv_nr
WHERE lv.lv_name LIKE 'Daten%'
AND   lvo.gebaeude = 'C'
  
```



Die Tabellen haben jeweils 10 Datensätze (vgl. Abbildung). Die Selektion einer Tabelle mit den Fachgebieten und den zugehörigen LV-Namen ergibt maximal ...

0 Datensätze

10 Datensätze

100 Datensätze

1.000 Datensätze

10.000 Datensätze

## - Equi-Join mit 'NULL'-Werten (Outer Join) - vor SQL92 (I) -

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

```
Tupel :=  
  [DISTINCT] {*                |  
  {<Ausdruck>} [AS <neue Spalte>]  
  [, <Tupel>]                  }
```

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

```
Tabellen :=  
  {<Tabelle> [<Alias>]}  
  [, <Tabellen>]
```

(+): Oracle-Syntax

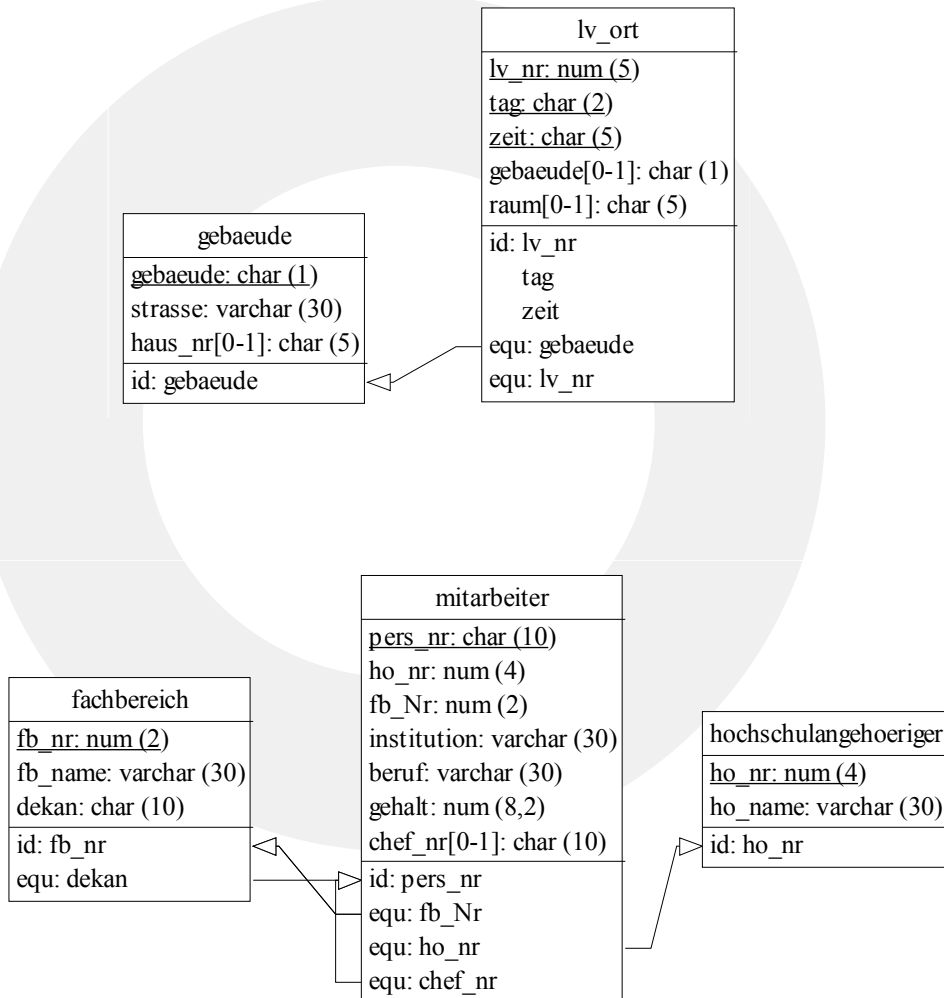
```
SELECT  *  
FROM    dbs_tab_lv_ort lvo,  
        dbs_tab_gebaeude g  
WHERE   lvo.gebaeude (+) = g.gebaeude
```

```
SELECT  ho.ho_nr, ho.ho_name,  
        fb.fb_nr, fb.fb_name,  
        m.beruf, m.gehalt  
FROM    dbs_tab_hochschulangehoeriger ho,  
        dbs_tab_fachbereich fb,  
        dbs_tab_mitarbeiter m  
WHERE   m.ho_nr (+) = ho.ho_nr  
AND     fb.fb_nr (+) = m.fb_nr
```

(+): Oracle-Syntax

das (+) steht auf der mit NULL-Werten aufzufüllenden Seite, sofern die Datensätze auf der anderen Seite dort keinen Join-Partner finden

## - Equi-Join mit 'NULL'-Werten (Outer Join) - vor SQL92 (II) -



```

SELECT  *
FROM    dbs_tab_lv_ort lvo,
        dbs_tab_gebaeude g
WHERE   lvo.gebaeude (+) = g.gebaeude;
  
```

```

SELECT  ho.ho_nr, ho.ho_name,
        fb.fb_nr, fb.fb_name,
        m.beruf, m.gehalt
FROM    dbs_tab_hochschulangehoeriger ho,
        dbs_tab_fachbereich fb,
        dbs_tab_mitarbeiter m
WHERE   m.ho_nr (+) = ho.ho_nr
AND     fb.fb_nr (+) = m.fb_nr;
  
```

(+) : Oracle-Syntax

das (+) steht auf der mit NULL-Werten aufzufüllenden Seite, sofern die Datensätze auf der anderen Seite dort keinen Join-Partner finden

## - Equi-Join mit 'NULL'-Werten (Outer Join) - ab SQL92 (I) -

```
Anfrage :=  
  SELECT    <Tupel>  
  FROM      <Tabellen>      [  
  WHERE     <Prädikat>]    [  
  ORDER BY  <Sortierung>   ]  
  
Tabellen :=  
  {({Tabellen | Tabelle [AS <Alias>] }  
    { [INNER]  
    | LEFT [OUTER]  
    | RIGHT [OUTER]  
    | FULL [OUTER] }  
    JOIN <Tabelle> [[AS] <Alias>]  
    ON <PRÄDIKAT>  
  ) }
```

Left-Outer-Join: ⚡

Right-Outer-Join: ⚡

FULL-Outer-Join: ⚡

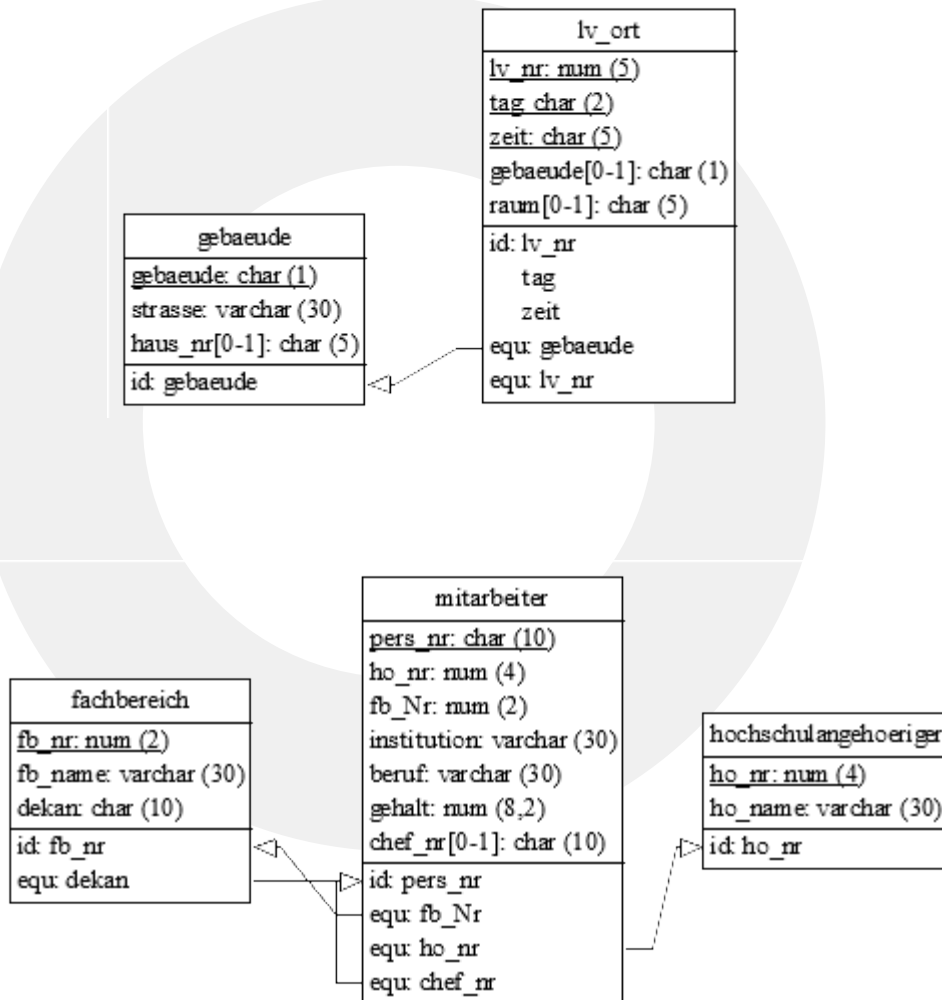
```
SELECT      *  
FROM        dbs_tab_lv_ort lvo  
RIGHT  
           OUTER JOIN dbs_tab_gebaeude g  
ON          lvo.gebaeude = g.gebaeude
```

```
SELECT      ho.ho_nr, ho.ho_name,  
            fb.fb_nr, fb.fb_name,  
            m.beruf, m.gehalt  
FROM        dbs_tab_hochschulangehoeriger ho  
LEFT  
           OUTER JOIN dbs_tab_mitarbeiter m  
ON          m.ho_nr = ho.ho_nr  
LEFT  
           OUTER JOIN dbs_tab_fachbereich fb  
ON          fb.fb_nr = m.fb_nr
```

Normierte-Syntax:

LEFT/RIGHT verweist auf die Seite, die auch ohne Join-Partner auf der anderen Seite ausgegeben werden soll, allerdings dann mit NULL-Werten auf der anderen Seite

## - Equi-Join mit 'NULL'-Werten (Outer Join) - ab SQL92 (II) -



```

SELECT      *
FROM        dbs_tab_lv_ort lvo
RIGHT
    OUTER JOIN  dbs_tab_gebaeude g
ON          lvo.gebaeude = g.gebaeude
  
```

```

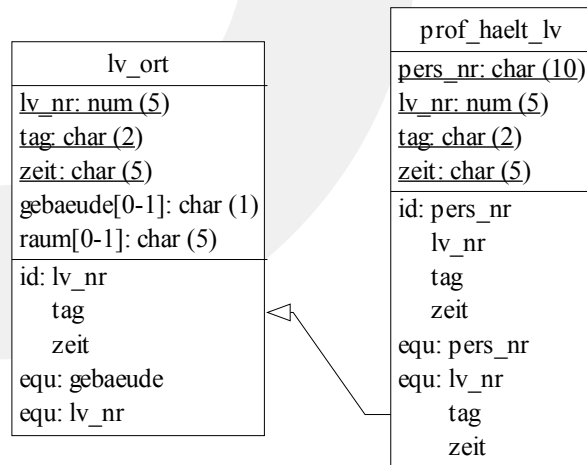
SELECT      ho.ho_nr, ho.ho_name,
            fb.fb_nr, fb.fb_name,
            m.beruf, m.gehalt
FROM        dbs_tab_hochschulangehoeriger ho
LEFT
    OUTER JOIN  dbs_tab_mitarbeiter m
ON          m.ho_nr = ho.ho_nr
LEFT
    OUTER JOIN  dbs_tab_fachbereich fb
ON          fb.fb_nr = m.fb_nr
  
```

### Normierte-Syntax:

LEFT/RIGHT verweist auf die Seite, die auch ohne Join-Partner auf der anderen Seite ausgegeben werden soll, allerdings dann mit NULL-Werten auf der anderen Seite

## - Vereinigung / Durchschnitt / Differenz -

```
Anfrage :=
  SELECT    <Tupel>
  FROM      <Tabellen>          [
  WHERE     <Prädikat>          [
  ORDER BY  <Sortierung>        [
  {UNION | INTERSECT | MINUS}
  <Anfrage>                      ]
```



```
SELECT *
FROM   dbs_tab_lv_ort
WHERE  gebaeude = 'F'
UNION
SELECT *
FROM   dbs_tab_lv_ort
WHERE  tag = 'Di'

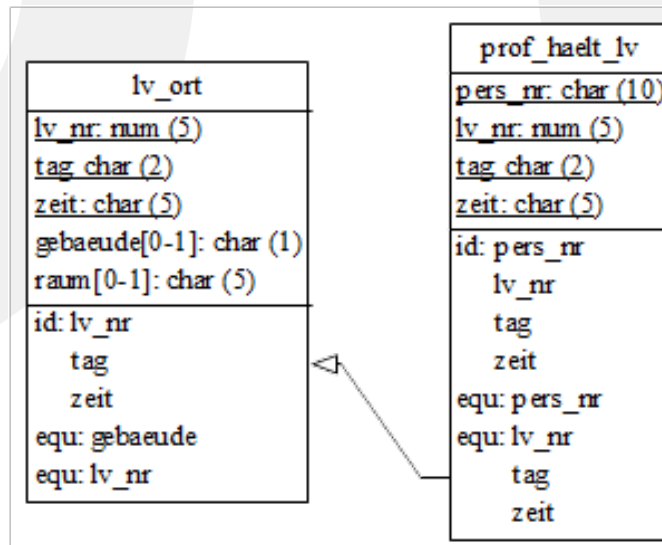
SELECT lv_nr
FROM   dbs_tab_lv_ort
WHERE  gebaeude = 'F'
INTERSECT
SELECT lv_nr
FROM   dbs_tab_prof_haelt_lv plv
WHERE  plv.tag = 'Di'

SELECT lv_nr
FROM   dbs_tab_lv_ort
WHERE  gebaeude = 'C'
MINUS
SELECT lv_nr
FROM   dbs_tab_prof_haelt_lv plv
WHERE  dbs_plv.pers_nr = '509514'
```

## - Vereinigung / Durchschnitt / Differenz -

```
SELECT raum
FROM lv_ort
WHERE gebaeude = 'F'
INTERSECT
SELECT zeit
FROM prof_haelt_lv plv
WHERE plv.tag = 'Di'

SELECT tag
FROM lv_ort
WHERE gebaeude = 'F'
INTERSECT
SELECT lv_nr
FROM prof_haelt_lv plv
WHERE plv.tag = 'Di'
```



```
SELECT lv_nr, tag
FROM lv_ort
WHERE gebaeude = 'F'
INTERSECT
SELECT lv_nr, tag
FROM prof_haelt_lv plv
WHERE plv.tag = 'Di'

SELECT tag
FROM lv_ort
WHERE gebaeude = 'F'
INTERSECT
SELECT lv_nr, tag
FROM prof_haelt_lv plv
WHERE plv.tag = 'Di'
```