Hochschule Aalen

Datenbanksysteme
5 SQL

Prof. Dr. Gregor Grambow

Hochschule Aalen
Fakultät Elektronik und Informatik

Überblick

Inhalt

- Kurze Historie
- Datenmanipulation
- Einfügen, Löschen, Ändern
- Abfragen, Tabellenausdrücke
- Sortieren
- Verbund - Join
- Gruppierungen
- Mengenoperationen
- Besondere Bedingungen
- Pattern Matching
- Sichten

Ziele

- Verschiedene Eigenschaften und Möglichkeiten von SQL kennen und anwenden können

Foliensatz basiert teilweise auf [Kemper, Eickler: Datenbanksysteme – Eine Einführung. Oldenbourg, 2015]

Kurze EBNF Wiederholung

- Erweiterte Backus-Naur-Form
- zur Definition von Grammatiken
- Terminalsymbole werden in Anführungszeichen dargestellt, Nichtterminalsymbole ohne
  - Terminalsymbole: "a", "b", "c", "1", "+", ...
  - Nichtterminalsymbole: A, BEGIN, vorzeichen, ...
- Die linke und die rechte Seite einer Produktionsregel werden statt durch einen Pfeil durch das Zeichen "::=" getrennt
  - `A ::= "b"`

Kurze EBNF Wiederholung

- Alternative
  - Kann ein Nichtterminalsymbol durch mehrere Symbolfolgen ersetzt werden, dann werden diese auf der rechten Seite durch senkrechte Striche getrennt, die dem ODER entsprechen.
  - `Alternative ::= "b" | "c" | "d".`
- Optionale Wiederholung
  - Kann ein Teil der rechten Seite einer Produktionsregel wiederholt werden, so wird er in geschweifte Klammern eingeschlossen. (0..n)
  - `OptionaleWiederholung ::= {"a"}.`
  - Alternativ: `OptionaleWiederholung ::= {"a"}*.`
- Option
  - Darf ein Teil einer Produktionsregel optional (also nicht zwingend) vorkommen, so wird dieser wie folgt in eckigen Klammern eingeschlossen. (0..1)
  - `Option ::= [" - " | " + " ] "123456"`
  - Äquivalent zu
  - `Option ::= "123456" | "+123456" | "-123456".`

Kurze EBNF Wiederholung

- Gruppierung
  - Einfache Gruppierung, auch für Abbildung verschiedener Optionen.
  - `Gruppierung ::= "a", ("a" | "b").`
  - Äquivalent zu
  - `Gruppierung ::= "aa" | "ab"`

- Beispiel:

```
CONSTANT ::= unsignedNumber | sign unsignedNumber.
sign ::= "+" | "-".
unsignedNumber ::= digit  {digit}.
digit ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" |
"8" | "9".
```

Referenzielle Bedingungen

- **Eine Attributmenge F ⊆ A wird als *Fremdschlüssel* in R = (A, ...) bezeichnet, falls**
  - **die Inklusionsabhängigkeit ($R|_F \subseteq R'|_{F'}$) gilt** (wobei R' ≠ R) **und**
  - **F' Schlüssel im *fremden* Schema R' ist**
-
- **SQL:** ***foreign-key-def ::=***
- **FOREIGN KEY** ***"(" column-name { "," column-name }* ")"***
- **REFERENCES** ***table-name ["(" column-name {"," column-name }* ")"]***
- ***[* ON *(* DELETE | UPDATE *)***
  (* NO ACTION | CASCADE | RESTRICT | SET DEFAULT | SET NULL *)].

SQL zur Schema-Definition

- *relationenschema ::=*
- **CREATE TABLE** *table-name* “**(**”*(attribute-def | integrity-def)*
  { “**,**” *(attribute-def | integrity-def) }** “**)**”**.**
- *attribute-def ::=*

  column-name (data-type | domain-name) [default-def] {column-constraint} .*
- *data-type ::=*

  *(***CHAR***[***ACTER***]* | **VARCHAR** | **INT***[***EGER***]* | **FLOAT** | **DEC***[***IMAL***]* | **DATE** | **BOOLEAN** | ***...)*.**
- *domain ::=* **CREATE DOMAIN** *domain-name* **AS** *data-type* **.**

Attribut-Bedingungen

- *column-constraint ::= [* **CONSTRAINT** *constraint-name]*
- *(* **UNIQUE** | **PRIMARY KEY** | **NOT NULL** | **CHECK** *“(” condition “)” | foreign-key-def)* **.**
-
- → *column-constraint* **kann immer durch** *integrity-def* **ersetzt werden.**

Integrität

- *integrity-def ::=*

  [**CONSTRAINT** *constraint-name] (key-def | foreign-key-def | ...) .*

- *key-def ::=*

  (**UNIQUE** | **PRIMARY KEY** *) "(" column-name {"," column-name }* ")" .*

- *foreign-key-def ::=*

  FOREIGN KEY *"(" column-name {"," column-name }* ")"*

  REFERENCES *table-name ["(" column-name {"," column-name }* ")"]*

  [**ON** *(* **DELETE** | **UPDATE** *)*

  (**NO ACTION** | **CASCADE** | **RESTRICT** | **SET DEFAULT** | **SET NULL** *)].*

Beispiel

```
CREATE TABLE  Mitarbeiter
(
Name       PNAME NOT NULL,
PersNr     INTEGER PRIMARY KEY,
Account    CHAR(10) UNIQUE,
Beruf      CHAR(20),
Wohnort    CHAR(30),
StKlasse   INT check (StKlasse BETWEEN 1 AND 6)
)

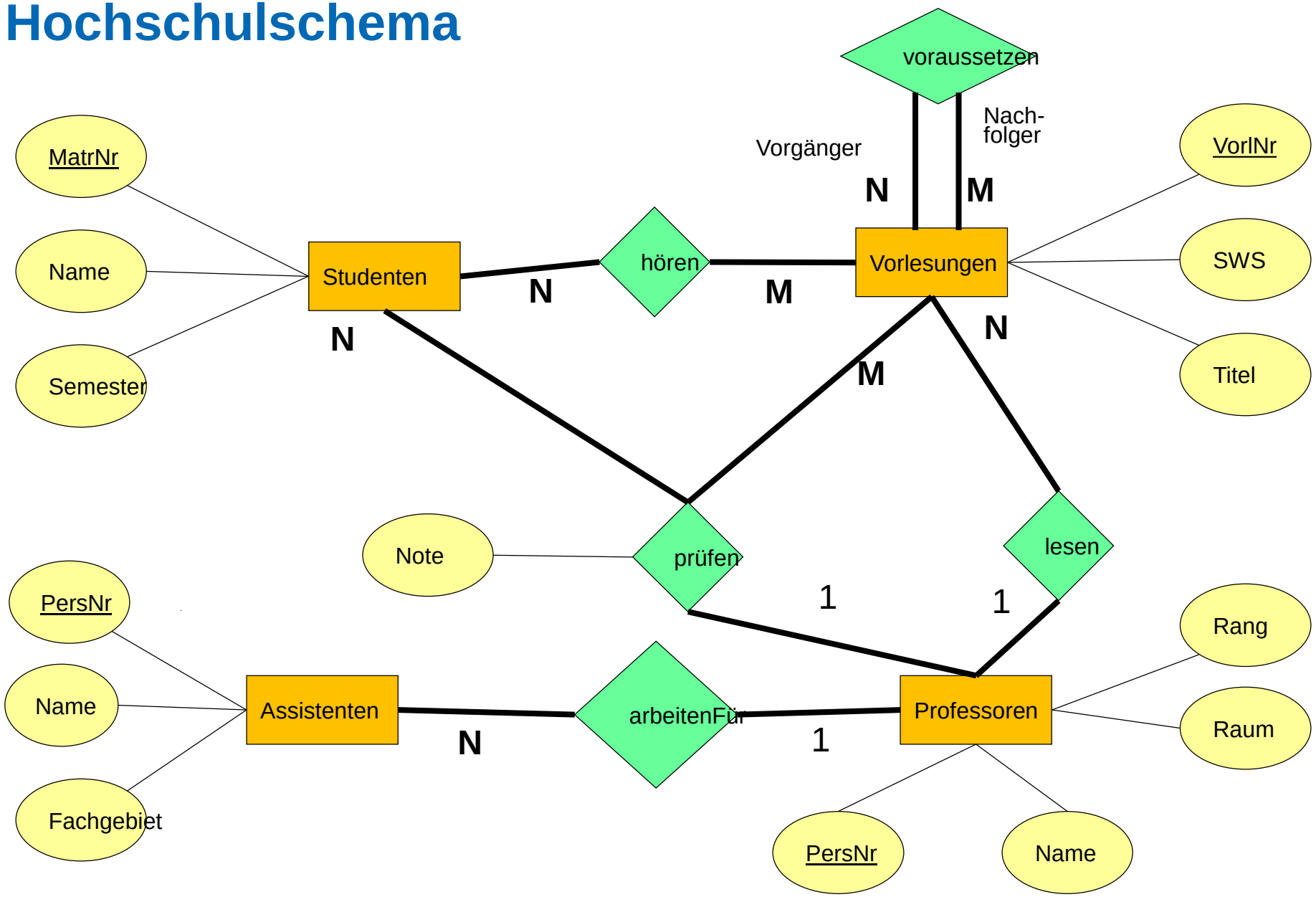
CREATE DOMAIN PNAME AS CHAR(30)
```

(Einfache) Datendefinition in SQL

- Datentypen
  - character (n), char (n)
  - character varying (n), varchar (n)
  - numeric (p,s), integer, decimal
  - blob oder raw für sehr große binäre Daten
  - clob für sehr große String-Attribute
  - date für Datumsangaben
  - xml für XML-Dokumente
- Hochschul-Bsp.:

```
create table Professoren
          (PersNr      integer PRIMARY KEY,
           Name        varchar (30) not null,
           Rang        character (2) );
```

Hochschulschema



Die relationale Hochschul-DB

Professoren			
PersNr	Name	Rang	Raum
2125	Sokrates	C4	226
2126	Russel	C4	232
2127	Kopernikus	C3	310
2133	Popper	C3	52
2134	Augustinus	C3	309
2136	Curie	C4	36
2137	Kant	C4	7

Studenten		
MatrNr	Name	Semester
24002	Xenokrates	18
25403	Jonas	12
26120	Fichte	10
26830	Aristoxenos	8
27550	Schopenhauer	6
28106	Carnap	3
29120	Theophrastos	2
29555	Feuerbach	2

Vorlesungen			
VorlNr	Titel	SWS	gelesen von
5001	Grundzüge	4	2137
5041	Ethik	4	2125
5043	Erkenntnistheorie	3	2126
5049	Mäeutik	2	2125
4052	Logik	4	2125
5052	Wissenschaftstheorie	3	2126
5216	Bioethik	2	2126
5259	Der Wiener Kreis	2	2133
5022	Glaube und Wissen	2	2134
4630	Die 3 Kritiken	4	2137

voraussetzen	
Vorgänger	Nachfolger
5001	5041
5001	5043
5001	5049
5041	5216
5043	5052
5041	5052
5052	5259

hören	
MatrNr	VorlNr
26120	5001
27550	5001
27550	4052
28106	5041
28106	5052
28106	5216
28106	5259
29120	5001
29120	5041
29120	5049
29555	5022
25403	5022

Assistenten			
PerslNr	Name	Fachgebiet	Boss
3002	Platon	Ideenlehre	2125
3003	Aristoteles	Syllogistik	2125
3004	Wittgenstein	Sprachtheorie	2126
3005	Rhetikus	Planetenbewegung	2127
3006	Newton	Keplersche Gesetze	2127
3007	Spinoza	Gott und Natur	2126

prüfen			
MatrNr	VorlNr	PersNr	Note
28106	5001	2126	1
25403	5041	2125	2
27550	4630	2137	2

Beispiele

```
CREATE TABLE Vorlesungen(
VorlNr          INTEGER PRIMARY KEY,
Titel           VARCHAR(30),
SWS             INTEGER,
gelesenVon      INTEGER REFERENCES Professoren(PersNR));

CREATE TABLE Assistenten(
PersNr          INTEGER PRIMARY KEY,
Name            VARCHAR(30) NOT NULL,
Fachgebiet      VARCHAR(30),
Boss            INTEGER,
FOREIGN KEY     (Boss) REFERENCES Professoren(PersNR));
```

SQL Infos

- https://www.postgresql.org/docs/ – spezifisch für pg!
- Literatur zu SQL (Kap. 0 Einführung)
  - http://de.wikibooks.org/wiki/Einf%C3%BChrung_in_SQL
- viele Tutorials -bspw.
  - https://www.postgresql.org/docs/current/static/tutorial.html
  - http://www.w3schools.com/sql/
- aber Achtung: ggf. unterschiedliche Modellierung

- Hinweise zur Installation von PostgreSQL:
- http://www.postgresql.org
- https://wiki.postgresql.org/wiki/Main_Page

Übungsserver

- 141.18.48.4 - erreichbar innerhalb der Hochschule (auch VPN)
- Standardport: 5432
- Verwendung:
- GUI: pgAdmin
- Kommandozeile: psql
- Übungsdatenbank: dbsys
- Übungsaccount: dbsys
- Eigene Tabellen mit Namenspräfix:

1. erster Buchstabe des Vornamens
2. die ersten beiden Buchstaben des Nachnamens
3. Unterstrich

- Umlaute bitte auflösen (ä → ae usw.)
  - Konflikte bei den Präfixen beachten

Kurze Historie

- 1974 Chamberlin und Boyce veröffentlichen die Sprache SEQUEL.
- 1977 aus SEQUEL geht SQL hervor.
- 1986 ANSI Norm für SQL, später ISO.
- 1989 Revision der Norm.
- **1992 SQL-92 (SQL-2) → 2. Revision der Norm.**
- 1995 Call-Level Interface kommt hinzu
- 1999 SQL:1999 (SQL-3) u.a. Persistent Stored Modules, Objektorientierte Erweiterungen, neue Basistypen, ...
- 2003 SQL:2003 XML Unterstützung, Multiset-Typen, Sequenzen, ...
- 2008 SQL:2008 weitere XML Unterstützung - XQuery (W3C)
- 2011 SQL:2011 minor language extensions …
- 2016 SQL:2016 Row Pattern Matching, JSON …

→ ISO/IEC 9075:2011

ISO/IEC 9075

- https://en.wikipedia.org/wiki/SQL#Interoperability_and_standardization :
- SQL implementations are incompatible between vendors and do not necessarily completely follow standards. In particular date and time syntax, string concatenation, NULLs, and comparison case sensitivity vary from vendor to vendor. A particular exception is PostgreSQL, which strives for standards compliance.

- ISO/IEC 9075-1 Framework (SQL/Framework)
- ISO/IEC 9075-2 Foundation (SQL/Foundation)
- ISO/IEC 9075-3 Call Level Interface (SQL/CLI)
- ISO/IEC 9075-4 Persistent Stored Modules (SQL/PSM)
- ISO/IEC 9075-9 Management of External Data (SQL/MED)
- ISO/IEC 9075-10 Object Language Bindings (SQL/OLB)
- ISO/IEC 9075-11 Information and Definition Schemas (SQL/Schemata)
- ISO/IEC 9075-13 Routines and Types using the Java Language (SQL/JRT)
- ISO/IEC 9075-14 XML-related specifications (SQL/XML)

- PostgreSQL: https://www.postgresql.org/docs/current/static/features.html

Grundlegendes

- Bezeichner (Namen) (Definition durch regulären Ausdruck - max. 128 Zeichen)
  - gewöhnlicher Bezeichner / regular identifier
  - [abcdefghijklmnopqrstuvwxyz]([abcdefghijklmnopqrstuvwxyz0123456789_])*
  - Groß-/Kleinschreibung wird nicht unterschieden,
  - Großbuchstaben dürfen wie Kleinbuchstaben verwendet werden
- begrenzte Bezeichner / delimited identifier
  - beliebige Zeichenfolge durch “"” begrenzt - bspw. "neue Bücher"
- SQL-Statements werden üblicherweise in 3 Kategorien unterteilt:
- Datendefinition → Schema-Definition

  (DDL = Data Definition Language)
- Datenmanipulation

  (DML = Data Manipulation Language)
- Programmierung (Module Language Statements)

Tabelle verändern

- **DROP TABLE** *table-name* .

```
drop table Professoren;
```

- **ALTER TABLE** *table-name (add-def | drop-def | ...)* .

- *add-def ::=*

```
alter  table Professoren
add    Geburtstag date;
```

  ADD *[* **COLUMN** *] attribute-def*

  | **ADD CONSTRAINT** *constraint-name (key-def | foreign-key-def | check-def)* .

- *drop-def ::=*

  DROP *[* **COLUMN** *] column-name*

  | **DROP CONSTRAINT** *constraint-name* .

Datenmanipulation

- **Änderungen**
  - **INSERT** *(Einfügen neuer Zeilen in eine Tabelle)*
  - **DELETE** *(Löschen von Zeilen)*
  - **UPDATE** *(Inhalt bestehender Zeilen ändern)*

- **Abfrage → Tabellenausdrücke**
  - **SELECT**

INSERT

- *insert-statement ::=*

  INSERT INTO *tabellen-name [“(” spalten-name-liste “)”] tabellen-ausdruck .*

- *spalten-name-liste ::= spalten-name { “,” spalten-name }* .*

  ***Liste:** <bezeichnung>-liste ::= <bezeichnung> { “,” <bezeichnung>}* .*

- *tabellen-ausdruck ::=* -- stark vereinfacht

  VALUES *“(” skalar-ausdruck-liste “)”*
  *{ “,” “(” skalar-ausdruck-liste “)” }**

  | *select-ausdruck*

  | **TABLE** *tabellen-name .*

skalar-ausdruck

- *skalar-ausdruck ::= literal | spalten-name | . . .*

- *literal ::=*

[sign] unsigned numeric literal	**3 -23.4**
| character string literal	**'Strings in einfache Quotes'**
| datetime literal	**DATE '2005-05-30'** **TIMESTAMP '2005-05-30 14:00:00'**
| interval literal	**INTERVAL '100' HOUR**
| boolean literal	**TRUE FALSE *UNKNOWN***
| . . .	

INSERT Beispiele

- Einfügen von Tupeln

- ```
  insert into Studenten (MatrNr, Name)
      values (28121,  `Archimedes‘);
  ```

- ```
  insert into hören
      select MatrNr, VorlNr
      from Studenten, Vorlesungen
      where Titel= `Logik‘ ;
  ```

DELETE

- ***Delete-Statement ::=***

  DELETE FROM *tabellen-name [* **WHERE** *bedingung]* **.**

- Beispiel:
- ```
  delete from Studenten
      where Semester > 13;
  ```
  ```

UPDATE

- *Update-Statement ::=*

  UPDATE *tabellen-name* **SET** *spalten-name* **"="** *ausdruck*
  { **","** *spalten-name* **"="** *ausdruck}**

  [**WHERE** *bedingung]* **.**

- Beispiel:
- ```
  update Studenten
  ```
- ```
      set Semester= Semester + 1;
  ```

SELECT

- Grundstruktur:

  SELECT spalten-name(n) (genauer: spalten-bezeichner)

  FROM relation(en)

  WHERE bedingung

- Verarbeitungsschema:

1. Bilde die Konkatenation der bei FROM angegebenen Relationen
2. Selektiere die Tupel, die die Bedingung erfüllen
3. Projiziere das Ergebnis auf die bei SELECT angegebenen Attribute

Vereinfachte Syntax von SELECT

- ***select-ausdruck::=***

  SELECT *[* **ALL** | **DISTINCT** *]* *(* *"*** **"** | *attribut-name-liste)*

  FROM *tabellen-name-liste*

  [**WHERE** *bedingung]* **.**

Einfache Anfrage

- **select** PersNr, Name
  from Professoren
  where Rang= ´C4´;

PersNr	Name
2125	Sokrates
2126	Russel
2136	Curie
2137	Kant

Duplikateliminierung

- ```
  select distinct Rang
      from Professoren
  ```

Rang
C3
C4
  ```

Syntax von SELECT

- *select-ausdruck ::=*

  SELECT *[* **ALL** | **DISTINCT** *]* *(* *"*** **"* | *select-element-liste)*

  FROM *tabellenreferenz-liste*

  [**WHERE** *bedingung]*

  [**GROUP BY** *spalten-name-liste]* *[* **HAVING** *bedingung]*

  [order-by-klausel].

- *select-element ::= skalar-ausdruck* *[* **AS** *name] .*

- *order-by-klausel ::=*

  ORDER BY *select-element-name [* **ASC** | **DESC** *]* *{ ",""select-element-name [***ASC** | **DESC** *]* *}* .*

- *tabellenreferenz ::=*

  tabellen-ausdruck-mit-join

  | *tabellen-name [[* **AS** *] bereichsvariable ["(" spalten-name-liste ")"]]*

  | *"(" tabellen-ausdruck ")" / [[* **AS** *] bereichsvariable ["(" spalten-name-liste ")"]] .*

Sortierung

```
select      PersNr, Name, Rang
    from    Professoren
    order by Rang desc, Name asc;
```

PersNr	Name	Rang
2136	Curie	C4
2137	Kant	C4
2126	Russel	C4
2125	Sokrates	C4
2134	Augustinus	C3
2127	Kopernikus	C3
2133	Popper	C3

Implizite und explizite Joins

- Impliziter Join via Bedingung

  Bsp.: SELECT * FROM A, B WHERE A.X = B.X

- Expliziter Join via Tabellenausdruck mit Join

- ***tabellen-ausdruck-mit-join ::=***

  tabellenreferenz* CROSS JOIN *tabellenreferenz

  | ***tabellenreferenz (* NATURAL | UNION *)* JOIN *tabellenreferenz***

  | ***tabellenreferenz [* INNER | LEFT *[*OUTER*]* | RIGHT *[*OUTER*]* | FULL *[*OUTER*]]* JOIN *tabellenreferenz***

  ***(* ON *bedingungs-ausdruck* | USING “(” *spalten-name-liste* “)” *)* .**

Die relationale Hochschul-DB

Professoren			
PersNr	Name	Rang	Raum
2125	Sokrates	C4	226
2126	Russel	C4	232
2127	Kopernikus	C3	310
2133	Popper	C3	52
2134	Augustinus	C3	309
2136	Curie	C4	36
2137	Kant	C4	7

Studenten		
MatrNr	Name	Semester
24002	Xenokrates	18
25403	Jonas	12
26120	Fichte	10
26830	Aristoxenos	8
27550	Schopenhauer	6
28106	Carnap	3
29120	Theophrastos	2
29555	Feuerbach	2

Vorlesungen			
VorlNr	Titel	SWS	gelesen von
5001	Grundzüge	4	2137
5041	Ethik	4	2125
5043	Erkenntnistheorie	3	2126
5049	Mäeutik	2	2125
4052	Logik	4	2125
5052	Wissenschaftstheorie	3	2126
5216	Bioethik	2	2126
5259	Der Wiener Kreis	2	2133
5022	Glaube und Wissen	2	2134
4630	Die 3 Kritiken	4	2137

voraussetzen	
Vorgänger	Nachfolger
5001	5041
5001	5043
5001	5049
5041	5216
5043	5052
5041	5052
5052	5259

hören	
MatrNr	VorlNr
26120	5001
27550	5001
27550	4052
28106	5041
28106	5052
28106	5216
28106	5259
29120	5001
29120	5041
29120	5049
29555	5022
25403	5022

Assistenten			
PerslNr	Name	Fachgebiet	Boss
3002	Platon	Ideenlehre	2125
3003	Aristoteles	Syllogistik	2125
3004	Wittgenstein	Sprachtheorie	2126
3005	Rhetikus	Planetenbewegung	2127
3006	Newton	Keplersche Gesetze	2127
3007	Spinoza	Gott und Natur	2126

prüfen			
MatrNr	VorlNr	PersNr	Note
28106	5001	2126	1
25403	5041	2125	2
27550	4630	2137	2

Join Beispiel

- Welcher Professor liest "Mäeutik"?

- ```
  select  Name,  Titel
      from Professoren , Vorlesungen
      where  PersNr  =  gelesenVon and  Titel = `Mäeutik‘ ;
  ```

$$\prod_{\text{Name, Titel}} (\sigma_{\text{PersNr} = \text{gelesenVon} \wedge \text{Titel} = \text{'Mäeutik'}} (\text{Professoren} \times \text{Vorlesungen}))$$
  ```

Join Beispiel

Professoren			
PersNr	Name	Rang	Raum
2125	Sokrates	C4	226
2126	Russel	C4	232
⋮	⋮	⋮	⋮
2137	Kant	C4	7

Vorlesungen			
VorlNr	Titel	SWS	gelesen Von
5001	Grundzüge	4	2137
5041	Ethik	4	2125
⋮	⋮	⋮	⋮
5049	Mäeutik	2	2125
⋮	⋮	⋮	⋮
4630	Die 3 Kritiken	4	2137

Verknüpfung X

Join Beispiel

PersN	Name	Rang	Raum	VorlNr	Titel	SWS	gelesen Von
2125	Sokrates	C4	226	5001	Grundzüge	4	2137
1225	Sokrates	C4	226	5041	Ethik	4	2125
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
2125	Sokrates	C4	226	5049	Mäeutik	2	2125
⋮	⋮		⋮	⋮	⋮	⋮	⋮
2126	Russel	C4	232	5001	Grundzüge	4	2137
2126	Russel	C4	232	5041	Ethik	4	2125
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
2137	Kant	C4	7	4630	Die 3 Kritiken	4	2137

↓ Auswahl

PersNr	Name	Rang	Raum	VorlNr	Titel	SWS	gelesen Von
2125	Sokrates	C4	226	5049	Mäeutik	2	2125

↓ Projektion

Name	Titel
Sokrates	Mäeutik

Weiteres Join Beispiel mit Aliasen

- Welche Studenten hören welche Vorlesungen?
-
```
select Name, Titel
    from Studenten, hören, Vorlesungen
    where Studenten.MatrNr = hören.MatrNr and
          hören.VorlNr = Vorlesungen.VorlNr;
```

- Alternativ:
-
```
select s.Name, v.Titel
    from Studenten s, hören h, Vorlesungen v
    where s.MatrNr = h.MatrNr and
          h.VorlNr = v.VorlNr
```

Weiteres Beispiel

- Welche Studenten kennen sich aus Vorlesungen?

- ```
  select s1.Name, s2.Name
      from Studenten s1, hoeren h1, hoeren h2, Studenten s2
      where h1.VorlNr = h2.VorlNr and h1.MatrNr = s1.MatrNr and
      h2.MatrNr = s2.MatrNr
  ```
  ```

Joins in SQL-92

- **cross join**: Kreuzprodukt
- **natural join**: natürlicher Join (nicht alle DBMS)
- **join** oder **inner join**: Theta-Join
- **left**, **right** oder **full outer join**: äußerer Join
- **union join**: Vereinigungs-Join (wird hier nicht vorgestellt)

Joins in SQL-92

- **cross join**
- Der Cross Join (auch Kartesisches Produkt oder Kreuzprodukt) verbindet jede Zeile der ersten Tabelle mit jeder Zeile der zweiten Tabelle.

- **`Select * from TabelleA cross join TabelleB`**

TabelleA

Index	Name
1	Jan
2	Tom
3	Anne

TabelleB

Index	Geb.Datum
1	16.05.1976
2	29.07.1985
5	08.11.2001

Ergebnis

TabelleA.Index	Name	TabelleB.Index	Geb.Datum
1	Jan	1	16.05.1976
1	Jan	2	29.07.1985
1	Jan	5	08.11.2001
2	Tom	1	16.05.1976
2	Tom	2	29.07.1985
2	Tom	5	08.11.2001
3	Anne	1	16.05.1976
3	Anne	2	29.07.1985
3	Anne	5	08.11.2001

[https://glossar.hs-augsburg.de/Verbundoperatoren_(Join)]

Joins in SQL-92

- **natural join**
- Der Natural Join verknüpft die beiden Tabellen über die Gleichheit der Felder, in Spalten mit gleichem Namen. Spalten mit gleichem Namen werden im Ergebnis nur einmal angezeigt. Haben die Tabellen keine Spalten mit gleichem Namen, wird der Natural Join automatisch zum Cross Join.
- `select * from TabelleA natural join TabelleB`

TabelleA

Index	Name
1	Jan
2	Tom
3	Anne

TabelleB

Index	Geb.Datum
1	16.05.1976
2	29.07.1985
5	08.11.2001

Ergebnis

Index	Name	Geb.Datum
1	Jan	16.05.1976
2	Tom	29.07.1985

[https://glossar.hs-augsburg.de/Verbundoperatoren_(Join)]

Joins in SQL-92

- **join oder inner join**
- Der Inner Join verbindet Datensätze aus zwei Tabellen, welche in beiden Tabellen denselben Werte enthalten. Die Spalten die in beiden Tabellen verglichen werden sollen, muss explizit angegeben werden.

- `select * from TabelleA inner join TabelleB on TabelleA.Index = TabelleB.Index`

TabelleA

Index	Name
1	Jan
2	Tom
3	Anne

TabelleB

Index	Geb.Datum
1	16.05.1976
2	29.07.1985
5	08.11.2001

Ergebnis

TabelleA.Index	Name	TabelleB.Index	Geb.Datum
1	Jan	1	16.05.1976
2	Tom	2	29.07.1985

[https://glossar.hs-augsburg.de/Verbundoperatoren_(Join)]

Joins in SQL-92

- **left**, **right** oder **full outer join**
- Der **left join** (auch **left outer join** genannt) erstellt eine so genannte linke Inklusionsverknüpfung. Diese schließt alle Datensätze aus der ersten (linken) Tabelle ein, auch wenn keine entsprechenden Werte für die Datensätze in der zweiten (rechten) Tabelle existieren. Die zu vergleichenden Spalten müssen explizit Angegeben werden.

```
select * from TabelleA left join TabelleB on
     TabelleA.Index = TabelleB.Index
```

TabelleA		TabelleB		Ergebnis			
Index	Name	Index	Geb.Datum	TabelleA.Index	Name	TabelleB.Index	Geb.Datum
1	Jan	1	16.05.1976	1	Jan	1	16.05.1976
2	Tom	2	29.07.1985	2	Tom	2	29.07.1985
3	Anne	5	08.11.2001	3	Anne	NULL	NULL

[https://glossar.hs-augsburg.de/Verbundoperatoren_(Join)]

Die relationale Hochschul-DB

Professoren			
PersNr	Name	Rang	Raum
2125	Sokrates	C4	226
2126	Russel	C4	232
2127	Kopernikus	C3	310
2133	Popper	C3	52
2134	Augustinus	C3	309
2136	Curie	C4	36
2137	Kant	C4	7

Studenten		
MatrNr	Name	Semester
24002	Xenokrates	18
25403	Jonas	12
26120	Fichte	10
26830	Aristoxenos	8
27550	Schopenhauer	6
28106	Carnap	3
29120	Theophrastos	2
29555	Feuerbach	2

Vorlesungen			
VorlNr	Titel	SWS	gelesen von
5001	Grundzüge	4	2137
5041	Ethik	4	2125
5043	Erkenntnistheorie	3	2126
5049	Mäeutik	2	2125
4052	Logik	4	2125
5052	Wissenschaftstheorie	3	2126
5216	Bioethik	2	2126
5259	Der Wiener Kreis	2	2133
5022	Glaube und Wissen	2	2134
4630	Die 3 Kritiken	4	2137

voraussetzen	
Vorgänger	Nachfolger
5001	5041
5001	5043
5001	5049
5041	5216
5043	5052
5041	5052
5052	5259

hören	
MatrNr	VorlNr
26120	5001
27550	5001
27550	4052
28106	5041
28106	5052
28106	5216
28106	5259
29120	5001
29120	5041
29120	5049
29555	5022
25403	5022

Assistenten			
PerslNr	Name	Fachgebiet	Boss
3002	Platon	Ideenlehre	2125
3003	Aristoteles	Syllogistik	2125
3004	Wittgenstein	Sprachtheorie	2126
3005	Rhetikus	Planetenbewegung	2127
3006	Newton	Keplersche Gesetze	2127
3007	Spinoza	Gott und Natur	2126

prüfen			
MatrNr	VorlNr	PersNr	Note
28106	5001	2126	1
25403	5041	2125	2
27550	4630	2137	2

Beispiel Äußere Joins

```
select p.PersNr, p.Name, f.PersNr, f.Note, f.MatrNr,
  s.MatrNr, s.Name
  from Professoren p left outer join
              (prüfen f left outer join Studenten s on f.MatrNr=  s.MatrNr)
     on p.PersNr=f.PersNr;
```

PersNr	p.Name	f.PersNr	f.Note	f.MatrNr	s.MatrNr	s.Name
2126	Russel	2126	1	28106	28106	Carnap
2125	Sokrates	2125	2	25403	25403	Jonas
2137	Kant	2137	2	27550	27550	Schopen-hauer
2136	Curie	-	-	-	-	-
⋮	⋮	⋮	⋮	⋮	⋮	⋮

Beispiel Äußere Joins

```
select p.PersNr, p.Name, f.PersNr, f.Note, f.MatrNr,
       s.MatrNr, s.Name
from Professoren p right outer join
                 (prüfen f right outer join Studenten s on f.MatrNr= s.MatrNr)
  on p.PersNr=f.PersNr;
```

PersNr	p.Name	f.PersNr	f.Note	f.MatrNr	s.MatrNr	s.Name
2126	Russel	2126	1	28106	28106	Carnap
2125	Sokrates	2125	2	25403	25403	Jonas
2137	Kant	2137	2	27550	27550	Schopen-hauer
-	-	-	-	-	26120	Fichte
⋮	⋮	⋮	⋮	⋮	⋮	⋮

Beispiel Äußere Joins

- ```
  select p.PersNr, p.Name, f.PersNr, f.Note, f.MatrNr,
         s.MatrNr, s.Name
  from Professoren p full outer join
              (prüfen f full outer join Studenten on f.MatrNr= s.MatrNr)
    on p.PersNr=f.PersNr;
  ```

p.PersNr	p.Name	f.PersNr	f.Note	f.MatrNr	s.MatrNr	s.Name
2126	Russel	2126	1	28106	28106	Carnap
2125	Sokrates	2125	2	25403	25403	Jonas
2137	Kant	2137	2	27550	27550	Schopen-hauer
-	-	-	-	-	26120	Fichte
2136	Curie	-	-	-	-	-
⋮	⋮	⋮	⋮	⋮	⋮	⋮
  ```

SQL so far ...

- CREATE TABLE
- PRIMARY KEY ( ), FOREIGN KEY ( ) REFERENCES ... ON DELETE ...
- ALTER TABLE *ADD COLUMN*
- INSERT INTO ... ( ) VALUES ( )
- 'Strings' und 'Datum' in einfache '
- DELETE FROM ... WHERE !!!!
- UPDATE ... SET ... WHERE
- SELECT *DISTINCT* ... FROM ... WHERE … ORDER BY
- Tabellenreferenzen: ... AS
- NATURAL JOIN
- INNER / OUTER JOIN

Gruppierung

- *[* **GROUP BY** *spalten-name-liste]*

- Durch die GROUP BY-Klausel werden Tupel gruppiert, die für jedes der angegebenen Attribute jeweils den gleichen Wert haben.
- Eine Abfrage kann sich nur entweder
  - auf Attribute beziehen, die innerhalb jeder Gruppe gleich sein müssen

    oder
  - auf Aggregationen von Attributen innerhalb der Gruppen

Aggregatfunktionen

- ***aggregatfunktion-referenz ::=***

  COUNT *“***(*)***”* → Anzahl aller Zeilen einer Tabelle / Gruppierung

  | *(* **AVG** | **MAX** | **MIN** | **SUM** | **COUNT** *)* → Mittelwert, Maximum, Minimum, Summe, Anzahl

  *“***(***”* *[* **ALL** | **DISTINCT** *] skalar-ausdruck “***)***”* **.**

- Berücksichtigung von Duplikaten kann durch Voranstellen von **DISTINCT** verhindert werden (außer bei **COUNT** (*)).
- Null-Werte werden grundsätzlich nicht berücksichtigt.

Beispiel (AVG): → Gib zu jedem Studenten seine Durchschnittsnote aus.

Beispiele

- ```
  select avg (Semester)
      from Studenten;
  ```

- ```
  select gelesenVon, sum (SWS)
      from Vorlesungen
      group by gelesenVon;
  ```

- ```
  select gelesenVon, Name, sum (SWS)
      from Vorlesungen, Professoren
      where gelesenVon = PersNr and Rang = ´C4´
      group by gelesenVon, Name
                  having avg (SWS) >= 3;
  ```
  ```

Beispiel zur Verarbeitung mit group by

Vorlesung x Professoren							
Vorl Nr	Titel	SWS	gelesen Von	PersNr	Name	Rang	Raum
5001	Grundzüge	4	2137	2125	Sokrates	C4	226
5041	Ethik	4	2125	2125	Sokrates	C4	226
...	...	...	...	...	...	...	...
4630	Die 3 Kritiken	4	2137	2137	Kant	C4	7

⇓ **where**-Bedingung

Beispiel zur Verarbeitung mit group by

VorlNr	Titel	SWS	gelesen Von	PersNr	Name	Rang	Raum
5001	Grundzüge	4	2137	2137	Kant	C4	7
5041	Ethik	4	2125	2125	Sokrates	C4	226
5043	Erkenntnistheorie	3	2126	2126	Russel	C4	232
5049	Mäeutik	2	2125	2125	Sokrates	C4	226
4052	Logik	4	2125	2125	Sokrates	C4	226
5052	Wissenschaftstheorie	3	2126	2126	Russel	C4	232
5216	Bioethik	2	2126	2126	Russel	C4	232
4630	Die 3 Kritiken	4	2137	2137	Kant	C4	7

⇓ Gruppierung

Beispiel zur Verarbeitung mit group by

VorlNr	Titel	SWS	gelesenVon	PersNr	Name	Rang	Raum
5041	Ethik	4	2125	2125	Sokrates	C4	226
5049	Mäeutik	2	2125	2125	Sokrates	C4	226
4052	Logik	4	2125	2125	Sokrates	C4	226
5043	Erkenntnistheorie	3	2126	2126	Russel	C4	232
5052	Wissenschaftstheo.	3	2126	2126	Russel	C4	232
5216	Bioethik	2	2126	2126	Russel	C4	232
5001	Grundzüge	4	2137	2137	Kant	C4	7
4630	Die 3 Kritiken	4	2137	2137	Kant	C4	7

⇓ **having**-Bedingung

Beispiel zur Verarbeitung mit group by

VorlN	Titel	SWS	gelesenVon	PersNr	Name	Rang	Raum
5041	Ethik	4	2125	2125	Sokrates	C4	226
5049	Mäeutik	2	2125	2125	Sokrates	C4	226
4052	Logik	4	2125	2125	Sokrates	C4	226
5001	Grundzüge	4	2137	2137	Kant	C4	7
4630	Die 3 Kritiken	4	2137	2137	Kant	C4	7

⇓ Aggregation (**sum**) und Projektion

gelesenVon	Name	**sum** (SWS)
2125	Sokrates	10
2137	Kant	8

Bedingungen für Gruppen

- *[* **HAVING** *bedingung]*

- Die Bedingung einer **HAVING**-Klausel wirken auf Gruppen:
- Im Ergebnis werden nur Gruppen berücksichtigt, die die Bedingung erfüllen.

- Beispiel:
  - Welche Studenten haben eine Durchschnittsnote von 1,7 oder besser?

Mengenoperationen

- **union** - Vereinigung von Mengen
- **intersect** - Schnitt von Mengen
- **except** - Differenz von Mengen

- Die Ergebnisse beider **select** - Anweisungen müssen bezüglich Anzahl und Typ der Attribute identisch sein.
- Ohne das Schlüsselwort **all** werden vor der Mengenoperation Duplikate aus den beiden Teilergebnissen entfernt.
- Die beteiligten **select** - Anweisungen dürfen nicht in Klammern geschrieben werden.

Beispiel

- Mengenoperation mit geschachtelter Anfrage

- ```
  select Name
      from Assistenten
  union all
  select Name
      from Professoren;
  ```

- Keine Duplikate wenn **all** nicht verwendet wird
  ```

CORRESPONDING

- **CORRESPONDING** schränkt das Ziel der Mengenoperation ein.
- Wird **CORRESPONDING** bei Mengenausdrücken
  - nicht angegeben, so werden die Spalten der Reihenfolge nach berücksichtigt
  - ohne Spalten-Namen angegeben, so werden alle Spalten berücksichtigt, die in beiden Tabellenausdrücken vorkommen.
  - mit Spalten-Namen angegeben, so werden nur die angegebenen Spalten berücksichtigt.

Beispiel

- **CORRESPONDING**

- ```
  select Name, PersNr
      from Assistenten
  intersect corresponding (PersNr)
  select Name, PersNr
      from Professoren;
  ```
  ```

Geschachtelte Anfragen

- Unteranfrage in der where-Klausel
- Welche Prüfungen sind besser als durchschnittlich verlaufen?

- ```
  select *
      from prüfen
      where Note < ( select avg (Note)
                   from prüfen );
  ```
  ```

Geschachtelte Anfragen

- Unteranfrage in der **select**-Klausel
- Für jedes Ergebnistupel wird die Unteranfrage ausgeführt
- Man beachte, dass die Unteranfrage *korreliert* ist (greift auf Attribute der umschließenden Anfrage zu)

- ```
  select PersNr, Name, ( select sum (SWS) as Lehrbelastung
                         from Vorlesungen
                         where gelesenVon=PersNr )
  from Professoren;
  ```
  ```

Unkorrelierte versus korrelierte Unteranfragen

- ```
  select s.*
       from Studenten s
       where exists
             (select p.*
             from Professoren p
             where p.GebDatum > s.GebDatum);
  ```

korreliert

- ```
  select s.*
       from Studenten s
       where s.GebDatum <
             (select max (p.GebDatum)
             from Professoren p);
  ```

unkorreliert

- Vorteil:
- Unteranfrage braucht nur einmal ausgewertet zu werden

Verwendung von ANY / ALL

- Die Schlüsselwörter **ANY** und **ALL** können in der **WHERE** und **HAVING** Klauseln in Verbindung mit Sub-Selects genutzt werden
- **ANY** liefert true und damit eine Ergebnismenge wenn irgendein Ergebnis des Sub-Selects die Bedingung erfüllt
- **ALL** liefert true damit eine Ergebnismenge wenn alle Ergebnisse des Sub-Selects die Bedingung erfüllen

- Beispiel:
- ```
  select s.name
  from student s, prüfen p
  where s.matnr = p.matnr
      and p.prNr = 2323
      and p.note <= all
      (select p.note from prüfung p where prNr = 2323);
  ```
  ```

Bedingte Ausdrücke

- *bedingter-ausdruck ::= (case-ausdruck | coalesce-ausdruck | ...)*

- *case-ausdruck ::=*

  CASE

  { **WHEN** *bedingungs-ausdruck* **THEN** *skalar-ausdruck }*$^{+}$

  [**ELSE** *skalar-ausdruck]*

  END

- *coalesce-ausdruck ::=*

  COALESCE *“*(*” skalar-ausdruck-liste “*)*”* .

„erster nicht-null Wert, der der Funktion übergeben wird, wird zurückgegeben, sonst null“

Bedingte Ausdrücke

- Beispiele:
- ```
  select player_name,
       year,
       case when year = 'SR'
            then 'yes'
            else null
            end
            as is_a_senior
  from college_football_players
  ```

- ```
  select Name, Color, ProductNumber,
  coalesce(Color, ProductNumber) as FirstNotNull
  from Products ;
  ```

Spezielle Sprachkonstrukte ("syntaktischer Zucker")

- ```
  select *
     from Studenten
     where Semester > = 1 and Semester < = 4;
  ```

- ```
  select *
  from Studenten
  where Semester between 1 and 4;
  ```

- ```
  select *
  from Studenten
  where Semester in (1,2,3,4);
  ```
  ```

Pattern Matching / String Funktionen

- Vergleich mit Zeichenketten: **LIKE**
- Vergleich mit Regulären Ausdrücken: **SIMILAR TO**
- Groß-/Kleinschreibung ist bei Vergleich (meist) relevant
- String Funktionen
- upper (*String*) lower (*String*)
- char_length (*String*) substring (*String* [**FROM** *int*] [**FOR** *int*])
- u.v.m. → PostgreSQL-Hilfe → String Functions and Operators

- Formatierungen
- to_char (*Wert*, *Format-String*) → bspw. to_char(avg(Note)/10,'FM9D0')
- u.v.m. → PostgreSQL-Hilfe → Data Type Formatting Functions

String-Vergleiche mit like

- Platzhalter "%" ; "_"
- "%" steht für beliebig viele (auch gar kein) Zeichen
- "_" steht für genau ein Zeichen

- ```
  select *
     from Studenten
     where Name like `T_eophrastos‘;
  ```

- ```
  select distinct s.Name
     from Vorlesungen v, hören h, Studenten s
     where s.MatrNr = h.MatrNr and h.VorlNr = v.VorlNr and
       v.Titel like `%thik%‘;
  ```

Nullwerte

- unbekannter Wert
- wird vielleicht später nachgereicht
- Nullwerte können auch im Zuge der Anfrageauswertung entstehen (Bsp. äußere Joins)
- manchmal sehr überraschende Anfrageergebnisse, wenn Nullwerte vorkommen

```
select count (*)
from Studenten
where Semester < 13 or Semester > =13
```

- Wenn es Studenten gibt, deren Semester-Attribut den Wert **null** hat, werden diese nicht mitgezählt
- Der Grund liegt in folgenden Regeln für den Umgang mit **null**-Werten begründet:

Auswertung bei Null-Werten

- In arithmetischen Ausdrücken werden Nullwerte propagiert, d.h. sobald ein Operand **null** ist, wird auch das Ergebnis **null**. Dementsprechend wird z.B. **null** + 1 zu null ausgewertet - aber auch **null** * 0 wird zu **null** ausgewertet.
- **null** Werte machen bei *Vergleichsoperationen* in Anfragen eine spezielle Logik nötig.
- SQL hat eine dreiwertige Logik, die nicht nur **true** und **false** kennt, sondern auch einen dritten Wert **unknown**. Diesen Wert liefern Vergleichsoperationen zurück, wenn mindestens eines ihrer Argumente null ist. Beispielsweise wertet SQL das Prädikat (PersNr=...) immer zu **unknown** aus, wenn die **PersNr** des betreffenden Tupels den Wert **null** hat.
- Diese Behandlung gilt für alle Operatoren von Suchbedingungen mit Ausnahme von IS NULL und EXISTS
- Logische Ausdrücke werden nach den folgenden Tabellen berechnet:

Auswertung bei Null-Werten

not	
true	false
unknown	unknown
false	true

and	true	unknown	false
true	true	unknown	false
unknown	unknown	unknown	false
false	false	false	false

or	true	unknown	false
true	true	true	true
unknown	true	unknown	unknown
false	true	unknown	false

Auswertung bei Null-Werten

- Diese Berechnungsvorschriften sind recht intuitiv. **Unknown or true** wird z.B. zu **true** - die Disjunktion ist mit dem **true**-Wert des rechten Arguments immer erfüllt, unabhängig von der Belegung des linken Arguments. Analog ist **unknown and false** automatisch **false** - keine Belegung des linken Arguments könnte die Konjunktion mehr erfüllen.
- In einer **where**-Bedingung werden nur Tupel weitergereicht, für die die Bedingung **true** ist. Insbesondere werden Tupel, für die die Bedingung zu **unknown** auswertet, nicht ins Ergebnis aufgenommen.
- Bei einer Gruppierung wird **null** als ein eigenständiger Wert aufgefasst und in eine eigene Gruppe eingeordnet.

Sichten

- Sichten (**VIEWS**) in SQL sind virtuelle Tabellen die auf dem Ergebnismenge einer Anfrage basieren.
- Können Daten aus einem oder mehreren Tabellen enthalten.
- Views zeigen immer aktuelle Daten. Die Daten werden bei jedem Zugriff auf die View neu geladen.
- Ermöglichen auf bestimmte Use Cases zugeschnittene Sichten auf den Datenbestand.

- z.B.: Datenschutz
-
```
create view prüfenSicht as
    select MatrNr, VorlNr, PersNr
    from prüfen
```

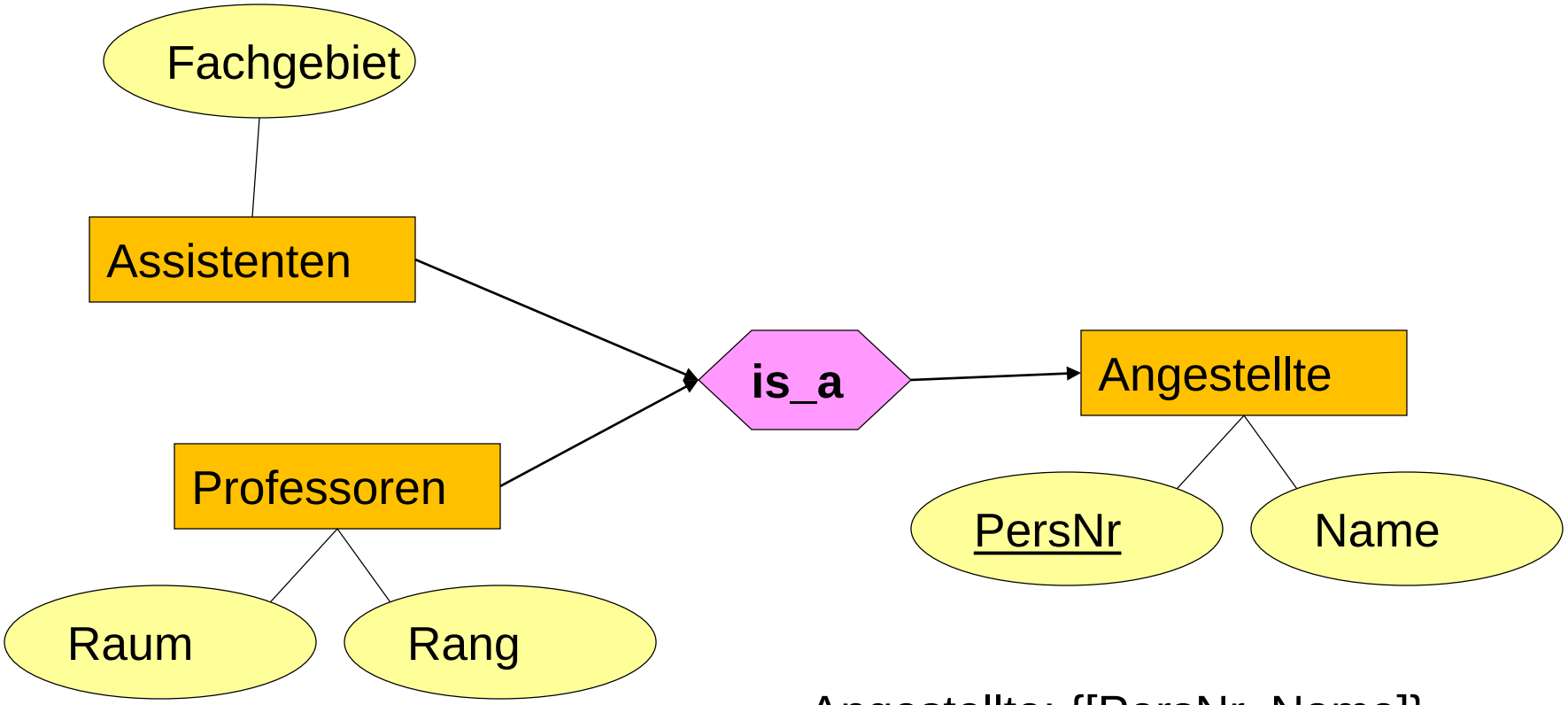
Sichten

- z.B.: für die Vereinfachung von Anfragen

- ```
  create view StudProf (Sname, Semester, Titel, Pname) as
      select s.Name, s.Semester, v.Titel, p.Name as PName
      from Studenten s, hören h, Vorlesungen v,
      Professoren p
      where s.Matr.Nr=h.MatrNr and h.VorlNr=v.VorlNr and
              v.gelesenVon = p.PersNr
  ```

- ```
  select distinct Semester
      from StudProf
      where PName=`Sokrates‘;
  ```

Relationale Modellierung der Generalisierung



- Angestellte: {[PersNr, Name]}
- Professoren: {[PersNr, Rang, Raum]}
- Assistenten: {[PersNr, Fachgebiet]}

Sichten zur Modellierung von Generalisierung

- ```
  create table Angestellte
      (PersNr     integer primary key,
      Name  varchar (30) not null);
  ```

- ```
  create table ProfDaten
      (PersNr     integer primary key,
      Rang        character(2),
      Raum        integer);
  ```

- ```
  create table AssiDaten
      (PersNr     integer primary key,
      Fachgebiet  varchar(30),
      Boss        integer);
  ```
  ```

Sichten zur Modellierung von Generalisierung

- ```
  create view Professoren as
      select *
      from Angestellte a, ProfDaten d
      where a.PersNr=d.PersNr;
  ```

- ```
  create view Assistenten as
      select *
      from Angestellte a, AssiDaten d
      where a.PersNr=d.PersNr;
  ```

- → Untertypen als Sicht

Zusammenfassung

- **CREATE TABLE**
- **PRIMARY KEY** ( ), **FOREIGN KEY** ( ) **REFERENCES** ... **ON DELETE** ...
- **ALTER TABLE** *(* **ADD** | **DROP** *)* *(* **COLUMN** | **CONSTRAINT** *)*
- **INSERT INTO** ... ( ) **VALUES** ( )
- **DELETE FROM** ... **WHERE** **UPDATE** ... **SET** ... **WHERE**
- **SELECT** ***DISTINCT*** ... **FROM** ... **WHERE**
- Tabellenreferenzen: ... **AS**
- **NATURAL JOIN**
- **INNER JOIN** | **OUTER JOIN** Join-Bedinungen: *(* **ON** | **USING** *)*
- Gruppierung: **GROUP BY, HAVING** Aggregatfunktionen
- Bedingte Ausdrücke: **CASE, COALESCE**
- Pattern Matching: **LIKE, SIMILAR TO**
- **VIEWS**