

Datenbanksysteme

Kap 3: SQL

Relationale Datenbanksprachen

- Formale Anfragekalküle
 - Relationale Algebra, Tupel/Domänenkalkül
- SQL – Structured Query Language
 - Als SEQUEL 1974-77 bei IBM für System R entwickelt
- QUEL (Query Language)
 - Zeitgleich zu SQL an der Berkely University entwickelt
 - Trotz "Überlegenheit zu SQL in vielen Bereichen" (Date) keine Marktdurchdringung
- QBE – Query by Example
 - Intuitiver grafischer Zugriff
 - In Frontends für "Joe User" realisiert (z.B. MS Access)
 - Meist basierend auf einem SQL-Unterbau

SQL Standardisierung

- SQL wird innerhalb der ANSI/ISO standardisiert
- Wichtige Etappen der Standardisierung :
 - SQL (1986, erste Standardisierung)
 - SQL2, SQL-92 (große Überarbeitung und Erweiterung)
 - SQL3, SQL:99 (objektrelationale Erweiterungen)
 - SQL:2003, SQL:2008 (Interoperabilität mit XML)
 - SQL:2011 (Temporale Erweiterungen)
 - SQL:2016 (Interoperabilität mit JSON)
- Heutige DBS unterstützen eine Obermenge einer Untermenge von SQL2
 - Viele proprietäre Erweiterungen
 - Portierung von einem zum anderen DBS schwierig
 - Gefahr des "Vendor Lockin"

Eigenschaften von SQL

- Deklarativ
 - Mit SQL spezifiziert man das **WAS** (welche Daten man haben will), nicht das **WIE** des Datenzugriffs
- Sequentiell
 - Kommandos werden sequentiell abgearbeitet
 - Keine Programmiersprache; insbesondere fehlen Variablen, Kontrollflusssteuerung, Prozeduren
 - Zur Realisierung einer Geschäftslogik muss SQL im allgemeinen in eine "Host-Sprache" eingebettet werden
- Prozedurale Erweiterungen
 - Z.B. PL/SQL (Oracle) und PL/PgSQL (PostgreSQL)
 - Werden wir im Zusammenhang mit Stored Procedures und Triggern behandeln

Informelle Unterteilung in

- **DDL – Data Definition Language**
 - Definition und Änderung des DB-Schemas und anderer Strukturen
 - Kommandos: CREATE, ALTER, DROP
- **DML – Data Manipulation Language**
 - Abfrage und Manipulation der Daten
 - Kommandos: SELECT, INSERT, UPDATE und DELETE
- **DCL – Data Control Language**
 - Steuerung des Datenzugriffs und der Datensicherheit
 - Kommandos:
 - BEGIN, COMMIT, ROLLBACK (Transaktionen)
 - GRANT, REVOKE (Rechteverwaltung)

SQL Syntax

- Kommandos durch Semikolon (;) getrennt
 - Nicht immer: in SQL-Interpreter ja, ESQL nicht
- Keywords und Identifier nicht case-sensitiv
 - Ausnahme: *quoted identifier* (z.B. "Bla" <> bla)
 - Zulässige Identifier: [_A-Za-z] [_A-Za-z0-9]*
- String-Konstanten in single quotes: 'bla bla'
 - Single Quotes in Strings können durch Verdoppeln escaped werden: 'Peter' 's house'
- Kommentare
 - Einzeilige Kommentare durch Doppelminus: -- bla
 - Mehrzeilige Kommentare wie in C: /* bla
bla */
 - SQL3 erlaubt Schachtelung: /*/* bla */*/

Beispiel-Datenbank

Professoren			
<u>PersNr</u>	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		
<u>MatrNr</u>	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			
<u>VorINr</u>	Titel	SWS	gelesenVon
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

Assistenten			
<u>PersINr</u>	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

voraussetzen	
<u>Vorgänger</u>	<u>Nachfolger</u>
5001	5041
5001	5043
5001	5049
5041	5216
5043	5052
5041	5052
5052	5259

hören	
<u>MatrNr</u>	<u>VorINr</u>
26120	5001
27550	5001
27550	4052
28106	5041
28106	5052
28106	5216
28106	5259
29120	5001
29120	5041
29120	5049
29555	5022
25403	5022

prüfen			
<u>MatrNr</u>	<u>VorINr</u>	<u>PersNr</u>	Note
28106	5001	2126	1
25403	5041	2125	2
27550	4630	2137	2

Ein SQL-Script zum Anlegen dieser Datenbank finden Sie im Moodle-Kurs.

CREATE TABLE – Anlegen einer Tabelle



- Definiert neue Tabelle mit Namen, Feldern, Wertebereichen und Beschränkungen
- 1. Näherung:

```
CREATE TABLE Vorlesungen (  
    VorlNr      INTEGER,  
    Titel       VARCHAR(30),  
    SWS         INTEGER,  
    gelesenVon  INTEGER  
);
```


Datentypen in SQL

- Jedes Feld kann nur Werte eines bestimmten Datentyps speichern

Datentyp	Beschreibung
CHARACTER(<i>n</i>) CHAR(<i>n</i>)	String der Länge <i>n</i> , am Ende ggf. mit Blanks aufgefüllt
CHAR VARYING(<i>n</i>) VARCHAR(<i>n</i>) TEXT	String mit variabler Länge (maximal <i>n</i>) Im allgemeinen gegenüber CHAR vorzuziehen Postgres-Datentyp für Strings variabler und unbegrenzter Länge
INTEGER, INT SMALLINT BIGINT	Ganzzahl mit Vorzeichen, Größe: 4 Bytes 2 Bytes 8 Bytes Postgres unterscheidet INT2, INT4, INT8
NUMERIC(<i>n</i> , <i>m</i>) NUMERIC(<i>n</i>)	Dezimalzahl mit <i>n</i> Stellen, davon <i>m</i> nach dem Komma; NUMERIC(<i>n</i>) = NUMERIC(<i>n</i> ,0)
BOOL	true, false oder unknown (Dreiwertige Logik)
DATE	Datum (4 Bytes, tagesgenau)
TIME	Uhrzeit (8 Bytes, mikrosekundengenau)
TIMESTAMP	Datum und Uhrzeit

Constraints

- Constraints werden **nach den Felddefinitionen** angegeben
 - Constraints, die sich nur **auf ein Feld beziehen**, können direkt bei der Felddefinition angegeben werden
 - Optional können Constraints **mit Namen versehen** werden

- **Beispiel**

```
CREATE TABLE example1 (  
    a INTEGER,  
    b INTEGER,  
    c VARCHAR(2) REFERENCES example2(a),  
    PRIMARY KEY(a, b),  
    CONSTRAINT check_b CHECK (b > 0)  
);
```

NOT NULL Constraint

- *<Attributdefinition>* NOT NULL
 - Keine Null-Werte erlaubt
 - Primärschlüssel-Attribute müssen immer NOT NULL sein

- Beispiel

```
CREATE TABLE Vorlesungen (  
    VorlNr INTEGER NOT NULL,  
    TITEL VARCHAR(30)  
);  
INSERT INTO Vorlesungen (VorlNr, Titel)  
VALUES (NULL, 'Grundlagen ...');
```

→ ERROR

Probleme beim Umgang mit NULL-Werten

- Empfehlung:
 - Wenn es geht, NULL-Werte verbieten!
 - Grund: Behandlung von NULL für den Applikationsprogrammierer unklar

- Beispiel:

```
CREATE Buch (  
    BuchID INT NOT NULL,  
    Ausleihfrist INT  
);
```

BuchID	Ausleihfrist
0815	30
4711	NULL

- Was passiert, wenn ein Benutzer das Buch '4711' ausleihen will?
 - Der "Vorsichtige": Buch nicht entleihbar
 - Der "Spendable": Leihfrist wird auf einen Standardwert gesetzt
 - Der "Sorglose": Fall wurde nicht bedacht → Programm stürzt ab

Default-Werte

- Mit der **DEFAULT**-Klausel kann ein Standard-Wert für eine Spalte angegeben werden
 - Wird verwendet, wenn beim INSERT kein expliziter Wert angegeben wird
- *<Attributdefinition>* **DEFAULT** *<Konstante>*
 DEFAULT *<Funktion>*
 DEFAULT NULL

- **Beispiele**

```
CREATE TABLE Vorlesungen (  
    SWS    INTEGER DEFAULT 4  
);  
  
CREATE TABLE Professoren (  
    Name   VARCHAR(30) DEFAULT CURRENT_USER  
);  
  
CREATE TABLE Studenten (  
    Name   VARCHAR(30) DEFAULT NULL  
);
```

Wertebereichs-Constraints mit CHECK

- Spaltenbezogene Constraints
 - Welche Werte für eine Spalte sind erlaubt?
 - Es reicht, sich das aktuell einzufügende/ändernde Tupel anzuschauen
- Syntax
 - *<Attributdefinition> CHECK (<Bedingung>)*
- Bedingungsausdrücke
 - Normalerweise statische Wertebereichseinschränkung
 - Vergleich des Attributs mit einer oder mehreren Konstanten

Beispiele für CHECK-Klausel

```
CREATE TABLE Vorlesungen (
```

```
    ...
```

```
    SWS    INTEGER CHECK (SWS>=2)
```

```
);
```

```
CREATE TABLE Professoren (
```

```
    ...
```

```
    Rang CHAR(2)
```

```
    CHECK (Rang IN ('C2', 'C3', 'C4'))
```

```
);
```

```
CREATE TABLE Studenten (
```

```
    ...
```

```
    Semester INTEGER
```

```
    CHECK (Semester BETWEEN 1 AND 14)
```

```
);
```

CHECK-Klauseln auf mehr als einer Spalte

- CREATE TABLE Artikel (
 Einkaufspreis NUMERIC
 CHECK (Einkaufspreis > 0),
 Verkaufspreis NUMERIC
 CHECK (Verkaufspreis > 0),
 CHECK (Einkaufspreis <= Verkaufspreis)
);

Alternativ:

- CREATE TABLE Artikel (
 Einkaufspreis NUMERIC
 CHECK (Einkaufspreis > 0),
 Verkaufspreis NUMERIC,
 CHECK (Verkaufspreis > 0 AND
 Einkaufspreis <= Verkaufspreis)
);

CHECK mit Subqueries

- Seit SQL-92 auch komplexe Bedingungen mit Subqueries möglich (dazu später mehr)
 - Allerdings von vielen DBMS-Herstellern noch nicht oder nur mit Einschränkungen unterstützt
- Beispiel
 - Stelle sicher, dass ein Händler nur aus einer Stadt kommt, in der auch Kunden wohnen
 - ```
CREATE TABLE Haendler (
 Stadt CHAR(30) CHECK (Stadt IN
 (SELECT Stadt FROM Kunde))
);
```

# Domains

- Mit Domains können häufig benötigte Einschränkungen auf Datentypen wiederverwendbar gemacht werden
- Syntax:
  - `CREATE DOMAIN domain_name AS data_type [CONSTRAINT constraint_name] CHECK (expression);`
- Beispiel:
  - `CREATE DOMAIN positive_int AS INTEGER CHECK (value > 0);`
- Verwendung:
  - `CREATE TABLE Product (  
...  
price positive_int  
);`

# NOT NULL/CHECK-Klausel

- Überprüfung ist Leistung des DBMS-Servers
  - Wird überprüft bei jedem Einfügen oder Ändern eines Tupels
- Vorteil
  - Überprüfung an zentraler Stelle
  - Aber: Applikation muss darauf vorbereitet sein, dass Einfüge/Änderungsoperation scheitern kann
- Nachteil
  - Mindert den Durchsatz
  - Beispiel:  
Name VARCHAR(40) NOT NULL CHECK (Name <> ' ')
- Empfehlung
  - Wohldosierter Gebrauch, insbesondere bei änderungsintensiven Anwendungen

# Beispiel: Spalten-Constraints

- CREATE TABLE Vorlesungen (  
    VorlNr          INTEGER NOT NULL,  
    Titel          VARCHAR(30) NOT NULL,  
    SWS             INTEGER CHECK (SWS>=2)  
                    DEFAULT 4,  
    gelesenVon INTEGER  
);

# Übersicht Tabellen-Constraints

- Tabellen-Constraints
  - Nicht nur ein einzelnes Tupel, sondern alle Tupel einer oder mehrerer Tabellen müssen berücksichtigt werden
- Intra-Tabelle (innerhalb einer Tabelle)
  - PRIMARY KEY (A,B)
    - Die Spalten A und B bilden den Primärschlüssel
  - UNIQUE (B,C)
    - Eindeutigkeit für alternative Schlüssel (B,C)
- Inter-Tabelle (über mehrere Tabellen)
  - FOREIGN KEY (C,D) REFERENCES S(C,D)
    - Referentielle Integrität
    - Fremdschlüssel (C,D)  $\rightarrow$  S.(C,D)
  - ASSERTION
    - Benutzerdefinierte Einschränkung über mehrere Tabellen

# PRIMARY KEY-Constraint

- In jeder Tabelle sollte ein(e) Attribut(kombination) als Primärschlüssel deklariert
  - Es darf keine zwei Tupel geben, die in den Primärschlüssel-Attributen identische Werte haben
  - PK-Attribute dürfen keinen NULL-Wert annehmen
  - PK-Attribute werden häufig durch Unterstreichen kenntlich gemacht

| Professoren   |            |      |      |
|---------------|------------|------|------|
| <u>PersNr</u> | 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    |

Bemerkung:  
MatrNr und VorlNr  
sind in der Tabelle  
hören für sich nicht  
eindeutig, aber die  
Kombination aus  
beiden Attributen!

| hören         |               |
|---------------|---------------|
| <u>MatrNr</u> | <u>VorlNr</u> |
| 26120         | 5001          |
| 27550         | 5001          |
| 27550         | 4052          |
| 28106         | 5041          |
| 28106         | 5052          |
| 28106         | 5216          |
| 28106         | 5259          |
| 29120         | 5001          |
| 29120         | 5041          |
| 29120         | 5049          |
| 29555         | 5022          |
| 25403         | 5022          |

# PRIMARY KEY-Constraint

Beispiel-Datenbank

| Tabelle 1 |    | Tabelle 2 |    | Tabelle 3 |    |
|-----------|----|-----------|----|-----------|----|
| PK        | FK | PK        | FK | PK        | FK |
| 1         | 2  | 3         | 4  | 5         | 6  |
| 2         | 3  | 4         | 5  | 6         | 7  |
| 3         | 4  | 5         | 6  | 7         | 8  |
| 4         | 5  | 6         | 7  | 8         | 9  |
| 5         | 6  | 7         | 8  | 9         | 10 |
| 6         | 7  | 8         | 9  | 10        | 11 |
| 7         | 8  | 9         | 10 | 11        | 12 |
| 8         | 9  | 10        | 11 | 12        | 13 |
| 9         | 10 | 11        | 12 | 13        | 14 |
| 10        | 11 | 12        | 13 | 14        | 15 |

- Beispiel: "inline"-Deklaration

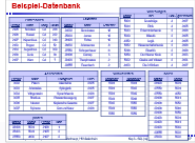
```
CREATE TABLE Professoren (
 PersNr INTEGER PRIMARY KEY,
 Name VARCHAR(30) NOT NULL,
);
```

- Beispiel: getrennte Deklaration

- wenn PK aus mehr als einem Attribut besteht

```
CREATE TABLE hören (
 MatrNr INTEGER,
 VorlNr INTEGER,
 PRIMARY KEY (MatrNr, VorlNr)
);
```

# UNIQUE-Constraint

A small thumbnail image in the top right corner showing a database table with multiple columns and rows of data, likely representing the 'Beispiel-Datenbank' mentioned in the caption.

- Mit der **UNIQUE**-Klausel wird sichergestellt, dass die Werte eines Attributs oder einer Attributkombination eindeutig sind
  - UNIQUE-Attribute haben auch Schlüsseleigenschaft, sind also Schlüsselkandidaten
- Beispiel: "inline"-Deklaration
  - ```
CREATE TABLE Professoren (  
    PersNr INTEGER PRIMARY KEY,  
    Raum INTEGER UNIQUE,  
);
```
- Beispiel: getrennte Deklaration (bei mehr als einem Attribut)
 - ```
CREATE TABLE Assistenten (
 Name VARCHAR(30),
 Fachgebiet VARCHAR(30),
 UNIQUE (Name, Fachgebiet)
);
```



# ASSERTIONS

- Mit Assertions lassen Integritätsconstraints über mehrere Tabellen hinweg sicherstellen
- Beispiel
  - Höchstens 30% aller Bücher einer Bibliothek sollen vorgemerkt werden können
  - ```
CREATE ASSERTION CHECK (  
    (SELECT COUNT(*) FROM Vormerkungen) /  
    (SELECT COUNT(*) FROM Bücher) <= 0.3)  
);
```
- **ASSERTION** wird *außerhalb* der Tabellendeklaration von Vormerkungen und Bücher spezifiziert
- Bereits seit SQL-92 standardisiert, aber leider kaum umgesetzt

Fremdschlüssel-Constraint

Beispiel-Datenbank

Tabelle 1: Professoren				Tabelle 2: Vorlesungen			
PersNr	Name	Abteilung	Telefon	VorlesungNr	Tag	Uhrzeit	Dozent
1	Prof. Dr. Müller	Mathematik	01234	1	Mo	10:00	Prof. Dr. Müller
2	Prof. Dr. Schmidt	Physik	05678	2	Di	14:00	Prof. Dr. Schmidt
3	Prof. Dr. Weber	Chemie	09012	3	Do	09:00	Prof. Dr. Weber
4	Prof. Dr. Fischer	Biologie	03456	4	Fr	16:00	Prof. Dr. Fischer
5	Prof. Dr. Klein	Medizin	07890	5	Sa	08:00	Prof. Dr. Klein

- Zur Erinnerung: Referentielle Integrität
 - Fremdschlüssel müssen auf existierende Tupel verweisen oder einen Nullwert enthalten
- Fremdschlüssel-Constraint wird im CREATE TABLE-Statement der referenzierenden Tabelle spezifiziert
 - ```
CREATE TABLE <RefTable> (
 <AttributDef> REFERENCES
 <MasterTable>[(<Attributname>)]
)
```
- (<Attributname>) kann weggelassen werden
  - Dann wird der PRIMARY KEY der Master-Tabelle als Ziel des Fremdschlüssel-Verweises genommen
  - Datentypen der referenzierenden und referenzierten Attribute müssen passen
- Beispiel:

```
CREATE TABLE Vorlesungen (
 ...
 gelesenVon INTEGER REFERENCES Professoren(PersNr)
)
```

# Fremdschlüssel-Constraint

- Ein Fremdschlüssel kann auch aus einer Kombination von Attributen bestehen
- In diesem Fall getrennte Deklaration des FK-Constraints hinter den Attributdefinitionen
  - `FOREIGN KEY (A, B) REFERENCES <MasterTable>(C,D)`
- Beispiel:
  - ```
CREATE TABLE Ausleihungen (  
    BuchNr INTEGER,  
    Vorname VARCHAR(30),  
    Name VARCHAR(30),  
    FOREIGN KEY (Vorname, Name) REFERENCES  
        Person(Vorname, Name)  
    )
```

Master- und Referenztabellen

Professoren			
<u>PersNr</u>	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

Vorlesungen			
<u>VorINr</u>	Titel	SWS	gelesenVon
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

Hören	
<u>MatrNr</u>	<u>VorINr</u>
26120	5001
27550	5001
27550	4052
28106	5041
28106	5052
28106	5216
28106	5259
29120	5001
29120	5041
29120	5049
29555	5022
25403	5022

Einhaltung referentieller Integrität

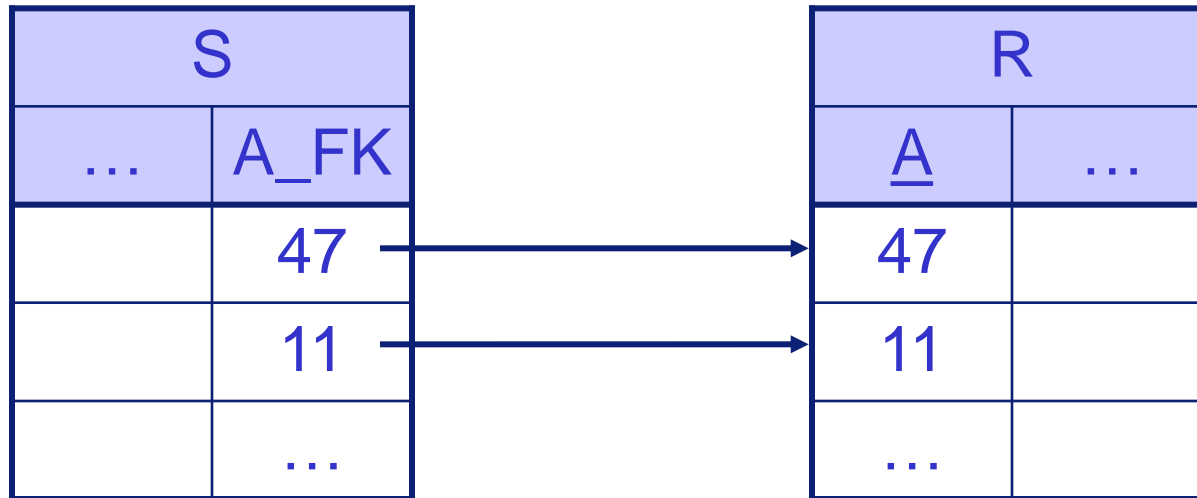
- Problem:
 - Was geschieht bei Änderungen (Updates oder Deletes) von Daten der referenzierten Tabelle (Mastertabelle)?
- Drei mögliche Reaktionen
 - `ON UPDATE/DELETE NO ACTION` (Standardeinstellung)
 - Zurückweisen der Änderungsoperation
 - `ON UPDATE/DELETE CASCADE`
 - Wenn der Schlüssel eines referenzierten Tupels geändert wird, wird diese Änderung in den Fremdschlüsselattributen der referenzierenden Tupel nachgezogen
 - `ON UPDATE/DELETE SET NULL`
 - Die Verweise in den referenzierenden Tupeln werden auf NULL gesetzt
 - Die Reaktionen können für Updates und Deletes unterschiedlich gesetzt werden

Beispiel

S:
referenzierende Tabelle

Ausgangszustand

R:
referenzierte Tabelle

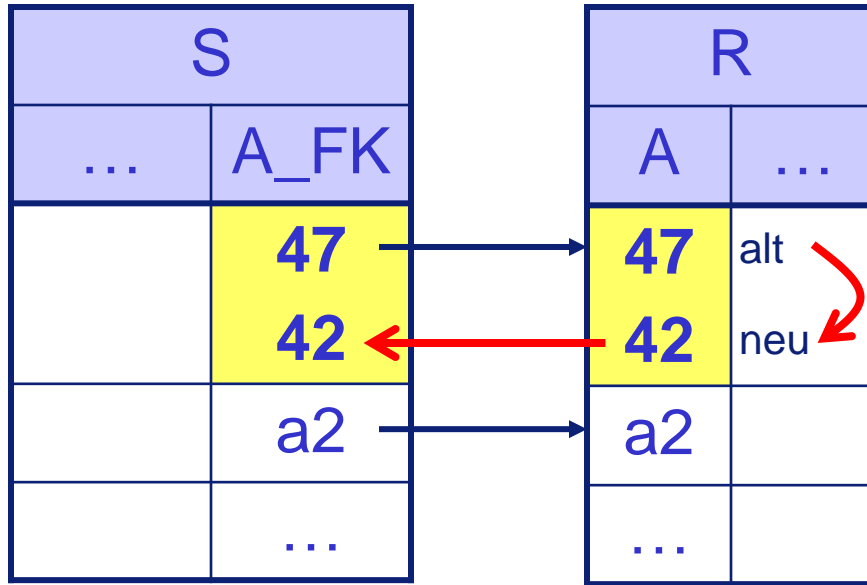


- Betrachte folgende Änderungsoperationen

UPDATE R
SET A = 42
WHERE A = 47

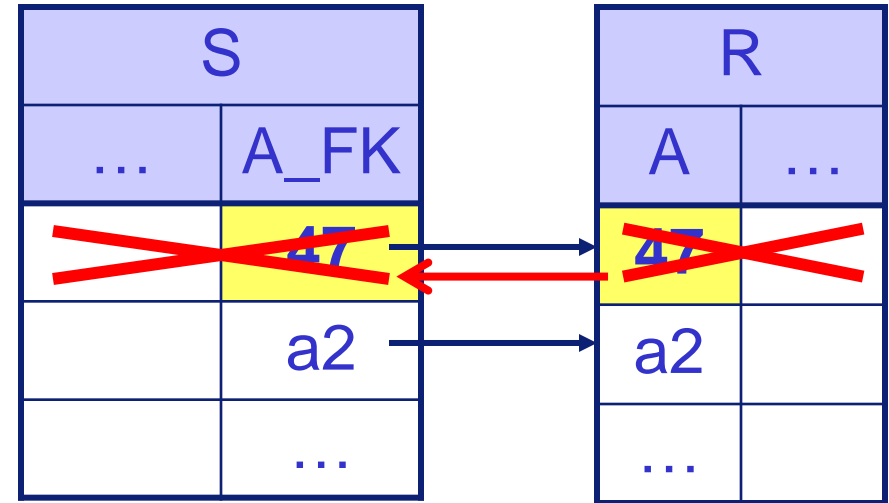
DELETE FROM R
WHERE A = 47

Kaskadierendes Ändern/Löschen



```
CREATE TABLE S (
  A_FK INTEGER REFERENCES R(A)
    ON UPDATE CASCADE
)
```

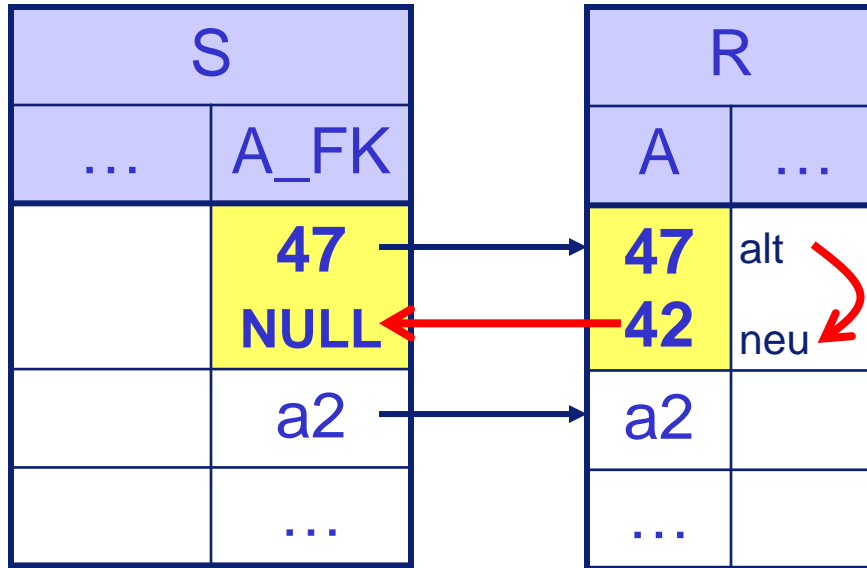
```
UPDATE R
  SET A = 42
  WHERE A = 47
```



```
CREATE TABLE S (
  A_FK INTEGER REFERENCES R(A)
    ON DELETE CASCADE
)
```

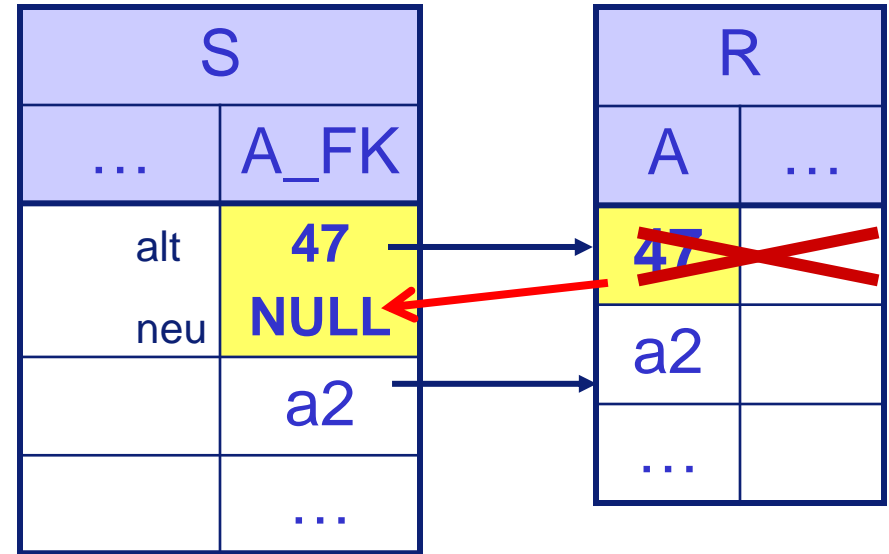
```
DELETE FROM R
  WHERE A = 47
```

Auf Null setzen



```
CREATE TABLE S (
  A_FK INTEGER REFERENCES R(A)
    ON UPDATE SET NULL
)
```

```
UPDATE R
  SET A = 42
  WHERE A = 47
```



```
CREATE TABLE S (
  A_FK INTEGER REFERENCES R(A)
    ON DELETE SET NULL
)
```

```
DELETE FROM R
  WHERE A = 47
```


Ausblick: Datenbank-Trigger

- ON UPDATE/DELETE CASCADE/SET NULL
 - Spezialfall einer referentiell ausgelösten **Korrekturaktion**
- Benutzerdefinierte Korrekturaktionen können mit Hilfe von Triggern definiert werden
 - Standardisiert seit SQL:99
 - Standard allerdings nur teilweise in DBMS-Produkten umgesetzt, häufig noch proprietäre Syntax
 - Einige Beispiele dazu später

-);

Das komplette Schema der Hochschul-DB (1)

```
CREATE TABLE Studenten (  
    MatrNr    INTEGER PRIMARY KEY,  
    Name      VARCHAR(30) NOT NULL,  
    Semester  INTEGER CHECK Semester BETWEEN 1 AND 18  
);
```

```
CREATE TABLE Professoren (  
    PersNr    INTEGER PRIMARY KEY,  
    Name      VARCHAR(30) NOT NULL,  
    Rang      CHAR(2) CHECK (Rang IN ('C2', 'C3', 'C4')),  
    Raum      INTEGER UNIQUE  
);
```

Das komplette Schema der Hochschul-DB (2)

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

```
CREATE TABLE Vorlesungen (  
    VorlNr          INTEGER PRIMARY KEY,  
    Titel           VARCHAR(30),  
    SWS             INTEGER,  
    gelesenVon      INTEGER REFERENCES Professoren  
                                ON DELETE SET NULL  
);
```

Hochschulschema mit Integritätsbedingungen (3)

```
CREATE TABLE hören (  
    MatrNr          INTEGER REFERENCES Studenten  
                   ON DELETE CASCADE,  
    VorlNr          INTEGER REFERENCES Vorlesungen  
                   ON DELETE CASCADE,  
    PRIMARY KEY (MatrNr, VorlNr)  
);
```

```
CREATE TABLE voraussetzen (  
    Vorgänger       INTEGER REFERENCES Vorlesungen  
                   ON DELETE CASCADE,  
    Nachfolger       INTEGER REFERENCES Vorlesungen  
                   ON DELETE NO ACTION,  
    PRIMARY KEY (Vorgänger, Nachfolger)  
);
```

Hochschulschema mit Integritätsbedingungen (4)

```
CREATE TABLE prüfen (  
    MatrNr INTEGER REFERENCES Studenten  
            ON DELETE CASCADE,  
    VorlNr INTEGER REFERENCES Vorlesungen  
            ON DELETE SET NULL,  
    PersNr INTEGER REFERENCES Professoren  
            ON DELETE SET NULL,  
    Note NUMERIC(2,1) CHECK (Note BETWEEN 0.7  
                                AND 5.0),  
    PRIMARY KEY (MatrNr, VorlNr)  
);
```

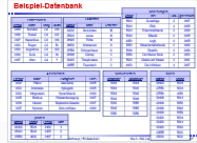
DROP/ALTER TABLE-Anweisung

- DROP TABLE <Tabellenname>
 - Löscht alle Tupel aus der Tabelle
 - Entfernt Tabellenkopf aus dem Datenbankschema

ALTER TABLE-Anweisung

- ALTER TABLE <Tabellenname>
 <Aktion>
 - Hinzufügen und Entfernen einer Spalte
 - Ändern eines Datentyps (z.B. Vergrößern einer Stringbreite)
 - Hinzufügen und Entfernen einer Tabellenbedingung
 - Darf nicht zu inkonsistentem DB-Zustand führen
 - z.B. nicht erfüllte Wertebereichsbedingungen oder ref. Integrität
 - Große Unterschiede in der Implementierung zwischen RDBMS-Herstellern

Beispiel: Spalte hinzufügen/löschen/ändern



Thumbnail of a database interface showing a table with columns like Name, Vorname, and Matrikelnummer.

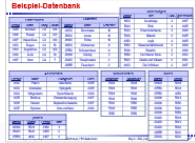
- ALTER TABLE Vorlesungen
ADD Credits TINYINTEGER;
- ALTER TABLE Vorlesungen
ALTER COLUMN Credits INTEGER;
- Nachträgliches Hinzufügen eines Foreign Keys
ALTER TABLE Vorlesungen
ADD CONSTRAINT FK_GelesenVon
FOREIGN KEY GelesenVon REFERENCES
Professoren(PersNr)

Anmerkungen

- Neues Attribut wird "hinten" angefügt
- Bereits existierende Tupel erhalten für das neue Attribut NULL-Wert oder Default-Wert
- NOT NULL-Klausel daher nur erlaubt, wenn gleichzeitig Default-Wert angegeben wird
- Löschen/Ändern einer Spalte nicht erlaubt, wenn
 - sie in einem Index verwendet wird
 - sie in einem PRIMARY KEY, UNIQUE, FOREIGN KEY oder CHECK-Constraint verwendet wird
- Ändern eines Spaltennamens
 - In Oracle über RENAME
 - Geht in MS-SQL nur über eine eingebaute Stored Procedure

Default-Werte hinzufügen/entfernen

Beispiel-Datenbank



- Default-Wert hinzufügen
 - ALTER TABLE Vorlesungen
ADD CONSTRAINT DF1 DEFAULT 4 FOR SWS;
- Default-Wert entfernen
 - ALTER TABLE Vorlesungen
DROP CONSTRAINT DF1;

Datenmanipulationssprache (DML)

- INSERT: Einfügen von Tupeln
 - DELETE: Entfernen von Tupeln
 - UPDATE: Ändern von Tupeln
-
- SELECT: Abfragen

INSERT-Anweisung

- `INSERT INTO <Tabellenname>`
`VALUES (v1, v2, ..., vn)`
 - Werte müssen in der passenden Reihenfolge angegeben werden
 - Beispiel:
`INSERT INTO Studenten`
`VALUES (24002, 'Xenokrates', 18);`
- Explizite Angabe von Spaltennamen möglich
 - Empfehlenswertere Variante (Warum?)
 - Fehlende Spalten werden mit NULL- oder Default-Werten gefüllt
 - Aufpassen bei NOT NULL-Spalten!
 - Beispiel:
`INSERT INTO Studenten`
`(MatrNr, Name) VALUES`
`(28121, 'Archimedes');`

Studenten		
MatrNr	Name	Semester
29120	Theophrastos	2
29555	Feuerbach	2
24002	Xenokrates	18
28121	Archimedes	NULL

INSERT-Anweisung

Beispiel-Datenbank

Tabelle 1: Studenten				Tabelle 2: Vorlesungen			
MatrNr	Nachname	Vorname	Geburtsdatum	VorlNr	Titel	Stunde	Dozent
1001	Meier	Anna	1990-01-15	1	Logik	10-12	Dr. Müller
1002	Schmidt	Thomas	1988-03-22	2	Mathematik	14-16	Prof. Weber
1003	Wagner	Sarah	1992-05-10	3	Physik	18-20	Dr. Klein
1004	Koch	Michael	1985-07-08	4	Chemie	08-10	Prof. Hoffmann
1005	Beck	Lena	1995-09-03	5	Informatik	12-14	Dr. Richter

- Übernahme von Werten aus Abfragen, z.B.
 - Alle Studenten hören die Logik-Vorlesung
INSERT INTO hören
SELECT MatrNr, VorlNr
FROM Studenten, Vorlesungen
WHERE Titel = 'Logik';
 - Der Professor, der 'Logik' liest, gibt auch die neue 2-stündige Vorlesung 'Logik 2' mit der Vorlesungsnummer '4711'

Spezielle Kommandos zum Füllen von Tabellen

- INSERT INTO ... SELECT ermöglicht das Füllen von Tabellen aus anderen Tabellen
- Nicht aber aus einer Datei
- Dafür gibt es DBMS-spezifische Tools, z.B.
 - sqlldr (Oracle)
 - \copy-Befehl in psql (PostgreSQL)
 - Beispiel: Kopieren von Daten in eine Tabelle aus einer csv-Datei mit Semikolon als Trennzeichen
 - `\copy tablename from 'filename.csv' delimiter ';'`

DELETE-Anweisung

Beispiel-Datenbank

Tab.	Spalte	Datum	Text	Nummer
1	1	2000-01-01	1	1
1	2	2000-01-01	1	1
1	3	2000-01-01	1	1
1	4	2000-01-01	1	1
1	5	2000-01-01	1	1
1	6	2000-01-01	1	1
1	7	2000-01-01	1	1
1	8	2000-01-01	1	1
1	9	2000-01-01	1	1
1	10	2000-01-01	1	1
1	11	2000-01-01	1	1
1	12	2000-01-01	1	1
1	13	2000-01-01	1	1
1	14	2000-01-01	1	1
1	15	2000-01-01	1	1
1	16	2000-01-01	1	1
1	17	2000-01-01	1	1
1	18	2000-01-01	1	1
1	19	2000-01-01	1	1
1	20	2000-01-01	1	1
1	21	2000-01-01	1	1
1	22	2000-01-01	1	1
1	23	2000-01-01	1	1
1	24	2000-01-01	1	1
1	25	2000-01-01	1	1
1	26	2000-01-01	1	1
1	27	2000-01-01	1	1
1	28	2000-01-01	1	1
1	29	2000-01-01	1	1
1	30	2000-01-01	1	1
1	31	2000-01-01	1	1
1	32	2000-01-01	1	1
1	33	2000-01-01	1	1
1	34	2000-01-01	1	1
1	35	2000-01-01	1	1
1	36	2000-01-01	1	1
1	37	2000-01-01	1	1
1	38	2000-01-01	1	1
1	39	2000-01-01	1	1
1	40	2000-01-01	1	1
1	41	2000-01-01	1	1
1	42	2000-01-01	1	1
1	43	2000-01-01	1	1
1	44	2000-01-01	1	1
1	45	2000-01-01	1	1
1	46	2000-01-01	1	1
1	47	2000-01-01	1	1
1	48	2000-01-01	1	1
1	49	2000-01-01	1	1
1	50	2000-01-01	1	1
1	51	2000-01-01	1	1
1	52	2000-01-01	1	1
1	53	2000-01-01	1	1
1	54	2000-01-01	1	1
1	55	2000-01-01	1	1
1	56	2000-01-01	1	1
1	57	2000-01-01	1	1
1	58	2000-01-01	1	1
1	59	2000-01-01	1	1
1	60	2000-01-01	1	1
1	61	2000-01-01	1	1
1	62	2000-01-01	1	1
1	63	2000-01-01	1	1
1	64	2000-01-01	1	1
1	65	2000-01-01	1	1
1	66	2000-01-01	1	1
1	67	2000-01-01	1	1
1	68	2000-01-01	1	1
1	69	2000-01-01	1	1
1	70	2000-01-01	1	1
1	71	2000-01-01	1	1
1	72	2000-01-01	1	1
1	73	2000-01-01	1	1
1	74	2000-01-01	1	1
1	75	2000-01-01	1	1
1	76	2000-01-01	1	1
1	77	2000-01-01	1	1
1	78	2000-01-01	1	1
1	79	2000-01-01	1	1
1	80	2000-01-01	1	1
1	81	2000-01-01	1	1
1	82	2000-01-01	1	1
1	83	2000-01-01	1	1
1	84	2000-01-01	1	1
1	85	2000-01-01	1	1
1	86	2000-01-01	1	1
1	87	2000-01-01	1	1
1	88	2000-01-01	1	1
1	89	2000-01-01	1	1
1	90	2000-01-01	1	1
1	91	2000-01-01	1	1
1	92	2000-01-01	1	1
1	93	2000-01-01	1	1
1	94	2000-01-01	1	1
1	95	2000-01-01	1	1
1	96	2000-01-01	1	1
1	97	2000-01-01	1	1
1	98	2000-01-01	1	1
1	99	2000-01-01	1	1
1	100	2000-01-01	1	1

- DELETE FROM <Tabellenname>
[WHERE <Bedingung>]
- DELETE FROM Studenten
WHERE Semester > 13;
- DELETE FROM Studenten;
 - Ohne WHERE-Bedingung werden alle Datensätze der Tabelle gelöscht. Aufpassen: es gibt kein Undo!
 - Alternative
 - TRUNCATE TABLE Studenten;
 - Unterschiede zu DELETE
 - Kein CASCADE/SET NULL bei Foreign Keys

UPDATE-Anweisung

Beispiel-Datenbank

Lehrer				Studenten			
Id	Name	Rang	Semester	Id	Name	Rang	Semester
1	Dr. Müller	C3	1	101	Max Müller	S1	1
2	Dr. Schmidt	C3	1	102	Anna Schmidt	S1	1
3	Dr. Weber	C3	1	103	Thomas Weber	S1	1
4	Dr. Meyer	C3	1	104	Sarah Meyer	S1	1
5	Dr. Klein	C3	1	105	Michael Klein	S1	1
6	Dr. Fischer	C3	1	106	Lena Fischer	S1	1
7	Dr. Bauer	C3	1	107	David Bauer	S1	1
8	Dr. Hoffmann	C3	1	108	Julia Hoffmann	S1	1
9	Dr. Richter	C3	1	109	Markus Richter	S1	1
10	Dr. Schulz	C3	1	110	Christina Schulz	S1	1

- UPDATE <Tabellenname>
SET <Attribut> = <Ausdruck>
[WHERE <Bedingung>]
- Alle C3-Profis zu C4-Profis befördern
 - UPDATE Professoren
SET Rang = 'C4'
WHERE Rang = 'C3';
- Alle Studenten ins nächste Semester
 - UPDATE Studenten
SET Semester = Semester + 1;

SELECT-Anweisung

- Allgemeine Form
 - SELECT <Attributliste>
FROM <Tabellenliste>
WHERE <Bedingung>
- <Attributliste>
 - Ausgabeattribute
- <Tabellenliste>
 - Tabellen, die an der Abfrage beteiligt sind
- <Bedingung>
 - Boolescher Ausdruck, mit dem Tupel gefiltert werden können
 - WHERE-Klausel kann fehlen → wird implizit als true ausgewertet

Beispiel: einfache SELECT-Abfrage

- Finde Professoren mit Rang C3 und gib deren Namen und Personalnummer aus
- SELECT** PersNr, Name
FROM Professor
WHERE Rang = 'C3';

Restrict

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

Project

PersNr	Name
2127	Kopernikus
2133	Popper
2134	Augustinus

*-Operator

- * in der SELECT-Klausel liefert alle verfügbaren Spalten
- Äquivalent:
 - SELECT PersNr, Name, Rang, Raum
FROM Professoren;
 - SELECT *
FROM Professoren;
- SELECT * ist flexibler
 - Wenn man das Tabellenschema nicht kennt
 - Aufpassen bei Änderungen des Tabellenschemas!

Operatoren in WHERE-Klausel

Operator	Beschreibung
=, >, <, >=, <=	gleich, kleiner, größer
<>, in Postgres auch !=	ungleich
BETWEEN x AND y	Bereichsprüfung
IN	Prüfung ob Wert in Menge enthalten ist
LIKE	Pattern Matching mit Wildcards _ (ein Zeichen), % (beliebig viele)
SIMILAR TO	Pattern Matching mit Posix 1003.2 regulären Ausdrücken (SQL3)
IS (NOT) NULL	Prüfung, ob Feld (nicht) leer ist

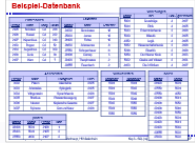
- Verknüpfung mit AND, OR
- Negation mit NOT

Wertebereiche in WHERE-Klauseln

```
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);
```



- LIKE ermöglicht Mustervergleich von Zeichenketten
 - %: beliebig viele (auch gar keine) Zeichen
 - _: genau ein Zeichen
- Beispiel
 - ```
SELECT *
FROM Studenten
WHERE Name LIKE 'T%eophrast_s';
```
- Bei weitem nicht so mächtig wie reguläre Ausdrücke
  - SIMILAR TO (SQL3) nur teilweise unterstützt
  - Viele RDBMS bieten proprietäre Erweiterungen für Freitextsuche, basierend auf POSIX Regular Expressions
  - PostgreSQL: siehe Abschnitt [Pattern Matching](#)

# "Dekodierung" mit dem CASE-Konstrukt

- ```
SELECT MatrNr, ( CASE
    WHEN Note < 1.5 THEN 'sehr gut'
    WHEN Note < 2.5 THEN 'gut'
    WHEN Note < 3.5 THEN 'befriedigend'
    WHEN Note < 4.0 THEN 'ausreichend'
    ELSE 'nicht bestanden'
END )
FROM Prüfen;
```
- Wert wird
 - von erster qualifizierender WHEN-Klausel übernommen
 - Oder von ELSE-Klausel, falls keine WHEN-Klausel gültig ist

Elimination von Duplikaten

- Ergebnistabellen können Duplikate enthalten
 - Performanzgründe: Identifikation der Duplikate erfordert teure Sortierung der Ergebnistabelle
- Duplikatelimination über das **DISTINCT**-Schlüsselwort
- `SELECT Rang`
`FROM Professoren;`

Rang
C4
C4
C3
C3
C3
C4
C4

`SELECT DISTINCT Rang`
`FROM Professoren;`

Rang
C4
C3

Sortierung der Ergebnistabelle

Beispiel-Datenbank

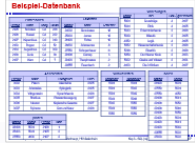
- Ergebnistabelle normalerweise unsortiert
- ORDER BY <Attribute> ASC|DESC
 - ASC: aufsteigend
 - DESC: absteigend
 - Sortierkriterien von links nach rechts
- SELECT PersNr, Name, Rang
FROM Professoren
ORDER BY Rang DESC,
Name ASC;

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

Umbenennung und konstante Ausgaben

- Spaltennamen im Ergebnis können umbenannt werden mit dem **AS**-Operator
- Auch Konstanten oder Ausdrücke können selektiert werden
- ```
SELECT Name,
 Semester * 6 AS "Studiendauer",
 'Monate' AS "Einheit"
FROM Studenten;
```

| Name         | Studiendauer | Einheit |
|--------------|--------------|---------|
| Xenokrates   | 108          | Monate  |
| Jonas        | 72           | Monate  |
| Fichte       | 60           | Monate  |
| Aristoxenos  | 48           | Monate  |
| Schopenhauer | 36           | Monate  |
| Carnap       | 18           | Monate  |
| Theophrastos | 12           | Monate  |
| Feuerbach    | 12           | Monate  |

A small thumbnail image in the top right corner showing a database table with multiple columns and rows of data, titled 'Beispiel-Datenbank'.

- Bisher Abfragen mit nur einer Eingabe-Tabelle
- In der FROM-Klausel können mehrere Tabellen aufgeführt werden
  - Durch Komma getrennt
  - SELECT ... FROM A, B
- Bedeutung
  - Kartesisches Produkt der beteiligten Tabellen
  - Kombiniere jedes Tupel der Tabelle A mit jedem Tupel der Tabelle B durch Konkatination
- SELECT \*  
FROM Professoren, Vorlesungen;

# Kartesisches Produkt

| Professoren   |          |      |      |
|---------------|----------|------|------|
| <u>PersNr</u> | Name     | Rang | Raum |
| 2125          | Sokrates | C4   | 226  |
| 2126          | Russel   | C4   | 232  |
| ⋮             | ⋮        | ⋮    | ⋮    |
| 2137          | Kant     | C4   | 7    |

| Vorlesungen   |                |     |            |
|---------------|----------------|-----|------------|
| <u>VorlNr</u> | Titel          | SWS | gelesenVon |
| 5001          | Grundzüge      | 4   | 2137       |
| 5041          | Ethik          | 4   | 2125       |
| ⋮             | ⋮              | ⋮   | ⋮          |
| 4052          | Logik          | 4   | 2125       |
| ⋮             | ⋮              | ⋮   | ⋮          |
| 4630          | Die 3 Kritiken | 4   | 2137       |

Kartesisches Produkt  
Professoren x Vorlesungen

| <u>PersNr</u> | Name     | Rang | Raum | <u>VorlNr</u> | Titel          | SWS | gelesenVon |
|---------------|----------|------|------|---------------|----------------|-----|------------|
| 2125          | Sokrates | C4   | 226  | 5001          | Grundzüge      | 4   | 2137       |
| 2125          | Sokrates | C4   | 226  | 5041          | Ethik          | 4   | 2125       |
| ⋮             | ⋮        | ⋮    | ⋮    | ⋮             | ⋮              | ⋮   | ⋮          |
| 2125          | Sokrates | C4   | 226  | 4052          | Logik          | 4   | 2125       |
| ⋮             | ⋮        | ⋮    | ⋮    | ⋮             | ⋮              | ⋮   | ⋮          |
| 2125          | Sokrates | C4   | 226  | 4630          | Die 3 Kritiken | 4   | 2137       |
| 2126          | Russel   | C4   | 232  | 5001          | Grundzüge      | 4   | 2137       |
| ⋮             | ⋮        | ⋮    | ⋮    | ⋮             | ⋮              | ⋮   | ⋮          |
| 2126          | Russel   | C4   | 232  | 4630          | Die 3 Kritiken | 4   | 2137       |
| ⋮             | ⋮        | ⋮    | ⋮    | ⋮             | ⋮              | ⋮   | ⋮          |
| 2137          | Kant     | C4   | 7    | 5001          | Grundzüge      | 4   | 2137       |
| ⋮             | ⋮        | ⋮    | ⋮    | ⋮             | ⋮              | ⋮   | ⋮          |
| 2137          | Kant     | C4   | 7    | 4630          | Die 3 Kritiken | 4   | 2137       |

# Verbund/Join von Tabellen

- Kartesisches Produkt von Tabellen für sich genommen meist nicht besonders interessant
- Stattdessen Verknüpfung von *zueinander passenden* Tupeln der beteiligten Tabellen
  - Meist über Fremdschlüsselbeziehungen
- Verknüpfung wird **Verbund** oder **Join** genannt
- Beispiel: Gib die Namen der Professoren zusammen mit den Titeln der von ihnen gelesenen Vorlesungen aus
  - ```
SELECT Name, Titel
FROM    Professoren, Vorlesungen
WHERE   PersNr = gelesenVon;
```

Konzeptionelle Abarbeitung eines Joins

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


Vorlesungen			
<u>VorlNr</u>	Titel	SWS	gelesenVon
5001	Grundzüge	4	2137
5041	Ethik	4	2125
⋮	⋮	⋮	⋮
4052	Logik	4	2125
⋮	⋮	⋮	⋮
4630	Die 3 Kritiken	4	2137

1. Schritt
Kartesisches Produkt
Professoren x Vorlesungen

<u>PersNr</u>	Name	Rang	Raum	<u>VorlNr</u>	Titel	SWS	gelesenVon
2125	Sokrates	C4	226	5001	Grundzüge	4	2137
2125	Sokrates	C4	226	5041	Ethik	4	2125
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
2125	Sokrates	C4	226	4052	Logik	4	2125
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
2125	Sokrates	C4	226	4630	Die 3 Kritiken	4	2137
2126	Russel	C4	232	5001	Grundzüge	4	2137
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
2126	Russel	C4	232	4630	Die 3 Kritiken	4	2137
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
2137	Kant	C4	7	5001	Grundzüge	4	2137
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
2137	Kant	C4	7	4630	Die 3 Kritiken	4	2137

Konzeptionelle Abarbeitung eines Joins

2. Schritt:
Auswertung der JOIN-Bedingung
WHERE PersNr = gelesenVon



PersNr	Name	Rang	Raum	VorlNr	Titel	SWS	gelesenVon
2125	Sokrates	C4	226	5001	Grundzüge	4	2137
2125	Sokrates	C4	226	5041	Ethik	4	2125
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
2125	Sokrates	C4	226	4052	Logik	4	2125
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
2125	Sokrates	C4	226	4630	Die 3 Kritiken	4	2137
2126	Russel	C4	232	5001	Grundzüge	4	2137
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
2126	Russel	C4	232	4630	Die 3 Kritiken	4	2137
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
2137	Kant	C4	7	5001	Grundzüge	4	2137
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
2137	Kant	C4	7	4630	Die 3 Kritiken	4	2137

Ergebnis des Joins

Professoren			
<u>PersNr</u>	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

Vorlesungen			
<u>VorINr</u>	Titel	SWS	gelesenVon
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

Nach der Auswertung der FROM und WHERE-Klauseln

<u>PersNr</u>	Name	Rang	Raum	<u>VorINr</u>	Titel	SWS	gelesenVon
2125	Sokrates	C4	226	5041	Ethik	4	2125
2125	Sokrates	C4	226	5049	Mäeutik	2	2125
2125	Sokrates	C4	226	4052	Logik	4	2125
2126	Russel	C4	232	5043	Erkenntnistheorie	3	2126
2126	Russel	C4	232	5052	Wissenschaftstheorie	3	2126
2126	Russel	C4	232	5216	Bioethik	2	2126
2133	Popper	C3	52	5259	Der Wiener Kreis	2	2133
2134	Augustinus	C3	309	5022	Glaube und Wissen	2	2134
2137	Kant	C4	7	5001	Grundzüge	4	2137
2137	Kant	C4	7	4630	Die 3 Kritiken	4	2137

3. Schritt
Projektion auf
die Attribute
der SELECT-
Klausel

Name	Titel
Sokrates	Ethik
Sokrates	Mäeutik
Sokrates	Logik
Russel	Erkenntnistheorie
Russel	Wissenschaftstheorie
Russel	Bioethik
Popper	Der Wiener Kreis
Augustinus	Glaube und Wissen
Kant	Grundzüge
Kant	Die 3 Kritiken

Verbund/Join von Tabellen

- Neben der JOIN-Bedingung können in der WHERE-Klausel weitere "normale" Bedingungen auftauchen
- Beispiel: Welcher Professor (Name) liest die Vorlesung "Logik"?
 - ```
SELECT Name, Titel
FROM Professoren, Vorlesungen
WHERE PersNr = gelesenVon AND
Titel = 'Logik';
```

| Name                  | Titel                           |
|-----------------------|---------------------------------|
| <del>Sokrates</del>   | <del>Ethik</del>                |
| <del>Sokrates</del>   | <del>Mäeutik</del>              |
| Sokrates              | Logik                           |
| <del>Russel</del>     | <del>Erkenntnistheorie</del>    |
| <del>Russel</del>     | <del>Wissenschaftstheorie</del> |
| <del>Russel</del>     | <del>Bioethik</del>             |
| <del>Popper</del>     | <del>Der Wiener Kreis</del>     |
| <del>Augustinus</del> | <del>Glaube und Wissen</del>    |
| <del>Kant</del>       | <del>Grundzüge</del>            |
| <del>Kant</del>       | <del>Die 3 Kritiken</del>       |

# Qualifizierender Zugriff auf Attribute/Tabellen

- Vermeidung von Mehrdeutigkeit bei gleichen Attributnamen/Tabellennamen
- Möglichkeit 1: Tabellename als Präfix
  - SELECT Name, Titel  
FROM Studenten, hören, Vorlesungen  
WHERE Studenten.MatrNr = hören.MatrNr  
AND hören.VorlNr = Vorlesungen.VorlNr;

| Studenten     |              |          |
|---------------|--------------|----------|
| <u>MatrNr</u> | 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        |

| Hören         |               |
|---------------|---------------|
| <u>MatrNr</u> | <u>VorlNr</u> |
| 26120         | 5001          |
| 27550         | 5001          |
| 27550         | 4052          |
| 28106         | 5041          |
| 28106         | 5052          |
| 28106         | 5216          |
| 28106         | 5259          |
| 29120         | 5001          |
| 29120         | 5041          |
| 29120         | 5049          |
| 29555         | 5022          |
| 25403         | 5022          |

| Vorlesungen   |                      |     |            |
|---------------|----------------------|-----|------------|
| <u>VorlNr</u> | Titel                | SWS | gelesenVon |
| 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       |

# Qualifizierender Zugriff auf Attribute/Tabellen

- Möglichkeit 2: Einführung von Tupelvariablen/Aliasen
  - ```
SELECT s.Name, v.Titel  
FROM Studenten [AS] s, hören [AS] h, Vorlesungen [AS] v  
WHERE s.MatrNr = h.MatrNr  
AND h.VorlNr = v.VorlNr;
```
- Wird insbesondere benötigt, wenn die *gleiche* Tabelle in unterschiedlichen *Rollen* in der WHERE-Klausel vorkommt

Beispiel: Tabellen in unterschiedlichen Rollen

- Betrachte Tabelle Voraussetzen
 - Finde den indirekten Vorgänger 2. Stufe von Vorlesung 5216

Voraussetzen	
<u>Vorgänger</u>	<u>Nachfolger</u>
5001	5041
5001	5043
5001	5049
5041	5216
5043	5052
5041	5052
5052	5259

Voraussetzen v1		Voraussetzen v2	
<u>Vorgänger</u>	<u>Nachfolger</u>	<u>Vorgänger</u>	<u>Nachfolger</u>
5001	5041	5001	5041
5001	5041	5001	5043
⋮	⋮	5001	5049
5001	5041	5041	5216
⋮	⋮	⋮	⋮
5001	5043	5001	5041
5001	5043	5001	5043
5001	5043	5001	5049
⋮	⋮	⋮	⋮

- Wie sieht die entsprechende SQL-Abfrage aus?

```
SELECT v1.Vorgänger
FROM   Voraussetzen v1, Voraussetzen v2
WHERE  v2.Nachfolger = 5216
      AND v1.Nachfolger = v2.Vorgänger;
```

Joins im modernen ANSI-Stil

- Klassischer Stil:
 - SELECT Name, VorlNr
FROM Studenten, hören
WHERE Studenten.MatrNr = hören.MatrNr;
- Seit SQL-92 ("ANSI-Stil"):
 - Natürlicher Join (implizit über gleichnamige Attribute)
SELECT Name, VorlNr
FROM Studenten NATURAL JOIN hören;
 - Theta-Join mit expliziter Join-Bedingung: ON-Klausel
SELECT Name, Titel
FROM Professoren [INNER] JOIN Vorlesungen
ON Professoren.PersNr =
Vorlesungen.gelesenVon;

Andere Join-Arten

- **CROSS JOIN**

- Identisch mit Kartesischem Produkt, d.h. folgende Abfragen sind äquivalent:
- `SELECT *`
`FROM Professoren, Studenten;`
- `SELECT *`
`FROM Professoren CROSS JOIN Studenten;`

- **Äußere Joins**

- Auch Teiltupel mit fehlendem Partner werden ins Ergebnis übernommen
- `LEFT [OUTER] JOIN`
- `RIGHT [OUTER] JOIN`
- `FULL [OUTER] JOIN`

Definition des OUTER JOIN

- `SELECT *`
`FROM T1 LEFT [OUTER] JOIN T2 ON <Bedingung>`
- **Resultat**
 - Spalten: alle Spalten beider Tabellen
 - Zeilen: alle Zeilen der linken Tabelle T1
 - Verlängert um:
 - Verbindbare Zeilen der rechten Tabelle T2
 - Oder NULL-Werte in den Spalten von T2
- **Analog**
 - `T1 RIGHT [OUTER] JOIN T2 ON <Bedingung>`
 - `T1 FULL [OUTER] JOIN T2 ON <Bedingung>`

Beispiel: LEFT OUTER JOIN

- Liste alle Professoren mit ihren Vorlesungen auf (sofern vorhanden, ansonsten mit NULL auffüllen)
- ```
SELECT *
FROM Professoren LEFT JOIN Vorlesungen
ON PersNr = gelesenVon;
```

# Ergebnis des Left Joins

| 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    |

| Vorlesungen |                      |     |            |
|-------------|----------------------|-----|------------|
| VorlNr      | Titel                | SWS | gelesenVon |
| 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       |

| PersNr | Name       | Rang | Raum | VorlNr | Titel                | SWS  | gelesenVon |
|--------|------------|------|------|--------|----------------------|------|------------|
| 2125   | Sokrates   | C4   | 226  | 5041   | Ethik                | 4    | 2125       |
| 2125   | Sokrates   | C4   | 226  | 5049   | Mäeutik              | 2    | 2125       |
| 2125   | Sokrates   | C4   | 226  | 4052   | Logik                | 4    | 2125       |
| 2126   | Russel     | C4   | 232  | 5043   | Erkenntnistheorie    | 3    | 2126       |
| 2126   | Russel     | C4   | 232  | 5052   | Wissenschaftstheorie | 3    | 2126       |
| 2126   | Russel     | C4   | 232  | 5216   | Bioethik             | 2    | 2126       |
| 2127   | Kopernikus | C3   | 310  | NULL   | NULL                 | NULL | NULL       |
| 2133   | Popper     | C3   | 52   | 5259   | Der Wiener Kreis     | 2    | 2133       |
| 2134   | Augustinus | C3   | 309  | 5022   | Glaube und Wissen    | 2    | 2134       |
| 2136   | Curie      | C4   | 36   | NULL   | NULL                 | NULL | NULL       |
| 2137   | Kant       | C4   | 7    | 5001   | Grundzüge            | 4    | 2137       |
| 2137   | Kant       | C4   | 7    | 4630   | Die 3 Kritiken       | 4    | 2137       |

## Vergleiche mit "normalem" Join

**Verbund/Join von Tabellen**

- Kartesisches Produkt von Tabellen für sich genommen meist nicht besonders interessant
- Stattdessen Verknüpfung von zueinander passenden Tupeln der beteiligten Tabellen
  - Meist über Fremdschlüsselbeziehungen
- Verknüpfung wird Verbund oder Join genannt

Beispiel: Gib die Namen der Professoren zusammen mit den Titeln der von ihnen gelesenen Vorlesungen aus

```

SELECT Name, Titel
FROM Professoren, Vorlesungen
WHERE PersNr = gelesenVon;

```

**Ergebnis des Joins**

Tab. 1: 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    |

Tab. 2: Vorlesungen

| VorlNr | Titel                | SWS | gelesenVon |
|--------|----------------------|-----|------------|
| 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       |

Tab. 3: Ergebnis des Joins

| PersNr | Name       | Rang | Raum | VorlNr | Titel                | SWS  | gelesenVon |
|--------|------------|------|------|--------|----------------------|------|------------|
| 2125   | Sokrates   | C4   | 226  | 5041   | Ethik                | 4    | 2125       |
| 2125   | Sokrates   | C4   | 226  | 5049   | Mäeutik              | 2    | 2125       |
| 2125   | Sokrates   | C4   | 226  | 4052   | Logik                | 4    | 2125       |
| 2126   | Russel     | C4   | 232  | 5043   | Erkenntnistheorie    | 3    | 2126       |
| 2126   | Russel     | C4   | 232  | 5052   | Wissenschaftstheorie | 3    | 2126       |
| 2126   | Russel     | C4   | 232  | 5216   | Bioethik             | 2    | 2126       |
| 2127   | Kopernikus | C3   | 310  | NULL   | NULL                 | NULL | NULL       |
| 2133   | Popper     | C3   | 52   | 5259   | Der Wiener Kreis     | 2    | 2133       |
| 2134   | Augustinus | C3   | 309  | 5022   | Glaube und Wissen    | 2    | 2134       |
| 2136   | Curie      | C4   | 36   | NULL   | NULL                 | NULL | NULL       |
| 2137   | Kant       | C4   | 7    | 5001   | Grundzüge            | 4    | 2137       |
| 2137   | Kant       | C4   | 7    | 4630   | Die 3 Kritiken       | 4    | 2137       |

## SQL kennt zwei Klassen von Funktionen

- normale Funktionen
  - werden auf *einzelne* Argumente angewendet
  - Typumwandlung, binäre Operatoren, String-Funktionen, Datum-Funktionen, ...
- Aggregatfunktionen
  - werden auf eine komplette Spalte oder eine Teilmenge einer Spalte angewandt
  - Maximum, Summe, Mittelwert, Anzahl, Auswahl verschiedener Werte (distinct), ...

# Typumwandlung

- Kompatible Typen lassen sich mithilfe des CAST-Operators umwandeln
  - CAST (Note AS FLOAT)
- Bemerkungen
  - Gewöhnungsbedürftige Syntax: "AS" statt ","
  - Die meisten DBS führen auch implizite Casts durch
  - Beispiel
  - Postgres castet '...' Konstanten nach Bedarf (auch in numerische oder Datumstypen)  

```
INSERT INTO Professoren(PersNr, Name, Rang, Raum)
VALUES ('2125', 'Sokrates', 'C4', '226');
```
  - Empfehlung: keine optimistischen Annahmen machen!

# Probleme bei CAST

- CAST kann mehrdeutig sein
  - `CAST('01.02.02' AS DATE)`
- Ergebnis abhängig vom eingestellten Datumsformat (Parameter des Servers oder der Client-Session)
  - 01. Februar 2002
  - 02. Januar 2002
  - 02. Februar 2001
  - Fehler, weil Server Datumsangabe in einem anderen Format erwartet
- Lösung:
  - Formatierte Umwandlung mit `to_date()`-Funktion
  - Analog: `to_char()`, `to_number()`, `to_time(stamp)()`

# Datumsformatierung

- String → Date
  - to\_date('01.02.02', 'DD.MM.YY')
- Date → String
  - to\_char(Geburtsdatum, 'DD.MM.YYYY')

| Formatkennzeichen | Beschreibung                                                                       |
|-------------------|------------------------------------------------------------------------------------|
| YYYY, YY          | Jahr vierstellig, zweistellig                                                      |
| MM<br>Month, Mon  | Monat (01-12)<br>Monat als Text ("Januar", "Jan")                                  |
| DD, DDD<br>D      | Tag des Monats (01-31), Tag des Jahres (001-366)<br>Tag der Woche (1-7, Sonntag=1) |
| HH24, HH am       | Stunde (00-23), (01-12) mit am/pm                                                  |
| MI, SS            | Minute (00-59), Sekunde (00-59)                                                    |

# Zahlenformatierung

- String → Zahl
  - `to_number('11-', '99S')`
- Zahl → String
  - `to_char(Note, '0.9')`

| Formatkennzeichen | Beschreibung                                                              |
|-------------------|---------------------------------------------------------------------------|
| 9                 | Ziffer ohne führende Nullen                                               |
| 0                 | Ziffer mit führender Null                                                 |
| S                 | Minuszeichen bei negativen Zahlen                                         |
| PL                | Minus- oder Pluszeichen                                                   |
| . ,               | Dezimalpunkt und Tausendergruppe                                          |
| D G               | Dezimalpunkt und Tausendergruppe unter Berücksichtigung von <i>locale</i> |

# String-Funktionen

| Funktion                                                                        | Beschreibung                                                   |
|---------------------------------------------------------------------------------|----------------------------------------------------------------|
| <code>str1    str2</code>                                                       | String-Konkatenation                                           |
| <code>lower(str), upper(str)</code>                                             | Konversion in Klein/Großbuchstaben                             |
| <code>substr(str, pos, len)</code><br><code>substr(str FROM pos FOR len)</code> | Extraktion Teilstring (pos0 = 1)<br>SQL2-Syntax                |
| <code>trim(str [, chars])</code><br><code>trim([chars] FROM str)</code>         | vorne und hinten abschneiden<br>SQL2-Syntax                    |
| <code>translate(str, from, to)</code>                                           | Zeichen austauschen mit from und to<br>als Übersetzungstabelle |

- **Beispiel:**

```
– SELECT upper(Name) || ' kostet ' ||
 trim(to_char(preis, '99D99')) || 'EUR.'
 AS PREISLISTE
 FROM Produkt;
```



# Mathematische und Datum-Funktionen

- Mathematische Funktionen

| Funktion                                                    | Beschreibung                              |
|-------------------------------------------------------------|-------------------------------------------|
| <code>+</code> <code>-</code> <code>*</code> <code>/</code> | arithmetische Operatoren                  |
| <code>abs(x)</code>                                         | Absolutwert                               |
| <code>trunc(x[, n])</code>                                  | Abschneiden auf <i>n</i> Nachkommastellen |
| <code>round(x[, n])</code>                                  | Runden auf <i>n</i> Nachkommastellen      |

- Datum-Funktionen

| Funktion                                                    | Beschreibung                                          |
|-------------------------------------------------------------|-------------------------------------------------------|
| <code>current_date</code><br><code>current_timestamp</code> | Aktuelles Datum bzw. Uhrzeit<br>SQL3: keine Klammern! |
| <code>age([ts1, ], ts2)</code>                              | Intervall <i>ts1-ts2</i>                              |
| <code>extract(feld FROM ts)</code>                          | Feldextraktion (z.B. year)                            |

# Aggregatfunktionen

- Aggregatfunktionen führen Operationen auf **Tupelmengen** aus und komprimieren diese zu einem Wert

| Aggregatfunktion | Beschreibung                                                                |
|------------------|-----------------------------------------------------------------------------|
| avg              | Durchschnitt                                                                |
| sum              | Summe                                                                       |
| min              | Minimum                                                                     |
| max              | Maximum                                                                     |
| count            | Anzahl der (nicht notwendigerweise verschiedenen) Attributwerte $\neq$ NULL |
| count(*)         | Anzahl der Zeilen                                                           |

- Wenn eine SELECT-Abfrage eine Aggregatfunktion erhält, dann wird max. ein Resultattupel erzeugt
  - Ausnahme: GROUP BY-Abfragen (später)

# Beispiele für Aggregatfunktionen

Beispiel-Datenbank

| Tabellennamen | Spaltennamen  | Datentypen |
|---------------|---------------|------------|
| Studenten     | MatrNr        | INT        |
| Semester      | Semester      | INT        |
| Vorlesungen   | VorNr         | INT        |
| Professoren   | PersNr        | INT        |
| gelesenVon    | VorNr         | INT        |
| gelesenVon    | PersNr        | INT        |
| gelesenVon    | Semester      | INT        |
| gelesenVon    | Wochenstunden | INT        |

- Durchschnittliche Semesterzahl aller Studierenden
  - SELECT **avg**(Semester)  
FROM Studenten;
- Anzahl aller Studenten sowie minimale und maximale Semesterzahl
  - SELECT **count**(\*), **min**(Semester), **max**(Semester)  
FROM Studenten;
- Gesamte Semesterwochenstunden der Vorlesungen von Sokrates
  - SELECT **sum**(SWS)  
FROM Vorlesungen JOIN Professoren  
ON gelesenVon = PersNr  
WHERE Name = 'Sokrates';

# GROUP BY-Klausel

- SELECT ... FROM ... WHERE ...  
GROUP BY <Attributliste>
- Gruppiert Tupel, die gleiche Werte in den Gruppierungsattributen haben, und reduziert diese zu einem Tupel
- Beispiel: Ermittle Anzahl der Semesterwochenstunden, die von einzelnen Professoren erbracht werden

```
SELECT gelesenVon, sum(SWS)
FROM Vorlesungen
GROUP BY gelesenVon;
```

| Vorlesungen  |                      |     |            |   |   |
|--------------|----------------------|-----|------------|---|---|
| <u>VorNr</u> | Titel                | SWS | gelesenVon |   |   |
| 5041         | Ethik                | 4   | 2125       | } | → |
| 5049         | Mäeutik              | 2   | 2125       |   |   |
| 4052         | Logik                | 4   | 2125       |   |   |
| 5043         | Erkenntnistheorie    | 3   | 2126       | } | → |
| 5052         | Wissenschaftstheorie | 3   | 2126       |   |   |
| 5216         | Bioethik             | 2   | 2126       |   |   |
| 5259         | Der Wiener Kreis     | 2   | 2133       | } | → |
| 5022         | Glaube und Wissen    | 2   | 2134       |   |   |
| 5001         | Grundzüge            | 4   | 2137       | } | → |
| 4630         | Die 3 Kritiken       | 4   | 2137       |   |   |

| gelesenVon | sum(SWS) |
|------------|----------|
| 2125       | 10       |
| 2126       | 8        |
| 2133       | 2        |
| 2134       | 2        |
| 2137       | 8        |

# Wirkungsweise von GROUP BY

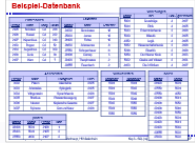
- Wenn man "geschachtelte Relationen" zulässt, kann man die Wirkungsweise von GROUP BY wie folgt erklären:
  - Nach der Auswertung der FROM- und WHERE-Klauseln wird das bisherige Zwischenergebnis nach den in der GROUP BY-Klausel angegebenen Attributen in "geschachtelte" Tupel gruppiert
  - hier: GROUP BY A, B

| A | B | C  | D   |
|---|---|----|-----|
| a | b | c1 | c2  |
| a | b | c3 | c4  |
| x | y | c3 | c4  |
| x | y | c5 | c6  |
| x | y | c7 | c8  |
| x | w | c9 | c10 |

| A | B | C  | D   |
|---|---|----|-----|
| a | b | c1 | c2  |
|   |   | c3 | c4  |
| x | y | c3 | c4  |
|   |   | c5 | c6  |
|   |   | c7 | c8  |
| x | w | c9 | c10 |

# SELECT-Attribute bei GROUP BY-Klauseln

- Beispiel vorher:
  - Aus 6 Tupel sind 3 Tupel mit mengenwertigen C,D-Attributen geworden
- Da im Relationenmodell keine mengenwertigen Attribute erlaubt sind, sind Beschränkungen bzgl. der SELECT-Attribute erforderlich, damit das Endergebnis wieder eine "flache" Relation ist.
  - In der SELECT-Klausel können bei GROUP BY-Abfragen nur Gruppierungs-Attribute und aggregierte Attribute stehen
  - Aggregat-Funktionen werden jeweils auf den Teilmengen von Werten pro Gruppe ausgeführt

A small thumbnail image in the top right corner showing a database table with multiple columns and rows of data, likely representing the 'Beispiel-Datenbank' mentioned in the caption.

- `SELECT ... FROM ... WHERE ...`  
`GROUP BY <Attributliste>`  
`HAVING <Gruppenbedingung>`
- Lässt nur Gruppen zu, die die Gruppenbedingung erfüllen
- Beispiel:
  - Betrachte nur die C4-Professoren, die überwiegend lange Vorlesungen halten (Durchschnitt  $\geq 3$ )
  - `SELECT gelesenVon, Name, sum(SWS)`  
`FROM Vorlesungen, Professoren`  
`WHERE gelesenVon = PersNr`  
`AND Rang = 'C4'`  
`GROUP BY gelesenVon, Name`  
`HAVING avg(SWS)  $\geq$  3;`

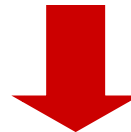
# Ausführung einer GROUP BY/HAVING-Abfrage

## Kartesisches Produkt bilden



| Vorlesung x Professoren |                  |     |             |        |          |      |      |
|-------------------------|------------------|-----|-------------|--------|----------|------|------|
| VorlNr                  | Titel            | SWS | gelesen Von | PersNr | Name     | Rang | Raum |
| 5001                    | Grundzüge        | 4   | 2137        | 2125   | Sokrates | C4   | 226  |
| 5041                    | Ethik            | 4   | 2125        | 2125   | Sokrates | C4   | 226  |
| ...                     | ...              | ... | ...         | ...    | ...      | ...  | ...  |
| 5259                    | Der Wiener Kreis | 2   | 2133        | 2133   | Popper   | C3   | 52   |
| ...                     | ...              | ... | ...         | ...    | ...      | ...  | ...  |
| 4630                    | Die 3 Kritiken   | 4   | 2137        | 2137   | Kant     | C4   | 7    |

**WHERE**-Bedingung: gelesenVon = PersNr AND Rang = 'C4'





# Ausführung einer GROUP BY/HAVING-Abfrage

| VorlNr | Titel                | SWS | gelesen<br>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    |

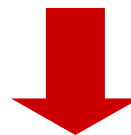
**Gruppierung ( GROUP BY gelesenVon, Name )**



# Ausführung einer GROUP BY/HAVING-Abfrage

| VorlNr | Titel                | SWS | gelesen Von | 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   | Wissenschaftstheorie | 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 (  $\text{avg}(\text{SWS}) \geq 3$  )**



# Ausführung einer GROUP BY/HAVING-Abfrage

| VorlNr | Titel          | SWS | gelesen Von | 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( SWS) ) und Projektion**



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

# GROUP BY/HAVING

- GROUP BY/HAVING gehört logisch zur SELECT-Klausel (nicht zur WHERE-Klausel)!
  - SQL erzeugt pro Gruppe ein Ergebnistupel
  - Deshalb sind als SELECT-Attribute dann nur Gruppierungsattribute oder aggregierte Attribute erlaubt
  - Nur so kann SQL sicherstellen, dass es für jede Spalte der Ergebnisrelation nur einen Wert pro Gruppe gibt
- Unterschied WHERE/HAVING
  - WHERE filtert auf Tupelebene
  - HAVING filtert auf Gruppenebene
    - nur Gruppierungsattribute oder aggregierte Attribute erlaubt!

# NULL-Werte in SQL

- SQL unterstützt einen Wert NULL für alle Wertebereiche
  - Bedeutung: "Wert nicht bekannt" (wird vielleicht später nachgeliefert)
  - Unterschiedlich zu numerischem Wert 0 oder der Zeichenkette der Länge 0 ("")
  - Es gilt auch:  $\text{NULL} \neq \text{NULL}$

# Behandlung von NULL-Werten

- Spezielle Selektionsbedingung
  - WHERE <Attribut> IS [NOT] NULL
  - Nicht: WHERE <Attribut> = NULL
- In arithmetischen Ausdrücken
  - NULL-Werte werden propagiert, wenn mindestens einer der Operanden NULL ist
- Beispiele
  - SELECT MatrNr, Semester + 1 FROM Studenten
    - NULL + 1 → NULL

# Beispiel: NULL in arithmetischen Ausdrücken

- SELECT MatrNr, Semester + 1  
FROM Studenten

| Studenten     |              |          |
|---------------|--------------|----------|
| <u>MatrNr</u> | Name         | Semester |
| 24002         | Xenokrates   | NULL     |
| 25403         | Jonas        | 12       |
| 26120         | Fichte       | 10       |
| 26830         | Aristoxenos  | 8        |
| 27550         | Schopenhauer | NULL     |
| 28106         | Carnap       | 3        |
| 29120         | Theophrastos | 2        |
| 29555         | Feuerbach    | 2        |



| <u>MatrNr</u> | Semester + 1 |
|---------------|--------------|
| 24002         | NULL         |
| 25403         | 13           |
| 26120         | 11           |
| 26830         | 9            |
| 27550         | NULL         |
| 28106         | 4            |
| 29120         | 3            |
| 29555         | 3            |

Gilt auch bei Multiplikation mit der Zahl 0!

NULL \* 0 → NULL

# NULL-Werte in Vergleichen (WHERE-Klausel)

- Vergleichsoperatoren in der WHERE-Klausel liefern den Wert **unknown** zurück, wenn mindestens eins der Argumente NULL ist
- In der WHERE-Klausel werden nur Tupel weitergereicht, für die die WHERE Bedingung **true** zurückliefert
  - SELECT \* FROM Studenten WHERE Semester >= 10
  - Tupel mit **NULL** >= 10 liefern **unknown** und werden deshalb zurückgewiesen

| Studenten     |              |          |
|---------------|--------------|----------|
| <u>MatrNr</u> | Name         | Semester |
| 24002         | Xenokrates   | NULL     |
| 25403         | Jonas        | 12       |
| 26120         | Fichte       | 10       |
| 26830         | Aristoxenos  | 8        |
| 27550         | Schopenhauer | NULL     |
| 28106         | Carnap       | 3        |
| 29120         | Theophrastos | 2        |
| 29555         | Feuerbach    | 2        |



| <u>MatrNr</u> | Name   | Semester |
|---------------|--------|----------|
| 25403         | Jonas  | 12       |
| 26120         | Fichte | 10       |



# Dreiwertige Logik in SQL

- Problem: wie wird **unknown** in komplexen Booleschen Ausdrücken behandelt?
  - WHERE NOT((Semester>=10) OR (MatrNr < 26500))
- Dreiwertige Logik mit Wahrheitswerten **true**, **false**, **unknown**:

| AND     | true    | unknown | false |
|---------|---------|---------|-------|
| true    | true    | unknown | false |
| unknown | unknown | unknown | false |
| false   | false   | false   | false |

| NOT     |         |
|---------|---------|
| true    | false   |
| unknown | unknown |
| false   | true    |

| OR      | true | unknown | false   |
|---------|------|---------|---------|
| true    | true | true    | true    |
| unknown | true | unknown | unknown |
| false   | true | unknown | false   |

# Beispiel: dreiwertige Logik in WHERE-Klauseln

- SELECT \*  
FROM Studenten  
WHERE NOT((Semester >= 10) OR (MatrNr < 26500));

z.B.: Schopenhauer

unknown

OR

false

NOT

unknown

unknown → Tupel wird zurückgewiesen

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

| Studenten |              |          |
|-----------|--------------|----------|
| MatrNr    | Name         | Semester |
| 26830     | Aristoxenos  | 8        |
| 28106     | Carnap       | 3        |
| 29120     | Theophrastos | 2        |
| 29555     | Feuerbach    | 2        |

# NULL-Werte und Aggregatfunktionen

- NULL-Werte werden bei der Auswertung von Aggregatfunktionen (avg, min, max, sum, count) ignoriert
- `SELECT sum( Semester )  
FROM Studenten; → 37`
- `SELECT avg( Semester )  
FROM Studenten; → 6,166 (= 37/6)`
- Ausnahme: `count(*)` !

| Studenten     |          |
|---------------|----------|
| <u>MatrNr</u> | Semester |
| 24002         | NULL     |
| 25403         | 12       |
| 26120         | 10       |
| 26830         | 8        |
| 27550         | NULL     |
| 28106         | 3        |
| 29120         | 2        |
| 29555         | 2        |

# NULL-Werte: count vs count(\*)

- Im Gegensatz zu allen anderen Aggregatfunktionen werden NULL-Werte bei count(\*) nicht ignoriert
  - Grund: count(\*) zählt Zeilen, nicht einzelne Werte
  - Aufpassen by WHERE-Klauseln: diese werden vor dem count(\*) ausgewertet!

- Beispiele:

- SELECT count(Semester)  
FROM Studenten; **Liefert 6**
- SELECT count(\*)  
FROM Studenten; **Liefert 8**
- SELECT count(\*)  
FROM Studenten  
WHERE Semester < 10  
OR Semester >= 10; **Liefert 6**

| Studenten     |          |
|---------------|----------|
| <u>MatrNr</u> | Semester |
| 24002         | NULL     |
| 25403         | 12       |
| 26120         | 10       |
| 26830         | 8        |
| 27550         | NULL     |
| 28106         | 3        |
| 29120         | 2        |
| 29555         | 2        |


# Noch ein warnendes Beispiel zum "Count(\*)-Bug"

- ```
SELECT    count(*),  
          sum(Semester),  
          sum(Semester)/count(*),  
          avg(Semester)  
FROM Studenten
```
- Liefert:

count(*)	sum(Semester)	sum(Semester)/ count(*)	avg(Semester)
8	37	4,625	6,166

NULL-Werte und Gruppierung

- Bei Gruppierung nach NULL-wertigem Attribut:
 - NULL-Wert wird wie eigener Wert behandelt (bildet ggf. eigene Gruppe)
 - `SELECT Semester
FROM Studenten
GROUP BY Semester;`



<u>MatrNr</u>	Semester
24002	NULL
25403	12
26120	10
26830	8
27550	NULL
28106	3
29120	2
29555	2

NULL
12
10
8
3
2

Anatomie von SELECT-Ausdrücken

Klausel	Reihenfolge	Semantik (was passiert?)
SELECT (DISTINCT)	5	Projektion: Übernehme nur die genannten Spalten, streiche die restlichen; Wende Funktionen (sum, avg, ...) an; Streiche Duplikate aus Ergebnis-Tabelle
FROM	1	Bilde das kartesische Produkt (... , ...) oder den Join (... JOIN ... ON ...) über die angegebenen Tabellen
WHERE	2	Streiche alle Tupel des Joins/kart. Produkts, die die WHERE-Bedingung nicht erfüllen
GROUP BY	3	Gruppieren Tupel
HAVING	4	Streiche alle Tupel-Gruppen, die die HAVING-Bedingung nicht erfüllen
ORDER BY	6	Sortiere das Ergebnis

Unterabfragen

- Ergebnis einer Abfrage kann als Unterabfrage (Subquery) anstelle eines Wertes oder einer Tabelle in einer übergeordneten Abfrage auftauchen
- Mögliche Stellen innerhalb der übergeordneten Abfrage

Ort der Subquery	Was liefert die Subquery?
SELECT	Einzelnen Wert/Tupel
FROM	Tabelle (Menge von Werten/Tupeln)
WHERE	Einzelnen Wert/Tupel (für Vergleiche) Tabelle in Zusammenhang mit EXISTS, IN, ALL, ANY-Quantoren

Unterabfrage in SELECT-Klausel

Beispiel-Datenbank

Professoren				Vorlesungen				Benutzer			
PersNr	Name	gelesenVon	Sum(SWS)	VorNr	Wochentag	Stunde	Wochentag	Stunde	BenNr	Name	gelesenVon
1	Prof. Dr. Weidenhaupt	1	2	1	Mo	10	1	10	1	Ben. Dr. Weidenhaupt	1
2	Prof. Dr. Müller	2	3	2	Di	11	2	11	2	Ben. Dr. Müller	2
3	Prof. Dr. Schmidt	3	1	3	Do	12	3	12	3	Ben. Dr. Schmidt	3
4	Prof. Dr. Weber	4	2	4	Fr	13	4	13	4	Ben. Dr. Weber	4
5	Prof. Dr. Hoffmann	5	3	5	Sa	14	5	14	5	Ben. Dr. Hoffmann	5

- ```
SELECT PersNr, Name,
 (SELECT sum(SWS) as Lehrbelastung
 FROM Vorlesungen
 WHERE gelesenVon = PersNr)

FROM Professoren ;
```
- **Bemerkungen**
  - Für jedes Ergebnistupel der Oberabfrage wird die Unterabfrage ausgeführt
  - Unterabfrage muss genau einen Wert liefern (keine Tabelle mit mehreren Tupeln)
  - In diesem Beispiel ist Unterabfrage *korreliert*, d.h. sie greift auf Attribute der umschließenden Abfrage zu
  - Äquivalentes Ergebnis hätte man auch mit JOIN und GROUP BY erreichen können (→ Übung)

# Unterabfrage in FROM-Klausel

Beispiel-Datenbank

| Tabellennamen | Spaltennamen | Datentypen | Primärschlüssel | Fremdschlüssel |
|---------------|--------------|------------|-----------------|----------------|
| Studenten     | MatrNr       | INT        | Ja              |                |
| Studenten     | Name         | VARCHAR    |                 |                |
| Studenten     | VorlAnzahl   | INT        |                 |                |
| hören         | MatrNr       | INT        |                 | Ja             |
| hören         | h_MatrNr     | INT        |                 | Ja             |
| hören         | h_Name       | VARCHAR    |                 | Ja             |
| hören         | h_VorlAnzahl | INT        |                 | Ja             |

- ```
SELECT tmp.MatrNr, tmp.Name, tmp.VorlAnzahl
FROM (SELECT s.MatrNr, s.Name,
            count(*) as VorlAnzahl
      FROM Studenten s, hören h
     WHERE s.MatrNr = h.MatrNr
     GROUP BY s.MatrNr, s.Name) tmp
WHERE tmp.VorlAnzahl > 2;
```
- Bemerkungen:
 - tmp ist Bezeichner für die von der Subquery zurückgegebene Tabelle
 - die Klammern (...) und der Name tmp um die SELECT-Anweisung wirken als Tabellenkonstruktor einer neuen Tabelle tmp

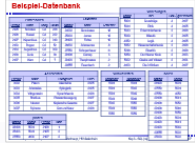
Unterabfrage in WHERE-Klausel: EXISTS

Beispiel-Datenbank

Tab.	Attribut	Wert
Professoren	PersNr	1
	Name	Prof. Dr. Weidenhaupt
	Abt.	Mathematik
	Lehrstuhl	Mathematik
Vorlesungen	VorNr	1
	Thema	Mathematik
	Abt.	Mathematik
	Lehrstuhl	Mathematik

- Form 1: EXISTS-Prädikat
 - SELECT <Attributliste>
FROM R_1, R_2, \dots, R_n
WHERE [NOT] EXISTS (SELECT ... FROM ... WHERE)
 - Subquery-Ausdruck EXISTS liefert nur TRUE (Subquery liefert wenigstens ein Tupel) oder FALSE (Subquery liefert leeres Resultat)
- Beispiel: Finde die Namen von Professoren, die keine Vorlesung halten
 - SELECT p.Name
FROM Professoren p
WHERE NOT EXISTS (SELECT *
FROM Vorlesungen v
WHERE v.gelesenVon=p.PersNr);

Unterabfrage in WHERE-Klausel: IN

A small thumbnail image in the top right corner showing a database table with multiple columns and rows of data, likely representing the 'Beispiel-Datenbank' mentioned in the caption.

- Form 2: IN-Prädikat

- SELECT <Attributliste>

- FROM R_1, R_2, \dots, R_n

- WHERE $R_{i1}.A_1 [, \dots, R_{in}.A_n]$ [NOT] IN
(SELECT ... FROM ... WHERE)

- Prüft, ob Wert (oder Tupel) in der Resultatmenge der Unterabfrage enthalten ist

- Die Liste der Attribute in der WHERE-Klausel muss zu den SELECT-Attributen der Unterabfrage passen (Anzahl und Wertebereiche)

- Beispiel

- SELECT Name

- FROM Professoren

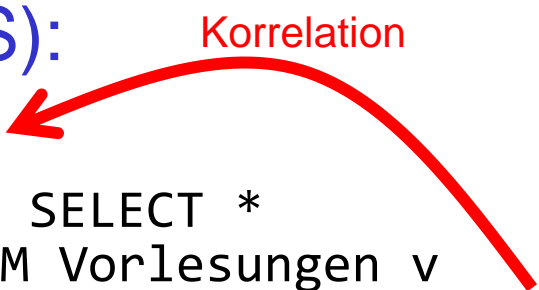
- WHERE PersNr NOT IN (SELECT gelesenVon
FROM Vorlesungen);

Korrelation

- Ober- und Unterabfrage miteinander korreliert sein
 - Hier über das Attribut PersNr

- Form 1 (EXISTS):

```
SELECT p.Name
FROM Professoren p
WHERE NOT EXISTS ( SELECT *
                   FROM Vorlesungen v
                   WHERE v.gelesenVon = p.PersNr );
```

A red curved arrow labeled "Korrelation" points from the text "Korrelation" to the expression "p.PersNr" in the WHERE clause of the subquery, indicating that the subquery's execution is dependent on the current row of the outer query.

– Unterabfrage wird wegen Korrelation für jedes Professorentupel ausgewertet

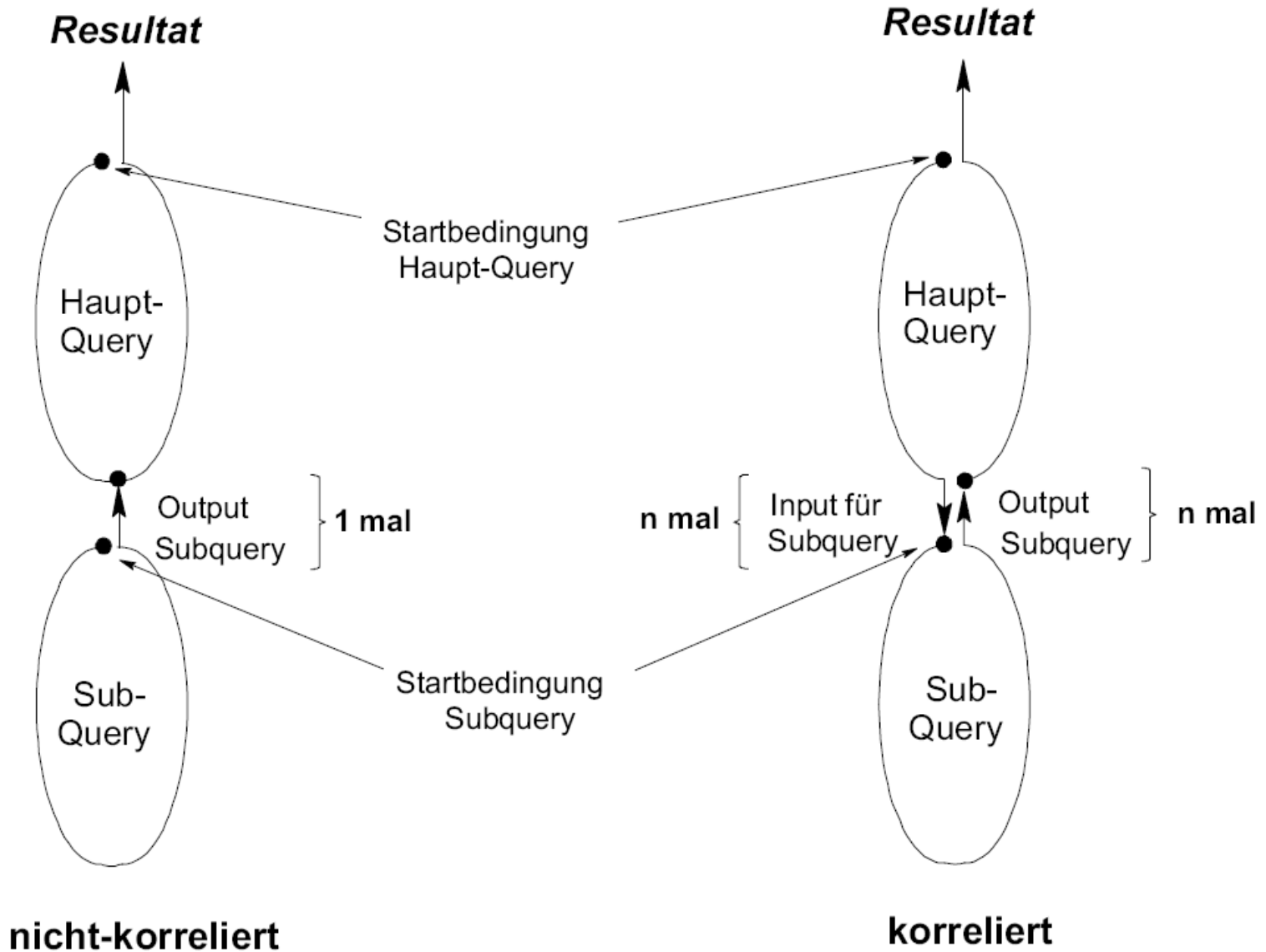
- Form 2 (IN):

```
SELECT Name
FROM Professoren
WHERE PersNr NOT IN ( SELECT gelesenVon FROM Vorlesungen )
```

A red arrow labeled "Unkorrelierte Unterabfrage" points from the text "Unkorrelierte Unterabfrage" to the subquery "(SELECT gelesenVon FROM Vorlesungen)", indicating that the subquery is executed once and its results are used for the entire outer query.

– Meist effizienter, da Unterabfrage nur einmal ausgewertet wird

Korrelierte vs. nicht-korrelierte Subqueries



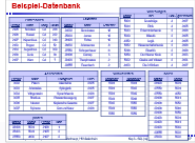
Unterabfrage in WHERE-Klausel: Vergleiche

- Form 3:

```
SELECT <Attributliste>
FROM R1, R2, ..., Rn
WHERE Ri1.A1 [, ..., Rin.An]
<OP> [ANY|ALL] ( SELECT ... FROM ... WHERE )
```

 - <OP>: <, <=, =, >=, >, <>
 - ALL:
 - vergleicht Attributwert aus Oberabfrage mit Werten der Unterabfrage
 - Wahr, wenn für **alle** Elemente der Unterabfrage die Bedingung wahr ist
 - ANY:
 - Wahr, wenn **für mindestens** ein Element der Unterabfrage die Bedingung wahr ist
 - Ohne ALL bzw. ANY muss die Subquery so formuliert sein, dass sie genau einen Wert bzw. Tupel zurückliefert (keine Menge)!

Beispiel: ALL-Quantor



- Finde Namen der Studenten mit den meisten Semestern
 - Benutzen Sie den ALL-Quantor!
 - ```
SELECT Name
FROM Studenten
WHERE Semester >= ALL (SELECT Semester
 FROM Studenten);
```
- Finde Namen der Studenten, die überdurchschnittlich viele Semester auf dem Buckel haben
  - ```
SELECT Name
FROM Studenten
WHERE Semester > ( SELECT avg( Semester )
                  FROM Studenten );
```


Mathematische Interpretation

SQL-Operator	Mathematischer Operator	Bedeutung
IN	\in	Wert der Oberabfrage ist in Resultatmenge der Unterabfrage enthalten
ANY	\exists	Wert der Oberabfrage erfüllt Bedingung für <u>wenigstens ein</u> Ergebnistupel der Unterabfrage
ALL	\forall	Wert der Oberabfrage erfüllt Bedingung für <u>alle</u> Ergebnistupel der Unterabfrage
EXISTS	$\neq \emptyset$	Ergebnis der Unterabfrage ist nicht leer

Äquivalente Formulierungen

- ANY/ALL/EXISTS erfordern Vergleiche mit einer oft großen Menge von Resultattupeln der Unterabfrage
 - Lässt sich häufig durch geschickte Umformulierung vermeiden (unter Verwendung von Aggregatfunktionen)
- Beispiel: Finde Namen der Studenten mit meisten Semestern

- Variante mit ALL

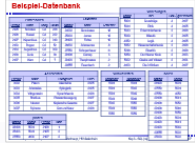
```
SELECT Name
FROM Studenten
WHERE Semester >= ALL ( SELECT Semester FROM Studenten );
```

- Variante mit Aggregatfunktion

```
SELECT Name
FROM Studenten
WHERE Semester = ( SELECT max(Semester) FROM Studenten );
```

Äquivalente Formulierungen

ANY/ALL/EXISTS	Alternative Formulierung
WHERE X = ANY (...)	WHERE X IN (...)
WHERE X < ANY (...)	WHERE X < (SELECT max(A))
WHERE X > ANY (...)	WHERE X > (SELECT min(A))
WHERE X <= ALL (...)	WHERE X = (SELECT min(A))
WHERE X >= ALL (...)	WHERE X = (SELECT max(A))
WHERE X <> ALL (...)	WHERE X NOT IN (...)
WHERE EXISTS (...)	WHERE 0 < (SELECT count(*) ...)
WHERE NOT EXISTS (...)	WHERE 0 = (SELECT count(*) ...)

A small thumbnail image in the top right corner showing a database table with multiple columns and rows of data, likely representing a 'Beispiel-Datenbank' (Example Database).

- Vereinigung:
 - <Tabellenausdruck 1> UNION [ALL] <Tabellenausdruck 2>
- Durchschnitt:
 - <Tabellenausdruck 1> INTERSECT [ALL] <Tabellenausdruck 2>
- Differenz:
 - <Tabellenausdruck 1> EXCEPT [ALL] <Tabellenausdruck 2>
 - In Oracle: MINUS [ALL]
- Automatische Duplikateliminierung
 - Kann durch das Schlüsselwort ALL unterdrückt werden
- Beispiele
 - (SELECT Name FROM Assistenten)
UNION
(SELECT Name FROM Professoren);
 - SELECT t.Name
FROM (TABLE Assistenten UNION TABLE Professoren) t;

Mengenoperationen

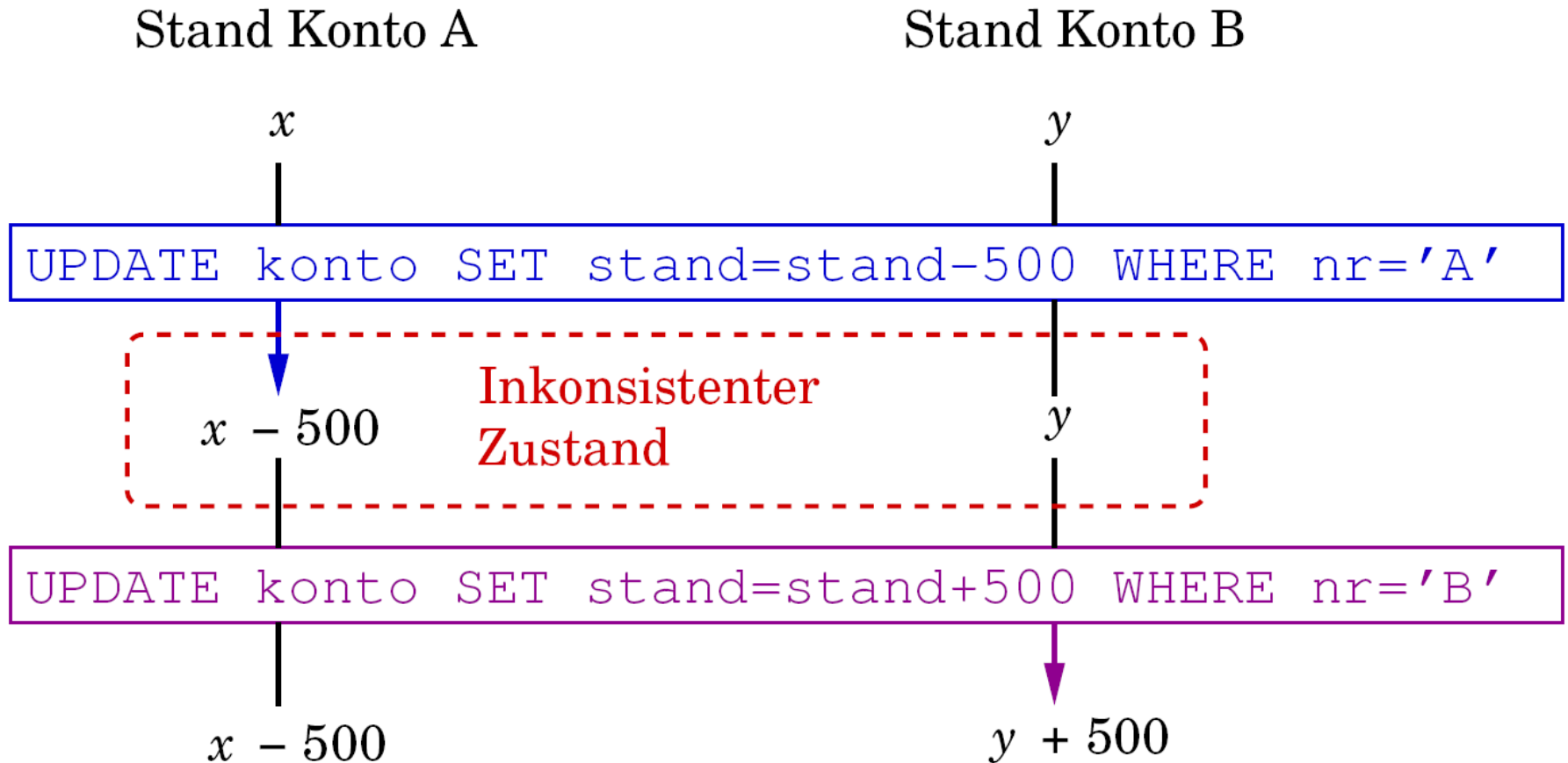
- Auf Vereinigungsverträglichkeit achten!
 - Spalten der beiden Tabellen müssen in Anzahl und Datentypen miteinander verträglich sein
 - Automatische Konvertierung von kompatiblen Datentypen in den "größeren" Datentypen
 - z.B Zeichenketten unterschiedlicher Länge
 - Evt. Alias für Spaltennamen einführen
 - Nicht zwingend erforderlich, aber für Ausgabe sinnvoll
- Beispiel:
 - Gegebene Tabellen:
 - Authors(au_lname: varchar(20), au_fname: varchar(30))
 - Employees(emp_lname: varchar(50), emp_fname: varchar(50))
 - ```
(SELECT au_lname AS LastName,
 au_fname AS FirstName FROM Authors)
EXCEPT
(SELECT emp_lname AS LastName,
 emp_fname AS FirstName FROM Employees);
```

# SQL Data Control Language

- Kommandos für Datensicherheit/Datenschutz
- Sicherheit vor *fehlerhaften* Zugriffen
  - Stichwort "Transaktionen"
  - SQL-Kommandos: BEGIN, COMMIT, ROLLBACK
- Schutz für *unberechtigten* Zugriffen
  - Stichwort "Benutzerrechte"
  - SQL-Kommandos: GRANT, REVOKE
- Vor allem wichtig im Mehrbenutzerbetrieb, d.h. bei Client-Server-Datenbanken

# Transaktionen

- Dienen dazu, mehrere SQL-Kommandos zu einer Einheit zusammenzufassen



## Transaktionen erfüllen das ACID-Prinzip

- Atomicity
  - Transaktion ist Einheit: Alles oder Nichts
- Consistency
  - Transaktion überführt konsistenten Zustand in einen konsistenten Zustand
  - Innerhalb Transaktion Inkonsistenz möglich
- Isolation
  - Änderungen in einer Transaktion sind bis zum Abschluss unsichtbar für andere Transaktionen
- Durability
  - Nach Abschluss der Transaktion bleiben Änderungen bestehen auch im Falle eines Systemabsturzes



# Start und Abschluss von Transaktionen

- SQL-Kommandos für Transaktionen

| Kommando                            | Bedeutung                                                      |
|-------------------------------------|----------------------------------------------------------------|
| BEGIN [WORK]<br>BEGIN [TRANSACTION] | Start einer Transaktion<br>Achtung: ggf. implizit (Oracle)     |
| COMMIT                              | Abschluss der Transaktion <i>mit</i> Übernahme der Änderungen  |
| ROLLBACK                            | Abschluss der Transaktion <i>ohne</i> Übernahme der Änderungen |

- Bemerkungen:

- In Oracle und SQL2 beginnt Transaktion *implizit* mit jedem "transaction-initiating" Kommando
- Die meisten anderen DBS (auch PostgreSQL) machen dagegen ein *auto-commit* nach jedem Statement, wenn nicht explizit eine längere Transaktion mit BEGIN gestartet wurde

# Benutzerrechte

- DBS hat eigene Benutzerverwaltung
- Anlage neuer User mit `CREATE USER ...`
- Ändern mit `ALTER USER ...`
- Kommandos sind nicht standardisiert
- Beispiel Passwort-Änderung:
  - Oracle: `ALTER USER usr IDENTIFIED BY 'pwd';`
  - PostgreSQL: `ALTER USER usr WITH PASSWORD 'pwd';`
- Auch Zuweisung Admin-Recht (DBA) systemspezifisch

# GRANT/REVOKE

- Der Anleger einer Tabelle ist ihr *Owner*
  - Sonst kann keiner auf die Tabelle zugreifen
- Ändern des Owner
  - `ALTER TABLE table_name OWNER TO new_owner;`
- Wenn auch andere User die Tabelle nutzen sollen, muss der Owner ihnen *Privileges* erteilen mit dem Kommando: `GRANT ... TO ...`
  - `GRANT SELECT ON Studenten TO PUBLIC;`
  - `GRANT UPDATE ON Studenten TO peter;`
- Entzug von Privileges mit: `REVOKE ... FROM ...`
  - `REVOKE UPDATE ON Studenten FROM peter;`

# Überblick Privilegien

| Privileg            | Berechtigung                                                                        |
|---------------------|-------------------------------------------------------------------------------------|
| SELECT              | Lesen                                                                               |
| INSERT              | Einfügen neuer Datensätze                                                           |
| UPDATE              | Ändern bestehender Datensätze                                                       |
| DELETE,<br>TRUNCATE | Löschen                                                                             |
| CREATE              | Erzeugen von Tabellen und anderen Objekten                                          |
| CONNECT             | Verbinden mit einer Datenbank                                                       |
|                     | Weitere Privilegien/Datenbankobjekte je nach DBS:<br>rule, references, trigger, ... |

- Vereinfachungen
  - ALL kann für alle Privilegien verwendet werden
  - PUBLIC kann für alle User verwendet werden

# Benutzergruppen

- Einfachere Rechteverwaltung mit *Groups*
- Anlegen Gruppe mit
  - CREATE GROUP grp;
- Privilegien dieser Gruppe zuweisen mit
  - GRANT ... TO GROUP grp;
- User in die Gruppe aufnehmen mit
  - ALTER GROUP grp ADD USER usr;
- User können aus Gruppe entfernt werden mit
  - ALTER GROUP grp DROP USER usr;

- SQL-Befehle können interaktiv über SQL-Interpreter eingegeben werden
  - Oracle: sqlplus, Postgres: psql
- Metakommandos
  - Befehle an den Interpreter
  - In psql durch Backslash gekennzeichnet, z.B. \d (describe), \i (import script), \set (set psql option)
  - Liste aller Metakommandos: man psql
- SQL-Kommandos
  - werden an den Datenbankserver weitergereicht
- Wie greift man aus einem Programm auf DB zu?
  - nächstes Kapitel

# Beispiel-Datenbank

| Professoren   |            |      |      |
|---------------|------------|------|------|
| <u>PersNr</u> | 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     |              |          |
|---------------|--------------|----------|
| <u>MatrNr</u> | 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   |                      |     |            |
|---------------|----------------------|-----|------------|
| <u>VorINr</u> | Titel                | SWS | gelesenVon |
| 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       |

| Assistenten    |              |                    |      |
|----------------|--------------|--------------------|------|
| <u>PersINr</u> | 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 |

| voraussetzen     |                   |
|------------------|-------------------|
| <u>Vorgänger</u> | <u>Nachfolger</u> |
| 5001             | 5041              |
| 5001             | 5043              |
| 5001             | 5049              |
| 5041             | 5216              |
| 5043             | 5052              |
| 5041             | 5052              |
| 5052             | 5259              |

| hören         |               |
|---------------|---------------|
| <u>MatrNr</u> | <u>VorINr</u> |
| 26120         | 5001          |
| 27550         | 5001          |
| 27550         | 4052          |
| 28106         | 5041          |
| 28106         | 5052          |
| 28106         | 5216          |
| 28106         | 5259          |
| 29120         | 5001          |
| 29120         | 5041          |
| 29120         | 5049          |
| 29555         | 5022          |
| 25403         | 5022          |

| prüfen        |               |               |      |
|---------------|---------------|---------------|------|
| <u>MatrNr</u> | <u>VorINr</u> | <u>PersNr</u> | Note |
| 28106         | 5001          | 2126          | 1    |
| 25403         | 5041          | 2125          | 2    |
| 27550         | 4630          | 2137          | 2    |