DATABASES

Lectures 4-5. SQL Basics

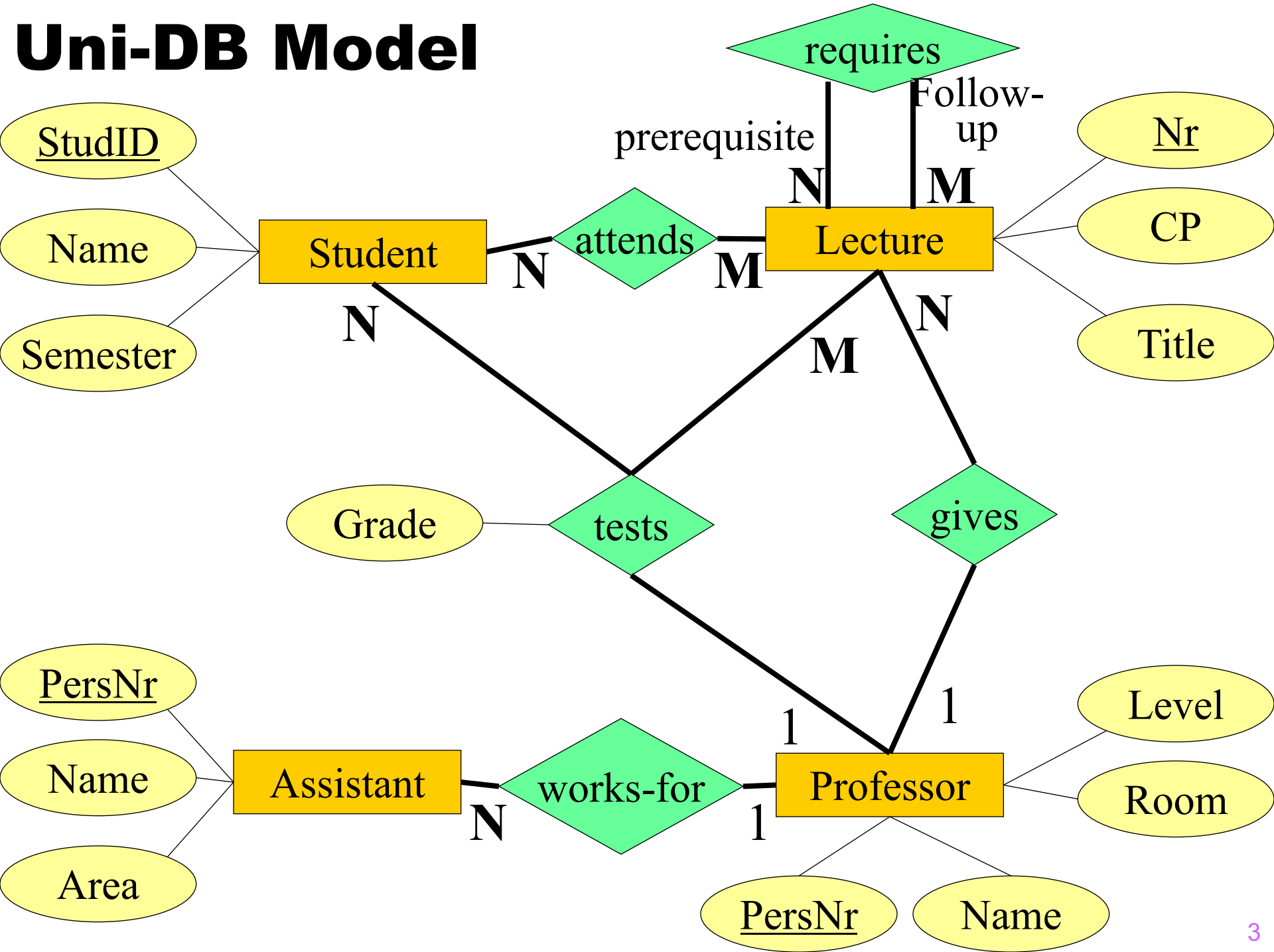
Alexander Breyman
Software Engineering Department
Computer Science Faculty

SQL (structured query language)

- A family of standards
  - Data definition language (DDL) - schemas
  - Data manipulation language (DML) - updates
  - Query language (Query) – reads

- History
  - 1974: first paper by Chamberlin&Boyce
  - SQL 92 (SQL 2): joins, outer-joins, ...
  - SQL 3: object-relational extensions
  - SQL/XML, etc.: domain-specific extensions

Uni-DB Model


requires
prerequisite
Follow-up
N
M
StudID
Name
Semester
Student
N
attends
M
Lecture
Nr
CP
Title
N
M
N
Grade
tests
gives
PersNr
Name
Area
Assistant
N
works-for
1
1
Professor
1
Level
Room
PersNr
Name

Professor

PersNr	Name	Level	Room
2125	Sokrates	FP	226
2126	Russel	FP	232
2127	Kopernikus	AP	310
2133	Popper	AP	52
2134	Augustinus	AP	309
2136	Curie	FP	36
2137	Kant	FP	7

Student

StudID	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

Lecture

Nr	Title	CP	PersNr
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

requires

Prereq.	Follow-up
5001	5041
5001	5043
5001	5049
5041	5216
5043	5052
5041	5052
5052	5259

attends

StudID	Nr
26120	5001
27550	5001
27550	4052
28106	5041
28106	5052
28106	5216
28106	5259
29120	5001
29120	5041
29120	5049
29555	5022
25403	5022

tests

StudID	Nr	PersNr	Grade
28106	5001	2126	1
25403	5041	2125	2
27550	4630	2137	2

Assistant

PerslNr	Name	Area	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

(Simple) Data Definition with SQL

Data types

- **character** *(n)*, ***char*** *(n)*
- **character varying** *(n)*, **varchar** *(n)*, **varchar2** *(n)*
- **numeric** *(precision, scale)*, **decimal**(precision,scale)
- **integer, bigint, smallint**
- **blob** or **raw** for large binaries
- **clob** for large string values
- **date, time, datetime**

Create Tables

- **create table** Professor

```
(PersNr      integer not null,
 Name        varchar (30) not null
 Level       character (2) default "AP");
```

DDL (ctd.)

Delete a Table

- **drop table** Professor**;**

Modify the structure of a Table

- **alter table** Professor **add column(**age **integer);**

Management of Indexes (Performance tuning)

- **create index** myIndex **on** Professor(name, age)**;**
- **drop index** myIndex**;**

Constraints. Why use a DBMS?

- Avoid redundancy and **inconsistency**
- Rich (declarative) access to the data
- Synchronize concurrent data access
- Recovery after system failures
- Security and privacy

- Reduce cost and pain to do something useful
  - There is always an alternative!!!

Integrity of Data

- Example Constraints
  - Keys
  - multiplicity of relationships
  - attribute domains
  - subset relationship for generalization
  - Referential integrity (foreign keys -> keys)

- Static Constraints
  - Constraints that any instance of a DB must meet

- Dynamic Constraints
  - Constraints on a state transition of the DB

Who checks? DB vs. App

- Why implement constraints in the DB?
  - Good way to annotate & document schema
  - DB is a central point (once and for all cases)
  - Safety net: in case you forget it in the app
  - Useful for DB-level optimization
    - Constraint: all students are older than 18 years.
    - Query: SELECT * FROM Student WHERE age < 17;
    - Query can be evaluated without looking at any student.

- Why implement constraints in the App?
  - Meaningful error messages.

- **It is important to do both!!!**

Referential Integritity Constraints

Foreign Keys

- Refer to tuple from a different relation
- E.g., PersNr in Lecture refers to a Professor

Definition: Referential Integritity

- For every foreign key one of the two conditions must hold
  - the value of the foreign key is *NULL* or
  - the referenced tuple must exist

- (Example on the Web: 404 Error becomes impossible)

Referential Integritity in SQL

- SQL Syntax to declare keys and foreign keys:
  - Key: **unique**
  - Primary key: **primary key**
  - Foreign key: **foreign key / references**

- Example:

```
create table R
        ( a integer primary key,
          b varchar(30) unique,
        ... );

create table S
        ( ...,
        c integer references R(c) );
```

Maintaining referential integrity?

Updates of referenced data which result in a violation

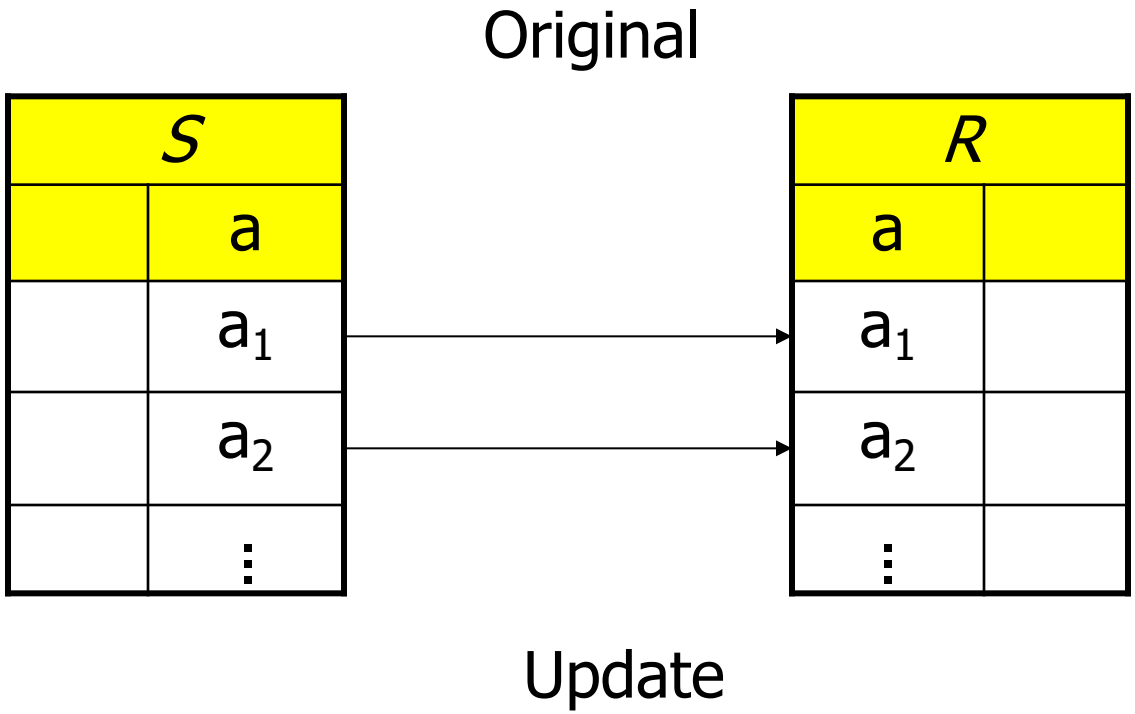
1. Default: reject the update (return an error)
2. **cascade:** propagate update
3. **set null:** set references to null

4. (Set references to default value. Not supported in SQL.)

The right choice depends on the ER model

- e.g. weak vs. strong entities
- relations that implement N:M relationships
- 1:N relations
- Exercise: extend rules for ER->relational translation!

Maintaining referential integrity



update *R*

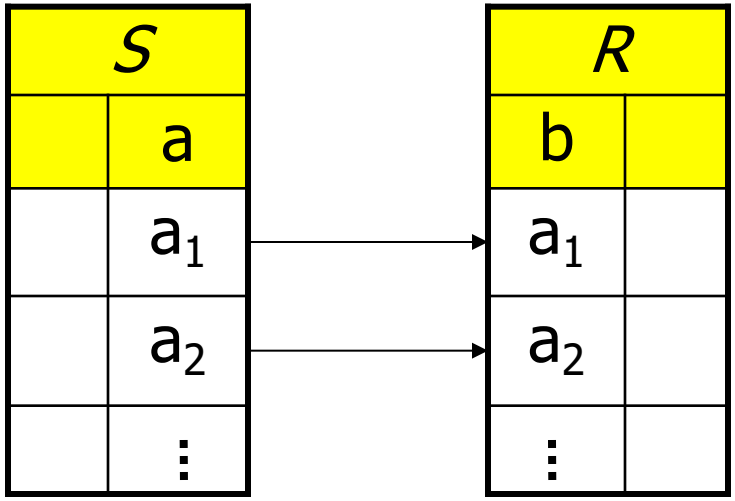
set a = a'$_1$

where a = a$_1$;

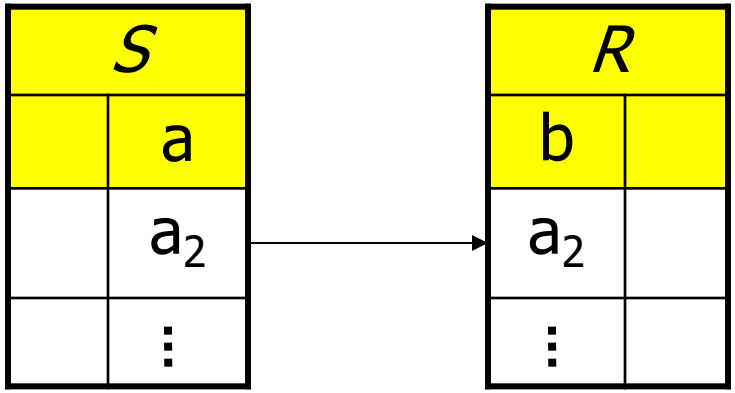
delete from *R*

where a = a$_1$;

Cascade (weak entities, n:m relationships)



Update of S



Delete in S

create table *S*

(...,

a **integer references** *R(b)*

on update cascade);

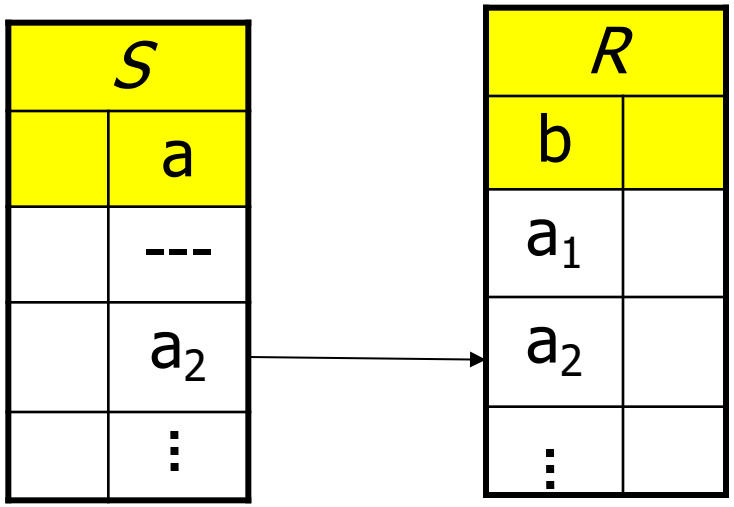
create table *S*

(...,

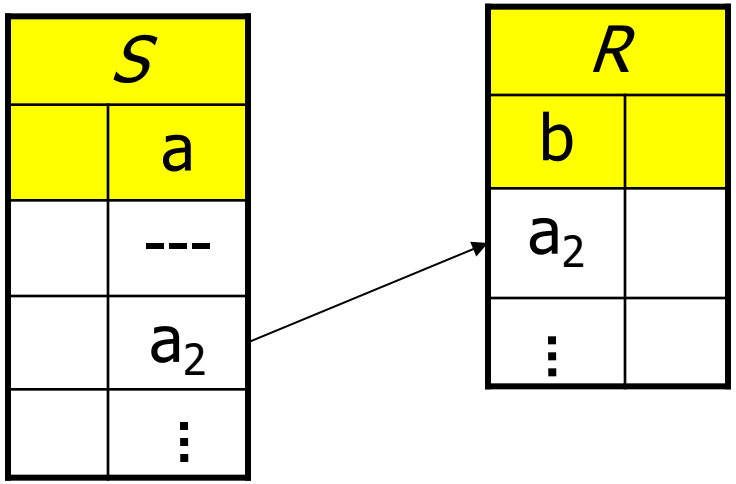
a **integer references** *R(b)*

on delete cascade);

Set Null (strong entities)



Update of S



Update of S

create table *S*

(...,

a **integer references** *R(b)*

on update set null);

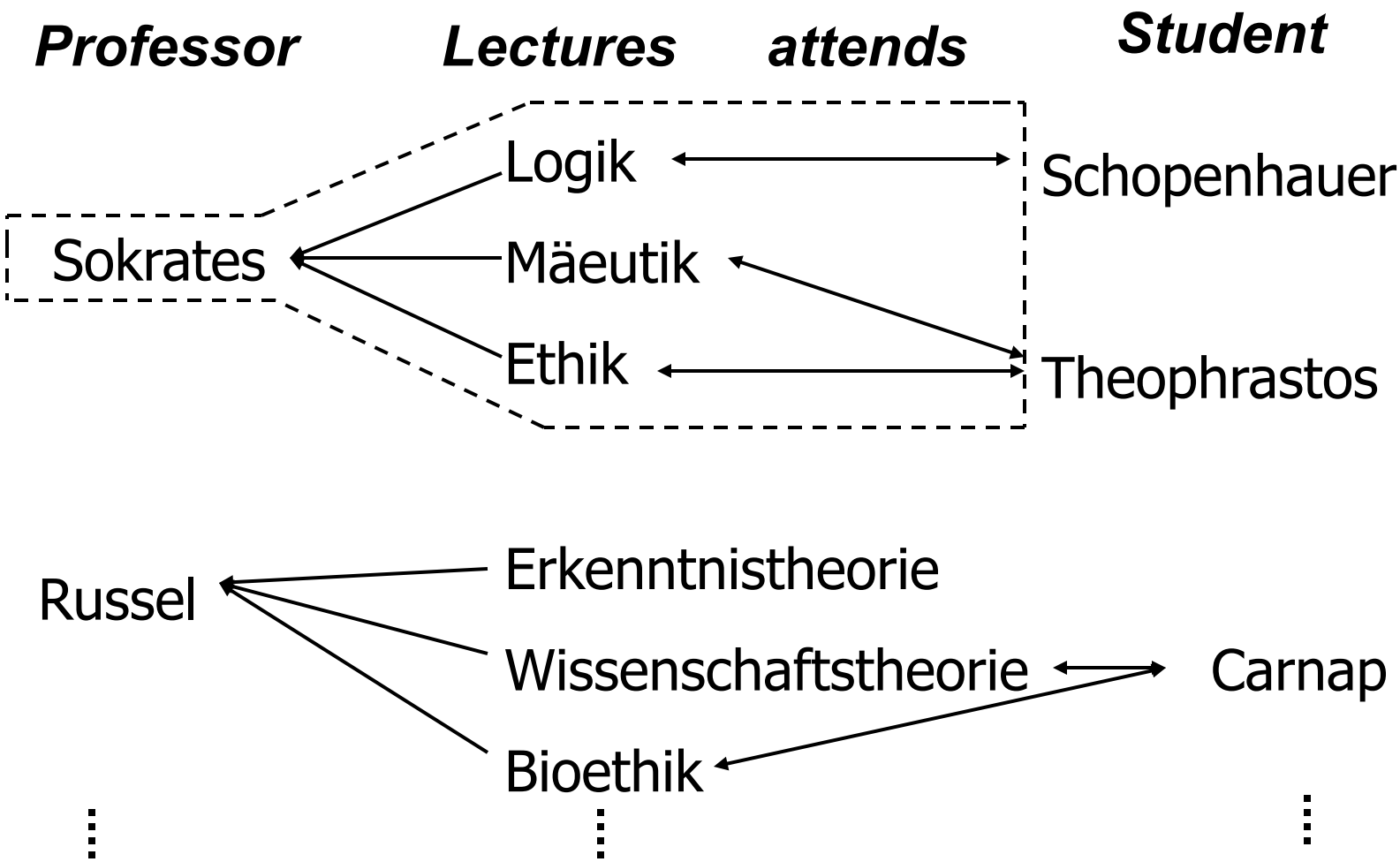
create table *S*

(...,

a **integer references** *R(b)*

on delete set null);

Cascading Deletes



```
create table Lecture
    ( ...,
    PersNr   integer
                    references Professor
                    on delete cascade);

create table attends
        ( ...,
        Nr    integer
                references Lecture
                on delete cascade);
```

Constraints on Domains

- Integer domains
  … **check** Semester **between** 1 **and** 13

- Enum types
  … **check** Level **in** (`Assistant´, `Associate´, `Full´) …

Uni-DB schema with Constraints

```
create table Student
    ( StudID       integer primary key,
    Name           varchar(30) not null,
    Semester       integer check Semester between 1 and 13),
create table Professor
    ( PersNr       integer primary key,
    Name           varchar(30) not null,
    Level          character(2) check (Level in (`AP´,`CP´,`FP´)),
    Room           integer unique );
```

```
create table Assistant
    ( PersNr        integer primary key,
    Name            varchar(30) not null,
    Area            varchar(30),
    Boss            integer,
    foreign key     (Boss) references Professor
                    on delete set null);
create table Lecture
    ( Nr            integer primary key,
    Title           varchar(30),
    CP              integer,
    PersNr          integer references Professor
                    on delete set null);
```

```
create table attends
   ( StudID          integer references Student
                     on delete cascade,
   Nr                integer references Lecture
                     on delete cascade,
   primary key  (StudID, Nr));
create table requires
   ( Prerequisite    integer references Lecture
                     on delete cascade,
   Follow-up         integer references Lecture
                     on delete cascade,
   primary key  (Prerequisite, Follow-up));
```

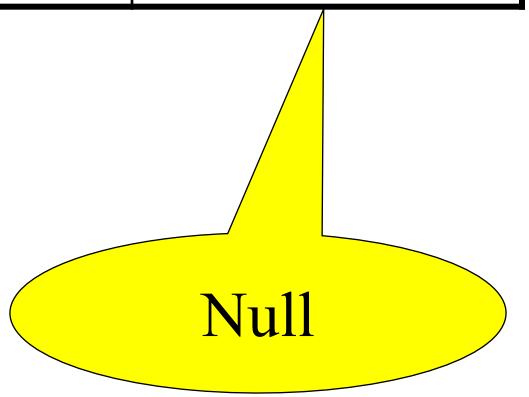
```
create table tests
    ( StudID        integer references Student
                    on delete cascade,
    Nr              integer references Lecture,
    PersNr          integer references Professor
                    on delete set null,
    Grade           numeric (3,2)
                    check (Grade between 1.0 and 6.0),
    primary key     (StudID, Nr));
```

Updates (DML)

Insert Tuples

```
insert into attends
        select StudID, Nr
        from Student, Lecture
        where Title= `Logik' ;

insert into Student (StudID, Name)
        values (28121,  `Archimedes');
```

Student		
StudID	Name	Semester
⋮	⋮	⋮
29120	Theophrastos	2
29555	Feuerbach	2
28121	Archimedes	-



Sequence Types (Oracle's Automatic Increment for Surrogates)

- **create sequence** PersNr_seq **increment by** 1 **start with** 1;
- **insert into** Professor(PersNr, Name)
  values(*PersNr_seq.nextval*, „Baseman");

- Syntax is vendor dependent
  - E.g., AUTO-INCREMENT Option in MySQL
  - Syntax above was standardized in SQL 2003
  - MS SQL Server since 2016

Updates (ctd.)

Delete tuples

delete Student

where Semester > 13;

Update tuples

update Student

set Semester= Semester + 1;

Queries

```
select    PersNr, Name
from      Professor
where     Level = ´FP´;
```

PersNr	Name
2125	Sokrates
2126	Russel
2136	Curie
2137	Kant

Queries: Sorting

select PersNr, Name, Level

from Professor

order by Level **desc**, Name **asc**;

PersNr	Name	Level
2136	Curie	FP
2137	Kant	FP
2126	Russel	FP
2125	Sokrates	FP
2134	Augustinus	AP
2127	Kopernikus	AP
2133	Popper	AP

Duplicate Elimination

select distinct Level

from Professor

Level
AP
FP

Professor

PersNr	Name	Level	Room
2125	Sokrates	FP	226
2126	Russel	FP	232
2127	Kopernikus	AP	310
2133	Popper	AP	52
2134	Augustinus	AP	309
2136	Curie	FP	36
2137	Kant	FP	7

Student

StudID	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

Lecture

Nr	Title	CP	PersNr
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

requires

Prereq.	Follow-up
5001	5041
5001	5043
5001	5049
5041	5216
5043	5052
5041	5052
5052	5259

attends

StudID	Nr
26120	5001
27550	5001
27550	4052
28106	5041
28106	5052
28106	5216
28106	5259
29120	5001
29120	5041
29120	5049
29555	5022
25403	5022

tests

StudID	Nr	PersNr	Grade
28106	5001	2126	1
25403	5041	2125	2
27550	4630	2137	2

Assistant

PerslNr	Name	Area	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

Queries: Joins

Who teaches Mäeutik?

select Name
from Professor, Lecture
where PersNr = ProfNr **and** Title = `Mäeutik‘ ;

$$\prod_{\text{Name}} (\sigma_{\text{PersNr} = \text{ProfNr} \wedge \text{Title} = \text{'Mäeutik'}} (\text{Professor} \times \text{Lecture}))$$

N.B.: Renamed Lecture.PersNr to ProfNr. Will show later how this can be done as part of a query.

Joins

Professor			
PersNr	Name	Level	Room
2125	Sokrates	FP	226
2126	Russel	FP	232
⋮	⋮	⋮	⋮
2137	Kant	FP	7

Lecture			
Nr	Title	CP	ProfNr
5001	Grundzüge	4	2137
5041	Ethik	4	2125
⋮	⋮	⋮	⋮
5049	Mäeutik	2	2125
⋮	⋮	⋮	⋮
4630	Die 3 Kritiken	4	2137

X

PersN	Name	Level	Room	Nr	Title	CP	ProfNr
2125	Sokrates	FP	226	5001	Grundzüge	4	2137
2125	Sokrates	FP	226	5041	Ethik	4	2125
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
2125	Sokrates	FP	226	5049	Mäeutik	2	2125
⋮	⋮		⋮	⋮	⋮	⋮	⋮
2126	Russel	FP	232	5001	Grundzüge	4	2137
2126	Russel	FP	232	5041	Ethik	4	2125
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
2137	Kant	FP	7	4630	Die 3 Kritiken	4	2137

↓ σ

PersNr	Name	Level	Room	VorlNr	Title	CP	ProfNr
2125	Sokrates	FP	226	5049	Mäeutik	2	2125

↓ π

Name
Sokrates

SQL -> Relational Algebra

SQL

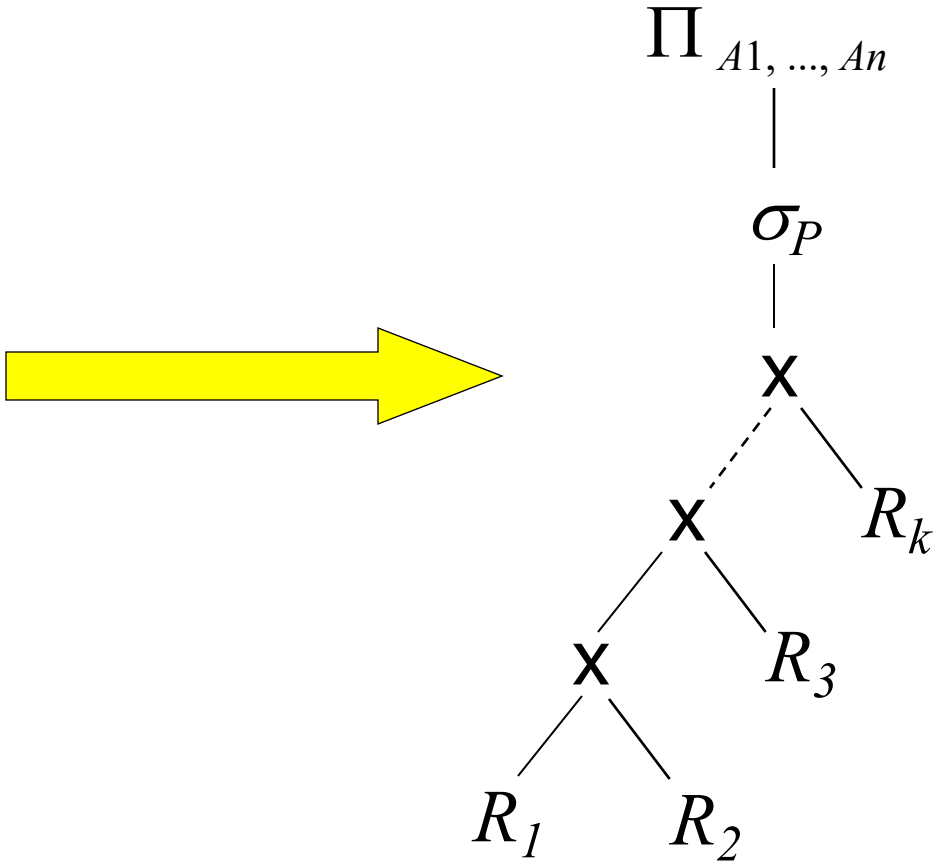
select A_1, ..., A_n

from R_1, ..., R_k

where P;

Relational Algebra

$$\Pi_{A1, ..., An}(\sigma_P (R_1 \times ... \times R_k))$$



Joins and Tuple Variables

Who attends which lecture?

```
select Name, Title
from Student, attends, Lecture
where Student.StudID = attends.StudID and
        attends.Nr = Lecture.Nr;
```

Alternative:

```
select s.Name, l.Title
from Student s, attends a, Lecture l
where s.StudID = a.StudID and
        a.Nr = l.Nr;
```

Rename of Attributes

Give title and professor of all lectures?

select Title, PersNr as ProfNr
from Lecture;

Set Operations

SQL supports: **union, intersect, minus**

```
( select Name
  from Assistant )
union
( select Name
  from Professor);
```

Grouping, Aggregation

Aggregate functions: **avg, max, min, count, sum**

select avg (Semester)
from Student;

Grouping, Aggregation

Aggregate functions: **avg, max, min, count, sum**

```
select avg (Semester)
from Student;

select PersNr, sum (CP)
from Lecture
group by PersNr;
```

Grouping, Aggregation

Aggregate functions: **avg, max, min, count, sum**

```
select avg (Semester)
from Student;

select PersNr, sum (CP)
from Lecture
group by PersNr;

select p.PersNr, Name, sum (CP)
from Lecture l, Professor p
where l.PersNr= p.PersNr and level = ´FP´
group by p.PersNr, Name
having avg (CP) >= 3;
```

Group By

Lecture x Professor							
Nr	Title	CP	PersNr	PersNr	Name	Level	Room
5001	Grundzüge	4	2137	2125	Sokrates	FP	226
5041	Ethik	4	2125	2125	Sokrates	FP	226
...	...	...	...	...	...	...	...
4630	Die 3 Kritiken	4	2137	2137	Kant	FP	7

⇓ σ (**where**)

Nr	Title	CP	PersNr	PersNr	Name	Level	Room
5001	Grundzüge	4	2137	2137	Kant	FP	7
5041	Ethik	4	2125	2125	Sokrates	FP	226
5043	Erkenntnis-theorie	3	2126	2126	Russel	FP	232
5049	Mäeutik	2	2125	2125	Sokrates	FP	226
4052	Logik	4	2125	2125	Sokrates	FP	226
5052	Wissenschafts-theorie	3	2126	2126	Russel	FP	232
5216	Bioethik	2	2126	2126	Russel	FP	232
4630	Die 3 Kritiken	4	2137	2137	Kant	FP	7

⇓ **group by**

Nr	Title	CP	PersNr	PersNr	Name	Level	Room
5041	Ethik	4	2125	2125	Sokrates	FP	226
5049	Mäeutik	2	2125	2125	Sokrates	FP	226
4052	Logik	4	2125	2125	Sokrates	FP	226
5043	Erkenntnistheorie	3	2126	2126	Russel	FP	232
5052	Wissenschaftstheo.	3	2126	2126	Russel	FP	232
5216	Bioethik	2	2126	2126	Russel	FP	232
5001	Grundzüge	4	2137	2137	Kant	FP	7
4630	Die 3 Kritiken	4	2137	2137	Kant	FP	7

⇓ **having**

Nr	Title	CP	PersNr	PersNr	Name	Level	Room
5041	Ethik	4	2125	2125	Sokrates	FP	226
5049	Mäeutik	2	2125	2125	Sokrates	FP	226
4052	Logik	4	2125	2125	Sokrates	FP	226
5001	Grundzüge	4	2137	2137	Kant	FP	7
4630	Die 3 Kritiken	4	2137	2137	Kant	FP	7

⇓ π & **sum (seclect)**

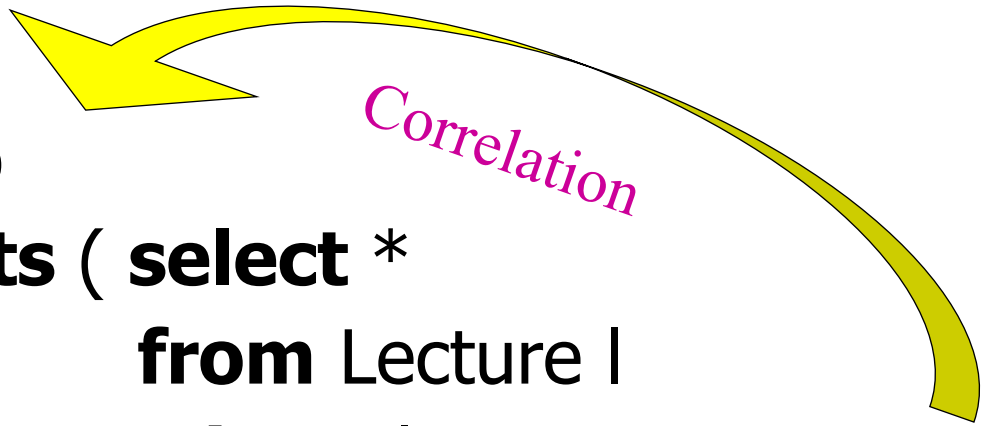
PersNr	Name	**sum** (CP)
2125	Sokrates	10
2137	Kant	8

Existential Quantification: exists sub-queries

```
select p.Name
from Professor p
where not exists ( select *
                   from Lecture l
                   where l.PersNr = p.PersNr );
```

Correlated Sub-queries

```
select p.Name
from Professor p
where not exists ( select *
                   from Lecture l
                   where l.PersNr = p.PersNr );
```



Uncorrelated Sub-query

```
select Name
from Professor
where PersNr not in ( select PersNr
                      from Lecture);
```

What is better? Correlated or uncorrelated?

Sub-queries with all

Not as powerful as relational division!

```
select Name
from Student
where Semester >= all ( select Semester
                        from Student);
```

Subqueries in SELECT, FROM

```
select PersNr, Name, ( select sum (CP) as load
                       from Lecture l
                       where p.PersNr=l.PersNr )
from Professor p;
```

```
select p.PersNr, Name, l.load
from Professor p, ( select PersNr, sum (CP) as load
                    from Lecture
                    group by PersNr ) l
where p.PersNr = l.PersNr;
```

Is this better than the simple Group By Query from before?

Query Rewrite

```
select *
from Assistant a
where exists
    ( select *
      from Professor p
      where a.Boss = p.PersNr and p.age < a.age);
```

- Equivalent Join Query: Why is this better?

```
select a.*
from Assistant a, Professor p
where a.Boss=p.PersNr and p.age < a.age;
```

Universal Quantifiaction

- SQL does not support relational division directly
- Need to play tricks

- This can be implemented in SQL:

```
select *
from Student s
where not exists
   (select *
    from Lecture l
    where l.CP = 4 and not exists
       (select *
        from attends a
        where a.Nr = l.Nr and a.StudID=s.StudID) );
```

Or do it this way

```
select a.StudID
from attends a
group by a.StudID
having count (*) = (select count (*) from Lecture);
```

Considering only 4 CP lectures

```
select a.StudID
from attends a, Lecture l
where a.Nr = l.Nr and l.CP = 4
group by a.StudID
having count (*) = (select count (*) from Lecture
                    where CP = 4);
```

Null Values (NULL = UNKNOWN)

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

vs.

```
select count (*)
from Student;
```

- Are those two queries equivalent?

Working with Null Values

1. Arithmetics: Propagate **null**: If an operand is null, the result is **null**.
   - **null** + 1 -> **null**
   - **null** * 0 -> **null**
2. Comparisons: New Boolean value **unknown**. All comparisons that involve a **null** value, evaluate to **unknown**.
   - **null** = **null** **-> unknown**
   - **null** < 13 **-> unknown**
   - **null** > **null** **-> unknown**
3. Logic: Boolean operators are evaluated using the following tables (next slide):

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

4. **where**: Only tuples which evaluate to **true** are part of the query result. (**unknown** and **false** are equivalent here):

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

5. **group by:** If exists, then there is a group for **null**.

```
select count (*)
from Student
group by Semester;
```

Predicates with null:

```
select count (*) from Student
where Semester is null;
```

Syntactic Sugar

```
select *
from Student
where Semester > = 1 and Semester < = 6;
```

```
select *
from Student
where Semester between 1 and 6;
```

```
select *
from Student
where Semester in (2,4,6);
```

case

```
select StudID, ( case when Grade >= 5.5 then ´sehr gut´
                      when Grade >= 5.0 then ´gut´
                      when Grade >= 4.5 then ´befriedigend´
                      when Grade >= 4.0 then ´ausreichend´
                      else ´nicht bestanden´end)
from tests;
```

- Behaves like a switch: evaluate from top to bottom
- No „break“ needed because at most one clause executed. Why is that?

Comparisons with like

- "%„ represents any sequence of characters (0 to n)
- "_„ represents exactly one character
- N.B.: For comparisons with **=** , % and _ are normal chars.

```
select *
from Student
where Name like ´T%eophrastos´;
```

```
select distinct Name
from Lecture l, attends a, Student s
where s.StudID = a.StudID and a.Nr = l.Nr
        and  l.Title like ´%thik%´;
```

Joins in SQL-92

- **cross join:** Cartesian product
- **natural join:**
- **join** or **inner join**: Theta-Join
- **left, right** or **full outer join**: outer join variants
- (union join: not discussed here)

select *
from R_1, R_2
where $R_1.A = R_2.B$;

select *
from R_1 **join** R_2 **on** $R_1.A = R_2.B$;

Left Outer Joins

```
select p.PersNr, p.Name, t.PersNr, t.Grade, t.StudID, s.StudID,
    s.Name
from Professor p left outer join
        (tests t left outer join Student s
                on t.StudID= s.StudID)
        on p.PersNr=t.PersNr;
```

PersNr	p.Name	t.PersNr	t.Grade	t.StudID	s.StudID	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	-	-	-	-	-
⋮	⋮	⋮	⋮	⋮	⋮	⋮

Right Outer Joins

select p.PersNr, p.Name, t.PersNr, t.Grade, t.StudID, s.StudID, s.Name

from Professor p **right outer join**

(tests t **right outer join** Student s **on**

t.StudID= s.StudID)

on p.PersNr=t.PersNr;

PersNr	p.Name	t.PersNr	t.Grade	t.StudID	s.StudID	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
:	:	:	:	:	:	:

Full Outer Joins

```
select p.PersNr, p.Name, t.PersNr, t.Grade, t.StudID,  s.StudID,
s.Name
from Professor p full outer join
                (tests t full outer join Student s on
                                                t.StudID= s.StudID)
                on p.PersNr=t.PersNr;
```

p.PersNr	p.Name	t.PersNr	t.Grade	t.StudID	s.StudID	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	-	-	-	-	-
⋮	⋮	⋮	⋮	⋮	⋮	⋮

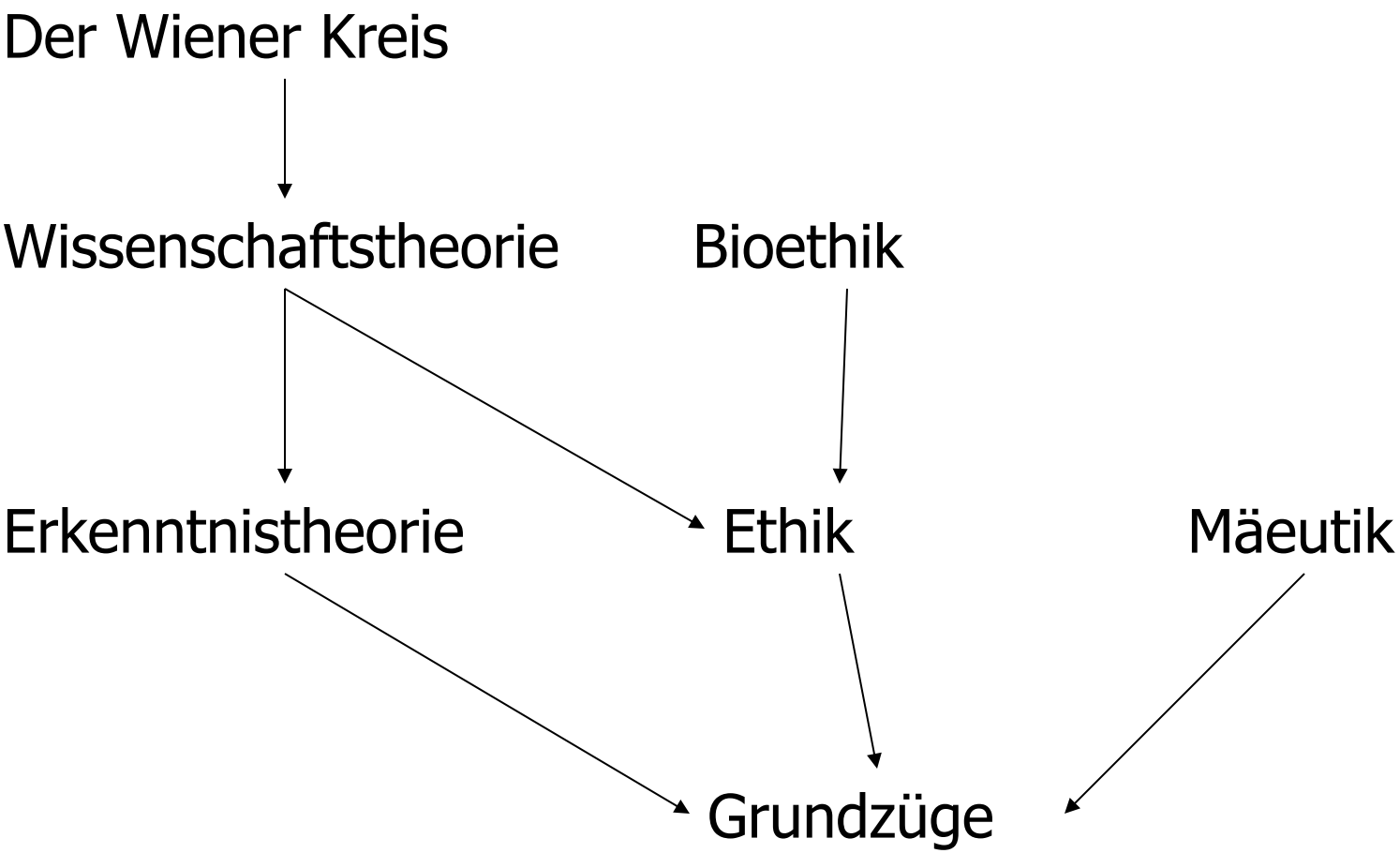
Recursion

select Prerequisite

from requires, Lecture

where Follow-up = Nr and

Title = `Der Wiener Kreis´

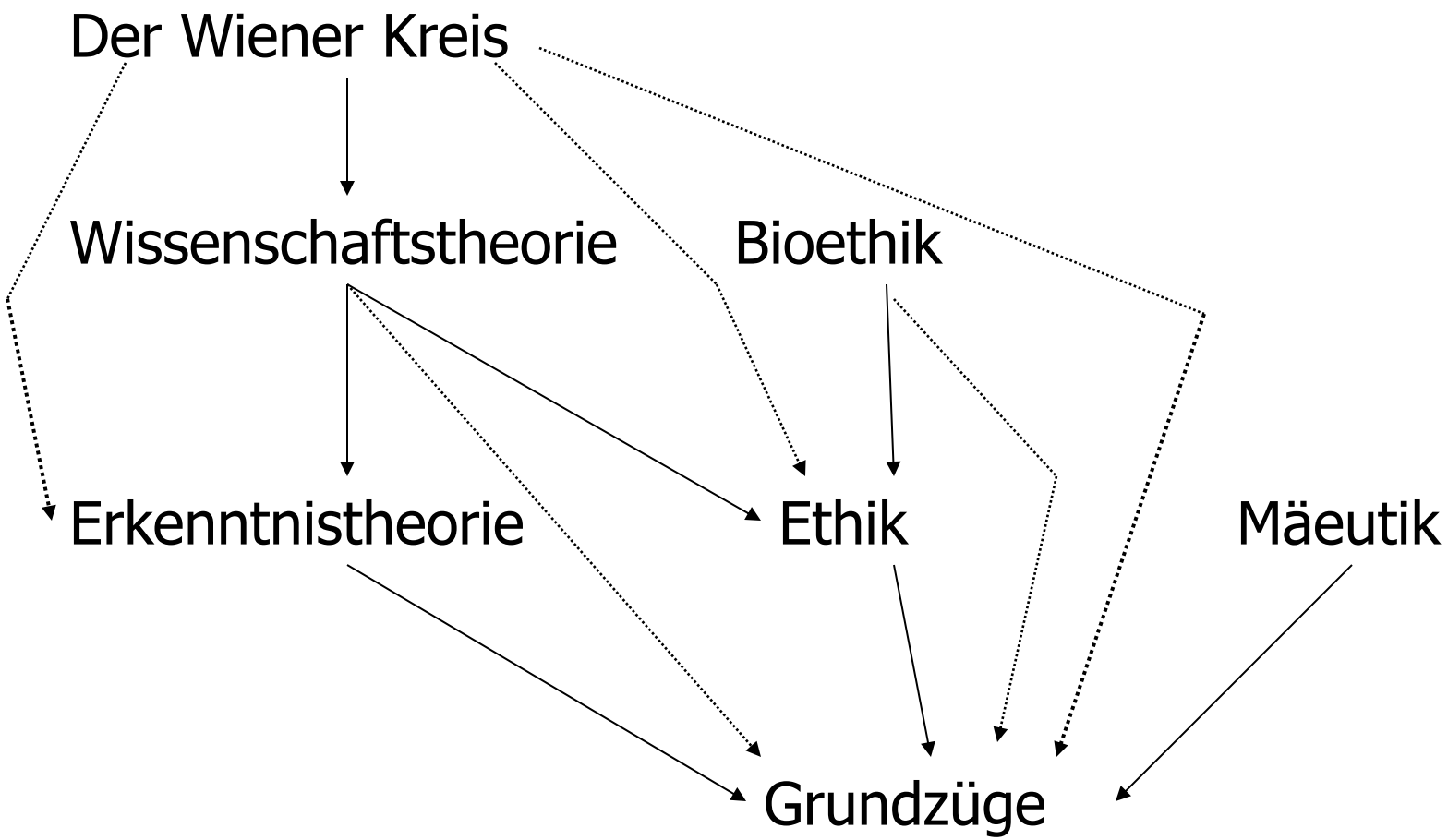

Der Wiener Kreis
Wissenschaftstheorie
Bioethik
Erkenntnistheorie
Ethik
Mäeutik
Grundzüge

Recursion

```
select l1.prerequisite
from requires l1, requires l2, Lecture l
where l1.Follow-up = l2.prerequisite and
        l2.Follow-up = l.Nr and
        l.Title=`Der Wiener Kreis´;
```

Requirements of „Wiener Kreis“ up to *N* levels

```
select l1.prerequisit
from requires l1
        ⋮
        requires ln_minus_1
        requires ln,

        Lecture l

where l1.follow-up = l2.prerequisite and
        ⋮

        ln_minus_1.follow-up = ln.prerequisite and

        ln.follow-up = l.Nr and

        l.Title= `Der Wiener Kreis´
```


Der Wiener Kreis
Wissenschaftstheorie
Bioethik
Erkenntnistheorie
Ethik
Mäeutik
Grundzüge

connect by Clause (Oracle)

```
select Title
from Lecture
where Nr in (select prerequisite
                    from requires
                    connect by follow-up =  prior prerequisite
                    start with follow-up = (select Nr
                                                    from Lecture
                                                    where Title = ... ));
```

Grundzüge
Ethik
Erkenntnistheorie
Wissenschaftstheorie

Recursion in DB2/SQL99

```
with TransLecture (First, Next)
as (select prerequisite, follow-up from requires
        union all
    select t.First, r.follow-up
    from TransLecture t, requires r
    where t.Next= r.prerequisite)

select Title from Lecture where Nr in
    (select First from TransLecture where Next in
        (select Nr from Lecture
         where Title = `Der Wiener Kreis´) )
```

Data Manipulation Language

Insert tuples

```
insert into attends
        select StudID, Nr
        from Student, Lecture
        where Title= `Logik´;

insert into Student (StudID, Name)
        values (28121,  `Archimedes´);
```

Student		
StudID	Name	Semester
⋮	⋮	⋮
29120	Theophrastos	2
29555	Feuerbach	2
28121	Archimedes	-

Deletion of tuples, Update

delete Student

where Semester > 13;

update Student

set Semester= Semester + 1;

Snapshot Semantics

1. Phase 1: mark tuples which are affected by the update
2. Phase 2: implement update on marked tuples

Otherwise, indeterministic execution of updates:

delete from requires
  where prerequisite **in** (**select** follow-up
                from requires);

requires	
Prerequisite	Follow-up
5001	5041
5001	5043
5001	5049
5041	5216
5043	5052
5041	5052
5052	5229

requires	
Prerequisite	Follow-up
5001	5041
5001	5043
5001	5049
5041	5216
5043	5052
5041	5052
5052	5229

```
delete from requries
    where Prerequisite in (select Follow-up
                           from requires);
```

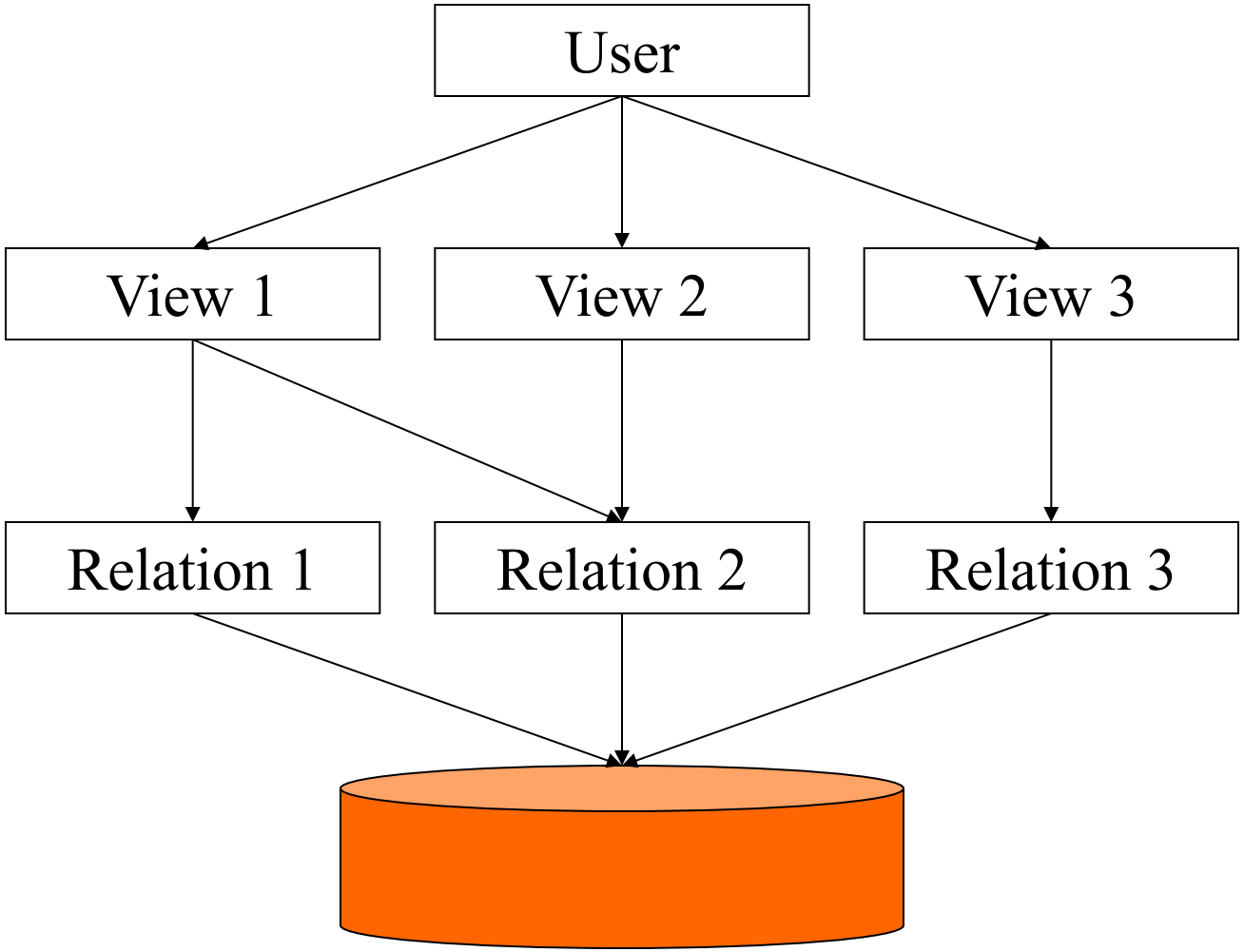
requires	
Prerequisite	Follow-up
5001	5041
5001	5043
5001	5049
5041	5216
5043	5052
5041	5052
5052	5229

```
delete from requries
    where Prerequisite in (select Follow-up
                           from requires);
```

Views for Logical Data Independence

Logical
data independence

Physical
data independence



Views ...

for privacy

```
create view testView as
    select StudID, Nr, PersNr
    from tests;
```

Views ...

for simpler queries

```
create view StudProf (Sname, Semester, Title, Pname) as
    select s.Name, s.Semester, l.Title, p.Name
    from Student s, attends a, Lecture l, Professor p
    where s.StudID=a.StudID and a.Nr=l.Nr and
              l.PersNr= p.PersNr;
```

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

Views for is-a relationships

```
create table Employee
        (PersNr    integer not null,
        Name       varchar (30) not null);
create table ProfData
        (PersNr    integer not null,
        Level      character(2),
        Room       integer);
create table AssiData
        (PersNr          integer not null,
        area             varchar(30),
        Boss             integer);
```

```
create view Professor as
        select *
        from Employee e, ProfData d
        where e.PersNr=d.PersNr;
create view Assistant as
        select *
        from Employee e, AssiData d
        where e.PersNr=d.PersNr;
```

➔ Subtypes implemented as a view

```
create table Professor
        (PersNr        integer not null,
        Name           varchar (30) not null,
        Level          character (2),
        Room           integer);
create table Assistant
        (PersNr        integer not null,
        Name           varchar (30) not null,
        area           varchar (30),
        Boss           integer);
create table OtherEmps
        (PersNr        integer not null,
        Name           varchar (30) not null);
```

```
create view Employee as
        (select PersNr, Name
        from Professor)
          union
        (select PersNr, Name
        from Assistant)
         union
        (select*
        from OtherEmps);
```

➔ Supertypes implemented as a view

Updatable Views

Example view which is not updatable

```
create view ToughProf (PersNr, AvgGrade) as
    select PersNr, avg(Grade)
    from tests
    group by PersNr;

update ToughProf set AvgGrade= 6.0
     where PersNr = 4711;

insert into ToughProf
    values (4711, 6.0);
```

SQL tries to avoid indeterminisms.

What about this?

```
create view ToughProf (PersNr, AvgGrade) as
    select PersNr, avg(Grade)
    from tests
    group by PersNr;

delete ToughProf
    where PersNr = 4711;
```

Views and Updates

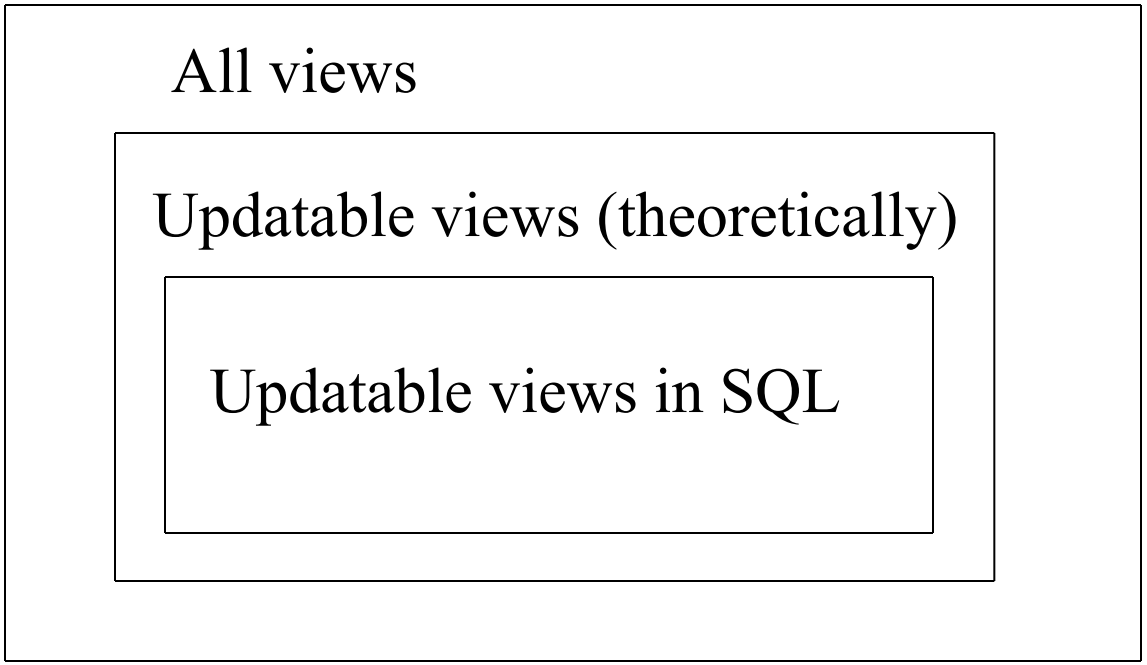
Example view which is not updatable

```
create view LectureView as
    select Title, CP, Name
    from Lecture l, Professor p
    where l.PersNr = p.PersNr;
insert into LectureView
    values (`Nihilismus', 2, `Nobody');
```

There are scenarios in which the „insert" is meaningful.
There are scenarios in which SQL would have to guess.
SQL is conservative and does not allow any scenario.

Views and Updates in SQL

- A SQL view is updatable iff
  - The view involves only one base relation
  - The view involves the key of that base relation
  - The view does NOT involve aggregates, group by, or duplicate-elmination



1:1 Relationships (Wedding)

```
create table Man(
   name   varchar(30)  primary key;
   spouse varchar(30)  references Woman);
create table Woman(
   name    varchar(30) primary key;
   spouse  varchar(30) references Man);
```

- Legal: Helga marries Hugo, but Hugo does not marry Helga.
  - Mutual marriage cannot be expressed in SQL.
  - How would you model marriage in SQL?

- N.B.: The real implementation is based on **transactions!**

Trigger (ECA Rules)

```
create trigger noDegradation
before update on Professor
for each row
when (old.Level is not null)
begin
    if :old.Level = 'Associate' and :new.Level = 'Assistant' then
        :new.Level := 'Associate';
    end if;
    if :old.Level = 'Full' then
        :new.Level := 'Full'
    end if;
    if :new.Level is null then
      :new.Level := :old.Level;
    end if;
end
```

Event

Condition

Action

Dangers of Triggers

```
create trigger weddingMan
after update on Man
for each row
when (true)
begin
    update Woman set spouse = :new.Name
    where name = :new.spouse;
    update Woman set spouse = null
    where name = :old.spouse;
end
```

- What happens if we write a weddingWoman trigger?
- Is marriage better modeled statically or dynamically?