Chapter 6': SQL – Data Retrieval

Content:

- More features for data retrieval in SQL: Aggregation, grouping, more joins, case ..

Next:

- Physical data organization: indexing

Aggregates and Grouping

select avg(semester) **from** students;

Students		
StudNr	**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

Result
1
7.625

Aggregates and Grouping

select semester, min(birthdate) **from** students **group by** semester

Students			
StudNr	**Name**	**Semester**	**birthdate**
24002	Xenokrates	12	1989-03-14
25403	Jonas	12	1994-01-14
26120	Fichte	10	1984-05-12
26830	Aristoxenos	3	1993-06-25
27550	Schopenhauer	3	1995-11-03
28106	Carnap	3	1990-08-30
29120	Theophrastos	2	1994-01-14
29555	Feuerbach	2	1993-07-01

Result	
Semester	**Semester**
1989-03-14	12
1984-05-12	10
1990-08-30	3
1993-07-01	2

Characteristics of Aggregates

- SQL creates one result tuple per group
- All attributes of the **select**-clause–except the aggregated–have to be listed in the **group by**-clause
- Thus SQL can make sure that the attribute value does not change within a group
- NULL value is an own group
- Aggregats **avg, max, min, count, sum**

Aggregates and Grouping

Find C4 professors that work for at most 8 hours a week on teaching.

Professors

PersNr	Name	Level	Room
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

Students

StudNr	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

Lectures

Lecture Nr	Title	Weekly Hours	Given_by
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

attend

StudNr	LectureNr
26120	5001
27550	5001
27550	4052
28106	5041
28106	5052
28106	5216
28106	5259
29120	5001
29120	5041
29120	5049
25403	5022
29555	5022
29555	5001

require

Predecessor	Successor
5001	5041
5001	5043
5001	5049
5041	5216
5043	5052
5041	5052
5052	5259

Assistants

PersNr	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

test

StudNr	LectureNr	PersNr	Grade
28106	5001	2126	1
25403	5041	2125	2
27550	4630	2137	2

Aggregates and Grouping

Find C4 professors that work for at most 8 hours a week on teaching.

```
select PersNr, Name, sum (WeeklyHours)
   from Lectures, Professors
   where Given_by = PersNr and Level = ´C4´
   group by PersNr, Name
        having sum (WeeklyHours) <= 8;
```

Example Execution

In the following slides we step through the (logical) execution of the following query:

```
select PersNr, Name, sum (WeeklyHours)
   from Lectures, Professors
   where Given_by = PersNr and Level = ´C4´
   group by PersNr, Name
        having sum (WeeklyHours) <= 8;
```

1. Cross Product

Lectures x Professors							
Lecture Nr	Title	Weekly Hours	Given_by	PersNr	Name	Level	Room
5001	Grundzüge	4	2137	2125	Sokrates	C4	226
5041	Ethik	4	2125	2133	Popper	C3	52
...	...	...	...	...	...	...	...
4630	Die 3 Kritiken	4	2137	2137	Kant	C4	7

⇓ Next: **where**-clause

2. Where Filtering

Lectures x Professors							
Lecture Nr	Title	Weekly Hours	Given_by	PersNr	Name	Level	Room
5001	Grundzüge	4	2137	2125	Sokrates	C4	226
5041	Ethik	4	2125	2125	Sokrates	C4	226
...	...	...	...	...	...	...	...
4630	Die 3 Kritiken	4	2137	2137	Kant	C4	7

⇓ Next: **join** condition

3. Join Condition

Lecture Nr	Title	Weekly Hours	Given_by	PersNr	Name	Level	Room
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

⇓ Next: group by

4. Grouping

Lecture Nr	Title	Weekly Hours	Given_by	PersNr	Name	Level	Room
5041	Ethik	4	2125	2125	Sokrates	C4	226
5049	Mäeutik	2	2125	2125	Sokrates	C4	226
4052	Logik	4	2125	2125	Sokrates	C4	226
5043	Erkenntnistheorie	3	2126	2126	Russel	C4	232
5052	Wissenschaftstheo.	3	2126	2126	Russel	C4	232
5216	Bioethik	2	2126	2126	Russel	C4	232
5001	Grundzüge	4	2137	2137	Kant	C4	7
4630	Die 3 Kritiken	4	2137	2137	Kant	C4	7

⇓ Next: **having** filtering

5. Having Filtering

Lecture Nr	Title	Weekly Hours	Given_by	PersNr	Name	Level	Room
5043	Erkenntnistheorie	3	2126	2126	Russel	C4	232
5052	Wissenschaftstheo.	3	2126	2126	Russel	C4	232
5216	Bioethik	2	2126	2126	Russel	C4	232
5001	Grundzüge	4	2137	2137	Kant	C4	7
4630	Die 3 Kritiken	4	2137	2137	Kant	C4	7

⇓ Next: Select aggregation (**sum**) and projektion

Result

PersNr	Name	**sum** (WeeklyHours)
2126	Russel	8
2137	Kant	8

Maximum / Minimum

Student with the highest StudNr

```
select  StudNr, Name
from Students
where StudNr =
   (select max(StudNr)
    from Student);
```

NOT

```
select Name, max(StudNr)
from Students;
```

Using the Result Set of a Sub-Query

```
select tmp.StudNr, tmp.Name, tmp.Number_of_Lectures
from (select s.StudNr, s.Name, count(*) as Number_of_Lectures
      from Students s, attend a
      where s.StudNr=a.StudNr
      group by s.StudNr, s.Name)  tmp
where tmp. Number_of_Lectures > 2;
```

StudNr	Name	Number_of_Lectures
28106	Carnap	4
29120	Theophrastos	3

… or alternativley

```
select tmp.StudNr, tmp.Name, tmp. Number_of_Lectures
 from (select s.StudNr, s.Name, count(*) as Number_of_Lectures
        from Students s, attend a
       where s.StudNr = a.StudNr
        group by s.StudNr, s.Name
        having count(*) > 2)  tmp;
```

Decision-Support-Query with Nested Sub-Queries

```
select  n.LectureNr, n.NumPerLect, t.TotalNum,
        n.NumPerLect /t.TotalNum as MarketShare
from    ( select LectureNr, count(*) as NumPerLect
          from attend
          group by LectureNr ) n,
        ( select count (*) as TotalNum
          from Students) t;
```

Result of the Query ?!

LectureNr	NumPerLect	TotalNum	MarketShare
4052	1	8	0
5001	3	8	0
5022	2	8	0
...	...	...	...

Decision-Support-Query with Nested Sub-Queries

```
select  n.LectureNr, n.NumPerLect, t.TotalNum,
        cast(n.NumPerLect as decimal(6,2)) / t.TotalNum
        as MarketShare
from    ( select LectureNr, count(*) as NumPerLect
          from attend
          group by LectureNr ) n,
        ( select count (*) as TotalNum
          from Students) t;
```

Professors

PersNr	Name	Level	Room
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

Students

StudNr	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

Lectures

Lecture Nr	Title	Weekly Hours	Given_by
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

attend

StudNr	LectureNr
26120	5001
27550	5001
27550	4052
28106	5041
28106	5052
28106	5216
28106	5259
29120	5001
29120	5041
29120	5049
25403	5022
29555	5022
29555	5001

require

Predecessor	Successor
5001	5041
5001	5043
5001	5049
5041	5216
5043	5052
5041	5052
5052	5259

Assistants

PersNr	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

test

StudNr	LectureNr	PersNr	Grade
28106	5001	2126	1
25403	5041	2125	2
27550	4630	2137	2

Result of the query

LectureNr	NumPerLect	TotalNum	MarketShare
4052	1	8	.125
5001	3	8	.375
5022	2	8	.25
...	...	...	...

Further Queries with Sub-Queries

```
select Name
from Professors
where PersNr not in ( select Given_by
                      from Lectures );

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

Case Construct

```
select StudNr, ( case when Grade < 1.5 then ´very good´
                      when Grade < 2.5 then ´good´
                      when Grade < 3.5 then ´satisfactory´
                      when Grade < 4.0 then ´sufficient´
                      else ´failed´end)
from test;
```

First qualifying **when**-clause is executed

Joins in SQL-92

- **cross join:** full cartesian product (not in all DBS!)
- **natural join:** equality test on all attributes with the same names, output of all attributes, those with the same names only once (not in all DBS!)

Name	StudNr
Carnap	1
Fichte	2

StudNr	LectureNr
2	Databases
2	Math

Joins in SQL-92

- **join** also called **inner join**: Theta Join (any conditions), equi-join (only equal condition)
- **left**, **right** or **full outer join**: keeps tuples with no join partner

Name	StudNr
Carnap	1
Fichte	2

StudNr	LectureNr
2	Databases
2	Math

Joins in SQL-92

- **semi-join**: no operator in SQL, expressed with **exists** or **in**

Name	StudNr
Carnap	1
Fichte	2

StudNr	LectureNr
2	Databases
2	Math

(Inner) Join

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

or alternatively

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

Outer Joins (left)

```
select p.PersNr, p.Name, t.PersNr, t.Grade,
   t.StudNr, s.StudNr, s.Name
from (Professors p
      left outer join test t
                  on p.PersNr = t.PersNr)
      left outer join Students s
                  on t.StudNr = s.StudNr;
```

Result

p.PersNr	p.Name	t.PersNr	t.Grade	t.StudNr	s.StudNr	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	null	null	null	null	null
⋮	⋮	⋮	⋮	⋮	⋮	⋮

Outer Joins (right)

```
select p.PersNr, p.Name, t.PersNr, t.Grade,
   t.StudNr, s.StudNr, s.Name
from (Professors p
      right outer join test t
                  on p.PersNr = t.PersNr)
      right outer join Students s
                  on t.StudNr = s.StudNr;
```

Result

p.PersNr	p.Name	t.PersNr	t.Grade	t.StudNr	s.StudNr	s.Name
2126	Russel	2126	1	28106	28106	Carnap
2125	Sokrates	2125	2	25403	25403	Jonas
2137	Kant	2137	2	27550	27550	Schopen-hauer
null	null	null	null	null	26120	Fichte
⋮	⋮	⋮	⋮	⋮	⋮	⋮

Outer Joins (full)

```
select p.PersNr, p.Name, t.PersNr, t.Grade,
   t.StudNr, s.StudNr, s.Name
from (Professors p
      full outer join test t
                  on p.PersNr = t.PersNr)
      full outer join Students s
                   on t.StudNr = s.StudNr;
```

Null values

- In SQL there is a special value **NULL**
- This value exists for all data types and represents values which are
  - unknown or
  - *not available* or
  - *not applicable*
- Null values can also emerge from query evaluation

- Test for NULL → **is NULL**

Example:

```
select   *
from     Professors
where    Room is NULL;
```

Null values cont.

- Null values are passed through in arithmetic expressions: at least one operand NULL → result is NULL as well
- Sometimes surprising query results, if Null values occur, e.g.:

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

- If there are students whose attribute value semester is a NULL value these are not counted
- The reason is three-valued logic with inclusion of NULL values

Null values cont.

sum(semester) = 14

count(*) = 3

count(semester) = 2

avg(semester) = 7

Students		
StudNr	**Name**	**Semester**
27550	Schopenhauer	4
25403	Jonas	null
26120	Fichte	10

Exercise SQL

Average Grade of the student Schopenhauer.

Name and number of given lectures for all professors.

Exercise SQL (cont.)

```
select name, avg(grade), count(*)
from students s left outer join test t
                        on s.studnr = t.studnr
where name = 'Schopenhauer'
group by name;

select persNr, name, count(lectureNr) as numLectures
from Professors p left outer join Lectures l
                        on p.persNr = l.given_by
where level = 'C4'
group by p.persNr, p.name
order by numLectures desc
```

Data Manipulation Language DML

Changes in the database: Insert

Insert of tuples by explicitly giving values:

insert into Students (StudNr, Name)

values (28121, 'Archimedes'), (4711, 'Pythagoras');

Changes in the database: Insert

Insert of tuples via a query

```
insert into attend
    select StudNr, LectureNr
    from Students, Lectures
    where Title= `Logik´;
```

(Mandatory registration of all students for ‚Logik')

Changes in the database : Insert

Insert of tuples from a file

Database system specific programs, e.g. DB2:

- **Import:**

```
IMPORT FROM studis.tbl OF DEL
INSERT INTO Students;
```

```
Analogously: EXPORT TO studis.tbl OF DEL
             SELECT * FROM Students;
```

- **Load:**

High-Performance alternative to import

Oracle: Load, Datapump, ...

Changes in the Database: delete, update

delete from Students

where Semester > 13;

Note: **delete from** Students

deletes all tuples from the relation

update Students

set Semester= Semester + 1;

Changes in two phase

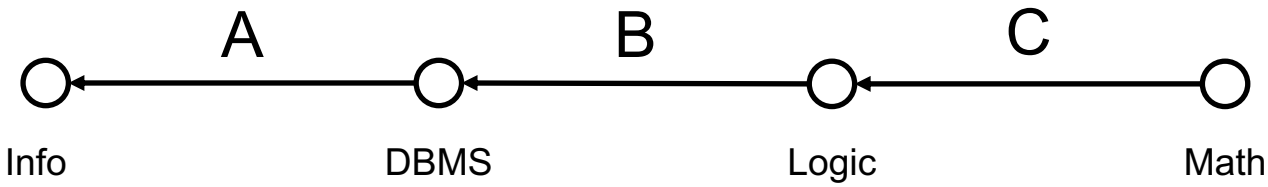
1. Candidates for changes are determined and marked
2. Changes are performed at the marked tuples

Otherwise changes can depend on the order of the tuples.

delete from Require
  where predecessor **in** (**select** successor
    from Require);

Example

```
delete from Require
      where predecessor in (select successor
                            from Require);
```



Example 2

```
delete from Require
       where predecessor in (select successor
                             from Require);
```

Require	
predecessor	successor
5001	5041
5001	5043
5001	5049
5041	5216
5043	5052
5041	5052
5052	5229

Execution in order of the example instance would (also) contain tuple (5052, 5229) erroneously as before all tuples with 5052 as *Successor* were deleted.

Data Definition Language DDL

Changes to the schema

- **drop table** <Table name>
- **alter table** <Table name>
  drop| **add column** <Attribute name> <Data type>
  alter column <Attribute name> **set default** <default>
  …

Further commands vendor specific, e.g. Oracle:

- **alter table** <Table name>
  - **modify** | **add column** <Attribute name> <Data type>
  - **drop column** <Attribute name>
  - **add** | **drop** | **enable** | **disable** <constraint clause>

Views ...

- Belong to DDL:
  create view <view name> **as** <select-statement>
- Often used to design queries more clear
- Are kind of a "virtual relation"
- Show an excerpt of the database

Advantages

- Simplify the access for certain user groups
- Can be used to restrict the access to the data

Disadvantages

- Not all (mostly none) views can be modified

Remember

```
select tmp.StudNr, tmp.Name, tmp. Number_of_Lectures
from (select s.StudNr, s.Name, count(*) as Number_of_Lectures
        from Students s, attend a
       where s.StudNr = a.StudNr
        group by s.StudNr, s.Name
        having count(*) > 2)  tmp;
```

… alternatively

```
create view tmp (StudNr, Name, Number_of_Lectures) as
(select s.StudNr, s.Name, count(*)
        from Students s, attend a
        where s.StudNr=a.StudNr
        group by s.StudNr, s.Name)

select * from tmp where Number_of_Lectures > 2;

drop view tmp;
```

… alternatively

```
with tmp (StudNr, Name, Number_of_Lectures) as
(select s.StudNr, s.Name, count(*)
        from Students s, attend a
        where s.StudNr=a.StudNr
        group by s.StudNr, s.Name)
select *
from tmp
where Number_of_Lectures > 2;
```

→ With creates a temporary table, only valid within the query

Simplifying Queries with Views

Complex query: Names of all professors who give a lecture with more weekly hours than the average weekly hours per lecture and with more than three assistants.

- Not all at once → divide into smaller more concise parts

- These parts can be realized by using views or or named intermediate results (‘with’)

Simplification

1. All professors with weekly hours more than the average of weekly hours:

```
create view AboveAverageWeeklyHours as
   select given_by
   from Lectures
   where WeeklyHours >
      (select avg (WeeklyHours) from Lectures);
```

Simplification

2. All professors with more than three assistants:

```
create view ManyAssistants as
   select Boss
   from Assistants
   group by Boss
   having count(*) > 3;
```

Simplification

- Combine
- Views can be used like common relations

```
select Name
from Professors
where PersNr in
   (select  given_by
   from AboveAverageWeeklyHours) and
            PersNr in
   (select Boss
   from ManyAssistants);
```

Expanding when executed

```
select Name
from Professors
where PersNr in
   (select  Given_by
   from (select Given_by                    AboveAverageWeeklyHours
   from Lectures
   where WeeklyHours >
      (select avg (WeeklyHours)
       from Lectures))) and
          PersNr in
   (select Boss
   from (select Boss
   from Assistants                          ManyAssistants
   group by Boss
   having count(*) > 3));
```

Views ...

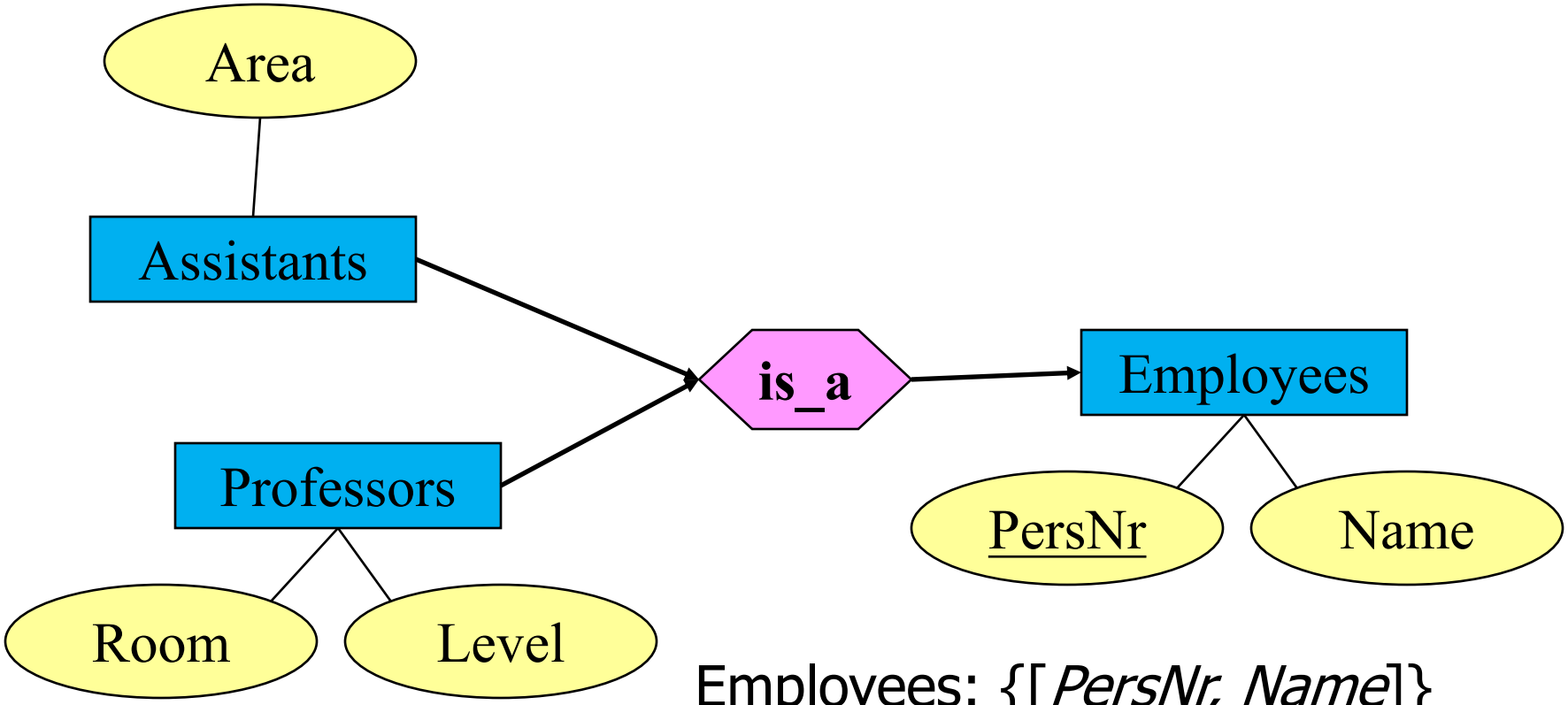
For data privacy

```
create view testView as
    select StudNr, LectureNr, PersNr
    from test
```

For statistics

```
create view TestQual(Name, QualLevel) as
      (select p.Name, avg(t.Grade)
       from Professors p join test t on
                        p.PersNr = t.PersNr
       group by p.Name, p.PersNr
       having count(*) > 50)
```

Relational Modelling of the Generalization



Employees: {[*PersNr, Name*]}

Professors: {[*PersNr, Level, Room*]}

Assistants: {[*PersNr, Area*]}

Table Definition

```
create table Employees
        (PersNr    integer not null,
         Name      varchar (30) not null);
create table ProfData
        (PersNr    integer not null,
         Level     character(2),
         Room      integer);
create table AssData
        (PersNr          integer not null,
         Area            varchar(30) );
```

Views to model generalization

```
create view Professors as
        select *
        from employees e, ProfData p
        where e.PersNr=p.PersNr;

create view Assistants as
        select *
        from Employees e, AssData d
        where e.PersNr=a.PersNr;
```

➔ subtypes as view

Table Definition

```
create table Professors
        (PersNr         integer not null,
        Name            varchar (30) not null,
        Level           character (2),
        Room            integer);

create table Assistants
        (PersNr         integer not null,
         Name           varchar (30) not null,
         Area           varchar (30) );

create table OtherEmployees
        (PersNr         integer not null,
         Name           varchar (30) not null);
```

Views to Model Generalization

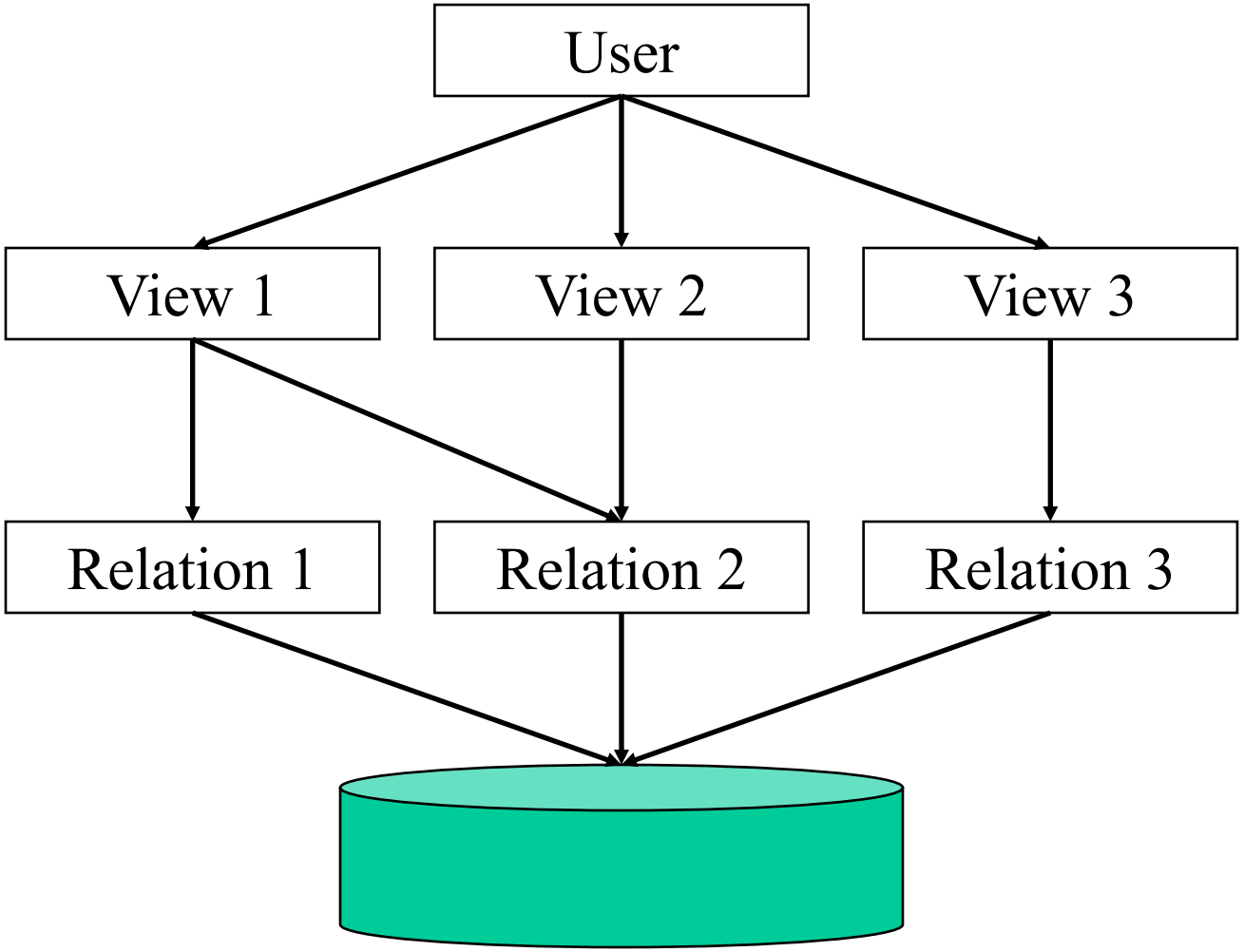
```
create view Employees as
        (select PersNr, Name
        from Professors)
          union
        (select PersNr, Name
        from Assistants)
         union
        (select *
        from OtherEmployees);
```

➔ supertype as view

Views to guarantee data independency

logical
data independency

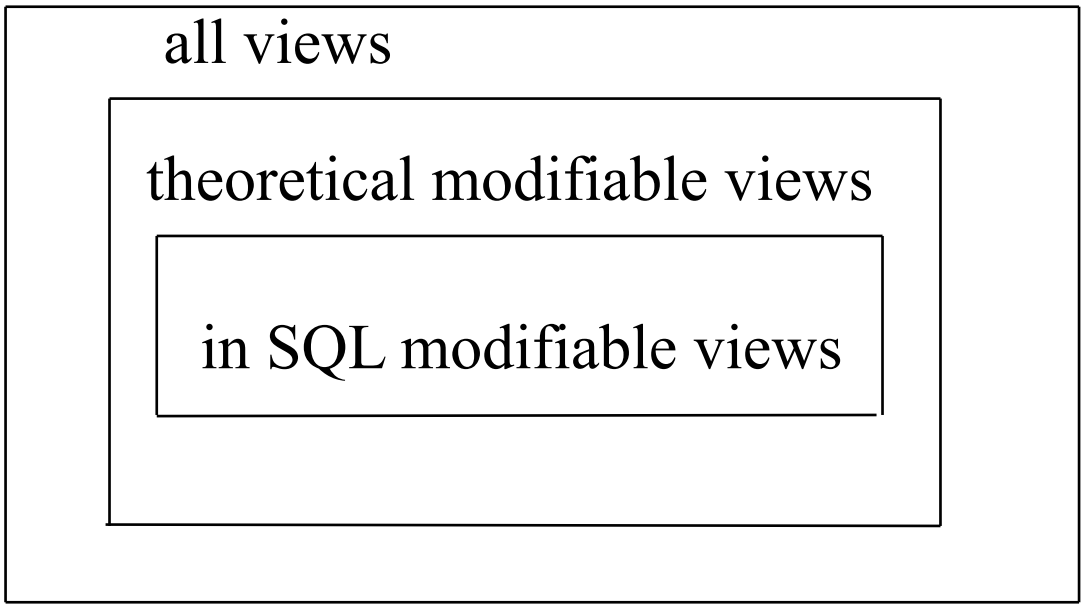
physical
data independency



Modifiability of views

In SQL

- Only one base relation
- Key must be part of
- No aggregation, grouping, duplicate elimination



Quiz

Table Airplane:

Producer	Type	NumberSeats
Boeing	B747-400	550
Boeing	B737-300	380
Airbus	A340-600	380
Airbus	A320-200	179
Airbus	A380	NULL

Every producer together with its type of airplane with the most seats

Result:

Producer	Type	SeatsMax
Boeing	B747-400	550
Airbus	A340-600	380

Quiz: Solution

```
with GroupProducer (Producer, SeatsMax)
as
   (select Producer, max (NumberSeats)
    from Airplane
    group by Producer)

select A.Producer, Type, SeatsMax
from Airplane A, GroupProducer G
where A.Producer = G.Producer and
          A.Seats = G.SeatsMax
```