Chapter 8: SQL – Data Managemet

Content:

- Using the data manipulation language in SQL to change data in a database system

Next:

- Physical data organization: indexing

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

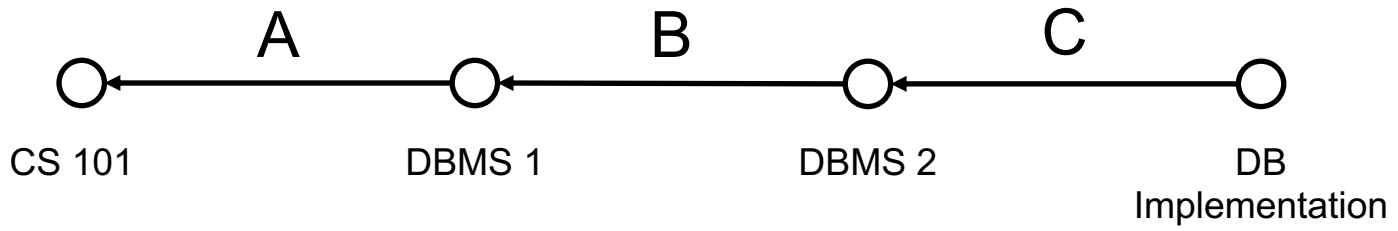
deletes all tuples from the relation

update Students

set Semester= Semester + 1;

Example

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



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);
```

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 this 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
```

… alternatively with view

```
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 with

```
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 ids 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 ids 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 ...

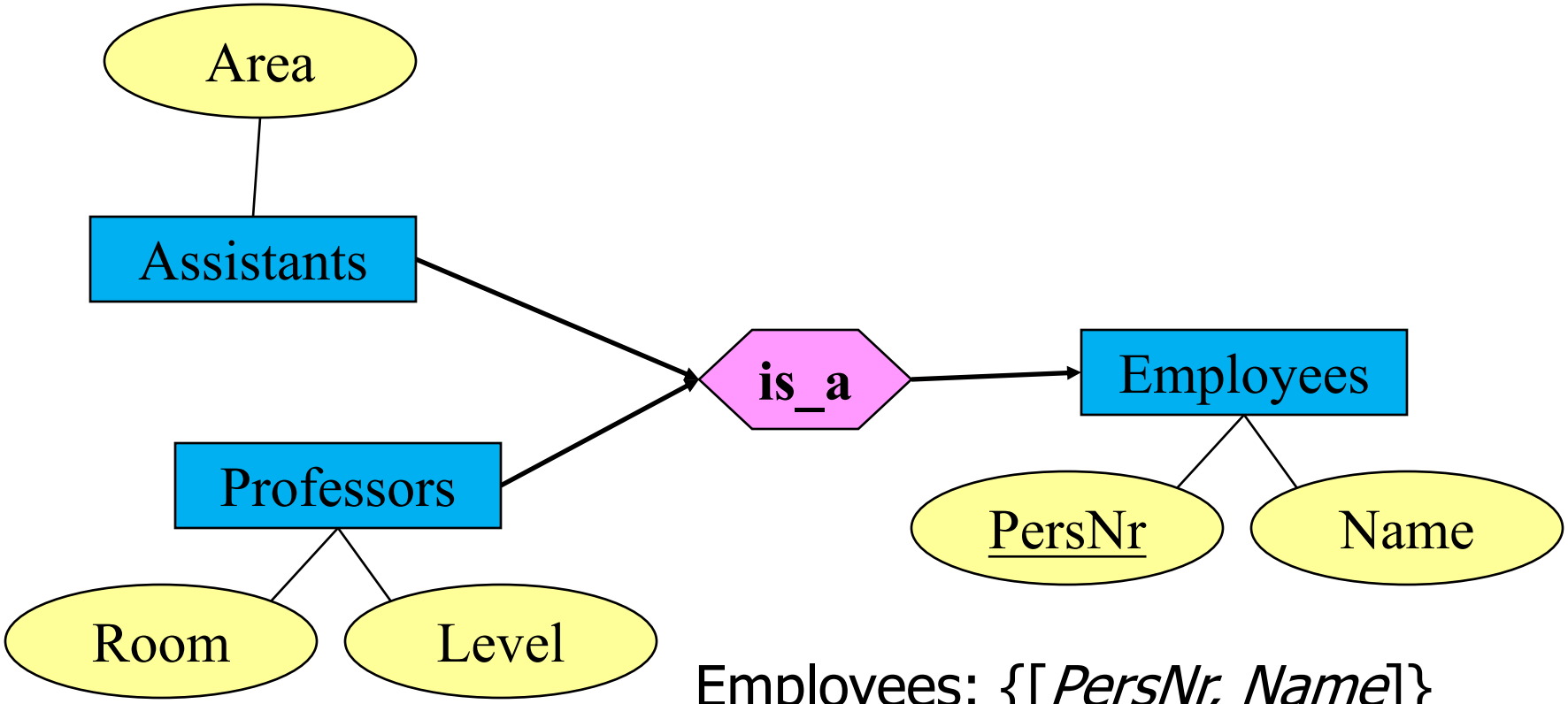
For data privacy

```
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

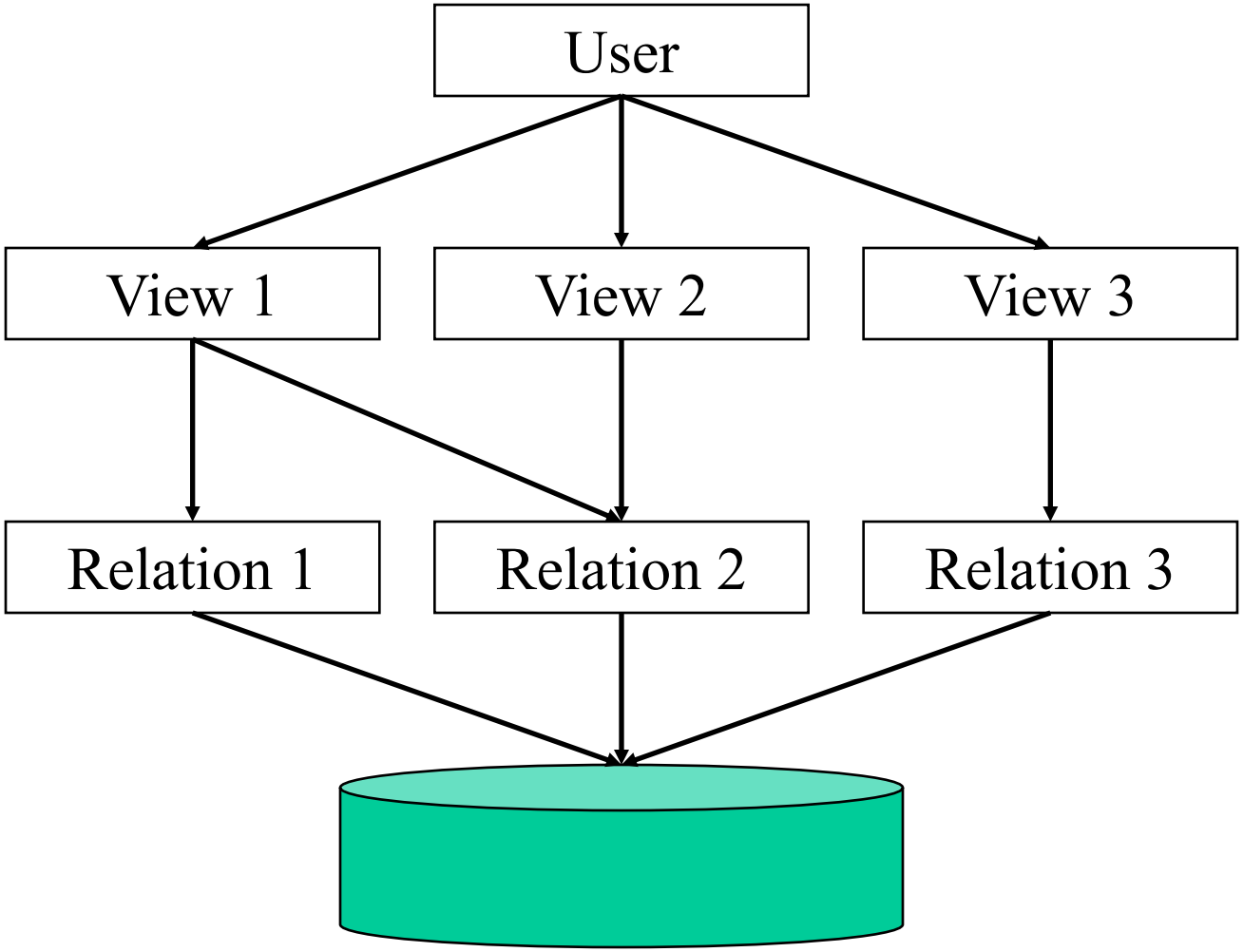
```
create view Employees as
        (select PersNr, Name
        from Professors)
          union
        (select PersNr, Name
        from Assistants)
         union
        (select *
        from OtherEmployees);
```

➔ supertype as view

Views to guarantee data independency

logical
data independency

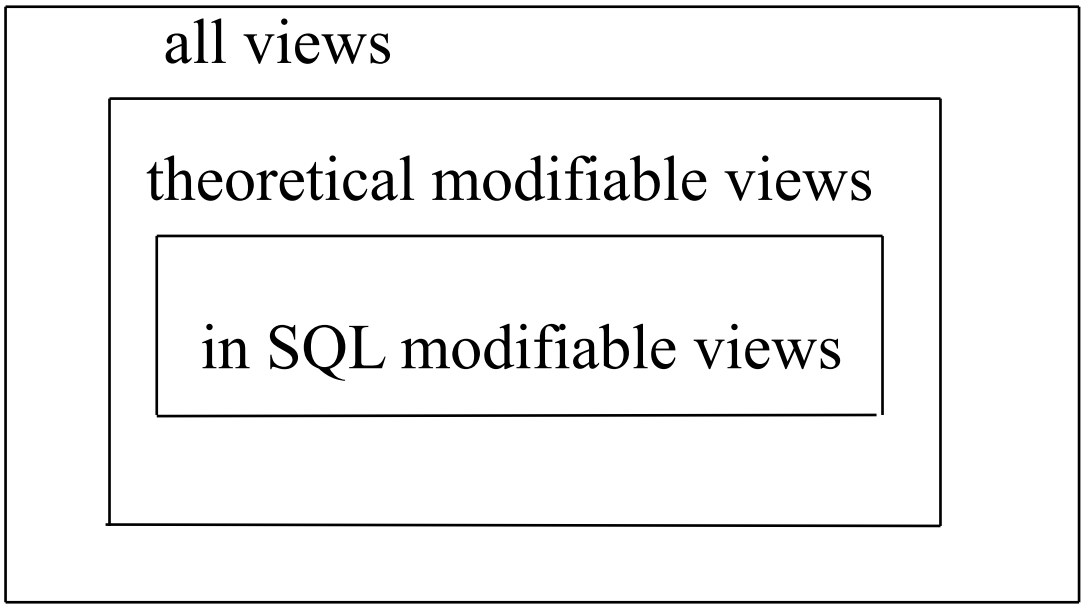
physical
data independency



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
```