

Maximum / Minimum

Gib mir den Studenten mit der größten MatrNr

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

NICHT

```
select Name, max(MatrNr)  
from Student;
```

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

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

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

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

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

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

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

Verwertung der Ergebnismenge einer Unteranfrage

```
select tmp.MatrNr, tmp.Name, tmp.VorlAnzahl
from (select s.MatrNr, s.Name, count(*) as VorlAnzahl
      from Studenten s, hoeren h
      where s.MatrNr=h.MatrNr
      group by s.MatrNr, s.Name) tmp
where tmp.VorlAnzahl > 2;
```

MatrNr	Name	VorlAnzahl
28106	Carnap	4
29120	Theophrastos	3

... oder auch

```
select tmp.MatrNr, tmp.Name, tmp.VorlAnzahl
from (select s.MatrNr, s.Name, count(*) as VorlAnzahl
      from Studenten s, hoeren h
      where s.MatrNr = h.MatrNr
      group by s.MatrNr, s.Name
      having count(*) > 2) tmp;
```

Decision-Support-Anfrage mit geschachtelten Unteranfragen

```
select h.VorlNr, h.AnzProVorl, g.GesamtAnz,  
       h.AnzProVorl/g.GesamtAnz as Marktanteil  
from   ( select VorlNr, count(*) as AnzProVorl  
        from hoeren  
        group by VorlNr ) h,  
       ( select count (*) as GesamtAnz  
        from Studenten) g;
```

Das Ergebnis der Anfrage ?!

VorlNr	AnzProVorl	GesamtAnz	Marktanteil
4052	1	8	0
5001	3	8	0
5022	2	8	0
...

Casting der Integer zu Decimal

```
select h.VorlNr, h.AnzProVorl, g.GesamtAnz,  
        cast(h.AnzProVorl as decimal(6,2)) / g.GesamtAnz  
as Marktanteil  
  
from ( select VorlNr, count(*) as AnzProVorl  
        from hören  
        group by VorlNr ) h,  
        ( select count (*) as GesamtAnz  
        from Studenten) g;
```

Das Ergebnis der Anfrage

VorlNr	AnzProVorl	GesamtAnz	Marktanteil
4052	1	8	.125
5001	3	8	.375
5022	2	8	.25
...

Weitere Anfragen mit Unteranfragen

```
select Name  
from Professoren  
where PersNr not in ( select gelesenVon  
                        from Vorlesungen );
```

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

Das 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;
```

Die **erste** qualifizierende **when**-Klausel
wird ausgeführt

Joins in SQL-92

- **cross join**: volles Kreuzprodukt (nicht in allen DBS!)
- **natural join**: natürlicher Join, Gleichheitstest auf alle gleichnamigen Attribute in den Relationen, Ausgabe aller Attribute, die gleichnamigen nur jeweils einmal (nicht in allen DBS!)
- **semi join**: kein Operator in SQL, ausgedrückt mit **exists** oder **in** Konstrukte (Reduktion **natural join** auf *linke* Relation)
- **join** (auch **inner join**): Theta-Join, Theta Prädikat über Attribute
- **left, right** oder **full outer join**: äußerer Join

(Inner) Join

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

oder auch

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

Äußere Joins (links)

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

Ergebnis

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

Äußere Joins (rechts)

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

Ergebnis

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

Äußere Joins (full)

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

Ergebnis

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

Veränderung am Datenbestand: Einfügen

Einfügen von Tupeln durch **Anfrage**

insert into hören

select MatrNr, VorlNr

from Studenten, Vorlesungen

where Titel= `Logik`;

Einfügen von Tupeln durch **explizite Wertangabe**

insert into Studenten (MatrNr, Name)

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

Veränderung am Datenbestand: Einfügen

Einfügen von Tupeln aus **Datei**

Datenbankspezifische Dienstprogramme, z.B. DB2:

- **Import:**

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

Analog: EXPORT TO studis.tbl OF DEL
SELECT * FROM Studenten;

- **Load:**

High-Performance Alternative zu Import

Oracle: Load, Datapump, ...

Veränderung am Datenbestand: Löschen, Verändern

delete Studenten

where Semester > 13;

Achtung: **delete** Studenten löscht gesamten **Inhalt** der Relation

update Studenten

set Semester = Semester + 1;

Zweistufiges Vorgehen bei Änderungen

1. die Kandidaten für die Änderung werden ermittelt und "markiert"
2. die Änderung wird an den in Schritt 1. ermittelten Kandidaten durchgeführt

Anderenfalls könnte die Änderungsoperation von der Reihenfolge der Tupel abhängen, wie folgendes Beispiel zeigt:

delete from voraussetzen

where Vorgänger **in** (**select** Nachfolger
from voraussetzen);

Beispiel

voraussetzen	
Vorgänger	Nachfolger
5001	5041
5001	5043
5001	5049
5041	5216
5043	5052
5041	5052
5052	5229

Ohne einen Markierungsschritt hängt das Ergebnis dieser Anfrage von der Reihenfolge der Tupel in der Relation ab. Eine Abarbeitung in der Reihenfolge der Beispielausprägung würde das letzte Tupel (5052, 5229) fälschlicherweise erhalten, da vorher bereits alle Tupel mit 5052 als *Nachfolger* entfernt wurden.

Veränderungen am Schema

- **drop table** <Tabellenname>

Weitere datenbankspezifisch, z.B. Oracle:

- **alter table** <Tabellenname>
 - **modify** <Attributname> <Datentyp>
 - **add** <Attributname> <Datentyp>
 - **drop column** <Attributname>
 - ...

Sichten ...

- gehören zur DDL:
create view <viewname> **as** <select-statement>
- oft verwendet, um Anfragen übersichtlicher zu gestalten
- stellen eine Art "virtuelle Relation" dar
- zeigen einen Ausschnitt aus der Datenbank
- Vorteile
 - vereinfachen den Zugriff für bestimmte Benutzergruppen
 - können eingesetzt werden, um den Zugriff auf die Daten einzuschränken
- Nachteil
 - nicht auf allen Sichten können Änderungsoperationen ausgeführt werden

Vereinfachung

Komplexe Anfrage: Finde die Namen aller Professoren, die Vorlesungen halten, die mehr als der Durchschnitt an Credits wert sind, und die mehr als drei Assistenten beschäftigen.

- nicht alles gleich auf einmal machen → kleinere übersichtlichere Teile
- diese Teile werden mit Hilfe von Sichten realisiert

Vereinfachung

1. Finde alle Vorlesungen mit überdurchschnittlich viel Credits:

```
create view ÜberSchnittCredit as  
select Nr, ProfPersNr  
from Vorlesung  
where Credits >  
  (select avg (Credits)  
   from Vorlesung);
```

Vereinfachung

2. Finde alle Professoren mit mehr als drei Assistenten:

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

Vereinfachung

- alles zusammensetzen
- Sichten können wie eine herkömmliche Relation angesprochen werden

```
select Name
from Professor
where PersNr in
  (select PersNr
   from ÜberSchnittCredit) and
  PersNr in
  (select Boss
   from VieleAssistenten);
```

Sichten ...

für den Datenschutz

```
create view prüfenSicht as  
  select MatrNr, VorINr, PersNr  
  from prüfen
```

Sichten ...

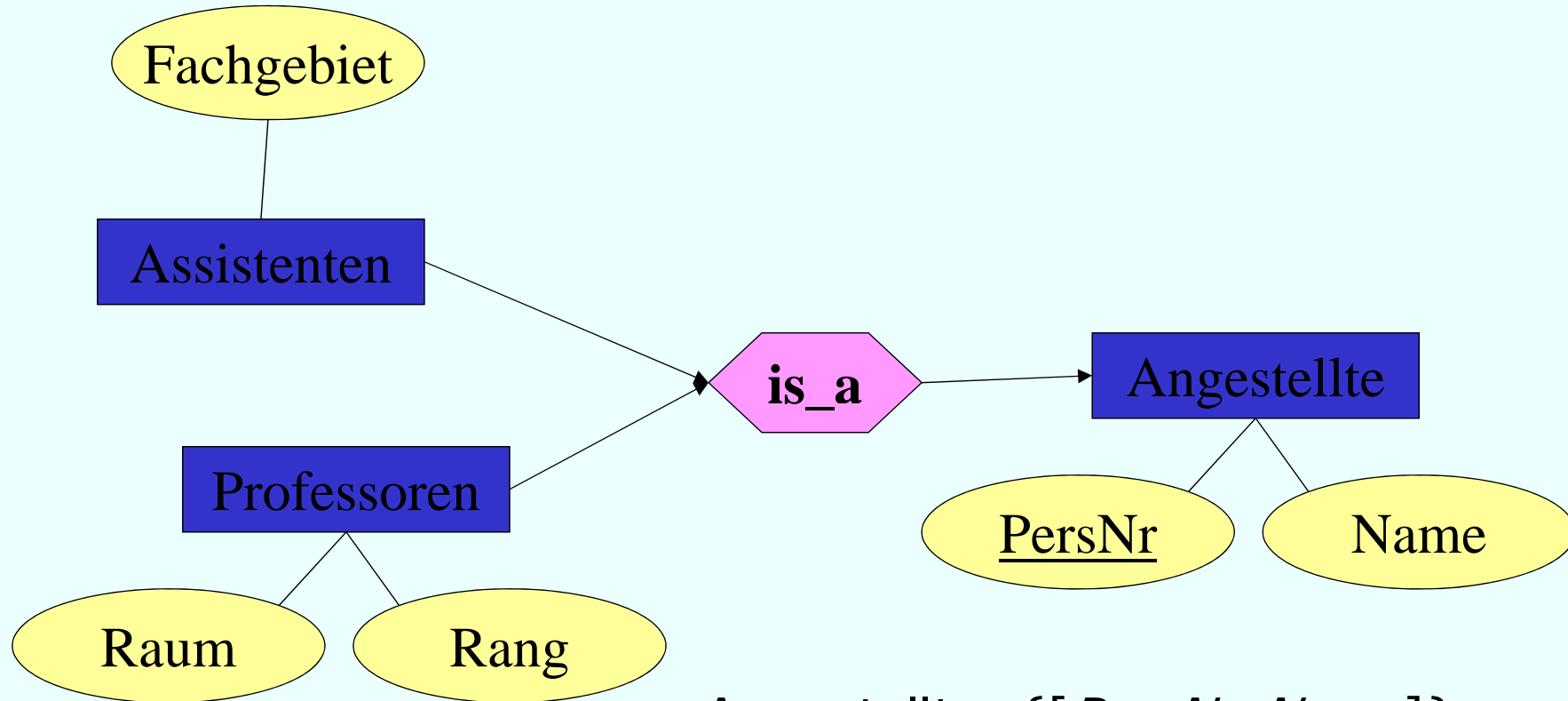
für den Datenschutz

```
create view prüfenSicht as
  select MatrNr, VorINr, PersNr
  from prüfen
```

für Statistik

```
create view PruefGuete(Name, GueteGrad) as
  (select prof.Name, avg(pruef.Note)
   from Professoren prof join pruefen pruef on
     prof.PersNr = pruef.PersNr
   group by prof.Name, prof.PersNr
   having count(*) > 50)
```

Relationale Modellierung der Generalisierung



Angestellte: $\{[\underline{PersNr}, Name]\}$

ProfDaten: $\{[\underline{PersNr}, Rang, Raum]\}$

AssiDaten: $\{[\underline{PersNr}, Fachgebiet]\}$

Tabellendefinitionen

```
create table Angestellte
```

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

```
create table ProfDaten
```

```
  (PersNr  integer not null,  
   Rang    character(2),  
   Raum    integer);
```

```
create table AssiDaten
```

```
  (PersNr    integer not null,  
   Fachgebiet varchar(30) );
```

Sichten zur Modellierung von Generalisierung

```
create view Professoren as
```

```
select *
```

```
from Angestellte a, ProfDaten d
```

```
where a.PersNr=d.PersNr;
```

```
create view Assistenten as
```

```
select *
```

```
from Angestellte a, AssiDaten d
```

```
where a.PersNr=d.PersNr;
```

→ Untertypen als Sicht

Tabellendefinitionen

create table Professoren

```
(PersNr integer not null,  
Name          varchar (30) not null,  
Rang          character (2),  
Raum          integer);
```

create table Assistenten

```
(PersNr integer not null,  
Name          varchar (30) not null,  
Fachgebiet    varchar (30) );
```

create table AndereAngestellte

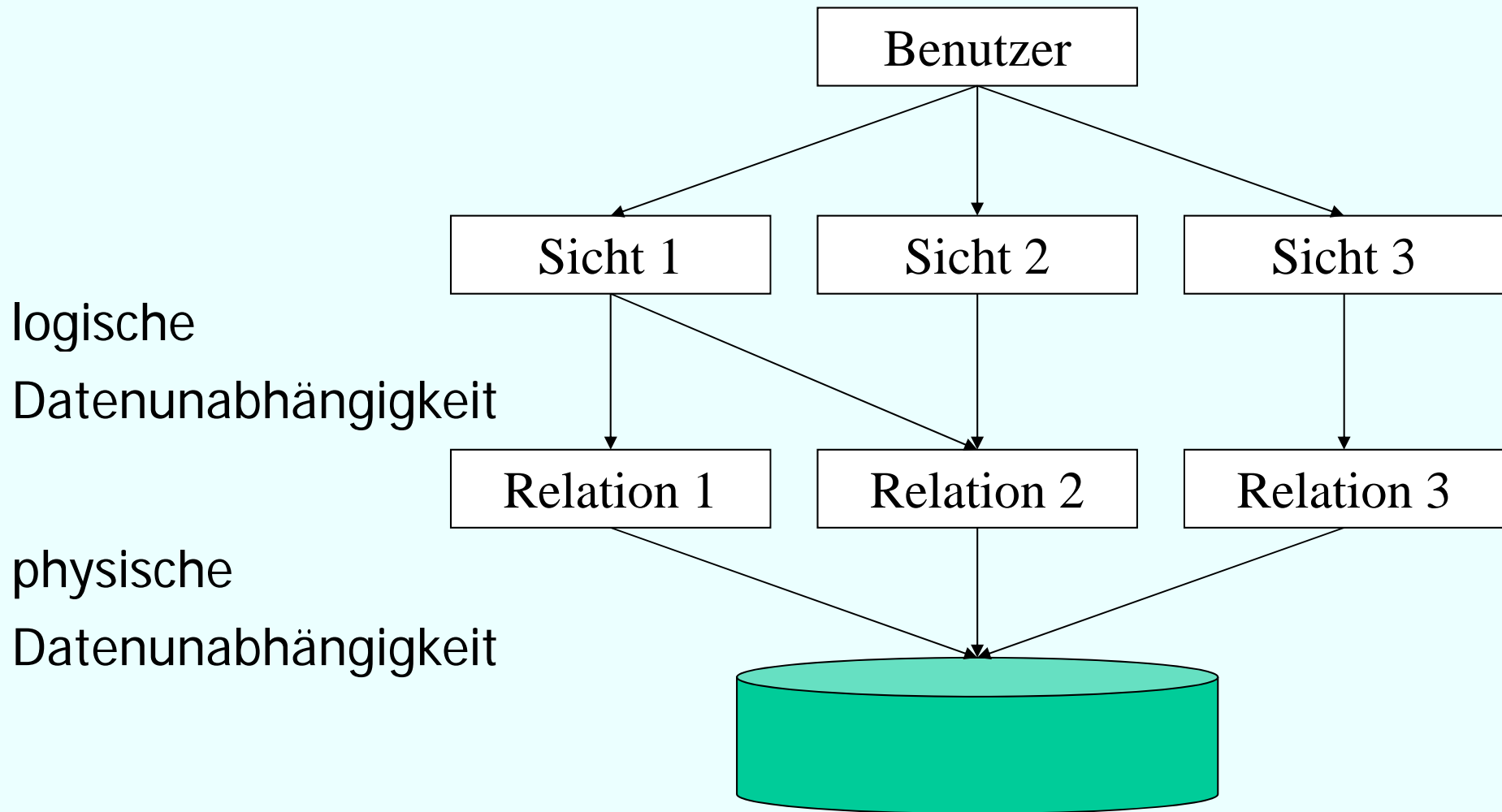
```
(PersNr integer not null,  
Name          varchar (30) not null);
```

Sichten zur Modellierung von Generalisierung

```
create view Angestellte as
    (select PersNr, Name
     from Professoren)
    union
    (select PersNr, Name
     from Assistenten)
    union
    (select *
     from AndereAngestellte);
```

→ **Obertyp als Sicht**

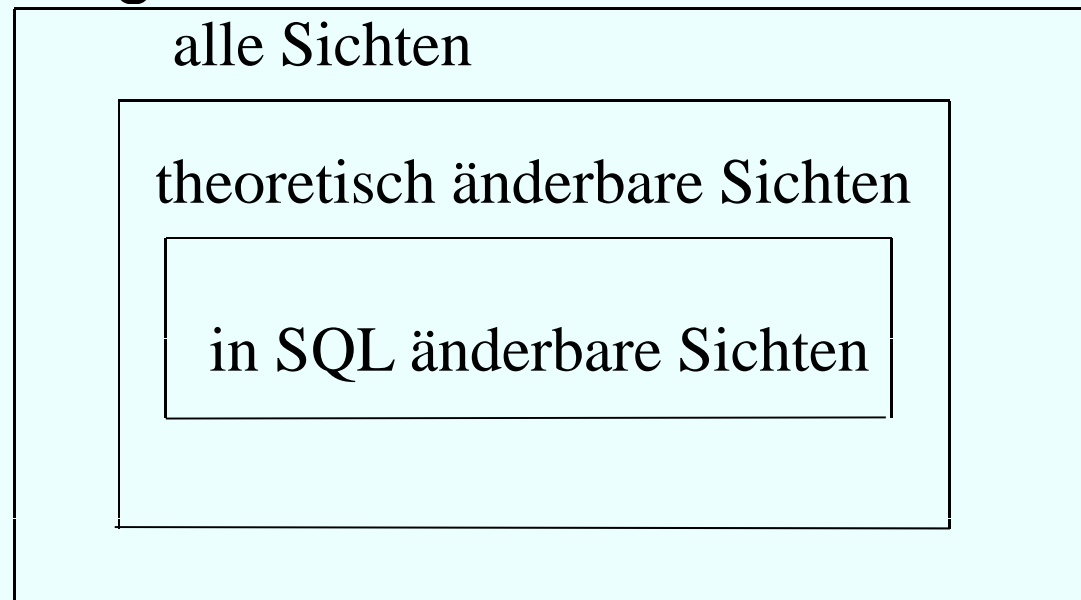
Sichten zur Gewährleistung von Datenunabhängigkeit



Änderbarkeit von Sichten

in SQL

- nur eine Basisrelation
- Schlüssel muss vorhanden sein
- keine Aggregatfunktionen, Gruppierung und Duplikateliminierung



Sichten

Lebensdauer, Gültigkeit

Löschen: **DROP VIEW *view-name***

Ungültige (inoperative) Views:

- Basisrelation wird gelöscht

- Rechteverlust des View-Erstellers

- View-Definition bleibt erhalten (ungültig markiert, kann durch Neudefinition reaktiviert werden)

Auswertung:

- Ersetzen der Sicht durch ihre Definition (~ Makro)

- keine** Speicherung (Materialisierung) der Sichtauswertung

System-generierte Werte

Künstlich erzeugte Werte, Surrogate, ohne Semantik, meist als Schlüssel, z.B. in DB2:

```
create table dept (  
    deptno smallint not null  
        generated always as  
        identity (start with 10,  
        increment by 1),  
    deptname varchar(50) not null);
```

Einfügen mit:

```
insert into dept values(default, 'I3');
```

Sequenzen

Erzeugt monotone Folge, z.B. in DB2:

```
create sequence SEQNAME  
start with 1  
increment by 1  
no maxvalue  
no cycle  
cache 24;
```

Abrufen:

```
insert into dept values (nextval for  
SEQNAME, 'I3' );
```