

Referentielle Integrität

- R und S sind zwei Relationen mit den Schemata R bzw. S
- k ist Primärschlüssel von R
- Dann ist $f \in S$ ein Fremdschlüssel, wenn für alle Tupel $s \in S$ gilt:
 - $s.f$ enthält entweder nur Nullwerte oder nur Werte ungleich Null
 - Enthält $s.f$ keine Nullwerte, so existiert ein Tupel $r \in R$ mit $s.f = r.k$
- Die Einhaltung dieser Eigenschaften wird *referentielle Integrität* genannt

DDL: Fremdschlüsselbedingung

```
CREATE TABLE Tabellename (  
    Attribut_1 Datentyp_1 [NOT NULL],  
    ...  
    Attribut_n Datentyp_n [NOT NULL],  
  
    [CONSTRAINT constraintname_pk] PRIMARY KEY  
        (Attribut_i, ..., Attribut_p),  
  
    CONSTRAINT constraintname_fk FOREIGN KEY  
        (Attribut_j, ..., Attribut_l) REFERENCES  
        Elterntabellename (Attribut_t, ..., Attribut_v));
```

DDL: Beispiel Fremdschlüssel

```
CREATE TABLE Professoren
```

```
(PersNr INTEGER NOT NULL,  
Name VARCHAR(30) NOT NULL,  
Rang CHAR(2),  
Raum INTEGER,  
PRIMARY KEY (PersNr) );
```

```
CREATE TABLE Vorlesungen
```

```
(VorINr INTEGER NOT NULL,  
Titel VARCHAR(30),  
SWS INTEGER,  
gelesenVon INTEGER,  
PRIMARY KEY (VorINr),  
CONSTRAINT gelesen_fk FOREIGN KEY  
(gelesen_von) REFERENCES Professoren (PersNr) );
```

DDL: Beispiel Fremdschlüssel (Kurzform)

```
CREATE TABLE Professoren  
  (PersNr INTEGER NOT NULL PRIMARY KEY,  
   Name VARCHAR(30) NOT NULL,  
   Rang CHAR(2),  
   Raum INTEGER);
```

```
CREATE TABLE Vorlesungen  
  (VorlNr INTEGER NOT NULL PRIMARY KEY,  
   Titel VARCHAR(30),  
   SWS INTEGER,  
   gelesenVon INTEGER REFERENCES Professoren);
```

DDL: Fremdschlüssel Varianten

- Änderungen an Schlüsselattributen können automatisch propagiert werden
- set null: alle Fremdschlüsselwerte, die auf einen Schlüssel zeigen, der geändert oder gelöscht wird, werden auf NULL gesetzt
- cascade: alle Fremdschlüsselwerte, die auf einen Schlüssel zeigen, der geändert oder gelöscht wird, werden ebenfalls auf den neuen Wert geändert bzw gelöscht

DDL: Beispiel Fremdschlüssel Varianten

```
CREATE TABLE Vorlesungen
  (VorINr INTEGER NOT NULL PRIMARY KEY,
  Titel VARCHAR(30),
  SWS INTEGER,
  gelesenVon INTEGER REFERENCES Professoren
  ON DELETE SET NULL);
```

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

Die relationale Algebra

σ Selektion

π Projektion

\times Kreuzprodukt

\bowtie Join (Verbund)

ρ Umbenennung

\ltimes Semi-Join (linker)

\rtimes Semi-Join (rechter)

$\ltimes\Join$ linker äußerer Join

$\rtimes\Join$ rechter äußerer Join

Allg. Mengenoperationen:

– Differenz

\div Division

\cup Vereinigung

\cap Durchschnitt

Beispiel Mengendurchschnitt

Finde die *PersNr* aller C4-Professoren, die mindestens eine Vorlesung halten.

$$\Pi_{\text{PersNr}}(\rho_{\text{PersNr} \leftarrow \text{gelesenVon}}(\text{Vorlesungen})) \cap \Pi_{\text{PersNr}}(\sigma_{\text{Rang}=\text{C4}}(\text{Professoren}))$$

Relationaler Tupelkalkül

Eine Anfrage im Relationenkalkül hat die Form

$$\{t \mid P(t)\}$$

mit t Tupelvariable und P Prädikat

einfaches **Beispiel:**

C4-Professoren

$$\{p \mid p \in \text{Professoren} \wedge p.\text{Rang} = \text{'C4'}\}$$

Relationenkalkül: Beispiel

Studenten mit mindestens einer Vorlesung von Curie

$$\{s \mid s \in \text{Studenten} \\ \wedge \exists h \in \text{hören}(s.\text{MatrNr}=h.\text{MatrNr} \\ \wedge \exists v \in \text{Vorlesungen}(h.\text{VorlNr}=v.\text{VorlNr} \\ \wedge \exists p \in \text{Professoren}(p.\text{PersNr}=v.\text{gelesenVon} \\ \wedge p.\text{Name} = \text{'Curie'})))))\}$$

Dieselbe Anfrage in SQL belegt die Verwandtschaft

```
select s.*
from Studenten s
where exists (
  select h.*
  from hören h
  where h.MatrNr = s.MatrNr and exists (
    select *
    from Vorlesungen v
    where v.VorlNr = h.VorlNr and exists (
      select *
      from Professoren p
      where p.Name = 'Curie' and
            p.PersNr = v.gelesenVon )))
```

Relationaler Domänenkalkül

Anfrage im Domänenkalkül hat die Form:

$$\{[v1, v2, \dots, vn] \mid P(v1, \dots, vn)\}$$

mit $v1, \dots, v2$ Domänenvariablen und P Prädikat

Beispiel:

MatrNr und Namen der Prüflinge von Sokrates

$$\{[m, n] \mid \exists ([m, n, s] \in \text{Studenten} \\ \wedge \exists p, v, g ([m, p, v, g] \in \text{prüfen} \\ \wedge \exists a, r, b ([p, a, r, b] \in \text{Professoren} \\ \wedge a = \text{'Sokrates'})))]\}$$

Ausdruckskraft

Die drei Sprachen

- relationale Algebra
 - relationaler Tupelkalkül, eingeschränkt auf sichere Ausdrücke
 - relationaler Domänenkalkül, eingeschränkt auf sichere Ausdrücke
- sind gleich mächtig

$\{n \mid \neg(n \in \text{Professoren})\}$ z.B. ist nicht sicher, da das Ergebnis unendlich ist

Gerüst SQL-Anfrage

select	<Attributliste>	5
from	<Relationenliste>	1
[where	<Prädikatsliste>	2
group by	<Attributliste>	3
having	<Prädikatsliste>	4
order by	<Attributliste>	6
fetch first	<Anzahl Ergebnistupel>]	7

Einfaches Beispiel

Anfrage:

"Gib mir die gesamte Information über alle Professoren,,

Professoren

```
select *  
from Professoren
```

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

Ergebnis

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

Attribute selektieren

Anfrage:

"Gib mir die PersNr und den Namen aller Professoren,,

```
select  PersNr, Name  
from    Professoren
```

Professoren

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

Ergebnis

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

Duplikateliminierung

- Im Gegensatz zur relationalen Algebra (Mengen!) eliminiert SQL keine Duplikate
- Falls Duplikateliminierung erwünscht, muss das Schlüsselwort **distinct** benutzt werden

- Beispiel:

Anfrage: „Welche Ränge haben Professoren?“

```
select distinct Rang
```

```
from Professoren
```

Ergebnis:

Rang
C3
C4

Where Klausel: Tupel selektieren

Anfrage:

"Gib mir die PersNr und den Namen aller Professoren, die den Rang C4 haben,,

```
select  PersNr, Name  
from    Professoren  
where   Rang= 'C4';
```

Ergebnis:

PersNr	Name
2125	Sokrates
2126	Russel
2136	Curie
2137	Kant

Where Klausel: Prädikate

- Prädikate in der where-Klausel können logisch kombiniert werden mit:

AND, OR, NOT

- Als Vergleichsoperatoren können verwendet werden:

=, <, <=, >, >=, between, like

Beispiel für between

Anfrage:

"Gib mir die Namen aller Studenten, die zwischen 1987-01-01 und 1989-01-01 geboren wurden,,

```
select Name  
from Student  
where Geburtstag between 1987-01-01 and 1989-01-01;
```

Anfrage äquivalent zu:

```
select Name  
from Student  
where Geburtstag >= 1987-01-01  
          and Geburtstag <= 1989-01-01;
```

String-Vergleiche

- Stringkonstanten müssen in einfachen Anführungszeichen eingeschlossen sein

Anfrage:

"Gib mir alle Informationen über den Professor mit dem Namen Kant,"

```
select *  
from Professoren  
where Name = 'Kant';
```

Suche mit Jokern (Wildcards)

Anfrage:

"Gib mir alle Informationen über Professoren,
deren Namen mit einem K anfängt"

```
select *  
from Professoren  
where Name like 'K%';
```

Mögliche Joker:

- `_` steht für ein beliebiges Zeichen
- `%` steht für eine beliebige Zeichenkette (auch der Länge 0)

Nullwerte

- In SQL gibt es einen speziellen Wert **NULL**
- Dieser Wert existiert für alle verschiedenen Datentypen und repräsentiert Werte, die
 - *unbekannt* oder
 - *nicht verfügbar* oder
 - *nicht anwendbar* sind.
- Nullwerte können auch im Zuge der Abfrageauswertung entstehen
- Auf NULL wird mit **is NULL** geprüft:

Beispiel:

```
select *  
from Professoren  
where Raum is NULL;
```

Nullwerte cont.

- Nullwerte werden in arithmetischen Ausdrücken durchgereicht: mindestens ein Operand NULL → Ergebnis ebenfalls NULL
- manchmal sehr überraschende Anfrageergebnisse, wenn Nullwerte vorkommen, z.B.:

```
select count (*)
```

```
from Studenten
```

```
where Semester < 13 or Semester > = 13
```

- Wenn es Studenten gibt, deren Semester-Attribut den Wert NULL hat, werden diese nicht mitgezählt
- Der Grund liegt in dreiwertiger Logik unter Einbeziehung von NULL-Werten:

Auswertung bei Null-Werten

- SQL: dreiwertige Logik, mit den Werten **true**, **false** und **unknown**
- **unknown** liefern Vergleichsoperationen zurück, wenn mindestens eines ihrer Argumente NULL ist.
- In einer **where**-Bedingung werden nur Tupel weitergereicht, für die die Bedingung **true** ist. Insbesondere werden Tupel, für die die Bedingung zu **unknown** auswertet, nicht ins Ergebnis aufgenommen.
- Bei einer Gruppierung wird NULL als ein eigenständiger Wert aufgefasst und in eine eigene Gruppe eingeordnet.
- Logische Ausdrücke werden nach den folgenden Tabellen berechnet:

Dreiwertige Logik-Tabellen

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

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

Anfragen über mehrere Relationen: Kreuzprodukt

- Falls mehrere Relationen in der from-Klausel auftauchen, werden sie mit einem Kreuzprodukt verbunden
- Beispiel:
Anfrage: "Gib alle Professoren und Vorlesungen aus,,

```
select *  
from Vorlesung, Professor;
```

Ergebnis???

Anfragen über mehrere Relationen: Joins

- Kreuzprodukte machen meistens keinen Sinn, interessanter sind Joins
- Joinprädikate werden in der where-Klausel angegeben:

```
select *  
from Vorlesung, Professor  
where gelesenVon = PersNr;
```

Anfragen über mehrere Relationen: Joins cont.

Welcher Professor liest "Mäeutik"?

```
select Name, Titel  
from Professoren, Vorlesungen  
where PersNr = gelesenVon  
       and Titel = 'Mäeutik';
```

Beispiel

Professoren				Vorlesungen			
PersNr	Name	Rang	Raum	VorlNr	Titel	SWS	gelesen Von
2125	Sokrates	C4	226	5001	Grundzüge	4	2137
2126	Russel	C4	232	5041	Ethik	4	2125
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
2137	Kant	C4	7	5049	Mäeutik	2	2125
				⋮	⋮	⋮	⋮
				4630	Die 3 Kritiken	4	2137

PersN	Name	Rang	Raum	VorlNr	Titel	SWS	gelesen Von
2125	Sokrates	C4	226	5001	Grundzüge	4	2137
1225	Sokrates	C4	226	5041	Ethik	4	2125
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
2125	Sokrates	C4	226	5049	Mäeutik	2	2125
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
2126	Russel	C4	232	5001	Grundzüge	4	2137
2126	Russel	C4	232	5041	Ethik	4	2125
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
2137	Kant	C4	7	4630	Die 3 Kritiken	4	2137

↓ Auswahl

PersN r	Name	Rang	Raum	VorlNr	Titel	SWS	gelesen Von
2125	Sokrates	C4	226	5049	Mäeutik	2	2125

↓ Projektion

Name	Titel
Sokrates	Mäeutik

Namenskollision

- gleichnamige Attribute in verschiedenen Relationen müssen aufgelöst werden

Beispiel:

Welche Studenten hören welche Vorlesungen?

```
select Name, Titel  
from Studenten, hören, Vorlesungen  
where Studenten.MatrNr = hören.MatrNr and  
hören.VorINr = Vorlesungen.VorINr;
```

Namenskollision cont.

Welche Studenten hören welche Vorlesungen?

Alternativ:

```
select s.Name, v.Titel
from Studenten s, hören h, Vorlesungen v
where s. MatrNr = h. MatrNr and
      h.VorlNr = v.VorlNr
```

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

Mengenoperationen

- In SQL gibt es auch die üblichen Operationen auf Mengen:
Vereinigung, Schnitt und Differenz
- Setzen wie in der relationalen Algebra gleiches Schema der verknüpften Ausgabe-Relationen voraus

(**select** Name
 from Assistenten)

union

(**select** Name
 from Professoren);

Duplikateliminierung

- Im Gegensatz zu **select** eliminiert **union** automatisch Duplikate
- Falls Duplikate im Ergebnis erwünscht sind, muss der **union all**-Operator benutzt werden

Schnitt, Mengendifferenz

Professoren **und** Assistenten

```
select Name from Professoren
```

```
intersect
```

```
select Name from Assistenten;
```

Professoren, **aber nicht** Assistenten

```
select Name from Professoren
```

```
except
```

```
select Name from Assistenten;
```

Sortierung

- Tupel in einer Relation sind nicht (automatisch) sortiert
- Ergebnis einer Anfrage kann mit Hilfe der **order by**-Klausel sortiert werden
- Es kann aufsteigend oder absteigend sortiert werden
- Default Sortierung: aufsteigend

Beispiel

```
select *  
from Studenten  
order by Name, Semester desc;
```

Geschachtelte Anfragen

- Anfragen können in anderen Anfragen geschachtelt sein, d.h. es kann mehr als eine select-Klausel geben
- Geschachteltes select kann in der where-Klausel, in der from-Klausel und sogar in einer select-Klausel selbst auftauchen
- Im Prinzip wird in der "inneren" Anfrage ein Zwischenergebnis berechnet, das in der "äußeren" benutzt wird

Select in Where-Klausel

- Zwei verschiedene Arten von Unteranfragen: korrelierte und unkorrelierte
- unkorreliert: Unteranfrage bezieht sich nur auf "eigene" Attribute
- korreliert: Unteranfrage referenziert auch Attribute der äußeren Anfrage

Unkorrelierte Unteranfrage

Namen aller Studenten, die VorlNr 5041 hören

```
select S.Name  
from Studenten S  
where S.MatrNr in  
(select h.MatrNr  
from hoeren h  
where h.VorlNr = 5041);
```

- Unteranfrage wird einmal ausgewertet
- für jedes Tupel der äußeren Anfrage wird geprüft, ob die MatrNr im Ergebnis der Unteranfrage vorkommt

Korrelierte Unteranfrage

Finde alle Professoren, für die Assistenten mit unterschiedlichen Fachgebieten arbeiten

```
select distinct P.Name  
from Professoren P, Assistenten A  
where A.Boss = P.PersNr  
and exists  
(select *  
from Assistent B  
where B.Boss = P.PersNr and A.Fachgebiet <> B.Fachgebiet);
```

- Für jedes Tupel der äußeren Anfrage hat innere Anfrage verschiedene Werte
- das exists-Prädikat ist wahr, wenn die Unteranfrage mind. ein Tupel enthält

Existenzquantor exists

```
select P.Name  
from Professoren P  
where not exists ( select *  
                    from Vorlesungen V  
                    where V.gelesenVon = P.PersNr );
```

Existenzquantor exists

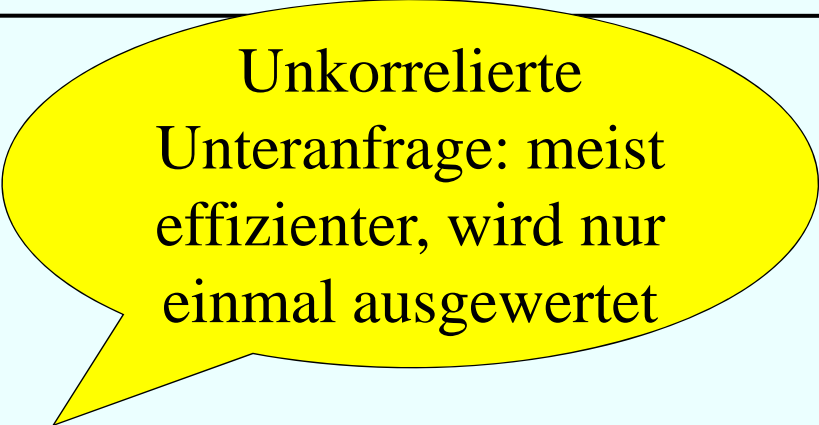
```
select P.Name  
from Professoren P  
where not exists ( select *  
                  from Vorlesungen V  
                  where V.gelesenVon = P.PersNr );
```



The diagram shows a yellow arrow pointing from the inner query (select * from Vorlesungen V where V.gelesenVon = P.PersNr) to the outer query (select P.Name from Professoren P). The arrow is labeled "Korrelation" in pink, indicating the correlation between the two queries.

Mengenvergleich

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



Unkorrelierte
Unteranfrage: meist
effizienter, wird nur
einmal ausgewertet