Datenmodellierung/Datenbanksysteme

VU 184.685/VU 184.686, WS 2022

Relationale Abfragesprachen – SQL

Johannes Fichte, Felix Winter

Institut für Logic and Computation
Technische Universität Wien

Acknowledgments

Die Folien sind eine Weiterentwicklung der Folien von Katrin Seyr und Sebastian Skritek, welche wiederum auf den zum Lehrbuch zur Verfügung gestellten Folien von A. Kemper basieren.

Der Inhalt basiert auf und behandelt Kapitel 4 des Lehrbuchs
(Kemper, Eickler: Datenbanksysteme – Eine Einführung)

(Kapitel 4.1 – 4.9, 4.11 – 4.15, 4.17)

Überblick

1. SQL einst und jetzt
2. Datenabfragesprache
3. Datendefinitionssprache
4. Datenmanipulationssprache

SQL einst und jetzt

SQL wurde auf Basis von relationaler Algebra und Relationenkalkül entwickelt

deklarative Sprache

Abarbeitung Übersetzung der Abfragen mittels Parser in relationale Algebra und Optimierung durch Anfragenoptimierer des DBMS

Relationen werden dargestellt in Form von Tabellen

SQL bietet eine standardisierte

- Datendefinitions-Sprache (DDL)
- Datenmanipulations-Sprache (DML)
- Anfrage (Query)-Sprache

SQL einst und jetzt

SQL 99: erweiterte SQL 92 um objektrelationale Konstrukte, rekursive Abfragen und Trigger

SQL 2003: bietet erweiterte Unterstützung von Nested Tables, von Merge Operationen und auf XML bezogene Eigenschaften

SQL 2006: Brücke zu XML, XQuery

SQL 2008, 2011: 2011 führt temporale Daten ein

SQL 2016: u.a. JSON, derzeit aktueller Standard

System R (IBM): erster DBMS Prototyp, Sprache: Structured English Query Language $\Rightarrow$ SQL

DBMS: Oracle (Oracle Corporation), Informix (Informix), SQL-Server (Microsoft), DB2 (IBM), PostgreSQL, MySQL

Datenabfragesprache

1. Einfache SQL-Anfragen
2. Anfragen über mehrere Relationen
3. Mengenoperationen
4. Aggregatfunktionen
5. Aggregate und Gruppierung
6. Geschachtelte Anfragen
7. Existenziell quantifizierte Anfragen
8. Allquantifizierte Anfragen
9. Nullwerte
10. Spezielle Sprachkonstrukte

Einfache SQL Anfragen

Example

```
select *
from ProfessorIn;
```

ProfessorIn

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

Einfache SQL Anfragen

Example

```
select Name, Raum
from ProfessorIn;
```

$\pi_{\text{Name,Raum}}(\text{ProfessorIn})$

ProfessorIn	
Name	Raum
Sokrates	226
Russel	232
Kopernikus	310
Popper	52
Augustinus	309
Curie	36
Kant	7

Einfache SQL Anfragen – Duplikate

Beispiel (Duplikatelimination)

Geben Sie alle Rangbezeichnungen für ProfessorInnen aus (rel. Algebra)

$$\pi_{\text{Rang}}(\text{ProfessorIn})$$

Ergebnis
Rang
C4
C3

Geben Sie alle Rangbezeichnungen für ProfessorInnen aus (SQL)

```
select Rang
from ProfessorIn;
```

Ergebnis
Rang
C4
C4
C4
C4
C3

Einfache SQL Anfragen – Duplikate

Beispiel (Duplikatelimination)

Geben Sie alle Rangbezeichnungen für ProfessorInnen ohne Duplikate aus

```
select distinct Rang
from ProfessorIn;
```

Ergebnis
Rang
C4
C3

`distinct` . . . eliminiert Duplikate aus dem Ergebnis

Einfache SQL Anfragen

Example

Finde die Raumnummer von Professor Popper

```
select Name, Raum
from ProfessorIn
where Name='Popper';
```

$$\pi_{\text{Name,Raum}}\left(\sigma_{\text{Name}='\text{Popper}'}(\text{ProfessorIn})\right)$$

ProfessorIn	
Name	Raum
Popper	52

Einfache SQL Anfragen

Struktur einer einfachen Anfrage:

```
select Attribute
from Tabelle
where Bedingung;
```

Attribute: Liste jener Attribute, die ausgegeben werden

Tabelle: Name der Tabelle, die durchsucht wird

Bedingung: Kriterium, das jedes ausgegebene Tupel erfüllen muss

Einfache SQL Anfragen

Beispiel

Geben Sie Personalnummer und Name aller C4 ProfessorInnen an:

ProfessorIn

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

Ergebnis

PersNr	Name
2125	Sokrates
2126	Russel
2136	Curie
2137	Kant

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

Einfache SQL Anfragen – Sortierung

Beispiel (Sortierung)

Geben Sie die Personalnummer, Name und Rang aller ProfessorInnen an. Sortieren Sie das Ergebnis absteigend nach dem Rang und aufsteigend nach dem Namen.

```
select PersNr, Name, Rang
from   ProfessorIn
order by Rang desc, Name asc;
```

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

Einfache SQL Anfragen – Sortierung

Beispiel (Sortierung)

Geben Sie die Personalnummer und den Namen aller ProfessorInnen an. Sortieren Sie das Ergebnis absteigend nach dem Rang und aufsteigend nach dem Namen.

```
select PersNr, Name
from   ProfessorIn
order by Rang desc, Name asc;
```

Ergebnis

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

Einfache SQL Anfragen

Beispiel

Welche ProfessorInnen lesen die LVA Mäeutik?
Information aus Tabelle: `ProfessorIn, Vorlesung`

```
ProfessorIn(PersNr, Name, Rang, Raum)
Vorlesung(VorlNr, Titel, SWS, gelesenVon)
```

```
select Name, Titel
from ProfessorIn, Vorlesung
where PersNr = gelesenVon and
      Titel = 'Maeutik';
```

Anfragen über mehrere Relationen

Anfragen über mehrere Relationen

Struktur einer Abfrage über mehrere Tabellen:

```
select Attribute
from Tabelle1, Tabelle2, ..., TabelleN
where Bedingungen;
```

Bedingungen: greifen auf die Attribute der verschiedenen Tabellen zu

Auswertung:

1. Bilde Kreuzprodukt der `from` Tabellen
2. Überprüfe für jede Zeile die `where` Bedingungen
3. Projiziere auf `select` Attribute

Anfragen über mehrere Relationen

Beispiel

Welche ProfessorInnen lesen die LVA Mäeutik?
Information aus Tabelle: `ProfessorIn, Vorlesung`

```
ProfessorIn(PersNr, Name, Rang, Raum)
Vorlesung(VorlNr, Titel, SWS, gelesenVon)
```

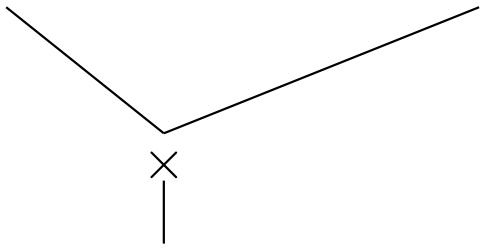
```
select Name, Titel
from ProfessorIn, Vorlesung
where PersNr = gelesenVon and
      Titel = 'Maeutik';
```

$$\pi_{\text{Name,Titel}}\left(\sigma_{\text{PersNr}=\text{gelesenVon}\wedge\text{Titel}=\text{'Maeutik'}}(\text{ProfessorIn} \times \text{Vorlesung})\right)$$

Anfragen über mehrere Relationen – Abarbeitung

1. Bilde Kreuzprodukt der `from` Tabellen

ProfessorIn			
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



Vorlesung			
VorlNr	Titel	SWS	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 drei Kritiken	4	2137

Anfragen über mehrere Relationen – Abarbeitung

$ProfessorIn \times Vorlesung$							
PersNr	Name	Rang	Raum	VorlNr	Titel	SWS	VPersNr
2125	Sokrates	C4	226	5001	Grundzüge	4	2137
2125	Sokrates	C4	226	5041	Ethik	4	2125
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
2125	Sokrates	C4	226	5049	Mäeutik	2	2125
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
2137	Kant	C4	7	4630	Die drei Kritiken	4	2137

2. Überprüfe für jede Zeile die `where` Bedingungen

$\sigma_{PersNr=VPersNr \wedge Titel='Mäeutik'}(ProfessorIn \times Vorlesung)$							
PersNr	Name	Rang	Raum	VorlNr	Titel	SWS	VPersNr
2125	Sokrates	C4	226	5049	Mäeutik	2	2125

Anfragen über mehrere Relationen – Abarbeitung

$\sigma_{PersNr=VPersNr \wedge Titel='Mäeutik'}(ProfessorIn \times Vorlesung)$

PersNr	Name	Rang	Raum	VorlNr	Titel	SWS	VPersNr
2125	Sokrates	C4	226	5049	Mäeutik	2	2125

3. Projiziere auf `select` Attribute

$\pi_{Name, Titel}(\dots)$

Name	Titel
Sokrates	Mäeutik

Anfragen über mehrere Relationen

Beispiel

Geben Sie Name und Matrikelnummer der Studierenden und den Titel der von ihnen gehörten Vorlesungen aus.

```
StudentIn(MatrNr, Name, Sem)
hoeren(MatrNr, VorlNr)
Vorlesung(VorlNr, Titel, SWS, gelesenVon)
```

```
select StudentIn.MatrNr, Name, Titel
from StudentIn, hoeren, Vorlesung
where StudentIn.MatrNr = hoeren.MatrNr and
      hoeren.VorlNr = Vorlesung.VorlNr;
```

Platzhalter für Tabellen und Umbenennung von Attributen

Beispiel

Selbe Anfrage bei der Vergabe von Platzhaltern für Tabellennamen:

```
select s.MatrNr, s.Name, v.Titel
from StudentIn s, hoeren h, Vorlesung v
where s.MatrNr = h.MatrNr and
      h.VorlNr = v.VorlNr;
```

Beispiel

Finde die Vorlesungsnummern der Vorlesungen, die indirekte Vorgänger 2. Stufe der VO 5216 sind (= Vorgänger der Vorgänger von 5216)

```
voraussetzen(VorgNr, NachfNr)
```

```
select v1.VorgNr as Vorg_Stufe_2
from voraussetzen v1, voraussetzen v2
where v1.NachfNr=v2.VorgNr and v2.NachfNr=5216;
```

FAKULTÄT FÜR INFORMATIK
Faculty of Informatics

Anfragen über mehrere Relationen

Struktur einer Abfrage über mehrere Tabellen:

```
select  Atrribut1 [ as ] a1,
        Attribut2 [ as ] a2,
        ...,
        AttributM [ as ] aM
from  Tabelle1 [ as ] t1,
      Tabelle2 [ as ] t2,
      ...,
      TabelleN [ as ] tN
where  Bedingungen;
```

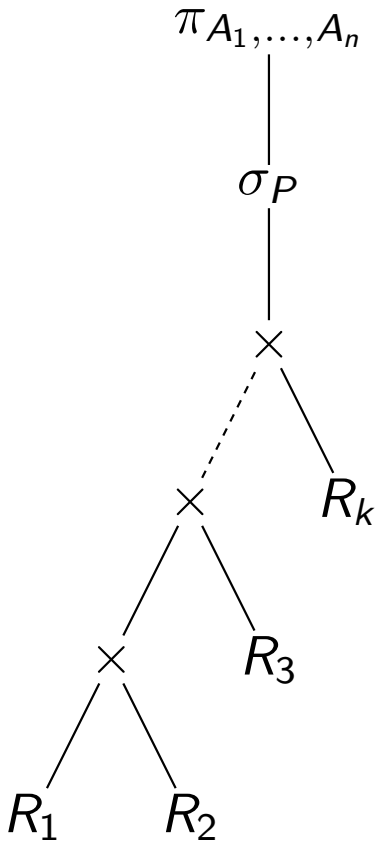
Übersetzung in relationale Algebra

Allgemeine Form einer (ungeschachteteln) SQL-Anfrage:

```
select A_1, ..., A_n
from R_1, ..., R_k
where P;
```

wird zu:

$$\pi_{A_1,\ldots,A_n}\sigma_P(R_1 \times \cdots \times R_k)$$



Mengenoperationen

Mengenoperationen

Anfragen mit typkompatiblen Ausgabeattributen können mittels Mengenoperationen verknüpft werden.

Ohne Duplikate: `union`, `intersect`, `except` (`minus`)

Mit Duplikaten: `union all`, `intersect all`, `except all`

Beispiel

Suchen Sie den Namen aller Assistenten oder ProfessorInnen

```
( select Name from AssistentIn )
union
( select Name from ProfessorIn );
```

Aggregatfunktionen

Aggregatfunktionen

Aggregatfunktionen sind Operationen, die nicht auf einzelnen Tupeln, sondern auf einer Menge von Tupeln arbeiten:

- `avg()`, `max()`, `min()`, `sum()` berechnen den Wert einer Menge von Tupeln,
- `count()` zählt die Anzahl der Tupel in der Menge.

Aggregatfunktionen

Beispiel

Geben Sie die durchschnittliche/maximale/minimale Inskriptionsdauer aller Studierenden an; geben Sie an, wieviele Studierende es gibt.

```
select avg(Semester)
from StudentIn;

select max(Semester)
from StudentIn;

select min(Semester)
from StudentIn;

select count(MatrNr)
from StudentIn;
```

StudentIn

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

Aggregatfunktionen

Beispiel

Was ist die Summe der von C4 ProfessorInnen gehaltenen Vorlesungsstunden?

```
select sum (SWS)
from   Vorlesung, ProfessorIn
where  gelesenVon = PersNr and Rang = 'C4';
```

Vorlesung

VorlNr	Titel	SWS	gelesenVon
5001	Grundzüge	4	2137
5041	Ethik	4	2125
5043	Erkenntnistheorie	3	2126
5049	Mäeutik	2	2125
4052	Logik	4	2125
5259	Der Wiener Kreis	2	2133
5216	Bioethik	2	2126
⋮	⋮	⋮	⋮

ProfessorIn

PersNr	Name	Rang
2125	Sokrates	C4
2126	Russel	C4
2133	Popper	C3
⋮	⋮	⋮

Aggregate und Gruppierung

Aggregate und Gruppierung

Problem: Wollen nicht die Summe aller Vorlesungsstunden absolut berechnen, sondern zu allen Vortragenden die Summe der von ihr/ihm gehaltenen Stunden.

Lösung: Bilde zu jeder/jedem Vortragenden eine Gruppe (mittels der `group by` Klausel) und summiere nur innerhalb dieser Gruppe.

Allgemein: Alle Zeilen einer Tabelle, die auf den Attributen der `group by` Klausel den selben Wert annehmen, werden zu einer Gruppe zusammengefasst und die Aggregatfunktionen werden bezüglich der Gruppe ausgewertet.

Bedingungen an die aggregierten Werte werden in der having Klausel angeführt.

Bsp: Aggregate und Gruppierung

Beispiele

Geben Sie zu jeder C4 Professorin die Summe der von ihr gehaltenen Vorlesungsstunden an.

```
ProfessorIn(PersNr, Name, Rang, Raum)
Vorlesung(VorlNr, Titel, SWS, gelesenVon)
```

```
select gelesenVon, sum (SWS)
from Vorlesung, ProfessorIn
where gelesenVon = PersNr and Rang = 'C4'
group by gelesenVon;
```

Bsp: Aggregate und Gruppierung

Beispiele

Geben Sie zu jedem C4 Professor, der vor allem lange Vorlesungen (Durchschnitt größer gleich 3) hält, die Summe der von ihr/ihm gehaltenen Vorlesungsstunden an.

```
ProfessorIn(PersNr, Name, Rang, Raum)
Vorlesung(VorlNr, Titel, SWS, gelesenVon)
```

```
select gelesenVon, Name, sum (SWS)
from Vorlesung, ProfessorIn
where gelesenVon = PersNr and Rang = 'C4'
group by gelesenVon, Name
having avg (SWS) >= 3;
```

Aggregate und Gruppierung – Abarbeitung

```
from Vorlesung, ProfessorIn
```

ProfessorIn × *Vorlesung*

PersNr	Name	Rang	Raum	VorlNr	Titel	SWS	gelesenVon
2125	Sokrates	C4	226	5001	Grundzüge	4	2137
2125	Sokrates	C4	226	5041	Ethik	4	2125
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
2125	Sokrates	C4	226	5049	Mäeutik	2	2125
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
2133	Popper	C3	52	5001	Grundzüge	4	2137
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
2137	Kant	C4	7	4630	Die drei Kritiken	4	2137

| where-Bedingung

Aggregate und Gruppierung

where-Bedingung

```
where gelesenVon = PersNr and Rang = 'C4'
```

PersNr	Name	Rang	Raum	VorlNr	Titel	SWS	gelesenVon
2137	Kant	C4	7	5001	Grundzüge	4	2137
2125	Sokrates	C4	226	5041	Ethik	4	2125
2126	Russel	C4	232	5043	Erkenntnistheorie	3	2126
2125	Sokrates	C4	226	5049	Mäeutik	2	2125
2125	Sokrates	C4	226	4052	Logik	4	2125
2126	Russel	C4	232	5052	Wissenschaftstheorie	3	2126
2126	Russel	C4	232	5216	Bioethik	2	2126
2137	Kant	C4	7	4630	Die drei Kritiken	4	2137

Gruppierung

Aggregate und Gruppierung

Gruppierung

```
group by gelesenVon, Name
```

PersNr	Name	Rang	Raum	VorlNr	Titel	SWS	gelesenVon
2125	Sokrates	C4	226	5041	Ethik	4	2125
2125	Sokrates	C4	226	5049	Mäeutik	2	2125
2125	Sokrates	C4	226	4052	Logik	4	2125
2126	Russel	C4	232	5043	Erkenntnistheorie	3	2126
2126	Russel	C4	232	5052	Wissenschaftstheorie	3	2126
2126	Russel	C4	232	5216	Bioethik	2	2126
2137	Kant	C4	7	5001	Grundzüge	4	2137
2137	Kant	C4	7	4630	Die drei Kritiken	4	2137

having-Bedingung

Aggregate und Gruppierung

having-Bedingung

```
having avg(SWS)>= 3
```

PersNr	Name	Rang	Raum	VorlNr	Titel	SWS	gelesenVon
2125	Sokrates	C4	226	5041	Ethik	4	2125
2125	Sokrates	C4	226	5049	Mäeutik	2	2125
2125	Sokrates	C4	226	4052	Logik	4	2125
2137	Kant	C4	7	5001	Grundzüge	4	2137
2137	Kant	C4	7	4630	Die drei Kritiken	4	2137

Aggregation (sum) und Projektion

```
select gelesenVon, Name, sum(SWS)
```

gelesenVon	Name	sum(SWS)
2125	Sokrates	10
2137	Kant	8

Aggregate und Gruppierung

Achtung: SQL erzeugt pro Gruppe ein Ergebnistupel

$\Rightarrow$ Alle in der `select` Klausel aufgeführten Attribute- außer den aggregierten - müssen auch in der `group by` Klausel aufgeführt werden.
So wird sichergestellt, dass die ausgegebenen Attribute sich nicht innerhalb der Gruppe ändern.

```
select gelesenVon, Name, sum (SWS)
........
group by gelesenVon, Name;
```

Bsp: Aggregate und Gruppierung

Beispiel

Geben Sie die Namen der Studierenden aus, die am längsten studieren.

```
StudentIn(MatrNr, Name, Semester)
```

```
select Name, max(Semester)
from StudentIn;
```

⇒ ORA-00937: not a single-group group function

⇒ ERROR: column ‘‘studenten.name’’ must appear in the GROUP BY clause or be used in an aggregate function

Bsp: Aggregate und Gruppierung

Beispiel (2. Versuch)

Geben Sie die Namen der Studierenden aus, die am längsten studieren.

```
StudentIn(MatrNr, Name, Semester)
```

```
select Name, max(Semester)
from StudentIn
group by Name;
```

$\Rightarrow$ Alle Tupel ?!?

Lösung: Geschachtelte Anfrage

Geschachtelte Anfragen

Geschachtelte Anfragen

Es gibt vielfältige Möglichkeiten `select` Anweisungen zu verknüpfen. Folgende Einteilung ist abhängig vom Ergebnis der Unteranfrage (ein Wert oder eine Relation)

- Ergebnis der Unteranfrage besteht aus einem Tupel mit einem Attribut (= ein Wert):
  - Unteranfrage anstelle eines skalaren Wertes in `select` bzw. `where` Klausel
- Ergebnis der Unterabfrage eine Relation
  - Unteranfrage in `from` Klausel
  - Mengenvergleiche: `in`, `not in`, `any`, `all`
  - Verknüpfung über Quantoren: `exists`

Bei geschachtelten Abfragen ist für die Laufzeit ausschlaggebend, ob es sich um korrelierte oder unkorrelierte Abfragen handelt

Geschachtelte Anfragen – Skalare Werte als Ergebnis

Unteranfrage in der `where` Klausel

Beispiel

Geben Sie die Namen der Studierenden aus, die am längsten studieren.

```
StudentIn(MatrNr, Name, Semester)
```

```
select Name
from StudentIn
where Semester = (select max (Semester)
                  from StudentIn );
```

Geschachtelte Anfragen – Skalare Werte als Ergebnis

Unteranfrage in der `select` Klausel

Beispiel

Geben Sie zu jedem Vortragenden seine Lehrbelastung aus.

```
ProfessorIn(PersNr, Name, Rang, Raum)
Vorlesung(VorlNr, Titel, SWS, gelesenVon)
```

```
select p.Name, (select sum (SWS)
                from Vorlesung
                where gelesenVon=p.PersNr)
from ProfessorIn p;
```

Achtung: Die Unteranfrage ist korreliert (= sie greift auf Attribute der umschließenden Anfrage zu). Für jedes Ergebnistupel wird die Unteranfrage einmal ausgeführt.

Geschachtelte Anfragen – Skalare Werte als Ergebnis

Beispiel (Entschachtelung mittels group by)

Geben Sie zu jeder Vortragenden ihre Lehrbelastung aus.

```
ProfessorIn(PersNr, Name, Rang, Raum)
Vorlesung(VorlNr, Titel, SWS, gelesenVon)
```

```
select p.PersNr, p.Name, sum (SWS)
from ProfessorIn p, Vorlesung v
where v.gelesenVon=p.PersNr
group by p.PersNr, p.Name
```

Geschachtelte Anfragen – Tabellen als Ergebnis

Unteranfrage in der `from` Klausel

Beispiel

Gesucht sind jene Studierende, die mehr als 2 Vorlesungen hören.

```
StudentIn(MatrNr, Name, Semester)
hoeren(MatrNr, VorlNr)
```

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

Geschachtelte Anfragen – Tabellen als Ergebnis

Beispiel (Entschachtelung mittels `having`)

Gesucht sind jene Studierende, die mehr als 2 Vorlesungen hören.

```
StudentIn(MatrNr, Name, Semester)
hoeren(MatrNr, VorlNr)
```

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

Geschachtelte Anfragen – Mengenvergleiche

Unteranfrage in der `where` Klausel
Mengenvergleich mit `in`/`not in`

Beispiel

Gesucht sind jene ProfessorInnen, die keine Lehrveranstaltungen halten.

```
ProfessorIn(PersNr, Name, Rang, Raum)
Vorlesung(VorlNr, Titel, SWS, gelesenVon)
```

```
select Name
from ProfessorIn
where PersNr not in (select gelesenVon
                     from Vorlesung);
```

Geschachtelte Anfragen – Mengenvergleiche

Mengenvergleich mit `in`/`not in`

Beispiel

Gesucht sind für alle Studierende jene Vorlesungen die sie gehört und zu denen sie eine positive Prüfung abgelegt haben.

```
hoeren(MatrNr, VorlNr)
pruefen(MatrNr, VorlNr, PersNr, Note)
```

```
select MatrNr, VorlNr
from hoeren
where (MatrNr, VorlNr) in (select MatrNr, VorlNr
                           from pruefen
                           where Note < 5);
```

Geschachtelte Anfragen – Mengenvergleiche

Beispiel (Entschachtelung mittels `intersect`)

Gesucht sind für alle Studierende jene Vorlesungen die sie gehört und zu denen sie eine positive Prüfung abgelegt haben.

```
hoeren(MatrNr, VorlNr)
pruefen(MatrNr, VorlNr, PersNr, Note)
```

```
(select MatrNr, VorlNr
 from hoeren)
intersect
(select MatrNr, VorlNr
 from pruefen
 where Note < 5);
```

Geschachtelte Anfragen – Mengenvergleiche

Mengenvergleich mit `any`/`all` (Achtung: kein Allquantor, nur der Vergleich eines Wertes mit einer Menge von Werten.)

Beispiel

Suchen Sie jene Studierende, die am längsten studieren.

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

gleichbedeutend mit:

```
select Name
from StudentIn
where Semester = (select max(Semester)
                  from StudentIn);
```

Geschachtelte Anfragen – Mengenvergleiche

Beispiel

Suchen Sie jene Studierende, die nicht am längsten studieren.

```
select Name
from StudentIn
where Semester < any (select Semester
                      from StudentIn);
```

gleichbedeutend mit:

```
select Name
from StudentIn
where Semester < (select max(Semester)
                  from StudentIn);
```

Geschachtelte Anfragen – Aggregatfunktionen

Geschachtelte Aggregatfunktionen sind nicht erlaubt
$\Rightarrow$ Unteranfrage verwenden

Beispiel

Suchen Sie C4 ProfessorInnen, die die meisten Vorlesungsstunden halten.

```
ProfessorIn(PersNr, Name, Rang, Raum)
Vorlesung(VorlNr, Titel, SWS, gelesenVon)
```

```
select gelesenVon, max(sum(SWS))
from Vorlesung, ProfessorIn
where gelesenVon = PersNr and Rang = 'C4'
group by gelesenVon;
```

Geschachtelte Anfragen – Aggregatfunktionen

Geschachtelte Aggregatfunktionen sind nicht erlaubt
$\Rightarrow$ Unteranfrage verwenden

Beispiel

Suchen Sie C4 ProfessorInnen, die die meisten Vorlesungsstunden halten.

```
select gelesenVon, sum(SWS)
from Vorlesung, ProfessorIn
where gelesenVon=PersNr and Rang='C4'
group by gelesenVon
having sum(SWS) >= all(
            select sum(SWS)
            from Vorlesung, ProfessorIn
            where gelesenVon=PersNr and Rang='C4'
            group by gelesenVon);
```

Geschachtelte Anfragen – Aggregatfunktionen

Geschachtelte Aggregatfunktionen sind nicht erlaubt – auch nicht bei korrekten Abfragen

Beispiel

Suchen Sie die größte Anzahl an Vorlesungsstunden welche von einer C4 Professorin gehalten werden.

```
select max(sum(SWS))
from Vorlesung, ProfessorIn
where gelesenVon=PersNr and Rang='C4'
group by gelesenVon;
```

Existentiell quantifizierte Anfragen

Existentiell quantifizierte Anfragen

Verknüpfung mit einer Unteranfrage durch Existenzquantor `exists`

`exists` überprüft, ob die Unterabfrage Tupel enthält oder nicht.

Beispiel

Gesucht sind all jene Studierende, die älter als der jüngste Professor sind.

```
select s.*
from StudentIn s
where exists (select p.*
              from ProfessorIn p
              where p.GebDatum > s.GebDatum);
```

Korrelierte Unteranfrage (`s.GebDatum` und `StudentIn s`)!

Existentiell quantifizierte Anfragen

Eine nicht korrelierte Lösung ist:

Beispiel (Nicht korreliert)

Gesucht sind all jene Studierende, die älter als der jüngste Professor sind.

```
select s.*
from StudentIn s
where s.GebDatum < (select max(p.GebDatum)
                    from ProfessorIn p);
```

oder

```
select s.* from StudentIn s
where s.GebDatum < any (select p.GebDatum
                        from ProfessorIn p);
```

oder ...

Existentiell quantifizierte Anfragen

Beispiel

Suchen Sie all jene ProfessorInnen, die keine Lehrveranstaltungen halten.

```
ProfessorIn(PersNr, Name, Rang, Raum)
Vorlesung(VorlNr, Titel, SWS, gelesenVon)
```

```
select Name
from ProfessorIn
where not exists (select *
                  from Vorlesung
                  where gelesenVon = PersNr);
```

Korrelierte Unteranfrage (`PersNr` und `ProfessorIn`)!

Existentiell quantifizierte Anfragen

Eine nicht korrelierte Lösung ist:

```
select Name
from ProfessorIn
where PersNr not in (select gelesenVon
                     from Vorlesung);
```

oder

```
(select PersNr
 from ProfessorIn)
except
(select gelesenVon
 from Vorlesung);
```

Existentiell quantifizierte Anfragen

Beispiel

Suchen Sie jene AssistentInnen, die für eineN ProfessorIn arbeiten, der/die jünger ist, als sie selbst.

```
ProfessorIn(PersNr, Name, Rang, Raum, GebDatum)
AssistentIn(PersNr, Name, GebDatum, Boss)
```

```
select a.*
from AssistentIn a
where exists (select p.*
              from ProfessorIn p
              where a.Boss = p.PersNr and
              p.GebDatum > a.GebDatum);
```

Korrelierte Unteranfrage (`a.GebDatum` und `AssistentIn a`)!

Existentiell quantifizierte Anfragen

Beispiel

Suchen Sie jene AssistentInnen, die für eineN ProfessorIn arbeiten, der/die jünger ist, als sie selbst.

```
ProfessorIn(PersNr, Name, Rang, Raum, GebDatum)
AssistentIn(PersNr, Name, GebDatum, Boss);
```

Eine nicht korrelierte Lösung mit Join ist:

```
select a.*
from AssistentIn a, ProfessorIn p
where a.Boss = p.PersNr and
      p.GebDatum > a.GebDatum;
```

Allquantifizierte Anfragen

Allquantifizierte Anfragen

Beispiel

Welche Studierende haben alle 4 stündigen Lehrveranstaltungen gehört?

```
StudentIn(MatrNr, Name, Semester)
hoeren(MatrNr, VorlNr)
Vorlesung(VorlNr, Titel, SWS, gelesenVon)
```

$$\text{hoeren} \div \pi_{\text{VorlNr}}\left(\sigma_{\text{SWS}=4}(\text{Vorlesung})\right)$$

SQL stellt keinen Allquantor zur Verfügung.

Allquantifizierte Anfragen – Nicht direkt in SQL

Beispiel

Welche Studierende haben alle 4 stündigen Lehrveranstaltungen gehört?

SQL stellt keinen Allquantor zur Verfügung. Realisierung durch

1. Logische Äquivalenz (mittels 2 $\times$ `not exists`)
2. Teilmengen (mittels `exists` und `except`)
3. Abzählen (mittels `count`)
4. Division (mittels `except`)

Allquantifizierte Anfragen – 1. "Logische Umformung"

Umformulierung in äqivalente Anfrage mit Negation und Existenzquantor (Prädikatenlogik)

$$\forall x F(x) \Leftrightarrow \neg \exists x (\neg F(x))$$

Beispiel

Welche Studierende haben alle 4 stündigen Lehrveranstaltungen gehört?

$\Leftrightarrow$ Suchen Sie jene Studierende für die gilt:
haben alle 4 stündigen Lehrveranstaltungen gehört

$\Leftrightarrow$ Suchen Sie jene Studierende für die nicht gilt:
haben eine 4 stündige Lehrveranstaltung nicht gehört

$\Leftrightarrow$ Suchen Sie jene Studierende für die nicht gilt:
es gibt eine 4 stündige Lehrveranstaltung,
die der/die Studierende nicht gehört hat.

Allquantifizierte Anfragen – 1. "Logische Umformung"

SQL Umsetzung folgt nun direkt aus:

Suchen Sie jene Studierende für die nicht gilt:
es gibt eine 4 stündige Lehrveranstaltung, für die nicht gilt:
die/der Studierende hat diese Lehrveranstaltung gehört.

Beispiel

```
select s.*
from StudentIn s
where not exists
          (select *
          from Vorlesung v
          where v.SWS = 4 and
          not exists (select *
                      from hoeren h
                      where h.VorlNr = v.VorlNr
                      and s.MatrNr = h.MatrNr));
```

Allquantifizierte Anfragen – 1. "Logische Umformung"

SQL Umsetzung folgt nun direkt aus:

Suchen Sie jene Studierende für die nicht gilt:
es gibt eine 4 stündige Lehrveranstaltung, für die nicht gilt:
die/der Studierende hat diese Lehrveranstaltung gehört.

Beispiel

```
select s.*
from StudentIn s
where not exists
          (select *
          from Vorlesung v
          where v.SWS = 4 and
          s.MatrNr not in (select h.MatrNr
                           from hoeren h
                           where h.VorlNr = v.VorlNr));
```

Allquantifizierte Anfragen – 1. "Logische Umformung"

Umformung im relationalen Tupelkalkül:

Beispiel

```
StudentIn(MatrNr, Name, Sem)
hoeren(MatrNr, VorlNr)
Vorlesung(VorlNr, Titel, SWS, gelesenVon)
```

$$\{s \mid s \in StudentIn \wedge \forall v \in Vorlesung(v.SWS = 4 \rightarrow \exists h \in hören(h.VorlNr = v.VorlNr \wedge h.MatrNr = s.MatrNr))\}$$

Allquantifizierte Anfragen – 1. "Logische Umformung"

Beispiel

$$\{s \mid s \in StudentIn \wedge \forall v \in Vorlesung(v.SWS = 4 \rightarrow \exists h \in hören(h.VorlNr = v.VorlNr \wedge h.MatrNr = t.MatrNr))\}$$

$$\Leftrightarrow$$

$$\{s \mid s \in StudentIn \wedge \neg\exists v \in Vorlesung\neg(v.SWS = 4 \rightarrow \exists h \in hören(h.VorlNr = v.VorlNr \wedge h.MatrNr = s.MatrNr))\}$$

$$\Leftrightarrow$$

$$\{s \mid s \in StudentIn \wedge \neg\exists v \in Vorlesung\neg(\neg(v.SWS = 4) \vee (\exists h \in hören(h.VorlNr = v.VorlNr \wedge h.MatrNr = s.MatrNr)))\}$$

$$\Leftrightarrow$$

$$\{s \mid s \in StudentIn \wedge \neg\exists v \in Vorlesung((v.SWS = 4) \wedge (\neg\exists h \in hören(h.VorlNr = v.VorlNr \wedge h.MatrNr = s.MatrNr)))\}$$

Allquantifizierte Anfragen – 1. "Logische Umformung"

Beispiel

$$\{s \mid s \in StudentIn \wedge \neg\exists v \in Vorlesung((v.SWS = 4) \wedge (\neg\exists h \in hören(v.VorlNr = h.VorlNr \wedge s.MatrNr = h.MatrNr)))\}$$

```
select s.*
from StudentIn s
where not exists
        (select *
        from Vorlesung v
        where v.SWS = 4 and
         not exists (select *
                     from hoeren h
                     where h.VorlNr = v.VorlNr
                     and s.MatrNr = h.MatrNr));
```

Allquantifizierte Anfragen – 2. "Teilmengen"

Suchen Sie alle Studierenden M für die gilt:
Menge aller 4 stündigen LVAs $\subseteq$ Menge der von M gehörten LVAs

"$\subseteq$" kein Operator in SQL, aber

Menge aller 4 stündigen LVAs $\subseteq$ Menge der von M gehörten LVAs

$$\Leftrightarrow$$

Menge aller 4 stündigen LVAs $-$ Menge der von M gehörten LVAs $= \emptyset$

$\emptyset$ . . . es existiert keine Vorlesung in der Mengendifferenz

Allquantifizierte Anfragen – 2. "Teilmengen"

Suchen Sie alle Studierende *M* für die nicht gilt:
Es gibt eine LVA in der Differenz
Menge aller 4 stündigen LVAs – Menge der von *M* gehörten LVAs

Beispiel

```
select s.*
from StudentIn s
where not exists
          ((select VorlNr
          from Vorlesung v
          where v.SWS = 4)
          except
          (select VorlNr
          from hoeren h
          where h.MatrNr = s.MatrNr))
```

Allquantifizierte Anfragen – 3. "Abzählen"

Allquantifizierung kann immer auch durch eine `count`-Aggregation ausgedrückt werden

Beispiel

Welche Studierende haben alle 4 stündigen Lehrveranstaltungen gehört?

1. Zähle, wie viele 4 stündigen LVAs jedeR Studierende besucht hat
2. zähle wieviele 4 stündige LVAs es gibt.

```
select s.MatrNr, s.Name
from StudentIn s, hoeren h, Vorlesung v
where s.MatrNr = h.Matrnr and
      h.VorlNr = v.VorlNr and
      v.SWS = 4
group by s.MatrNr, s.Name
having count (*) = (select count (*) from Vorlesung
                    where SWS = 4);
```

Allquantifizierte Anfragen – 4. Relationale Algebra?

Beispiel

Welche Studierenden haben alle 4-stündigen Vorlesungen gehört?

hören

MatrNr	VorlNr
26120	5001
27550	5001
27550	4052
28106	5041
28106	5001
28106	4052
28106	4630
29120	5001
29120	5041
29120	5049

$\div$

$\text{hören} \div \pi_{VorlNr}\sigma_{SWS=4}(Vorlesung)$

$\pi_{VorlNr}\sigma_{SWS=4}(Vorlesung)$
VorlNr
5001
5041
4052
4630

$=$

$R \div S$
MatrNr
28106

Allquantifizierte Anfragen – 4. Relationale Algebra?

Beispiel

Welche Studierenden haben alle 4-stündigen Vorlesungen gehört?

hören	
MatrNr	VorlNr
26120	5001
27550	5001
27550	4052
28106	5041
28106	5001
28106	4052
28106	4630
29120	5001
29120	5041
29120	5049

Suche jene Studierende x, so dass

$(x, 5001) \in$ hören, $(x, 5041) \in$ hören,
$(x, 4052) \in$ hören, $(x, 4630) \in$ hören

1. Finde alle Studierende so dass dies nicht gilt.
   1. "Maximum:" Alle Studierenden haben alle 4 stündigen LVAs gehört.
   2. Welche dieser Paare sind nicht in hören?
   3. Welche Studierende kommen in mind. einem so einem Paar vor?
2. Alle "anderen" Studierenden sind gesucht.

Allquantifizierte Anfragen – 4. Relationale Algebra?

Die Division kann durch die primitiven Operatoren ausgedrückt werden

1. Bilde alle Paare von Studierenden und 4 stündigen LVAs
   Inhalt der DB wenn alle Studierende alle 4 stündigen LVAs absolviert hätten $\rightarrow$ suche Abweichungen
2. Entferne davon alle Paare (*Studentin*, *LVA*) an LVAs die eine Studierende besucht hat (d.h. der Inhalt von `hoeren`)
   Diese Paare aus Studierenden und 4 stündigen LVAs "fehlen" in der DB
3. Aus den übrig gebliebenen Paaren, sammle die Studierenden
   Studierende die mindestens eine 4 stündige LVA nicht besucht haben
4. Ziehe diese Studierende von der Menge aller Studierenden ab
   Studierende die alle 4 stündigen LVAs besucht haben

$$R \div S = \pi_{\mathcal{R}-\mathcal{S}}(R) - \pi_{\mathcal{R}-\mathcal{S}}((\pi_{\mathcal{R}-\mathcal{S}}(R) \times S) - R)$$

Allquantifizierte Anfragen – 4. Relationale Algebra?

1. Bilde alle Paare von Studierenden und 4 stündigen LVAs
2. Entferne davon alle Paare (*Student*, *LVA*) an LVAs die ein Student besucht hat (d.h. der Inhalt von `hoeren`)
3. Aus den übrig gebliebenen Paaren, sammle die Studierenden
4. Ziehe diese Studierende von der Menge aller Studierenden ab

Beispiel

```
(select sja.MatrNr from StudentIn sja)
except
(select snein.MatrNr
 from ((select s.MatrNr, v.VorlNr
        from StudentIn s, Vorlesung v
        where v.SWS = 4)
       except
       (select * from hoeren)) snein);
```

Nullwerte

Nullwerte

Beispiel

StudentIn

MatrNr	Name	Semester
24002	Xenokrates	18
25403	Jonas	12
26120	Fichte	10
26830	Aristoxenos	8
27550	Schopenhauer	6
28106	Carnap	3

Finde die Anzahl der Studierenden.

Nullwerte

Beispiel

Finde die Anzahl der Studierenden:

```
select count (*) from StudentIn;                    (6)

select count (MatrNr) from StudentIn;               (6)

select count (Semester) from StudentIn;             (6)

select count (*) from StudentIn
where Semester < 13 or Semester >= 13;              (6)
```

Nullwerte

Nullwerte entstehen

- wenn kein Wert in der Datenbank vorhanden ist,
- im Zuge der Anfrageauswertung (Bsp. äußere Joins)

Beispiel

StudentIn

MatrNr	Name	Semester
24002	Xenokrates	18
25403	Jonas	12
26120	Fichte	NULL
26830	Aristoxenos	8
27550	Schopenhauer	6
28106	Carnap	3

Finde die Anzahl der Studierenden.

Nullwerte

Beispiel

Finde die Anzahl der Studierenden:

```
select count (*) from StudentIn;                    (6)

select count (MatrNr) from StudentIn;               (6)

select count (Semester) from StudentIn;             (5)

select count (*) from StudentIn
where Semester < 13 or Semester >= 13;              (5)
```

Einschub: `count` revisited

Verhalten von `count()`:

- `count(*)`:
  Zählt alle Tupel (inkl. Duplikate)
- `count(attribute)`:
  Zählt die Anzahl der von NULL verschiedenen Werte in der angegebenen Spalte (inkl. Duplikate)
- `count(distinct attribute)`:
  Zählt die Anzahl der verschiedenen von NULL verschiedenen Werte in der angegebenen Spalte (d.h: ohne Duplikate)

Nullwerte

Beispiel

Gibt es Tupel, bei denen der Wert für Semester unbekannt ist, so gilt:

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

$$\neq$$

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

Nullwerte: Behandlung in Ausdrücken

Arithmetische Ausdrücke: Nullwerten werden propagiert: ist ein Operand null so ist das Ergebnis null.

Beispiel

null + 1 = null; null * 0 = null; ...

Vergleichsoperatoren: SQL hat dreiwertige Logik: true, false, unknown. Das Resultat ist unknown, wenn mindestens eines der Argumente null ist.

Beispiel

(Semester > 13) liefert unknown, wenn das Semester unbekannt ist, d.h. den Wert null annimmt.

Achtung! z.B. auch (Semester = null)

Nullwerte – Auswertung logischer Ausdrücke

Logische Ausdrücke: werden nach den folgenden Tabellen berechnet:

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

Nullwerte – Auswertung

where-Bedingung: es werden nur Tupel weitergereicht, für die die Bedingung zu true auswertet. Tupel, für die die Bedingung zu unknown auswertet, werden nicht ins Ergebnis aufgenommen.

Gruppierung: wird null als ein eigenständiger Wert aufgefasst und in eine eigene Gruppe eingeordnet.

Nullwerte werden abgefragt mittels: `is null` bzw. `is not null`.

Beispiel

```
select *
from Student
where Semester is null;
```

Spezielle Sprachkonstrukte

Weitere Sprachkonstrukte

- `between`
- `like`
- `case`
- Joins
  - `cross join`
  - `natural join`
  - `join`
  - `left`, `right`, `full outer join`

between

Beispiele

Suchen Sie jene Studierende, die zwischen dem ersten und dem vierten Semester inskribiert sind:

```
select * from StudentIn
where Semester >= 1 and Semester <= 4;
```

```
select * from StudentIn
where Semester between 1 and 4;
```

```
select * from StudentIn
where Semester in (1,2,3,4);
```

like

Vergleich von Zeichenketten: Platzhalter '%' und '_'

- %: beliebig viele (auch gar kein) Zeichen
- _: genau ein Zeichen

Beispiele

Suchen Sie die Matrikelnummer von Theophrastos, wobei Sie nicht wissen, ob er sich mit 'h' schreibt:

```
select * from StudentIn
where Name like 'T%eophrastos';
```

Suchen Sie die Matrikelnummern jener Studierende, die mindestens eine LVA über Ethik gehört haben:

```
select distinct MatrNr
from Vorlesung v, hoeren h
where h.VorlNr = v.VorlNr and
      v.Titel like '%thik%';
```

case

Beispiel

Wandeln Sie die Prüfungsnoten in Werte der Studienabteilung um:

```
select MatrNr, (case when Note < 1.5 then 'S1'
                     when Note < 2.5 then 'U2'
                     when Note < 3.5 then 'B3'
                     when Note < 4.0 then 'G4'
                     else 'N5' end)
from pruefen;
```

Die erste qualifizierende `when`-Klausel wird ausgeführt

Joins

SQL unterstützt folgende Join-Schlüsselworte:

`cross join`	Kreuzprodukt
`natural join`	natürlicher Join
`[inner] join`	Theta-Join (oder inner join)
`(left|right|full) outer join`	äußerer Join

`cross join`:

$$R_1 \times R_2$$

```
select * from ProfessorIn, Vorlesung;
```

```
select * from ProfessorIn cross join Vorlesung;
```

`natural join`:

$$R_1 \bowtie R_2$$

Die Werte jener Spalten, deren Attributnamen dieselben sind, werden gleichgesetzt.

Beispiel

Geben Sie Name und Matrikelnummer der Studierenden und den Titel der von ihnen gehörten Vorlesungen aus.

```
StudentIn(MatrNr, Name, Sem)
hoeren(MatrNr, VorlNr)
Vorlesung(VorlNr, Titel, SWS, gelesenVon)
```

natural join:

Beispiel

Geben Sie Name und Matrikelnummer der Studierenden und den Titel der von ihnen gehörten Vorlesungen aus.

```
StudentIn(MatrNr, Name, Sem)
hoeren(MatrNr, VorlNr)
Vorlesung(VorlNr, Titel, SWS, gelesenVon)
```

```
select StudentIn.MatrNr, Name, Titel
from StudentIn, hoeren, Vorlesung
where StudentIn.MatrNr=hoeren.MatrNr and
      hoeren.VorlNr=Vorlesung.VorlNr;
```

```
select MatrNr, Name, Titel
from StudentIn natural join hoeren
               natural join Vorlesung;
```

[inner] join:

$$R_1 \bowtie_\theta R_2$$

Beispiel

Welche ProfessorInnen lesen die LVA Mäeutik?

```
ProfessorIn(PersNr, Name, Rang, Raum)
Vorlesung(VorlNr, Titel, SWS, gelesenVon)

select Name, Titel
from ProfessorIn, Vorlesung
where PersNr = gelesenVon and
      Titel = 'Maeutik';
```

```
select Name, Titel
from ProfessorIn join
     Vorlesung on PersNr=gelesenVon
where Titel='Maeutik';
```

(left|right|full) outer join:

$$R_1 (\bowtie | \ltimes | \rtimes) R_2$$

Beispiel

Geben Sie eine Liste aller Studierenden aus und ihre Noten zu den abgeprüften Vorlesungen.

```
select s.MatrNr, s.Name, p.VorlNr, p.Note
from StudentIn s left outer join pruefen p
                 on s.MatrNr=p.MatrNr;
```

```
select s.MatrNr, s.Name, p.VorlNr, p.Note
from pruefen p right outer join StudentIn s
               on s.MatrNr=p.MatrNr;
```

StudentIn ⟕ *prüfen*			
MatrNr	Name	VorlNr	Note
24002	Xenokrates		
25403	Jonas	5041	2
26120	Fichte		
26830	Aristoxenos		
27550	Schopenhauer	4630	2
28106	Carnap	5001	1
29120	Theophrastos		
29555	Feuerbach		

Ergebnis ohne Verwendung des outer Joins:

StudentIn ⋈ *prüfen*			
MatrNr	Name	VorlNr	Note
25403	Jonas	5041	2
27550	Schopenhauer	4630	2
28106	Carnap	5001	1

coalesce()

Beispiel

Geben Sie eine Liste aller Studierenden aus und ihre Noten zu den abgeprüften Vorlesungen. Für Studierende die keine Vorlesung besucht haben sollen VorlNr und Note den Wert 0 annehmen.

```
select  s.MatrNr, s.Name,
        coalesce(p.VorlNr,0),
        coalesce(p.Note,0)
from  StudentIn s left outer join pruefen p
                  on s.MatrNr=p.MatrNr;
```

Achtung: Typen müssen kompatibel sein

concat()

Beispiel

```
StudentIn(MatrNr, Vorname, Nachname, Semester);
```

Geben Sie den Namen aller Studierenden aus

```
select Vorname || Nachname from StudentIn;
```

oder

```
select concat(Vorname, Nachname) from StudentIn;
```

(oft auch “+” anstatt “||”)

Überblick

1. SQL einst und jetzt
2. Datenabfragesprache
3. Datendefinitionssprache
4. Datenmanipulationssprache

Datendefinitionssprache

Datentypen: Konstrukte für Zeichenketten, Zahlen, Datum, ...

```
character(n),           char(n)
character varying(n), varchar(n)

numeric(p,s)
integer, int

date

blob, raw     %fuer grosse binaere bzw. Text Daten
clob          %fuer grosse Text Daten
```

Datendefinitionssprache

Schemadefinition und -veränderung

```
create table ProfessorIn
    (PersNr  integer primary key,
     Name    varchar(10) not null,
     Rang    character(2));
```

```
create table voraussetzen
    (VorgNr  integer,
     NachfNr integer,
     primary key (VorgNr, NachfNr)
    );
```

Datendefinitionssprache

Schemadefinition und -veränderung

```
drop table ProfessorIn;
```

```
alter table ProfessorIn rename to Vortragende;

alter table ProfessorIn add Raum integer;

alter table ProfessorIn drop Raum;

alter table ProfessorIn alter
                          Name type varchar(30);
```

```
alter table ProfessorIn alter
          column Name set data type varchar(30);
```

Datenmanipulationssprache – Tupel Einfügen

Einfügen von Tupeln in eine angelegte Tabelle

```
ProfessorIn(PersNr:integer, Name:varchar(30),
            Rang:character(2), Raum:integer)
```

Beispiel

Einfügen der Professorin Curie:

```
insert into ProfessorIn
values(2136, 'Curie', 'C4', 36);
```

Einfügen mehrerer ProfessorIn:

```
insert into ProfessorIn
values (2125, 'Sokrates', 'C4', 226),
       (2126, 'Russel', 'C4', 232);
```

Datenmanipulationssprache – Tupel Einfügen

Das Ergebnis einer Anfrage in eine Tabelle eintragen:

Beispiel

Eintragen aller Studierende zur Vorlesung ‘Logik’:

```
hoeren (MatrNr, VorlNr)
StudentIn (MatrNr, Name, Sem)
Vorlesung (VorlNr, Titel, SWS, PersNr)
```

```
insert into hoeren
select MatrNr, VorlNr
from StudentIn, Vorlesung
where Titel='Logik';
```

Datenmanipulationssprache – Tupel Löschen

Auflisten der zu löschenden Tupeln:

Beispiel

Löschen des Herrn Kant aus der ProfessorInnentabelle:

```
delete from ProfessorIn
values (2137, 'Kant', 'C4', 7);
```

Lösche alle Tupel die eine Bedingung erfüllen:

Beispiel

Löschen des Herrn Kant aus der ProfessorInnentabelle:

```
delete from ProfessorIn where PersNr=2137;
```

```
delete from ProfessorIn where PersNr < 2137;
```

Datenmanipulationssprache – Tupel Löschen

Beispiel

Lösche alle Studierende, die die Vorlesung ‘Logik’ besuchen

```
hoeren (MatrNr, VorlNr)
StudentIn (MatrNr, Name, Sem)
Vorlesung (VorlNr, Titel, SWS, PersNr)
```

```
delete from StudentIn
where MatrNr in (
  select MatrNr
  from hoeren, Vorlesung
  where hoeren.VorlNr = Vorlesung.VorlNr and
        Titel = 'Logik');
```

Datenmanipulationssprache – Tupel Verändern

Verändern von Tupeln

Beispiel

Erhöhen der Semesteranzahl aller Studierender um 1:

```
update StudentIn
set Semester = Semester + 1;
```

Beispiel

Alle C3 ProfessorInnen mit einer Personalnummer über 2500 erhalten den Rang C2

```
update ProfessorIn
set Rang = 'C2'
where Rang = 'C3' and PersNr > 2500;
```

Datenmanipulationssprache – Tupel Verändern

Verändern von Tupeln

Beispiel

```
ProfessorIn(PersNr:integer, Name:varchar(30),
            Rang:character(2), Raum:integer,
            Lehrbelastung: integer)
Vorlesung (VorlNr, Titel, SWS, PersNr)
```

Trage für jeden Professor seine Lehrbelastung (=Summe SWS) ein

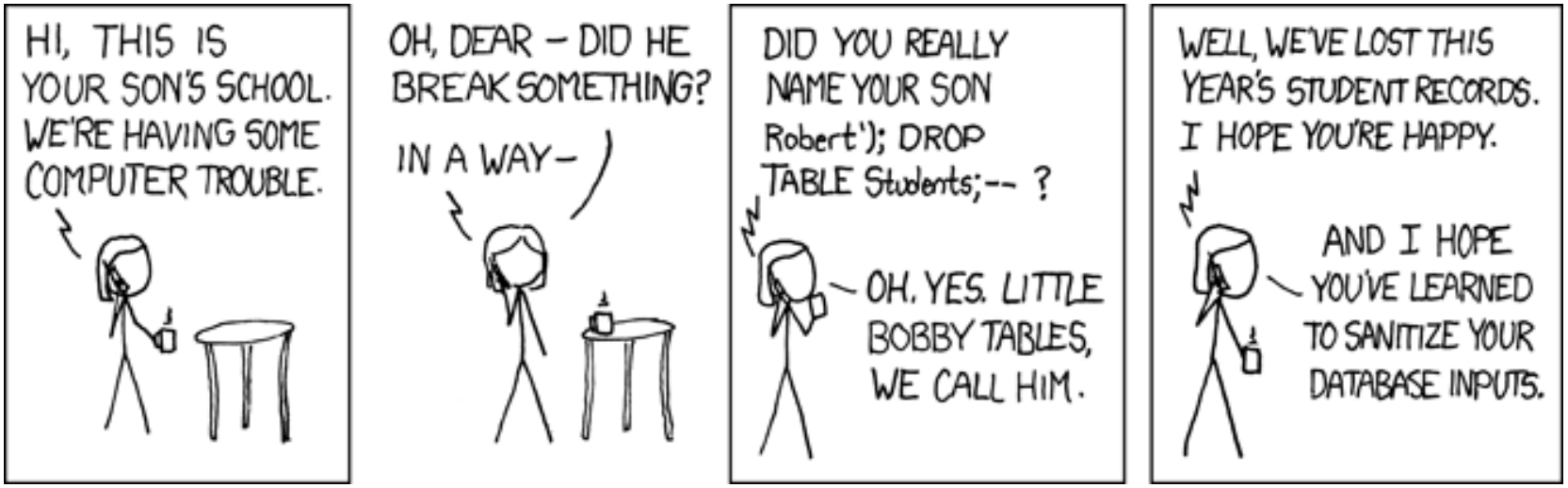
```
    update ProfessorIn
    set Lehrbelastung = (
      select sum(SWS)
      from Vorlesung v
      where v.PersNr = ProfessorIn.PersNr)
```

(Anmerkung: Lehrbelastung sollte so nicht gespeichert werden)

Lernziele

- Einfache SQL-Anfragen
- Anfragen über mehrere Relationen
- Mengenoperationen
- Aggregatfunktionen
- Aggregate und Gruppierung
- Geschachtelte Anfragen
- Existenziell quantifizierte Anfragen
- Allquantifizierte Anfragen
- Nullwerte
- Spezielle Sprachkonstrukte

Ein Wort zur Sicherheit – SQL Injections



(c) xkcd.com