Relationale Abfragesprachen

VO Datenmodellierung

Katrin Seyr

Institut für Informationssysteme
Technische Universität Wien

Überblick

SQL einst und jetzt

SQL wurde auf Basis von relationaler Algebra und Relationenkalkül entwickelt

deklarative Sprache

mengenorientierte Sprache

Abarbeitung Übersetzung der Abfragen mittels Parser in relationale Algebra und Optimierung durch Anfragenoptimierer des DBMS (VO Datenbanksysteme)

Relationen werden dargestellt in Form von Tabellen

SQL stellt eine standardisierte

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

FAKULTÄT FÜR !NFORMATIK

SQL einst und jetzt

SQL 99: derzeit aktueller Standard, 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, nicht in DBMS realisiert

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

Datendefinitionssprache

Datentypen: Konstrukte für Zeichenketten, Zahlen und Datum

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

numeric(p,s)
integer, int

date

blob, raw      %für große binäre bzw. Text Daten
clob           %für große Text Daten
```

Datendefinitionssprache

Schemadefinition und -veränderung

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

```
drop table Professoren;
```

```
alter table Professoren add (Raum integer);
alter table Professoren modify (Name varchar(30));
```

FAKULTÄT FÜR !NFORMATIK

Datenmanipulationssprache

Einfügen von Tupeln in eine angelegte Tabelle

Beispiel

Einfügen der Professorin Curie:

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

Beispiel

Eintragen aller Studenten zur Vorlesung 'Logik':

```
insert into hören
select MatrNr, VorlNr
from Studenten, Vorlesungen
where Titel='Logik';
```

Datenmanipulationssprache

Löschen von Tupeln

Beispiel

Löschen des Herrn Kant aus der Professorentabelle:

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

Verändern von Tupeln

Beispiel

Erhöhen der Semesteranzahl aller Studierender um 1:

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

Datenabfragesprache

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

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 Professoren an:

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

Ergebnis

PersNr	Name
2125	Sokrates
2126	Russel
2136	Curie
2137	Kant

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

Einfache SQL Anfragen

Beispiel (Sortierung)

Sortieren Sie die Personalnummer, Name und Rang aller Professoren absteigend nach dem Rang und aufsteigend nach dem Namen.

```
select  PersNr, Name, Rang
from Professoren
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

Beispiel (Duplikatelimination)

Geben Sie alle Rangbezeichnungen für Professoren ohne Duplikate aus

```
select distinct Rang
from Professoren;
```

Ergebnis
Rang
C4
C3

Anfragen über mehrere Relationen

Steht die gewünschte Information in mehreren Tabellen, so müssen diese in der `where` Klausel verknüpft werden.
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 und wähle passende aus
3. Projiziere auf `select` Attribute

Anfragen über mehrere Relationen

Beispiel

Welche Professoren lesen die LVA Mäeutik?
Information aus Tabelle: `Professoren, Vorlesungen`

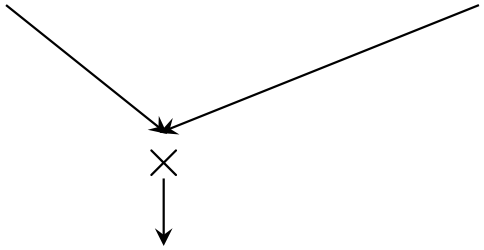
```
Professoren(PersNr, Name, Rang, Raum)
Vorlesungen(VorlNr, Titel, SWS, gelesenVon)
```

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

Anfragen über mehrere Relationen

Abarbeitung:

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



Vorlesungen			
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

$Professoren \times Vorlesungen$							
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

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

$\Pi_{Name, Titel}(\ldots)$	
Name	Titel
Sokrates	Mäeutik

FAKULTÄT FÜR !NFORMATIK

Anfragen über mehrere Relationen

Beispiel

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

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

bzw. bei Vergabe von Platzhaltern für Tabellennamen:

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

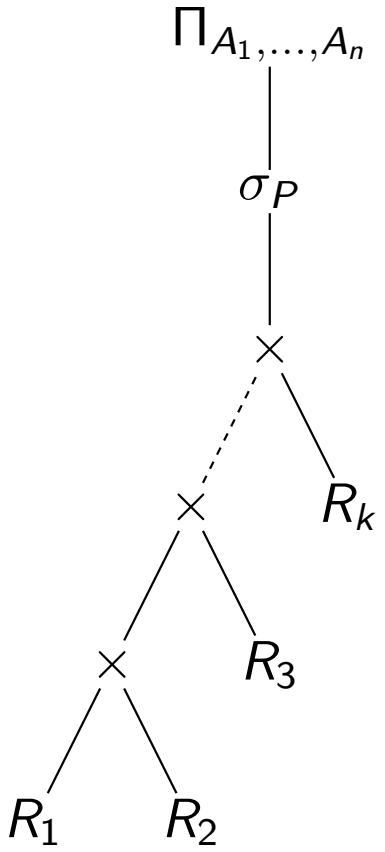
Übersetzung in relationale Algebra

Allgemeine Form einer (ungeschachteteln) SQL-Anfrage:

```
select A1, ..., An
from R1, ..., Rk
where P;
```

wird zu:

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



Mengenoperationen

Anfragen mit typkompatiblen Ausgabeattributen können mittels Mengenoperationen verknüpft werden.
`union` (ohne Duplikate), `union all`, `intersect`, `except` (minus)

Beispiel

Suchen Sie den Namen aller Assistenten oder Professoren

```
( select Name from Assistenten )
union
( select Name from Professoren);
```

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.

Beispiel

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

```
select avg(Semester)
from Studenten;
select max(Semester)
from Studenten;
select min(Semester)
from Studenten;
select count(MatrNr)
from Studenten;
```

Studenten

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

Geben Sie die Summe der von C4 Professoren gehaltenen Vorlesungsstunden an.

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

Vorlesungen

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
⋮	⋮	⋮	⋮

Professoren

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

Aggregate und Gruppierung

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

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

Allgemein: Alle Zeilen einer Tabelle, die auf den Attributen der `group by` Klause 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 Klause angeführt.

Bsp: Aggregate und Gruppierung

Beispiele

Geben Sie zu jedem C4 Professor die Summe der von ihm gehaltenen Vorlesungsstunden an.

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

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

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

Aggregate und Gruppierung

Abarbeitung:

Professoren × *Vorlesungen*							
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

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

FAKULTÄT FÜR !NFORMATIK

Aggregate und Gruppierung

Gruppierung

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

FAKULTÄT FÜR !NFORMATIK

Aggregate und Gruppierung

having-Bedingung

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

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

Aggregate und Gruppierung

Achtung: SQL erzeugt pro Gruppe ein Ergebnistupel

⇒ 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 den Namen der Studenten aus, die am längsten studieren.

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

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

Beispiel (2. Versuch)

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

⇒ Alle Tupel ?!?

Lösung: Geschachtelte Anfrage

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

Unteranfrage in der `where` Klausel

Beispiel

Welche Studenten studieren am längsten?

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

Geschachtelte Anfragen

Unteranfrage in der `select` Klausel

Beispiel

Geben Sie zu jedem Vortragenden seine Lehrbelastung aus.

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

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

Geschachtelte Anfragen

Unteranfrage in der `from` Klausel

Beispiel

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

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

Geschachtelte Anfragen

Beispiel (Entschachtelung mittels `having`)

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

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

Geschachtelte Anfragen

Mengenvergleich mit `in`/`not in`

Beispiel

Suchen Sie alle jene Professoren, die keine Lehrveranstaltungen halten.

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

Geschachtelte Anfragen

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

Beispiel

Suchen Sie jene Studenten, die am längsten studieren.

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

gleichbedeutend mit:

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

Geschachtelte Anfragen

Geschachtelte Aggregatfunktionen sind im Allgemeinen nicht erlaubt ⇒ Unteranfrage verwenden

Beispiel

Suchen Sie C4 Professoren, die die meisten Vorlesungsstunden halten.

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

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

Existentiell quantifizierte Anfragen

Verknüpung einer Anfrage mit einer Unteranfrage durch Existenzquantor `exists`
`exists` überprüft, ob die Unterabfrage Tupel enthält oder nicht.

Beispiel

Suchen Sie alle jene Studenten, die älter als der jüngste Professor sind.

```
select s.*
from Studenten s
where exists (select p.*
              from Professoren p
              where p.GebDatum > s.GebDatum);
```

Korrelierte Unteranfrage (`s.GebDatum` und `Studenten s`)! Eine nicht korrelierte Lösung ist:

```
select s.*
from Studenten s
where s.GebDatum < (select max(p.GebDatum)
                    from Professoren p);
```

Existentiell quantifizierte Anfragen

Beispiel

Suchen Sie alle jene Professoren, die keine Lehrveranstaltungen halten.

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

Korrelierte Unteranfrage (`PersNr` und `Professoren`)! Eine nicht korrelierte Lösung ist:

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

Existentiell quantifizierte Anfragen

Beispiel

Suchen Sie jene Assistenten, die für einen Professor arbeiten, der jünger ist, als sie selbst.

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

Korrelierte Unteranfrage (`a.GebDatum` und `Assistenten a`)! Eine nicht korrelierte Lösung mit Join ist:

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

Allquantifizierte Anfragen

SQL stellt keinen Allquantor zur Verfügung. Realisierung durch Umformulierung in äqivalente Anfrage mit Negation und Existenzquantor (Prädikatenlogik)

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

Beispiel

Welche Studenten haben alle 4 stündigen Lehrveranstaltungen gehört?

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

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

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

Allquantifizierte Anfragen

SQL Umsetzung folgt nun direkt aus:

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

Beispiel

```
select s.*
from Studenten s
where not exists
          (select *
          from Vorlesungen v
          where v.SWS = 4 and
          s.MatrNr not in (select h.MatrNr
                           from hören h
                           where h.VorlNr = v.VorlNr));
```

Allquantifizierte Anfragen

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

Beispiel

Welche Studenten haben alle 4 stündigen Lehrveranstaltungen gehört?

1. Zähle zu jedem Studenten, wieviele 4 stündigen LVAs er besucht hat
2. zähle wieviele 4 stündige LVAs es gibt.

```
select s.MatrNr, s.Name
from Studenten s, hören h, Vorlesungen 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 Vorlesungen
                    where SWS = 4);
```

Nullwerte

Nullwerte entstehen

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

Beispiel

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

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

$\neq$

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

Nullwerte

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.

Nullwerte

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

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

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

between

Beispiele

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

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

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

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

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

```
select distinct MatrNr
from Vorlesungen v, hören 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 'B2'
                     when Note < 3.5 then 'U3'
                     when Note < 4.0 then 'G4'
                     else 'N5' end)
from prüfen;
```

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 Professoren, Vorlesungen;
```

```
select * from Professoren cross join Vorlesungen;
```

`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.

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

```
select MatrNr, Name, Titel
from Studenten natural join  hören
                natural join Vorlesungen;
```

[inner] join:

$$R_1 \bowtie_\theta R_2$$

Beispiel

Welche Professoren lesen die LVA Mäeutik?

```
Professoren(PersNr, Name, Rang, Raum)
Vorlesungen(VorlNr, Titel, SWS, gelesenVon)

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

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

(left|right|full) outer join:

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

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 Studenten s left outer join prüfen p
                 on s.MatrNr=p.MatrNr;
```

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

Studenten ⟕ *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:

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

Sichten (Views)

... sind gespeicherte Anfragen, die als virtuelle Tabelle zur Verfügung stehen.

... waren ursprünglich nur für den Lesezugriff gedacht, sind aber unter bestimmten Einschränkungen auch update-fähig.

... werden bei jedem Zugriff dynamisch neu erstellt.

... sind nicht Teil des physischen Schemas

Verwendung:

- Konzept zur Anpassung an verschiedene Benutzer
- Vereinfachung von Anfragen
- Realisierung der Generalisierung

DBMS proprietär:

- materialized views (ORACLE): präkompilierte nicht mehr virtuelle Anfrage
- indexed views (SQL Server): zusätzlich gespeicherter Index

verwendet zur Beschleunigung von sehr häufig durchgeführten Anfragen.

Anpassung an unterschiedliche Benutzergruppen

Beispiel

Aus Datenschutzgründen soll im Allgemeinen nur die Prüfungsliste, nicht aber die Note lesbar sein:

```
create view prüfenSicht as
select MatrNr, VorlNr, PersNr
from prüfen;
```

Vereinfachung von Anfragen

Beispiel

Vermeidung der ständigen Verwendung des Joins

$$\textit{Studenten} \bowtie \textit{hören} \bowtie \textit{Vorlesungen} \bowtie \textit{Professoren}$$

```
create view StudProf(Sname, Semester, Titel, Pname) as
select s.Name, s.Semester, v.Titel, p.Name
from Studenten s, hören h, Vorlesungen v, Professoren p
where s.MatrNr=h.MatrNr and
      h.VorlNr=v.VorlNr and
      v.gelesenVon = p.PersNr;
```

Suchen Sie Name, Semester aller Studenten von Sokrates

```
select distinct Name, Semester from StudProf
where Pname='Sokrates';
```

Realisierung der Generalisierung

Sichten realisieren Inklusion und Vererbung:

- Tupel des Untertyps sollen auch automatisch zum Obertyp gehören
- Attribute des Obertyps: sollen automatisch vererbt werden.

$$\Rightarrow$$

entweder Untertypen oder Obertypen als Sicht definieren
Entscheidung aufgrund der Häufigkeit der Zugriffe

Realisierung der Generalisierung

Beispiel (Untertypen als Sicht)

```
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),
                        Boss       integer);


create view Professoren as select *
from Angestellte natural join ProfDaten;

create view Assistenten as select *
from Angestellte natural join  AssiDaten;
```

Realisierung der Generalisierung

Beispiel (Obertypen als Sicht)

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


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

Updates auf Sichten

Problematisch: `insert`, `update`, `delete` auf eine Sicht

Beispiele (nicht updatefähige Sichten)

```
create view WieHartAlsPrüfer(PersNr, Durchschnitt) as
select PersNr, avg(Note)
from prüfen
group by PersNr;
```

```
create view VorlesungenSicht as
select Titel, SWS, Name
from Vorlesungen, Professoren
where gelesenVon = PersNr;
```

Updates auf Sichten

Einschränkungen der update-fähigen Sichten in SQL auf:

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