

## 5. SQL

---

- ER-Modell
- Relationenmodell
- relationale Anfragesprachen
- **SQL**
- Entwurfstheorie
- Transaktionen

# Relationenmodell vs. SQL-Tabellen

## Relationenmodell

vs.

## SQL Tabellen

Relation

Tabelle

Attribut

Spalte

Tupel

Zeile

keine *null*-Werte

*null*-Werte möglich

keine doppelten Tupel

Tupel-Duplikate möglich

keine Sortierung der Tupel

Sortierung der Tupel möglich

## Notationen:

<...> (Ersetzen durch entsprechenden Inhalt),

[...] (optionaler Bestandteil),

| (oder)

## SQL = Structured Query Language

- 1974: SEQUEL in *System R* (Vorgänger)
- 1979: SQL in *Oracle V2*
- 1986: SQL1 / SQL-86 (erster ANSI-Standard)
- 1992: SQL2 / SQL-92 Standard
- weitere Standards/Erweiterungen (1999, 2003, ...)
- verschiedene Dialekte (*mySQL*, *PostgreSQL*, *Oracle*, ...)

### Disclaimer

Die Vorlesung orientiert sich hauptsächlich am SQL-92 Standard.

Befehle, die hier nicht besprochen wurden, werden in den Lösungen nicht akzeptiert.

## **Data Definition Language (DDL):** Definitionssprache

- Befehle zur Definition des Datenbankschemas
- Tabellen erzeugen/ändern/löschen

## **Data Query Language (DQL):** Anfragesprache

- Befehle zur Abfrage von Daten
- Daten auslesen/berechnen

## **Data Manipulation Language (DML):** Manipulationssprache

- Befehle zur Datenmanipulation
- Datensätze erzeugen/ändern/löschen

## Data Definition Language (DDL): Definitionssprache

- Befehle zur Definition des Datenbankschemas
- Tabelle/Relationenschema erstellen (*create table*)
  - Festlegen der Spaltennamen
  - Festlegen der Domänen der Spalten (Datentypen)
  - Festlegen von Integritätsbedingungen
- Bestehendes Schema ändern (*alter table*)
- Tabelle löschen (*drop table*)

## Tabellenschema erstellen

```
create table <Tabellenname> (  
    <Spaltendefinitionen>,  
    [<Integritätsbedingungen>]);
```

**Spaltendefinition:** <Spaltenname> <Datentyp> [<Integritätsbedingungen>]

## Zeichenketten

- *char*(*n*): Zeichenkette mit fester Länge *n*
- *varchar*(*n*): Zeichenkette mit variabler Länge *n*

## Zahlen

- *integer/int*: ganzzahlige Werte
- *decimal*(*n*, *q*): Festkommazahl mit *n* Stellen und davon *q* Nachkommastellen
- *float/real*: Gleitkommazahlen
- *boolean*: Wahrheitswerte

## Datum/Zeit

- *date*: Datum im Format 'YYYY-MM-DD'
- *time*: Zeit im Format 'hh:mm:ss'
- *datetime*: Kombination 'YYYY-MM-DD hh:mm:ss'



# Erstellung - Beispiel

Vorlesung(ID, Titel, CP, gehalten\_von)    Prof(ID, Vorname, Nachname)

```
create table Vorlesung(  
  ID int,  
  Titel varchar(250),  
  CP int,  
  gehalten_von int);
```

## *not null*

- Stellt sicher, dass keine *null*-Werte in einer Spalte stehen.

### Beispiel:

Vorlesung(ID, Titel, CP, gehalten\_von)    Prof(ID, Vorname, Nachname)

```
create table Vorlesung(  
  ID int not null,  
  Titel varchar(250),  
  CP int,  
  gehalten_von int not null);
```

# Integritätsbedingungen

## *default* <Wert>

- Legt für neue Einträge einen default-Wert fest, wenn kein anderer Wert übergeben wird.

## Beispiel:

Vorlesung(ID, Titel, CP, gehalten\_von)    Prof(ID, Vorname, Nachname)

```
create table Vorlesung(  
  ID int not null,  
  Titel varchar(250) default 'tbd',  
  CP int,  
  gehalten_von int not null);
```

## *unique*

- Stellt sicher, dass keine doppelten Werte vorhanden sind.
- *unique*(<Spalte1>,<Spalte2>,...) für Kombination von mehreren Spalten

## Beispiel:

Vorlesung(ID, Titel, CP, gehalten\_von)    Prof(ID, Vorname, Nachname)

```
create table Vorlesung(  
  ID int not null unique,  
  Titel varchar(250) default 'tbd',  
  CP int,  
  gehalten_von int not null,  
  unique(Titel, gehalten_von));
```

## *primary key*

- Legt den Primärschlüssel für die Tabelle fest
- stellt automatisch *not null* und *unique* sicher
- kann nur einmal pro Tabelle definiert werden

# Erstellung - Beispiel

Prof(ID, Vorname, Nachname)

```
create table Prof(  
  ID int not null unique primary key,  
  Vorname char(50),  
  Nachname char(50));
```

Alternativ:

```
create table Prof(  
  ID int not null unique,  
  Vorname char(50),  
  Nachname char(50),  
  primary key(ID));
```

hoert(MatrNr, VID)

```
create table hoert(  
  MatrNr int,  
  VID int,  
  primary key (MatrNr,VID));
```

⚠ Primärschlüssel/Fremdschlüssel aus mehreren Attributen können nicht inline definiert werden!

## *auto\_increment*

- Generiert eine fortlaufende Nummer beim Einfügen
- gerne verwendet für die automatische Generierung künstlicher Schlüssel

### Beispiel:

Prof(ID, Vorname, Nachname)

```
create table Prof(  
  ID int auto_increment primary key,  
  Vorname char(50),  
  Nachname char(50));
```



## *foreign key*

- referenziert den Primärschlüssel einer anderen Tabelle
- *foreign key* (<Spalte>) *references* <Tabelle2>(<PK von Tabelle 2>)
- auf Datentypen achten!
- verhindert, dass nicht vorhandene Werte eingefügt werden, aber verhindert nicht *null*-Werte

# Erstellung - Beispiel

Vorlesung(ID, Titel, CP, gehalten\_von)    Prof(ID, Vorname, Nachname)

```
create table Prof(  
  ID int primary key,  
  [...]);
```

```
create table Vorlesung(  
  ID int primary key,  
  Titel char(50),  
  CP int,  
  gehalten_von int not null,  
  foreign key(gehalten_von) references Prof(ID)  
);
```

# Erstellung - Beispiel

Studi(MatrNr, Vorname, Nachname, Fach, Semester)    hoert(MatrNr, VID)  
Vorlesung(ID, Titel, CP, gehalten\_von)

```
create table Studi(  
  MatrNr int primary key,  
  [...]);
```

```
create table Vorlesung(  
  ID int primary key,  
  [...]);
```

```
create table hoert(  
  MatrNr int,  
  VID int,  
  primary key (MatrNr,VID),  
  foreign key (MatrNr) references Studi(MatrNr),  
  foreign key (VID) references Vorlesung(ID) );
```

## 1:1-Beziehung (Kapazitätserhaltende Abbildung)

- Ein Fremdschlüssel als *primary key*
- Ein Fremdschlüssel als *unique [not null]*

Lehrerin(ID, ...) leitet(ID, Nr) Klasse(Nr, ...)

```
create table leitet(  
  ID int unique not null,  
  Nr int primary key,  
  foreign key (ID) references Lehrerin(ID),  
  foreign key (Nr) references Klasse(Nr));
```

## Tabellenschema löschen

```
drop table <Tabellenname>;
```

- löscht nicht nur Einträge, sondern die gesamte Tabelle
- für Löschen von Einträgen siehe *delete from* (Folie 91)

### Beispiel:

```
drop table Vorlesung;
```

## Tabellenschema verändern

```
alter table <Tabellenname>  
    <Modifikationen>;
```

Modifikationen der Spaltendefinition:

- Spalte hinzufügen: *add column* <Spaltendefinition>
- Spalte löschen: *drop column* <Spalte>
- Spalte umbenennen: *rename column* <alterName> *to* <neuerName>
- Spalte modifizieren (z.B. neuer Datentyp): *alter column* <Spaltendefinition>

## Tabellenschema verändern

```
alter table <Tabellenname>  
    <Modifikationen>;
```

Modifikationen der Integritätsbedingungen:

- Bedingung hinzufügen: *add* <Integritätsbedingung> oder *add constraint* <BedName> <Integritätsbedingung>
- Bedingung löschen: *drop* <Integritätsbedingung> oder *drop constraint* <BedName>

## Änderung - Beispiel

```
create table Personal(  
  Name char(50) primary key);
```

Änderungen:

```
alter table Personal add column PersonalNr int;
```

```
alter table Personal drop primary key;
```

```
alter table Personal add primary key(PersonalNr); oder z.B  
alter table Personal add constraint pk_personal primary key(PersonalNr);
```

```
alter table Personal alter column Name varchar(250) not null;
```



## Data Query Language (DQL): Anfragesprache

- Befehle zur Abfrage von Daten
- Daten auslesen/berechnen

SQL ist eine *relational vollständige* relationale Anfragesprache, aber noch mächtiger als die Relationenalgebra

- Nullwerte, Sortierung, Berechnungen, Zählen, Gruppierungen etc.

```
select <Spalten/Funktionen>  
from <Tabellen>  
[where <Bedingungen>]  
[group by <Gruppierungsspalten>]  
[having <Gruppierungsbedingung>]  
[order by <Spalten>;]
```

1. Bilden des Kreuzprodukts der Tabellen im *from*-Klausel
2. Entfernen der Tupel, die den *where*-Klausel nicht erfüllen
3. Gruppierung der Tupel gemäß *group by*-Klausel
4. Entfernen der Gruppen, die den *having*-Klausel nicht erfüllen
5. Auswahl/Evaluierung der Ausdrücke im *select*-Klausel
6. Sortierung des Resultats gemäß *order by*-Klausel

```
select *  
from <Tabellen>;
```

- enthält die Tabellen, die für die Anfrage nötig sind.
- bei mehreren Tabellen wird das Kreuzprodukt gebildet

*select* \*

- Sonderzeichen \* im *select* liefert alle Spalten, d.h. die gesamte Tabelle wird ausgegeben

## Basic Anfragen: select-Klausel

```
select [distinct] <Spalten>  
from <Tabellen>;
```

*select* <Spalten>:

- wählt die Spalten aus, die in der Liste genannt sind

*select distinct* <Spalten>:

- entfernt zusätzlich Duplikate im Ergebnis
- $\pi_A(R)$  entspricht 

```
select distinct A  
from R;
```

⚠ *select* entspricht nicht Selektion in der Relationenalgebra!

## Basic Anfragen - Beispiel

Studi(MatrNr, Vorname, Nachname, Fach, Semester)   hoert(MatrNr, VID)  
Vorlesung(ID, Titel, CP, gehalten\_von)   Prof(ID, Vorname, Nachname)

**Anfrage:** Geben Sie alle Informationen zu den Vorlesungen aus.

```
select *  
from Vorlesung;
```

**Anfrage:** Geben Sie die Titel aller Vorlesungen aus. Jeder Titel soll nur einmal in der in der Ausgabe auftauchen.

```
select distinct Titel  
from Vorlesung;
```

## Basic Anfragen: where-Klausel

```
select <Spalten>  
from <Tabellen>  
where <Bedingung>;
```

- enthält Selektionsprädikate mit Spalten und Konstanten
- Logische Verknüpfung mehrerer Bedingungen mit *and*, *or* oder *not*

- $\sigma_{\langle \text{Bedingung} \rangle}(R)$  entspricht  

```
select distinct *  
from R  
where <Bedingung>;
```

## Basic Anfragen - Beispiel

Studi(MatrNr, Vorname, Nachname, Fach, Semester)   hoert(MatrNr, VID)  
Vorlesung(ID, Titel, CP, gehalten\_von)   Prof(ID, Vorname, Nachname)

**Anfrage:** Geben Sie die vollständigen Namen aller Informatik-Studis aus.

```
select Vorname, Nachname  
from Studi  
where Fach = 'Informatik';
```

## Basic Anfragen - Beispiel

Studi(MatrNr, Vorname, Nachname, Fach, Semester)   hoert(MatrNr, VID)  
Vorlesung(ID, Titel, CP, gehalten\_von)   Prof(ID, Vorname, Nachname)

**Anfrage:** Geben Sie die MatrNr aller Studis aus, die eine Vorlesung mit dem Titel Datenbanken hören.

```
select MatrNr  
from hoert, Vorlesung  
where VID = ID  
      and Titel = 'Datenbanken';
```



Vergleich von Zeichenketten mit Wildcards in der *where*-Klausel

```
<Attribut> [not] like <Spezialkonstante>
```

Wildcards

- % Wildcard für kein oder beliebig viele Zeichen
- \_ Wildcard für genau ein Zeichen

## Wildcards - Beispiel

Studi(MatrNr, Vorname, Nachname, Fach, Semester) hoert(MatrNr, VID)  
Vorlesung(ID, Titel, CP, gehalten\_von) Prof(ID, Vorname, Nachname)

**Anfrage:** Geben Sie alle Studis aus, deren Nachname mit A anfängt.

```
select *  
from Studi  
where Nachname like 'A%';
```

**Anfrage:** Geben Sie alle Studis aus, deren Vorname mindestens 3 Zeichen enthält.

```
select *  
from Studi  
where Vorname like '___%';
```

- temporäre Namen für die Anfrage und Anzeige
- mit *as*-Operator (kann je nach Dialekt weggelassen werden)

## Alias-Namen für Spalten im Ergebnis

```
select <Spaltenname> [as] <Alias-Name>  
from <Tabellen>;
```

- häufig bei berechneten Spalten ohne eigenen Namen eingesetzt (siehe Folie 56)

## Alias-Namen für Tabellen in der Anfrage

```
select <Spalten>  
from <Tabellenname> [as] <Alias-Name>;
```

- nützlich für Unteranfragen, lange Tabellennamen, komplexe Ausdrücke
- nötig bei Selbstjoin (siehe Folie 50)

# Alias-Namen - Beispiel

```
Studi(MatrnNr, Vorname, Nachname, Fach, Semester)  hoert(MatrnNr, VID)  
Vorlesung(ID, Titel, CP, gehalten_von)    Prof(ID, Vorname, Nachname)
```

**Anfrage:** Geben Sie für alle Vorlesungen, die vom Studi mit der MatrNr 123 gehört werden, den Titel und den Nachnamen des Profs aus.

```
select Titel, Nachname  
from hoert, Vorlesung, Prof  
where VID = Vorlesung.ID  
      and gehalten_von = Prof.ID  
      and MatrNr = 123;
```

```
select Titel, Nachname  
from hoert, Vorlesung V, Prof P  
where VID = V.ID  
      and gehalten_von = P.ID  
      and MatrNr = 123;
```

## Alias-Namen - Beispiel

```
Studi(MatrnNr, Vorname, Nachname, Fach, Semester)  hoert(MatrnNr, VID)  
Vorlesung(ID, Titel, CP, gehalten_von)  Prof(ID, Vorname, Nachname)
```

**Anfrage:** Geben Sie für alle Vorlesungen, die vom Studi mit der MatrNr 123 gehört werden, den Titel und den Nachnamen des Profs in der Form (VorlesungTitel, ProfName) aus.

```
select Titel as VorlesungTitel, Nachname as ProfName  
from hoert, Vorlesung V, Prof P  
where VID = V.ID  
      and gehalten_von = P.ID  
      and MatrNr = 123;
```

```
select <Spalten>  
from <Tabellen>  
[...]  
order by <Spalten> [asc | desc];
```

- Sortierung der Einträge im Resultat
- *asc*: Aufsteigende Sortierung (default)
- *desc*: Absteigende Sortierung
- mehrere Sortierattribute möglich:  
Sortierung nach der erstgenannten Spalte, bei Werte-Gleichheit nach der zweitgenannten Spalte, usw.

## Sortierung - Beispiel

Studi(MatrNr, Vorname, Nachname, Fach, Semester)   hoert(MatrNr, VID)  
Vorlesung(ID, Titel, CP, gehalten\_von)   Prof(ID, Vorname, Nachname)

**Anfrage:** Geben alle Informationen zu Studis absteigend sortiert nach der Semesteranzahl aus.

```
select *  
from Studi  
order by Semester desc;
```

## Sortierung - Beispiel

Studi(MatrnNr, Vorname, Nachname, Fach, Semester) hoert(MatrnNr, VID)  
Vorlesung(ID, Titel, CP, gehalten\_von) Prof(ID, Vorname, Nachname)

**Anfrage:** Geben Sie die vollständigen Namen aller Studis aus, die die Vorlesung mit der ID 123 hören. Das Ergebnis soll in alphabetischer Reihenfolge der Nachnamen angegeben werden.

```
select Vorname, Nachname  
from Studi, hoert  
where Studi.MatrnNr = hoert.MatrnNr  
      and VID = 123  
order by Nachname;
```



- Kreuzprodukt: *cross join*
- Theta- und Equi-Join: *join on/using*
- Natural Join: *natural join*
- Outer Joins: *left/right/full outer join*
- Selbstjoin

```
select *  
from <Tabelle1>, <Tabelle2>  
[where <Join-Bedingung>];
```

```
select *  
from <Tabelle1> cross join <Tabelle2>  
[where <Join-Bedingung>];
```

- entspricht mit *distinct*:  $\sigma_{\langle \text{Join-Bedingung} \rangle} (R \times S)$
- beide Anfragen liefern das gleiche Ergebnis

```
select *  
from <Tabelle1> join <Tabelle2> on <Join-Bedingung>;
```

- entspricht mit *distinct*:  $R \bowtie_{\langle \text{Join-Bedingung} \rangle} S$
- Join-Bedingung: <Attribut von R>  $\theta$  <Attribut von S>
- $\theta$ : Vergleichsoperator (<, <=, =, >=, >, <>)
- Equi-Join: mit = als Vergleichsoperator

## Join - Beispiel

Studi(MatrNr, Vorname, Nachname, Fach, Semester)   hoert(MatrNr, VID)  
Vorlesung(ID, Titel, CP, gehalten\_von)   Prof(ID, Vorname, Nachname)

**Anfrage:** Geben Sie die Fächer aller Studis aus, die die Vorlesung mit der ID 123 hören.

```
select Fach
from Studi, hoert
where Studi.MatrNr = hoert.MatrNr
and VID = 123;
```

```
select Fach
from Studi join hoert on Studi.MatrNr = hoert.MatrNr
where VID = 123;
```

## Join - Beispiel

Studi(MatrNr, Vorname, Nachname, Fach, Semester)    hoert(MatrNr, VID)  
Vorlesung(ID, Titel, CP, gehalten\_von)    Prof(ID, Vorname, Nachname)

**Anfrage:** Geben Sie die Fächer aller Studis aus, die eine Vorlesung mit dem Titel Datenbanken hören.

```
select Fach  
from Studi S  
join hoert H on S.MatrNr = H.MatrNr  
join Vorlesung V on V.ID = H.VID  
where Titel = 'Datenbanken';
```

```
select *  
from <Tabelle1> join <Tabelle2> using (<Attribut>);
```

- ähnlich Equi-Join mit einem **gleichnamigen** Attribut
- Attribut-Spalte nur einmal im Resultat, nicht doppelt

## Join - Beispiel

Studi(MatrNr, Vorname, Nachname, Fach, Semester) hoert(MatrNr, VID)  
Vorlesung(ID, Titel, CP, gehalten\_von) Prof(ID, Vorname, Nachname)

**Anfrage:** Geben Sie die Fächer aller Studis aus, die die Vorlesung mit der ID 123 hören.

```
select Fach
from Studi join hoert
  on Studi.MatrNr = hoert.MatrNr
where VID = 123;
```

```
select Fach
from Studi join hoert
  using (MatrNr)
where VID = 123;
```

```
select *  
from <Tabelle1> natural join <Tabelle2>;
```

- entspricht mit *distinct*:  $R \bowtie S$
- Equi-Join über **alle** gleichnamigen Attribute
- alle gleichnamigen Attribute nur einmal im Resultat
- ohne gleichnamige Attribute gleiches Resultat wie beim Kreuzprodukt



## Join - Beispiel

Studi(MatrNr, Vorname, Nachname, Fach, Semester)   hoert(MatrNr, VID)  
Vorlesung(ID, Titel, CP, gehalten\_von)   Prof(ID, Vorname, Nachname)

**Anfrage:** Geben Sie die Fächer aller Studis aus, die die Vorlesung mit der ID 123 hören.

```
select Fach
from Studi join hoert
    using (MatrNr)
where VID = 123;
```

```
select Fach
from Studi natural join hoert
where VID = 123;
```

# Join - Beispiele

R

A	B
a	1
b	2
c	3

S

A	C
a	3
d	7
b	2

*R join S on R.A=S.A*

R.A	R.B	S.A	S.C
a	1	a	3
b	2	b	2

*natural join*

A	B	C
a	1	3
b	2	2

*R join S using (A)*

A	B	C
a	1	3
b	2	2

```
select *  
from <Tabelle1> <Alias-Name1>, <Tabelle1> <Alias-Name2>;
```

- Mehrfacher Zugriff auf dieselbe Tabelle innerhalb *from*-Klausel
- Alias-Namen notwendig!

## Selbstjoin - Beispiel

Studi(MatrNr, Vorname, Nachname, Fach, Semester)    hoert(MatrNr, VID)  
Vorlesung(ID, Titel, CP, gehalten\_von)    Prof(ID, Vorname, Nachname)

**Anfrage:** Geben Sie die Nachnamen alle Studis aus, die mehrfach vorkommen.

```
select S.Nachname  
from Studi R, Studi S  
where R.Nachname = S.Nachname  
      and R.MatrNr <> S.MatrNr;
```

```
select *  
from <Tabelle1> full|left|right outer join <Tabelle2>  
on ...;
```

- Inner Join:  
Nur Tupel mit korrespondierenden Einträgen in beiden Tabellen werden übernommen.
- Outer Join:  
Auch Tupel ohne korrespondierende Einträge werden übernommen und fehlende Attributwerte mit *null* aufgefüllt
  - *full outer join*: in beiden Operanden
  - *left outer join*: im linken Operanden
  - *right outer join*: im rechten Operanden

# Join - Beispiele

$R \langle \text{left} | \text{full} | \text{right} \rangle \text{ outer join } S \text{ on } R.A = S.A$

R

A	B
a	1
b	2
c	3

S

A	C
a	3
d	7
b	5

*full*

R.A	R.B	S.A	S.C
a	1	a	3
b	2	b	5
c	3	<i>null</i>	<i>null</i>
<i>null</i>	<i>null</i>	d	7

*left*

R.A	R.B	S.A	S.C
a	1	a	3
b	2	b	5
c	3	<i>null</i>	<i>null</i>

*right*

R.A	R.B	S.A	S.C
a	1	a	3
b	2	b	5
<i>null</i>	<i>null</i>	d	7

## Outer Join - Beispiel

Studi(MatrnNr, Vorname, Nachname, Fach, Semester) hoert(MatrnNr, VID)  
Vorlesung(ID, Titel, CP, gehalten\_von) Prof(ID, Vorname, Nachname)

**Anfrage:** Geben Sie die IDs von Vorlesungen zusammen mit dem Nachnamen der Profs aus, die diese Vorlesung halten. Geben Sie auch die IDs aller Vorlesungen aus, die von niemanden gehalten werden.

```
select V.ID, Nachname  
from Vorlesung V left outer join Prof P on V.gelalten_von = P.ID;
```

```
select V.ID, Nachname  
from Prof P right outer join Vorlesung V on V.gelalten_von = P.ID;
```

## Outer Join - Beispiel

```
Studi(MatrnNr, Vorname, Nachname, Fach, Semester)  hoert(MatrnNr, VID)
Vorlesung(ID, Titel, CP, gehalten_von)    Prof(ID, Vorname, Nachname)
```

**Anfrage:** Geben Sie die IDs von Vorlesungen zusammen mit dem Nachnamen der Profs aus, die diese Vorlesung halten. Geben Sie auch die IDs aller Vorlesungen aus, die von niemanden gehalten werden, und die Nachnamen der Profs, die keine Vorlesungen halten, aus.

```
select V.ID, Nachname
from Vorlesung V full outer join Prof P on V.gehalten_von = P.ID;
```



```
select <Spalten | arithmetischer Ausdruck>  
from <Tabellen>  
[where <Bedingung mit arithmetischem Ausdruck>];
```

- Operatoren (Auswahl):
  - für Zahlen:  $+$ ,  $-$ ,  $*$ ,  $/$
  - für Zeichenketten: *concat* (Konkatenation), *char\_length* (Anzahl Zeichen)
- berechnete Spalte hat keinen richtigen Namen, aber man kann einen Alias-Namen festlegen (Folie 34)

## Berechnungen - Beispiel

```
select ID, Preis * 1.19 as BruttoPreis  
from Produkt;
```

Produkt

ID	Name	Preis
1	a	13.44
2	b	3.77
3	c	8.39

Ergebnis:

ID	BruttoPreis
1	15.99
2	4.49
3	9.99

## Berechnungen - Beispiel

Studi(MatrNr, Vorname, Nachname, Fach, Fachsemester, Hochschulsemester)

**Anfrage:** Geben Sie alle Studis aus, deren Anzahl an Hochschulsemester mehr als doppelt so hoch wie die Anzahl an Fachsemestern ist.

```
select *  
from Studi  
where Hochschulsemester > 2*Fachsemester;
```

**Anfrage:** Generieren Sie eine Liste der Email-Adressen der Studis.

```
select concat(Vorname, ',', Nachname, '@hhu.de') as Email  
from Studi;
```

```
select <Spalten | arithmetischer Ausdruck | Aggregatfunktion>  
from <Tabellen>;
```

- *sum/avg/min/max(<Attribut>)*
  - *sum*: Summe der Werte einer Spalte
  - *avg*: durchschnittlichen Wert einer Spalte
  - *min* bzw. *max*: minimaler bzw. maximaler Wert in einer Spalte
- nur bei numerischen Wertebereichen
- *null*-Werte werden ignoriert
- *sum/avg(distinct <Attribut>)*: nur unterschiedliche Werte gehen in die Berechnung ein

## Aggregation - Beispiel

Studi(MatrNr, Vorname, Nachname, Fach, Semester)    hoert(MatrNr, VID)  
Vorlesung(ID, Titel, CP, gehalten\_von)    Prof(ID, Vorname, Nachname)

**Anfrage:** Geben Sie die durchschnittliche Semesteranzahl aller Studis aus.

```
select avg(Semester)
from Studi;
```

**Anfrage:** Geben Sie die kleinste und größte CP-Anzahl aller Vorlesungen aus.

```
select min(CP), max(CP)
from Vorlesung;
```

```
select <Spalten | arithmetischer Ausdruck | Aggregatfunktion>  
from <Tabellen>;
```

- *count* (<Attribut>): Anzahl der Werte in einer Spalte
  - *null*-Werte werden ignoriert
  - *count(distinct* <Attribut>): zählt unterschiedliche Werte
- *count* (\*): Zählt alle Tupel in einer Tabelle
  - Auch Tupel, die *null*-Werte enthalten, werden mitgezählt

## Aggregation - Beispiel

Studi(MatrNr, Vorname, Nachname, Fach, Semester)   hoert(MatrNr, VID)  
Vorlesung(ID, Titel, CP, gehalten\_von)   Prof(ID, Vorname, Nachname)

**Anfrage:** Geben Sie die Anzahl aller Vorlesungen (Gesamtanzahl) aus.

```
select count(*) as Gesamtanzahl  
from Vorlesung;
```

**Anfrage:** Geben Sie die Anzahl aller Vorlesungen aus, die von mindestens einer Person gehört werden.

```
select count(distinct VID)  
from hoert;
```

- Das Ergebnis von Aggregatfunktionen besteht aus **einem einzigen Wert!**  
→ **genau eine Zeile im Ergebnis!**
- ohne Gruppierung (siehe Folie 64) sind keine weiteren Attribute in der *select*-Klausel erlaubt

**Anfrage:** Geben Sie die ID und die CP-Anzahl der Vorlesungen mit der größten CP-Anzahl aus.

```
select Vid, max(CP)  
from Vorlesung;
```

⚠ So nicht! Keine gültige Anfrage!



```
select <...>
from <Tabellen>
[where <Bedingungen>]
group by <Spalten>;
```

- gruppierte Tabelle = geschachtelte Tabelle (nicht im Relationenmodell/1NF)
- in der *select*-Klausel erlaubt
  - Spalten aus der *group by*-Klausel
  - Aggregatfunktionen auf den restlichen Spalten der Tabelle

A	B		
		C	D
a	b	1	2
		3	4

A	B	max(C)	sum(D)
a	b	3	6

# Gruppierung - Beispiel

```
select B, sum(D) as Summe  
from R  
group by A,B;
```

R

A	B	C	D
a	b	1	2
a	b	3	4
b	c	1	3
c	c	4	2
c	c	7	5

Gruppierung:

A	B		
		C	D
a	b	1	2
		3	4
b	c	1	3
c	c	4	2
		7	5

Endergebnis:

B	Summe
b	6
c	3
c	7

## Gruppierung - Beispiel

Studi(MatrNr, Vorname, Nachname, Fach, Semester) hoert(MatrNr, VID)  
Vorlesung(ID, Titel, CP, gehalten\_von) Prof(ID, Vorname, Nachname)

**Anfrage:** Geben Sie für alle Vorlesungen jeweils die ID und die Anzahl der teilnehmenden Studis (VID, AnzahlStudis) aus.

Geben Sie nur Vorlesungen mit  
AnzahlStudis > 0 aus.

```
select VID, count(MatrNr) as AnzahlStudis  
from hoert  
group by VID;
```

Geben Sie auch Vorlesungen mit  
AnzahlStudis = 0 aus.

```
select ID as VID, count(MatrNr) as AnzahlStudis  
from Vorlesung left join hoert on VID=ID  
group by ID;
```

# Gruppierung mit Having

```
select <...>  
from <Tabellen>  
[where <Bedingungen>]  
group by <Spalten>  
having <Bedingungen>;
```

- üblicherweise in Kombination mit *group by*-Klausel (ohne ist ganze Tabelle eine Gruppe)
- filtert Gruppen aus der gruppierten Relation aus
- Bedingungen dürfen enthalten:
  - Spalten aus der *group by*-Klausel
  - Aggregatfunktionen auf den restlichen Spalten der Tabelle

# Gruppierung mit Having - Beispiel

```
select B, sum(D) as Summe  
from R  
group by A,B  
having max(C) < 5;
```

R

A	B	C	D
a	b	1	2
a	b	3	4
b	c	1	3
c	c	4	2
c	c	7	5

Gruppierung:

A	B		
		C	D
a	b	1	2
		3	4
b	c	1	3
c	c	4	2
		7	5

Endergebnis:

B	Summe
b	6
c	3

## Gruppierung mit Having - Beispiel

Studi(MatrNr, Vorname, Nachname, Fach, Semester)   hoert(MatrNr, VID)  
Vorlesung(ID, Titel, CP, gehalten\_von)   Prof(ID, Vorname, Nachname)

**Anfrage:** Geben Sie für alle Vorlesungen, an denen mindestens 20 Studis teilnehmen, die CP-Anzahl aus.

```
select CP  
from Vorlesung, hoeren  
where VID = ID  
group by VID, CP  
having count(MatrNr) >= 20;
```

# null-Werte

*null* steht für einen unbekannten Wert

*unique*-Integritätsbedingung

- mehrere *null*-Werte verletzen nicht *unique*-Bedingung

Vergleiche

- Jeder Vergleich (<, =, <>, ...) ergibt *unknown* (anstatt *true/false*)  
→ dreiwertige Logik

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

not	
true	false
unknown	unknown
false	true

# null-Werte

## where/having-Klausel

- *unknown* wird bei Auswertung wie *false* behandelt
- eigenes Prädikat *<Spalte> is [not] null*

## Arithmetische Ausdrücke

- Berechnungen mit *null*-Wert ergeben immer *null*

## Aggregatfunktionen

- *null*-Werte werden ignoriert (außer bei *count(\*)*)

## Gruppierung

- *null*-Werte im Gruppierungsattribut werden als identisch betrachtet und bilden eigene Gruppe

A	B	C
a	1	null
null	3	4
null	1	3
a	4	2

group by A

A		
	B	C
a	1	null
	4	2
null	1	3
	3	4



Unteranfrage = gültiger *select-from-where*-Block

- in *where*-Klausel
  - mit Vergleichsoperator
  - mit *some/any* oder *all*
  - mit *in* oder *not in*
  - mit *exists* oder *not exists*
- in *from*-Klausel

## in where-Klausel

*where a  $\theta$  (R)*      bzw.      *where <Ausdruck> <Vergleichsoperator> (<Resultat>)*

- **Ausdruck a:** ein Attribut oder eine Konstante (mehrere Attribute nur unter Einschränkungen)
- **Vergleichsoperator  $\theta$ :**  $<$ ,  $<=$ ,  $>=$ ,  $>$ ,  $=$ ,  $<>$
- **Resultat R** der Unteranfrage
  - muss mit Ausdruck a *kompatibel* sein  
→ gleich viele Spalten, passender Wertebereich (Zeichenketten/Zahlen)
  - üblicherweise mit Aggregatfunktion

 Unteranfrage darf nur eine Zeile zurückgeben

## Geschachtelte Anfragen - Beispiel

Studi(MatrNr, Vorname, Nachname, Fach, Semester) hoert(MatrNr, VID)  
Vorlesung(ID, Titel, CP, gehalten\_von) Prof(ID, Vorname, Nachname)

**Anfrage:** Geben Sie die IDs aller Vorlesungen mit der höchsten CP-Anzahl aus.

```
select ID  
from Vorlesung  
where CP = (select max(CP)  
           from Vorlesung);
```

## in where-Klausel

*where a  $\theta$  all (R)*

*where a  $\theta$  some | any (R)*

- Ausdruck a, Resultat R und Vergleichsoperator  $\theta$  wie auf Folie 73.
- **a  $\theta$  some (R)**: wahr, wenn mindestens ein Tupel t in R existiert, sodass A  $\theta$  t gilt.  
→ Existenzquantor
  - *some* und *any* sind gleichbedeutend (ggf. Dialekt)
- **a  $\theta$  all (R)**: wahr, wenn für alle Tupel t in R gilt, dass A  $\theta$  t erfüllt ist.  
→ Allquantor

## Geschachtelte Anfragen - Beispiel

Studi(MatrNr, Vorname, Nachname, Alter, Fach, Semester) Prof(ID, Vorname, Nachname, Alter)

**Anfrage:** Geben Sie die Namen der Profs aus, die älter sind als alle Studis.

```
select Vorname, Nachname  
from Prof  
where Alter >all (select Alter from Studi);
```

**Anfrage:** Geben Sie die Namen der Studis aus, die älter sind als irgendein Prof.

```
select Vorname, Nachname  
from Studi  
where Alter >some (select Alter from Prof);
```

## in where-Klausel

*where* a [not] in (R)

- Ausdruck a und Resultat R wie auf Folie 73.
- *a in (R)*: wahr, wenn mindestens ein Tupel in R gleich a ist
- *a not in (R)*: wahr, wenn kein Tupel in R gleich a ist  
→ Anfragen mit Differenz

# Geschachtelte Anfragen

- *in* und *=some* sind äquivalent

```
select Nachname  
from Studi S  
where MatrNr =some (select MatrNr  
    from hoert);
```

```
select Nachname  
from Studi  
where MatrNr in (select MatrNr  
    from hoert);
```

- *not in* und *<>all* sind äquivalent

```
select Nachname  
from Studi  
where MatrNr <>all (select MatrNr  
    from hoert);
```

```
select Nachname  
from Studi  
where MatrNr not in (select MatrNr  
    from hoert);
```

## Geschachtelte Anfragen - Beispiel

Studi(MatrNr, Vorname, Nachname, Fach, Semester)    hoert(MatrNr, VID)  
Vorlesung(ID, Titel, CP, gehalten\_von)    Prof(ID, Vorname, Nachname)

**Anfrage:** Geben Sie die Nachnamen aller Studis aus, die die Vorlesung mit der ID 15 hören.

```
select Nachname  
from Studi S, hoert H  
where S.MatrNr = H.MatrNr  
      and H.VID = 15;
```

```
select Nachname  
from Studi S  
where S.MatrNr in (select H.MatrNr  
                  from hoert H  
                  where H.VID = 15);
```



## Geschachtelte Anfragen - Beispiel

Studi(MatrNr, Vorname, Nachname, Fach, Semester)   hoert(MatrNr, VID)  
Vorlesung(ID, Titel, CP, gehalten\_von)   Prof(ID, Vorname, Nachname)

**Anfrage:** Geben Sie die Nachnamen aller Studis aus, die die Vorlesung mit der ID 15 **nicht** hören.

```
select Nachname  
from Studi S  
where S.MatrNr not in (select H.MatrNr  
          from hoert H  
          where H.VID = 15);
```

## in where-Klausel

*where* [not] exists (R)

- Resultat R: beliebige Unteranfrage
- *exists* (R): wahr, wenn R mindestens 1 Tupel enthält.
- *not exists* (R): wahr, wenn R leer ist.

## Geschachtelte Anfragen - Beispiel

Studi(MatrNr, Vorname, Nachname, Fach, Semester) hoert(MatrNr, VID)  
Vorlesung(ID, Titel, CP, gehalten\_von) Prof(ID, Vorname, Nachname)

**Anfrage:** Geben Sie die Nachnamen aller Profs aus, die eine Vorlesung halten.

```
select Nachname
from Prof
where exists (select *
              from Vorlesung
              where gehalten_von = Prof.ID);
```

- In inneren Anfragen können Attribute der Tabellen der *from*-Klausel der äußeren Anfrage verwendet werden!

## Geschachtelte Anfragen - Beispiel

Studi(MatrnNr, Vorname, Nachname, Fach, Semester) hoert(MatrnNr, VID)  
Vorlesung(ID, Titel, CP, gehalten\_von) Prof(ID, Vorname, Nachname)

**Anfrage:** Geben Sie die MatrNr aller Studis aus, die **ausschließlich** Vorlesungen hören, die 5 CPs bringen.

→ Studis, die Vorlesungen hören, aber **keine** Vorlesung hören, die **nicht** 5 CP bringt.

```
select MatrNr
from hoert
where not exists (select *
                  from Vorlesung
                  where VID = ID
                  and CP <> 5);
```

## Geschachtelte Anfragen - Beispiel

**Anfrage:** Geben Sie die MatrNr aller Studis aus, die **alle** Vorlesungen gehört haben.  
→ es gibt **keine** Vorlesung, die dieser Studi **nicht** gehört hat.

**Relationenalgebra:**  $hoert \div \rho_{ID \rightarrow VID}(\pi_{ID}(Vorlesung))$

```
select S.MatrNr
from Studi S
where not exists (select *
  from Vorlesung
  where not exists (select *
    from hoert H
    where H.VID = V.ID
    and H.MatrNr = S.MatrNr));
```

Alternative (⚠ funktioniert nur unter Einhaltung bestimmter Integriätsbedingungen!)

```
select MatrNr
from hoert
group by MatrNr
having count(*) =
  (select count(*)
   from Vorlesungen);
```

## in from-Klausel

- SQL-Anfrage (*select-from-where*-Block) anstelle eines Tabellennamens

Mitarbeiterin(ID, Name, Gehalt)    Abteilung(Name, Leitung, Budget)

**Anfrage:** Geben Sie die Namen aller Mitarbeiter:innen aus, deren Gehalt höher ist als das durchschnittliche Budget aller Abteilungen.

```
select Mitarbeiterin.Name  
from Mitarbeiterin, (select avg(Budget) as AvgBudget from Abteilung) as tmp  
where Mitarbeiterin.Gehalt > tmp.AvgBudget;
```

```
<select-from-where-Block 1>  
union|intersect|except  
<select-from-where-Block 2>
```

- Attribute beider Blöcke müssen *kompatibel* sein  
→ gleich viele Spalten, passender Wertebereich (Zeichenketten/Zahlen)
- Resultat hat Attributnamen des ersten Blocks
- keine Duplikate (doppelte Zeilen) im Ergebnis
- *except* und *intersect* durch geschachtelte Anfragen in *where*-Klausel ersetzbar

## Data Manipulation Language (DML): Manipulationssprache

- Tupel einfügen (*insert into*)
- Tupel löschen (*delete from*)
- Tupel ändern (*update*)

⚠ Alle Operationen nur auf Basisrelationen möglich



```
insert into <Basisrelation> [(<Attributliste>)]  
values <Tupel> [, <Tupel2>, ...];
```

- Fügt ein oder mehrere Tupel in die Tabelle ein
- Tupel muss zum Schema/Tabellendefinition passen
- Beim Einfügen werden Integritätsbedingungen sichergestellt
- Attributliste:
  - nicht nötig bei vollständigen Tupeln
  - hilfreich bei unklarer Reihenfolge
  - unbedingt nötig bei unvollständigen Tupeln  
→ fehlende Werte werden mit *null* aufgefüllt

## Tupel einfügen - Beispiel

```
Studi(MatrNr, Vorname, Nachname, Fach, Semester)  hoert(MatrNr, VID)  
Vorlesung(ID, Titel, CP, gehalten_von)  Prof(ID, Vorname, Nachname)
```

**Anfrage:** Fügen Sie hinzu, dass die Studis mit der MatrNr 123 und MatrNr 456 die Vorlesung mit der ID 42 hört.

```
insert into hoert  
values (123,42), (456,42);
```

```
insert into hoert (VID,MatrNr)  
values (42,123), (42,456);
```

- Attributliste nicht nötig bei vollständigen Tupeln
- Attributliste hilfreich bei unklarer Reihenfolge

## Tupel einfügen - Beispiel

```
Studi(Matrn̄r, Vorname, Nachname, Fach, Semester)  hoert(Matrn̄r, VID)  
Vorlesung(ID, Titel, CP, gehalten_von)  Prof(ID, Vorname, Nachname)
```

**Anfrage:** Fügen Sie eine Vorlesung mit den Titel Datenbanken hin, die vom Prof mit der ID 5 gehalten wird. Gehen Sie davon aus, dass die ID automatisch generiert wird.

```
insert into Vorlesung (Titel, gehalten_von)  
values ('Datenbanken',5);
```

- Attributliste **unbedingt nötig** bei unvollständigen Tupeln  
→ fehlende Werte werden mit *null* aufgefüllt  
(oder ggf. default-Werte oder auto-generierte Werte)

```
delete from <Basisrelation>  
[where <Bedingung>];
```

- ohne *where*-Klausel:
  - löscht alle Tupel in der Tabelle
- mit *where*-Klausel:
  - löscht alle Tupel in der Tabelle, welche die Bedingung erfüllen
  - übliche *where*-Klauseln mit geschachtelten Anfragen etc. möglich
- *delete* vs. *drop* (Folie 20):
  - *delete from R*: löscht nur Inhalt, aber Schema der Tabelle bleibt erhalten
  - *drop table R*: löscht Tabelle inklusive Inhalt und Schema

## Tupel löschen - Beispiel

Studi(MatrNr, Vorname, Nachname, Fach, Semester)   hoert(MatrNr, VID)  
Vorlesung(ID, Titel, CP, gehalten\_von)   Prof(ID, Vorname, Nachname)

**Anfrage:** Löschen Sie den Studi mit der MatrNr 123.

```
delete from Studi  
where MatrNr = 123;
```

**Anfrage:** Löschen Sie alle Vorlesungen, die von niemanden gehört werden.

```
delete from Vorlesung  
where ID not in(select VID from hoert);
```

```
update <Basisrelation>  
set <Attribut> = <Ausdruck> [, <Attribut2> = <Ausdruck2>, ...]  
[where <Bedingung>];
```

- Ohne *where*-Bedingung werden alle Tupel geändert
- mögliche Änderungen:
  - Attribut auf konstanten Wert oder Wert eines anderen Attributs setzen
  - neuen Attributwert aus anderen Attributwerten oder Konstanten berechnen
  - geschachtelte Anfragen in Änderungen möglich
- mehrere Änderungen gleichzeitig möglich

## Tupel ändern - Beispiel

Studi(MatrNr, Vorname, Nachname, Fach, Semester)   hoert(MatrNr, VID)  
Vorlesung(ID, Titel, CP, gehalten\_von)   Prof(ID, Vorname, Nachname)

**Anfrage:** Erhöhen Sie die Anzahl der Semester aller Studis um 1.

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

**Anfrage:** Setzen Sie die CP-Anzahl der Vorlesung mit der ID 15 auf die höchste vorhandene CP-Anzahl.

```
update Vorlesung  
set CP = (select max(CP) from Vorlesung V)  
where ID = 15;
```