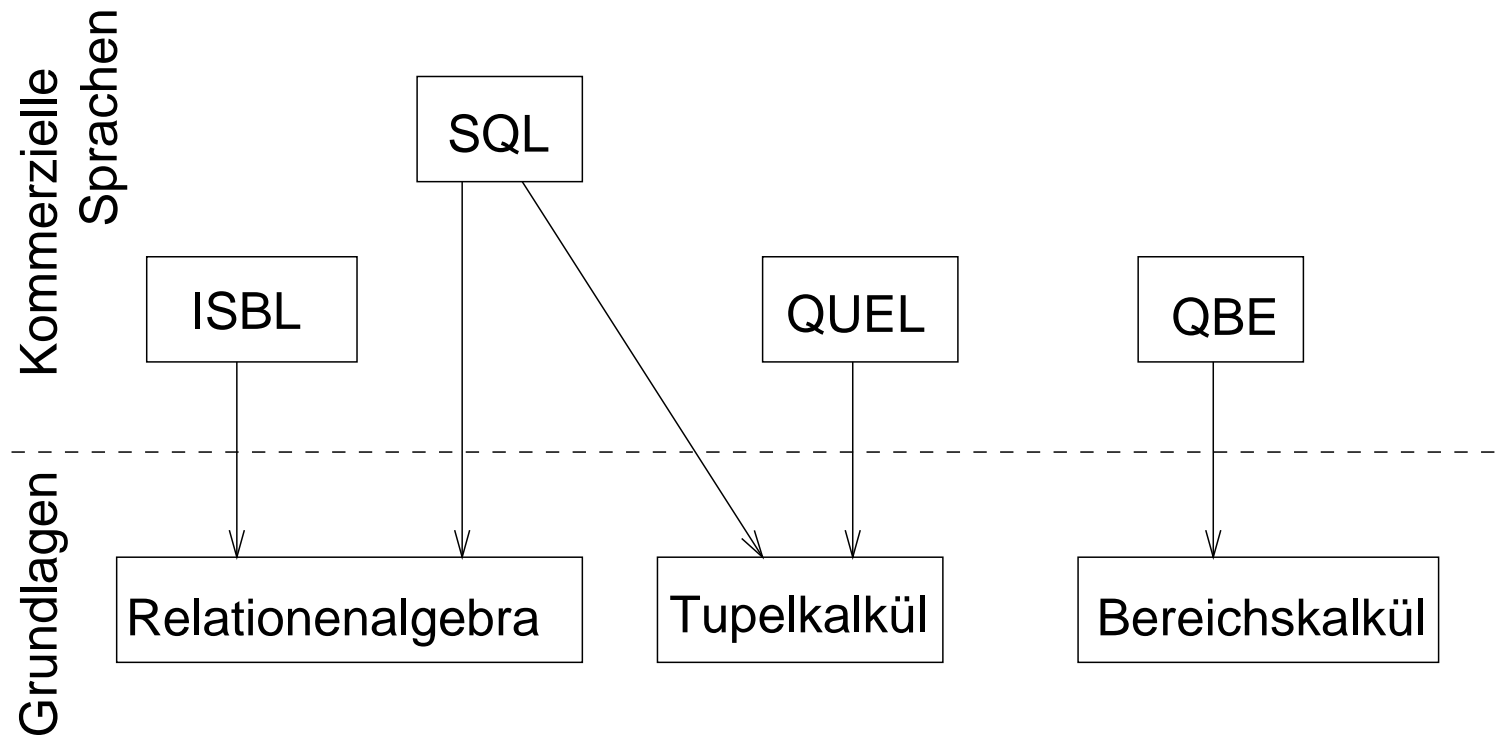


# Relationale Datenbanksprachen

- ▣ SQL-Kern
- ▣ Weitere Sprachkonstrukte von SQL
- ▣ SQL-Versionen

# Sprachen und ihre Grundlagen



# SQL-Kern

## **select**

- Projektionsliste
- *arithmetische Operationen und Aggregatfunktionen*

## **from**

- zu verwendende Relationen
- eventuelle Umbenennungen (durch Tupelvariable oder 'alias')

## **where**

- Selektionsbedingungen
- Verbundbedingungen
- Geschachtelte Anfragen (wieder ein SFW-Block)

## **group by**

- *Gruppierung für Aggregatfunktionen*

## **having**

- *Selektionsbedingungen an Gruppen*

# from-Klausel

## Syntax

```
select *  
from relationenliste
```

## Beispiel

```
select *  
from Bücher
```

liefert die gesamte Relation Bücher

# Kartesisches Produkt

- Bei mehr als einer Relation hinter **from**: kartesisches Produkt

```
select * from Bücher, Ausleih
```

Einführung von Tupelvariablen: etwa auf eine Relation mehrfach zugreifen

```
select * from Bücher eins, Bücher zwei
```

- Ergebnis hat acht Spalten:

```
eins.Inventarnr, eins.Titel, eins.ISBN, eins.Autor,  
zwei.Inventarnr, zwei.Titel, zwei.ISBN, zwei.Autor
```

- *Selbst-Verbund* (Self-Join) für tupelübergreifende Selektionen

## Bezug zum Tupelkalkül

korrespondierende Kalkülausdrücke automatisch sicher

```
select ...  
from Bücher eins, Bücher zwei  
where ...
```

$$\{\dots \mid \text{Bücher}(eins) \wedge \text{Bücher}(zwei) \wedge (\dots)\}$$

## SQL-92-Spezialitäten

- Verbunde als explizite Operatoren

- kartesisches Produkt

```
select * from Bücher, Ausleih
```

```
select * from Bücher cross join Ausleih
```

- Verbund über Verbundbedingungen

```
select * from Bücher, Ausleih
```

```
where Bücher.Inventarnr = Ausleih.Inventarnr
```

- **join**-Operator:  $\theta$ -Verbund

```
select * from Bücher join Ausleih
```

```
on Bücher.Inventarnr = Ausleih.Inventarnr
```

## Weitere Verbunde in SQL-92

- Gleichverbund

```
select * from Bücher join Ausleih  
                using (Inventarnr)
```

- natürlicher Verbund

```
select * from Bücher natural join Ausleih
```

- jeder SFW-Block hinter **from** (SQL-92 orthogonal)

# Äußere Verbunde

Statt **inner join** nun **outer join** (dangling tuples übernehmen und mit Nullwerten auffüllen)

- **full outer join**: in beiden Operanden
- **left outer join**: im linken Operanden
- **right outer join**: im rechten Operanden

# Äußere Verbunde II

LINKS

A	B
1	2
2	3

RECHTS

B	C
3	4
4	5

NATURAL JOIN

A	B	C
2	3	4

OUTER

A	B	C
1	2	⊥
2	3	4
⊥	4	5

LEFT

A	B	C
1	2	⊥
2	3	4

RIGHT

A	B	C
2	3	4
⊥	4	5

## Die select-Klausel

Relationenalgebra: abschließende Projektion

Relationenkalkül: Zielliste

```
select [distinct] {attribut | arithmetischer-ausdruck |  
                    aggregat-funktion }  
from ...
```

- Attribute aus **from**-Relationen
- Arithmetische Ausdrücke über Attributen und Konstanten
- Aggregatfunktionen über Attributen

**distinct**: Ergebnismenge statt Multimenge

# Projektionsergebnis Menge oder Multimenge

```
select Name from Ausleih
```

<b>Name</b>
Meyer
Schulz
Müller
Meyer

```
select distinct Name from Ausleih
```

<b>Name</b>
Meyer
Schulz
Müller

## Tupelvariablen und Relationennamen

Angabe der Attributnamen durch Präfix ergänzen

```
select ISBN from Bücher
```

und

```
select Bücher.ISBN from Bücher
```

Tupelvariable kann benutzt werden:

```
select eins.ISBN, zwei.Titel  
from Bücher eins, Bücher zwei
```

## Tupelvariablen und Relationennamen II

```
select ISBN, Titel, Stichwort      (falsch!)  
from Bücher, Buch_Stichwort  
where Bücher.ISBN = Buch_Stichwort.ISBN
```

```
select Bücher.ISBN, Titel, Stichwort  (richtig)  
from Bücher, Buch_Stichwort  
where Bücher.ISBN = Buch_Stichwort.ISBN
```

## Bezug zum Tupelkalkül

**select:** Zielliste im Tupelkalkül

Letzte Anfrage entspricht

$$\{ \textit{eins.ISBN}, \textit{zwei.Titel} \mid \text{Bücher}(\textit{eins}) \wedge \text{Bücher}(\textit{zwei}) \}$$

# Die where-Klausel

Selektionsbedingung der Relationenalgebra oder Verbundbedingung

**select ... from ... where** bedingung

Bedingung:

- *Konstanten-Selektion*

attribut  $\theta$  konstante

- *Attribut-Selektion* zwischen Attributen mit kompatiblen Wertebereichen:

attribut1  $\theta$  attribut2

## Verbundbedingung

`relation1.attribut = relation2.attribut`

Beispiel: natürlicher Verbund

```
select Bücher.Titel, Bücher_Stichwort.Stichwort
from Bücher, Buch_Stichwort
where Bücher.ISBN = Buch_Stichwort.ISBN
```

auch Gleichverbund und  $\theta$ -Verbund erlaubt

# Bereichsselektion

attribut **between** konstante1 **and** konstante2

Abkürzung für

attribut  $\geq$  konstante1 **and** attribut  $\leq$  konstante2

Beispiel

```
select Matrikelnummer from Prüft  
where Note between 1.0 and 2.0
```

# Ungewißheitsselektion

- theoretisch nur Abkürzung für disjunktiv verknüpfte Bedingung
- Syntax
  - attribut **like** spezialkonstante
- Spezialkonstante kann beinhalten
  - ◆ '%' (kein oder beliebig viele Zeichen)
  - ◆ '\_' (genau ein Zeichen)

## Ungewißheitsselektion II

Anwendung: Selektion nach Büchern von Benjamin/Cummings

```
select * from Bücher where Verlagsname like 'Benj%Cummings%'
```

ist Abkürzung für

```
select * from Bücher  
where Verlagsname = 'Benjamin Cummings'  
      or Verlagsname = 'Benjamin/Cummings'  
      or Verlagsname = 'Benjamin-Cummings'  
      or Verlagsname = 'Benjamin and Cummings'  
      or Verlagsname = 'BenjXFDGYWCummingsSCHlumpf'  
      or ...
```

## Weitere Bedingungen

- *Null-Selektion*

attribut **is null**

- *Quantifizierte Bedingungen*, wenn ein Argument in Vergleich Menge liefert (**all**, **any**, **some** und **exists**)
- boolesche Ausdrücke mit Konnektoren **or**, **and** und **not**

## Bezug zum Tupelkalkül

- **where**-Klausel: qualifizierende Formel in Tupelkalkülanfragen

```
select Bücher.Titel, Buch_Stichwort.Stichwort
from Bücher, Buch_Stichwort
where Bücher.ISBN = Buch_Stichwort.ISBN and
      ( Verlagsname = 'Thomson' or Verlagsname = 'ITP' );
```

- entspricht im Tupelkalkül

$$\{ \quad b.\text{Titel}, s.\text{Stichwort} \mid \text{Bücher}(b) \wedge \text{Buch\_Stichwort}(s) \wedge \\ b.\text{ISBN} = s.\text{ISBN} \wedge \\ (b.\text{Verlagsname} = \text{'Thomson'} \vee b.\text{Verlagsname} = \text{'ITP'}) \}$$

# Schachtelung von Anfragen

- **where**-Klausel kann geschachtelt werden
- SFW-Blöcke liefern im allgemeinen mehrere Werte
- Vergleiche mit Wertemengen
  - ◆ Standardvergleiche in Verbindung mit Quantoren **all** ( $\forall$ ) oder **any** ( $\exists$ )
  - ◆ spezielle Prädikate für den Zugriff auf Mengen **in** und **exists**

# Das in-Prädikat und geschachtelte Anfragen

- Syntax:

```
attribut in ( SFW-block )
```

- Beispiel:

```
select Titel from Bücher  
where ISBN in ( select ISBN from Empfiehlt )
```

- natürlicher Verbund mit nachfolgender Projektion

# Das in-Prädikat und geschachtelte Anfragen II

## ■ Abarbeitung

1. Ergebnis der inneren **select**-Anweisung hinter **in** als Liste von Konstanten einsetzen

2. Dann modifizierte Anfrage

```
select Titel from Bücher
```

```
where ISBN in
```

```
( '3-929821-31-1', '0-201-53771-0',  
  '3-89319-175-5', '0-8053-1753-8' )
```

abearbeiten

## Verzahnt geschachtelte Anfragen

- in der inneren Anfrage Relationen- oder Tupelvariablen-Name aus dem **from**-Teil der äußeren Anfrage verwenden

```
select Nachname
from Personen
where 1.0 in ( select Note
                 from Prüft
                 where PANr = Personen.PANr)
```

## Verzahnt geschachtelte Anfragen II

### ■ Abarbeitung

1. In der äußeren Anfrage das erste Personen-Tupel untersuchen  
Ergebnis in innere Anfrage einsetzen
2. innere Anfrage  
**select** Note  
**from** Prüft  
**where** PANr = 4711  
auswerten, liefert Werteliste ( 2.0, 2.3 )
3. Ergebnis der inneren Anfrage in die äußere einsetzen  
1.0 **in** ( 2.0, 2.3 ) ergibt **false**  
ersten Prüfer nicht berücksichtigen
4. in der äußeren Anfrage das zweite Personen-Tupel untersuchen usw.

## Das exists-Prädikat

- testet, ob Ergebnis der inneren Anfrage nicht leer

```
select ISBN
from Buch_Exemplare
where exists
    ( select *
      from Ausleihe
      where Inventarnr = Buch_Exemplare.Inventarnr)
```

## exists: Simulation des Allquantors

```
select Lehrstuhlbezeichnung
from Professoren
where not exists
  ( select * from Liest
    where Liest.PANr = Professoren.PANr
    and not exists ( select *
                     from Prüft
                     where Prüft.PANr =
                           Professoren.PANr
                     and Prüft.V_Bezeichnung =
                           Liest.V_Bezeichnung))
```

## Bezug zum Tupelkalkül

- **exists**-Prädikat:  $\exists$ -Quantor des Tupelkalküls
- andere Schachtelungsoperatoren ebenfalls auf Quantoren zurückführen
- $\forall$ -Quantor mit  $\forall\varphi \equiv \neg\exists\neg\varphi$  simulieren

## SQL-92: Tupelbildungen

- row constructors bilden Tupel aus Konstanten oder Attributen  $(e_1, \dots, e_n)$

```
where ( select Studienfach, Immatrikulationsdatum
from Studenten
where Matrikelnummer = 'HR0-912291')
=
('Informatik', '1.10.91')
```

- Attribute müssen *kompatibel* sein (siehe unten)

$$(a_1, \dots, a_n) < (b_1, \dots, b_n)$$

wahr, wenn ein  $j$  existiert, für das  $a_j < b_j$  und  $a_i = b_i$  für alle  $i < j$  gilt  
(lexikographische Ordnung)

# Kompatible Attribute

- Attribute sind kompatibel bei kompatiblen Wertebereichen
- Zwei Wertebereiche sind kompatibel, wenn sie
  - ◆ gleich sind oder
  - ◆ beides auf `character` basierende Wertebereiche sind (unabhängig von der Länge der Strings) oder
  - ◆ beides numerische Wertebereiche sind (unabhängig von dem genauen Typ) wie `integer` oder `float`)
- Kompatible Attribute können in Vergleichen und Mengenoperationen benutzt werden

## SQL-89: Vereinigung

- SQL-89: Vereinigung **union** einzige Mengenoperation

SFW\_block1 **union** SFW\_block2

- Beispiel:

**select** A, B, C **from** R1 **union** **select** A, C, D **from** R2

*Attributkompatibilität:* A von R1 und A von R2,  
B von R1 und C von R2, C von R1 und D von R2

- Ergebnis: Attributnamen des linken Operanden
- Vereinigung nur als “äußerste” Operation erlaubt.

## Simulation der Differenz

$\pi[\text{PANr}] (\text{Mitarbeiter}) - \pi[\text{PANr}] (\text{Studenten})$

in SQL

```
select PANr from Mitarbeiter  
where PANr not in ( select PANr  
                     from Studenten )
```

## Vereinigung und äußere Verbunde

```
select * from Personen left outer natural join Pers_Telefon
```

umgesetzt

```
select P.PANr, P. Vorname, P.Nachname, P.PLZ,  
       P.Ort, P.Straße, P.HNr, P.Geburtsdatum, T. Telefon  
from Personen P, Pers_Telefon T where P.PANr = T. PANr
```

**union**

```
select P.PANr, P. Vorname, P.Nachname, P.PLZ,  
       P.Ort, P.Straße, P.HNr, P.Geburtsdatum, null  
from Personen P  
where not exists ( select *  
                  from Pers_Telefon T  
                  where P.PANr = T.PANr )
```

## Vereinigung, Durchschnitt und Differenz in SQL-92

**union**, **intersect** und **except** orthogonal in andere Anfragen einsetzbar

```
select count(*)  
from ( (select PANr from Professoren)  
      union  
      (select PANr from Studenten) )
```

**corresponding**-Klausel: zwei Relationen nur über ihren gemeinsamen Bestandteilen vereinigen

```
select count(*)  
from ( Professoren union corresponding Studenten) )
```

## Vergleich Relationenalgebra und SQL

Relationenalgebra	SQL-89	SQL-92
Projektion	<b>select distinct</b>	<b>select distinct</b>
Selektion	<b>where</b> ohne Schachtelung	<b>where</b> ohne Schachtelung
Verbund	<b>from, where</b>	<b>from, where</b> <b>from</b> mit <b>join</b> oder <b>natural join</b>
Umbenennung	<b>from</b> mit Tupelvariable	<b>from</b> mit Tupelvariable <b>as</b>
Differenz	<b>where</b> mit Schachtelung	<b>where</b> mit Schachtelung <b>except corresponding</b>
Durchschnitt	<b>where</b> mit Schachtelung	<b>where</b> mit Schachtelung <b>intersect corresponding</b>
Vereinigung	<b>union</b> (nicht orthogonal)	<b>union corresponding</b>

## Weitere Sprachkonstrukte von SQL

- Operationen auf Wertebereichen
- Aggregatfunktionen
- **group by** und **having**
- Quantoren und Mengenvergleiche
- Beispiele für Selbst-Verbund
- **order by**
- Nullwerte
- Änderungs-Operationen

## Operationen auf Wertebereichen

- innerhalb von **select** und **where**: statt Attribute auch *skalare Ausdrücke*
  - ◆ numerischen Wertebereiche: etwa +, −, \*, /
  - ◆ Strings: **char\_length**, Konkatenation ||, **substring** (Teilzeichenkette)
  - ◆ Datumstypen, Zeitintervalle: **current\_date**, **current\_time**, +, −, \*
- Ausdrücke werden tupelweise ausgewertet

**select** ISBN, Preis / 1.44 **from** Buch\_Versionen

Ergebnis

ISBN	
3-89319-175-5	54,86
0-8053-1753-8	50,24
0-8053-1753-8	61,70
0-201-53771-0	60,73
3-929821-31-1	54,86

## SQL-92-Spezialitäten

Bsp.: zweite Spalte nicht benannt, in SQL-89 über Spaltennummer identifizierbar:

```
select 2 from Ergebnis
```

in SQL-92 : Attributname zuordnen:

```
select ISBN, Preis ÷ 1.44 as Dollar_Preis  
from Buch_Versionen
```

# Aggregatfunktionen

- built-in-Funktionen: tupelübergreifend
  - ◆ **count**: Anzahl der Werte einer Spalte oder (Spezialfall **count(\*)**) Anzahl der Tupel einer Relation
  - ◆ **sum**: Summe der Werte einer Spalte
  - ◆ **avg**: arithmetisches Mittel der Werte einer Spalte
  - ◆ **max** bzw. **min**: größter bzw. kleinster Wert einer Spalte
- Argumente einer Aggregatfunktion:
  - ◆ Attribut der durch **from** spezifizierten Relation
  - ◆ gültiger skalarer Ausdruck
  - ◆ bei **count** auch \*
- Vor Argument (außer bei **count(\*)**) optional: **distinct** oder **all** (**all** Voreinstellung)
- Nullwerte werden vor Anwendung aus Wertemenge eliminiert (außer bei **count(\*)**)

## Aggregatfunktionen: Beispiele

```
select sum(Preis) from Buch_Versionen
```

```
select count(*) from Professoren
```

```
select count(distinct PANr) from Prüft
```

```
select avg(all Note) from Prüft  
where V_Bezeichnung = 'Datenbanken I'
```

~> **all** notwendig

```
select Matrikelnummer from Prüft  
where Note < ( select avg (all Note) from Prüft )
```

## group by und having

### ■ Syntax

```
select ... from ... [ where ... ]  
[ group by attributliste ]  
[ having bedingung ]
```

### ■ Semantik (virtuelle geschachtelte Relation):

- ◆ Relationenschema  $R$  und Attributmenge hinter Gruppierung  $G$
- ◆ schachteln nach Attributen  $R - G$ , d.h. für gleiche  $G$ -Werte werden Resttupel in Relation gesammelt
- ◆ der **where**-Klausel genügende Tupel also schachteln gemäß

$$\nu[(R - G; N)](r(R))$$

## group by und having II

- **having** ist Selektionsbedingung auf gruppierter Relation
- darf Bezug nehmen auf
  - ◆ Gruppierungsattribute
  - ◆ beliebige Aggregatfunktionen über Nicht-Gruppierungsattributen

# Gruppierung: Schema

Schritt 1:	A	B	C	D	<b>from und where</b>
	1	2	3	4	
	1	2	4	5	
	2	3	3	4	
	3	3	4	5	
	3	3	6	7	
Schritt 2:	A	B	N		<b>group by A, B</b>
			C	D	
	1	2	3	4	
			4	5	
	2	3	3	4	
	3	3	4	5	
Schritt 3:	A	sum(D)	N		<b>select A, sum(D)</b>
			C	D	
	1	9	3	4	
			4	5	
	2	4	3	4	
	3	12	4	5	
Schritt 4:	A	sum(D)	<b>having A &lt; 4 and sum(D) &lt; 10 and max(C) = 4</b>		
	1	9			

## Gruppierung: Beispiele

```
select count(*) as Anzahl, PANr  
from Ausleihe  
group by PANr
```

Anzahl	PANr
2	7754
1	4711
1	5588
2	9912

## Gruppierung: Beispiele II

```
select count(*), Name from Ausleihe  
group by Name  
having count(*) > 1
```

	PANr
2	7754
2	9912

```
select Matrikelnummer from Prüft  
group by Matrikelnummer  
having avg(Note) < (select avg(Note) from Prüft)
```

# Quantoren und Mengenvergleiche

- Syntax

attribut  $\theta$  { **all** | **any** | **some** } ( **select** attribut  
from ... **where** ... )

- Bedeutung **all** Allquantor, **any**, **some** Existenzquantoren

- Beispiele

```
select PANr, Immatrikulationsdatum  
from Studenten  
where Matrikelnummer = any ( select Matrikelnummer  
from Prüft )
```

## Quantoren und Mengenvergleiche II

```
select Note from Prüft
where Matrikelnummer = 'HR0-912291'
and   Note  $\geq$  all ( select Note from Prüft
                       where Matrikelnummer = 'HR0-912291' )
```

Anwendbarkeit eingeschränkt: Test auf Mengen-Gleichheit

$$\forall x \in M_1 : \exists x \in M_2 \wedge \forall x \in M_2 : \exists x \in M_1$$

in SQL so nicht umsetzbar:

*Gib alle Bücher aus, an denen 'Vossen' und 'Witt' gemeinsam als Autoren beteiligt waren*

## Selbst-Verbund

- letzte Anfrage erst mit Selbst-Verbund zu lösen
- Vergleich von Wertemengen

```
select B_A_1.ISBN from Buch_Autor B_A_1, Buch_Autor B_A_2
where B_A_1.ISBN = B_A_2.ISBN
      and B_A_1.Autor = 'Vossen' and B_A_2.Autor = 'Witt'
```

B_A_1.ISBN	B_A_1.Autor	B_A_2.ISBN	B_A_2.Autor
3-89319-175-5	Vossen	3-89319-175-5	Vossen
3-89319-175-5	Vossen	3-89319-175-5	Witt
3-89319-175-5	Vossen	0-8053-1753-8	Elmasri
...			
3-89319-175-5	Witt	3-89319-175-5	Vossen
...			

## Selbst-Verbund II

- Zählen von Wertemengen

```
select distinct X.PANr  
from Prüft X, Prüft Y  
where X.PANr = Y.PANr  
and X.Matrikelnummer <> Y.Matrikelnummer
```

## order by-Klausel

- Menge von Tupeln  $\rightsquigarrow$  Liste

- Syntax

**order by** attributliste

- Beispiel

```
select Matrikelnummer, Note  
from Prüft  
where V_Bezeichnung = 'Datenbanken I'  
order by Note asc
```

- aufsteigend (**asc**) oder absteigend (**desc**) sortieren

## order by-Klausel II

- Sortierung wird auf das Ergebnis der jeweils vorangehenden SFW-Anfrage angewendet, also FALSCH:

```
select Matrikelnummer  
from Prüft  
where V_Bezeichnung = 'Datenbanken I'  
order by Note    (falsch!)
```

## Behandlung von Nullwerten

- skalare Ausdrücke: Ergebnis **null**, sobald Nullwert in die Berechnung eingeht
- In allen Aggregatfunktionen bis auf **count(\*)** werden Nullwerte vor Anwendung der Funktion entfernt
- Fast alle Vergleiche mit Nullwert ergeben Wahrheitswert **unknown** (statt **true** oder **false**). Ausnahme: **is null** ergibt **true**, **is not null** ergibt **false**
- Boolesche Ausdrücke basieren dann auf dreiwertiger Logik

## Behandlung von Nullwerten II

<b>and</b>	<b>true</b>	<b>unknown</b>	<b>false</b>
<b>true</b>	<b>true</b>	<b>unknown</b>	<b>false</b>
<b>unknown</b>	<b>unknown</b>	<b>unknown</b>	<b>false</b>
<b>false</b>	<b>false</b>	<b>false</b>	<b>false</b>

<b>or</b>	<b>true</b>	<b>unknown</b>	<b>false</b>
<b>true</b>	<b>true</b>	<b>true</b>	<b>true</b>
<b>unknown</b>	<b>true</b>	<b>unknown</b>	<b>unknown</b>
<b>false</b>	<b>true</b>	<b>unknown</b>	<b>false</b>

<b>not</b>	
<b>true</b>	<b>false</b>
<b>unknown</b>	<b>unknown</b>
<b>false</b>	<b>true</b>

# Änderungsoperationen

- Einfügen von Tupeln in Basisrelationen (oder Sichten) **insert**
- Löschen von Tupeln aus Basisrelationen (oder Sichten) **delete**
- Ändern von Tupeln in Basisrelationen (oder Sichten) **update**

Diese Operationen jeweils als

- Eintupel-Operationen (etwa die Erfassung einer neuen Ausleiherung)
- Mehrtupel-Operationen (erhöhe das Gehalt aller Mitarbeiter um 4.5%)

SQL: vor allem Mehrtupel-Operationen

Änderungsoperationen auf Sichten: später

# update

## ■ Syntax

**update** basisrelation

**set** attribut\_1 = ausdr\_1, ... , attribut\_n = ausdr\_n

[ **where** bedingung ]

## ■ Beispiele

Angestellte

Name	Gehalt
Meyer	3000
Schulz	3500
Bond	7200
Schulz	4400

**update** Angestellte **set** Gehalt = Gehalt + 1000

**where** Gehalt < 5000

## update II

Angestellte

Name	Gehalt
Meyer	4000
Schulz	4500
Bond	7200
Schulz	5400

**update** Angestellte **set** Gehalt = 6000 **where** Name = 'Bond'

**update** Angestellte **set** Gehalt = 3000

## **delete**

```
delete from basisrelation [ where bedingung ]
```

```
delete from Ausleihe where Invnr = 4711
```

Standardfall ist Löschen mehrerer Tupel:

```
delete from Ausleihe where Name = 'Meyer'
```

Löschen der gesamten Relation:

```
delete from Ausleihe
```

## insert

```
insert into basisrelation [ (attribut_1, ... , attribut_n) ]  
values (konstante_1, ... , konstante_n)
```

```
insert into Buch (Invnr, Titel) values (4867, 'Wissensbanken')
```

```
insert into Buch  
values (4867, 'Wissensbanken', '3-876', 'Karajan')
```

```
insert into basisrelation [ (attribut_1, ... , attribut_n) ]  
    SQL-anfrage
```

```
insert into Kunde (select LName, LAdr, 0 from Lieferant )
```

# SQL-Versionen

## ■ Geschichte

- ◆ SEQUEL (1974, IBM Research Labs San Jose)
- ◆ SEQUEL2 (1976, IBM Research Labs San Jose)
- ◆ SQL (1982, IBM)
- ◆ ANSI-SQL (SQL-86; 1986)
- ◆ ISO-SQL (SQL-89; 1989; drei Sprachen Level 1, Level 2, + IEF)
- ◆ (ANSI / ISO) SQL2 (SQL-92)
- ◆ (ANSI / ISO) SQL3 (geplant)

## ■ SQL

- ◆ DDL, (SSL,) IQL, DML
- ◆ Sichtdefinition, Transaktionsdefinition, Rechtevergabe, Integritätssicherung

## SEQUEL2

- bot mehr als SQL-89
  - ◆ **intersect** und **minus** neben **union**
- und sogar mehr als SQL-92
  - ◆ Mengenvergleiche mit **set** und zwei SFW-Blöcken
- Beispiele:

```
select Matrnr from Prüfungen X
where ( select Fach from Prüfungen where Matrnr = X.Matrnr)
      =
      ( select Fach from Prüfungen where Matrnr = 1034)
```

## SEQUEL2 II

- Statt = auch CONTAINS möglich

```
select Name from Studenten
where Matrnr in ( select Matrnr
                    from Prüfungen
                    where set(Prüfern) = (
                        select Prüfern
                        from Prüfungen ))
```

- **set** ermittelt alle Prüfern pro Matrnr

# SQL-89

- Level 1
  - ◆ keine Nullwerte
  - ◆ keine Selektionsbedingungen mit  $\neq$  oder **exists**
  - ◆ keine **union**-Operation
  - ◆ ...
- Level 2: wie hier beschrieben
- Level 2 + IEF (Integrity Enhancement Feature)
  - ◆ **check**-Klausel: **where**-Klausel als Integritätsbedingung
  - ◆ Definition von Primärschlüsseln und Fremdschlüsseln

## SQL-92

- neue Datentypen (wie `interval`)
- Domänenkonzept (**`create domain`**, **`alter domain`**)
- Änderung des Datenbankschemas: **`alter table`** und **`drop table`**
- allgemeine Integritätsbedingungen (mehrere Tabellen)
- `string`-Operationen erweitert
- Namen für abgeleitete Spalten
- **`join`**, **`cross join`**, **`natural join`**, **`outer join`** als eigene Operatoren
- auch **`intersect`** und **`except`**

## SQL-92 II

- Sprache fast vollständig orthogonal (etwa **union**, SFW hinter **from**)
- dreiwertige Logik
- **set transaction**: verschiedene Isolationsstufen (siehe Datenbanken II)
- Embedded SQL und Dynamic SQL sind Teil der Norm (siehe nächstes Kapitel)
- Data Dictionary ist Teil der Norm

## SQL-92 III

Feature in SQL-92	Entry	Intermediate	Full
Datum, Intervalltypen, <b>domain</b>	—	+	+
string-Operationen	—	+	+
<b>join</b>	—	+	+
<b>except, intersect</b>	—	+	+
<b>alter, drop table</b>	—	+	+
<b>set transaction</b>	—	+	+
Dynamic SQL	—	+	+
<b>union</b> orthogonal	—	+	+
andere Orthogonalitätsverbesserungen	—	+	+
<b>corresponding</b> bei Mengenoperationen	—	—	+
dreiwertige Logik	—	—	+
allgemeine Integritätsbedingungen	—	—	+
<b>check</b> mit Bezug zu anderen Tabellen	—	—	+
<b>alter domain</b>	—	—	+
Tabellenkonstruktoren	—	—	+