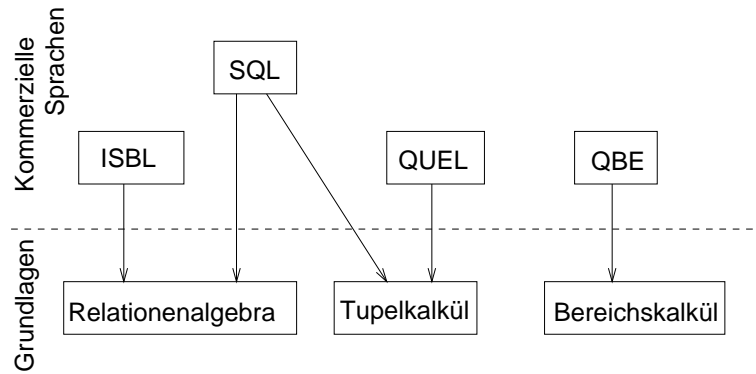


Sprachen und ihre Grundlagen



from-Klausel

Syntax

```
select *
from relationenliste
```

Beispiel

```
select *
from Bücher
```

liefert die gesamte Relation Bücher

Relationale Datenbanksprachen

- ➡ SQL-Kern
- ➡ Weitere Sprachkonstrukte von SQL
- ➡ SQL-Versionen

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*

Bezug zum Tupelkalkül

korrespondierende Kalkülausdrücke automatisch sicher

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

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

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)

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

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: θ -Verbund

```
select * from Bücher join Ausleih  
on Bücher.Inventarnr = Ausleih.Inventarnr
```

Äußere Verbunde II

LINKS	<table border="1"><tr><th>A</th><th>B</th></tr><tr><td>1</td><td>2</td></tr><tr><td>2</td><td>3</td></tr></table>	A	B	1	2	2	3	RECHTS	<table border="1"><tr><th>B</th><th>C</th></tr><tr><td>3</td><td>4</td></tr><tr><td>4</td><td>5</td></tr></table>	B	C	3	4	4	5	NATURAL JOIN	<table border="1"><tr><th>A</th><th>B</th><th>C</th></tr><tr><td>2</td><td>3</td><td>4</td></tr></table>	A	B	C	2	3	4												
A	B																																		
1	2																																		
2	3																																		
B	C																																		
3	4																																		
4	5																																		
A	B	C																																	
2	3	4																																	
OUTER	<table border="1"><tr><th>A</th><th>B</th><th>C</th></tr><tr><td>1</td><td>2</td><td>⊥</td></tr><tr><td>2</td><td>3</td><td>4</td></tr><tr><td>⊥</td><td>4</td><td>5</td></tr></table>	A	B	C	1	2	⊥	2	3	4	⊥	4	5	LEFT	<table border="1"><tr><th>A</th><th>B</th><th>C</th></tr><tr><td>1</td><td>2</td><td>⊥</td></tr><tr><td>2</td><td>3</td><td>4</td></tr></table>	A	B	C	1	2	⊥	2	3	4	RIGHT	<table border="1"><tr><th>A</th><th>B</th><th>C</th></tr><tr><td>2</td><td>3</td><td>4</td></tr><tr><td>⊥</td><td>4</td><td>5</td></tr></table>	A	B	C	2	3	4	⊥	4	5
A	B	C																																	
1	2	⊥																																	
2	3	4																																	
⊥	4	5																																	
A	B	C																																	
1	2	⊥																																	
2	3	4																																	
A	B	C																																	
2	3	4																																	
⊥	4	5																																	

Projektionsergebnis Menge oder Multimenge

`select Name from Ausleih`

Name
Meyer
Schulz
Müller
Meyer

`select distinct Name from Ausleih`

Name
Meyer
Schulz
Müller

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

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

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
```

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
```

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
```

Bezug zum Tupelkalkül

select: Zielliste im Tupelkalkül

Letzte Anfrage entspricht

```
{eins.ISBN, zwei.Titel | Bücher(eins)  $\wedge$  Bücher(zwei)}
```

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 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 ...
```

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 θ -Verbund erlaubt

Ungewißheitsselektion

■ theoretisch nur Abkürzung für disjunktiv verknüpfte Bedingung

■ Syntax

attribut **like** spezialkonstante

■ Spezialkonstante kann beinhalten

- ◆ '%' (kein oder beliebig viele Zeichen)
- ◆ '_' (genau ein Zeichen)

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

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

Das in-Prädikat und geschachtelte Anfragen

- Syntax:

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

- Beispiel:

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

- natürlicher Verbund mit nachfolgender Projektion

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**

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**

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)
```

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)
```

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 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.

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

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

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))
```

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)

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, 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 )
```

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.

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 )
```

Weitere Sprachkonstrukte von SQL

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

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
```

Vergleich Relationenalgebra und SQL

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

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

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 II

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

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(*)**)

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

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

Gruppierung: Beispiele

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

Anzahl	PANr
2	7754
1	4711
1	5588
2	9912

Quantoren und Mengenvergleiche

■ Syntax

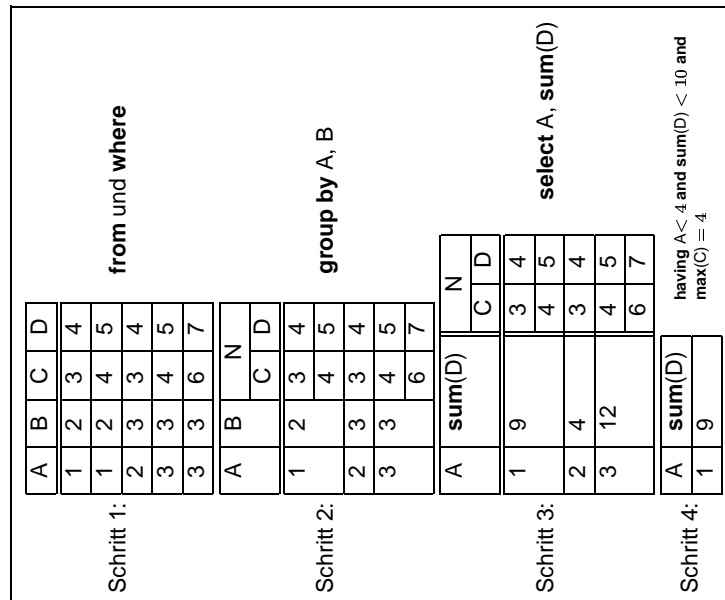
```
attribut  $\theta$  { all | any | some } ( select attribut
from ... where ... )
```

■ Bedeutung all Allquantor, any, some Existenzquantoren

■ Beispiele

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

Gruppierung: Schema



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)
```

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
...			

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

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 II

- Zählen von Wertemengen
 - select distinct** X.PANr
 - from** Prüft X, Prüft Y
 - where** X.PANr = Y.PANr
 - and** X.Matrikelnummer <> Y.Matrikelnummer

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

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!)
    
```

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 Ausleihung)
- Mehrtupel-Operationen (erhöhe das Gehalt aller Mitarbeiter um 4.5%)

SQL: vor allem Mehrtupel-Operationen

Änderungsoperationen auf Sichten: später

Behandlung von Nullwerten II

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

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

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)

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

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

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)
```

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-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 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-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

- 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 III

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