3. Anfragesprachen für Relationale Datenbanken

- Kriterien für Anfragesprachen
- Relationenalgebra

  mathematisches Fundament für relationale Anfragesprachen
- SQL-Kern

  Basiskonzepte zum Formulieren von Datenbank-Anfragen
- Weitere Sprachkonstrukte von SQL
- Änderungsoperationen in SQL
- SQL-Versionen

3.1. Kriterien für Anfragesprachen

allgemeine Kriterien:
(nicht nur für relationale Anfragesprachen)

Ad-Hoc-Formulierung: Der Benutzer soll eine Anfrage formulieren können, ohne ein vollständiges Programm schreiben zu müssen.

Deskriptivität: Der Benutzer soll formulieren „Was will ich haben?“ und nicht „Wie komme ich an das, was ich haben will?“ (letzteres ist *operational*).

Mengenorientiertheit: Jede Operation soll auf Mengen von Daten gleichzeitig arbeiten (a-set-at-a-time), nicht navigierend nur auf einzelnen Elementen (one-tuple-at-a-time).

Abgeschlossenheit: Das Ergebnis ist wieder eine Relation und kann wieder als Eingabe für die nächste Anfrage verwendet werden.

Adäquatheit: Alle Konstrukte des zugrundeliegenden Datenmodells werden unterstützt.

Kriterien für Anfragesprachen II

Orthogonalität: Sprachkonstrukte sind in ähnlichen Situationen auch ähnlich anwendbar.

Optimierbarkeit: Die Sprache besteht aus wenigen Operationen, für die es Optimierungsregeln gibt.

Effizienz: Jede Operation ist effizient ausführbar (im Relationenmodell hat jede Operation eine Komplexität $\leq O(n^2), n$ Anzahl der Tupel einer Relation).

Sicherheit: Keine Anfrage, die syntaktisch korrekt ist, darf in eine Endlosschleife geraten oder ein unendliches Ergebnis liefern.

Eingeschränktheit: (folgt aus Sicherheit, Optimierbarkeit, Effizienz) Die Anfragesprache darf keine komplette Programmiersprache sein.

Vollständigkeit: Die Sprache muß mindestens die Anfragen einer Standardsprache (wie etwa die der Relationenalgebra) ausdrücken können.

3.2. Relationenalgebra

Anfragealgebra:

- (Mathematik:) eine *Algebra* ist definiert durch Wertebereich und auf diesem definierte Operatoren
- für Datenbankanfragen:
  Inhalte der Datenbank sind Werte, und Operatoren definieren Funktionen zum Berechnen von Anfrageergebnissen

Spezialfall: Relationenalgebra

Anfrage: Folge von Operationen, die aus den Basisrelationen eine Ergebnisrelation berechnet.

Ergebnisrelation

- interaktiv auf dem Bildschirm anzeigen,
- per Programm weiterverarbeiten („Einbettung“)

Sicht: Folge von Operationen, die unter einem Sichtnamen langfristig abgespeichert wird und unter diesem Namen wieder aufgerufen werden kann (ergibt eine Sichtrelation)

Snapshot: Ergebnisrelation einer Anfrage, die unter einem Snapshot-Namen abgelegt wird, aber nie ein zweites Mal (mit geänderten Basisrelationen) berechnet wird (etwa Jahresbilanzen)

Relationenalgebra

Betrachtung der Operationen

- Projektion (Spalten ausblenden; Zeichen π)
- Selektion (Zeilen heraussuchen; Zeichen σ)
- Verbund / Join (Tabellen verknüpfen; Zeichen $\bowtie$)
- Vereinigung (Zeichen $\cup$)
- Differenz (Zeichen $-$)
- Umbenennung (Spalten umbenennen; Zeichen β)

am laufenden Beispiel:

Ausleih

Invnr	Name
4711	Meyer
1201	Schulz
0007	Müller
4712	Meyer

Buch

Invnr	Titel	ISBN	Autor
0007	Dr. No	3-125	James Bond
1201	Objektbanken	3-111	Heuer
4711	Datenbanken	3-765	Vossen
4712	Datenbanken	3-891	Ullman
4717	Pascal	3-999	Wirth

Projektion

- Syntax:

$$\pi_{\texttt{attributmenge}}(\texttt{relation})$$

- Semantik:

$$\pi_X(r) := \{t(X) \mid t \in r\}$$

für $r(R)$ und $X \subseteq R$ Attributmenge in R.

[$t(X)$: Restriktion von t auf Attribute in X; bzw. X Menge der Attribute zu t]

Beispiel 1:

- Projektion auf ein Attribut

$$\pi_{\texttt{Name}}(\texttt{Ausleih})$$

ergibt als Ergebnisrelation:

Name
Meyer
Schulz
Müller

- doppelte Ergebnistupel eliminiert ($\rightarrow$ Duplikateleminierung)

Projektion II

Beispiel 2:

- Projektion auf Attributmenge

$$\pi_{\texttt{Invnr},\texttt{ISBN}}(\texttt{Buch})$$

ergibt:

Invnr	ISBN
0007	3-125
1201	3-111
4711	3-765
4712	3-891
4717	3-999

einfache Optimierungsregel für Projektionen:

- bei vielen Projektionen hintereinander reicht die zuletzt ausgeführte auch allein:

$$\pi_{\texttt{Invnr}}(\pi_{\texttt{Invnr},\texttt{ISBN}}(\texttt{Buch}))$$

ergibt optimiert

$$\pi_{\texttt{Invnr}}(\texttt{Buch})$$

Selektion

- Syntax:

$$\sigma_{\texttt{bedingung}}(\texttt{relation})$$

- Semantik:

$$\sigma_F(r) := \{t \mid t \in r \wedge F(t) = \texttt{true}\}$$

Selektionsbedingung F:

- *Konstanten-Selektion:*

$$\texttt{Attribut}\ \theta\ \texttt{Konstante}$$

boolesches Prädikat θ ist $=$ oder $\neq$, bei linear geordneten Wertebereichen auch $\leq$, $<$, $\geq$ oder $>$

- *Attribut-Selektion:*

$$\texttt{Attribut1}\ \theta\ \texttt{Attribut2}$$

- F logische Verknüpfung mehrerer Konstanten- oder Attribut-Selektionen mit $\wedge, \vee$ oder $\neg$

Selektion II

Beispiel:

- Selektion

$$\sigma_{\texttt{Name} \leq \text{`N'}}(\texttt{Ausleih})$$

ergibt

Invnr	Name
4711	Meyer
0007	Müller
4712	Meyer

einfache Optimierungsregeln:

- Selektionen lassen sich in der Reihenfolge beliebig vertauschen
- Manchmal lassen sich Projektion und Selektion vertauschen;

  Voraussetzung:
  Selektionsattribute kommen in Projektionsliste vor

Verbund

- Syntax des (natürlichen) Verbundes (natural join):

$$\texttt{Relation1} \bowtie \texttt{Relation2}$$

- Semantik:

$$r_1 \bowtie r_2 := \{t \mid t \ (R_1 \cup R_2) \wedge (\forall i \in \{1,2\}\, \exists t_i \in r_i : t_i = t(R_i)\,)\,\}$$

- Verbund verknüpft Tabellen über gleichbenannten Spalten bei gleichen Attributwerten

Beispiel:

- Verbund $\texttt{Ausleih} \bowtie \texttt{Buch}$

  ergibt:

Name	Invnr	Titel	ISBN	Autor
Müller	0007	Mr. No	3-125	James Bond
Schulz	1201	Objektbanken	3-111	Heuer
Meyer	4711	Datenbanken	3-765	Vossen
Meyer	4712	Datenbanken	3-891	Ullman

- nicht ausgeliehenes Pascal-Buch verschwindet: Tupel, die keinen Partner finden (dangling tuples), werden eliminiert
- *später:* outer join, der „dangling tuples“ übernimmt

Verbund II

$$\pi_{\texttt{Autor}}(\texttt{Buch}) \bowtie \pi_{\texttt{Invnr}}(\texttt{Ausleih})$$

entartet zu *kartesischem Produkt*:

Autor	Invnr
James Bond	4711
James Bond	1201
James Bond	0007
James Bond	4712
Heuer	4711
Heuer	1201
Heuer	0007
Heuer	4712
Vossen	4711
...	...

Eigenschaften des Verbunds:

- aus $R_1 \cap R_2 = \{\}$ folgt $r_1 \bowtie r_2 = r_1 \times r_2$
- Projektion nicht inverse Operation zu Verbund: $\pi_{R_1}(r_1 \bowtie r_2) \subseteq r_1$
- Verbund nicht die inverse Operation zu zwei Projektionen (nur bei Verbundtreue)
- Verbund kommutativ: $r_1 \bowtie r_2 = r_2 \bowtie r_1$
- Verbund assoziativ: $(r_1 \bowtie r_2) \bowtie r_3 = r_1 \bowtie (r_2 \bowtie r_3)$
- Daher erlaubt: $\bowtie_{i=1}^{p} r_i$

Umbenennung

- Syntax:

$$\beta_{\texttt{neu}\leftarrow\texttt{alt}}(\texttt{relation})$$

- Semantik:
  - verändert nicht die Relation (Tupelmenge)
  - verändert das zugehörige Relationenschema durch Änderung des Attributnamens `alt` in `neu`
- Umbenennung ermöglicht nun Vereinigung, Differenz und Durchschnitt von Relationen mit unterschiedlichen Relationenschemata (wegen verschiedener Benennung von Attributen)

Beispiel:

- Buch1

Autor1
James Bond
Heuer
Vossen
Ullman
Wirth

Buch2

Autor2
Witt
Vossen
Silberschatz
Meier
Wirth

- $\beta_{\texttt{Autor1}\leftarrow\texttt{Autor2}}(\texttt{Buch2})$

  hat gleiches Relationenschema wie `Buch1`

Mengenoperationen: Vereinigung

- Syntax: `relation1` $\cup$ `relation2`
- Semantik: $r_1 \cup r_2 := \{t \mid t \in r_1 \vee t \in r_2\}$

Beispiel:

- `Buch1` $\cup$ $\beta_{\texttt{Autor1} \leftarrow \texttt{Autor2}}(\texttt{Buch2})$

  ergibt

Autor1
James Bond
Heuer
Vossen
Ullman
Wirth
Witt
Silberschatz
Meier

Mengenoperationen: Differenz

- Syntax: `relation1` $-$ `relation2`
- Semantik: $r_1 - r_2 := \{t \mid t \in r_1 \wedge t \notin r_2\}$

Beispiel:

- `Buch1` $-$ $\beta_{\texttt{Autor1} \leftarrow \texttt{Autor2}}(\texttt{Buch2})$

  ergibt

Autor1
James Bond
Heuer
Ullman

Mengenoperationen: Durchschnitt

- Syntax: `relation1` $\cap$ `relation2`
- Semantik: $r_1 \cap r_2 := \{t \mid t \in r_1 \wedge t \in r_2\}$

Beispiel:

- $\texttt{Buch1} \cap \beta_{\texttt{Autor1} \leftarrow \texttt{Autor2}}(\texttt{Buch2})$

  ergibt

Autor1
Vossen
Wirth

Unabhängigkeit und Vollständigkeit

- Minimale Relationenalgebra:

$$\Omega = \{\pi, \sigma, \bowtie, \beta, \cup, -\}$$

- unabhängig: kein Operator kann weggelassen werden ohne Vollständigkeit zu verlieren
- andere unabhängige Menge: $\bowtie$ und β durch $\times$ ersetzen
- *Relationale Vollständigkeit:* Ω (oder gleich ausdrucksstarke unabhängige Menge von Operatoren) ist das Maß für die Ausdrucksfähigkeit relationaler Anfragesprachen
- *Strenge relationale Vollständigkeit:* zu jedem Ausdruck mit Operatoren aus Ω gibt es einen Ausdruck auch mit der anderen Menge von Operationen

Problem: Quantoren

- Allquantor in Relationenalgebra ausdrücken, obwohl in Selektionsbedingungen nicht erlaubt
- *Division* (kann aus Ω hergeleitet werden)
- Gegeben seien $r_1(R_1)$ und $r_2(R_2)$ mit $R_2 \subseteq R_1$, $R' = R_1 - R_2$.
  Dann ist

$$\begin{aligned} r'(R') &= \{t \mid \forall t_2 \in r_2 \exists t_1 \in r_1 : \; t_1(R') = t \\ &\qquad\qquad\qquad\qquad \wedge t_1(R_2) = t_2\} \\ &=: r_1 \div r_2 \end{aligned}$$

- Division von r_1 durch r_2

$$r_1 \div r_2 = \pi_{R'}(r_1) - \pi_{R'}((\pi_{R'}(r_1) \bowtie r_2) - r_1)$$

Division: Beispiel

r_1

PILOT	FLUGZ.
Snoopy	707
Snoopy	727
Snoopy	747
Meyer	707
Meyer	727
Müller	707
Müller	727
Müller	747
Müller	777
Lüdenscheid	727

r_2

FLUGZ.
707
727
747

r_3

FLUGZ.
707

$r_1 \div r_2$ ergibt

r'

PILOT
Snoopy
Müller

$r_1 \div r_3$ ergibt

r'

PILOT
Snoopy
Meyer
Müller

Algebra-Erweiterungen

- Nullwerte

r_1

A	B
a	b
a	∃

r_2

B	C
b	c
∃	c

∃: „Wert existiert, aber zur Zeit nicht bekannt“

Verbund

A	B	C
a	b	c

- Nullwerte: **d**efinit- und **m**aybe-Markierungen

r_1

A	B	STATUS
a	b	**d**
a	∃	**d**

r_2

B	C	STATUS
b	c	**d**
∃	c	**d**

$r_1 \mid\!\bowtie r_2$ liefert

A	B	C	STATUS
a	b	c	**d**
a	b	c	**m**
a	b	c	**m**
a	∃	c	**m**

- Operationen auf Wertebereichen (Spaltenerweiterung, abgeleitete Attribute)

3.3. 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*

order by

- *Ausgabereihenfolge für das Ergebnis (Sortierung)*

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 um auf eine Relation mehrfach zuzugreifen

  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

Die select-Klausel

Relationenalgebra: abschließende Projektion

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

(vgl. Reationenalgebra: Duplikateleminierung!)

Projektionsergebnis: Menge oder Multimenge

select `Name` **from** `Ausleih`

Name
Meyer
Schulz
Müller
Meyer

select distinct `Name` **from** `Ausleih`

Name
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*

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 θ konstante

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

  attribut1 θ 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 θ-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
```


Textmuster-Selektion

- theoretisch nur Abkürzung für disjunktiv verknüpfte Bedingung
- Syntax:

  `attribut` **like** `spezialkonstante`

- Spezialkonstante kann beinhalten
  - '%' (kein oder beliebig viele Zeichen)
  - '_' (genau ein Zeichen)

Beispiel:
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* (explizite Behandlung von Nullwerten):

  `attribut` **is null**

  (notwendig, da Nullwerte sich in Vergleichen sonst „merkwürdig“ verhalten)

- *Quantifizierte Bedingungen*, wenn ein Argument in einem Vergleich eine Menge liefert

  (**all**, **any**, **some** und **exists**)

  $\rightsquigarrow$ Schachtelung von Anfragen

- 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**

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

- 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' )
```

abarbeiten

Verzahnt geschachtelte Anfragen

- in der inneren Anfrage Relationennamen oder Name der Tupelvariable aus dem **from**-Teil der äußeren Anfrage verwenden

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

- 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 ) )
```

Frage: Was soll diese Anfrage berechnen?

- **exists**-Prädikat: $\exists$-Quantor
- 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

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
  ```

  Attributkompatiblität: `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:

  $$\pi_{\texttt{PANr}}(\texttt{Mitarbeiter}) - \pi_{\texttt{PANr}}(\texttt{Studenten})$$

  in SQL:

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

3.4. 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

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'

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
    Attributmenge hinter Gruppierung: G
  - schachteln nach Attributen $R - G$, d.h. für gleiche G-Werte werden Resttupel in Relation gesammelt
  - d.h. Tupel, die die **where**-Klausel erfüllen, werden in Gruppen mit gleichen G-Werten zusamengefaßt

- **having** ist Selektionsbedingung auf gruppierter Relation

- darf Bezug nehmen auf
  - Gruppierungsattribute
  - beliebige Aggregatfunktionen über Nicht-Gruppierungsattributen

Gruppierung: Schema

Schritt 1:

A	B	C	D
1	2	3	4
1	2	4	5
2	3	3	4
3	3	4	5
3	3	6	7

from und **where**

Schritt 2:

A	B	N	
		C	D
1	2	3	4
		4	5
2	3	3	4
3	3	4	5
		6	7

group by A, B

Schritt 3:

A	**sum**(D)	N	
		C	D
1	9	3	4
		4	5
2	4	3	4
3	12	4	5
		6	7

select A, **sum**(D)

Gruppierung in N *zur Übersicht mitangegeben*

Schritt 3':

A	**sum**(D)
1	9

select A, **sum**(D)
...
having A< 4 **and sum**(D) < 10
and max(C) $= 4$

Gruppierung: Beispiele

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

Anzahl	PANr
2	7754
1	4711
1	5588
2	9912

```
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 = 'HRO-912291'
and   Note ≥ all(select Note from Prüft
                 where Matrikelnummer = 'HRO-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
		...	

- 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
- 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

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

3.5. Weitere Sprachkonstrukte von SQL

vor allem Erweiterungen in SQL-92 gegenüber SQL-89

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
```

Weitere Verbunde in SQL-92

- Gleichverbund (equi join)

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

- natürlicher Verbund

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

- jeder SFW-Block hinter **from** (Orthogonalität)

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

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	$\perp$
2	3	4
$\perp$	4	5

LEFT

A	B	C
1	2	$\perp$
2	3	4

RIGHT

A	B	C
2	3	4
$\perp$	4	5

Vereinigung und äußere Verbunde

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

umgesetzt (in SQL-89):

```
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))
```

SQL-92: Tupelbildungen

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

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

- Attribute müssen *kompatibel* sein (siehe unten)

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

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

SQL-92: Benennung von Spalten

Beispiel:

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**

3.6. 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

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 )
```

3.7. 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 ($\rightsquigarrow$ SQL:1999; seit 2000)
- SQL
  - DDL, (SSL,) IQL, DML
  - Sichtdefinition, Transaktionsdefinition, Rechtevergabe, Integritätssicherung

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`)
- selbstdefinierbare Wertebereiche (**create domain**, **alter domain**)
- Änderung des Datenbankschemas: **alter table** und **drop table**
- allgemeine Integritätsbedingungen (über 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
- Embedded SQL und Dynamic SQL sind Teil der Norm

  $\rightsquigarrow$ Datenbankanwendungsprogrammierung
- Data Dictionary ist Teil der Norm

  $\rightsquigarrow$ Speicherung der Schemainformation in Systemtabellen

SQL-92 III

Feature in SQL-92	Entry	Intermediate	Full
Datum, Intervalltypen, **domain**	−	+	+
`string`-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	−	−	+