Karlsruher Institut für Technologie

# Datenbanksysteme

## Kapitel 8: Relationale Datenbanksprache SQL

Lehrstuhl für Systeme der Informationsverwaltung, Fakultät für Informatik

# Zur Information

- Dritte Übung findet Montag, den 03.07.2017 statt

- DBS-Klausur Fr. 04.08.2017 um 12:00 Uhr (1-stündig)
  - Anmeldung ist online möglich ab sofort bis zum 31.07.2017
  - Abmeldung online ist möglich bis einschließlich dem 03.08.2017
  - Danach kann man sich im Hörsaal bis direkt vor Klausurbeginn noch abmelden

# SQL-Kern

- **select**
  - Projektionsliste,
  - *arithmetische Operationen und Aggregatfunktionen*,
- **from**
  zu verwendende Relationen, eventuell Umbenennungen,
- **where**
  - Selektions-, Verbundbedingungen,
  - geschachtelte Anfragen (wieder 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.

# Zugrundeliegende Relationen für manche Beispiele

Künstler

| KID | NAME | LAND | JAHR |
|---|---|---|---|
| 1012 | Neil Young | Kanada | 1945 |
| 1014 | Rammstein | Deutschland | 1994 |
| 1015 | The Ramones | USA | 1974 |
| 1016 | Eric Fish | Deutschland | 1969 |

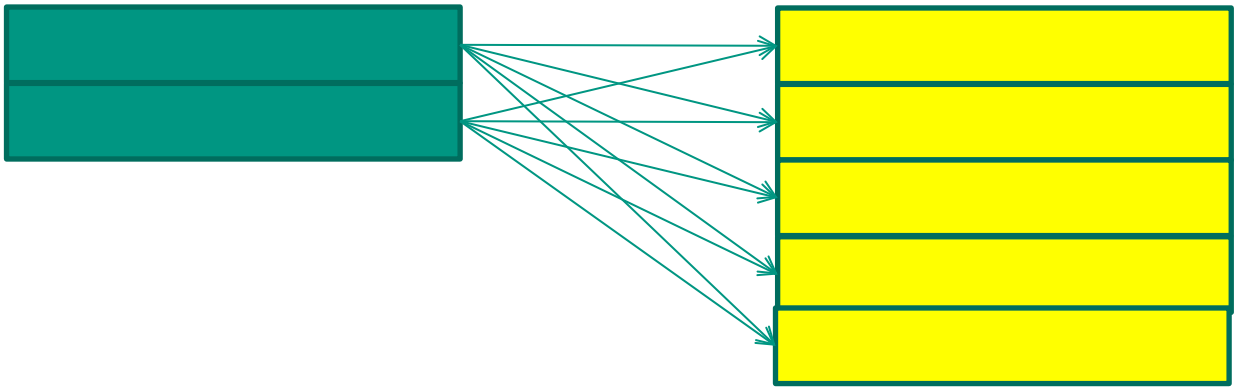
Titel

| TITLE ID | NAME | ART | GRÖSSE | KID |
|---|---|---|---|---|
| 102 | Neil Young – Heart of Gold | mp3 | 2.920kb | 1012 |
| 103 | Rammstein – Ich liebe Neil Young | wma | 4.234kb | 1014 |
| 104 | Neil Young – Old Man | mp3 | 3.161kb | 1012 |
| 105 | Neil Young – Four Strong Winds | wma | 5.125kb | 1012 |

# Kartesisches Produkt

- Bei mehr als einer Relation hinter **from**: Kartesisches Produkt.

```
select * from Künstler, Titel
```



# Natürlicher Verbund

- `select * from Künstler natural join Titel`
- Oft besser als herkömmliche Formulierung, weil
  - übersichtlicher,
  - weniger fehleranfällig
    (z. B. vergißt man leicht ein Attribut,
    wenn man alle explizit aufzählen muß).

# Theta-Join

- Verbund über Verbundbedingungen:

```
select * from Künstler, Titel
where Künstler.KID = Titel.KID
```

- **join**-Operator: θ-Verbund

```
select *
from Künstler join Titel
     on Künstler.KID = Titel.KID
```

# Theta Join – Beispiel

PROF

| PName | PVorname |
|---|---|
| Saake | Gunter |
| Rautenstrauch | Claus |
| Böhm | Klemens |

DKE

| Vorname | Name |
|---|---|
| Klemens | Böhm |
| Erik | Buchmann |
| Ralf | Duckstein |

$$PROF \bowtie_{PVorname<Vorname} DKE$$

| PName | PVorname | Vorname | Nachname |
|---|---|---|---|
| Saake | Gunter | Klemens | Böhm |
| Saake | Gunter | Ralf | Duckstein |
| Rautenstrauch | Claus | Klemens | Böhm |
| Rautenstrauch | Claus | Erik | Buchmann |
| Rautenstrauch | Claus | Ralf | Duckstein |
| Böhm | Klemens | Ralf | Duckstein |

Statt < auch ≥, ≠, usw. möglich.

# Self-Join – Illustration (1)

- Zugrundeliegende Relation:

SNUser

| NAME | VORNAME |
|---|---|
| Böhm | Klemens |
| Buchmann | Erik |
| Duckstein | Ralf |
| Saake | Gunter |

X

| NAME | VORNAME |
|---|---|
| Böhm | Klemens |
| Buchmann | Erik |
| Duckstein | Ralf |
| Saake | Gunter |

- Gewünscht: Alle Paare von Social Network Usern:

| NAME | VORNAME | NAME | VORNAME |
|---|---|---|---|
| Böhm | Klemens | Böhm | Klemens |
| Böhm | Klemens | Buchmann | Erik |
| … | … | … | … |
| Buchmann | Erik | Böhm | Klemens |
| … | … | … | … |

- *Self-Join/Selbstverbund*:
  Kartesisches Produkt einer Tabelle mit sich selbst.

# Self-Join – Illustration (2)

- Einführung von Tupelvariablen:
  Etwa auf eine Relation mehrfach zugreifen.

```
select * from SNUser eins, SNUser zwei
where eins.Alter < zwei.Alter
```

- Ergebnis hat vier Spalten:
  *eins*.Name, *eins*.Vorname,
  *zwei*.Name, *zwei*.Vorname

# Orthogonalität

- Jeder SFW-Block kann
  auch hinter **from** Klausel eingesetzt werden (Orthogonalität).
  (Nicht mit SQL-89.)
- select *
  from (select A, B from R where ….)
  where …

# Äußere Verbunde (1)

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.
- Beispiel:

```
select * from Verlage right outer join Bücher2
using(VName)
```

# Äußere Verbunde (2)

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 |

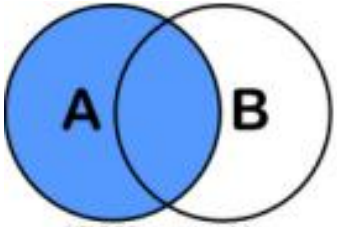
RIGHT

| A | B | C |
|---|---|---|
| 2 | 3 | 4 |
| ⊥ | 4 | 5 |

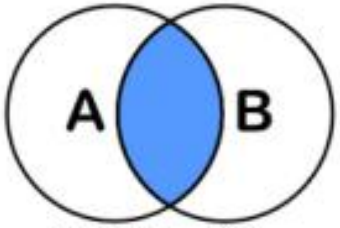


z

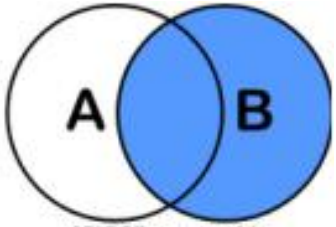
# Verbundtypen



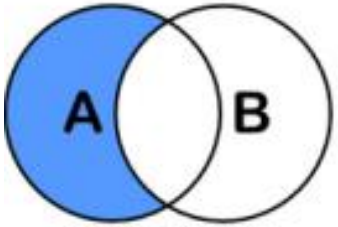

```
SELECT <auswahl>
FROM tabelleA A
LEFT JOIN tabelleB B
ON A.key = B.key
```



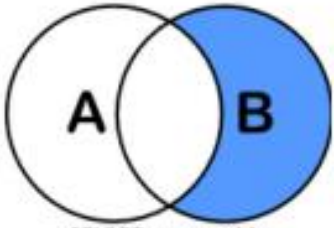

```
SELECT <auswahl>
FROM tabelleA A
INNER JOIN tabelleB B
ON A.key = B.key
```



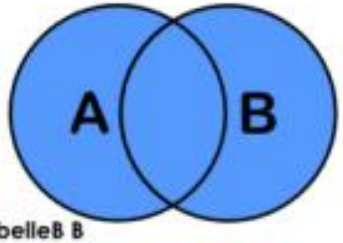

```
SELECT <auswahl>
FROM tabelleA A
RIGHT JOIN tabelleB B
ON A.key = B.key
```



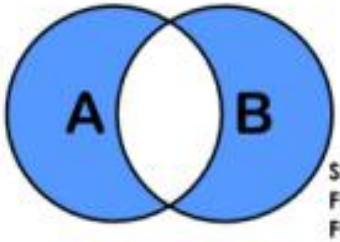

```
SELECT <auswahl>
FROM tabelleA A
LEFT JOIN tabelleB B
ON A.key = B.key
WHERE B.key IS NULL
```




```
SELECT <auswahl>
FROM tabelleA A
RIGHT JOIN tabelleB B
ON A.key = B.key
WHERE A.key IS NULL
```




```
SELECT <auswahl>
FROM tabelleA A
FULL OUTER JOIN tabelleB B
ON A.key = B.key
```




```
SELECT <auswahl>
FROM tabelleA A
FULL OUTER JOIN tabelleB B
ON A.key = B.key
WHERE A.key IS NULL
OR B.key IS NULL
```

Einleitung

SQL-Kern

- Übersicht
- from
- select
- where
- Mengenop.

Weitere Konstrukte

Änderungen

Schluß

Z

# Die select-Klausel

- Abschließende Projektion, Zielliste.
- Syntax:

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

- Bestandteile:
  - Attribute aus **from**-Relationen,
  - arithmetische Ausdrücke über Attributen und Konstanten,
  - Aggregatfunktionen über Attributen.
- **distinct**: Ergebnismenge statt Multimenge.
  Warum wohl?

# Laufendes Beispiel

Künstler

| KID | NAME | LAND | JAHR |
|---|---|---|---|
| 1012 | Neil Young | Kanada | 1945 |
| 1014 | Rammstein | Deutschland | 1994 |
| 1015 | The Ramones | USA | 1974 |
| 1016 | Eric Fish | Deutschland | 1969 |

Titel

| TITLE ID | NAME | ART | GRÖSSE | KID |
|---|---|---|---|---|
| 102 | Neil Young – Heart of Gold | mp3 | 2.920kb | 1012 |
| 103 | Rammstein – Ich liebe Neil Young | wma | 4.234kb | 1014 |
| 104 | Neil Young – Old Man | mp3 | 3.161kb | 1012 |
| 105 | Neil Young – Four Strong Winds | wma | 5.125kb | 1012 |

# Projektion: Menge oder Multimenge

- **select** Land **from** Künstler

| Land |
|---|
| Kanada |
| Deutschland |
| USA |
| Deutschland |

- **select distinct** Land **from** Künstler

| Land |
|---|
| Kanada |
| USA |
| Deutschland |



Z

# Die where-Klausel

- Selektionsbedingung oder Verbundbedingung.
- Syntax:

```
select ...from ...where bedingung
```

- Bedingung:
  - *Konstanten-Selektion*:

    `attribut θ konstante`
  - *Attribut-Selektion* zwischen Attributen mit kompatiblen Wertebereichen:

    `attribut1 θ attribut2`

# Konstantenselektion

- Beispiel:
  **select** *
  **from** Künstler
  **where** Land = 'Deutschland‘
- String-Auszeichnung in SQL mittels ''

# Verbundbedingung – Alternative Definition

- `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.

- 
```
select Bücher.Titel,
       Bücher_Stichwort.Stichwort
from Bücher inner join Buch_Stichwort
on (Bücher.ISBN = Buch_Stichwort.ISBN)
```

# Ungewißheitsselektion (1)

- Syntax:

```
attribut like spezialkonstante
```

- Spezialkonstante kann beinhalten
  - '%' – kein oder beliebig viele Zeichen,
  - '_' – genau ein Zeichen.
- Prinzipiell geht alles was man von regulären Ausdrücken her kennt

# Ungewißheitsselektion (2)

- 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 =
                    'BenjXFCummingsSCHlumpf'
or ...
```

# Weitere Bedingungen

- *Null-Selektion*

  ```
  attribut is null
  ```

- boolesche Ausdrücke mit Konnektoren **or**, **and** und **not**,
- *quantifizierte Bedingungen*,
  wenn ein Argument in Vergleich Menge liefert (**all**, **any**, **some** und **exists**;
  wird gleich besprochen).

# Das in-Prädikat (1)

- Syntax:

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

- Beispiel:

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

Bücher

| ISBN | Titel | Verlag |
|---|---|---|
| | | |

Empfiehlt

| PANr | ISBN |
|---|---|
| | |

# Das in-Prädikat (2)

- 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.

# Das in-Prädikat (3)

- Wie sieht äquivalente Anfrage ohne Schachtelung aus?

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

# Verzahnt geschachtelte Anfragen (1)

- *Verzahnt geschachtelt* – in der inneren Anfrage Relationen- oder Tupelvariablen-Name aus dem **from**-Teil der äußeren Anfrage verwenden.
- Beispiel:

| Personen | PANr | Nachname | Büro |
|---|---|---|---|
| | | | |

| Prüft | PANr | Matrikel | Vorlesung | Ort | Zeit | Note |
|---|---|---|---|---|---|---|
| | | | | | | |

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

# Verzahnt geschachtelte Anfragen (2)

- 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).

# Verzahnt geschachtelte Anfragen (3)

- Abarbeitung (Fortsetzung):

  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.

- Was leistet Anfrage also, anschaulich gesprochen?

# Verzahnt geschachtelte Anfragen (4)

| Personen | PANr | Nachname | Büro |
|---|---|---|---|
| | | | |

| Prüft | PANr | Matrikel | Vorlesung | Ort | Zeit | Note |
|---|---|---|---|---|---|---|
| | | | | | | |

- ```
  select Nachname
      from Personen
      where 1.0 in (select Note
                    from Prüft
                    where PANr = Personen.PANr)
  ```
- Wie sieht äquivalente Anfrage ohne Schachtelung aus?
  ```

# Das exists-Prädikat

- Testet, ob Ergebnis der inneren Anfrage nicht leer.

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

Buch_Exemplare

| Invnr | ISBN | Standort |
|---|---|---|
| | | |

Ausleihe

| Invnr | Ausleiher-Nr | Ausleih-Datum | Frist |
|---|---|---|---|
| | | | |

- Was leistet die Anfrage (anschaulich)?

# Kompatible Attribute (1)

- Attribute sind kompatibel bei kompatiblen Wertebereichen.
- Zwei Wertebereiche sind kompatibel, wenn sie
  - gleich sind,
  - beide auf `character` basierende Wertebereiche sind (unabhängig von der Länge der Strings), oder
  - beide numerische Wertebereiche sind (unabhängig vom genauen Typ) wie `integer` oder `float`.

# Kompatible Attribute (2)

- Kompatible Attribute können in Vergleichen und Mengenoperationen benutzt werden.
- Beispiel: '... `where Salary < 50000`', und `Salary` ist vom Typ `float`.

# SQL-92: Tupelbildungen

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

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

- ($a_1$, ..., $a_n$) < ($b_1$, ..., $b_n$)
  – wahr, wenn ein j existiert, für das $a_j$<$b_j$ und $a_i$=$b_i$ für alle i<j gilt (lexikographische Ordnung).
  Beispiel: (Böhm, Klemens, Karlsruhe) > (Böhm, Karlheinz, Äthiopien)
- Attribute müssen *kompatibel* sein.
  Beispiel: (Böhm, Klemens, 76133) nicht vergleichbar mit (Böhm, Karlheinz, Äthiopien)

# Vereinigung

- Vereinigung **union** / **union all**:

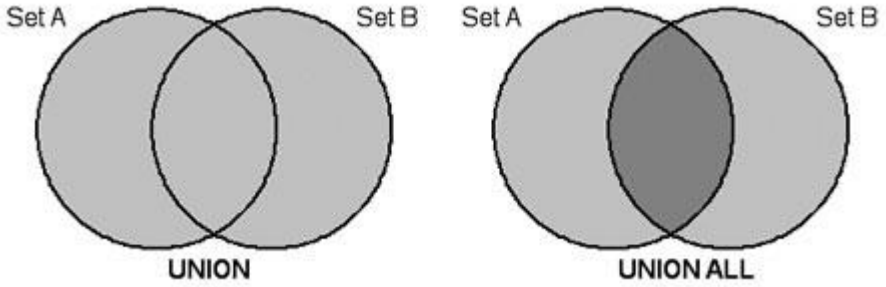
```
SFW_block1 union SFW_block2
```

- Beispiel:

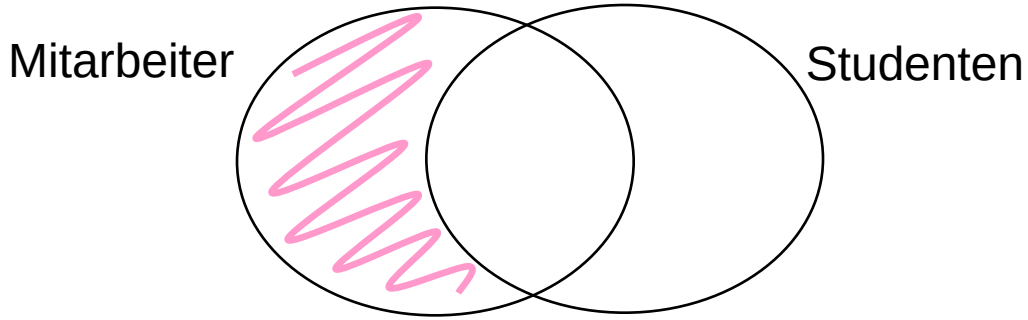
```
select A, B, C from R1
      union
select A, C, D from R2
```

*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.
- **union** (mit Duplikateliminierung) vs. **union all**



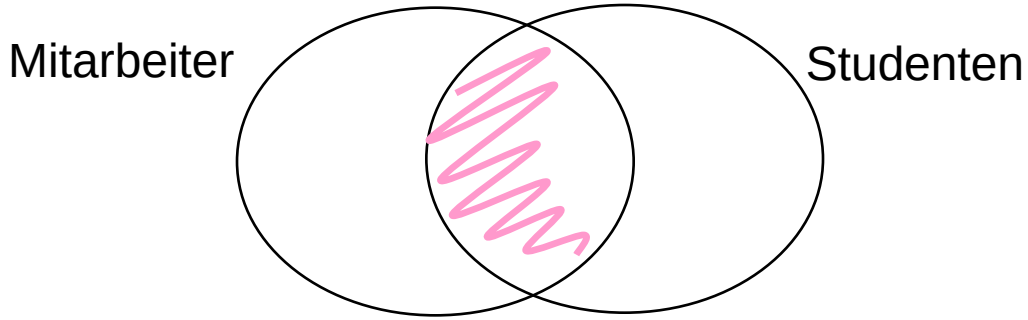
# Mengendifferenz




- Mengenoperator ‚except' oder ‚minus'
- Anschaulich: Alle Mitarbeiter, die **keine** Studenten sind
- ```
  select PANr from Mitarbeiter
  EXCEPT select PANr from Studenten
  ```
- In Alternative SQL:

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

# Mengendurchschnitt




- Mengenoperator ‚**intersect**‘
- Anschaulich: Alle Mitarbeiter, die **auch** Studenten sind

- ```
  select PANr from Mitarbeiter
  INTERSECT select PANr from Studenten
  ```



Z
  ```


Photo by Susana

Karlsruher Institut für Technologie

# Datenbanksysteme

## Kapitel 8: Relationale Datenbanksprache SQL - Fortsetzung

Lehrstuhl für Systeme der Informationsverwaltung, Fakultät für Informatik

# Tupel-orientierte Auswertung

- Q1: Alle Lokale in denen Knieweich ausgeschenkt wird
  - `Select Lokal from LC where Cocktail= 'Knieweich'`
  - Teste jedes Tupels ob Attributwerte passen
- Q2: Alle Lokale in denen **KEIN** Knieweich ausgeschenkt wird
  - `(Falsch) Select Lokal from LC where Cocktail ≠ 'Knieweich'`
  - `(Richtig) Select Lokal from LC where Lokal not in (Select Lokal from LC where Lokal Cocktail = 'Knieweich')`
  - `(Alternativ) Select Lokal from LC where Lokal` **`MINUS`** `Select Lokal from LC where Lokal Cocktail = 'Knieweich'`

LC

| Lokal | Cocktail |
|---|---|
| L1 | Cuba Libre |
| L1 | Knieweich |
| L2 | Cuba Libre |
| L2 | Swimmingpool |
| L3 | Knieweich |

Sämtliche Lokale in denen es einen anderen Cocktail als ‚Knieweich' gibt

# 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 (1)

- Innerhalb von **select** und **where**:
  Statt Attributen auch *skalare Ausdrücke*.
  - Numerische Wertebereiche: etwa + , - , * , /,
  - Strings: **char_length**, Konkatenation .. , **substring** (Teilzeichenkette),
  - Datumstypen, Zeitintervalle:
    **current_date**, **current_time**, + , - , *.

# Operationen auf Wertebereichen (2)

- 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 |

# Umbenennung

- Im vorigen Beispiel: Zweite Spalte nicht benannt.
- In SQL: Attributnamen zuordnen.

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

Ergebnis

| ISBN | Dollar_Preis |
|---|---|
| 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(all Preis) from Buch_Versionen`
- `select sum(distinct Preis) from Buch_Versionen`
- `select sum(Preis) from Buch_Versionen`

- `select count(distinct PANr) from Prüft`
- `select count(*) from Professoren`
- `select avg(all Note) from Prüft where V_Bezeichnung = 'Datenbanken I'`
- `select Matrikelnummer from Prüft where Note < (select avg(all Note) from Prüft)`

# Aggregatfunktionen (1)

- 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 *.

# Aggregatfunktionen (2)

- Vor Argument (außer bei **count**(*)) optional: **distinct** oder **all** (**all** Voreinstellung).
- Nullwerte werden vor Anwendung aus Wertemenge eliminiert, außer bei **count**(*).

# Group-by Operator (1)

| Marke | Datum | Bundesland | Anzahl |
|---|---|---|---|
| BMW | 07.01. 1994 | Hessen | 28 |
| BMW | 08.01. 1994 | Bayern | 37 |
| BMW | 07.01. 1994 | Saarland | 41 |
| Opel | 07.01. 1994 | Hessen | 48 |
| Opel | 08.01. 1994 | Bayern | 62 |
| Opel | 08.01. 1994 | Saarland | 5 |
| Opel | 09.01. 1994 | Saarland | 95 |
| Audi | 07.01. 1994 | Hessen | 55 |
| Audi | 08.01. 1994 | Bayern | 52 |
| Audi | 09.01. 1994 | Bayern | 27 |
| Audi | 10.01. 1994 | Bayern | 62 |

Marke

sum Anzahl

| Marke | Anzahl |
|---|---|
| BMW | 106 |
| Opel | 210 |
| Audi | 196 |

Marke

avg(Anzahl)

| Marke | Anzahl |
|---|---|
| BMW | 35.33 |
| Opel | 52.5 |
| Audi | 49 |

Marke

max(Anzahl)

| Marke | Anzahl |
|---|---|
| BMW | 41 |
| Opel | 95 |
| Audi | 62 |

Bundesland

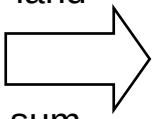
sum(Anzahl)

| Bundesland | Anzahl |
|---|---|
| Hessen | 131 |
| Bayern | 240 |
| Saarland | 141 |

# Group-by Operator (2)

| *Marke* | *Datum* | *Bundesland* | *Anzahl* |
|---|---|---|---|
| BMW | 07.01. 1994 | Hessen | 28 |
| BMW | 08.01. 1994 | Bayern | 37 |
| BMW | 07.01. 1994 | Saarland | 41 |
| Opel | 07.01. 1994 | Hessen | 48 |
| Opel | 08.01. 1994 | Bayern | 62 |
| Opel | 08.01. 1994 | Saarland | 5 |
| Opel | 09.01. 1994 | Saarland | 95 |
| Audi | 07.01. 1994 | Hessen | 55 |
| Audi | 08.01. 1994 | Bayern | 52 |
| Audi | 09.01. 1994 | Bayern | 27 |
| Audi | 10.01. 1994 | Bayern | 62 |

Marke, Bundes-land



sum (Anzahl)

| *Marke* | *Bundesland* | *Anzahl* |
|---|---|---|
| BMW | Hessen | 28 |
| BMW | Bayern | 37 |
| BMW | Saarland | 41 |
| Opel | Hessen | 48 |
| Opel | Bayern | 62 |
| Opel | Saarland | 100 |
| Audi | Hessen | 55 |
| Audi | Bayern | 141 |

# group by und having (2)

- Beispiele von eben:
  - **select** Marke, sum(Anzahl)
    **from** Zulassungen
    **group by** Marke
  - **select** Marke, avg(all Anzahl)
    **from** Zulassungen
    **group by** Marke
  - **select** Marke, max(Anzahl)
    **from** Zulassungen
    **group by** Marke
  - **select** Bundesland, sum(Anzahl)
    **from** Zulassungen
    **group by** Bundesland

# group by und having (1)

- Syntax

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

- Semantik (virtuelle geschachtelte Relation):
  - Relationenschema R und Attributmenge hinter Gruppierung G.
  - Schachteln nach Attributen R $\rightarrow$ G,
    d. h. für gleiche G-Werte werden Resttupel in Relation gesammelt.

# group by und having (3)

- Beispiele von eben (Forts.):
  - **select** Marke, Bundesland, sum(Anzahl)
    **from** Zulassungen
    **group by** Marke, Bundesland

# group by und having (4)

- Ist diese Anfrage zulässig?
  **select** Marke, Datum, sum(Anzahl)
  **from** Zulassungen
  **group by** Marke, Bundesland

# group by und having (4)

- Ist diese Anfrage zulässig?
  **select** Marke, Datum, sum(Anzahl)
  **from** Zulassungen
  **group by** Marke, Bundesland

- Manchmal implizite Konvertierung
  **select** Marke, FIRST(Datum), sum(Anzahl)
  **from** Zulassungen
  **group by** Marke, Bundesland

# group by und having (5)

- **having** ist Selektionsbedingung auf gruppierter Relation,
- darf Bezug nehmen auf
  - Gruppierungsattribute,
  - beliebige Aggregatfunktionen über Nicht-Gruppierungsattributen.
- Beispiel:

```
select panr, sum(entlohnung)
from anstellungen
group by panr
having sum(entlohnung)>10000;
```

# Gruppierung: Schema

- Relation REL:

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

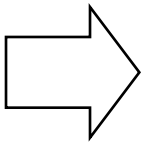
- Anfrage:

```
select A, sum(D) from REL where ...
group by A, B
having A<4 and sum(D)<10 and max(C)=4
```

# Gruppierung: Schritt 1

- **from** und **where**

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

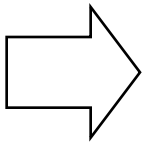


| A | B | C | D |
|---|---|---|---|
| 1 | 2 | 3 | 4 |
| 1 | 2 | 4 | 5 |
| 2 | 3 | 3 | 4 |
| 3 | 3 | 4 | 5 |
| 3 | 3 | 6 | 7 |

# Gruppierung: Schritt 2

- **group by** A, B

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

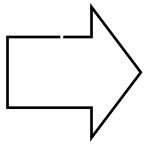


| A | B | N | |
|---|---|---|---|
| | | C | D |
| 1 | 2 | 3 | 4 |
| | | 4 | 5 |
| 2 | 3 | 3 | 4 |
| 3 | 3 | 4 | 5 |
| | | 6 | 7 |

# Gruppierung: Schritt 3

- **select** A, **sum**(D)

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

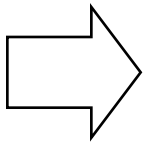


| A | B | sum(D) | N | |
|---|---|---|---|---|
| | | | C | D |
| 1 | 2 | 9 | 3 | 4 |
| | | | 4 | 5 |
| 2 | 3 | 4 | 3 | 4 |
| 3 | 3 | 12 | 4 | 5 |
| | | | 6 | 7 |

# Gruppierung: Schritt 4

- **having** A<4 **and sum**(D)<10 **and max**(C)=4

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



| A | **sum**(D) |
|---|---|
| 1 | 9 |

# Gruppierung: Beispiele (1)

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

- Ergebnis:

| Anzahl | PANr |
|---|---|
| 2 | 7754 |
| 1 | 4711 |
| 1 | 5588 |
| 2 | 9912 |

- Zugrundeliegende Relation:

Ausleihe

| PANr | BuchNr | ... |
|---|---|---|
| | | |

# Gruppierung: Beispiele (2)

- ```
  select count(*), PANr from Ausleihe
  group by PANr
  having count(*) > 1
  ```
- Ergebnis:

| | PANr |
|---|---|
| 2 | 7754 |
| 2 | 9912 |

- ```
  select Matrikelnummer from Prüft
  group by Matrikelnummer
  having avg(Note) < (select avg(Note)
                      from Prüft)
  ```
- Zugrundeliegende Relation:

Prüft

| Matrikelnummer | Note | ... |
|---|---|---|
| | | |

# Quantoren und Mengenvergleiche (1)

- Syntax

```
attribut  θ{
all | any | some }  (select attribut
                        from ...where ...)
```

- Bedeutung: **all** Allquantor,
  **any**, **some** Existenzquantoren, äquivalent.
- Beispiele:

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

# Quantoren und Mengenvergleiche (2)

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

- Wie sieht äquivalente Anfrage aus,
  die ohne Allquantor und Schachtelung auskommt?

# Quantoren und Mengenvergleiche (3)

```
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 geht nicht mit **all**, **any**.
  Definition von Mengen-Gleichheit:
  $\forall x \in M_1: \exists y \in M_2: x=y \wedge \forall x \in M_2: \exists y \in M_1: x=y$

# Quantoren und Mengenvergleiche (4)

- In SQL nur mit Hilfe der Quantoren nicht umsetzbar:
  *Gib alle Bücher aus, an denen 'Vossen' und 'Witt' gemeinsam als Autoren beteiligt waren.*

Buch_Autor

| ISBN | Author |
|---|---|
| | |

# Selbst-Verbund (1)

- Letzte Anfrage erst mit Selbst-Verbund zu lösen.
- Vergleich von Wertemengen:

```
select BA_1.ISBN
from Buch_Autor BA_1, Buch_Autor BA_2
where BA_1.ISBN = BA_2.ISBN
and BA_1.Autor = 'Vossen'
and BA_2.Autor = 'Witt'
```

| BA_1.ISBN | BA_1.Author | BA_2.ISBN | BA_2.Author |
|---|---|---|---|
| 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 (2)

- „Alle Prüfer, die zwei oder mehr Studenten geprüft haben.“
- Zählen von Wertemengen

| Prüft | PANr | Matrikelnummer | ... |
|---|---|---|---|
| | | | |

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

- Wie die Anfrage formulieren bei mehr als zwei Studenten?

# order-by Klausel (1)

- Menge von Tupeln ~> 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 (2)

- 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!)
```

- `Note` kommt nicht in select-Klausel vor.

# order-by Klausel (3) – Top k Anfragen

- Anfrage, die die besten *k* Elemente bzgl. einer Rangfunktion liefert

```
select w1.Name, count(*) as Rang
from WEINE w1, WEINE w2
where w1.Jahrgang <= w2.Jahrgang -- Schritt 1
group by w1.Name, w1.WeinID      -- Schritt 2
having count(*) <= 4             -- Schritt 3
order by Rang                    -- Schritt 4
```

| Name | Rang |
|---|---|
| Zinfandel | 1 |
| Creek Shiraz | 2 |
| Chardonnay | 3 |
| Pinot Noir | 4 |

# order-by Klausel (3) – Top k Anfragen

- Anfrage, die die besten *k* Elemente bzgl. einer Rangfunktion liefert

```
select w1.Name, count(*) as Rang
from WEINE w1, WEINE w2
where w1.Jahrgang <= w2.Jahrgang -- Schritt 1
group by w1.Name, w1.WeinID      -- Schritt 2
having count(*) <= 4             -- Schritt 3
order by Rang                    -- Schritt 4
```

- Ermittlung der k = 4 jüngste Weine Erläuterung
  - Schritt 1: Zuordnung aller Weine die älter sind
  - Schritt 2: Gruppierung nach Namen, Berechnung des Rangs
  - Schritt 3: Beschränkung auf Ränge ≤ 4
  - Schritt 4: Sortierung nach Rang

# Behandlung von Nullwerten (1)

- 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**.

# Behandlung von Nullwerten (2)

- Selbst ein Vergleich A=A ist bei Vorliegen von Nullwerten keine Tautologie mehr, sondern ergibt unknown.
  Derartiger Vergleich in where-Klausel also nicht eliminierbar.
- **`select`** `*` **`from`** `Person`
  **`where`** `Nachname=Nachname`
  nicht dasselbe wie
  **`select`** `*` **`from`** `Person`

| Vorname | Nachname |
|---|---|
| Gunter | Saake |
| Klemens | NULL |

- Boolesche Ausdrücke basieren dann auf dreiwertiger Logik.

# Behandlung von Nullwerten (3)

| *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 |

# Behandlung von Nullwerten (4)

- Fast alle Vergleiche mit Nullwert ergeben Wahrheitswert **unknown** statt **true** oder **false**.
- Ausnahme:
  **is null** ergibt **true**, **is not null** ergibt **false**.

# Rekursive Anfragen

- Anwendung: *Bill of Material*-Anfragen, Berechnung der transitiven Hülle (Flugverbindungen etc.)

- Beispiel: Komme ich von Nurioopta nach Lyndoch?

| Abfahrt | Ankunft | Distanz |
|---|---|---|
| Nuriootpa | Penrice | 7 |
| **Nuriootpa** | **Tanunda** | 7 |
| Tanunda | Seppeltsfield | 9 |
| **Tanunda** | **Bethany** | 4 |
| **Bethany** | **Lyndoch** | 14 |

# Rekursive Anfragen (2)

- Busfahrten mit max. zweimalige Umsteigen ab ’Nuriootpa‘

```
select Abfahrt, Ankunft
from BUSLINIE
where Abfahrt = ’Nuriootpa’
      union
select B1.Abfahrt, B2.Ankunft
from BUSLINIE B1, BUSLINIE B2
where B1.Abfahrt = ’Nuriootpa’ and B1.Ankunft
= B2.Abfahrt
      union
select B1.Abfahrt, B3.Ankunft
from BUSLINIE B1, BUSLINIE B2, BUSLINIE B3
where B1.Abfahrt = ’Nuriootpa’ and B1.Ankunft
= B2.Abfahrt
and B2.Ankunft = B3.Abfahrt
```

# Rekursion in SQL:2003

- Formulierung über erweiterte **`with recursive`**-Anfrage
- Notation

```
with recursive rekursive-tabelle as (
anfrage-ausdruck -- rekursiver Teil
)
[traversierungsklausel] [zyklusklausel]
anfrage-ausdruck -- nicht rekursiver Teil
```

- nicht rekursiver Teil: Anfrage auf Rekursionstabelle
- Message: Geht, aber komplex…

# Rekursion in SQL:2003 - Beispiel

- **Problem**: Zyklen bei Rekursion
  - Einschränkung der Rekursionstiefe

```
with recursive TOUR(Abfahrt, Ankunft, Umsteigen)
as (
  select Abfahrt, Ankunft, 0
  from BUSLINIE
  where Abfahrt = 'Nuriootpa'
    union all
  select T.Abfahrt, B.Ankunft, Umsteigen + 1
  from TOUR T, BUSLINIE B
  where T.Ankunft = B.Abfahrt and Umsteigen < 2
)
select distinct * from TOUR
```

# 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%.“)

# update (1)

- Syntax:

```
update basisrelation
set attribut_1 = ausdr_1, ...,
      attribut_n = ausdr_n
[ where bedingung ]
```

- Beispiele:

Angestellte

| Name | Gehalt |
|---|---|
| Meyer | 3000 |
| Bond | 7200 |
| Schulz | 4400 |

- `update Angestellte set Gehalt=Gehalt+1000 where Gehalt < 5000`
- `update table1 SET column1 = (SELECT expression1 FROM table2 WHERE conditions) [WHERE bedingung];`

# update (2)

Angestellte

| Name | Gehalt |
|---|---|
| Meyer | 4000 |
| Bond | 7200 |
| Schulz | 5400 |



- `update Angestellte set Gehalt = 6000 where Name = 'Bond'`
- Was leistet das folgende Statement?
  `update Angestellte set Gehalt = 3000`

# delete

- Syntax:

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

- Beispiel:

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

- Standardfall ist Löschen mehrerer Tupel:

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

- Löschen der gesamten Relation:

```
delete from Ausleihe
```

Nicht dasselbe wie

```
drop table Ausleihe
```

# Laufendes Beispiel

Künstler

| KID | NAME | LAND | JAHR |
|---|---|---|---|
| 1012 | Neil Young | Kanada | 1945 |
| 1014 | Rammstein | Deutschland | 1994 |
| 1015 | The Ramones | USA | 1974 |
| 1016 | Eric Fish | Deutschland | 1969 |

Titel

| TITLE ID | NAME | ART | GRÖSSE | KID |
|---|---|---|---|---|
| 102 | Neil Young – Heart of Gold | mp3 | 2.920kb | 1012 |
| 103 | Rammstein – Ich liebe Neil Young | wma | 4.234kb | 1014 |
| 104 | Neil Young – Old Man | mp3 | 3.161kb | 1012 |
| 105 | Neil Young – Four Strong Winds | wma | 5.125kb | 1012 |

# insert (1)

- **insert into** Künstler
  **values** (1022,'Raul Seixas',1945,'Brasilien')
- **insert into** basisrelation
  [(attribut_1, ..., attribut_n)]
  **values** (konstante_1, ..., konstante_n)
- **insert into** Künstler(KID, Name)
  **values** (1022,'Raul Seixas')

# insert (2)

- **insert into** basisrelation
  [ (attribut_1, ..., attribut_n) ]
  SQL-anfrage
- **insert into** Kunde (**select** LName, LAdr, 0 **from** Lieferant)

Kunde

| Name | Adr | AnzahlAuftraege |
|---|---|---|
| ... | ... | ... |

Lieferant

| LName | LAdr | Produkt |
|---|---|---|
| ... | ... | ... |

# Mögliche Prüfungsfragen

- Formulieren diverser Anfragen, auch komplexerer Natur, in SQL.
- Was ist eine Umbenennung im SQL-Kontext?
  Wann wird sie gebraucht?
- Geben Sie ein sinnvolles Beispiel für eine Anfrage,
  die eine having-Klausel enthält.
- Geben Sie ein Beispiel für eine Anfrage
  mit einer having-Klausel, bei der
  - man die having-Klausel
    durch eine where-Klausel ersetzen kann,
  - dies nicht so ist.
- Erläutern Sie, warum im SQL-Kontext
  ‚A == A‘ keine Tautologie ist.