Kapitel 6.3 SQL als Anfragesprache

Kap. 6.3.1 Allgemeine Begriffe

Identifiers: Var-Name ~ Tupel-Variable
Table-Name ~ Rel., View
Field-Name ~ Attribut-Bez.

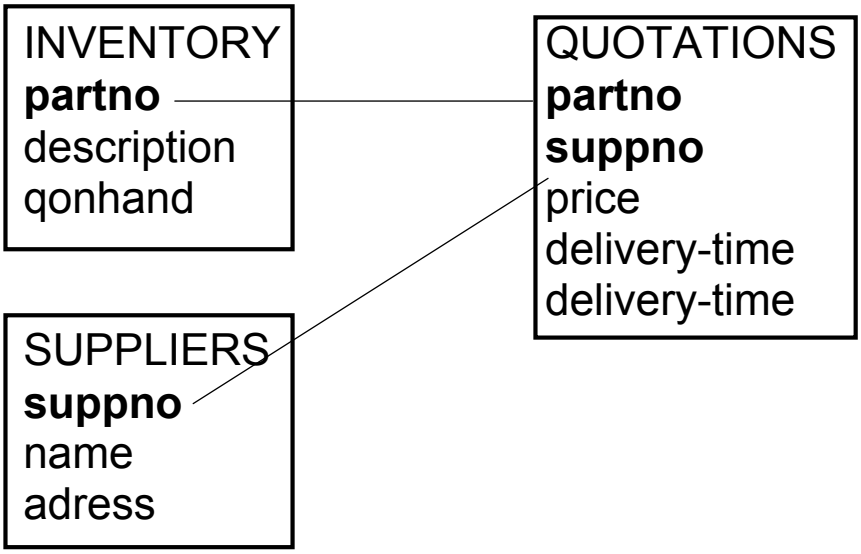
Key-Words:

select	***from***
where	***order by***
key is	***group by***
integer	***real***
string	***char (*)***

Konstantenbez.: .5 1e-3
2.34 2. 1.5E5
"string with \" quotes inside" mit escape character /

Query-Block: QB

Antwort auf QB ist Menge von Tupeln, i-e. selbst wieder Relation (wie bei rel. Algebra)
in QB "Zugriff" auf Tupel mit Tupel-Variablen



QB Beispiel:

select q. suppno, q. partno
from QUOTATIONS q
where q. price ≤ 19.50

Erweiterung um name, adress?

select q. suppno, q. partno, s.name, s.address
from QUOTATIONS q, SUPPLIERS s
where **q.price ≤ 19.50** **and q.suppno = s.suppno**

Kap. 6.3.2 Subqueries:

Beispiel: x, y $x \in R, y \in S$

Q (x, y) := $Q1(x) \wedge x \in R2 \wedge R3 = R4 \wedge Q2(y, x)$

Abkürzung für: $\exists z \in R2 (x = z)$ bzw

$\exists z \in R2 (x.1 = z.i1 \wedge x.2 = z.i2 \wedge ... \wedge x.k = z.ik)$

Subquery entsteht, wenn R2 selbst als Anfrage spezifiziert ist:

R2 := **(select** z.i1, z.i2, ..., z.ik
from R5 z
where ... **)**

außerdem können R3, R4 als Anfragen spezifiziert sein:

⇨ komplexe Struktur für Q(x, y) :

Q1 ∧ **[** x ∈ (**select** z.i1, ..., z.ik
from R5 z
where ...) **]**
∧ (**select** ...) = (**select** ...) ∧ Q2 (y, x)

Innere Blöcke sind ***subqueries*** vom **äußeren** Query-Block Q

Korrelierte Subqueries

Def.: QB **referenziert** Tupel-Variable t ⇔ t kommt in QB vor.

Def: der QB **von** t : genau der QB, in dem t definiert wird,
i.e. **from** T t vorkommt

⇨ t kann nur in **innerem** QB_i von QB vorkommen,
nicht in äußeren

⇨ genau wie Gültigkeitsbereiche bei Blockstruktur!

Def.: QB und QB_i sind korreliert, wenn QB der Query-Block von t ist und t in QB_i referenziert wird.

(QB_i ist zu QB korrelierte subquery)

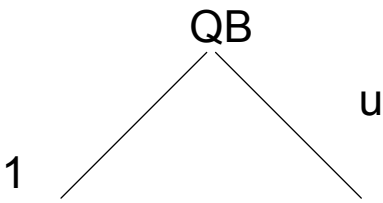
Konsequenzen:

- QB_i muß für jede neue Instantiierung von t neu ausgewertet werden, d.h. innerhalb der Instantiierungs-schleife für t.
- Unkorrelierte Subquery QB_j kann vorher berechnet und in Zwischenergebnis-Variable gemerkt werden.

Siehe auch: Schleifenoptimierung bei algorithmischen Sprachen; Teilausdrücke, die von Laufvariable unabhängig sind!

Beispiel: QB = "Lieferanten, die genau alle Muttern liefern"

```
select   q1. name
   from  SUPPLIERS  q1
   where
      (select   q2.partno                    } von q1 gelieferte
       from     QUOTATIONS q2                } Teile = QB1
       where    q1. suppno = q2. suppno)     }
       =
      (select   i.partno                     } alle Muttern
       from     INVENTORY I                  }  = QB2
       where    i.description = "NUT")       }
```



partno, die	partno
von q1 geliefert	der Muttern
werden	
für jedes q1	genau 1x
neu auswerten!	auswerten!

Hinweis: Subqueries werden hauptsächlich für Mengenvergleiche benötigt.

x $\in$ R läßt sich auch durch Join ausdrücken: mit Hilfe einer zusätzlichen Tupelvariable.

z $\in$ R plus konjunktiver Teil zu Query: $\wedge$ z = x

bzw. z.i1 = x.1 ... z.ik = x.k

Kap. 6.3.3 Unterdrückung von Tupel-Variablen, Verschattung

```
... from   QUOTATIONS q, SUPPLIERS s
    where  q.suppno = s. suppno
                 |||
    from   QUOTATIONS, SUPPLIERS
    where  QUOTATIONS. suppno = SUPPLIERS. suppno
```

d.h. verwende Relationsnamen als Tupelvariablen,
jeweilige Rolle abhängig vom Kontext,
formal aber sehr unsauber!

Hinweis: Tupel Variablen erforderlich bei self-joins, z. B.

Großvater-Abfrage über

```
create table Vater
    (Kind string,
    Papa string)

select V1. Kind, V2. Papa
  from   Vater V1, Vater V2
  where  V1. Papa = V2. Kind
```

V1, V2 sind erforderlich und werden unabhängig instantiiert.

Kap. 6.3.4 Syntaktische Vereinfachung in SQL:

1. Ersetze Tupel-Variable r durch Relations-Bezeichnung R, wenn nur 1 Tupel-Variable bezüglich R vorkommt.

2. Lasse bei Attribut-Variablen die Tupel-Variable weg, wenn sie eindeutig ergänzbar ist. Siehe Beispiel 6.3.6

Beispiel für Verschattung:
nochmals: "Lieferanten, die genau alle Muttern liefern"


```
select  q1.suppno
        from    quotations q1
        where (select       q2.partno                    }
                   from     quotations q2                }   QB1
                   where    q1.suppno = q2.suppno)       }
               =                                         }
               (select   i.Partno                        }
                   from     INVENTORY i                  }   QB2
                   where    i.description = "NUT" )      }
```

i kann ganz unterdrückt werden in QB2
q2 kann ganz unterdrückt werden in QB1
q1 kann ganz unterdrückt werden in Q, aber nicht in QB1
weil quotations aus Q verschattet ist durch lokales quotations aus QB1

Kap. 6.3.5 Aggregatfunktionen

über alle Attributwerte einer Rel. einschl. Duplikate:

Funktion	**Ergebnis Typ:**
sum (attr.)	integer, real, ...
avg (attr.)	real
min	integer, real, string, DateTime
max	integer, real, string, DateTime
count (∗)	integer

```
select avg (q.price), min (q.price),
       count (∗), sum (q.qonorder)
  from QUOTATIONS q
  where q.partno > 200
```


Hinweis:

- sum, avg, min, max ignorieren **null**-Werte
- Aggregat-Funktionen über beliebige Ausdrücke, z. B.
  sum (price ∗ qonorder)
  zur Berechnung kurzfristiger Verbindlichkeiten.

Hinweis: Ergebnis = 1 Tupel

Allg. Fall:

```
select a, b,..., c, agg(d), ..., agg(e)
  QA(a,b,…,c, d, …, e)
  group by  d, ...,e
```

Kap. 6.3.6 Ausdrücke in SQL:

Operatoren: + - * /

null op x = **null**
x op **null** = **null**
op **null** = **null**

Vergleichsoperatoren: = < > <= >= <>

auf **integer, real, string** (lexik. Ordn.)
between: für Intervall-Abfragen
attr. **between** expr1 **and** expr 2

string-Vergleich mit **like:**
x **like** "B%"
% wildcard für beliebige Zeichenreihe
_ joker für beliebiges Einzelzeichen

z. B. **select** * **from** PROF
where name **like** "B%"

select * **from** PROF
where name **like** "Ba_er"

Mengenvergleiche:

Tupel **in** Set hier kann Tupel explizit gegeben sein oder ein QB
not in mit genau 1 Ergebnistupel

Set1 **contains** Set2

not contains

=

<>

Logische Operatoren:

and or not

LIEF:

L #	LNAME	STATUS	STADT
L1	Smith	20	London
L2	Jones	10	Paris
L3	Blake	30	Paris
L4	Clark	20	London
L5	Adams	30	Athen

TEILE:

T	TNAME	FARBE	GEWICHT
T1	Mutter	rot	12
T2	Bolzen	grün	17
T3	Schranke	blau	17
T4	Schranke	rot	14
T5	Welle	blau	12
T6	Rad	rot	19

LT:

L #	T #	A
L1	T1	3
L1	T2	2
L1	T3	4
L1	T4	2
L1	T5	1
L1	T6	1
L2	T1	3
L2	T2	4
L3	T3	4
L3	T5	2
L4	T2	2
L4	T4	3
L4	T5	4
L5	T5	5