Kapitel 6.4 Quantifizierung in SQL

"Lieferanten, die mindestens 1 Teil wie 'L2' liefern"

Prinzip: 1. Teile von 'L2' aus LT
2. L# von Lieferanten, die Teile aus 1. liefern
3. Lieferanten und L# finden

$Y \in$ LT

$X \in$ LT

$L \in$ LIEF

$\{ L \mid \exists X : (L.L\# = X.L\# \wedge L.L\# \neq \text{'L2'} \wedge \exists Y : (X.T\# = Y.T\# \wedge Y.L\# = \text{'L2'}))] \}$

ersetze: $\exists X : (L.L\# = X.L\# \wedge L.L\# \neq$ 'L2' durch
$L.L\# \in$ {Menge der X.L# ...}

$\Rightarrow$ SQL mit Range-Variablen , aber ohne Quantoren

```
select * from LIEF      L
  where L.L# ≠ 'L2'  ∧
    L.L# ∈
    select  X.L#   from  LT    X
      where X.T# ∈
        select Y.T#  from  LT   Y      } Teile
          where L# = 'L2'              } von 'L2'
```

Ohne explizite Range Variablen und ohne Quantoren

select ∗ **from** LIEF
  where L.L# ≠ 'L2' ∧
    L# ∈
    select L# **from** LT
      where T# ∈
        select T# **from** LT } Teile
          where L# = 'L2' } von 'L2'

Allgemeine Prinzipien zur Vermeidung von ∃:

∃ Y : (X.i = Y.j ∧ QA (Y))

⇓

X.i ∈ **select** Y.j **from** ... **where** QA (Y)

Prinzip zur Vermeidung von ∀

∀ Y : (X.i ≠ Y.j ∧ QA (Y))

⇓

X.i ∉ **select** Y.j **from** ... **where** QA (Y)

∃ **syntaktisch weglassen**

wird implizit ergänzt, wenn Y frei ist und in Projektions-Liste nicht vorkommt:

{ X | ∃Y : (X.i = Y.j ∧ QA (Y)) **}**

⇓

{ X | (X.i = Y.j ∧ QA (Y)) **}**

mit Duplikatelimination: wegen der teuren Duplikatelimination wird sie in SQL unterlassen (Tabelle ≠ Relation), aber erzwungen durch **unique**

select unique X ...

Beispiele: "T# aller Teile mit Alternativ-Lieferanten"

Version 1: als ***nested (korrelierte) subquery*** mit ∈

```
select X.T#
    from LT  X
      where X.T# ∈     (select   Y.T#
                  from LT  Y
                  where   Y.L# ≠ X.L#)
```

Antwort Größe ohne und mit unique??

Version 1a:

- ohne entbehrliche Range-Variablen, Mißbrauch der Relationsnamen!
- Auflösung von Mehrdeutigkeit durch Gültigkeitsbereichs-Regeln!

```
select T#
    from LT  X
      where T# ∈   (select   T#
                    from  LT
                    where    L# ≠  X.L#)
```


Version 2: mit =, aber ohne ∃ , nicht als nested subquery, sondern als Join

$X \in LT, Y \in LT$

$X \mid X.T\# = Y.T\# \wedge X.L\# \neq Y.L\#$

```
select X.T#
  from LT  X,   LT  Y
  where X. T# = Y.T# ∧ X.L# ≠  Y.L#
```

hier ist Y implizit mit ∃ quantifiziert, weil Y nicht in Projektionsliste ist.

Version 3: mit ∃ und =

$X \in LT, Y \in LT$

$\{ X \mid \exists Y : (X.T\# = Y.T\# \wedge X.L\# \neq Y.L\#) \}$

```
select *
  from LT   X
  where exists (select *
                from  LT.Y
                where   X.T# = Y.T# ∧
                      X.L# ≠  Y.L#)
```


früheres Beispiel:

"Namen der Lieferanten, die 'T2' liefern"?

Formulierung 1:

```
  select X. LNAME
    from LIEF   X, LT   Y
    where X.L# = Y.L# ∧ Y.T# = 'T2'
```

Formulierung 2: $X \in$ LIEF, $Y \in$ LT

$X \mid \exists Y : (X.L\# = Y.L\# \wedge Y.T\# = \text{'T2'})$

~ **select** ∗ **from** LIEF X

where exists (**select** ∗ **from** LT Y

where X.L# = Y.L# ∧ Y.T# = 'T2')

Formulierung 3:

select ∗ **from** LIEF X

where X.L# **in**

(**select** Y.L# **from** LT Y

where Y.T# = 'T2')

Formulierung 4: wie 3.

... **where** X.L# = **any** (...)

Möglichkeiten für ∃

X.L# = **any**(...)
≠ **any** (...)
< **any**(...)
in (...)

Hinweis: any entspricht immer ∃ in SQL, aber nicht in Englisch:
"Präsident makes more than any secretary"

Formulierungsvarianten für: **{** A | ∃X : (Q(A,X)) **}**

1. Weglassen, X ist nicht in Projektionsliste
2. **where exists**
3. **where** A **in** ...
4. **where** A = **any**

Behandlung von ∀: **{** X | ∀Y : (Q (X,Y)) **}**

X.attr. = **all** (...) nur bei singleton Ergebnis von (...) sinnvoll !

≠ **all** (...)

<

≤

ect. für weitere Vergleiche

z. B. **{** X.attr1| ∀Y (X.attr2 < Y.attr3) …**}**

in SQL: **select** X.attr1 **from** R X

where X.attr2 < all

(**select** Y.attr3 **from** ...)

Unterdrückung von ∀: ersetzen durch **not** ∃ **not**

∀ X : P(X) ersetzen durch **not** ∃ X : **not** P(X)

"Namen der Lieferanten, die nicht 'T2' liefern"

select X.LName **from** LIEF X

where 'T2' ≠ **all** (**select** Y.T#

from LT Y

where Y.L# = X.L#)

≠ **all** äquivalent zu ∉

wirkliche Bedeutung von ≠ **all:**

$\forall Y : (Y.T\# \neq \text{'T2'})$

$\Downarrow$

$\neg \neg \forall Y : (Y.T\# \neq \text{'T2'})$

$\Downarrow$

$\neg \exists : \neg (\quad ... \quad)$

$\neg \exists Y : (Y.T\# = \text{'T2'})$ liefert:

select X.LName **from** LIEF X
where not exists (**select** Y.T#
from LT Y
where Y.L# = X.L# $\wedge$ Y.T# = 'T2')

Allg. Ansatz zur Vermeidung von $\forall$:

X : $\forall Y\ Q(X,Y)$

$\Downarrow$

X : $\neg \neg \forall Y\ Q(X,Y)$

$\Downarrow$

X : $\neg \exists Y \neg Q(X,Y)$

$\Downarrow$

not exists (**select** Y ... **where** $\neg$ Q(X,Y)

d.h. die quantifizierte Variable steht erst hinter dem **select** !!!

Beispiel: "Finde Ersatzlieferanten für L2"
"L# der Lieferanten, die alle Teile liefern, die L2 liefert."

Version 1: Mit Zwischenrelation ZER
"von L2 gelieferte T#"

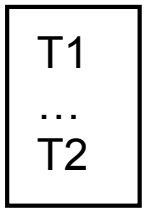
QB-T#: **select** T# **from** LT
where L# = 'L2'

create table ZER (...);
insert into ZER: (QB-T#)

Version 2: mit View-Def. für ZER

create view ZER (T#)
as
(select T# from LT
where L# = ´L2´)

ZER:=



$\forall \equiv \neg\neg\forall \equiv \neg\exists\neg$ ...

" L# von X aus LIEF:
es gibt kein Teil aus ZER, das nicht geliefert wird
von X.L#"

$X \in$ LIEF | X liefert alle Teile, die ´L2´ liefert

$X \in$ LIEF | $(\forall Z \in ZER : (\exists Y \in LT : (Z.T\# = Y.T\# \wedge Y.L\# = X.L\#)))$

$\wedge\ X.L\# \neq$ ´L2´

$|||$

$X \in$ LIEF | $(\neg\neg\ \forall\ Z \in$ ZER ...

$|||$

$X \in$ LIEF | $(\neg\exists Z \in ZER : (\neg\exists Y \in LT : (Z.T\# = Y.T\# \wedge Y.L\# = X.L\#)))$

$\wedge\ X.L\# \neq$ ´L2´


```
select * from LIEF   X
   where not exists
      (select * from ZER   Z
         where not exists
            (select * from LT   Y
               where   Z.T# = Y.T#
                  ∧ Y.L# = X.L#))
      ∧ X.L#  ≠ ´L2´
```

Mit Text-Ersetzung für ZER:

```
select * from  LIEF   X
        where not exists
           [ select Z.T#  from LT   Z
           where Z.L# = 'L2'
           and
           not exists
           (select *  from LT   Y
               where Z.T# = Y.T#
                        ∧ Y.L# = X.L#) ]
       ∧ X:L# ≠ ´L2´
```


Alternative mit Mengenvergleich:

```
select * from LIEF   X
   where (select Z.T# from LT   Z
           where Z.L# = X.L#)
         contains
          (select Y.T# from LT   Y
           where Y.L# = 'L2')
```

Teile von X

Teile von 'L2'

Kapitel 6.5 Updates in SQL

X : QA (X,Y)

select X **from** R **where** ....
analog dazu:

update R:
R.i = Ausdruck
where ... bzw.

update R
set R.1 = A1, ..., R.k = Ak
where ⟨Bedingung, die mit Hilfe von Attr. Namen definiert ist⟩

Beispiel:
"Erhöhe Status der Lieferanten in London um 5"

```
update LIEF
set STATUS = STATUS + 5
where CITY = 'LONDON'
```

} i.e. Mengen-update mit Kollektiv-Zuweisung

Ausführung: 2-stufig!!

1. Bestimme zu ändernde Tupelmenge
2. Ändere simultan

Beispiel: update LT
set A = A +1
where A < 3

keine Fixpunkt-Berechnung!

Beispiel: "Erhöhe Status um 1, außer für Lieferanten mit höchstem Status"

```
update LIEF
  set STATUS = STATUS + 1
  where STATUS < any (select STATUS
                      from LIEF )
```

Merke: any entspricht existentieller Quantifizierung !!

Änderung mehrerer Tabellen:

z. B. Umbenennung 'T1' $\rightarrow$ 'T7'
in Relationen TEILE und LT: verwende Transaktion:

```
begintrans
    update TEILE set T# = 'T7'
      where T# = 'T1';
    update LT set T# = 'T7'
      where T# = 'T1'
endtrans
```

illegale Zwischenzustände der DB, z. B. referentielle Integrität verletzt

Einfüge-Operation insert:

1 oder mehrere Tupel

```
insert into TEILE
    values ('T8', 'Bolzen', 'rot', 13)
```

Komponenten-Reihenfolge entsprechend Vereinbarung wichtig!

```
insert into LT (L#, T#)
   values ('L1', 'T8')

   co LT.Anzahl  wird null co
   betroffene Attribute angeben, (L#, T#)!
```


Einfügen mehrerer Tupel:

```
insert into LT (L#, T#) values
      ( ('L1', 'T8'),
        ('L3', 'T8'),
        ('L2', 'T3') )
```

Beispiel für Zwischenrelation ZER:

```
create table ZER (T#  char (2));
   co die von 'L2' gelieferten T# co
insert into ZER
   select T# from LT
      where L# = 'L2'
```

Lösch-Operation

delete from LIEF
  where L# = 'L3'

Verletzung von Integritätsbedingungen?

z. B. **select** L# $\subseteq$ **select** L#
  from LT **from** LIEF

Verletzungen finden durch:
select L# **from** LT
  where L# $\notin$
    (**select** L# **from** LIEF)

Relation leeren:

delete LT ~ **delete from** LT
  where true

{ hier LT = $\emptyset$, unterscheide von **drop** LT }

Lösche Lieferanten 'L2':

```
begintrans
   delete from LIEF
        where L# = 'L2';
   delete from LT
        where L# = 'L2'
endtrans
```

Ohne Transaktion:

```
delete from LT
    where L# = 'L2';
delete from LIEF
    where L# = 'L2'
```

Hinweis: Relationen, die aus Beziehungen bei Modellierungen enstanden sind bzw. "schwache Entitäten" zuerst löschen. ⇒ Referentielle Integrität!

Kap 6.6 Mächtigkeit von relationalen Sprachen

Kap 6.6.1 Abbildung SQL → rel. Algebra

```
select y.? ... z.?
    from R1 y, ..., Rk z, ...
    where Q (y, ..., z, ...)
```

Schritt 1: Bilde Cartesisches Produkt über alle zu Tupel-Var. gehörige Relationen

R1 x R2 x ... x R x ... = **U** ∋ u
{ Rj können mehrfach in U vorkommen}
U heißt "universelle Relation" für Q

Schritt 2: Baue Q (x, ..., y, ...) von innen, beginnend mit Elementarausdrücken, nach außen ab, baue relationalen Ausdruck RQ (x, ..., y, ...) der rel. Algebra entsprechend auf.
RQ als relationaler Ausdruck oder **Operatorbaum.**

Sei x.i ν y.j Elementarausdruck in Q, i.e. atomares Prädikat in Infix-Notation,
z.B. mit ν als Vergleichsoperator von SQL

x.i $\sim$ u.m y.i $\sim$ u.n wobei u.m, u.n Attribute in U seien
Sei Ausdruck Q in Pränex-Normalform, d. h. Quantoren ganz außen.

$\forall \exists$ ... (... ( ) ... ( ) ... ( ))

el. Ausdruck = Atom

keine Quantoren

x.i. ν y.i $\rightarrow$ $\sigma_{u.m \, \nu \, u.n}$ (U)

Seien A, B prädikatenlog. Formeln, die schon in Operator - Bäume RA bzw. RB übersetzt sind:

$A \wedge B \rightarrow RA \cap RB$

Hinweis: RA, RB sind artverträglich, weil RA $\subseteq$ U, RB $\subseteq$ U

$$A \vee B \quad \rightarrow \quad RA \cup RB$$
$$\neg B \quad \rightarrow \quad U \setminus B$$

Bei Übersetzung Klammern bzw. Präzedenzen befolgen.

Schritt 3: $\exists$ **Quantifizierung**: sei A schon in RA übersetzt

$\exists y\ A \rightarrow \pi_{\text{Kompliment der y-Attribute in U}}$ (RA)

Hinweis: y kann in Q in der PNF (Pränex NormalForm) nur einmal quantifiziert sein, d.h. Q hat nach systematischen Variablen-Umbenennungen die Form

$\forall z\ \exists y$ ...Q (x, y, ...)

Schritt 4: ∀-Quantifizierung:

$\{ x \mid x \in R \wedge y \in S \wedge \forall y\, P(x, y) \} \quad =$

$\sigma_{P(x, y)} (R \times S) \text{ -:- } S$

Beispiel

Stud	
Hans	2
Franz	4
Josef	5

Vorles
DBS1, Bayer
DBS2, Kossm.

hört
2, DBS1
2, DBS2
4, DBS2
5, DBS1

$\{ s \mid s \in \text{Stud} \wedge v \in \text{Vorles} \wedge \forall v\, \text{hört}(s, v) \} \quad =$

$\sigma_{\text{hört}(s, v)} (\text{Stud} \times \text{Vorles}) \text{ -:- Vorles}$

Stud	x	Vorles
Hans	2	DBS1, Bayer
Hans	2	DBS2, Kossm.
Franz	4	DBS1, Bayer
Franz	4	DBS2, Kossm.
Josef	5	DBS1, Bayer
Josef	5	DBS2, Kossm.

hört
2, DBS1
2, DBS2
4, DBS2
5, DBS1

$\sigma_{\text{hört(s, v)}}$ (Stud x Vorles) =

Hans	2	DBS1, Bayer
Hans	2	DBS2, Kossm.
Franz	4	DBS2, Kossm.
Josef	5	DBS1, Bayer

-:- **Vorles**

DBS1, Bayer
DBS2, Kossm.

=

Hans	2

Beispiel: "Namen der Lieferanten, die Teil 1 oder 2 liefern"

1) **select** X.LName
   from LIEF X, LT Y
   where X.L# = Y.L# $\wedge$
   (Y.T# = 'T1' $\vee$ Y.T# = 'T2')

2) bzw. mit **where** $\exists$Y : (...)
   select X.Lname
   from LIEF X
   where exists (select Y **from** LT Y
   where X.L# = Y.L# $\wedge$
   (Y.T# = 'T1' $\vee$ Y.T# = 'T2') **)**

U = LIEF x **LT**

L#	LNAME	STATUS	STADT	L#	T#	A
1	2	3	4	5	6	7

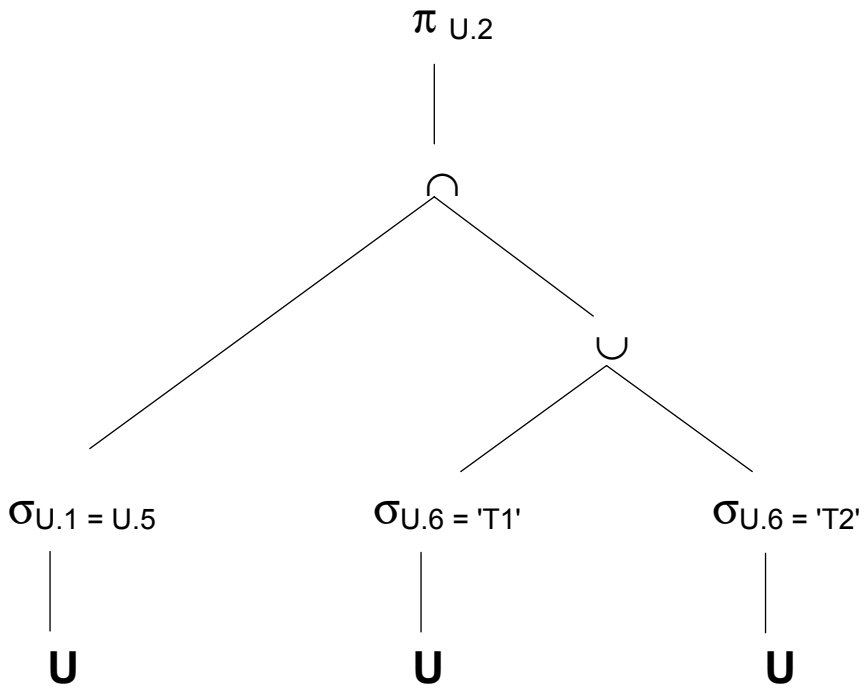
rel. Ausdruck entprechend Formulierung 1):

$$\pi_{U.2} \left[\sigma_{U.1 = U.5} (U) \cap (\sigma_{U.6 = 'T1'} (U) \cup \sigma_{U.6 = 'T2'} (U)) \right]$$

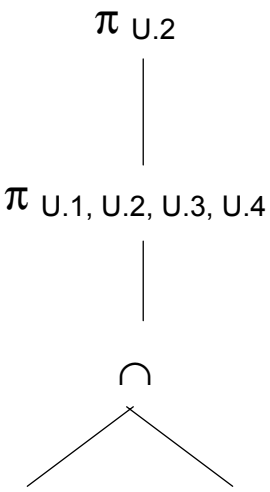
rel. Ausdruck entprechend Formulierung 2):

$$\pi_{U.2} \left(\pi_{U.1, U.2, U.3, U.4} \left[\sigma_{U.1=U.5} (U) \cap (\sigma_{U.6=´T1´} (U) \cup \sigma_{U.6=´T2´} (U)) \right] \right)$$

Operatorbaum 1):



Operatorbaum 2): mit $\exists y$



wie bei 1)

Offensichtliche Optimierungsmöglichkeit:
Kombination der beiden π

6.6.2 Abb. Relationale Algebra → SQL

1. Mengenoperationen: siehe auch Kapitel 3.2

Union-Operation: R1 ∪ R2

$R1 \cup R2 \quad : \quad y \in R1, z \in R2$

$\{ x \mid x = y \vee x = z\}$

select x **from** R1 y, R2 z
where $x = y \vee x = z$

Syntaktisches Problem: kein Bereich für x, um x unabhängig zu instantiieren

Lösung 1: Mengen-Algebra für Bereich?

select x **from** (R1 $\cup$ R2) x **where true**

Lösung 2: Kunstgriff: führe alle Domänen D_i im Typsystem als einstellige Relationen ein

Sei R1 $\subseteq$ D1 x D2 x ... x Dk

R2 $\subseteq$ D1 x D2 x ... x Dk

Wegen Strukturverträglichkeit von R1, R2

select x1, x2, ..., xk

from D1 x1, D2 x2, ..., Dk xk, R1 y, R2 z

where (x1 = y.1 $\wedge$ x2 = y.2 $\wedge$ ... $\wedge$ xk = y.k)

$\vee$ (x1 = z.1 $\wedge$ x2 = z.2 $\wedge$ ... $\wedge$ xk = z.k)

Schnitt: R1 $\cap$ R2 : x $\in$ R1 y $\in$ R2

select x **from** R1 x, R2 y

where x = y

Differenz: R1 \ R2 $x \in R1$ $y \in R2$

select x **from** R1 x
where $\neg \exists$ (**select** $*$ **from** R2 y
where x = y)

bzw. mit $\forall \neg$ $\forall y (x \neq y)$

...

where x $\neq$ **all** (**select** $*$ **from** R2 y)

2. Weitere Operationen

Projektion: $\pi_{A1, .., Ak}(R)$:

select x.A1, x.A2, ..., x.Ak **from** R x
bzw.
select A1, ..., Ak **from** R

Selektion bzw. Restriktion: $\sigma_{P(t)}(R)$:

select t **from** R t
where P (t)

Cartesisches Produkt: R x S

sei R (A1, ..., Ak), S (B1, ..., Bm)

```
select x.A1, ..., x.Ak, y.B1, ..., y.Bm
    from R x, S y
```

Equi-Join $\varepsilon_{i,j}$: R $\varepsilon_{i,j}$ S

```
select x.A1, ..., x.Ak, y.B1, ..., y.Bm
            from R x, S y
            where x.Ai = y.Bj
```


Natural Join $\iota_{i,j}$: R $\iota_{i,j}$ S

```
select x.A1, ..., x.Ak, y,B1, ..., y.Bj-1, y.Bj+1, ... y.Bm
        from    R  x,  S  y
        where   x.Ai = y.Bj
```

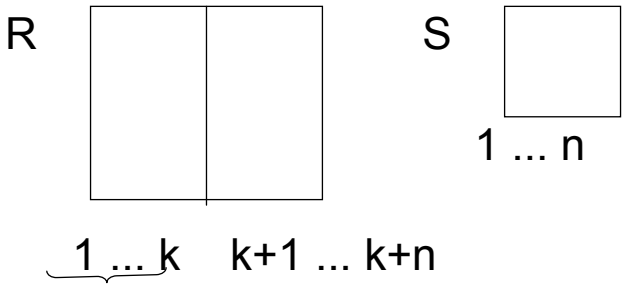
Verallgemeinerter Verbund: R $\nu_{P(R,S)}$ S

```
select x.A1, ..., x.Ak, y.B1, ..., y.Bm
        from R  x,  S  y
        where   P(x,y)
```

Hinweis: für zusammengesetzte rel. Ausdrücke ist volle Orthogonalität von SQL erforderlich oder Übersetzung in Folge von SQL-Anweisungen mit View-Definitionen oder Hilfsvariablen für relationale Zwischenergebnisse

$\iota \equiv \pi \circ \varepsilon_{i,j}$ $\nu \equiv \sigma \circ x$

Division: **R / S :**



R'= **select** r.1, r.2, ..., r.k **from** R r

$R / S = \{r' : r' \in R' \wedge \forall s \in S (\exists r \in R (r' \circ s = r))\}$

$\quad ... \quad \neg \neg \forall \quad ...$

$= \{ r' : r' \in R' \wedge \neg \exists s \in S (\neg \exists r \in R : (r' \circ s = r))\}$

```
select r' from R' r'
  where not exists
    select s from S s
      where not exists
        select r from R  r
          where r' ∘ s = r
```

bzw. wegen fehlenden Tupel-Vergleichs in SQL :

expandiere **where** r´ ∘ s = r zu :

... **where**

r'.1 = r.1 ∧ r'.2 = r.2 ∧ ... ∧ r'.k = r.k

∧ s.1 = r.k + 1 ∧ ... ∧ s.n = r.k + n

Hier mehrere SQL statements bzw. mit voller Orthogonalität:

select r' **from**

(**select** r.1, ..., r.k **from** R r) r'

where not exists ...

Satz: Relationale Algebra und SQL (bzw. Tupelkalkül und Domänenkalkül) haben gleiche Mächtigkeit.

Bew: Übersetzbarkeit in beide Richtungen (informell).

Def: ***Relationale Vollständigkeit***

Eine Sprache heißt „relational vollständig“, wenn sie die Ausdrucksmächtigkeit der relationalen Algebra (bzw. von SQL) hat.

Frage: Allgemeine Berechenbarkeit mit SQL?

z.B. primitiv rekursiv
μ-rekursiv
Turing-Mächtigkeit, ...

Antwort: ***Einbettung von SQL in host Sprachen!***
siehe spätere Kapitel.