3 Relationale DB-Sprachen

3.1 Klassifikation

(Aspekte von) DB-Sprachen, Teilsprachen:

- DDL (Data Definition Language); beinhaltet Schemadefinition

- DML (Data Manipulation Language); beinhaltet Anfragen und Änderungen

Klassifikation von Anfragesprachen:

(a) deskriptiv:

Auszuwählende Objekte werden als Gesamtheit beschrieben (Was?)

(b) prozedural:

Ergebnis wird durch Folge von Operationen konstruiert (Wie?)

(b1) Operationen auf Objektmengen, z.B. Relationen

(b2) Operationen auf einzelnen Objekten, z.B. Tupeln, Sätzen

3.2 Relationenalgebra (RA)

$\mathcal{R}$ sei Menge aller endlichen Relationen (inklusive Schemata) mit Operationen $\omega : \mathcal{R}(\times ... \times \mathcal{R}) \to \mathcal{R}$

Grundlage für Anfragen der Form

τ (RelName$_1$, ..., RelName$_k$) : DB-Zustände $\to \mathcal{R}$

wobei τ relationaler Term, d.h. Operation oder korrekte Zusammensetzung von Operationen, ist

Grundoperationen:

Gegeben seien endliche Relationen $R, S, ...$ mit zugehörigen Schemata $R(A_1, ..., A_n), S(B_1, ..., B_m), ...$

Bezeichnung: $Rang(R) = n$

1. Vereinigung $R \cup S$
2. Differenz $R - S$
3. (Kartesisches) Produkt $R \times S$
4. Projektion $\pi_{\overline{A}}(R)$ mit $\overline{A} = A_{i_1}, ..., A_{i_k}$ und $1 \leq i_1, ..., i_k \leq Rang(R)$
5. Selektion $\sigma_\varphi(R)$ mit φ atomare Formel
6. Umbenennung $\delta_{C \leftarrow A_i}(R)$ mit $C \notin \{A_1, ..., A_n\}$

Abgeleitete Operationen:

1. Durchschnitt $R \cap S$

2. verallgemeinerte Selektion $\sigma_{\varphi}(R)$

   φ ist logische Verknüpfung mittels $\wedge$, $\vee$ und $\neg$ von atomaren Formeln

3. Verbund $R *_{A_i \Theta B_j} S$

   Equiverbund, falls $\Theta \equiv =$

4. Natürlicher Verbund $R * S$

5. Division $R : S$

Relationenalgebra und Anfragesprachen

Relationale Operationen lassen sich zu Termen über Relationennamen und konstanten Relationen zusammensetzen; diese relationalen Terme definieren eine prozedurale Anfragesprache

Relationenalgebra ist ein Maß für die **Ausdrucksfähigkeit** von Datenmanipulationssprachen $\mathcal{L}$:

- $\mathcal{L}$ **relational vollständig** gdw. jeder Term der Relationenalgebra kann in $\mathcal{L}$ simuliert werden

- $\mathcal{L}$ **streng relational vollständig** gdw. jeder Term der Relationenalgebra kann durch eine einzige Anweisung in $\mathcal{L}$ simuliert werden

Grenzen der Relationenalgebra

Es gibt keinen Term der Relationenalgebra, der zu jeder zweistelligen Relation R deren transitive Hülle

$$R^* = R \cup \{(x, y) \mid \exists z_1, ..., z_k :$$

$$(x, z_1) \in R \wedge (z_1, z_2) \in R \wedge ... \wedge (z_k, y) \in R\}$$

berechnet

3.3 Relationenkalküle

Relationenkalküle sind Grundlage für deskriptive Anfragesprachen

3.3.1 Bereichskalkül (BK)

Syntax: (über gegebenen Datentypen und gegebenem relationalem DB-Schema)

Terme:

(i) Konstanten zu Datentypen

(ii) Bereichsvariablen über Datentypen
z.B. x:integer, y:string

(iii) Wenn $u_1, ..., u_n$ Terme (bzgl. $D_1, ..., D_n$) und f Datentypoperation (bzgl. $D_1 \times ... \times D_n \to D$), so ist $f(u_1, ..., u_n)$ ein Term (bzgl. D)

Atomare Formeln:

(i) $\Theta(u_1, ..., u_n)$, wobei Θ boolsche Operation (z.B. $n = 2$ und $\Theta \equiv <$, also $< (u_1, u_2)$ oder $u_1 < u_2$)

(ii) $R(u_1, ..., u_n)$, wobei R Relationenname mit Schema $R(A_1 : D_1, ..., A_n : D_n)$ und u_i Term bzgl. D_i $(i = 1..n)$

Formeln:

(i) Jede atomare Formel ist eine Formel

Alle Vorkommen von Variablen darin sind frei

(ii) Wenn φ_1, φ_2 Formeln sind, so auch $(\varphi_1 \wedge \varphi_2)$, $(\varphi_1 \vee \varphi_2)$, $\neg(\varphi_1)$

Freie bzw. gebundene Vorkommen von Variablen in φ_1, φ_2 bleiben frei bzw. gebunden in $(\varphi_1 \wedge \varphi_2)$, $(\varphi_1 \vee \varphi_2)$, $\neg(\varphi_1)$

(iii) Wenn φ Formel und x Variable, so sind auch $\exists x(\varphi)$ und $\forall x(\varphi)$ Formeln

Freie Vorkommen von x in φ sind in $\exists x(\varphi)$, $\forall x(\varphi)$ gebunden, alle anderen Variablenvorkommen bleiben frei bzw. gebunden

Quantifizierungen dürfen auch auf Datentypen hinweisen, der Typ der Variablen ist aber bereits durch ihre Deklaration gegeben: Erlaubt ist $\exists x : D(\varphi)$ und $\forall x : D(\varphi)$

(iv) Klammerung wie üblich

Semantik: Fest gegeben seien Wertebereiche $|\, D \,|$ zu Datentypen D und zugehörige Operationen; es sei σ ein Zustand und β eine Belegung, d.h.:

$$\begin{array}{lcccc} \sigma & : & \text{Relationennamen} & \rightarrow & \text{Relationen} \\ & & R & \mapsto & \sigma(R) \\ \beta & : & \text{Variablen} & \rightarrow & \text{Werte} \\ & & x : D & \mapsto & \beta(x) \in |\, D \,| \end{array}$$

$[\sigma, \beta](u)$ bezeichnet den **Wert** des Termes u im Zustand σ unter der Belegung β

$[\sigma, \beta] \models \varphi$ bedeutet: Formel φ **gilt** im Zustand σ unter der Belegung β

Definition:

$[\sigma, \beta] \models R(u_1, ..., u_n)$ **gdw.**
$([\sigma, \beta](u_1), ..., [\sigma, \beta](u_n)) \in \sigma(R)$

$[\sigma, \beta] \models \Theta(u_1, ..., u_n)$ **gdw.** $[\sigma, \beta](\Theta(u_1, ..., u_n))$ wahr ist

$[\sigma, \beta] \models \varphi_1 \wedge \varphi_2$ **gdw.** $[\sigma, \beta] \models \varphi_1$ und $[\sigma, \beta] \models \varphi_2$

$[\sigma, \beta] \models \varphi_1 \vee \varphi_2$ **gdw.** $[\sigma, \beta] \models \varphi_1$ oder $[\sigma, \beta] \models \varphi_2$

$[\sigma, \beta] \models \neg\varphi_1$ **gdw.** nicht $[\sigma, \beta] \models \varphi_1$

$[\sigma, \beta] \models \exists x(\varphi)$ **gdw.** es gibt Belegung β' mit
$\beta'(Y) = \beta(Y)$ für $Y \neq x$ und $[\sigma, \beta'] \models \varphi$ ist wahr

$[\sigma, \beta] \models \forall x(\varphi)$ **gdw.** für alle Belegungen β' mit
$\beta'(Y) = \beta(Y)$ für $Y \neq x$ ist $[\sigma, \beta'] \models \varphi$ wahr

Ausdrücke des Bereichskalküls und Anfragen

Syntax: $\{x_1[: D_1], ..., x_n[: D_n] \mid \varphi\}$ wobei φ Formel mit freien Variablen $x_1 : D_1, ..., x_n : D_n$; $x_1, ..., x_n$ sind die Ergebnisvariablen und φ nennt man Qualifikation

Semantik: (in gegebenem Zustand σ)
$\{x_1, ..., x_n \mid \varphi\}$ bestimmt folgende Relation über
$\mid D_1 \mid \times ... \times \mid D_n \mid$:
$\{(d_1, ..., d_n) \mid$ es gibt eine Belegung β mit
$\beta(x_i) = d_i (i = 1..n),$ so daß $[\sigma, \beta] \models \varphi\}$ gilt

Definition: Ein Ausdruck heißt sicher, wenn er bestimmte syntaktische Einschränkungen erfüllt (ein sicherer Ausdruck bestimmt in jedem Zustand eine endliche Relation)

Satz: Der sichere Anteil des BK (und damit der BK) ist streng relational vollständig, d.h. zu jedem Term τ der Relationenalgebra gibt es einen äquivalenten sicheren Ausdruck α des BK

τ und α sind äquivalent gdw. sie in jedem Zustand die gleiche Relation bestimmen

Satz: Zu jedem sicheren Ausdruck des BK gibt es einen äquivalenten Term der Relationenalgebra

3.3.2 Tupelkalkül (TK)

Syntax: Terme

(i) Konstanten zu Datentypen

(ii) Tupelvariablen r, s, t, ... über dem Kreuzprodukt von Datentypen zusammen mit Komponentennamen $r : (A_1 : D_1, ..., A_n : D_n)$

(iii) Bildung von Komponenten $r.A_i$

(iv) Anwendung von Datentypoperationen

Atomare Formeln

(i) $\Theta(u_1, ...u_n)$ boolscher Term

(ii) $r = s$, wobei r, s Tupelvariable mit passenden Datentypen

(iii) $R(r)$, wobei R Relationenname und r Tupelvariable mit passenden Datentypen

Formeln: analog zum Bereichskalkül

Semantik: (nur Abweichungen vom Bereichskalkül)

Belegung β:

$$\begin{array}{lcll} \beta & : & \text{Variablen} & \rightarrow \quad \text{Tupel} \\ & & r & \mapsto \quad \beta(r) = (d_1, .., d_n) \in \mid D_1 \mid \times ... \times \mid D_n \mid \end{array}$$

$[\sigma, \beta](r.A_i) \;=\; \beta(r)_i \quad (= d_i)$

$[\sigma, \beta] \models R(r)$ gdw. $\beta(r) \in \sigma(R)$

Abkürzung: Gegeben $R(A_1 : D_1, ..., A_n : D_n)$

$\exists\ r : R\ (\varphi) :\Leftrightarrow \exists\ r : (A_1 : D_1, ..., A_n : D_n)\ (R(r) \wedge \varphi)$

$\forall\ r : R\ (\varphi) :\Leftrightarrow \forall\ r : (A_1 : D_1, ..., A_n : D_n)\ (R(r) \Rightarrow \varphi)$

Tupelkalkülausdrücke die nur solche Quantifizierungen verwenden und Komponenten von Ausgabevariable existentiell binden sind sicher

Ausdrücke des Tupelkalküls

$\{r : (A_1[: D_1], ..., A_n[: D_n]) \mid \varphi\}$

wobei r einzige freie Variable in φ; dies bestimmt:

$\{\beta(r) \mid \beta \text{ Belegung, so daß } [\sigma, \beta] \models \varphi\}$

Satz: Zu jedem (sicheren) Ausdruck des BK gibt es einen äquivalenten (sicheren) Ausdruck des TK

Satz: Zu jedem (sicheren) Ausdruck des TK gibt es einen äquivalenten (sicheren) Ausdruck des BK

Es sind also

1. Relationenalgebra,
2. der sichere Anteil des Bereichskalküls und
3. der sichere Anteil des Tupelkalküls

in ihrer Ausdrucksfähigkeit äquivalent

3.4 Kalkülbasierte Sprachen

Basierend auf Tupelkalkül mit
(r:R)-Quantifizierungen: SQL, QUEL

Basierend auf Bereichskalkül: QBE

Beispielschema: KAL-Schema

KUNDE(KName, KAdr, Kto)

AUFTRAG(KName, Ware, Menge)

LIEFERANT(LName, LAdr, Ware, Preis)

3.4.1 SQL (Structured Query Language)

Zunächst Anfragen im SQL-Kern:
select-from-where-Blöcke mit and, or, not und
Unteranfragen mittels exists, in, all und any und union
nur auf oberster Ebene

Grundschema:

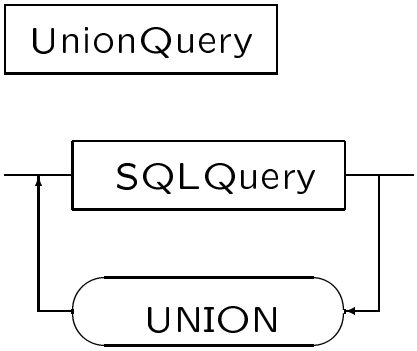
```
select   'Resultatsliste'
from     'Relationenliste'
where    'Prädikat'
```

Es gilt: Jeder sichere Ausdruck des Tupelkalküls kann durch eine SQL-Anfrage äquivalent dargestellt werden

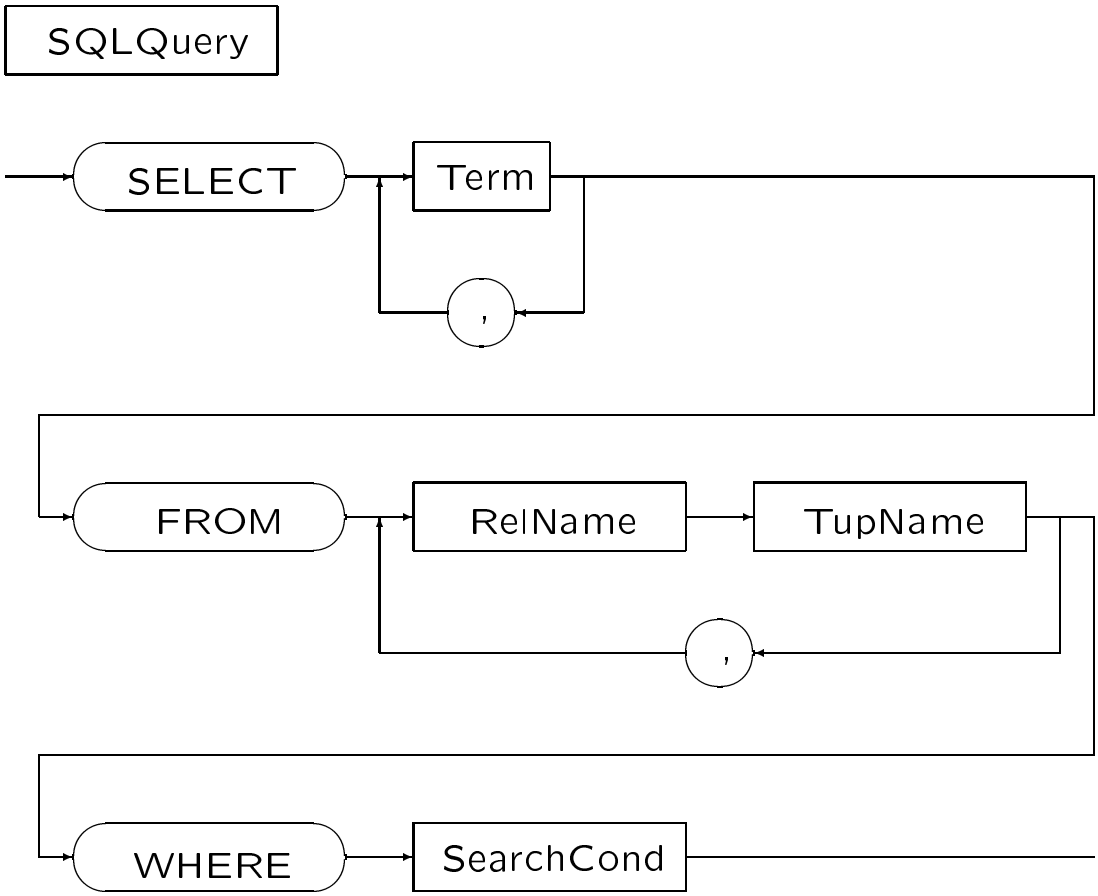
$\Rightarrow$

SQL (sogar der SQL-Kern) ist streng **relational vollständig**

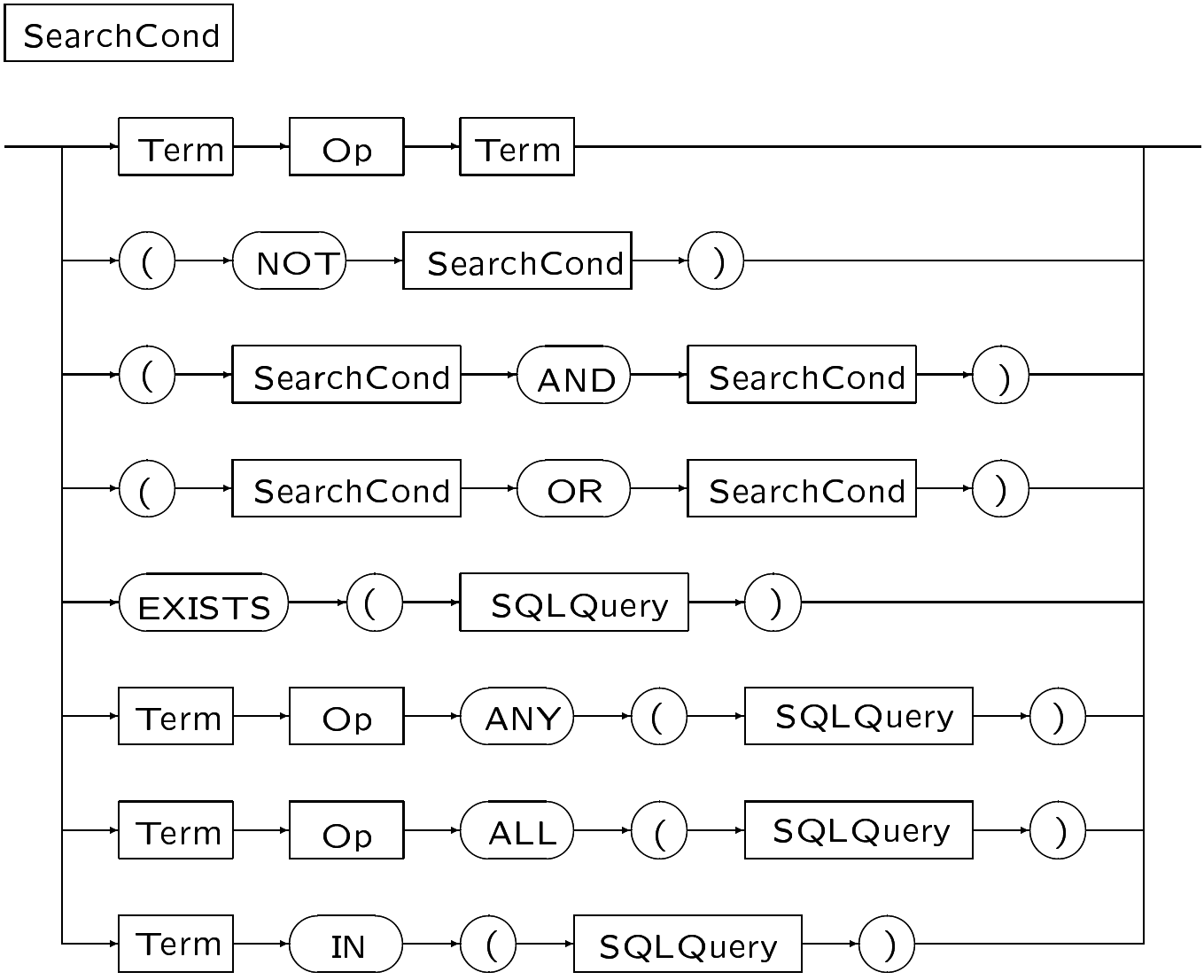
Syntax von UnionQuery



Syntax von SQLQuery



Syntax von SearchCond



Übersetzung SQL in den TK: Voraussetzungen

τ, τ_1, τ_2, ... sind Terme und φ, φ', ... Formeln

- Gegebene SQL-Anfrage SELECT τ_1, ..., τ_n FROM R_1 s_1, ..., R_m s_m WHERE φ und alle Unteranfragen sind vollständig mit expliziten Tupelvariablen qualifiziert:

  Falls Attribut A_j in Resultatsterm τ_i oder in Formel φ auftaucht, ist Auftreten von der Form $s_i.A_j$, wobei s_i in der FROM Klausel (oder im Falle einer Unteranfrage u.U. im FROM-Teil einer äußeren Anfrage) auftaucht und A_j Attribut einer entsprechenden Relation R_i ist

- Namen von Tupelvariablen sind eindeutig ($i \neq j \Rightarrow s_i \neq s_j$)

- Resultatsterme und Formel φ verwenden nur deklarierte Tupelvariable

- Resultatsterme τ_i können auch Konstanten sein

- Bei Vergleichen wie '$\tau_1 = \tau_2$' oder '$\tau_1 =$ ANY (SELECT τ_2 FROM ... WHERE ...)' haben τ_1 and τ_2 den gleichen Datentyp

- Für UNION-Ausdrücke wird angenommen, daß τ_i und τ_i' den gleichen Datentyp haben

Übersetzung SQL in den TK: Regeln Teil 1

$$\textbf{sql2tc}[\![\ \text{SELECT } \tau_1, \ldots, \tau_n \text{ FROM } R_1\ s_1, \ldots, R_m\ s_m \text{ WHERE } \varphi\ ]\!] :=$$
$$\{\ r : (\ \text{Res}_1, \ldots, \text{Res}_n\ ) \mid (\ \exists\ s_1{:}R_1, \ldots, s_m{:}R_m\ )$$
$$(\ r.\text{Res}_1 = \tau_1 \wedge \ldots \wedge r.\text{Res}_n = \tau_n \wedge \textbf{sql2tc}[\![\ \varphi\ ]\!]\ )\ \}$$

$$\textbf{sql2tc}[\![\ (\ \text{NOT } \varphi\ )\ ]\!] := (\ \neg\ \textbf{sql2tc}[\![\ \varphi\ ]\!]\ )$$

$$\textbf{sql2tc}[\![\ (\ \varphi_1 \text{ AND } \varphi_2\ )\ ]\!] :=$$
$$(\ \textbf{sql2tc}[\![\ \varphi_1\ ]\!] \wedge \textbf{sql2tc}[\![\ \varphi_2\ ]\!]\ )$$

$$\textbf{sql2tc}[\![\ (\ \varphi_1 \text{ OR } \varphi_2\ )\ ]\!] :=$$
$$(\ \textbf{sql2tc}[\![\ \varphi_1\ ]\!] \vee \textbf{sql2tc}[\![\ \varphi_2\ ]\!]\ )$$

$$\textbf{sql2tc}[\![\ \tau_1\ \omega\ \tau_2\ ]\!] := \tau_1\ \omega\ \tau_2$$

$$\textbf{sql2tc}[\![\ \tau\ \omega \text{ ALL } (\ \text{SELECT } s_i.\text{A FROM } R_1\ s_1, \ldots, R_i\ s_i, \ldots, R_m\ s_m \text{ WHERE } \varphi\ )\ ]\!] :=$$
$$(\ \forall\ s_i{:}R_i\ )$$
$$((\exists\ s_1{:}R_1, \ldots, s_{i-1}{:}R_{i-1}, s_{i+1}{:}R_{i+1}, \ldots, s_m{:}R_m)$$
$$\textbf{sql2tc}[\![\ \varphi\ ]\!]) \Rightarrow \tau\ \omega\ s_i.\text{A})$$

$$\textbf{sql2tc}[\![\ \tau\ \omega \text{ ANY } (\ \text{SELECT } s_i.\text{A FROM } R_1\ s_1, \ldots, R_i\ s_i, \ldots, R_m\ s_m \text{ WHERE } \varphi\ )\ ]\!] :=$$
$$(\ \exists\ s_i{:}R_i\ )$$
$$((\exists\ s_1{:}R_1, \ldots, s_{i-1}{:}R_{i-1}, s_{i+1}{:}R_{i+1}, \ldots, s_m{:}R_m)$$
$$\textbf{sql2tc}[\![\ \varphi\ ]\!]) \wedge \tau\ \omega\ s_i.\text{A})$$

Übersetzung SQL in den TK: Regeln Teil 2

sql2tc⟦ τ IN (SELECT s_i.A
FROM R_1 s_1, ..., R_i s_i, ..., R_m s_m
WHERE φ) ⟧ :=

(∃ s_i:R_i)
(((∃ s_1:R_1, ..., s_{i-1}:R_{i-1}, s_{i+1}:R_{i+1}, ..., s_m:R_m)
sql2tc⟦ φ ⟧) ∧ τ = s_i.A)

sql2tc⟦ EXISTS (SELECT r_1.A_1, ..., r_n.A_n
FROM R_1 s_1, ..., R_m s_m
WHERE φ) ⟧ :=

(∃ s_1:R_1, ..., s_m:R_m) **sql2tc**⟦ φ ⟧

sql2tc⟦ SELECT τ_1, ..., τ_n
FROM R_1 s_1, ..., R_m s_m
WHERE φ

UNION

SELECT τ_1', ..., τ_n'
FROM R_1' s_1', ..., R_k' s_k'
WHERE φ' ⟧ :=

{ r : (Res_1, ..., Res_n) |

(∃ s_1:R_1, ..., s_m:R_m)
(r.Res_1 = τ_1 ∧ ... ∧ r.Res_n = τ_n ∧ **sql2tc**⟦ φ ⟧)
∨
(∃ s_1':R_1', ..., s_k':R_k')
(r.Res_1 = τ_1' ∧ ... ∧ r.Res_n = τ_n' ∧ **sql2tc**⟦ φ' ⟧) }

UNION-Regel kann auf den Fall mit mehr als 2 UNION-Ausdrücken verallgemeinert werden

Übersetzung SQL in den TK: Theoreme

(a) τ_1 IN(SELECT τ_2 FROM ρ WHERE φ)
$\Leftrightarrow$
$\tau_1 =$ ANY(SELECT τ_2 FROM ρ WHERE φ)

(b) NOT(τ_1 ω ALL (SELECT τ_2 FROM ρ
WHERE φ))
$\Leftrightarrow$
τ_1 $\overline{\omega}$ ANY(SELECT τ_2 FROM ρ WHERE φ)

(c) NOT(τ_1 ω ANY (SELECT τ_2 FROM ρ
WHERE φ))
$\Leftrightarrow$
τ_1 $\overline{\omega}$ ALL(SELECT τ_2 FROM ρ WHERE φ)

(d) τ_1 ω ANY(SELECT τ_2 FROM ρ WHERE φ)
$\Leftrightarrow$
EXISTS (SELECT $\tau(\rho)$ FROM ρ
(WHERE (φ) AND τ_1 ω τ_2)

(e) τ_1 ω ALL(SELECT τ_2 FROM ρ WHERE φ)
$\Leftrightarrow$
NOT EXISTS (SELECT $\tau(\rho)$ FROM ρ
WHERE (φ) AND τ_1 $\overline{\omega}$ τ_2)

In (b) und (c) ist $\overline{\omega}$ die Negation von ω, e.g. $\overline{\leq}$:=>
In (d) and (e) bezieht sich $\tau(\rho)$ alle Attribute in ρ
Es gilt $\rho \equiv$ R_1 s_1, ... , R_m s_m

Übersetzung SQL in die RA (ohne Optimierung)

sql2ra⟦SELECT $\tau_1, ..., \tau_n$ FROM $d_1, ..., d_m$ WHERE φ ⟧
:= $\pi_{\tau_1,...,\tau_n}$(**sql2ra**⟦φ⟧($rename⟦d_1⟧ \times ... \times rename⟦d_m⟧$))

sql2ra⟦SQL-Bedingung⟧(RA-Ausdruck) wie folgt:

sql2ra⟦φ_1 AND φ_2⟧(r) :=
sql2ra⟦φ_1⟧(r) ∩ **sql2ra**⟦φ_2⟧(r)

sql2ra⟦φ_1 OR φ_2⟧(r) :=
sql2ra⟦φ_1⟧(r) ∪ **sql2ra**⟦φ_2⟧(r)

sql2ra⟦NOT φ_1⟧(r) := **sql2ra-**⟦φ_1⟧(r)

sql2ra⟦τ_1 ω τ_2⟧(r) := $\sigma_{\tau_1\ \omega\ \tau_2}$(r)

sql2ra⟦EXISTS
(SELECT * FROM $d_1, ..., d_m$ WHERE φ)⟧(r) :=
$\pi_{attrs⟦r⟧}$(**sql2ra**⟦φ⟧(
r × $rename⟦d_1⟧ \times ... \times rename⟦d_m⟧$))

sql2ra⟦τ_1 ω ANY
(SELECT τ_2 FROM $d_1, ..., d_m$ WHERE φ)⟧(r) :=
$\pi_{attrs⟦r⟧}$(**sql2ra**⟦φ⟧($\sigma_{\tau_1\ \omega\ \tau_2}$(
r × $rename⟦d_1⟧ \times ... \times rename⟦d_m⟧$)))

sql2ra⟦τ_1 ω ALL
(SELECT τ_2 FROM $d_1, ..., d_m$ WHERE φ)⟧(r) :=
sql2ra-⟦τ_1 $\overline{\omega}$ ANY
(SELECT τ_2 FROM $d_1, ..., d_m$ WHERE φ)⟧(r)

sql2ra-⟦SQL-Bedingung⟧(RA-Ausdruck) wie folgt:

sql2ra-⟦φ⟧(r) := r - $\pi_{attrs⟦r⟧}$(**sql2ra**⟦φ⟧(r))

Mit $R(A_1, ..., A_n)$ gilt $rename⟦R\ r⟧$:=
$\delta_{r.A_1 \leftarrow A_1,...,r.A_n \leftarrow A_n}(R) = \delta_{r.A_1 \leftarrow A_1}(\ ...\ \delta_{r.A_n \leftarrow A_n}(R)\ ...\ )$

SQL(-Kern): Weitere Sprachmittel

- Änderungen
  - Einfügen:
    insert into 'Relation' values ('Werte')
    insert into 'Relation' ('Select-Anfrage')
  - Löschen:
    delete from 'Relation' where 'Prädikat'
  - Modifizieren:
    update 'Relation' set 'Attribut' = 'Ausdruck'
    where 'Prädikat'

- Tabellen- und Indexdefinition:
  create table 'Relation' ('Attributliste')
  create [unique] index
  on 'Relation' ('Attributliste')

- Anfragen mit Resultatstermen

- Abfrage auf Nullwerte

- Ausgabesortierung:
  select ... from ... where ...
  order by 'Attributliste'

3.4.2 QUEL (QUEry Language)

Anfragen:

Grundform: (im QUEL-Kern)

range of r_1 is R_1

...

range of r_k is R_k

retrieve [into S] [unique] ([A_1 =]u_1, ..., [A_n =]u_n)

[where φ]

[sort by $A_{i_1}, ..., A_{i_k}$]

Erklärungen:

r_1, ..., r_k Tupelvariablen

R_1, ..., R_k, S Relationennamen

A_1, ..., A_n Attributnamen (optional)

u_1, ..., u_n Datenterme

φ Formel des Tupelkalküls mit freien Variablen $r_1, ..., r_k$

- ohne Quantoren und

- ohne Prädikate R(...)

$\widehat{=}$ Selektionsformel

Semantik:

$[S :=]\{s : (A_1, ..., A_n) \mid \exists\ r_1 : R_1, ..., \exists\ r_k : R_k$

$(\varphi \wedge s.A_1 = u_1 \wedge ... \wedge s.A_n = u_n)\}$

Folgerung: Nur Ausdrücke des Tupelkalküls der Form

$\{r \mid \exists$ ...$\exists$ ⟨Rest ohne Quantoren⟩$\}$

(oder äquivalente Ausdrücke) lassen sich durch (je) eine QUEL-Anfrage darstellen

$\Rightarrow$ Der QUEL-Kern ist **nicht streng relational vollständig**

QUEL-Änderungen

range of r is R

Löschen: delete r where φ

Einfügen: append to R(...) where φ

Ändern: replace r(...) where φ

Durch Folgen von QUEL-Anweisungen lassen sich beliebige relationale Terme darstellen

$\Rightarrow$ QUEL-Kern ist **relational vollständig**

3.4.3 QBE (Query by Example)

Sprachelemente: Tabellengerüste mit Einträgen und Condition-Box

Relationenname	Attribut-1	...	Attribut-n
{ 'leer' | [P.] | ¬ }	'BspElement'	...	'BspElement'
...	...	...	...
{ 'leer' | [P.] | ¬ }	'BspElement'	...	'BspElement'

CONDITIONS
'Prädikat'

'BspElement' ::= 'leer' |

[P.] ['Vergleichsoperator'] {'Variable'|'Konstante'}

Erklärungen:

- Beispielelemente (BspElement) $\widehat{=}$ Bereichsvariablen
- leere Spalten $\widehat{=}$ (implizite) paarweise verschieden Bereichsvariablen
- Zeile in Relation $R \mathrel{\widehat{=}} R(u_1, ..., u_n) \wedge \varphi$
  wobei $u_1, ..., u_n$ Terme in Spalten 1 bis n und φ Konjunktion der Zeilenbedingungen

Semantik/Ausdrucksfähigkeit von QBE

QBE-Anfrage (mit 'P.' nur in einer Zeile) $\widehat{=}$

$\{x_1, ...x_m \mid \exists\ y_1, ...\exists\ y_n$

$[\bigwedge_i \langle i\text{-te positive Zeile}\rangle$

$\wedge \bigwedge_j \neg[\exists\ z_1, ...\exists\ z_p \langle j\text{-te negative Zeile}\rangle]$

$\wedge \bigwedge_k \langle k\text{-te Bedingung in Condition-Box}\rangle]\}$

$x_1, ..., x_m$ alle (impliziten/expliziten) Variablen mit P.

$y_1, ..., y_n$ alle restlichen Variablen in positiven Zeilen

$z_1, ..., z_p$ restlichen impliziten Variablen in j-ter negierter Zeile

Ausgabevariablen (x_i) und restliche Variablen (y_j) sind in mindestens einer Zeile positiv gebunden; in der Condition-Box kommen nur solche Variablen (x_i oder y_j) oder Konstanten vor

Folgerung: Als Semantik von einzelnen QBE-Anfragen ergeben sich bis auf Äquivalenz nur folgende Ausdrücke des Bereichskalküls:

$\{... \mid \exists ... \exists ... (... \wedge \neg(\exists ... \exists ...) \wedge ...)\} =$

$\{... \mid \exists ... \exists ... \forall ... \forall ... \langle$Rest ohne Quantoren$\rangle\}$

$\Rightarrow$ Der QBE-Kern ist **nicht streng relational vollständig**

Jedoch kann jeder Term der Relationenalgebra durch eine Folge von QBE-Anfragen simuliert werden

$\Rightarrow$ Der QBE-Kern ist **relational vollständig**

3.5 Weitere Sprachkonzepte

Gruppierung: hier dargestellt am allgemeinen Format für SQL-Anfragen

select ... from ... where ...
group by ... having ... order by ...

Auswertungsreihenfolge und -prinzip:

1. from-Kausel: Produkt der angegebenen Tabellen

2. where-Kausel: Selektion der qualifizierenden Zeilen

3. group by-Kausel: Spezifikation von Gruppierungsattributen; Einteilung der Zwischenrelation in Gruppen mit gleichen Gruppierungsattributwerten

4. having-Kausel: Qualifikation von Gruppen

   Selektion der qualifizierenden Gruppen; in der Qualifikation müssen Ausdrücke pro Gruppe genau einen Wert liefern; insbesondere erlaubt:
   (A) Aggregationsfunktionen (s.u.) auf Nicht-Gruppierungsattribute oder
   (B) Gruppierungsattribute

5. select-Klausel: Projektion; falls group by verwendet, ähnliche Einschränkungen wie in having-Klausel

6. order by-Klausel: Sortierung

Aggregationsfunktionen

Aggregationsfunktionen bilden Multimengen von Werten ($\widehat{=}$ Mengen mit Duplikaten) auf einzelne Werte ab

Beispiele: count, sum, avg, max, min

Sichten

- Sichtdefinitionen $\widehat{=}$ externe DB-Schemata

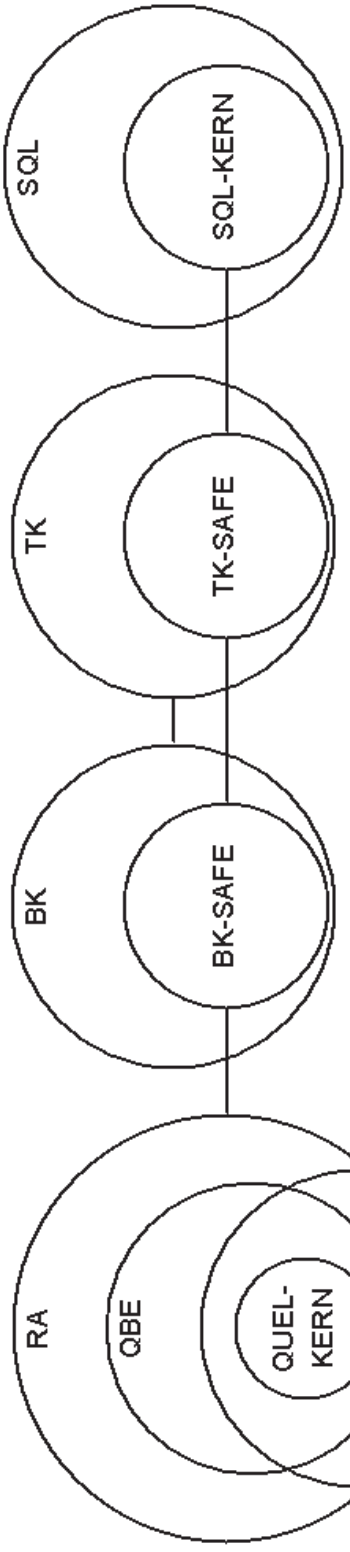
- Sichten (Virtuelle Relationen) können (mit Einschränkung) wie gespeicherte Relationen benutzt werden; ihre Definition wird bei jeder Benutzung neu ausgewertet

  Anfragen auf Sichten durch 'Anfragemodifikation' in Anfragen auf der DB übersetzt

- Vorteile:
  - logische Datenunabhängigkeit
  - Vereinfachung von Anfragen
  - Beschränkung von Zugriffen (Datenschutz)

- Problematisch: Änderung von Sichten

Sprachmächtigkeit im Überblick



Unterschiede anhand von Anfrageklassen/Beispielen

BK/TK - BK-SAFE/TK-SAFE : Nicht-sichere Anfragen;
SQL - SQL-KERN : Aggregationsfunktionen/Gruppierung;
DATALOG - RA : Rekursive Anfragen; RA - DATALOG: Differenz;
RA - QBE : Anfragen beginnend mit Allquantor; QBE - QUEL : Minimumsuche