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 $\omega : \mathcal{R}(\times ... \times \mathcal{R}) \rightarrow \mathcal{R}$

Grundlage für Anfragen der Form

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

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

Grundoperationen

Vorsicht: keine einheitlichen Grundoperationen in der Literatur

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

Relationenalgebra (Ergänzung von Details)

Grundoperationen

Gegeben: $R(A_1, ..., A_n), B(B_1, ..., B_m)$

- Vereinigung $R \cup S$

  Voraussetzung: R und S verträglich, d.h. $Rang(R) = Rang(S)$ und A_i und B_i haben gleiche Wertebereiche für alle i

  Ergebnisschema: $R'(A_1, ..., A_n)$

  Ergebnisrelation:
  $R' :=$ Mengenvereinigung von R und S

- Differenz $R - S$

  Vorausetzung: wie $R \cup S$

  Ergebnisschema: wie $R \cup S$

  Ergebnisrelation:
  $R' :=$ Mengendifferenz von R und S

- Kartesisches Produkt

  Vorausetzung: $\{A_1, ..., A_n\} \cap \{B_1, ..., B_m\} = \emptyset$

  Ergebnisschema: $R'(A_1, ..., A_n, B_1, ..., B_m)$

  Ergebnisrelation: $R' := \{r \circ s | r \in R \wedge s \in S\}$

  $r \circ s := (r_1, ..., r_n, s_1, ..., s_m)$ für $r = (r_1, ..., r_n) \in R$ und $s = (s_1, ..., s_m) \in S$

- Projektion $\pi_{\overline{A}}(R)$

  Vorausetzung: $\overline{A} = A_{i_1}, ..., A_{i_k}$ mit
  $1 \leq k \leq Rang(R) = n$ und
  $1 \leq i_1, ..., i_k \leq Rang(R) = k$

  Ergebnisschema: $R'(A_{i_1}, ..., A_{i_k})$

  Ergebnisrelation: $R' := \{\pi_{i_1,...,i_k}(r) | r \in R\}$

  $\pi_{i_1,...,i_k}(r) := (r_{i_1}, ..., r_{i_k})$ für $r = (r_1, ..., r_n)$

  Beispiel: mit $R(A_1, A_2, A_3)$ ergibt $\pi_{A_3,A_1}(R)$ das Ergebnisschema $R'(A_3, A_1)$; es gilt:
  $\pi_{A_3,A_1}(R) = \pi_{A_{i_1},A_{i_2}}(R)$ mit $k = 2, i_1 = 3, i_2 = 1$

- Selektion $\sigma_\phi(R)$

  Vorausetzung: ϕ ist atomare Formel $\Theta(u_1, ..., u_n)$; $u_1, ..., u_n$ sind Konstanten, Attributenamen oder Terme darüber; Θ ist boolesche Operation, wie Vergleiche $=, \neq, <, \leq, >, \geq$ oder andere boolesche Abfragen

  Ergebnisschema: $R'(A_1, ..., A_n)$

  Ergebnisrelation: $R' := \{r | r \in R \wedge \phi$ ergibt nach Ersetzung der Attributnamen A_i durch Komponenten r_i den Wahrheitswert TRUE$\}$

- Umbenennung $\delta_{C \leftarrow A_i}(R)$

  Voraussetzung: C Attributname mit
  $C \notin \{A_1, ..., A_n\}$

  Ergebnisschema: $R'(A_1, ..., A_{i-1}, C, A_{i+1}, ..., A_n)$

  Ergebnisrelation: $R' := R$

- Nachtrag zum Produkt:

  Falls $R \neq S$ und $\{A_1, ..., A_n\} \cap \{B_1, ..., B_m\} = \{C\}$ mit $A_i = B_j = C$ gilt, werden für $R \times S$ implizite Umbenennungen vorgenommen:
  $\delta_{R.C \leftarrow A_i}(R) \times \delta_{S.C \leftarrow B_j}(S)$
  Ergebnis $R'(A_1, ..., R.C, ..., A_n, B_1, ..., S.C, ..., B_m)$

  Falls R=S gilt, werden für $R \times S = R \times R$ implizite Umbenennungen vorgenommen, so daß für $R(A, B, ...)$ Als Ergebnis entsteht:
  $R'(A_1, B_1, ..., A_2, B_2, ...)$

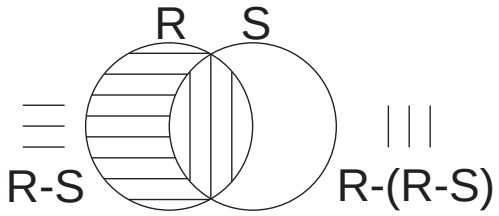
Abgeleitete Operationen

- Durchschnitt $R \cap S$

  Voraussetzung: wie $R \cup S$

  Definition: $R \cap S :=$ Mengendurchschnitt von R und S

  Ableitung: $R \cap S = R - (R - S)$



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

  Voraussetzung: Θ ist passender Vergleichsoperator

  Definition: $R *_{A_i \Theta B_j} S := \{r \circ s | r \in R \wedge s \in S \wedge r_i \Theta s_j\}$

  Ableitung: $R *_{A_i \Theta B_j} S = \sigma_{A_i \Theta B_j}(R \times S)$

- natürlicher Verbund $R * S$

  Voraussetzung: $C_1, ..., C_k$ seien alle gemeinsamen Attributnamen in R und S, also o.B.d.A. $R(A_1, ..., A_{n-k}, C_1, ..., C_k)$ und $S(B_1, ..., B_{m-k}, C_1, ..., C_k)$

  Definition: 'Verbinde R und S in natürlicher Weise über ihre gemeinsamen Attribute'

  Ableitung: $R * S = \pi_{A_1,...,A_{n-k},R.C_1,...,R.C_k,B_1,...,B_{m-k}}($
  $\sigma_{R.C_1=S.C_1 \wedge ... \wedge R.C_k=S.C_k}(R \times S))$

- Division $R : S$

  Analogie zur Division auf den natürlichen Zahlen

  $n : m$ ist die größte ganze Zahl mit $(n : m) * m \leq n$
  $R : S$ ist die größte Relation mit $(R : S) \times S \subseteq R$

  Voraussetzung: Attribute von S sind auch Attribute von R, also o.B.d.A. $R(A_1, ..., A_{n-m}, B_1, ..., B_m)$ und $S(B_1, ..., B_m)$

  Definition: $R : S := \{\pi_{A_1,...,A_{n-m}}(r) | r \in R \wedge$

  $\forall s \in S(\pi_{A_1,...,A_{n-m}}(r) \circ s \in R)\}$

  Ermittle alle $\pi_{A_1,...,A_{n-m}}$-Projektionen von Tupeln, die in den S-Komponenten alle in S aufgelisteten Wertkombinationen haben

  Ableitung: $R : S =$
  $\pi_{A_1,...,A_{n-m}}(R) - \pi_{A_1,...,A_{n-m}}((\pi_{A_1,...,A_{n-m}}(R) \times S) - R)$

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. $D_1 \times ... \times D_n \rightarrow D$), 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:

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

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

$[\sigma, \beta] \models R(r)$ 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:

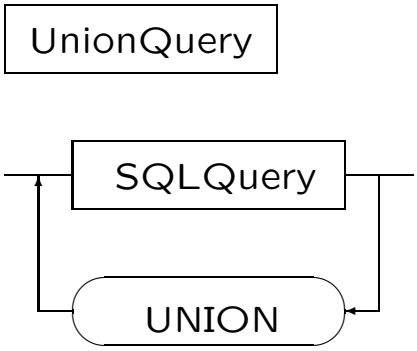
```
select   'Resultatsliste'
from     'Relationenliste'
where    'Prädikat'
```

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

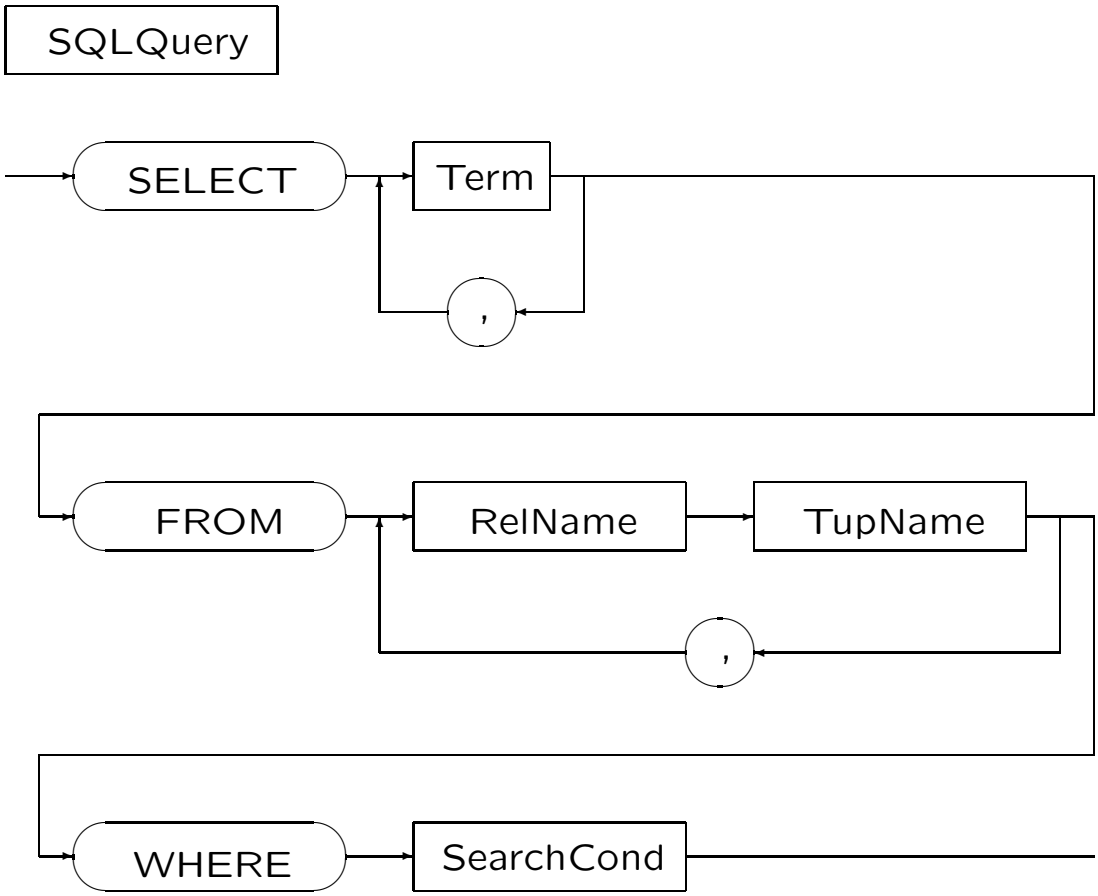
$\Rightarrow$

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

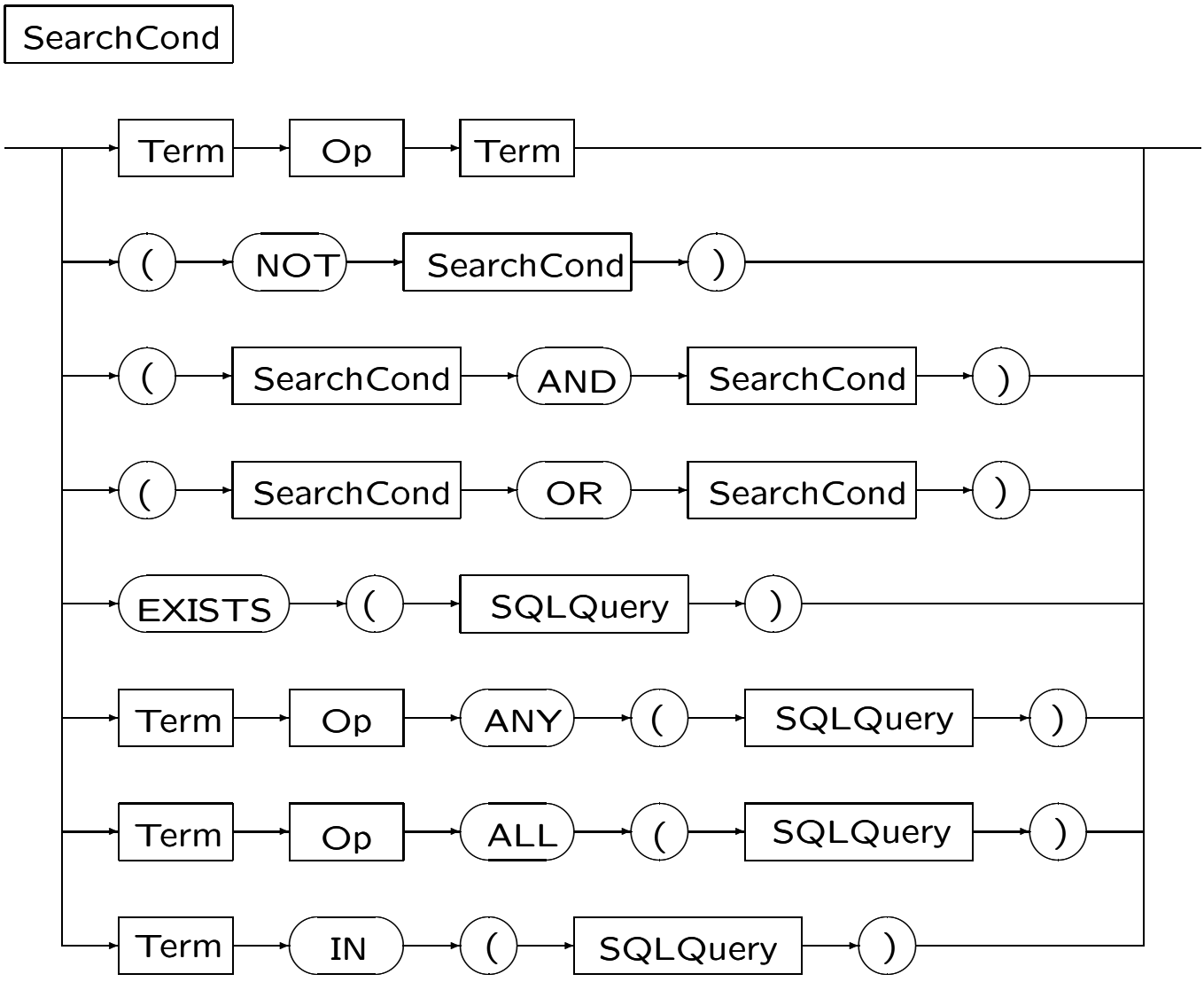
Syntax von UnionQuery



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

sql2tc$[\![$ SELECT τ_1, ..., τ_n
FROM R_1 s_1, ..., R_m s_m
WHERE φ $]\!]$:=
{ r : (Res_1, ..., Res_n) | ($\exists$ s_1:R_1, ..., s_m:R_m)
(r.Res_1 = τ_1 $\wedge$ ... $\wedge$ r.Res_n = τ_n $\wedge$ **sql2tc**$[\![$ φ $]\!]$) }

sql2tc$[\![$ (NOT φ) $]\!]$:= ($\neg$ **sql2tc**$[\![$ φ $]\!]$)

sql2tc$[\![$ (φ_1 AND φ_2) $]\!]$:=
(**sql2tc**$[\![$ φ_1 $]\!]$ $\wedge$ **sql2tc**$[\![$ φ_2 $]\!]$)

sql2tc$[\![$ (φ_1 OR φ_2) $]\!]$:=
(**sql2tc**$[\![$ φ_1 $]\!]$ $\vee$ **sql2tc**$[\![$ φ_2 $]\!]$)

sql2tc$[\![$ τ_1 ω τ_2 $]\!]$:= τ_1 ω τ_2

sql2tc$[\![$ τ ω ALL (SELECT s_i.A
FROM R_1 s_1, ..., R_i s_i, ..., R_m s_m
WHERE φ) $]\!]$:=
($\forall$ s_i:R_i)
(((($\exists$ s_1:R_1, ..., s_{i-1}:R_{i-1}, s_{i+1}:R_{i+1}, ..., s_m:R_m)
sql2tc$[\![$ φ $]\!]$) $\Rightarrow$ τ ω s_i.A)

sql2tc$[\![$ τ ω ANY (SELECT s_i.A
FROM R_1 s_1, ..., R_i s_i, ..., R_m s_m
WHERE φ) $]\!]$:=
($\exists$ s_i:R_i)
(((($\exists$ s_1:R_1, ..., s_{i-1}:R_{i-1}, s_{i+1}:R_{i+1}, ..., s_m:R_m)
sql2tc$[\![$ φ $]\!]$) $\wedge$ τ ω s_i.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 ⟧) ∧ $\tau = s_i$.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

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

(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:

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

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

$\Rightarrow$ 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

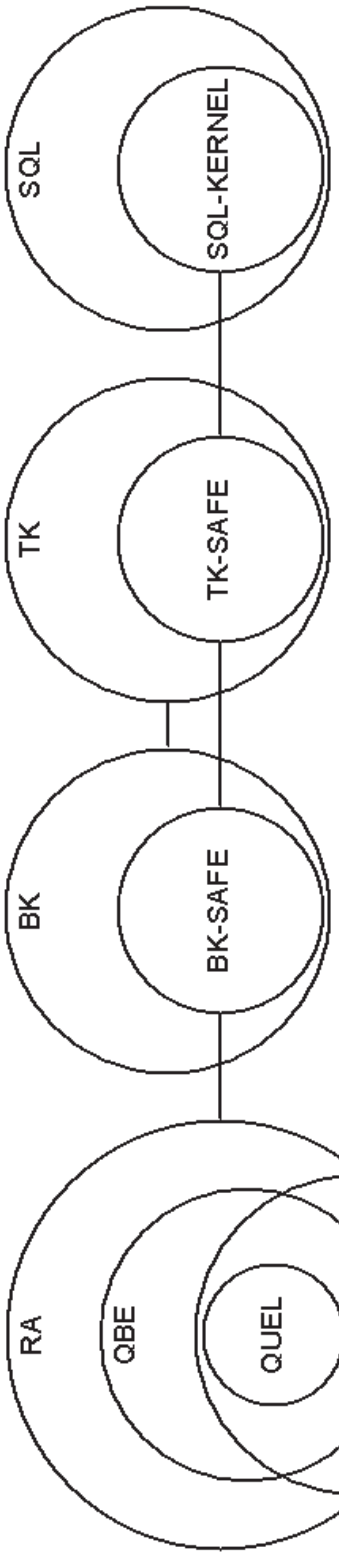
- Sichten (Virtuelle Relationen) können (mit Einschränkung) wie gespeicherte Relationen benutzt werden; ihre Definition wird bei jeder Benutzung neu ausgewertet

  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-KERNEL : Aggregationsfunktionen/Gruppierung;
DATALOG - RA : Rekursive Anfragen; RA - DATALOG: Differenz;
RA - QBE : Anfragen beginnend mit Allquantor; QBE - QUEL : Minimumsuche

SQL-Anfragen

Beispielschema: KAL-Schema

KUNDE(KName, KAdr, Kto)

AUF[TRAG](KName, Ware, Menge)

LIEF[ERANT](LName, LAdr, Ware, Preis)

- Welche Kunden (KName) haben ihr Konto überzogen?

```
select KName
from   KUNDE
where  Kto<0
```

- Welche Lieferanten (LName, LAdr) liefern Milch oder Mehl?

```
select distinct LName, LAdr
from   LIEF
where  Ware = 'Milch' or Ware = 'Mehl'

select L.LName, L.LAdr
from   LIEF L
where  L.Ware = 'Milch' or L.Ware = 'Mehl'

select LIEF.LName, LIEF.LAdr
from   LIEF
where  LIEF.Ware = 'Milch' or LIEF.Ware = 'Mehl'
```

- Welche Lieferanten (LName, Ware) aus Bremen liefern von Weiss in Auftrag gegebene Waren?

```
select LName, LIEF.Ware
from   LIEF, AUF
where  LAdr like '%Bremen%' and
       LIEF.Ware = AUF.Ware and
       KName = 'Weiss'
```

```
select LName, Ware
from   LIEF
where  LAdr like '%Bremen%' and
       Ware = any ( select Ware
                    from   AUF
                    where  KName = 'Weiss' )
```

```
select LName, Ware
from   LIEF
where  LAdr like '%Bremen%' and
       Ware in ( select Ware
                 from   AUF
                 where  KName = 'Weiss' )
```

- Welche Lieferanten (LName) liefern mindestens eine Ware, die auch Grau liefert?

```
select L.LName
from   LIEF L, LIEF LG
where  L.Ware = LG.Ware and LG.LName = 'Grau'
```

```
select L.LName
from   LIEF L
where  exists
       ( select LG.Ware
         from   LIEF LG
         where  L.Ware = LG.Ware and
                LG.LName = 'Grau' )
```

```
select L.LName
from   LIEF L
where  exists
       ( select *
         from   LIEF LG
         where  L.Ware = LG.Ware and
                LG.LName = 'Grau' )
```

- Welche Lieferanten (alle Attribute) liefern welche Waren genau so preiswert wie alle Lieferanten, die diese Ware liefern?

```
select *
from   LIEF L
where  Preis <= all ( select Preis
                      from   LIEF
                      where  Ware = L.Ware )
```

- Stelle eine Liste von Kunden (KName, KAdr) und möglichen Lieferanten (LName, LAdr) ihrer Aufträge zusammen.

```
select KName, KAdr, LName, LAdr
from   KUNDE, LIEF
where  exists ( select *
                from   AUF
                where  Ware=LIEF.Ware and
                       KName=KUNDE.KName )
```

```
select KName, KAdr, LName, LAdr
from   KUNDE, LIEF
where  Ware = any ( select Ware
                    from   AUF
                    where  KName=KUNDE.KName )
```

```
select KName, KAdr, LName, LAdr
from   KUNDE, LIEF
where  KName = any ( select KName
                     from   AUF
                     where  Ware=LIEF.Ware )
```

- Welche Lieferanten (LName) liefern mindestens alle Waren, die Grau liefert?

```
select LName
from   LIEF L
where  not exists
       ( select Ware
         from   LIEF
         where  LName='Grau' and
                not Ware in
                  ( select Ware
                    from   LIEF
                    where  LName=L.LName ) )
```

```
select RES.LName
from   LIEF RES
where  not exists
       ( select GR.Ware
         from   LIEF GR
         where  GR.LName='Grau' and
                not GR.Ware in
                  ( select RESWA.Ware
                    from   LIEF RESWA
                    where  RESWA.LName=RES.LName ) )
```

- Welche Waren sind in Auftrag gegeben oder lieferbar?

```
( select Ware from AUF )
union
( select Ware from LIEF )
```

Aggregationsfunktionen und Gruppierung

- Motivierendes Beispiel

  STUD(Matnr,SName,Fach,Sem)

  Anzahl MI-Studis je Semesterzahl, wobei Anzahl > 10

```
select   Sem, count(Matnr)
from     STUD
where    Fach='MI'
group by Sem
having   count(Matnr)>10
```

  mögliches Ergebnis

Sem	count(Matnr)
2	28
4	15

- Bemerkung zu `select distinct ...` versus `select ...`

  Im Kontext von Aggregationsfunktionen sind Duplikate wichtig; in den SQL-Kern-Anfragen war die Angabe `distinct` der Einfachheit halber weggelassen worden

KUNDE(KName, KAdr, Kto)

AUF[TRAG](KName, Ware, Menge)

LIEF[ERANT](LName, LAdr, Ware, Preis)

- Anzahl von Kunden

```
select count(*) from KUNDE

select count(KName) from KUNDE
```

- Summe der Auftragsmengen je Ware

```
select   Ware, sum(Menge)
from     AUF
group by Ware
```

AUF	KName	Ware	Menge
	'Schwarz'	'Mehl'	100
	'Schwarz'	'Salz'	300
	'Weiss'	'Mehl'	200
	'Weiss'	'Salz'	300

AUF by Ware	KName	Ware	Menge
	'Schwarz'	'Mehl'	100
	'Weiss'	'Mehl'	200
	'Schwarz'	'Salz'	300
	'Weiss'	'Salz'	300

Ware,sum(Menge)	Ware	sum(Menge)
	'Mehl'	300
	'Salz'	600

- Waren die von mehr als einem Lieferanten angeboten werden

```
select   Ware
from     LIEF
group by Ware
having   count(LName)>1
```

auch im Kern:

```
select distinct Ware
from   LIEF L1
where  exists (select *
               from   LIEF L2
               where  L1.Ware=L2.Ware and
                      L1.LName<>L2.LName)
```

ebenfalls einfache Mimimum- und Maximum-Suche im Kern formulierbar

- Anzahl der verschiedenen Lieferanten die Mehl oder Milch anbieten

```
select count(distinct LName)
from   LIEF
where  Ware='Mehl' or Ware='Milch'
```

i.A. unterschiedlich von

```
select count(LName)
from   LIEF
where  Ware='Mehl' or Ware='Milch'
```

- Durchschnitt aller Gesamtauftragsmengen von Waren

  in SQL-89 nicht mit einer Anweisung formulierbar, da Schachtelung von Aggregationen ausgeschlossen; zu SQL-92 siehe unten

- Aufträge mit maximaler Menge (über eine Unteranfrage)

```
select KName, Ware
from   AUF A
where  not exists(select *
                  from   AUF
                  where  Menge>A.Menge)
```

- Aufträge mit maximaler Menge (mit Aggregationsfunktion)

```
select KName, Ware
from   AUF
where  Menge=(select max(Menge) from AUF)
```

- Waren zusammen mit minimalem, durchschnittlichem und maximalem Preis alphabetisch nach Waren sortiert

```
select   Ware, min(Preis), avg(Preis), max(Preis)
from     LIEF
group by Ware
order by Ware
```

- Waren deren maximaler Preis um höchstens 5% höher ist als der minimale Preis, i.e. Waren mit geringer Preisspanne

```
select   Ware
from     LIEF
group by Ware
having   max(Preis)<=min(Preis)*1.05
```

SQL-89 VERSUS SQL-92

- In SQL-92 jeder SFW-Block in from-Klausel einsetzbar

```
ESD(Employee,Salary,Department)

select   Department, avg(salary)
from     ESD
group by Department

select D.Department, A.SalAverage
from   (select distinct Department
        from   ESD                  ) as D,
       (select avg(Salary) as SalAverage
        from   ESD
        where  ESD.Department=D.Department ) as A
```

- Schachtelung von Aggregationen

```
select 'AggFunc'(InterResult)
from   (select 'AggFunc'('Attribute') as InterResult
        from   'Relation')

Maximum der Durchschnitte

select max(Average)
from   (select   avg(Salary) as Average
        from     ESD
        group by Department)
```

Erweiterter Tupelkalkül

- Erinnerung: Projektion in der RA und in SQL

R	A	B
	a1	b1
	a1	b2
	a2	b1

$\pi_A(R) =$

R'	A
	a1
	a2

Äquivalent in SQL ist

`select distinct A from R` $\rightarrow \{a1, a2\}$

Folgende SQL-Anfrage ist streng genommen in der RA und den Kalkülen nicht formulierbar

`select A from R` $\rightarrow \{a1, a1, a2\}$

- SQL-Aggregationsfunktionen sind nicht Teil der RA bzw. der Kalküle

- Es folgt: Erweiterung der RA bzw. der Kalküle notwendig

  Hier diskutiert Kalkülerweiterung

- Beispiel: STUDENT(Matnr,SName,Fach,Sem)

  $\{\ t : (Sem)\ \mid\ \exists\ s : STUDENT\ (t.Sem = s.Sem)\ \}$

  $\dashv[\ \ Sem(s)\ \mid\ s : STUDENT\ \ ]\vdash$

- $\{\ Term\ \mid\ Formel\ \}$ ergibt Menge

  $\dashv[\ \ Term\ \mid\ Deklaration\ \wedge\ Formel\ \ ]\vdash$ ergibt Multimenge (Bag)

- Multimengen lassen sich als Mengen zusammen mit einer Zählfunktion auffassen: $\{a1, a1, a2\} \equiv \{(a1 \mapsto 2), (a2 \mapsto 1)\}$

  Allgemein $Bag \equiv (Set, occurrences : Set \rightarrow NAT)$

- Deklarationen binden Variablen an Mengen

- Mächtigkeit der entstehenden Multimenge richtet sich nach den möglichen Zuweisungen an die deklarierten Variablen

- Standardfunktionen CNT (count), SUM, MAX, MIN, AVG, BTS (BagToSet)

Beispielanfragen im erweiterten Tupelkalkül

- Anzahl von Kunden

  $CNT \dashv\{\ k | k : KUNDE\ \}\vdash$

  $CNT \dashv\{\ KName(k) | k : KUNDE\ \}\vdash$

- Summe der Auftragsmengen je Ware

  $$\dashv\{\ Ware(a),\ SUM \dashv\{\ Menge(a') | a' : AUF \wedge Ware(a') = Ware(a)\ \}\vdash \mid a : AUF\ \}\vdash$$

  $$BTS \dashv\{\ Ware(a),\ SUM \dashv\{\ Menge(a') | a' : AUF \wedge Ware(a') = Ware(a)\ \}\vdash \mid a : AUF\ \}\vdash$$

  $$\dashv\{\ w,\ SUM \dashv\{\ Menge(a) | a : AUF \wedge Ware(a) = w\ \}\vdash \mid w : BTS \dashv\{\ Ware(a) | a : AUF\ \}\vdash\ \}\vdash$$

  im Ergebnis der 1. Formulierung taucht dieselbe Ware so oft auf wie sie Lieferanten hat; 2. und 3. Formulierung äquivalent zur entsprechenden SQL-Anfrage

- Waren die von mehr als einem Lieferanten angeboten werden

  $$BTS \dashv\{\ Ware(l) \mid l : LIEF \wedge CNT \dashv\{\ LName(l') | l' : LIEF \wedge Ware(l') = Ware(l)\ \}\vdash\ > \ 1\ \}\vdash$$

- Anzahl der verschiedenen Lieferanten die Mehl oder Milch anbieten

$$CNT(BTS \text{-}[\ LName(l)| \\ l : LIEF \wedge \\ (Ware(l) = `Mehl' \vee Ware(l) = `Milch')\ ]\text{-}\ )$$

- Durchschnitt aller Gesamtauftragsmengen von Waren

$$AVG(\ \text{-}[\ SUM\ \text{-}[\ Menge(a')|a' : AUF \wedge Ware(a') = w\ ]\text{-}\ | \\ w : BTS\ \text{-}[\ Ware(a)|a : AUF\ ]\text{-}\ \ ]\text{-}\ )$$

- Aufträge mit maximaler Menge (über eine Unteranfrage)

$$\text{-}[\ KName(a), Ware(a)| \\ a : AUF \wedge \\ CNT\ \text{-}[\ \ Ware(a')| \\ a' : AUF \wedge \\ Menge(a') > Menge(a)\ ]\text{-}\ = 0\ ]\text{-}$$

- Aufträge mit maximaler Menge (mit Aggregationsfunktion)

$$\text{-}[\ KName(a), Ware(a)| \\ a : AUF \wedge \\ Menge(a) = MAX\ \text{-}[\ Menge(a)|a : AUF\ ]\text{-}\ \ ]\text{-}$$

- Waren zusammen mit minimalem, durchschnittlichem und maximalem Preis alphabetisch nach Waren sortiert

$$\begin{aligned}&\dashv[\ Ware(l),\\ &\quad MIN\ \dashv[\ Preis(l')|l' : LIEF \wedge Ware(l') = Ware(l)\ ]\dashv\ ,\\ &\quad AVG\ \dashv[\ Preis(l')|l' : LIEF \wedge Ware(l') = Ware(l)\ ]\dashv\ ,\\ &\quad MAX\ \dashv[\ Preis(l')|l' : LIEF \wedge Ware(l') = Ware(l)\ ]\dashv\ |\\ &\quad l : LIEF\ ]\dashv_{\ sort(1)}\end{aligned}$$

- Waren deren maximaler Preis um höchstens 5% höher ist als der minimale Preis, i.e. Waren mit geringer Preisspanne

$$\begin{aligned}BTS\ \dashv[\ &Ware(l)|\\ &l : LIEF \wedge\\ &(MAX\ \dashv[\ Preis(l')|l' : LIEF \wedge\\ &\qquad Ware(l') = Ware(l)\ ]\dashv\ ) <=\\ &(MIN\ \dashv[\ Preis(l')|l' : LIEF \wedge\\ &\qquad Ware(l') = Ware(l)\ ]\dashv\ ) * 1.05\ ]\dashv\end{aligned}$$

Vorteile des erweiterten Tupelkalküls

- Gruppierung nicht als separates Konzept notwendig, sondern durch geschachtelte Anfrageterme darstellbar

- Geschachtelte Aggregationsfunktionen darstellbar

- Durch Einführung von Deklarationen als separates Konzept nahe an SQL

QUEL-Anfragen

- Welche Lieferanten liefern von Weiss in Auftrag gegebene Waren?

```
range of a is AUF, l is LIEF
retrieve (Name=l.LName,l.Ware)
  where a.KName='Weiss' and l.Ware=a.Ware
```

- Welche Lieferanten liefern Milch oder Mehl?

```
range of l is LIEF
retrieve (l.LName,l.LAdr)
  where l.Ware='Milch' or l.Ware='Mehl'
```

  Mit Duplikatbeseitigung: `retrieve unique`

```
range of l is LIEF
retrieve unique (l.LName,l.LAdr)
  where l.Ware='Milch' or l.Ware='Mehl'
```

  Im Folgenden `unique` der Einfachheit halber weggelassen

- Welche Lieferanten liefern mindestens eine Ware die auch Grau liefert?

```
range of l1, l2 is LIEF
retrieve (l1.LName)
  where l1.Ware=l2.Ware and l2.LName='Grau'
```

- Zu welchen Kunden gibt es keinen Auftrag?

  $\pi_{KName}(KUNDE) - \pi_{KName}(AUF)$

```
range of k is KUNDE
retrieve into OHNE-AUF (Name=k.KName)
range of a is AUF, oa is OHNE-AUF
delete oa where oa.Name=a.KName
```

- (R $\times$ S)-T

  R(A1,...,An), S(B1,...,Bm),
  T(C1,...,Cn,Cn+1,...,Cn+m)

```
range of r is R, s is S
retrieve into U
  (C1=r.A1,...,Cn=r.An,Cn+1=s.B1,...,Cn+m=s.Bm)
range of u is U
range of t is T
delete u where u.C1=t.C1 and ... and u.Cn+m=t.Cn+m
```

- Welche Lieferanten liefern welche Waren genau so preiswert wie alle Lieferanten, die diese Ware liefern?

```
range of l1, l2 is LIEF
retrieve into TEURER-LIEF (l1.LName,l1.Ware)
  where l1.Ware=l2.Ware and l1.Preis>l2.Preis   (A)
retrieve into BILLIGER-LIEF
  (l1.LName,l1.LAdr,l1.Ware,l1.Preis)           (B)
range of bl is BILLIGER-LIEF, tl is TEURER-LIEF
delete bl
  where bl.LName=tl.LName and bl.Ware=tl.Ware   (C)
```

LIEF	LName	LAdr	Ware	Preis
	Date	...	DBS	40
	Ullman	...	DBS	60
	Elmasri	...	DBS	35

TEURER-LIEF	LName	Ware
	Date	DBS
	Ullman	DBS

nach (A)

BILLIGER-LIEF	LName	LAdr	Ware	Preis
	Date	...	DBS	40
	Ullman	...	DBS	60
	Elmasri	...	DBS	35

nach (B)

BILLIGER-LIEF	LName	LAdr	Ware	Preis
	Elmasri	...	DBS	35

nach (C)

- Beispiel zu append: Füge alle Mehl-Lieferanten als Kunden ein

```
range of l is LIEF
append to KUNDE (KName=l.LName, KAdr=l.LAdr, Kto=0)
  where l.Ware='Mehl'
```

- Beispiel zu replace: Halbiere die Auftragsmengen von Weiss

```
range of a is AUF
replace a (a.Menge = a.Menge*0.5)
  where a.KName='Weiss'
```

Überblick zu den QUEL-Statements

- range of s is S
- retrieve into S where p(s) $\widehat{=}$ $S := \{s|p(s)\}$
- append to S where p(s) $\widehat{=}$ $S := S \cup \{s|p(s)\}$
- delete s where p(s) $\widehat{=}$ $S := S - \{s|p(s)\}$

Darstellung der Relationen-Operationen in QUEL

Gegeben R(A1, ..., An) und S(B1, ..., Bm)

- $R \cup S$

```
range of r is R
retrieve into RuS (r.A1,...,r.An)
range of s is S
append to RuS (s.B1,...,s.Bm)
```

- $R - S$

```
range of r is R
retrieve into R-S (r.A1,...,r.An)
range of r-s is R-S, s is S
delete r-s where r-s.A1=s.B1 and ... and r-s.An=s.Bm
```

- $R \times S$

```
range of r is R, s is S
retrieve into RxS (r.A1,...,r.An,s.B1,...,s.Bm)
```

- $\pi_{\overline{A}}(R)$ mit $\overline{A} = A_{i1}...A_{ik}$

```
range of r is R
retrieve into PI(R) (r.A[i1],...,r.A[ik])
```

- $\sigma_{\varphi}(R)$

```
range of r is R
retrieve into SIGMA(R) (r.A1,...,r.An) where phi
```

Motivation zu QBE

- Anfragen an ein DBMS werden oft nicht nur über eine textuelle Anfragesprache gestellt, sondern auch über eine graphische Benutzungsschnittstelle bzw. graphische Anfragesprache ermöglicht

- hier StarBase (StarOffice) als Beispiel verwendet; ähnliche Beobachtungen gelten auch für andere DBMS

- Anfragen im WWW werden oft über Formulare gestellt in denen bestimmte Felder ausgefüllt werden; auch solche Formular-Anfragen kann man als nicht-textuelle Anfragen auffassen

- "Urahn" solcher Anfragesprachen ist QBE

- QBE war die erste nicht-textuelle Anfragesprache (1975)

	Recipe	Country	Ingredient	NeedAmount	StoreAmount
▶	Pizza	Italy	Egg	1	2
	Tortilla	Spain	Egg	4	2
	BoiledEgg	Germany	Egg	1	2
	Pizza	Italy	Cheese	100	500
	Lasagne	Italy	Cheese	200	500

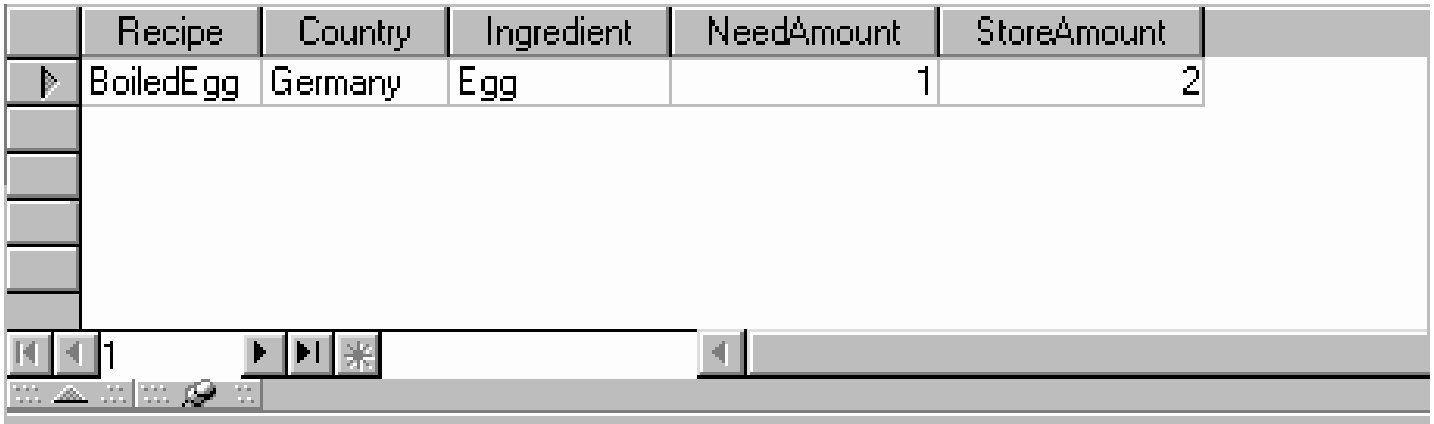
1 0/5

Feld	Recipe	Country	Ingredient	Amount	Amount
Alias				NeedAmount	StoreAmount
Tabelle	Origin	Origin	Need	Need	Store
Sortierung					
Sichtbar	☑	☑	☑	☑	☑
Kriterium			= 'Egg'		
oder			= 'Cheese'		

```
Origin(Recipe,Country)
Need(Recipe,Ingredient,Amount)
Store(Ingredient,Amount)
```

```
+-----------+------+------+
|           | Cou  | Ing  |
+-----------+------+------+
| Kriterium |      | ='E' |
+-----------+------+------+
| oder      |      | ='C' |
+-----------+------+------+
```

```
select Origin."Recipe", Origin."Country",
       Need."Ingredient", Need."Amount" as "NeedAmount",
       Store."Amount" as "StoreAmount"
from   "Store" Store, "Need" Need, "Origin" Origin
where  ( Origin."Recipe" = Need."Recipe" and
         Store."Ingredient" = Need."Ingredient" ) and
       ( Need."Ingredient" = 'Egg' or
         Need."Ingredient" = 'Cheese' )
```



Recipe	Country	Ingredient	NeedAmount	StoreAmount
BoiledEgg	Germany	Egg	1	2

Feld	Recipe	Country	Ingredient	Amount	Amount
Alias				NeedAmount	StoreAmount
Tabelle	Origin	Origin	Need	Need	Store
Sortierung					
Sichtbar	☑	☑	☑	☑	☑
Kriterium		= 'Germany'	= 'Egg'		
oder			= 'Cheese'		

```
Origin(Recipe,Country)
Need(Recipe,Ingredient,Amount)
Store(Ingredient,Amount)
```

```
+-----------+------+------+
|           | Cou  | Ing  |
+-----------+------+------+
| Kriterium | ='G' | ='E' |
+-----------+------+------+
| oder      |      | ='C' |
+-----------+------+------+
```

```
Cou='G' and ( Ing='E' or Ing='C' )
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
                      ( Cou='G' and Ing='E' ) or Ing='C'
```

```
select Origin."Recipe", Origin."Country",
       Need."Ingredient", Need."Amount" as "NeedAmount",
       Store."Amount" as "StoreAmount"
from   "Store" Store, "Need" Need, "Origin" Origin
where  ( Need."Recipe" = Origin."Recipe" and
         Need."Ingredient" = Store."Ingredient" ) and
       ( Origin."Country" = 'Germany' and
         ( Need."Ingredient" = 'Egg' or
           Need."Ingredient" = 'Cheese' ) )
```

	Recipe	Country	Ingredient	NeedAmount	StoreAmount
▶	BoiledEgg	Germany	Egg	1	2
	Pizza	Italy	Cheese	100	500
	Lasagne	Italy	Cheese	200	500

1

Feld	Recipe	Country	Ingredient	Amount	Amount
Alias				NeedAmount	StoreAmount
Tabelle	Origin	Origin	Need	Need	Store
Sortierung					
Sichtbar	☑	☑	☑	☑	☑
Kriterium		= 'Germany'	= 'Egg'		
oder			= 'Cheese'		

```
Origin(Recipe,Country)
Need(Recipe,Ingredient,Amount)
Store(Ingredient,Amount)
```

	Cou	Ing
Kriterium	='G'	='E'
oder		='C'

```
Cou='G' and ( Ing='E' or Ing='C' )

                    ( Cou='G' and Ing='E' ) or Ing='C'
                    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

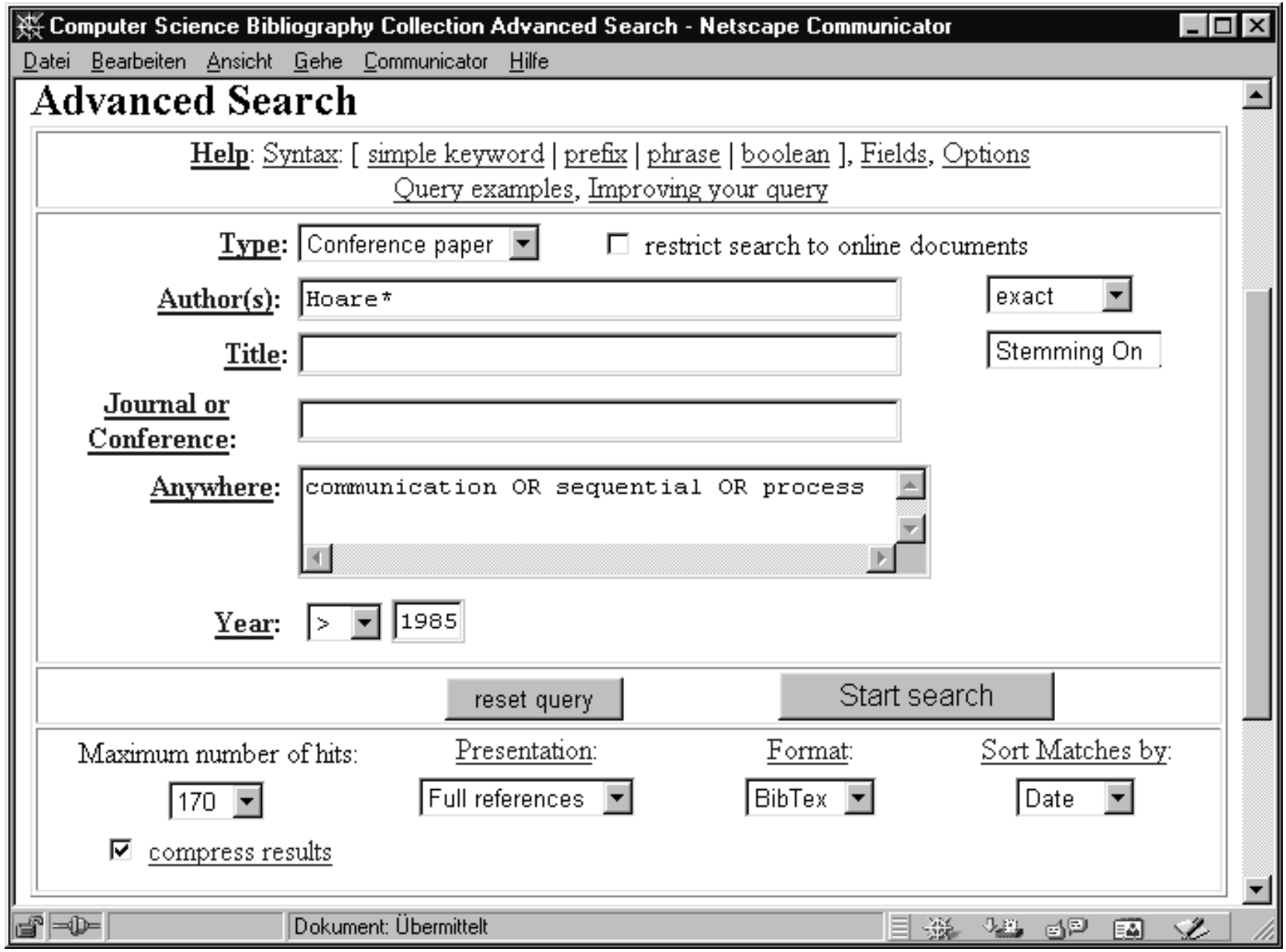
```
select Origin."Recipe", Origin."Country",
       Need."Ingredient", Need."Amount" as "NeedAmount",
       Store."Amount" as "StoreAmount"
from   "Store" Store, "Need" Need, "Origin" Origin
where  ( Need."Recipe" = Origin."Recipe" and
         Need."Ingredient" = Store."Ingredient" ) and
       ( ( Origin."Country" = 'Germany' and
         Need."Ingredient" = 'Egg' ) or
         Need."Ingredient" = 'Cheese' )
```

Problem: Formale Interpretation der Tabelle

Konjunktion von disjunktiven Einzelbedingungen
'`X and (Y or Z)`' = 'Spalten mit `and` verbinden'

VERSUS

Disjunktion von konjunktiven Einzelbedingungen
'`(X and Y) or Z`' = 'Zeilen mit `or` verbinden'


Computer Science Bibliography Collection Advanced Search - Netscape Communicator
Datei Bearbeiten Ansicht Gehe Communicator Hilfe
Advanced Search
Help: Syntax: [simple keyword | prefix | phrase | boolean], Fields, Options
Query examples, Improving your query
Type: Conference paper
restrict search to online documents
Author(s): Hoare*
exact
Title:
Stemming On
Journal or Conference:
Anywhere: communication OR sequential OR process
Year: > 1985
reset query
Start search
Maximum number of hits:
170
Presentation:
Full references
Format:
BibTex
Sort Matches by:
Date
compress results
Dokument: Übermittelt

QBE-Anfragen

- Welche Lieferanten liefern von Weiss in Auftrag gegebene Waren für unter 100,-?

LIEF	LName	LAdr	Ware	Preis
	P.		P._Mehl	<100

AUF	KName	Ware	Menge
	Weiss	_Mehl	

P. = 'Print'

oder mit kürzerem Variablennamen

LIEF	LName	LAdr	Ware	Preis
	P.		P._m	<100

AUF	KName	Ware	Menge
	Weiss	_m	

- natural join von $R(A, B), S(B, C)$

R	A	B
P.		_b

S	B	C
	_b	P.

- Welche Lieferanten liefern Milch oder Mehl?

LIEF	LName	LAdr	Ware
	P.	P.	_mehl

CONDITIONS
_mehl=Milch or _mehl=Mehl

- Welche Lieferanten liefern mindestens eine Ware die auch Grau liefert?

LIEF	LName	Ware
	P.	_mehl
	Grau	_mehl

- Konto und Aufträge aller Kunden

KUNDE	KName	Kto
	_weiss	_42

AUF	KName	Ware	Menge
	_weiss	_mehl	_21

temporäre Ausgabetabelle

KUNDENAUF	Name	Menge	Ware	Konto
P.	_weiss	_21	_mehl	_42

- Aufträge mit maximaler Menge

AUF	KName	Ware	Menge
P.			_42
¬			$>$_42

- Welches sind die Kunden ohne Auftrag?
  $\pi_{KName}(KUNDE) - \pi_{KName}(AUF)$
  Wie P. auch D. (= 'Delete') und I. (='Insert')
  - 1. Lösung
    1.Anweisung

AUF	KName
	_weiss

A-KUNDE	KName
I.	_weiss

    A-KUNDE temporäre Zwischenrelation
    2. Anweisung

KUNDE	KName
P.	_weiss

A-KUNDE	KName
¬	_weiss

  - 2. (elegantere) Lösung

KUNDE	KName
P.	_weiss

AUF	KName
¬	_weiss

  - 3. Lösung
    1. Anweisung

KUNDE	KName
	_weiss

OA-KUNDE	KName
I.	_weiss

    2. Anweisung

OA-KUNDE	KName
D.	_weiss

AUF	KName
	_weiss

    3. Anweisung

OA-KUNDE	KName
P.	

Darstellung der Relationen-Operationen in QBE

Gegeben R(A1, ..., An) und S(B1, ..., Bm)

- $R \cup S$
  - 1. Lösung

    1. Anweisung

R	A1	...	An
	x1	...	xn

RuS	A1	...	An
I.	x1	...	xn

    2. Anweisung

S	B1	...	Bn
	x1	...	xn

RuS	A1	...	An
I.	x1	...	xn

  - 2. Lösung

R	A1	...	An
	x1	...	xn

S	B1	...	Bn
	y1	...	yn

RuS	A1	...	An
I.	z1	...	zn

CONDITIONS
(z1=x1 and ... and zn=xn) or (z1=y1 and ... and zn=yn)

- $R - S$

  1. Anweisung

R	A1	...	An
	x1	...	xn

R-S	A1	...	An
I.	x1	...	xn

  2. Anweisung

S	B1	...	Bn
	x1	...	xn

R-S	A1	...	An
D.	x1	...	xn

- $R \times S$

R	A1	...	An
	x1	...	xn

S	B1	...	Bm
	y1	...	ym

RxS	A1	...	An	B1	...	Bm
I.	x1	...	xn	y1	...	ym

- $\pi_{\overline{A}}(R)$ mit $\overline{A} = A_{i1}...A_{ik}$

R	A1	...	An
	x1	...	xn

PI(R)	A[i1]	...	A[in]
I.	x[i1]	...	x[in]

- $\sigma_{\varphi}(R)$

R	A1	...	An
	x1	...	xn

CONDITIONS
φ

SIGMA(R)	A1	...	An
I.	x1	...	xn

Beispiel: QBE nicht relational vollständig

$R(A), S(B)$

$$\{a1 \mid \quad R(a1) \wedge \\ \forall b(S(b) \Rightarrow \exists a2[R(a2) \wedge b < a1 \wedge b < a2 \wedge a1 < a2])\}$$

Alle a1 in R, so dass (1) alle b in S kleiner als a1 sind und (2) ein a2 in R existiert mit a1 kleiner als a2 und alle b in S kleiner als a2 sind

Entscheidend: Formel fängt mit Allquantor, nicht mit Existenzquantor (wie in QBE-Semantik) an

$$\{a1 \mid \quad R(a1) \wedge \\ \neg\exists b\neg(S(b) \Rightarrow \exists a2[R(a2) \wedge b < a1 \wedge b < a2 \wedge a1 < a2])\}$$

Versuch 1 einer Formulierung in QBE

R	A
	P. _a1
¬	_a2

S	B
¬	_b

CONDITIONS
_b<_a1 and _b<_a2 and _a1<_a2

Variablen in CONDITIONS erfüllen nicht geforderte Bedingungen

Versuch 2 einer Formulierung in QBE

R	A
	P. _a1
¬	'_a2'>_a1

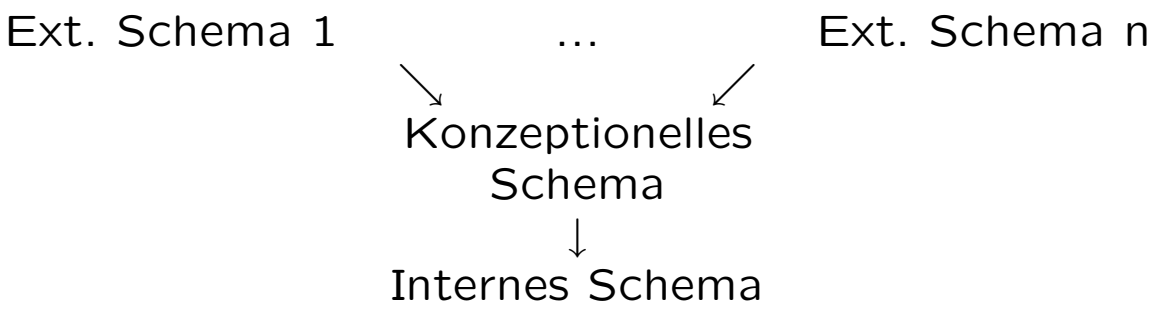
S	B
¬	'_b'<_a1

CONDITIONS
'_b'<'_a2'

Variablen '_a2' und '_b' nicht explizit ansprechbar

Sichten

- Erinnerung: Drei-Ebenen-Architektur



- Beispiel

  KUNDE(KName, KAdr, Kto)

  AUF[TRAG](KName, Ware, Menge)

  LIEF[ERANT](LName, LAdr, Ware, Preis)

- Sicht: Aufträge des Kunden Weiss

```
create view WEISS_AUF (Anzahl, Bezeichnung)
as select Menge, Ware
   from   AUF
   where  KName='Weiss'
```

  Korrespondenz zwischen Sicht-Attributen und Anfrage-Attributen durch Reihenfolge

  Beispielanfrage

```
select sum(Anzahl)
from   WEISS_AUF
where  Bezeichnung='Mehl' or Bezeichnung like '%mehl'
```

- wird durch Anfragemodifikation übersetzt nach

```
select sum(Menge)
from   AUF
where  KName='Weiss' and (Ware='Mehl' or
                          Ware like '%mehl')
```

- Anfragemodifikation allgemein:
  aus Anfrage $select\ \tau\ from\ \beta\ where\ \phi$
  an Sicht $create\ view\ \beta\ as\ select\ \tau'\ from\ \rho\ where\ \phi'$
  wird $select\ \overline{\tau}\ from\ \rho\ where\ \phi'\ \wedge \overline{\phi}$
  wobei $\overline{\tau}$ und $\overline{\phi}$ aus τ und ϕ
  durch Umbenennung der Sichtattribute aus β
  in die Originalattribute aus ρ hervorgehen

- Beispiel

  STUD(Matnr, SName, Fach, Sem)
  AUS(Doknr, Matnr, Datum)
  BUCH(Doknr, Titel, Verlag, Ort, Jahr)
  AUT(Doknr, AName)
  DESK(Doknr, Schlagwort)

- Sicht: Universalrelation = natural full outer join über allen Relationenschemata eines gegebenen relationalen DB-Schemas

```
create view BIB_UNIV(Matnr, SName, Fach, Sem, Doknr,
                     Datum, Titel, Verlag, Ort, Jahr,
                     AName, Schlagwort)
as select STUD.Matnr, SName, Fach, Sem, AUS.Doknr,
          Datum, Titel, Verlag, Ort, Jahr, AName,
          Schlagwort
   from   ( ( ( STUD natural full outer join AUS )
                     natural full outer join BUCH )
                     natural full outer join AUT )
                     natural full outer join DESK
```

Anfrage 'Zimmermann'-Autoren an Sicht

```
select distinct AName
from   BIB_UNIV
where  SName='Zimmermann'
```

Anfrage 'Zimmermann'-Autoren an Original-Schemata

```
select distinct AName
from   AUT, AUS, STUD
where  AUT.Doknr=AUS.Doknr and
       AUS.Matnr=STUD.Matnr and
       SName='Zimmermann'
```

- Sicht: Autorenpaare mit einem gemeinsamen Buchtitel

```
create view AUTPAAR(AName1, AName2, Titel)
as select A1.AName, A2.AName, Titel
   from   AUT A1, AUT A2, BUCH
   where  A1.Doknr=A2.Doknr and
          A1.Doknr=BUCH.Doknr and
          A1.AName<A2.AName
```

Anfrage

Koautoren von Date (ohne Duplikate) an Sicht

```
( select distinct AName1
  from   AUTPAAR
  where  AName2='Date' ) -- alphabetisch vor Date
union
( select distinct AName2
  from   AUTPAAR
  where  AName1='Date' ) -- alphabetisch nach Date
```

Koautoren von Date (ohne Duplikate) an Original-Schemata

```
( select A1.AName
  from   AUT A1, AUT A2, BUCH
  where  A1.Doknr=A2.Doknr and A1.Doknr=BUCH.Doknr and
         A1.AName<A2.AName and A2.AName='Date' )
union
( select A2.AName
  from   AUT A1, AUT A2, BUCH
  where  A1.Doknr=A2.Doknr and A1.Doknr=BUCH.Doknr and
         A1.AName<A2.AName and A1.AName='Date' )
```

- Sichten Abstraktionsmittel um Wiederholungen in der Formulierung von Anfragen zu vermeiden

- Sichten werden eingerichtet für bestimmte Anwendergruppen einer DB

- weitere Beispiele für Sichten

  Sicht Sekretariat

```
create view SEKRETARIAT(Matnr, SName, Fach,
                        Sem, NochDoks?)
as ( select distinct Matnr, SName, Fach,
            Sem, true
     from   BIB_UNIV BU
     where  exists (select *
                    from   AUS
                    where  Matnr=BU.Matnr) )
   union
   ( select distinct Matnr, SName, Fach,
            Sem, false
     from   BIB_UNIV BU
     where  not exists (select *
                        from   AUS
                        where  Matnr=BU.Matnr) )
```

  Sicht kann also auf Sicht definiert werden

- Sicht SAM (Signatur Ausleihe Matrikel)

```
create view SAM(Signatur, AusleihDatum, Matrikel)
as select distinct Doknr, Datum, Matnr
   from   BIB_UNIV
```

  Sicht kann Relationen (oder Sichten) und Attribute dem Anwendungsbereich entsprechend umbenennen, also Anwendungsvokabular verwenden

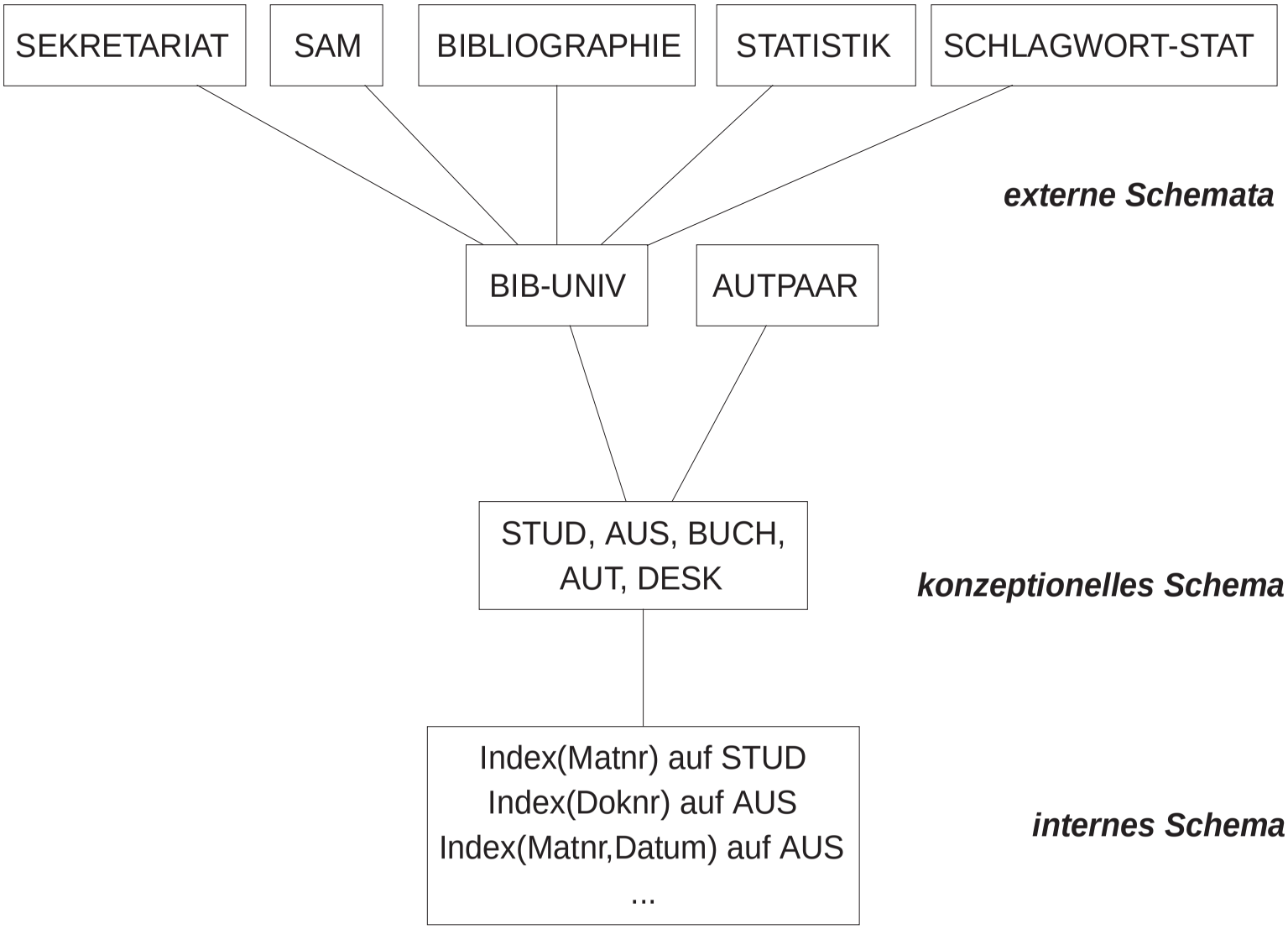
- Sicht Bibliographie

```
create view BIBLIOGRAPHIE(Doknr, Titel, Verlag,
                          Ort, Jahr, AName, Schlagwort)
as select distinct Doknr, Titel, Verlag, Ort, Jahr,
          AName, Schlagwort
   from   BIB_UNIV
```

- Sicht Statistik

```
create view STATISTIK(AnzahlStudis, AnzahlFaecher,
                      MinSemesterZahl, MaxSemesterZahl,
                      AnzahlDokumente)
as select count(distinct Matnr), count(distinct Fach),
          min(Sem), max(Sem), count(distinct Doknr)
   from   BIB_UNIV

create view SCHLAGWORT_STAT(Schlagwort, AnzahlDokumente)
as select   Schlagwort, count(distinct Doknr)
   from     BIB_UNIV
   group by Schlagwort
```


SEKRETARIAT
SAM
BIBLIOGRAPHIE
STATISTIK
SCHLAGWORT-STAT
externe Schemata
BIB-UNIV
AUTPAAR
STUD, AUS, BUCH,
AUT, DESK
konzeptionelles Schema
Index(Matnr) auf STUD
Index(Doknr) auf AUS
Index(Matnr,Datum) auf AUS
...
internes Schema

Probleme bei Änderung von Sichten

Beispiel-Schema

ESD(Employee:string, Salary:int, Dept:string)

DM(Dept:string, Manager:string)

- Projektionssicht $ED := \pi_{Employee,Dept}(ESD)$

  ```
  create view ED
  as select Employee, Dept
     from   ESD

  insert into ED values ('Zuse','Informatik')
  ```

  wird zu

  ```
  insert into ESD values ('Zuse',null,'Informatik')
  ```

  erfordert Vorhandensein von Nullwerten

  Nullwert steht hier für undefiniert

  daher in SQL kein `insert` auf Sichten, in denen an den entsprechenden Stellen Nullwerte verboten sind

- Selektionssicht
  $ES' := \sigma_{Salary>80K}(\pi_{Employee,Salary}(ESD))$
  $[= \pi_{Employee,Salary}(\sigma_{Salary>80K}(ESD))]$

```
create view ES'
as select Employee, Salary
   from   ESD
   where  Salary>80K
```

```
update ES' set Salary=70K where Employee='Zuse'
```

wird durch Anfragemodifikation zu

```
update ESD set Salary=70K where Employee='Zuse' and
                                Salary>80K
```

würde zur Löschung aus der Sicht führen

wird in SQL zurückgewiesen

- Verbundsicht $ESDM := ESD * DM$

```
create view ESDM
as select Employee, Salary, ESD.Dept, Manager
   from   ESD, DM
   where  ESD.Dept=DM.Dept

insert into ESDM values('Knuth',70K,'Info','Wirth')
```

Ausgangszustand: `('Info','Wirth')` $\notin$ `DM`

aber `('Math','Wirth')` $\in$ `DM`

induzierte Veränderung auf ESD

```
(1) insert into ESD values('Knuth',70K,'Info')
```

induzierte Veränderung auf DM

```
(2.a) insert into DM values('Info','Wirth')
```

oder

```
(2.b) update DM set Dept='Info' where Manager='Wirth'
```

in beiden Nachfolgezuständen ist das Tupel `('Knuth',70K,'Info','Wirth')` in der Sicht ESDM, also der Wunsch des inserts erfüllt

damit Anweisung nicht eindeutig übersetzbar

- weiteres Beispiel

```
delete ESDM where Employee='Knuth' and Salary=70K and
                  Dept='Info' and Manager='Wirth'
```

hat drei mögliche Realisierungen auf der Ebene des konzeptionellen Schemas

```
(1) delete ESD where Employee='Knuth' and
                     Salary=70K and Dept='Info'

(2) delete DM where Dept='Info' and
                    Manager='Wirth'

(3) delete ESD where Employee='Knuth' and
                     Salary=70K and Dept='Info'
    delete DM  where Dept='Info' and
                     Manager='Wirth'
```

solche Anweisungen werden wegen mangelder Eindeutigkeit in SQL zurückgewiesen

keine Änderungen auf Verbundsichten bei derartigen Konflikten

- berechnete Sicht

```
create view DS(Dept,SumSalary)
as select   Dept, sum(Salary)
   from     ESD
   group by Dept

update DS set SumSalary=SumSalary*2 where Dept='CS'
```

  nicht eindeutig übersetzbar; im Beispiel: Verteilung der Verdopplung der Gehaltssumme auf die Mitarbeiter nicht eindeutig abbildbar

  in SQL keine Änderungen auf berechneten Sichten

- rekursive Sichten

  siehe Beispiel zur transitiven Hülle von Relationen; Relationen ELT(Eteil,Kind) und VOR(Vorfahr,Nachkomme)

```
create view VOR(Vorfahr,Nachkomme)
as ( select Eteil, Kind
     from   ELT )
   union
   ( select ELT.Eteil, VOR.Nachkomme
     from   ELT, VOR
     where  ELT.Kind=VOR.Vorfahr )
```

  in SQL-89 verboten; erlaubt in Datalog; siehe auch Deduktive Datenbanken

```
VOR(X,Y) :- ELT(X,Y).
VOR(X,Y) :- ELT(X,Z), VOR(Z,Y).
```

- Sichten in ER-Modell

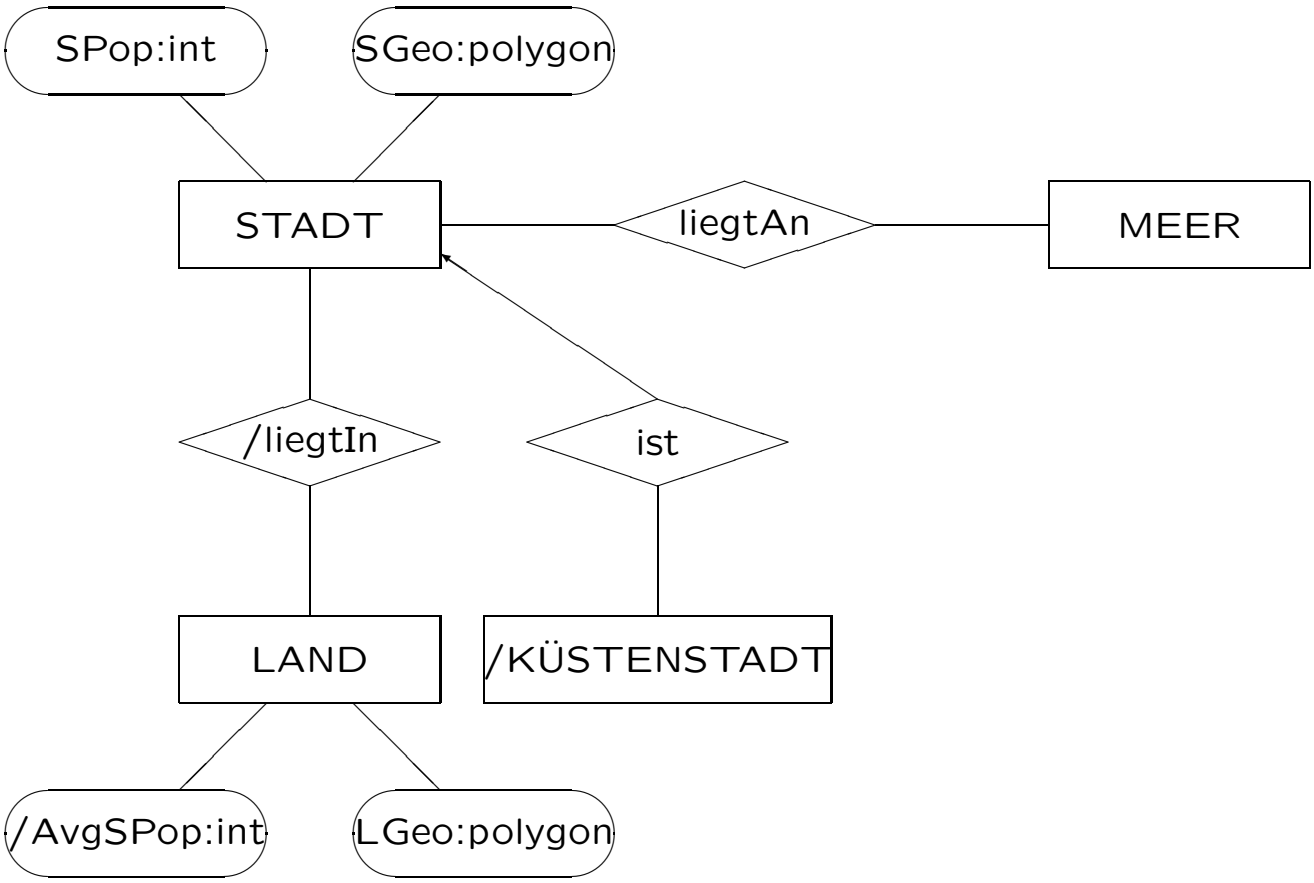
  im Prinzip sind Sichten im ER-Modell für alle Schemakomponenten (Attribute, Entititypen, Relationshiptypen) denkbar

  da Sichten immer mit Ableitungsregeln verbunden sind, spricht hier auch von abgeleiteten Schemakomponenten

  Beispiel

  Vorausgesetzt: komplexer Datentyp polygon;
  etwa: polygon = List(point);
  point = Record(int,int)

  für polygon sind auch entsprechende Operationen definiert; etwa:
  polygonCuts: polygon $\times$ polygon $\rightarrow$ bool




```
liegtIn(s:STADT,l:LAND) :=
   polygonCut(SGeo(s),LGeo(l))
AvgSPop(l:LAND) :=
   avg(select SPop(s) from STADT s where liegtIn(s,l))
KÜSTENSTADT :=
   select s from STADT s where (exists MEER m liegtAn(s,m))
```