3 Relationale Anfragesprachen

3.1 Relationale Algebra

Grundoperationen (Gegeben $R(A_1, ..., A_n), S(B_1, ..., B_m)$)

- 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 $1 \leq i_1, ..., i_k \leq Rang(R) = n$

  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 $\sigma_\varphi(R)$

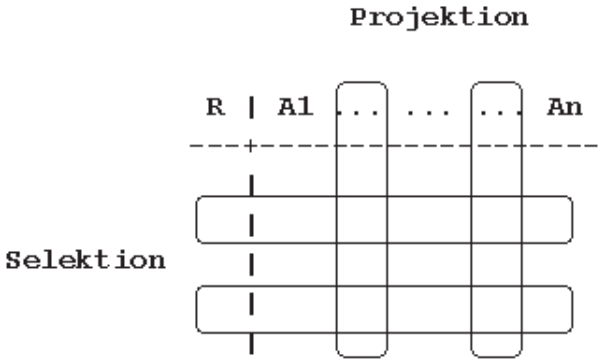
  Vorausetzung: φ ist atomare Formel $\Theta(u_1, ..., u_n)$; $u_1, ..., u_n$ sind Konstanten, Attributenamen oder Terme darüber; Θ ist boolesche Operation, wie Vergleiche $=, \neq, <, \leq, >, \geq$ oder andere boolesche Abfragen

  Beispiel: $(Gehalt * 1.04) + 120 > 4.000$, substring(Adresse,'Bremen')

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

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

  Beispiel: Mit $R(A, B)$ ergibt sich aus $\sigma_{B<b7}(R)$ als Ergebnis $\{r \mid r \in R \wedge r_2 < b7\}$



- 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:

  $\delta_{R.C \leftarrow A_i}(R) \times \delta_{S.C \leftarrow B_j}(S)$

  Es entsteht das Ergebnisschema $R'(A_1, ..., R.C, ..., A_n, B_1, ..., S.C, ..., B_m)$

  Falls R=S gilt, werden für $R \times S = R \times R$ implizite Umbenennungen vorgenommen, so daß für $R(A, B, ...)$ folgendes Ergebnisschema entsteht:

  $R'(A_1, B_1, ..., A_2, B_2, ...)$

Beispiel

R	A	B
	a1	b1
	a1	b2
	a2	b1

S	A	C
	a2	b1
	a3	b3

- $R \cup S$

R'	A	B
	a1	b1
	a1	b2
	a2	b1
	a3	b3

- $R - S$

R'	A	B
	a1	b1
	a1	b2

- $R \times S$ (zu den Umbennungen bei der Diskussion von δ genaueres)

R'	R.A	B	S.A	C
	a1	b1	a2	b1
	a1	b2	a2	b1
	a2	b1	a2	b1
	a1	b1	a3	b3
	a1	b2	a3	b3
	a2	b1	a3	b3

- $\pi_A(R)$

R'	A
	a1
	a2

- $\sigma_{A<a2}(R)$

  [Es gilt die lexikographische Sortierung: $a1 < a2 < a3 < b1 < b2 < b3$]

R'	A	B
	a1	b1
	a1	b2

- $\delta_{B \leftarrow C}(S)$

S'	A	B
	a2	b1
	a3	b3

Beispielschema

- STUDENT(Matnr:int,SName:string,Fach:string,Sem:int)
- AUSLEIHE(Doknr:string,Matnr:int,Datum:string)
- BUCH(Doknr:string,Titel:string,Verlag:string,Ort:string,Jahr:int)
- AUTOREN(Doknr:string,AName:string)
- DESKRIPTOREN(Doknr:string,Schlagwort:string)

1. Namen der Studenten und Autoren

   $\pi_{AName}(AUTOREN) \cup \pi_{SName}(STUDENT)$
2. Namen der Autoren die keine Studenten sind

   $\pi_{AName}(AUTOREN) - \pi_{SName}(STUDENT)$
3. Welcher Student könnte welches Buch ausleihen?

   $STUDENT \times BUCH$
4. Welcher Student studiert im wievielten Semester?

   $\pi_{SName,Sem}(STUDENT)$

5. Bücher aus dem Jahr der Gutenberg-Bibel

   $\sigma_{Jahr=1455}(BUCH)$

6. Benenne Schlagwort in Stichwort um

   $\delta_{Stichwort \leftarrow Schlagwort}(DESKRIPTOREN)$

7. Verknüpfung von AUSLEIHE und BUCH über gemeinsame Dokumentnummern

   $\sigma_{AUSLEIHE.Doknr=BUCH.Doknr}(AUSLEIHE \times BUCH)$

   mit expliziter Umbenennung

   $\sigma_{AUSLEIHE.Doknr=BUCH.Doknr}(\delta_{AUSLEIHE.Doknr \leftarrow Doknr}(AUSLEIHE) \times \delta_{BUCH.Doknr \leftarrow Doknr}(BUCH))$

8. Erster Vergleich SQL vs RA

```
select Datum
from   STUDENT, AUSLEIHE
where  SName='Zimmermann' and STUDENT.Matnr=AUSLEIHE.Matnr
```

   $\pi_{Datum}(\sigma_{STUD.Matnr=AUS.Matnr}(\sigma_{SName='Zimmermann'}(STUD \times AUS)))$
   $\pi_{Datum}(\sigma_{SName='Zimmermann'}(\sigma_{STUD.Matnr=AUS.Matnr}(STUD \times AUS)))$
   $\pi_{Datum}(\sigma_{STUD.Matnr=AUS.Matnr}(\sigma_{SName='Zimmermann'}(STUD) \times AUS))$
   (Relationennamen abgekürzt)

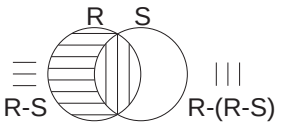
abgeleitete Operationen

- Durchschnitt $R \cap S$

  Voraussetzung: wie $R \cup S$

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

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



- verallgemeinerte Selektion $\sigma_{\varphi}(R)$ (ÜBUNGSAUFGABE)

  Vorausetzung: φ logische Verküpfung von atomaren Formeln mittels $\wedge, \vee \neg$

  Definition: analog zur einfachen Selektion

- 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 natürliche 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 $S(B_1, ..., B_m)$

  Ergebnisschema: $R'(A_1, ..., A_{n-m})$

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

U	A	B
4	a
4	b
5	b
5	c
6	a
6	b
6	c

V	B
b
c

U:V

U'	A
5
6

- R mod S

  Analogie zur Modulo-Operation auf den natürlichen Zahlen

  $n \ mod \ m$ ist die Differenz zwischen $(n : m) * m$ und n
  $R \ mod \ S$ ist die Differenz zwischen $(R : S) \times S$ und R

  Voraussetzung: wie bei der Division

  Ergebnisschema: $R'(A_1, ..., A_{n-m}, B_1, ..., B_m) = R'(A_1, ..., A_n)$

  Ableitung:

  $$\begin{aligned} R \ mod \ S &:= R - (R : S) \times S \\ &= R - (\pi_{A_1,...,A_{n-m}}(R) - \pi_{A_1,...,A_{n-m}}((\pi_{A_1,...,A_{n-m}}(R) \times S) - R)) \times S \end{aligned}$$

  Es gilt:

  $((n : m) * m) + (n \ mod \ m) = n \quad ((7 : 3) * 3) + (7 \ mod \ 3) = (2 * 3) + 1 = 6 + 1 = 7$
  $((R : S) \times S) \cup (R \ mod \ S) = R$

Beispiele (R und S wie oben)

R	A	B
	a1	b1
	a1	b2
	a2	b1

S	A	C
	a2	b1
	a3	b3

- $R \cap S$

R'	A	B
	a2	b1

- $\sigma_{A=a1 \vee B=b1}(R)$

R'	A	B
	a1	b1
	a1	b2
	a2	b1

- $R *_{B<C} S$

R'	R.A	B	S.A	C
	a1	b1	a3	b3
	a1	b2	a3	b3
	a2	b1	a3	b3

- $R * S$

R'	R.A	B	C
	a2	b1	b1

- $R : T = R : \pi_B(R)$

T	B
	b1
	b2

R'	A
	a1

Probe:

R' x T	A	B
	a1	b1
	a1	b2

$\subseteq$ R

- $R \; mod \; T$

R'	A	B
	a2	b1

1. Welche Informatik-Studenten haben eine Semesterzahl von mindestens 5?
   $\sigma_{Sem \geq 5 \wedge Fach='Informatik'}(STUDENT)$
2. Verbinde STUDENT und AUTOREN über die Attribute SName und AName
   $STUDENT *_{SName=AName} AUTOREN$

3. Titel der Ullman-Bücher

   $\pi_{Titel}(\sigma_{AName='Ullman'}(BUCH * AUTOREN))$

   $\pi_{Titel}(BUCH * (\sigma_{AName='Ullman'}(AUTOREN)))$ [VERSCHIEBUNG von σ]

   $\pi_{Titel}(\sigma_{AName='Ullman'}(\sigma_{BUCH.Doknr=AUTOREN.Doknr}(BUCH \times AUTOREN)))$

   Hier in der Notation eine implizite Umbenennung verwendet; eigentlich:

   $\pi_{Titel}(\sigma_{AName='Ullman'}(\sigma_{BUCH.Doknr=AUTOREN.Doknr}($
   $\delta_{BUCH.Doknr \leftarrow Doknr}(BUCH) \times \delta_{AUTOREN.Doknr \leftarrow Doknr}(AUTOREN))))$

4. Welche Mitarbeiter arbeiten in allen Projekten?

   klassisches Beispiel zur Division über dem Relationenschema
   JOB(Mitarb,Projekt)

   $JOB : \pi_{Projekt}(JOB) =$

   $\{\pi_{Mitarb}(job) | job \in JOB \wedge \forall projekt \in \pi_{Projekt}(JOB)[\pi_{Mitarb}(job) \circ projekt \in JOB]\} =$

   $$\pi_{Mitarb}(JOB) - \pi_{Mitarb}(\underbrace{\underbrace{[\pi_{Mitarb}(JOB) \times \pi_{Projekt}(JOB)] - JOB}_{\{(Mitarb,Projekt) | (Mitarb,Projekt) \notin JOB\}}}_{\{Mitarb | \exists Projekt (Mitarb,Projekt) \notin JOB\}})$$

   $JOB \ mod \ \pi_{Projekt}(JOB) =$ alle Mitarbeiter, die nicht in allen Projekten arbeiten

5. Bücher mit DB und PS (Programmiersprachen) als Schlagwort

   $BUCH * ($
   $\pi_{Doknr}(\sigma_{Schlagwort='DB'}(DESKRIPTOREN)) \cap$
   $\pi_{Doknr}(\sigma_{Schlagwort='PS'}(DESKRIPTOREN)))$

   { ('DB'), ('PS') } : S(Schlagwort) $\hat{=}$ konstante Relation über Schlagworten

   $BUCH * (DESKRIPTOREN : \{(`DB'), (`PS')\})$

6. Dokumentnummern der ältesten Bücher

   $\pi_{Doknr}(BUCH) - \pi_{Doknr}(\sigma_{Jahr_1 > Jahr_2}(\pi_{Doknr,Jahr}(BUCH) \times \pi_{Jahr}(BUCH)))$

   Rechts von $-$ werden die Dokumentnummern von Büchern berechnet, zu denen es ein Buch gibt das vor diesem Buch erschienen ist, d.h. Dokumentnummern von 'neueren' Büchern

   - DB-Zustand mit einem Dokument

BUCH	Doknr	Titel	Verlag	Ort	Jahr
	D1	...	...	...	1980

$\pi_{Doknr,Jahr}(BUCH) \times \pi_{Jahr}(BUCH)$	Doknr	$Jahr_1$	$Jahr_2$
	D1	1980	1980

   $\pi_{Doknr}(\sigma_{Jahr_1 > Jahr_2}(...)) = \emptyset$

   Ergebnis $\{D1\}$

   - DB-Zustand mit zwei Dokumenten

BUCH	Doknr	Titel	Verlag	Ort	Jahr
	D1	...	...	...	1980
	D2	...	...	...	1981

$\pi_{Doknr,Jahr}(BUCH) \times \pi_{Jahr}(BUCH)$	Doknr	Jahr_1	Jahr_2
	D1	1980	1980
	D1	1980	1981
	D2	1981	1980
	D2	1981	1981

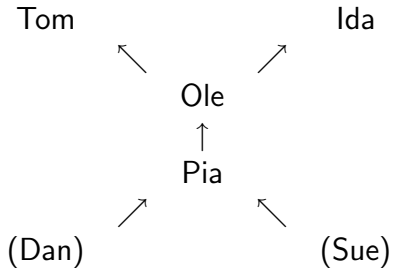
$\pi_{Doknr}(\sigma_{Jahr_1 > Jahr_2}(...)) = \{D2\}$
Ergebnis $\{D1\}$

7. Dokumentnummern der Bücher, die alle vorkommenden Schlagworte haben
   $DESKRIPTOREN : \pi_{Schlagwort}(DESKRIPTOREN)$
8. Welche Autoren liest Studi Zimmermann?
   $\pi_{AName}(\sigma_{SName=`Zimmermann'}((AUTOREN * AUSLEIHE) * STUDENT))$
   linker Verbund über Doknr; rechter Verbund über Matnr
   $\pi_{AName}((AUTOREN * AUSLEIHE) * \sigma_{SName=`Zimmermann'}(STUDENT))$

[weitere Beispiele]

- Beispiel: transitive Hülle

ELTERN	Eteil	Kind
	(Dan)	(Pia)
	(Sue)	(Pia)
	Pia	Ole
	Ole	Tom
	Ole	Ida



ELTERN* = VORFAHREN	Vorfahr	Nachkomme
	(Dan)	(Pia)
	(Sue)	(Pia)
	Pia	Ole
	Ole	Tom
	Ole	Ida
	———	———
	(Dan)	(Ole)
	(Dan)	(Tom)
	(Dan)	(Ida)
	(Sue)	(Ole)
	(Sue)	(Tom)
	(Sue)	(Ida)
	Pia	Tom
	Pia	Ida

Satz zur RA [besagt]: $\neg \exists$ Term $\tau \in$ RA mit $\tau(ELTERN) = VORFAHREN$ für alle DB-Zustände

[Es gibt allerdings Terme der RA zur Berechnung von R^k für jedes $k \in N$]

Aber: $ELTERN^2 =$
$ELTERN \cup \pi_{Eteil_1,Kind_2}(ELTERN *_{Kind_1=Eteil_2} ELTERN)$

$ELTERN^3 = ...$

[ÜBUNGSAUFGABE: Geben Sie einen Term der RA an, der $ELTERN^3$ korrekt berechnet, nicht aber $ELTERN^4$. Weiter anzugeben: (1) Zustand in dem es mindestens eine Urgrossmutter/Urgrossvater-Urenkel-Beziehung gibt (Kantenfolge der Länge 3), (2) Auswertung des Term dort, (3) Zustand in dem es mindestens eine Ururgrossmutter/Ururgrossvater-Ururenkel-Beziehung gibt (Kantenfolge der Länge 4), (4) Auswertung des Terms dort.]

- auch Integritätsbedingungen (IBen) mit RA formulierbar; IB = Gleichung oder Inklusion zwischen RA-Termen

  - STUD[ENT](Matnr:int,SName:string,Fach:string,Sem:int)
  - AUS[LEIHE](Doknr:string,Matnr:int,Datum:string)
  - BUCH(Doknr:string,Titel:string,Verlag:string,Ort:string,Jahr:int)
  - AUT[OREN](Doknr:string,AName:string)
  - DESK[RIPTOREN](Doknr:string,Schlagwort:string)

  Beispiel: Doknr aus AUS kommt in BUCH vor

  $\pi_{Doknr}(AUS) \subseteq \pi_{Doknr}(BUCH)$

  Beispiel: {Matnr} ist Schlüssel in STUD

  $\sigma_{Matnr_1=Matnr_2 \wedge (SName_1 \neq SName_2 \vee Fach_1 \neq Fach_2 \vee Sem_1 \neq Sem_2)}(STUD \times STUD) = \emptyset$

STUD	Matnr	SName	Fach	Sem
	1234	Ada	MI	5
	1234	Bob	MI	5

IB gilt hier nicht; durch Relationenmodell Zustand auch schon verboten

  Beispiel: maximal eine Ausleihe pro Doknr und Datum

  $\sigma_{Doknr_1=Doknr_2 \wedge Datum_1=Datum_2 \wedge Matnr_1 \neq Matnr_2}(AUS \times AUS) = \emptyset$

AUS	Doknr	MatNr	Datum
	D4711	M42	31.12.99
	D4711	M43	31.12.99

IB gilt hier nicht

  Beispiel: nur Piloten eingesetzt für Abflüge

  ANGESTELLTER(...,Personal#)

  PILOT(#Flugst,Personal#)

  eingesetzt_fuer(Datum,Flug#,Personal#)

  $\pi_{Personal\#}(eingesetzt\_fuer) \subseteq \pi_{Personal\#}(PILOT)$

3.2 Bereichskalkül

Motivation relationale Kalküle

- SQL-Anfragen

```
select T1, ..., Tn
from   R1, ..., Rk
where  F
```

- Formel F im where-Teil mit and, or, not und Unteranfragen: exists (select... from... where...); auch weitere Formen von Unteranfragen

  [F damit im Prinzip Formel des Tupelkalküls: $\vee$, $\wedge$, $\neg$, $\exists$, $\forall$ ($\equiv \neg\exists\neg$)]

Beispiel: Terme 123:int; 'Ullman':string; x:int; max(123+x,y):int

Beispiel: atomaren Formeln max(x,y) $<$ 100; DESK(d,'DB')

Beispiel: Formeln; Schemata R(A:int,B:int), S(C:int,B:int)

$R(4, 16) \wedge S(16, 18)$ [keine Variablen]

$\exists x(R(13, x) \wedge S(x, 14))$ [in Klammern: gebundene Vorkommen von x]

$R(4, x) \wedge \forall x(R(13, x) \Rightarrow z \leq x)$ Erinnerung: $(A \Rightarrow B)$ gdw $(\neg A \vee B)$

[1. Vorkommen von x frei, letztes und vorletztes gebunden; Vorkommen von z frei]

[Analogie: Gültigkeitsbereich von Variablen in Formeln VERSUS Verwendung von Variablen in Block-orientierten Programmiersprachen; Bindung einer Variablen mittels $\exists\ x\ (\varphi)$ oder $\forall\ x\ (\varphi)$ VERSUS Deklaration einer Variablen in einem Block]

Beispiel: Gültigkeit von Formeln

- [Der betrachtete Zustand σ ist durch folgende Tabellen festgelegt]

R	A:int	B:int	
	1	2	
	2	4	$\sigma(R)$
	3	6	

S	C:int	B:int	
	1	2	
			$\sigma(S)$
	4	4	

- [Es wird nun die Gültigkeit der folgenden Formeln φ_1 und φ_2 unter verschiedenen Belegungen β untersucht]

  $\varphi_1 \equiv R(x, y) \wedge \exists x(\ \underbrace{S(x, y) \wedge x - 1 < y}_{\varphi_2}\ ) \wedge z = \ x + y$

  [freie und gebundene Vorkommen der Variablen in obiger Formel diskutieren; alle Vorkommen von y,z frei; x frei und gebunden]

- $\beta_1 : x \mapsto 4, y \mapsto 4, z \mapsto 6$

  $\beta_2 : x \mapsto 2, y \mapsto 4, z \mapsto 6$

- Gilt $[\sigma, \beta_1] \models \varphi_2$? Antwort: ja

  Es gilt $[\sigma, \beta_1] \models S(x, y)$ und $[\sigma, \beta_1] \models (x - 1 < y)$

  $(x, y) = (4, 4) \in \sigma(S)$ und $x - 1 = 4 - 1 = 3 < 4 = y$

- Gilt $[\sigma, \beta_2] \models \varphi_1$? Antwort: ja

  $[\sigma, \beta_2] \models R(x, y)$ da $(2, 4) \in \sigma(R)$ und

  $[\sigma, \beta_2] \models \exists x(\varphi_2)$ und

  $[\sigma, \beta_2] \models (z = x + y)[\equiv (6 = 2 + 4)]$.

  [Man beachte, daß] $[\sigma, \beta_2] \models \exists x(\varphi_2)$ gilt, da $[\sigma, \beta_1] \models \varphi_2$ gilt

  [Für die Gültigkeit von φ_1, kann man sich also eine neue, quasi lokale Belegung der Variablen x suchen]

- Gilt $[\sigma, \beta_1] \models \varphi_1$? Antwort: nein

  Es gilt $(x, y) = (4, 4) \notin \sigma(R)$ oder $z = 6 \neq 8 = 4 + 4 = x + y$.

- Gilt $[\sigma, \beta_2] \models \varphi_2$? Antwort: nein

  Es gilt $(x, y) = (2, 4) \notin \sigma(S)$.

Beispiele: Anfragen

- Titel der Ullman-Bücher

  $\{t \mid \exists d, v, o, j, d2, an(BUCH(d, t, v, o, j) \wedge AUT(d2, an) \wedge$
  $d = d2 \wedge an = `Ullman')\}$

  eigentlich: $\{t \mid \exists d\ (\exists v\ (\exists o\ (\exists j\ (\exists d2\ (\exists an\ (...)))))))\}$

  $\{t \mid \exists d, v, o, j(BUCH(d, t, v, o, j) \wedge$
  $\exists d2, an(AUT(d2, an) \wedge d = d2 \wedge an = `Ullman'))\}$

  $\{t \mid \exists d, v, o, j(BUCH(d, t, v, o, j) \wedge AUT(d, `Ullman'))\}$

  $\{t : string \mid \exists d : string, v : string, o : string, j : int$
  $(BUCH(d, t, v, o, j) \wedge AUT(d, `Ullman'))\}$

- Bücher mit DB und PS als Schlagwort

  $\{d, t, v, o, j | BUCH(d, t, v, o, j) \wedge$
  $DESK(d, `DB') \wedge DESK(d, `PS')\}$

- Dokumentnummern der ältesten Bücher

  $\{d \mid \exists t, v, o, j(BUCH(d, t, v, o, j) \wedge$
  $\forall d2, t2, v2, o2, j2(BUCH(d2, t2, v2, o2, j2) \Rightarrow j \leq j2))\}$

  Was ist das Delta zu:

  $\{d \mid \exists t, v, o, j(BUCH(d, t, v, o, j) \wedge$
  $\forall d2, t2, v2, o2, j2(BUCH(d2, t2, v2, o2, j2) \wedge j \leq j2))\}$

  $\equiv$

$\{d \mid \exists t, v, o, j(BUCH(d, t, v, o, j) \wedge \forall j2(j \leq j2 \wedge ...))\}$

ergibt $\emptyset$

- Dokumentnummern der Bücher, die alle vorkommenden Schlagworte haben

$$\{d \mid \exists sw(\quad DESK(d, sw) \wedge \\ \forall sw2((\exists d2\ DESK(d2, sw2)) \Rightarrow DESK(d, sw2)))\}$$

- Welche Autoren liest Studi Zimmermann

$\{an \mid \exists d, m, dt, f, sm$
$(AUT(d, an) \wedge AUS(d, m, dt) \wedge STUD(m, `Zimmermann', f, sm))\}$

$\{an \mid \exists d(AUT(d, an) \wedge$
$\exists m, dt(AUS(d, m, dt)) \wedge$
$\exists f, sm(STUD(m, `Zimmermann', f, sm)))\}$

$\{an \mid \exists d(AUT(d, an) \wedge$
$\exists m(AUS(d, m, *) \wedge STUD(m, `Zimmermann', *, *)))\}$

*-Notation im BK: Jeder * entpricht einer neuen, existentiell quantifizierten Bereichsvariablen mit dem kleinstmöglichen Gültigkeitsbereich

[Obacht: *-Notation im BK damit unterschiedlich von *-Notation in SQL]

sichere Ausdrücke

- Beispiel für einen nicht-sicheren Ausdruck

  $\{x, y \mid \neg R(x, y)\}$

  [Ergebnis der obigen Anfrage im allgemeinen unendlich; prinzipielles Problem: x,y kommen nicht in Relationen vor]

- [Idee: Jede] Ergebnisvariable x in nichtnegierter Form an einen Wert aus einer Relation oder an Konstante binden [und so endliches Ergebnis garantieren]

  $R_1(..., x, ...) \vee ... \vee R_p(..., x, ...) \vee x = c_1 \vee ... \vee x = c_k$

  mit $p, k \in N_0$ und $p + k \geq 1$

  [aus einer Bindung dieser Gestalt ergibt sich, daß das Ergebnis immer endlich ist]

- syntaktisches Kriterium für BK-Ausdrücke

$\{x_1, ..., x_n \mid$
$[R_1(..., x_1, ...) \vee ... \vee R_p(..., x_1, ...) \vee x_1 = c_1 \vee ... \vee x_1 = c_k] \wedge$
$...$ $\equiv$ SAFE
$[S_1(..., x_n, ...) \vee ... \vee S_m(..., x_n, ...) \vee x_n = d_1 \vee ... \vee x_n = d_q] \wedge$
$\varphi\}$

mit beliebigem φ

Sicherer Ausdruck: Ausdruck der Gestalt SAFE oder durch einfache Umformungen daraus herleitbarer Ausdruck

- R(A:int,B:int), S(C:int,B:int)

  $\{x \mid R(x,*) \vee R(*,x) \vee S(x,*) \vee S(*,x) \vee x = 42 \vee x = 43\}$ erfüllt Kriterium

  R(A:int)

  $\{x \mid R(x)\}$ erfüllt Kriterium

  $\{x \mid \exists y \exists z(x = y \wedge y = z \wedge R(z))\}$ ist daraus herleitbar

  Beispiel: nicht-sicherer Ausdruck kann komplexer sein als obiges Beispiel

  $\{x, y \mid \neg R(x,y)\}$

  $B \equiv R(x,y)$

  $\{x, y \mid \neg B \wedge (A \vee \neg A)\}$

  $\{x, y \mid (\neg B \wedge A) \vee (\neg B \wedge \neg A)\}$

  $\{x, y \mid \neg\neg(\neg B \wedge A) \vee \neg\neg(\neg B \wedge \neg A)\}$

  $\{x, y \mid \neg(B \vee \neg A) \vee \neg(B \vee A)\}$

  $\{x, y \mid (B \vee \neg A) \Rightarrow \neg(B \vee A)\}$

  $\{x, y \mid (A \Rightarrow B) \Rightarrow \neg(B \vee A)\}$

  $A \equiv \exists z(R(x,z))$

  $\{x, y \mid (\exists z(R(x,z)) \Rightarrow R(x,y)) \Rightarrow \neg(R(x,y) \vee \exists w(R(x,w)))\}$

- alle obigen Anfragen an Bib-DB (die Grundformen ohne die Variationen) sind sicher

- Sicherheit nur hinreichendes Kriterium für endliches Ergebnis

  Beispiel: nicht-sicherer BK-Ausdruck, der stets endliches Ergebnis liefert zum Schema $R(A : int)$

  $\{x : int \mid \exists\ min : int\ \exists\ max : int\ (\ R(min) \wedge R(max) \wedge$
  $\forall\ z : int(R(z) \Rightarrow (min \leq z \wedge z \leq max)) \wedge$
  $min \leq x \wedge x \leq max\ )\}$

Darstellung von RA-Operationen im BK; gegeben $R(A_1, ..., A_n)$ und $S(B_1, ..., B_m)$

- $R \cup S = \{x_1, ..., x_n \mid R(x_1, ..., x_n) \vee S(x_1, ..., x_n)\}$
- $R - S = \{x_1, ..., x_n \mid R(x_1, ..., x_n) \wedge \neg S(x_1, ..., x_n)\}$
- $R \times S = \{x_1, ..., x_n, x_{n+1}, ..., x_{n+m} \mid R(x_1, ..., x_n) \wedge S(x_{n+1}, ..., x_{n+m})\}$
- $\pi_{\overline{A}}(R) = \{x_{i_1}, ..., x_{i_k} \mid \exists x_{i_{k+1}}, ..., \exists x_{i_n} R(x_1, ..., x_n)\}$ mit $\overline{A} = A_{i_1}, ..., A_{i_k}$ und $\{i_1, ..., i_k, i_{k+1}, ..., i_n\} = \{1, ..., n\}$

  Beispiel: Mit $R(A_1, A_2, A_3)$ wird $\pi_{A_3,A_2}(R)$ übersetzt nach $\{x_3, x_2 \mid \exists x_1 R(x_1, x_2, x_3)\}$ oder in *-Notation $\{x_3, x_2 \mid R(*, x_2, x_3)\}$

- $\sigma_\varphi(R) = \{x_1, ..., x_n \mid R(x_1, ..., x_n) \wedge \varphi'\}$

  mit $\varphi' \equiv \varphi$ mit Variablen x_i statt Attributnamen A_i

  Beispiel: $\sigma_{A_1=42}(R) = \{x_1, x_2, x_3 \mid R(x_1, x_2, x_3) \wedge x_1 = 42\}$

3.3 Tupelkalkül

Flüge von Ullman?

BUCHUNG(Name:string,Adr:string,Flug#:string,Datum:string)

- RA-Term

  $\pi_{Flug\#,Datum}(\sigma_{Name='Ullman'}(BUCHUNG))$

- BK-Ausdruck

  $\{f, d | \exists n, a(BUCHUNG(n, a, f, d) \wedge n = 'Ullman')\}$

  $\{f, d | BUCHUNG('Ullmann', *, f, d)\}$

- TK-Ausdruck

  $\{r : (Flug\#, Datum) \mid \exists b : (Name, Adr, Flug\#, Datum)(BUCHUNG(b) \wedge$
  $b.Name = 'Ullman' \wedge r.Flug\# = b.Flug\# \wedge r.Datum = b.Datum)\}$

  $\{r : (Flug\#, Datum) \mid \exists b : BUCHUNG$
  $(b.Name = 'Ullman' \wedge r.Flug\# = b.Flug\# \wedge r.Datum = b.Datum)\}$

Beispiel: Anfragen in TK

- Titel der Ullman-Bücher

  $\{r : (Titel) \mid \exists b : (Doknr, Titel, Verlag, Ort, Jahr), a : (Doknr, AName)$
  $(BUCH(b) \wedge AUT(a) \wedge$
  $r.Titel = b.Titel \wedge b.Doknr = a.Doknr \wedge a.AName = 'Ullman')\}$

  $\{r : (Titel) \mid \exists b : BUCH, a : AUT$
  $(r.Titel = b.Titel \wedge b.Doknr = a.Doknr \wedge a.AName = 'Ullman')\}$

- Beispiel zur Abkürzung $r : R$ mit $R(A : int, B : int)$

  - Beispielzustand

R	A	B
	1	2
	1	4

  - Für $\varphi \equiv (r.A = 1 \wedge 2 < r.B \wedge r.B < 4)$ gilt:

    $\exists\ r : (A : int, B : int)\ \varphi$ ist wahr

    $\exists\ r : R\ \varphi$ ist falsch

    $[\exists\ r : R\ \varphi] \equiv [\exists\ r : (A : int, B : int)(R(r) \wedge \varphi)]$

- Bücher mit Datenbanken (DB) und Programmierspachen (PS) als Schlagwort

  $\{r : (Titel) \mid \exists b : BUCH, d1 : DESK, d2 : DESK$
  $(r.Titel = b.Titel \wedge$
  $b.Doknr = d1.Doknr \wedge d1.Schlagwort = 'DB' \wedge$
  $b.Doknr = d2.Doknr \wedge d2.Schlagwort = 'PS')\}$

- Dokumentnummern der ältesten Bücher

  $\{r : (Doknr) \mid \exists b : BUCH(r.Doknr = b.Doknr \wedge$
  $\forall b' : BUCH(b.Jahr \leq b'.Jahr))\}$

- Dokumentnummern der Bücher die alle vorkommenden Schlagworte haben

  $\{r : (Doknr) \mid \exists d : DESK(r.Doknr = d.Doknr \wedge$
  $\forall d' : DESK(DESK(d.Doknr, d'.Schlagwort)))\}$

  ERWEITERUNG TK: Für $R(A_1, ..., A_n)$ gilt:

  $R(t_1, ..., t_n) :\Leftrightarrow \exists\ r : R\ (r.A_1 = t_1 \wedge ... \wedge r.A_n = t_n)$
  $\Leftrightarrow \exists\ r : (A_1, ..., A_n)\ (R(r) \wedge r.A_1 = t_1 \wedge ... \wedge r.A_n = t_n)$

- Welche Autoren liest Zimmermann?

  $\{r : (AName) \mid \exists aut : AUT, aus : AUS, s : STUD$
  $(r.AName = aut.AName \wedge aut.Doknr = aus.Doknr \wedge$
  $aus.Matnr = s.Matnr \wedge s.SName = `Zimmermann')\}$

  $\{r : (AName) \mid \exists aut : AUT(r.AName = aut.AName \wedge$
  $\exists aus : AUS(aut.Doknr = aus.Doknr \wedge$
  $\exists s : STUD(aus.Matnr = s.Matnr \wedge s.SName = `Zimmermann')))\}$

  - SQL-Formulierung der Beispielanfrage nach den Zimmermann-Autoren somit auch anders formulierbar
  - ```
    select AName
    from   AUT, AUS, STUD
    where  AUT.Doknr=AUS.Doknr and
           AUS.Matnr=STUD.Matnr and
           SName="Zimmermann"
    ```
  - mittels obiger Äquivalenzrechnung

    ```
    select AName
    from   AUT
    where  exists (select *
                   from   AUS, STUD
                   where  AUT.Doknr=AUS.Doknr and
                          AUS.Matnr=STUD.Matnr and
                          SName="Zimmermann")
    ```
  - oder sogar

    ```
    select AName
    from   AUT
    where  exists (select *
                   from   AUS
                   where  AUT.Doknr=AUS.Doknr and
                          exists (select *
                                  from   STUD
                                  where  AUS.Matnr=STUD.Matnr and
                                         SName="Zimmermann"))
    ```
    ```

Vergleich BK VERSUS TK [bzgl. Kriterium Bequemlichkeit]

- Vorteil TK: eine Tupelvariable statt mehrerer zugehöriger Bereichsvariablen
- Nachteil TK: Überlappungen von Tupelvariablen erfordern zusätzliche Konjunktionen von Komponentengleichungen
- Nachteil BK: In $R(x_1, ..., x_n)$ Position, nicht Name entscheidend
- Ausdrucksfähigkeit äquivalent

3.4 SQL

Beispielschema: KAL-Schema

KUNDE(KName, KAdr, Kto)

AUF[TRAG](KName, Ware, Menge)

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

1. Welche Kunden (KName) haben ihr Konto überzogen?

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

   äquivalenter Algebra-Term

   $\pi_{KName}(\sigma_{Kto<0}(KUNDE))$

   äquivalenter Tupelkalkül-Ausdruck

   $\{r : (KName) | \exists k : KUNDE(r.KName = k.KName \wedge k.Kto < 0)\}$

2. `select` τ `from` R_1, R_2 `where` $\varphi \equiv \pi_\tau(\sigma_\varphi(R_1 \times R_2))$

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

   SQL liefert im allgemeinen als Ergebnis keine Menge sondern eine Multimenge, d. h. eine Kollektion in der Elemente mehrmals auftreten können. Beispielsweise würde ein Lieferant der Milch und Mehl liefert im Ergebnis zweimal auftauchen. Diese Duplikate können mittels `select distinct` eliminiert werden.

   (select distinct TERM ...) : set(type)

   (select TERM ...) : bag(type)

   (...) union (...) : set(type)

   (...) union all (...) : bag(type)

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

$\{r : (LName, LAdr) |$
$\exists l : LIEF(r.LName = l.LName \wedge r.LAdr = l.LAdr \wedge$
$(l.Ware = `Milch' \vee l.Ware = `Mehl'))\}$

äquivalent mit expliziter Tupelvariable

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

oder Relationenname als implizite Tupelvariable

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

4. 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'
```

`like`: Substringsuche mit % $\widehat{=}$ Zeichenkette beliebiger Länge; _ $\widehat{=}$ Zeichenkette der Länge 1

SQL-Regeln für die Verwendung von Tupelvariablen (Qualifikationen)

- Tupelvariable können weggelassen werden solange ein Attribut eindeutig einer Relation bzw. einer impliziten Deklaration einer Tupelvariablen zugeordnet werden kann.
- Fehlt eine Tupelvariable, so wird angenommen, daß dort die Tupelvariable steht, die im logisch nächsten select-from-where-Block deklariert wurde.
- Der Relationenname kann als implizit deklarierte Tupelvariable verwendet werden.
- Tupelvariable müssen bei Mehrdeutigkeiten benutzt werden, falls z. B. ein Attribut in mehreren Relationen auftritt.
- Es gilt: implizite und explizite Tupelvariable können verwendet werden
- Es gilt: Tupelvariablen sind obligatorisch bei Mehrdeutigkeiten

ANFRAGE $\widehat{=}$

$\pi_{LName,LIEF.Ware}($
$\sigma_{substring(`Bremen',LAdr) \wedge LIEF.Ware=AUF.Ware \wedge KName=`Weiss'}($
$LIEF \times AUF))$

$\widehat{=}$

$\{r : (LName, Ware) | \exists l : LIEF, a : AUF$
$(r.LName = l.LName \wedge r.Ware = l.Ware \wedge$
$substring(`Bremen', l.LAdr) \wedge$
$l.Ware = a.Ware \wedge a.KName = `Weiss')\}$

äquivalent

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

$\{r : (LName, Ware) | \exists l : LIEF$
$(r.LName = l.LName \wedge r.Ware = l.Ware \wedge$
$substring(`Bremen', l.LAdr) \wedge$
$\exists a : AUF(l.Ware = a.Ware \wedge a.KName = `Weiss'))\}$

äquivalent zu `= any` ist `in`

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

5. 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'
```

$\{r : (LName) | \exists l : LIEF, lg : LIEF$
$(r.LName = l.LName \wedge l.Ware = lg.Ware \wedge lg.LName = `Grau')\}$

```
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' )
```

$\{r : (LName) | \exists l : LIEF(r.LName = l.LName \wedge$
$\exists lg : LIEF(l.Ware = lg.Ware \wedge lg.LName = `Grau'))\}$

6. 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)
```

$$\{r : (LName, LAdr, Ware, Preis) | \exists l : LIEF(r = l \wedge \\ \forall l' : LIEF(l'.Ware = l.Ware \Rightarrow l.Preis <= l'.Preis))\}$$

Man beachte, daß diese Anfrage nicht kann entschachtelt werden kann. Die Angabe $*$ in der `select`-Liste liefert alle Attribute.

FRAGE: Was ist das Delta zu: obige Implikation durch Konjunktion ersetzen?

7. 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)
```

$$\{r : (KName, LName, LAdr) | \exists k : KUNDE, l : LIEF \\ (r.KName = k.KName \wedge r.LName = l.LName \wedge r.LAdr = l.LAdr \wedge \\ \exists a : AUF(a.Ware = l.Ware \wedge a.KName = k.KName))\}$$

Die Anfrage kann auch mit `= any` anstelle von `exists` formuliert werden:

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

oder

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

8. 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))
```

$\{r : (LName) | \exists res : LIEF$
$(r.LName = res.LName \wedge$
$\neg(\exists gr : LIEF(gr.LName = `Grau' \wedge$
$\neg(\exists reswa : LIEF(gr.Ware = reswa.Ware \wedge reswa.LName = res.LName)))))\}$

$\{r : (LName) | \exists res : LIEF$
$(r.LName = res.LName \wedge$
$(\forall gr : LIEF(gr.LName = `Grau' \Rightarrow$
$(\exists reswa : LIEF(gr.Ware = reswa.Ware \wedge reswa.LName = res.LName)))))\}$

RA-Term $\pi_{LName,Ware}(LIEF) : \pi_{Ware}(\sigma_{LName=`Grau'}(LIEF))$

9. Welche Waren sind in Auftrag gegeben oder lieferbar?

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

$\{r : (Ware) | \quad \exists a : AUF(r.Ware = a.Ware) \vee$
$\exists l : LIEF(r.Ware = l.Ware)\}$

Diese Anfrage kann in SQL nicht ohne `union` formuliert werden.

Generelle Anforderungen (τ, τ_1 und τ_2 sind Terme und φ, φ_1 and φ_2 Formeln):

- Eine gegebene SQL Anfrage 'SELECT τ_1, ..., τ_n FROM R_1 s_1, ..., R_m s_mWHERE φ' und alle Unteranfragen sind vollständig mit expliziten Tupelvariablen qualifiziert: Falls ein Attribut A_j in einem Resultatsterm τ_i oder in einer Formel φ auftaucht, ist dieses Auftreten von der Form $s_i.A_j$, wobei s_i in der FROM Klausel (oder im Falle einer Unteranfrage vielleicht im FROM-Teil einer äußeren Anfrage) auftaucht und A_j ein Attribut einer entsprechenden Relation R_i ist.

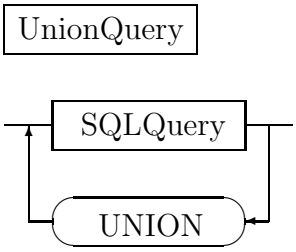


Abbildung 1: Syntax von UnionQuery.

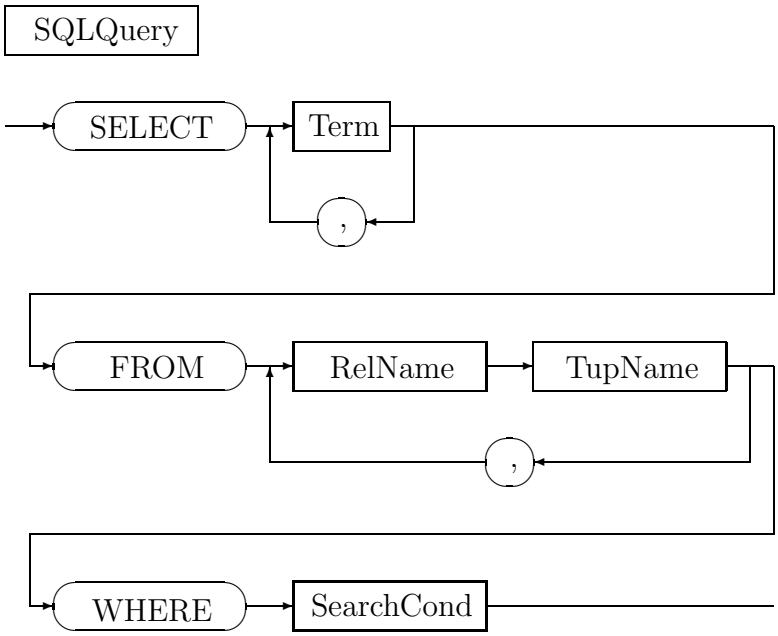


Abbildung 2: Syntax von SQLQuery.

- Die Namen von Tupelvariablen sind eindeutig ($i \neq j \Rightarrow s_i \neq s_j$) und nur einmal deklariert.
- Die Resultatsterme und auch die Formel φ verwenden nur deklarierte Tupelvariable.
- Die Resultatsterme τ_i können auch Konstanten sein.
- Bei Vergleichen wie '$\tau_1 = \tau_2$' oder 'τ_1 = ANY (SELECT τ_2 FROM ... WHERE ...)' haben τ_1 and τ_2 den gleichen Datentyp.
- Für UNION-Ausdrücke nehmen wir an, daß τ_i und τ_i' den gleichen Datentyp haben (i∈1..n).

sql2tc⟦ SELECT τ_1, ..., τ_n FROM R_1 s_1, ..., R_m s_m 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 ⟧) }

sql2tc⟦ (NOT φ) ⟧ := (¬ **sql2tc**⟦ φ ⟧)

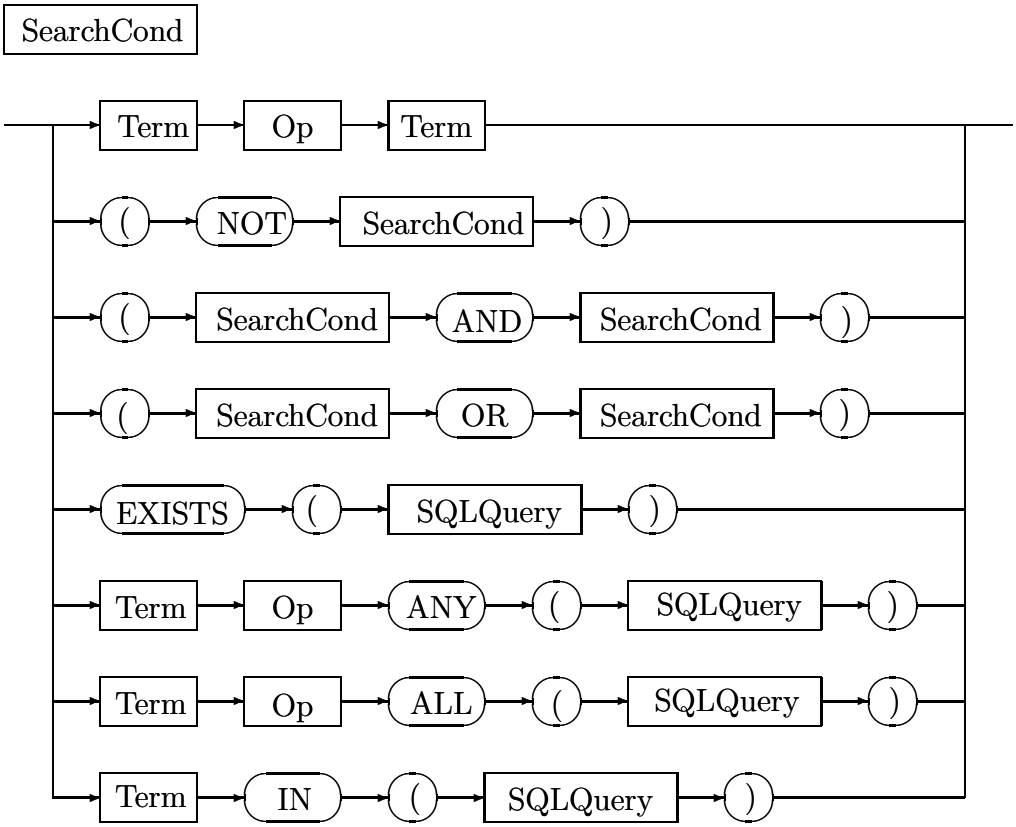


Abbildung 3: Syntax von SearchCond.

sql2tc⟦ (φ_1 AND φ_2) ⟧ := (**sql2tc**⟦ φ_1 ⟧ ∧ **sql2tc**⟦ φ_2 ⟧)

sql2tc⟦ (φ_1 OR φ_2) ⟧ := (**sql2tc**⟦ φ_1 ⟧ ∨ **sql2tc**⟦ φ_2 ⟧)

sql2tc⟦ $\tau_1 \omega \tau_2$ ⟧ := $\tau_1 \omega \tau_2$

sql2tc⟦ τ ω ALL (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 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) ⟧ :=
(∃ 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 ω s_i.A)

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) |
($\exists$ s_1:R_1, ..., s_m:R_m) (r.$\mathsf{Res}_1 = \tau_1 \wedge$ ... $\wedge$ r.$\mathsf{Res}_n = \tau_n \wedge$ **sql2tc**$[\![\varphi]\!]$) $\vee$
($\exists$ s_1':R_1', ..., s_k':R_k') (r.$\mathsf{Res}_1 = \tau_1$'$\wedge$ ... $\wedge$ r.$\mathsf{Res}_n = \tau_n$' $\wedge$ **sql2tc**$[\![\varphi$' $]\!]$) $\}$

Die UNION-Regel kann auf den Fall mit mehr als 2 Operanden verallgemeinert werden.

Beispiel

$$
\begin{array}{l}
select\ * \\
from\ LIEF\ L \\
where\ Preis <= all\ (\ select\ Preis \\
\qquad\qquad\qquad\qquad from\ LIEF \\
\qquad\qquad\qquad\qquad where\ Ware = L.Ware\ )
\end{array}
$$

$$
\begin{array}{l}
\mathbf{sql2tc}[\![\ select\ l1.LName, l1.LAdr, l1.Ware, l1.Preis \\
\qquad\quad from\ LIEF\ l1 \\
\qquad\quad where\ l1.Preis <= all\ (\ select\ l2.Preis \\
\qquad\qquad\qquad\qquad\qquad\qquad from\ LIEF\ l2 \\
\qquad\qquad\qquad\qquad\qquad\qquad where\ l2.Ware = l1.Ware)]\!]
\end{array}
$$

$$
\begin{array}{l}
\{r : (Res_1, Res_2, Res_3, Res_4) | \\
\quad \exists l1 : LIEF(r.Res_1 = l1.LName \wedge r.Res_2 = l1.LAdr \wedge r.Res_3 = l1.Ware \wedge r.Res_4 = l1.Preis \wedge \\
\quad \mathbf{sql2tc}[\![l1.Preis <= all\ (\ select\ l2.Preis \\
\qquad\qquad\qquad\qquad\qquad\qquad from\ LIEF\ l2 \\
\qquad\qquad\qquad\qquad\qquad\qquad where\ l2.Ware = l1.Ware)]\!])\}
\end{array}
$$

$$
\begin{array}{l}
\{r : (Res_1, Res_2, Res_3, Res_4) | \\
\quad \exists l1 : LIEF(r.Res_1 = l1.LName \wedge r.Res_2 = l1.LAdr \wedge r.Res_3 = l1.Ware \wedge r.Res_4 = l1.Preis \wedge \\
\qquad \forall l2 : LIEF(l2.Ware = l1.Ware \Rightarrow l1.Preis <= l2.Preis))\}
\end{array}
$$

Theorem:

(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 $\tau_1 \omega \tau_2$)

(e) τ_1 ω ALL (SELECT τ_2 FROM ρ WHERE φ) $\Leftrightarrow$
NOT EXISTS (SELECT $\tau(\rho)$ FROM ρ WHERE (φ) AND $\tau_1 \overline{\omega} \tau_2$)

(f) EXISTS (SELECT τ FROM ρ WHERE φ) $\Leftrightarrow$ EXISTS (SELECT * FROM ρ WHERE φ)

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

Beweis: Es soll $\rho_i \equiv$ s_1:R_1, ..., s_{i-1}:R_{i-1}, s_{i+1}:R_{i+1}, ..., s_m:R_m gelten.

zu (a)

$\mathbf{sql2tc}[\![$ τ_1 IN (SELECT τ_2 FROM ρ WHERE φ) $]\!] \Leftrightarrow$
$(\ \exists\ s_i{:}R_i\ )\ (\ (\ (\ \exists \rho_i\ )\ \mathbf{sql2tc}[\![\varphi]\!]\ )\ \wedge\ \tau_1 = \tau_2) \Leftrightarrow$
$\mathbf{sql2tc}[\![$ $\tau_1 =$ ANY (SELECT τ_2 FROM ρ WHERE φ) $]\!]$

zu (b)

$\mathbf{sql2tc}[\![$ NOT ($\tau_1\ \omega$ ALL (SELECT τ_2 FROM ρ WHERE φ)) $]\!] \Leftrightarrow$
$\neg\ (\ (\ \forall\ s_i{:}R_i\ )\ (\ (\ \exists \rho_i\ )\ \mathbf{sql2tc}[\![\varphi]\!]\ )\ \Rightarrow\ \tau_1 \omega \tau_2\ ) \Leftrightarrow$
$\neg\ (\ (\ \forall\ s_i{:}R_i\ )\ (\ (\ \forall \rho_i\ )\ \neg\ \mathbf{sql2tc}[\![\varphi]\!]\ )\ \vee\ \tau_1 \omega \tau_2\ ) \Leftrightarrow$
$\neg\ (\ (\ \forall\ s_i{:}R_i\ )\ (\ \forall \rho_i\ )\ (\ \neg\ \mathbf{sql2tc}[\![\varphi]\!] \vee \tau_1 \omega \tau_2\ )\ ) \Leftrightarrow$
$(\ \exists\ s_i{:}R_i\ )\ (\ \exists \rho_i\ )\ (\ \mathbf{sql2tc}[\![\varphi]\!] \wedge \tau_1 \overline{\omega} \tau_2\ ) \Leftrightarrow$
$\mathbf{sql2tc}[\![$ $\tau_1\ \overline{\omega}$ ANY (SELECT τ_2 FROM ρ WHERE φ) $]\!]$

(c), (d), (e) und (f) analog. q.e.d.

Beispiel: Anwendung $\mathbf{sql2ra}[\![...]\!]$

- KUNDE(KNr,KName), BEST(KNr,Ware)
- Beispiel 1

  $\mathbf{sql2ra}[\![select\ \ k.KName$
  $from\ \ KUNDE\ k, BEST\ b$
  $where\ k.KNr = b.KNr\ and\ b.Ware = 42]\!] =$

  $\pi_{k.KName}(\sigma_{k.KNr=b.KNr}(K \times B) \cap \sigma_{b.Ware=42}(K \times B))$

  $K \times B =$

  $\delta_{k.KNr \leftarrow KNr, k.KName \leftarrow KName}(KUNDE) \times \delta_{b.KNr \leftarrow KNr, b.Ware \leftarrow Ware}(BEST)$
- Beispiel 2

  $\mathbf{sql2ra}[\![select\ \ k.KName$
  $from\ \ KUNDE\ k$
  $where\ not\ \underbrace{exists\ (select\ *\ from\ BEST\ b\ where\ k.KNr = b.KNr)}_{EXISTS'}]\!] =_{(Regel\ SELECT)}$

  $\pi_{k.KName}(\mathbf{sql2ra}[\![not\ EXISTS']\!](\ \underbrace{rename[\![KUNDE\ k]\!]}_{KUNDE' = \delta_{k.KNr \leftarrow KNr, k.KName \leftarrow KName}(KUNDE)}\ )) =_{(Regel\ NOT)}$

  $\pi_{k.KName}(\mathbf{sql2ra\text{-}}[\![EXISTS']\!](KUNDE')) =_{(Regel\ \mathbf{sql2ra\text{-}})}$

  $\pi_{k.KName}(KUNDE' -$
  $\pi_{k.KNr,k.KName}(\mathbf{sql2ra}[\![EXISTS']\!](KUNDE'))) =_{(Regel\ EXISTS)}$

  $\pi_{k.KName}(KUNDE' -$
  $\pi_{k.KNr,k.KName}($
  $\mathbf{sql2ra}[\![k.KNr = b.KNr]\!](KUNDE'\ \times\ \underbrace{rename[\![BEST\ b]\!]}_{BEST' = \delta_{b.KNr \leftarrow KNr, b.Ware \leftarrow Ware}(BEST)}\ ))) =_{(Regel\ \tau_1\ \omega\ \tau_2)}$

  $\pi_{k.KName}(KUNDE' -$
  $\pi_{k.KNr,k.KName}(\sigma_{k.KNr=b.KNr}(KUNDE' \times BEST')))$

SQL-Änderungen

- Füge den Kunden UniHB ein

```
insert into KUNDE
values ('UniHB', '...28334 Bremen...', 0)
```

Semantik $insert(KUNDE, (`UniHB', `...28334\ Bremen...', 0))$

- Füge alle Lieferanten als Kunden ein

```
insert into KUNDE (select LName, LAdr, 0
                   from   LIEF )
```

$\widehat{=}$

$insert(KUNDE, r)$ für $r \in$

$\{r : (LName, LAdr, Kto) | \exists l : LIEF$
$(r.LName = l.LName \wedge r.LAdr = l.LAdr \wedge r.Kto = 0)\}$

- Lösche alle Aufträge von Kunden mit negativem Kontostand

```
delete from AUF
where       KName in (select KName from KUNDE where Kto<0)
```

$\widehat{=}$

$delete(AUF, a)$ für alle
$\{a : AUF | \exists k : KUNDE(a.KName = k.KName \wedge k.Kto < 0)\}$

- Halbiere alle Auftragsmengen von Weiss

```
update AUF
set    Menge = Menge * 0.5
where  KName = 'Weiss'
```

$\widehat{=}$

$change(AUF, a, a')$ wobei $a' = a$ außer $a'.Menge = a.Menge * 0.5$ für
$\{a : AUF | a.KName = `Weiss'\}$

Weitere Sprachmittel im SQL-Kern

- Tabellendefinition

```
create table KUNDE ( KName varchar(20) not null,
                     KAdr  varchar(50),
                     Kto   decimal(7,2) )
```

```
 KUNDE | KName   KAdr Kto
-------+-----------------
       | Grau    HB   100
       | Weiss   HB   null
       | Schwarz null 300
```

[`not null` bedeutet, daß keine Nullwerte erlaubt sind]

`integer` $\rightarrow |\, integer \,| = \mathbb{Z} \cup \{\perp\}$

`integer not null` $\rightarrow |\, integer \,| = \mathbb{Z}$

`varchar(50)` kennzeichnet einen String variabler Länge mit maximaler Länge 50; `char(3)` String der festen Länge 3; `decimal(7,2)` Dezimalzahl mit insgesamt 7 Ziffern davon 2 nach dem Komma (also 5 Ziffern vor dem Komma, 2 Ziffern danach)

[null: weissNicht, keineAngabe, unbekannt, nichtGesetzt, trifftNichtZu, undefiniert, ...]

```
create unique index on KUNDE(KName)
create index on KUNDE(KAdr,Kto)
```

Index $\widehat{=}$ Speicherungsstruktur für effizienten Zugriff

KUNDE	Tupelidentifier	KName	KAdr	Kto
	t1	Blau	HB	100
	t2	Gelb	HB	100
	t3	Rot	HH	300

Index(KName)	Indexwert	referenzierte Tupelmenge
	(Blau)	{t1}
	(Gelb)	{t2}
	(Rot)	{t3}

Index(KAdr,Kto)	Indexwert	referenzierte Tupelmenge
	(HB,100)	{t1,t2}
	(HH,300)	{t3}

Bei Angabe `unique` muss referenzierte Tupelmenge einelementig sein

`create table` nach SQL'89 - Level 2 mit IEF

```
create table 'Relation'
  ( 'Attributliste'
      [, primary key ('Attributnamenliste')
        [, foreign key ('Attributnamenliste')
           references 'Relation'('Attributnamenliste') ] ] )
```

Attributmenge nach `foreign key` muss Schlüssel in der nach `references` bezeichneten Relation sein

KUNDE(KName, KAdr, Kto)

AUF[TRAG](KName, Ware, Menge)

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

```
create table AUF
  ( KName varchar(20),
    Ware varchar(20),
    Menge integer,
    primary key (KName,Ware),
    foreign key (KName) references KUNDE(KName) )
```

- Anfragen mit Termen

```
select Ware, Menge*100*Preis
from   AUF, LIEF
where  AUF.Ware=LIEF.Ware and
       Menge*Preis>=1000

Kurz: select Ware, Menge*100*Preis from ...
```

- Abfragen von Nullwerte

```
select *
from   LIEF
where  Preis is null

Kurz: select ... where Preis is null
```

- Ausgabesortierung

```
select   *
from     AUF
order by Ware, KName

Kurz: select ... order by Ware
```

SQL-Versionen

SQL-89 : Level-1, Level-2, IEF (Integrity Enhancement Feature); 150 Seiten

SQL-92 (SQL2): Entry-Level, Intermediate-Level, Full-Level

SQL-99 (SQL3): Core SQL Support, Enhanced SQL Support; 2100 Seiten

... SQL4 ...

Hier nun: Delta von SQL-92 zu SQL-89

INNER JOINS (statt Keyword 'join' auch 'inner join' bzw. 'inner natural join')

```
select S                    = select S
from   R1 join R2 on A1=A2    from   R1, R2
where  P                       where  A1=A2 and P

select S                    = select S
from   R1 join R2 using (A)   from   R1, R2
where  P                       where  R1.A=R2.A and P

select S                    = select S
from   R1 natural join R2     from   R1, R2
where  P                       where  R1.A1=R2.A1 and ... and
                                      R1.An=R2.An and P

[auch f"ur 'select * ...' gilt Gleichheit; d.h. natural join
vollzieht *keine* Projektion; siehe 'SQL-99 Complete, Really', S. 588]

select S                    = select S
from   R1 cross join R2       from   R1, R2
where  P                       where  P
```

OUTER JOINS (null-Werten für unvollständige Tupel)

```
R1 | A B   R2 | B C
---+-----  ----+-----
   | 1 2      | 3 4
   | 2 3      | 4 5

R1 inner natural join R2 | A B C
-------------------------+-------
                         | 2 3 4

R1 natural full outer join R2 | A    B C      auch full outer natural join
------------------------------+-------------
                              | 1    2 null
                              | 2    3 4
                              | null 4 5

R1 natural left outer join  R2 | A B C
-------------------------------+-------
                               | 1 2 null
```

```
                                | 2 3 4

R1 natural right outer join R2 | A    B C
-------------------------------+----------
                                | 2    3 4
                                | null 4 5
```

natural left outer join : alle Tupel aus dem linken Operanden übernehmen

natural right outer join: alle Tupel aus dem rechten Operanden übernehmen

STATT natural join AUCH join on ... ODER join using ...

Eigenschaften

- $\pi_{R1}(R1\ natural\ inner\ join\ R2) \subseteq R1$

- $\pi_{R1}(R1\ natural\ full\ outer\ join\ R2) = R1$

```
select A, R1.B, C
from   R1 natural outer join R2
=
(select A, R1.B, C
 from   R1, R2
 where  R1.B=R2.B) -- R1 natural join R2
union
(select A, B, null
 from   R1
 where  not exists (select *
                    from   R2
                    where  R2.B=R1.B)) -- left outer join tuples
union
(select null, B, C
 from   R2
 where  not exists (select *
                    from   R1
                    where  R1.B=R2.B)) -- right outer join tuples
```

Mengenoperationen

union (bereits in SQL-89), intersect, except

```
(select ... from ... where ...)
{ union | intersect | except } [ all | distinct ]
(select ... from ... where ...)

[ corresponding [by ('Attributnamenliste')] ]
```

Jeder SFW-Block in from-Klausel einsetzbar: laut 'Heuer/Saake, 1. korrigierter Nachdruck, S. 267' und 'Heuer/Saake, 2. aktualisierte und erweiterte Auflage, S. 344'; interessant für Aggregationsfunktionen und Gruppierung (siehe 'Weitere Sprachkonzepte')

Tupelvariablenbenennung für Zwischenergebnisse aus from-Klausel

```
select V.A
from   (...) as V
```

Ergebnisspaltenbenennung

```
select R.A as B
from   R
```

Beispiele

```
select KJA.KName
from   (KUNDE natural join AUFTRAG) as KJA

select count(*) as Anzahl -- Aggregationsfunktionen s.u.
from   R

[ NICHT VL ...

select NA.Name, NA.Adr -- unklar ob dies funktioniert
from   ((select KName as Name, KAdr as Adr from KUNDE)
        intersect
        (select LName as Name, LAdr as Adr from LIEF)) as NA

...]
```

Tupelbildung in where-Klausel

```
where (select ... from ... where ... ) = (C1, C2, ..., Cn)
```

'neue' SQL-Sprachmittel (abgeleitet aus vorhandenen Sprachmitteln)

- F1 implies F2 := not F1 or F2
- forall (select * from R where F)
  :=
  not exists (select * from R where not F)

- existsExactlyOne (select * from R where F)

  :=

```
exists (select *
        from   R r1
        where  F and
               not exists (select * from R r2 where F and r1<>r2))
```

  Alternative Bezeichnung: existsUnique oder existsOne oder exists1?

3.5 QUEL

1. Welche Lieferanten liefern von Weiss in Auftrag gegebene Waren?

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

2. Welche Lieferanten liefern Milch oder Mehl?

   [Wie SQL liefert auch QUEL im allgemeinen Duplikate im Ergebnis. Diese Duplikate können mit `retrieve unique` eliminiert werden. Für diesen Abschnitt nehmen wir eine solche Duplikatbeseitigung für alle Anweisungen als implizit gegeben an.]

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

3. Welche Lieferanten liefern mindestens eine Ware die auch Grau liefert?

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

   [QUEL-Änderungsanweisungen erklären]

4. 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
```

5. (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
```

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

   In QUEL nicht mit einer Anfrage formulierbar. Folgende Anfrage liefert ein falsches Ergebnis.

```
range of l1, l2 is LIEF
rerieve (l1.Lname,l1.LAdr,l1.Ware,l1.Preis)
  where l1.Preis<L2.Preis and l1.Ware=l2.Ware
```

   Datenbankzustand

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

   Ergebnis

	LName	LAdr	Ware	Preis
	Date	...	DBS	40
	Elmasri	...	DBS	35

   Die Anfrage liefert alle Lieferanten-Tupel, die ihre Ware günstiger als irgendein Lieferant liefern, also alle Lieferanten bis auf die teuersten.

   Korrekte Lösung

```
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)

7. 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'
```

8. 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
```

3.6 QBE

1. 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	

äquivalenter Tupelkalkülausdruck

$\{n, w | \exists a, p, m$
$(LIEF(n, a, w, p) \wedge AUF(`Weiss', w, m) \wedge p < 100)\}$

$\{n, w | \exists p$
$(LIEF(n, *, w, p) \wedge AUF(`Weiss', w, *) \wedge p < 100)\}$

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

AUF	KName	Ware	Menge
	Weiss	_m	

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

R	A	B
P.		_b

S	B	C
	_b	P.

```
select A, R.B, C
from   R, S
where  R.B=A.B
```

3. Welche Lieferanten liefern Milch oder Mehl?

LIEF	LName	LAdr	Ware
	P.	P.	_mehl

CONDITIONS
_mehl=Milch or _mehl=Mehl

$\{n, a | \exists w, p$
$(LIEF(n, a, w, p) \wedge (w = `Milch' \vee w = `Mehl'))\}$

$\{n, a | \exists w$
$(LIEF(n, a, w, *) \wedge (w = `Milch' \vee w = `Mehl'))\}$

4. Welche Lieferanten liefern mindestens eine Ware die auch Grau liefert?

LIEF	LName	Ware
	P.	_mehl
	Grau	_mehl

$\{n | \exists a, w, p(LIEF(n, a, w, p) \wedge$
$\exists a', p' LIEF(`Grau', a', w, p'))\}$

$\{n | \exists w(LIEF(n, *, w, *) \wedge LIEF(`Grau', *, w, *))\}$

5. 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

$\{n, m, w, k | \exists a(KUNDE(n, a, k) \wedge AUF(n, w, m))\}$

$\{n, m, w, k | (KUNDE(n, *, k) \wedge AUF(n, w, m))\}$

6. Aufträge mit maximaler Menge

AUF	KName	Ware	Menge
P.			_42
$\neg$			$>$_42

$\{n, w, m | AUF(n, w, m) \wedge$
$\neg[\exists n', w', m'(AUF(n', w', m') \wedge m' > m)]\}$

$\{n, w, m | AUF(n, w, m) \wedge$
$[\forall n', w', m'(AUF(n', w', m') \Rightarrow m' \leq m)]\}$

$\{n, w, m | AUF(n, w, m) \wedge$
$[\forall m'(AUF(*, *, m') \Rightarrow m' \leq m)]\}$

7. Welches sind die Kunden ohne Auftrag?

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

Wie P. auch D. und I.

- 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.	

$\{k|\exists a, s(KUNDE(k,a,s) \wedge \forall k', w, m(AUF(k', w, m) \Rightarrow k \neq k'))\}$

$\{k|\exists a, s(KUNDE(k,a,s) \wedge \forall k', w, m(\neg AUF(k', w, m) \vee k \neq k'))\}$

$\{k|\exists a, s(KUNDE(k,a,s) \wedge \forall k', w, m \neg(AUF(k', w, m) \wedge k = k'))\}$

$\{k|\exists a, s(KUNDE(k,a,s) \wedge \neg\exists k', w, m(AUF(k', w, m) \wedge k = k'))\}$

$\{k|\exists a, s(KUNDE(k,a,s) \wedge \neg\exists w, m(AUF(k, w, m)))\}$

$\{k|KUNDE(k,*,*) \wedge \neg AUF(k,*,*)\}$

Achtung: $\neg AUF(k,*,*) \Leftrightarrow \neg\exists w, mAUF(k,w,m)$ und es gilt nicht
$\neg AUF(k,*,*) \Leftrightarrow \exists w, m \neg AUF(k,w,m)$ da
$\neg\exists w, mAUF(k,w,m) \Leftrightarrow \forall w, m \neg(AUF(k,w,m))$

[durch Bedingung auf Folie 3-23 (Ausgabevariablen ...) gewährleistet:
Generalvoraussetzung an Variablen

- Jede explizite oder Ausgabevariable muss in mindestens einer positiven Zeile vorkommen
- fehlerhaftes Beispiel

KUNDE	KName
¬	P. _weiss

]

Darstellung der Relationen-Operationen in QBE für gegebene Relationen 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

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

CONDITIONS
φ

Beispiel: QBE nicht relational vollständig

$R(A), S(B)$

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

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

```
Zustand 1: R = {(10)} S={(10)}; Ergebnis: {}
```

```
Zustand 2: R = {(10),(11),(12)} S={(10)}; Ergebnis: {11}
```

$$\{r1 : (A) \mid \; R(r1) \wedge \\ \forall s : S \; \exists r2 : R \; [s.B < r1.A \wedge s.B < r2.A \wedge r1.A < r2.A]\}$$

$$\{r1 : (A) \mid \; R(r1) \wedge \\ \neg\exists s : S \; \neg\exists r2 : R \; [s.B < r1.A \wedge s.B < r2.A \wedge r1.A < r2.A]\}$$

```
select *
from   R r1
where  not exists(select *
                  from   S s
                  where  not exists(select *
                                    from   R r2
                                    where  s.B<r1.A and s.B<r2.A and r1.A<r2.A))
```

QBE-Versuch 1

R	A
	P. _a1
¬	_a2

S	B
¬	_b

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

Variablen in CONDITIONS erfüllen nicht geforderte Bedingungen

QBE-Versuch 2

R	A
	P. _a1
¬	'_a2'>_a1

S	B
¬	'_b'<_a1

CONDITIONS
'_b'<'_a2'

Variablen '_a2' und '_b' nicht explizit ansprechbar

3.7 Vergleich von RA, BK, TK, SQL, QUEL, QBE

Kriterien: Einfachheit, Verständlichkeit, Bequemlichkeit, Notationsaufwand, implizite VERSUS explizite Konstrukte, Operationalisierbarkeit, Anschaulichkeit

Ausdruckstärke: $RA = BK_{safe} = TK_{safe} = SQL_{kernel} \supset QBE_{kernel} \supset QUEL_{kernel}$

$\pi_{AName}(\sigma_{SName=`Zimmermann'}(AUT * AUS * STUD))$

$\{aname \mid \exists doknr, matnr$
$(AUT(doknr, aname) \wedge$
$\exists datum(AUS(doknr, matnr, datum)) \wedge$
$\exists fach, sem(STUD(matnr, `Zimmermann', fach, sem)))\}$

$\{r : (AName) \mid \exists aut : AUT, aus : AUS, s : STUD$
$(r.AName = aut.AName \wedge aut.Doknr = aus.Doknr \wedge$
$aus.Matnr = s.Matnr \wedge s.SName = `Zimmermann')\}$

```
select AName
from   AUT
where  exists (select *
               from   AUS, STUD
               where  AUT.Doknr=AUS.Doknr and
                      AUS.Matnr=STUD.Matnr and
                      SName='Zimmermann')
```

```
range of aut is AUT, aus is AUS, s is STUD
retrieve (aut.AName)
  where aut.Doknr=aus.Doknr and aus.Matnr=s.Matnr and
        s.SName='Zimmermann'
```

AUT	AName	Doknr
	P.	_d

AUS	Doknr	Matnr
	_d	_m

AUS	Matnr	SName
	_m	Zimmermann

3.8 Weitere Sprachkonzepte

3.8.1 Aggregationsfunktionen und Gruppierung

- Motivierendes Beispiel

STUDENT(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)

folgende Anfragen gehen teilweise von dieser Schlüsselstruktur aus

1. Anzahl von Kunden

   ```
   select count(*) from KUNDE
   ```

   ```
   select count(KName) from KUNDE
   ```

2. 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
	'Weiss'	'Zucker'	500

AUF by Ware	KName	Ware	Menge
	'Schwarz'	'Mehl'	100
	'Weiss'	'Mehl'	200
	'Schwarz'	'Salz'	300
	'Weiss'	'Salz'	300
	'Weiss'	'Zucker'	500

Ware,sum(Menge)	Ware	sum(Menge)
	'Mehl'	300
	'Salz'	600
	'Zucker'	500

3. 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

4. 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'
```

5. Durchschnitt aller Gesamtauftragsmengen von Waren

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

6. Aufträge mit maximaler Menge (über eine Unteranfrage)

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

7. Aufträge mit maximaler Menge (mit Aggregationsfunktion)

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

8. 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
```

9. 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 'AggFunction'(InterResult)
from   (select 'AggFunction'('Attribute') as InterResult
        from   'Relation')

Maximum der Durchschnitte in ESD(Employee,Salary,Department)

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(<u>Matnr</u>,SName,Fach,Sem)

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

  $\{\!\{\ Sem(s)\ \mid\ s : STUDENT\ \}\!\}$
- $\{\ Term\ \mid\ Formel\ \}$ ergibt Menge

  $\{\!\{\ Term\ \mid\ Deklaration\ \wedge\ Formel\ \}\!\}$ ergibt Multimenge (Bag)
- Multimengen lassen sich als Mengen zusammen mit einer Zählfunktion auffassen: $\{a1, a1, a2\} \equiv \{(a1 \mapsto 2), (a2 \mapsto 1)\}$

  Allgemein $Bag \equiv (Set, occurrences : Set \rightarrow \mathbb{N})$
- 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

1. Anzahl von Kunden

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

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

2. Summe der Auftragsmengen je Ware

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

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

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

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

3. Waren die von mehr als einem Lieferanten angeboten werden

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

4. Anzahl der verschiedenen Lieferanten die Mehl oder Milch anbieten

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

5. Durchschnitt aller Gesamtauftragsmengen von Waren

   $$AVG(\{ \quad SUM \{ Menge(a') | a' : AUF \wedge Ware(a') = w \} \mid \\ w : BTS \{ Ware(a) | a : AUF \} \})$$

6. Aufträge mit maximaler Menge (über eine Unteranfrage)

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

7. Aufträge mit maximaler Menge (mit Aggregationsfunktion)

   $$\{ \quad KName(a), Ware(a) | \\ a : AUF \wedge \\ Menge(a) = MAX \{ Menge(a) | a : AUF \} \}$$

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

$$\{\ Ware(l),\ MIN\ \{\ Preis(l')|l' : LIEF \wedge Ware(l') = Ware(l)\ \}\ ,\ AVG\ \{\ Preis(l')|l' : LIEF \wedge Ware(l') = Ware(l)\ \}\ ,\ MAX\ \{\ Preis(l')|l' : LIEF \wedge Ware(l') = Ware(l)\ \}\ |\ l : LIEF\ \}_{\ sort(1)}$$

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

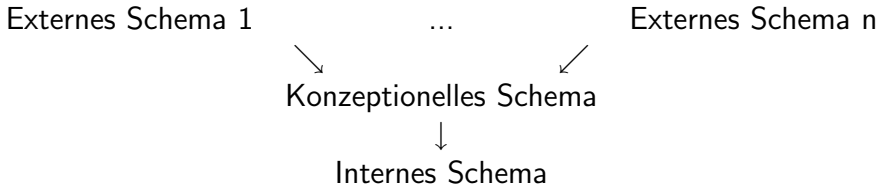
$$BTS\ \{\ Ware(l)|\ l : LIEF \wedge\ (MAX\ \{\ Preis(l')|l' : LIEF \wedge Ware(l') = Ware(l)\ \}\ )\ <=\ (MIN\ \{\ Preis(l')|l' : LIEF \wedge Ware(l') = Ware(l)\ \}\ ) * 1.05\ \}$$

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

3.8.2 Sichten

- Erinnerung: Drei-Ebenen-Architektur



- Beispiel

  KUNDE(<u>KName</u>, KAdr, Kto)

  AUF[TRAG](<u>KName</u>, <u>Ware</u>, Menge)

  LIEF[ERANT](<u>LName</u>, LAdr, <u>Ware</u>, 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 $select\ \tau\ from\ \beta\ where\ \varphi$

  an Sicht $create\ view\ \beta\ as\ select\ \tau'\ from\ \rho\ where\ \varphi'$

  wird $select\ \overline{\tau}\ from\ \rho\ where\ \varphi'\ \wedge \overline{\varphi}$

  wobei $\overline{\tau}$ und $\overline{\varphi}$ 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 = Outer natural 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 outer natural join AUS )
                     outer natural join BUCH )
                     outer natural join AUT )
                     outer natural 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

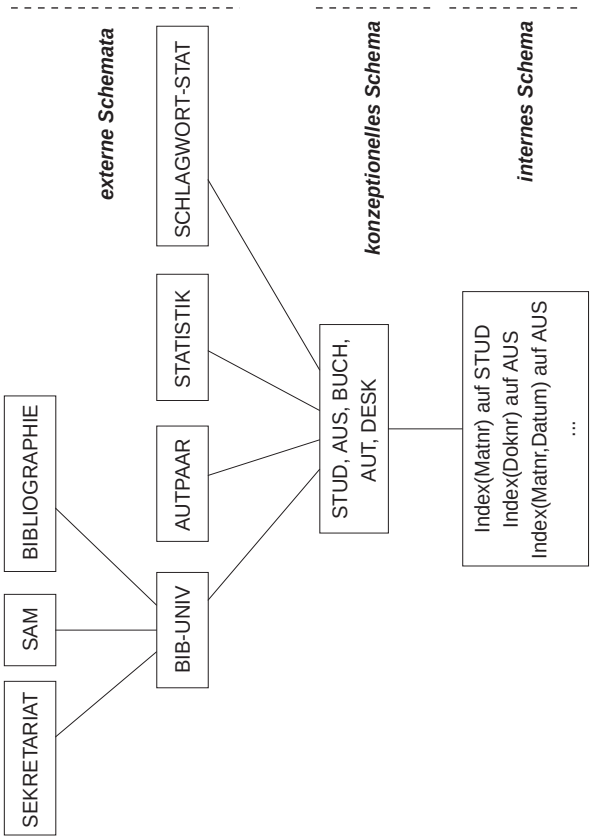
- Sicht Bibliographie

```
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
```



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

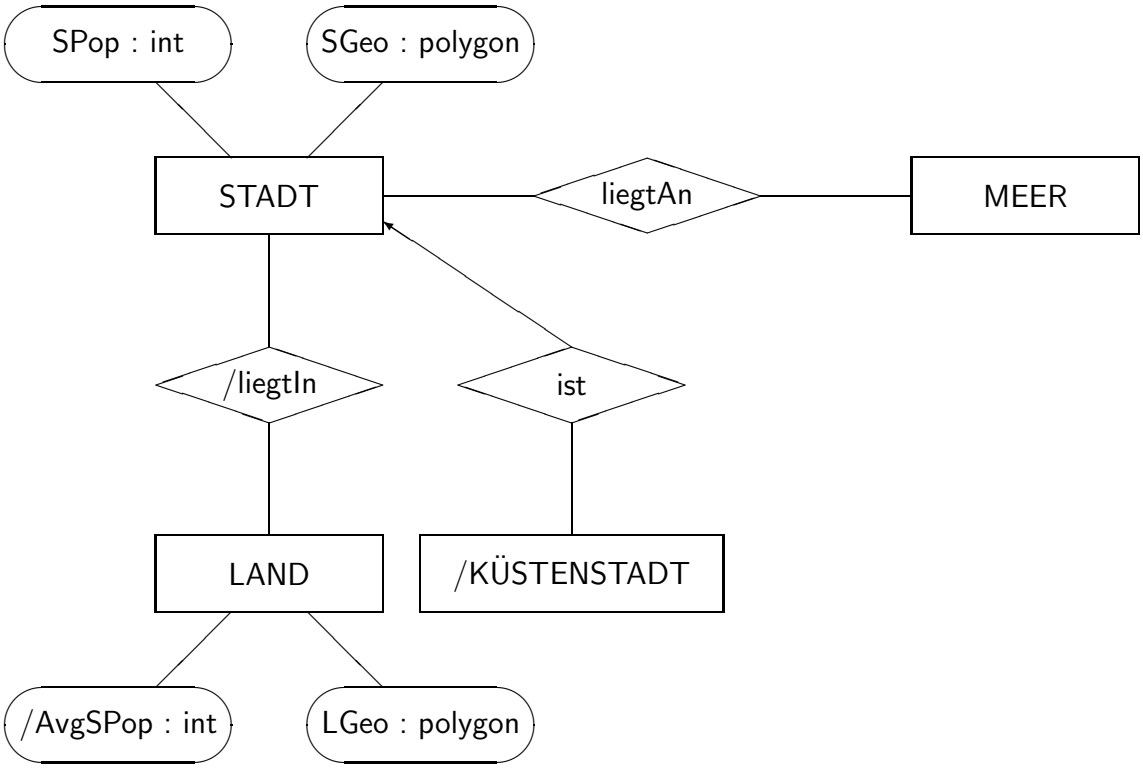
```
VOR(X,Y) :- ELT(X,Y).
VOR(X,Y) :- ELT(X,Z), VOR(Z,Y).
```

- 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

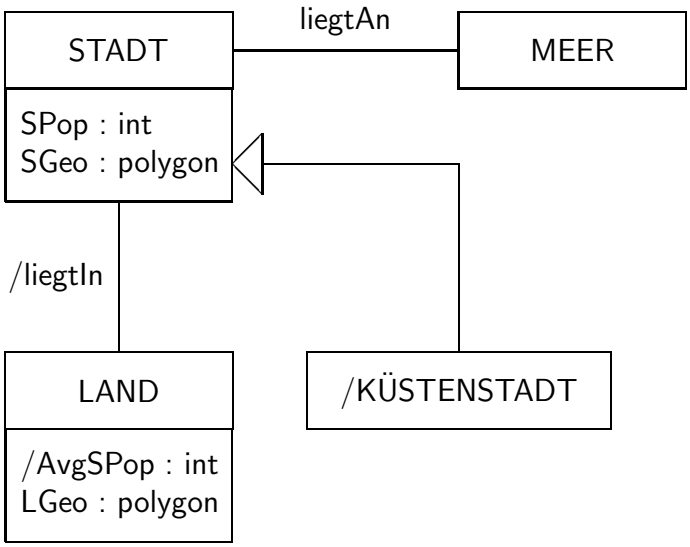
Vorausgesetzt: komplexer Datentyp polygon; etwa: polygon = List(point);
point = Record(int,int); für polygon sind auch entsprechende Operationen
definiert; etwa: polygonCuts: polygon × polygon → bool



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))


STADT
liegtAn
MEER
SPop : int
SGeo : polygon
/liegtIn
LAND
/KÜSTENSTADT
/AvgSPop : int
LGeo : polygon