Kapitel 2: Relationenmodell

2.1 Grundbegriffe

Das Relationenmodell beruht auf Arbeiten von E.F. Codd und anderen um 1970.

Grundprinzip:
Alle Daten werden in Form von mathematischen Relationen (Teilmengen eines kartesischen Produkts von Wertemengen) repräsentiert.

Definition:
Gegeben sei eine Menge von Wertebereichen primitiver Datentypen {D1, ..., Dm}, die als "*Domains*" bezeichnet werden.
Eine *Relation* R ist ein Paar R = (s,v) mit
- einem *Schema* s = {A1, ..., An}, das aus einer Menge von *Attributen* (genauer: Attributnamen) besteht und für jedes Attribut Ai einen Domain dom(Ai) $\in$ {D1, ..., Dm} festlegt, und
- einer *Ausprägung* (auch *Wert* genannt) v $\subseteq$ dom(A1) $\times$ dom(A2) $\times$ ... $\times$ dom(An).

Schema und Ausprägung von R werden auch mit *sch(r)* und *val(R)* bezeichnet.

Bemerkungen:
- Die Elemente der Ausprägung einer Relation heißen *Tupel*. Für den Wert eines Tupels t bezüglich eines Attributs Ai schreibt man t.Ai (seltener auch t[Ai] oder Ai(t)).
- Die Tupel einer Relation bilden eine (ungeordnete) Menge. Es gibt also keine Reihenfolge unter den Tupeln, und je zwei Tupel müssen sich in mindestens einem Attribut unterscheiden.
- Die Attribute einer Relation bilden eine (ungeordnete) Menge; es gibt also keine Reihenfolge unter den Attributen.
- Als Domains von Attributen sind nur primitive Wertebereiche zugelassen, d.h. Wertebereiche elementarer Datentypen (im wesentlichen INTEGER, REAL, BOOLEAN), die Wertebereiche von Bereichstypen und Aufzählungstypen sowie der Wertebereich der Zeichenketten (ARRAY OF CHAR).
  Diese Einschränkung wird auch als *1. Normalform* des Relationenmodells bezeichnet.
- Eine relationale Datenbank ist eine Menge von Relationen. Die Menge der betreffenden Relationenschemata wird als Datenbankschema bezeichnet.
  Häufige Schreibweise für Relationenschemata:
  R(A1, ..., An)
  Kunden (KNr, Name, Stadt, Saldo, Rabatt)
  Produkte (PNr, Bez, Gewicht, Preis, Lagerort, Vorrat)
  Bestellungen (BestNr, Monat, Tag, KNr, PNr, Menge, Summe, Status)

Informelle Sprechweise:

Relation	=	Tabelle
Schema	=	Tabellenkopf (Überschrift der Tabelle)
Attribut	=	Spalte der Tabelle (Feld)
Tupel	=	Zeile der Tabelle (engl.: row; Datensatz, Record)

Beispieldatenbank:

Kunden

KNr	Name	Stadt	Saldo	Rabatt
1	Lauer	Merzig	− 1080.00	0.10
2	Schneider	Homburg	− 800.00	0.20
3	Kirsch	Homburg	0.00	0.10
4	Schulz	Merzig	0.00	0.10
5	Becker	Dillingen	0.00	0.05
6	Meier	Saarlouis	− 3800.00	0.05

Produkte

PNr	Bez	Gewicht	Preis	Lagerort	Vorrat
1	Papier	2.000	20.00	Homburg	10000
2	Platte	1.000	2500.00	Saarbrücken	400
3	Drucker	5.000	2000.00	Merzig	200
4	Bildschirm	5.000	3000.00	Merzig	80
5	Disketten	0.500	20.00	Homburg	5000
6	Maus	0.250	100.00	Homburg	200
7	Speicher	0.100	200.00	Saarbrücken	2000

Bestellungen

BestNr	Monat	Tag	KNr	PNr	Menge	Summe	Status
1	7	16	1	1	100	1800.00	bezahlt
2	7	21	1	1	100	1800.00	bezahlt
3	9	30	1	2	4	9000.00	bezahlt
4	9	30	1	3	1	1800.00	bezahlt
5	9	30	1	4	10	27000.00	bezahlt
6	10	15	1	5	50	900.00	bezahlt
7	10	28	1	6	2	180.00	geliefert
8	11	2	1	7	5	900.00	neu
9	10	26	2	1	100	1600.00	bezahlt
10	11	2	2	5	50	800.00	neu
11	9	28	3	5	50	900.00	bezahlt
12	10	28	3	7	10	1800.00	bezahlt
13	4	15	4	1	50	900.00	bezahlt
14	5	31	6	1	200	3800.00	bezahlt
15	6	30	6	7	10	1900.00	geliefert
16	7	31	6	1	100	1900.00	geliefert

Konsequenz der 1. Normalform

Studenten

Name	Fachbereich	...	Systemkenntnisse
Meier	14		{Oracle, C, Unix}
Schmidt	14		{Pascal, Oracle}
Kunz	14		{Oracle}
Müller	2		∅

ist nicht zulässig!

Korrekte Repräsentation im Relationenmodell:

Studenten

Name	Fachbereich	...	Systemkenntnis
Meier	14		Oracle
Meier	14		C
Meier	14		Unix
Schmidt	14		Pascal
Schmidt	14		Oracle
Kunz	14		Oracle
Müller	2		–

oder besser:

Studenten

Name	Fachbereich	...
Meier	14	
Schmidt	14	
Kunz	14	
Müller	2	

Kenntnisse

Name	Systemkenntnis
Meier	Oracle
Meier	C
Meier	Unix
Schmidt	Pascal
Schmidt	Oracle
Kunz	Oracle

Inhärente Integritätsbedingungen des Relationenmodells

Definitionen:

Sei R eine Relation. Eine Attributmenge $K \subseteq sch(R)$ heißt *Schlüsselkandidat*, wenn zu jedem Zeitpunkt für je zwei Tupel $t1, t2 \in val(R)$ gelten muß:

$t1.K = t2.K \Rightarrow t1 = t2$

und wenn es keine echte Teilmenge von K gibt, die diese Eigenschaft hat.
(Dabei bedeutet $t1.K = t2.K$ präzise: $\forall A \in K: t1.A = t2.A$.)
Bemerkung: Jede Relation hat mindestens einen Schlüsselkandidaten.

Ein Attribut einer Relation R, das in mindestens einem Schlüsselkandidaten vorkommt, heißt *Schlüsselattribut*.

Der *Primärschlüssel* (engl.: primary key) einer Relation ist ein Schlüsselkandidat, der explizit ausgewählt wird.

Eine Attributmenge $F \subseteq sch(S)$ einer Relation S ist ein *Fremdschlüssel* (engl.: foreign key) in S, wenn es eine Relation R gibt, in der F Primärschlüssel ist.

Ein Attribut A eines Tupels t hat einen *Nullwert*, wenn der Wert t.A undefiniert oder unbekannt ist. (Gedanklich werden alle Domains um einen solchen Nullwert erweitert.)
Beispiele:
Vorrat für Produkte, die nicht gelagert werden, sondern bei Bedarf extern gekauft werden.
Note von Studenten, die ihre Prüfung noch nicht abgelegt haben.
Informatikkenntnisse von Studenten zu Beginn des Studiums.

Schreibweise:
In einem Relationenschema wird der Primärschlüssel unterstrichen; weitere Schlüsselkandidaten werden oft gestrichelt unterstrichen.

Das Relationenmodell (genauer: eine Implementierung des Relationemodells) garantiert die folgenden Integritätsbedingungen:

Primärschlüsselbedingung ("Entity Integrity"):

Für jede Relation muß ein Primärschlüssel festgelegt werden. Der Primärschlüssel eines Tupels darf nie den Nullwert annehmen.

Beispiel:
Kunden (KNr, Name, Stadt, Saldo, Rabatt)
Produkte (PNr, Bez, Gewicht, Preis, Lagerort, Vorrat)
Bestellungen (BestNr, Monat, Tag, KNr, PNr, Menge, Summe, Status)

Fremdschlüsselbedingung ("Referential Integrity"):

Für jeden Wert eines Fremdschlüssels in einer Relation R muß in den referenzierten Relationen jeweils ein Tupel mit demselben Wert als Primärschlüssel existieren, oder der Wert des Fremdschlüssels muß der Nullwert sein.

Beispiel für Nullwert als Fremdschlüssel:
Produkte (PNr, ..., Lagerort)
Lager (Lagerort, Adresse, Verwalter, ...)

2.2 Relationenalgebra

Die Relationenalgebra bildet ein Fundament für Datenbankanfragesprachen (Query Languages), indem sie Grundoperationen für Relationen definiert.

- Eine Operation hat eine oder mehrere Relationen als Operanden und stets wiederum eine Relation als Ergebnis.
  → Abgeschlossenheit der Relationenalgebra.
- Das Schema eines Operationsresultats kann von den Schemata der Operanden verschieden sein.

Mengenoperationen:

Für zwei Relationen R, S mit $sch(R) = sch(S)$ sind die üblichen Mengenoperationen definiert:

- Vereinigung (Union) $R \cup S$:
  $sch(R \cup S) = sch(R)$
  $val(R \cup S) = \{t \mid t \in val(R) \vee t \in val(S)\}$
- Durchschnitt (Intersection) $R \cap S$:
  $sch(R \cap S) = sch(R)$
  $val(R \cap S) = \{t \mid t \in val(R) \wedge t \in val(S)\}$
- Differenz (Difference) $R - S$:
  $sch(R - S) = sch(R)$
  $val(R - S) = \{t \mid t \in val(R) \wedge t \notin val(S)\}$

Selektion σ (Filterung, Auswahl von Zeilen einer Tabelle):

Sei F eine Boolesche Formel über einfachen Vergleichsbedingungen zwischen zwei Attributen einer Relation oder einem Attribut und einer Konstanten. Das Resultat einer Selektion $\sigma[F](R)$ auf einer Relation R (auch $\sigma_F(R)$ geschrieben) ist wie folgt definiert:

$sch(\sigma[F](R)) = sch(R)$
$val(\sigma[F](R)) = \{t \mid t \in R \wedge F(t)\}$ wobei F(t) bedeutet, daß t die Bedingung F erfüllt.

Die Menge der möglichen Filterformeln F ist wie folgt präzise definiert:

1) Für Attribute A, B von R mit dom(A)=dom(B), Konstanten $c \in dom(A)$ und Vergleichsoperationen $\theta \in \{=, \neq, <, >, \leq, \geq\}$ sind
   $A \theta B$ und $A \theta c$ zulässige Filterbedingungen.
2) Falls F1 und F2 zulässige Filterbedingungen sind, dann sind auch
   $F1 \wedge F2$, $F1 \vee F2$ und $\neg F1$ zulässig.
3) Nur die aufgrund von 1) und 2) erzeugten Filterbedingungen sind zulässig.

Projektion π (Auswahl von Spalten einer Tabelle):

Sei $A \subseteq sch(R)$ eine Teilmenge der Attribute einer Relation R. Das Resultat einer Projektion $\pi[A](R)$ auf der Relation R (auch $\pi_A(R)$ geschrieben) ist wie folgt definiert:

$sch(\pi[A](R)) = A$
$val(\pi[A](R)) = \{t \mid \exists\, r \in val(R): t.A = r.A\}$

Achtung: Die Projektion beinhaltet eine Duplikateliminierung.

Beispiele für die Selektion:

1) Finde alle Homburger Kunden.
→ σ[Stadt='Homburg'] (Kunden)
→ Resultat:

KNr	Name	Stadt	Saldo	Rabatt
2	Schneider	Homburg	− 800.00	0.20
3	Kirsch	Homburg	0.00	0.10

2) Finde alle Homburger Kunden, die einen Rabatt von mindestens 15% haben.
→ σ[Stadt='Homburg' ∧ Rabatt >= 0.15] (Kunden)
→ Resultat:

KNr	Name	Stadt	Saldo	Rabatt
2	Schneider	Homburg	− 800.00	0.20

Beispiele für die Projektion:

3) Gib alle Produktbezeichnungen aus.
→ π[Bez] (Produkte)
→ Resultat:

Bez
Papier
Platte
Drucker
Bildschirm
Disketten
Maus
Speicher

4) Gib alle Lagerorte von Produkten aus.
→ π[Lagerort] (Produkte)
→ Resultat:

Lagerort
Homburg
Saarbrücken
Merzig

(Natural) Join $|\times|$

(Natürlicher Verbund, Verbindung zweier Relationen über gleiche Attributnamen und gleiche Attributwerte der Tupel):

Seien R, S Relationen mit Schemata A=sch(R) und B=sch(S). Das Resultat des Joins $R |\times| S$ ist wie folgt definiert:

$$sch(R|\times|S) = sch(R) \cup sch(S)$$
$$val(R|\times|S) = \{t \mid \exists\, r \in val(R)\ \exists\, s \in val(S): t.A = r.A \wedge t.B = s.B\}$$

Beispiele für den Join:

1) Führe alle Bestellungen zusammen mit den dazugehörigen Produktdaten auf.
→ Bestellungen $|\times|$ Produkt
→ Resultat:

Best Nr	Monat	Tag	KNr	PNr	Menge	Summe	Status	Bez	Gewicht	Preis	Lagerort	Vorrat
1	7	16	1	1	100	1800.00	bezahlt	Papier	2.000	20.00	Homburg	10000
2	7	21	1	1	100	1800.00	bezahlt	Papier	2.000	20.00	Homburg	10000
3	9	30	1	2	4	9000.00	bezahlt	Platte	1.000	2500.00	Saarbr.	400
⋮												

2) $R |\times| S$ mit:

R

R1	J
a	1
b	2
c	2
d	3

S

S1	J
w	2
x	2
y	3
z	4

→ Resultat:

R1	J	S1
b	2	w
b	2	x
c	2	w
c	2	x
d	3	y

Zuweisung:

Seien R und S zwei Relationen mit sch(R)={A1, ..., An} und sch(S)={B1, ..., Bn}, so daß für alle i gilt: dom(Ai)=dom(Bi). Die Zuweisung *R := S* bedeutet, daß sich die Ausprägung von R wie folgt ändert: val(R) = val(S).
Die Zuweisung gehört nicht wirklich zur Relationenalgebra. Sie ist nützlich als Notationshilfe für Zwischenresultate, für Umbenennungen von Attributnamen und um Änderungsoperationen ausdrücken zu können.

Beispiele für komplexere Anfragen:

1) Finden Sie die Namen der Kunden mit negativem Saldo.
   → π[Name] (σ[Saldo<0.0] (Kunden))

2) Finden Sie die Namen der Kunden, die eine unbezahlte Bestellung haben, die vor Anfang Oktober erfolgte.
   → π[Name] (σ[Status<>'bezahlt' ∧ Monat<10] (Bestellungen) | × | Kunden)
   oder mit Zwischenresultaten:
   B := σ[Status<>'bezahlt' ∧ Monat<10] (Bestellungen)
   BK := B | × | Kunden
   π[Name] (BK)

3) Finden Sie die Namen der Homburger Kunden, die seit Anfang September ein Produkt aus Homburg geliefert bekommen haben.
   → π[Name] (σ[Monat >= 9 ∧ Status<>'neu'] (Bestellungen)
   | × | σ[Lagerort='Homburg' (Produkte)
   | × | σ[Stadt='Homburg'] (Kunden))
   oder mit Zwischenresultaten:
   B := σ[Monat >= 9 ∧ Status<>'neu'] (Bestellungen)
   P := σ[Lagerort='Homburg' (Produkte)
   K := σ[Stadt='Homburg'] (Kunden)
   BP := B | × | P
   BPK := BP | × | K
   π[Name] (BPK)

4) Finden Sie die Kundennummern der Kunden, von denen mindestens eine Bestellung registriert ist.
   → π[KNr] (Kunden | × | Bestellungen)

5) Finden Sie die Kundennummern der Kunden, von denen keine Bestellung registriert ist.
   → π[KNr] (Kunden) − π[KNr] (Kunden | × | Bestellungen)
   oder:
   π[KNr] (Kunden) − π[KNr] (Bestellungen)

6) Finden Sie die Namen der Kunden, von denen keine Bestellung registriert ist.
   → π[Name] (Kunden) − π[Name] (Kunden | × | Bestellungen)
   oder:
   π[Name] (Kunden | × | (π[KNr](Kunden) − π[KNr](Bestellungen)

Division $\div$:

Seien R, S Relationen mit Schemata A=sch(R) und B=sch(S), so daß $B \subset A$. Mit Q=A−B sei die Menge der Attribute bezeichnet, die in R vorkommen, nicht aber in S. Das Resultat der Division $R \div S$ ist wie folgt definiert:

$sch(R \div S) = A - B$

$val(R \div S) = \{t \mid \forall s \in val(S) \quad \exists r \in val(R): r.B = s.B \wedge t = r.Q\}$

Intuitive Bedeutung:
Ein Tupel t ist in $R \div S$ genau dann, wenn für alle S−Tupel s ein zusammengesetztes Tupel <t,s> in R enthalten ist.

Bemerkung:
Die Division ist die einfachste Art, Anfragen der Form "... für alle ..." auszudrücken.

Beispiel für die Division:

Finden Sie die Kundennummern derjenigen Kunden, die alle überhaupt lieferbaren Produkte irgendwann bestellt haben.

→ π[KNr, PNr] (Bestellungen) $\div$ π[PNr] (Produkte)

π[KNr, PNr] (Bestellungen)

KNr	PNr
1	1
1	2
1	3
1	4
1	5
1	6
1	7
2	1
2	5
3	5
3	7
4	1
6	1
6	7

π[PNr] (Produkte)

PNr
1
2
3
4
5
6
7

→ Resultat:

KNr
1

Satz:

Seien R(A,B,C), T(A,B) und S(C) Relationen, so daß $R = T \times S$. Dann gilt: $T = R \div S$.

Kartesisches Produkt $\times$:

Seien R, S zwei Relationen mit Schemata A=sch(R) und B=sch(S). Sei A' ein Schema, bei dem alle Attributnamen Ai, die auch in sch(S) vorkommen, unbenannt sind in R.Ai, und sei B' ein Schema, bei dem alle Attributnamen Ai, die auch in sch(R) vorkommen, umbenannt sind in S.Ai. Das Resultat des kartesischen Produkts R $\times$ S ist wie folgt definiert:

$sch(R \times S) = A' \cup B'$

$val(R \times S) = \{t \mid \exists\, r \in val(R) \quad \exists\, s \in val(S):\ t.A' = r.A \text{ und } t.B' = s.B\}$

Intuitive Bedeutung:
R $\times$ S enthält alle möglichen Kombinationen von R−Tupeln und S−Tupeln.

Beispiel für das kartesische Produkt:

R $\times$ S mit:

R

R1	**J**
a	1
b	2
c	2

S

S1	**J**
w	2
x	2

→ Resultat:

R1	**R.J**	**S1**	**S.J**
a	1	w	2
b	2	w	2
c	2	w	2
a	1	x	2
b	2	x	2
c	2	x	2

Rückführung des Join auf das kartesische Produkt:

Seien R und S zwei Relationen mit Schemata A=sch(R) und B=sch(S), und sei J=sch(R) $\cap$ sch(S). Es gilt (bis auf Umbenennungen von Attributen):

$R \mid\times\mid S = \pi[A \cup B - \{S.J\}]\ (\ \sigma[R.J=S.J]\ (\ R \times S\ ))$

Rückführung der Division auf das kartesische Produkt:

Satz:
Seien R und S zwei Relationen mit Schemata A=sch(R) und B=sch(S) mit $A \supset B$, und sei Q=sch(R) − sch(S). Es gilt (bis auf Umbenennungen von Attributen):

$$R \div S = \pi[Q](R) - \pi[Q]\,(\,(\,\pi[Q](R) \times S\,) - R\,)$$

Beispiel:

R := π[KNr, PNr] (Bestellungen)	
S := π[PNr] (Produkte)	
T1 := π[KNr](R) × S	alle überhaupt möglichen Bestellungen
T2 := T1 − R	potentiell mögliche, aber nicht erfolgte Bestellungen
T3 := π[KNr] (T2)	Kunden, die nicht alle Produkte bestellt haben
T4 := π[KNr] (R) − T3	Kunden, die alle Produkte bestellt haben

$\theta-$**Join**:

Seien R, S zwei Relationen mit Schemata sch(R) und sch(S), so daß $sch(R) \cap sch(S) = \emptyset$. Seien $A \subseteq sch(R)$ und $B \subseteq sch(S)$. Sei ferner θ eine der Vergleichsoperationen $=, \neq, <, >, \leq, \geq$. Das Resultat des $\theta-$Joins $R \mid\times\mid [A\ \theta\ B]\ S$ (auch $R \mid\times\mid_{A\,\theta\,B} S$ geschrieben) ist wie folgt definiert:

$sch(R \mid\times\mid [A\ \theta\ B]\ S) = sch(R \times S) = sch(R) \cup sch(S)$
$val(R \mid\times\mid [A\ \theta\ B]\ S) = val(\sigma[A\ \theta\ B](R \times B))$

Wenn θ die Vergleichsoperation $=$ ist, dann heißt der $\theta-$Join auch Equijoin.
In analoger Weise läßt sich der $\theta-$Join zu einem Join mit allgemeiner Boolescher Filterformel erweitern, wobei dieselben Formeln wie bei der Selektion zulässig sind.

Beispiele für den $\theta-$Join:

1) Finde alle Kunden, die an einem Lagerort wohnen.
$\rightarrow \pi$[KNr, Name, Stadt, ...] (Kunden $\mid\times\mid$[Stadt=Lagerort] Produkte)

2) Finde alle bisherigen Bestellungen, die den momentanen Vorrat erschöpfen würden.
$\rightarrow$ B(BestNr,Monat,Tag,KNr,B.PNr, ...) := Bestellungen(BestNr, Monat, Tag, KNr, PNr, ...)
π[BestNr, ...] (B $\mid\times\mid$[B.PNr=PNr $\wedge$ Menge >= Vorrat] Produkte)

3) Finde alle Paare von Kunden, die in derselben Stadt wohnen.
$\rightarrow$ K1(K1.KNr, K1.Name, K1.Stadt, ...) := Kunden(KNr, Name, Stadt, ...)
K2(K2.KNr, K2.Name, K2.Stadt, ...) := Kunden(KNr, Name, Stadt, ...)
π[K1.KNr, K2.KNr] (K1 $\mid\times\mid$[K1.Stadt = K2.Stadt] K2)
bzw. besser:
π[K1.KNr, K2.KNr] (K1 $\mid\times\mid$[K1.Stadt = K2.Stadt $\wedge$ K1.KNr < K2.KNr] K2)
um jedes (echte) Kundenpaar nur einmal aufzulisten

saloppere Schreibweise häufig:
π[KNr1, KNr2] (Kunden $\mid\times\mid$[Stadt1 = Stadt2 $\wedge$ KNr1 < KNr2] Kunden)

Semi−Join und Anti−Join:

Seien R, S zwei Relationen mit Schemata sch(R) und sch(S), so daß $sch(R) \cap sch(S) = J$. Seien $A \subseteq sch(R)$ und $B \subseteq sch(S)$. Das Resultat des Semi−Joins $R \mid\times S$ und des Anti−Joins $R \mid- S$ sind wie folgt definiert:

$sch(R \mid\times S) = sch(R)$
$val(R \mid\times S) = \{r \mid r \in val(R) \wedge \exists\, s \in val(S):\ r.J = s.J\}$

$sch(R \mid- S) = sch(R)$
$val(R \mid- S) = \{r \mid r \in val(R) \wedge \neg (\exists\, s \in val(S):\ r.J = s.J)\}$

Outer Join

Seien R und S zwei Relationen mit Schemata A=sch(R) und B=sch(S), und sei J=sch(R) ∩ sch(S).
Bezeichne ferner ω den Nullwert.
Das Resultat des Outer Joins R |*| S ist wie folgt definiert:

$$sch(R \mid *\mid S) = sch(R) \cup sch(S)$$

$$val(R \mid *\mid S) = val(R \mid \times \mid S) \cup \{t \mid \exists r \in val(R) : t.A=r.A \wedge \neg (\exists s \in val(S) : t.J=s.J) \wedge t.(B-J)=\omega \} \cup \{t \mid \exists s \in val(S) : t.B=s.B \wedge \neg (\exists r \in val(R) : t.J=r.J) \wedge t.(A-J)=\omega \}.$$

Beispiel für den Outer Join:

Gib alle Kunden mit 5 % Rabatt zusammen mit ihren Bestellungen aus, und zwar auch, wenn für einen Kunden gar keine Bestellungen vorliegen.

$$\sigma [Rabatt = 0.05] (Kunden \mid *\mid Bestellungen)$$

KNr	Name	Stadt	Saldo	Rabatt	BestNr	...
5	Becker	Dillingen	0.00	0.05	ω	...
6	Meier	Saarlouis	−3800.00	0.05	14	...
6	Meier	Saarlouis	−3800.00	0.05	15	...
6	Meier	Saarlouis	−3800.00	0.05	16	...

Rückführung des Outer Join auf Join und kartesisches Produkt:

Seien NR und NS Relationen mit sch(NR)=sch(R)−sch(S) und sch(NS)=sch(S)−sch(R), die jeweils genau ein Tupel enthalten, dessen Attribute alle ω als Wert haben.
Dann ist:

$$R \mid *\mid S = (R \mid \times \mid S) \cup ((R - \pi[sch(R)] (R \mid \times \mid S)) \times NS) \cup (NR \times (S - \pi[sch(S)] (R \mid \times \mid S)))$$

Left Outer Join:

$$sch(R \mid\mid * \mid S) = sch(R) \cup sch(S)$$

$$val(R \mid\mid * \mid S) = val(R \mid \times \mid S) \cup \{t \mid \exists r \in val(R) : t.A=r.A \wedge \neg (\exists s \in val(S) : t.J=s.J) \wedge t.(B-J)=\omega \}.$$

Right Outer Join:

$$sch(R \mid * \mid\mid S) = sch(R) \cup sch(S)$$

$$val(R \mid * \mid\mid S) = val(R \mid \times \mid S) \cup \{t \mid \exists s \in val(S) : t.B=s.B \wedge \neg (\exists r \in val(R) : t.J=r.J) \wedge t.(A-J)=\omega \}.$$

Äquivalenzregeln, Minimalität und Vollständigkeit der Relationenalgebra

Seien R, S, T Relationen, P, P1, P2 Prädikate, R1, R2, S1 Teilmengen der Attribute von R bzw. S. Es gelten die folgenden Äquivalenzregeln:

Kommutativitätsregeln:

1) $\pi[R1]\,\sigma[P]\,(R) = \sigma[P]\,\pi[R1]\,(R)$
falls P nur R1−Attribute enthält

2) $R \,|x|\, S = S \,|x|\, R$

Assoziativitätsregeln:

3) $R \,|x|\, (S \,|x|\, T) = (R \,|x|\, S) \,|x|\, T$

Idempotenzregeln:

4) $\pi[R1]\,(\pi[R2]\,(R)) = \pi[R1]\,(R)$
falls $R1 \subseteq R2$

5) $\sigma[P1]\,(\sigma[P2]\,(R)) = \sigma[P1 \wedge P2]\,(R)$

Distributivitätsregeln:

6) $\pi[R1]\,(R \cup S) = \pi[R1](R) \cup \pi[R1](S)$

7) $\sigma[P]\,(R \cup S) = \sigma[P](R) \cup \sigma[P](S)$

8) $\sigma[P]\,(R \,|x|\, S) = \sigma[P](R) \,|x|\, S$
falls P nur R−Attribute enthält

9) $\pi[R1,S1]\,(R \,|x|\, S) = \pi[R1](R) \,|x|\, \pi[S1](S)$
falls Joinattribute $\subseteq R1 \cup S1$

10) $R \,|x|\, (S \cup T) = (R \,|x|\, S) \cup (R \,|x|\, T)$

Invertierungsregeln:

11) $\pi[sch(R)]\,(R \,\|^*|\, S) = R$

Diese Regeln bilden die Grundlage für eine automatische Optimierung von deklarativen Anfragen in Form von Ausdrücken der Relationenalgebra.

Definition:

Die Menge der relationenalgebraischen Ausdrücke über einer Menge von Relationen R1, ..., Rn ist wie folgt definiert:

(i) R1, ..., Rn sind Ausdrücke.

(ii) Wenn R, S, T, Q Ausdrücke sind,
F eine Filterformel über dem Resultatschema von R ist,
A eine Teilmenge des Resultatschemas von R ist,
S und T dasselbe Resultatschema haben und $sch(R) \supset sch(Q)$ gilt,
dann sind
$\sigma[F](R)$, $\pi[A](R)$, $R \,|x|\, S$, $R \times S$, $R \,|^*|\, S$, $S \cap T$, $S \cup T$, $S - T$, $R \div Q$
auch Ausdrücke.

(iii) Nur die gemäß (i) und (ii) erzeugten Ausdrücke sind relationalgebraische Ausdrücke.

Satz:

$\times, \pi, \sigma, \cup$ und $-$ bilden eine minimale Menge von Operationen, mit denen sich alle Operationen der Relationenalgebra ausdrücken lassen.

Beweis: siehe Vorlesung

Definition:

Eine Anfragesprache heißt *relational vollständig*, wenn sich damit alle Anfragen der (minimalen) Relationenalgebra ausdrücken lassen.

2.3 Tupel−Relationenkalkül

Anfragen sind prädikatenlogische Mengenspezifikation der Form
$\{t \mid F(t)\}$,
wobei F ein prädikatenlogischer Ausdruck über einer Menge von Tupelvariablen ist mit t als einziger freier Variable, oder der Form
$\{<t1.A1, ..., tn.An> \mid F(t1, ..., tn)\}$,
wobei F ein prädikatenlogischer Ausdruck über einer Menge von Tupelvariablen ist mit t1, ..., tn als (nicht notwendigerweise verschiedenen) einzigen freien Variablen und A1, ..., An Attributnamen sind.

Die Menge der zulässigen prädikatenlogischen Ausdrücke F ist wie folgt präzise definiert:

1) Für eine Tupelvariable r und eine Relation R ist $r \in R$ (auch R(r) geschrieben) ein zulässiger Ausdruck.

2) Für Tupelvariable r, s und Attribute A, B mit dom(A)=dom(B), Konstanten $c \in dom(A)$ und Vergleichsoperationen $\theta \in \{=, \neq, <, >, \leq, \geq\}$ sind $r.A\ \theta\ s.B$ und $r.A\ \theta\ c$ zulässige Ausdrücke.

3) Falls F1 und F2 zulässige Ausdrücke sind, dann sind auch $F1 \wedge F2$, $F1 \vee F2$ und $\neg F1$ zulässig.

4) Falls F ein zulässiger Ausdruck mit einer freien Tupelvariable r ist, dann sind auch $\exists$ r: F(r) und $\forall$ r: F(r) zulässige Ausdrücke.

5) Falls F ein zulässiger Ausdruck ist, dann ist auch (F) zulässig.

6) Nur die aufgrund von 1) bis 5) erzeugten Ausdrücke sind zulässig.

Beispiele:

1) Finden Sie (die Namen) alle(r) Kunden mit negativem Saldo.
   $\rightarrow \{t \mid t \in \text{Kunden} \wedge t.\text{Saldo} < 0.0\}$ bzw. $\{t.\text{Name} \mid t \in \text{Kunden} \wedge t.\text{Saldo} < 0.0\}$

2) Finden Sie die Namen aller Kunden, die eine unbezahlte Bestellung haben, die vor Anfang Oktober erfolgte.
   $\rightarrow \{t.\text{Name} \mid t \in \text{Kunden} \wedge$
   $\exists\, b: b \in \text{Bestellungen} \wedge$
   $t.\text{KNr}=b.\text{KNr} \wedge b.\text{Monat} < 10 \wedge b.\text{Status} \neq \text{'bezahlt'}\}$

3) Finden Sie die Namen der Homburger Kunden, die seit Anfang September ein Produkt aus Homburg geliefert bekommen haben, jeweils mit der Bezeichnung des entsprechenden Produkts.
   $\rightarrow \{k.\text{Name}, p.\text{Bez} \mid k \in \text{Kunden} \wedge p \in \text{Produkte} \wedge$
   $\exists\, b: b \in \text{Bestellungen} \wedge$
   $k.\text{Stadt}=\text{'Homburg'} \wedge b.\text{Monat} \geq 9 \wedge p.\text{Lagerort'}=\text{'Homburg'} \wedge$
   $k.\text{KNr}=b.\text{KNr} \wedge b.\text{PNr}=p.\text{PNr}\}$

4) Finden Sie die Kunden, von denen mindestens eine Bestellung registriert ist.
   $\rightarrow \{t \mid t \in \text{Kunden} \wedge \exists\, b: b \in \text{Bestellungen} \wedge b.\text{KNr}=t.\text{KNr}\}$

5) Finden Sie die Kunden, von denen keine Bestellung registriert ist.
   $\rightarrow \{t \mid t \in \text{Kunden} \wedge \neg(\exists\, b: b \in \text{Bestellungen} \wedge b.\text{KNr}=t.\text{KNr})\}$ oder
   $\rightarrow \{t \mid t \in \text{Kunden} \wedge \forall\, b: b \in \text{Bestellungen} \Rightarrow b.\text{KNr} \neq t.\text{KNr}\}$

6) Finden Sie die Kunden, die alle überhaupt lieferbaren Produkte irgendwann bestellt haben.
   $\rightarrow \{t \mid t \in \text{Kunden} \wedge \forall\, p: p \in \text{Produkte} \Rightarrow$
   $(\exists\, b: b \in \text{Bestellungen} \wedge b.\text{PNr}=p.\text{PNr} \wedge b.\text{KNr}=t.\text{KNr})\}$

Domain−unabhängige Ausdrücke

Problem: Anfragen liefern u.U. unendliche Resultatmengen.
Beispiel: $\{t \mid \neg(t \in R)\}$ für eine endliche Relation R.

Lösungsidee:
Eine Formel des Tupelrelationenkalküls heißt *domain−unabhängig* genau dann, wenn für jede mögliche Ausprägung der Datenbank die Resultatmenge der Formel nur von der Datenbank, nicht aber von den zugrundeliegenden Domains abhängt.
Man sollte nur domain−unabhängige Formeln als Anfragen verwenden.

Präzisierung:

Sei F eine geschlossene Formel des Tupelrelationenkalküls. Eine *Interpretation* von F ist eine Abbildung der Relationsnamen, Konstanten und Variablen in F auf Relationsausprägungen, elementaren Konstanten und Tupeln, die aus einem *Universum* von Werten gebildet werden.
Wir sagen, daß F erfüllt ist, wenn die Interpretation von F in dem entsprechenden Universum erfüllt ist (bei kanonischer Interpretation der Junktoren und Quantoren).

Sei F(t) eine Formel des Tupelrelationenkalküls mit einer einzigen freien Variable t (in einer Anfrage $\{t \mid F(t)\}$). Die Interpretation von F(t) ist die Menge aller Tupel, die aus Elementen des gewählten Universums gebildet werden, so daß bei Substitution der Tupel für die freie Variable t die Formel F(t) im Universum erfüllt ist.

Wahl des Universums:
- Variante 1 − *unbeschränkte Interpretation*:
  das Universum besteht aus der Vereinigung aller Domains der Datenbank
- Variante 2 − *beschränkte Interpretation*:
  das Universum besteht aus dem *aktiven Domain* der Datenbank und einer Formel F, d.h. der Vereinigung aller Attributwerte der Relationsausprägungen der Datenbank und aller Konstanten in F

Beispiele:

Sei R eine Relation mit sch(R)={A}, dom(A)={1,2}, val(R)={<1>}.
Der Ausdruck $\{t \mid \neg(t \in R)\}$ hat
  als unbeschränkte Interpretation den Wert {<2>} und
  als beschränkte Interpretation den Wert ∅.
Der Ausdruck $\{t \mid \exists\, x : \neg(x=t) \wedge x.A=1\}$ hat
  als unbeschränkte Interpretation den Wert {<2>} und
  als beschränkte Interpretation den Wert ∅.
Der Ausdruck $\{t \mid t \in R \wedge \forall\, x : x=t\}$ hat
  als unbeschränkte Interpretation den Wert ∅ und
  als beschränkte Interpretation den Wert {<1>}.

Definition:

Eine Formel des Tupelrelationenkalküls heißt *domain−unabhängig* genau dann, wenn ihre unbeschränkte Interpretation für jede mögliche Wahl von Domains, die den aktiven Domain umfassen, mit ihrer beschränkten Interpretation übereinstimmt.

Sichere Ausdrücke

Die ”Sicherheit” von Ausdrücken ist eine hinreichende, syntaktisch prüfbare Bedingung für Domain−Unabhängigkeit.

Idee:
Eine Formel F(x) mit einer Variablen x heißt beschränkt, wenn für jede Interpretation von F gilt: wenn F(x) wahr ist, dann liegt die Interpretation von x im aktiven Domain von F.
Eine Formel F(x) mit einer Variablen x heißt unbeschränkt, wenn für jede Interpretation von F gilt: wenn die Interpretation von x nicht im aktiven Domain von F liegt, muß F(x) wahr sein.
Durch syntaktische Einschränkungen von Formeln können

- freie Variablen beschränkt werden
  (so daß in $\{t \mid F(t)\}$ die Formel F(t) nur für t aus dem aktiven Domain erfüllbar ist),
- durch Existenzquantoren gebundene Variablen beschränkt werden
  (so daß in $\exists x : G(x)$ die Formel G(x) nur für x aus dem aktiven Domain erfüllbar ist) und
- durch Allquantoren gebundene Variablen unbeschränkt werden
  (so daß in $\forall x : G(x)$ die Formel G(x) für Interpretationen von x, die nicht im aktiven Domain liegen, immer wahr ist).

Definition:

Sei F(t) eine Formel des Tupelrelationenkalküls mit freier Variable t. Zu jeder Tupelvariablen in F läßt sich ein Schema (d.h. eine Menge Attributen) aus F und dem Datenbankschema ableiten. Die *Beschränktheit* und die *Unbeschränktheit* der Attribute von Tupelvariablen in einer Formel F sind wie folgt induktiv definiert.

1) In $t \in R$ sind alle Attribute von t beschränkt.
2) In $t.A=c$ mit einer Konstanten c ist t.A beschränkt.
3) In $r.A=s.B \wedge F$ ist r.A beschränkt, wenn s.B in F beschränkt ist, und s.B beschränkt, wenn r.A in F beschränkt ist..

4a) In $F \wedge G$ ist t.A genau dann beschränkt, wenn t.A in F oder in G beschränkt ist.
4b) In $F \wedge G$ ist t.A genau dann unbeschränkt, wenn t.A in F und in G unbeschränkt ist.
5a) In $F \vee G$ ist t.A genau dann beschränkt, wenn t.A in F und in G beschränkt ist.
5b) In $F \vee G$ ist t.A genau dann unbeschränkt, wenn t.A in F oder in G unbeschränkt ist.
6a) In $\neg F$ ist t.A beschränkt genau dann, wenn t.A in F unbeschränkt ist.
6b) In $\neg F$ ist t.A unbeschränkt genau dann, wenn t.A in F beschränkt ist.
7a) In $\exists x$: F(x) ist t.A. beschränkt genau dann, wenn t.A in F beschränkt ist.
7b) In $\exists x$: F(x) ist t.A. unbeschränkt genau dann, wenn t.A in F unbeschränkt ist.
8a) In $\forall x$: P(x) ist t.A beschränkt genau dann, wenn t.A in F beschränkt ist.
8b) In $\forall x$: P(x) ist t.A unbeschränkt genau dann, wenn t.A in F unbeschränkt ist.
9) Nur die gemäß 1) bis 8) beschränkten bzw. unbeschränkten Attribute von Tupelvariablen sind beschränkt bzw. unbeschränkt.

Eine Formel des Tupelrelationenkalküls ist *sicher* (engl.: safe) genau dann, wenn

a) alle Attribute aller freien Tupelvariablen beschränkt sind,
b) für jede Teilformel der Form $\exists x$: P(x) alle Attribute von x in P beschränkt sind,
c) für jede Teilformel der Form $\forall x$: P(x) alle Attribute von x in P unbeschränkt sind.

Satz:

Jede sichere Formel des Tupelrelationenkalküls ist domain−unabhängig.

Beispiele (für das Schema R(A,B) und z.B. die Ausprägung R={<2,2>}):

- $\{t \mid \neg(t \in R) \vee t.A=2 \vee t.B=2\}$ ist unsicher.
- $\{t \mid \neg(t \in R) \wedge t.A=2\}$ ist unsicher.
- $\{t \mid \neg(t \in R) \wedge t.A=2 \wedge t.B=2\}$ ist sicher.

- $\{t \mid \exists s: (s \in R \wedge s.A=t.A)\}$ ist unsicher.
- $\{t \mid \exists s: (s \in R \vee s=t)\}$ ist unsicher.
- $\{t \mid \exists s: (s \in R \wedge s=t)\}$ ist sicher.
- $\{t \mid \neg(\exists s: (s \in R \wedge s=t))\}$ ist unsicher.

- $\{t \mid \forall s: (s \in R \wedge s=t)\}$ ist unsicher.
- $\{t \mid \forall s: (s \in R \Rightarrow s=t)\}$ ist unsicher.
- $\{t \mid \forall s: (s \in R \Rightarrow (s=t \wedge t.A=2 \wedge t.B=2))\}$ ist unsicher.
- $\{t \mid t \in R \wedge \forall s: (s \in R \Rightarrow (s=t \wedge t.A=2 \wedge t.B=2))\}$ ist sicher.
- $\{t \mid \forall s: (s \in R \wedge (s=t \wedge t.A=2 \wedge t.B=2))\}$ ist unsicher.
- $\{t \mid \forall s: (s \in R \Rightarrow \neg(s=t))\}$ ist unsicher.

- $\{t \mid t \in K \wedge \forall p: p \in P \Rightarrow (\exists b: b \in B \wedge b.PNr=p.PNr \wedge b.KNr=t.KNr)\}$ ist sicher.

2.4 Domain−Relationenkalkül

Anfragen sind prädikatenlogische Mengenspezifikationen der Form
$\{<x1, ..., xn> \mid F(x1, ..., xn)\}$
wobei F ein prädikatenlogischer Ausdruck über einer Menge von Domain−Variablen ist mit x1, ..., xn als einzigen freien Variablen.

Die Menge der zulässigen prädikatenlogischen Ausdrücke F ist wie folgt präzise definiert:

1) Für Domain−Variablen x1, ..., xn und eine Relation R mit n Attributen ist
$<x1,...,xn> \in R$ (auch R(x1, ..., xn) geschrieben) ein zulässiger Ausdruck.

2) Für Domain−Variablen x, y, Konstanten c und Vergleichsoperationen
$\theta \in \{=, \neq, <, >, \leq, \geq\}$ sind $x \theta y$ und $x \theta c$ zulässige Ausdrücke.

3) Falls F1 und F2 zulässige Ausdrücke sind, dann sind auch
$F1 \wedge F2$, $F1 \vee F2$ und $\neg F1$ zulässig.

4) Falls F ein zulässiger Ausdruck mit einer freien Domain−Variable x ist, dann sind auch $\exists x: F(x)$ und $\forall x: F(x)$ zulässige Ausdrücke.

5) Falls F ein zulässiger Ausdruck ist, dann ist auch (F) zulässig.

6) Nur die aufgrund von 1) bis 5) erzeugten Ausdrücke sind zulässig.

Zusätzliche Einschränkungen sind nötig für sichere Ausdrücke (analog zu Kapitel 2.3).

Beispiele:

1) Finden Sie (die Namen) alle(r) Kunden mit negativem Saldo.
→ {<k,n,st,sa,r> | <k,n,st,sa,r> ∈ Kunden ∧ sa < 0.0} bzw.
→ {<n> | ∃ k,st,sa,r: <k,n,st,sa,r> ∈ Kunden ∧ sa < 0.0}

2) Finden Sie die Namen aller Kunden, die eine unbezahlte Bestellung haben, die vor Anfang Oktober erfolgte.
→ {<n> | ∃ k,st,sa,r: <<u>k</u>,n,st,sa,r> ∈ Kunden ∧
∃ b,m,t,p,me,su,stat: <b,m,t,<u>k</u>,p,me,su,stat> ∈ Bestellungen ∧
m < 10 ∧ stat ≠ 'bezahlt'}

3) Finden Sie die Namen der Homburger Kunden, die seit Anfang September ein Produkt aus Homburg geliefert bekommen haben, jeweils mit der Bezeichnung des entsprechenden Produkts.
→ {<n,bez> | ∃ k,st,sa,r: <<u>k</u>,n,st,sa,r> ∈ Kunden ∧
∃ p,bez,g,pr,l,v: <<u>p</u>,bez,g,pr,l,v> ∈ Produkte ∧
∃ b,m,t,me,su,stat: <b,m,t,<u>k</u>,<u>p</u>,me,su,stat> ∈ Bestellungen ∧
st='Homburg' ∧ m ≥ 9 ∧ l='Homburg'}

4) Finden Sie die Kunden, von denen mindestens eine Bestellung registriert ist.
→ {<k,n,st,sa,r> | <<u>k</u>,n,st,sa,r> ∈ Kunden ∧
∃ b,m,t,p,me,su,stat: <b,m,t,<u>k</u>,p,me,su,stat> ∈ Bestellungen}

5) Finden Sie die Kunden, von denen keine Bestellung registriert ist.
→ {<k,n,st,sa,r> | <k,n,st,sa,r> ∈ Kunden ∧
∀ b,m,t,k',p,me,su,stat:
<b,m,t,k',p,me,su,stat> ∈ Bestellungen ⇒ k ≠ k'}

6) Finden Sie die Kunden, die alle überhaupt lieferbaren Produkte irgendwann bestellt haben.
→ {<k,n,st,sa,r> | <<u>k</u>,n,st,sa,r> ∈ Kunden ∧
∀ p: (∃ bez,g,pr,l,v: <<u>p</u>,bez,g,pr,l,v> ∈ Produkte) ⇒
(∃ b,m,t,me,su,stat: <b,m,t,<u>k</u>,<u>p</u>,me,su,stat> ∈ Bestellungen) }

Eine Anfragesprache auf der Basis des Domain−Relationenkalküls: Query−by−Example (QBE)

(bzw. MS−Access u.a. als kommerzielle Variante)

Idee:
Bedingungen eines Ausdrucks des Domain−Relationenkalküls werden in Form von ”Beispielelementen” in entsprechende Fenster eingetragen, wobei für jedes Relationen−schema ein Fenster vorgegeben wird.
Gleiche Elemente in verschiedenen Fenstern stehen für ”Joinbedingungen”.

Beispiel:

Finden Sie die Namen der Homburger Kunden, die seit Anfang September ein Produkt aus Homburg (oder Saarbrücken) geliefert bekommen haben, dessen Preis über 100 DM liegt, jeweils mit der Bezeichnung des entsprechenden Produkts.

Kunden

KNr	Name	Stadt	Saldo	Rabatt
_55	P.	Homburg		

Bestellungen

BestNr	Monat	Tag	KNr	PNr	Menge	Summe	Status
	>= 9		_55	_33			

Produkte

PNr	Bez	Gewicht	Preis	Lagerort	Vorrat
_33	P.		> 100.00	Homburg Saarbrücken	

2.5 Mächtigkeit relationaler Anfragesprachen

Anfragesprachen existierender Datenbanksysteme basieren überwiegend entweder auf
dem Tupel−Relationenkalkül (z.B. Ingres/Quel) oder
dem Domain−Relationenkalkül (z.B. DB2/QMF oder MS−Access) oder
einer Kombination von Relationenalgebra und Tupel−Relationenkalkül (SQL).

Satz:

Alle Anfragen, die sich mit der Relationenalgebra ausdrücken lassen, lassen sich auch mit dem sicheren Tupel−Relationenkalkül oder dem sicheren Domain−Relationenkalkül ausdrücken.

Alle Anfragen, die sich mit dem sicheren Tupel−Relationenkalkül ausdrücken lassen, lassen sich auch mit der Relationenalgebra ausdrücken.

Alle Anfragen, die sich mit dem sicheren Domain−Relationenkalkül ausdrücken lassen, lassen sich auch mit der Relationenalgebra ausdrücken.

Beweis: siehe Vorlesung

Bemerkung:
Relationenalgebra, sicherer Tupel−Relationenkalkül und sicherer Domain−Relationenkalkül haben also dieselbe Ausdrucksmächtigkeit.

2.6 Erweiterung des Domain−Relationenkalküls um Rekursion

In DBS werden Informationen traditionellerweise ausschließlich *extensional* − in Form von *Daten (Fakten)* − repräsentiert. In wissensbasierten Systemen und in modernen, um deduktive Fähigkeiten erweiterten DBS können Informationen zusätzlich auch *intensional* − in Form von *Regeln* − repräsentiert werden. Mit Hilfe der Regeln können aus den Fakten weitere Informationen hergeleitet werden.

Beispiel mit rein extensionaler Repräsentation:

Flüge (F)

FlugNr	Abflugort	Zielort	...
LH58	Frankfurt	Chicago	
AA371	Chicago	Phoenix	
DA77	Phoenix	Yuma	
AA70	Frankfurt	Dallas	
AA351	Dallas	Phoenix	
UA111	Chicago	Dallas	

Flugverbindungen (V)

Abflugort	Zielort
Frankfurt	Chicago
Frankfurt	Dallas
Frankfurt	Phoenix
Frankfurt	Yuma
...	...

Beispiel mit intensionaler Repräsentation der Flugverbindungen:

als Fixpunktgleichung in der Relationenalgebra:

$$V = F \cup (V \,|x|\, [V.Zielort = F.Abflugort]\ F)$$

mit der Semantik, daß V die bzgl. $\subseteq$ kleinste Relation ist, für die Fixpunktgleichung erfüllt ist.

als Menge von (Prolog−artigen) Regeln:

Flugverbindungen (a,z) :− Flüge (a,z)
Flugverbindungen (a,z) :− Flugverbindungen (a,o), Flüge (o,z)

als Formeln des Domain−Relationenkalküls:

$\forall$ a, z: Flüge (a,z) $\Rightarrow$ Flugverbindungen (a,z)
$\forall$ a, z, o: Flugverbindungen (a,o) $\wedge$ Flüge (o,z) $\Rightarrow$ Flugverbindungen (a,z)

bzw.

$\forall$ a, z: $\langle a,z\rangle \in$ Flüge $\Rightarrow$ $\langle a,z\rangle \in$ Flugverbindungen
$\forall$ a, z, o: $\langle a,o\rangle \in$ Flugverbindungen $\wedge$ $\langle o,z\rangle \in$ Flüge $\Rightarrow$ $\langle a,z\rangle \in$ Flugverbindungen

Datalog

Definitionen:

Gegeben sei eine endliche Menge von Prädikaten (Prädikatsymbolen).
Eine *Hornklausel* über dieser Prädikatenmenge ist eine logische Formel der Form

$$\forall X1, ..., Xn: \quad P1 (Z11, Z12, ..., Z1k_1) \wedge ... \wedge Pm (Zm1, Zm2, ..., Zmk_m) \Rightarrow P0 (Z01, Z02, ..., Z0k_0)$$

mit k_i−stelligen Prädikaten Pi, Variablen X1, ..., Xn und
Konstanten oder Variablen Zij, so daß Zij entweder eine Konstante ist oder eine der Variablen X1, ..., Xn.

Die Formel

$$P0 (Z01, Z02, ..., Z0k_0)$$

wird als "Kopf" (engl.: head) der Hornklausel bezeichnet,
die Formel

$$P1 (Z11, Z12, ..., Z1k_1) \wedge ... \wedge Pm (Zm1, Zm2, ..., Zmk_m)$$

als "Rumpf" (engl.: body).

Gegeben sei eine endliche Menge von Prädikaten.
Ein *Datalog−Programm* besteht aus

- einer Menge von Fakten der Form Pi (V1, V2, ..., Vk)
  mit einem k−stelligen Prädikat Pi und Konstanten V1, ..., Vk und
- einer Menge von Regeln in Form von Hornklauseln über den gegebenen Prädikaten.

Ein *Datalog−Programm mit Negation* besteht aus

- einer Menge von Fakten wie in einem einfachen Datalog−Programm und
- einer Menge von Regeln in Form von Klauseln vom selben Typ wie bei Datalog, bei denen jedoch Prädikate im Rumpf negiert sein können.

Eine *Anfrage* (Ziel; engl: goal) für ein Datalog−Programm ist ein Ausdruck der Form P(Z1, ..., Zk) mit einem k−stelligen Prädikat P und Konstanten oder Variablen Z1, ..., Zk, so daß mindestens eines der Zj eine (freie) Variable ist.

Eine rekursive Regel heißt *linear*, wenn das Kopfprädikat genau einmal unter den Rumpfprädikaten auftaucht und keines der anderen Rumpfprädikate als Kopfprädikat einer anderen rekursiven Regel auftritt.
Ein Datalog−Programm heißt linear, wenn es nur lineare oder nichtrekursive Regeln hat.

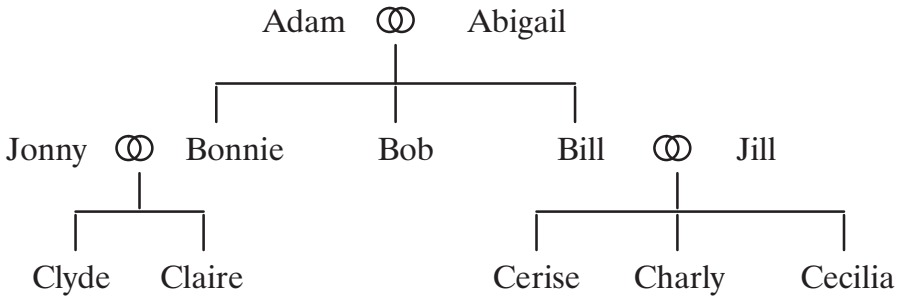
Satz:

Die Menge der Anfragen, die sich mit Datalog mit Negation ausdrücken lassen, ist eine echte Obermenge der Menge von Anfragen, die sich mit dem Domain−Relationenkalkül ausdrücken lassen.

Bemerkung:

Im Gegensatz zu Prolog ist Datalog wirklich nichtprozedural (z.B. ist die Reihenfolge der Regeln irrelevant) und soll große, persistente Sammlungen von Fakten und Regeln verarbeiten.

Beispiel:



Fakten

Mann (Adam), Mann (Jonny), Mann (Bob), Mann (Bill), Mann (Clyde) ,Mann (Charly)
Frau (Abigail), Frau (Bonnie), Frau (Jill), Frau (Claire), Frau (Cerise), Frau (Cecilia)
Ehepaar (Adam, Abigail), Ehepaar (Jonny, Bonnie), Ehepaar (Bill, Jill)
Elternteil (Adam, Bonnie), Elternteil (Adam, Bob), Elternteil (Adam, Bill),
Elternteil (Abigail, Bonnie), Elternteil (Abigail, Bob), Elternteil (Abigail, Bill),
Elternteil (Jonny, Clyde), Elternteil (Jonny, Claire),
Elternteil (Bonnie, Clyde), Elternteil (Bonnie, Claire),
Elternteil (Bill, Cerise), Elternteil (Bill, Charly), Elternteil (Bill, Cecilia),
Elternteil (Jill, Cerise), Elternteil (Jill, Charly), Elternteil (Jill, Cecilia)
SelbeGeneration (Adam, Abigail),
SelbeGeneration (Adam, Adam), SelbeGeneration (Abigail, Abigail)
SpieltMit (Clyde, Charly)

Regeln

Elternteil (X, Y) $\Rightarrow$ Vorfahren (X, Y)
Elternteil (X, Y) $\wedge$ Vorfahren (Y, Z) $\Rightarrow$ Vorfahren (X, Z)
Elternteil (X, Y) $\wedge$ Mann (X) $\Rightarrow$ Vater (X, Y)
Elternteil (X, Y) $\wedge$ Frau (X) $\Rightarrow$ Mutter (X, Y)
Elternteil (X, Y) $\wedge$ Elternteil (X, Z) $\wedge$ $Y \neq Z$ $\Rightarrow$ Geschwister (Y, Z)
Elternteil (X, Y) $\wedge$ Elternteil (U, W) $\wedge$ Geschwister (X, U) $\wedge$ Frau (W) $\Rightarrow$ Cousine (Y, W)
Elternteil (X, Y) $\wedge$ Elternteil (U, W) $\wedge$ SelbeGeneration (X, U) $\Rightarrow$ SelbeGeneration (Y, W)
Geschwister (X, Y) $\Rightarrow$ SpieltMit (X, Y)
SpieltMit (Clyde, Y) $\Rightarrow$ SpieltMit (Claire, Y)
TRUE $\Rightarrow$ SpieltMit (Cecilia, Y) *(leerer Rumpf)*
Elternteil (X, Y) $\Rightarrow$ Verwandt (X, Y)
Geschwister (X, Y) $\Rightarrow$ Verwandt (X, Y)
Verwandt (X, Y) $\Rightarrow$ Verwandt (Y, X)
Verwandt (X, Y) $\wedge$ Verwandt (Y, Z) $\Rightarrow$ Verwandt (X, Z) *(nichtlinear)*
Mann (X) $\wedge$ $\neg$ Ehepaar (X, Y) $\Rightarrow$ Ledig (X) *(mit Negation)*
Frau (Y) $\wedge$ $\neg$ Ehepaar (X, Y) $\Rightarrow$ Ledig (Y) *(mit Negation)*

Beispiele für Anfragen

Ledig (X)
Cousine (Clyde, X)
SelbeGeneration (Clyde, X)

Auswertung von Datalog−Programmen mit prädikatenlogischer Resolution

Beispiel:

Fakten:

Flüge (Frankfurt, Chicago)
Flüge (Chicago, Phoenix)
...

Regeln:

Flüge (a,z) ⇒ Flugverbindungen (a,z)
Flugverbindungen (a,o) ∧ Flüge (o,z) ⇒ Flugverbindungen (a,z)

Anfrage:

Flugverbindungen (Frankfurt, x)

Dies entspricht der prädikatenlogischen Formel (mit F für Flüge und V für Flugverbindungen):

[F(Frankfurt, Chicago) ∧ F(Chicago, Phoenix) ∧ ... ∧
(∀ a, z : ¬ F(a,z) ∨ V(a,z)) ∧
(∀ a, o, z : ¬ V(a,o) ∨ ¬ F(o,z) ∨ V(a,z))]
⇒ V (Frankfurt, x)

Gesucht sind alle Interpretationen von x, für die die Formel gültig ist,
also alle Interpretationen, so daß die negierte Formel unerfüllbar ist.

Für die Resolution besteht die Aufgabe somit darin, alle Werte für x zu bestimmen, so daß aus der folgenden Klauselmenge die leere Klausel ableitbar ist:

{ {F(Frankfurt, Chicago)}, {F(Chicago, Phoenix)}, ...,
{¬ F(a,z), V(a,z)}, {¬ V(a,o), ¬ F(o,z), V(a,z)}, {¬ V(Frankfurt, x)} }

Problem:

Bei den üblichen Depth−First−Suchstrategien nach resolvierbaren Klauseln
führt die Resolution a la Prolog zu einer wiederholten Traversierung aller Fakten,
was bei großen, auf Magnetplatte gespeicherten Faktenmengen höchst ineffizient ist.

Effiziente Auswertung von linearen Datalog−Programmen (ohne Negation)

Vereinfachung der Darstellung:
Sei P ein rekursives Kopfprädikat mit
genau einer nichtrekursiven Regel Q1(...) ∧ ... ∧ Qm(...) ⇒ P(...) und
genau einer linearen Regel R1(...) ∧ ... ∧ Rn(...) ∧ P(...) ⇒ P(...).

Naive Bottom−up−Auswertung:

P_0 := Auswertung des Rumpfes Q1(...) ∧ ... ∧ Qm(...)
i := 1
repeat
  P_i := Auswertung des Rumpfes R1(...) ∧ ... ∧ Rn(...) ∧ P_{i-1}(...) /* Join */
  P_i := $P_i \cup P_{i-1}$
  i := i+1
until $P_i = P_{i-1}$ /* Erreichung eines Fixpunkts */
P := P_i

Semi−naive Bottom−up−Auswertung (Delta−Iteration):

P_0 := Auswertung des Rumpfes Q1(...) ∧ ... ∧ Qm(...)
Δ_0 := P_0
i := 1
repeat
  Δ_i := Auswertung des Rumpfes R1(...) ∧ ... ∧ Rn(...) ∧ Δ_{i-1}(...) /* Join */
  Δ_i := $\Delta_i - P_{i-1}$
  P_i := $P_{i-1} \cup \Delta_i$
  i := i+1
until $\Delta_i = \emptyset$ /* Erreichung eines Fixpunkts */
P := P_i

Beispiel:

Flüge (a,z) ⇒ Flugverbindungen (a,z)
Flugverbindungen (a,o) ∧ Flüge (o,z) ⇒ Flugverbindungen (a,z)
Anfrage:
Flugverbindungen (Frankfurt, z)

Naive Auswertung:

i	P_i
0	(F,C), (C,P), (P,Y), (F,D), (D,P), (C,D)
1	(F,C), (C,P), (P,Y), (F,D), (D,P), (C,D), (F,P), (C,Y), (D,Y)
2	(F,C), (C,P), (P,Y), (F,D), (D,P), (C,D), (F,P), (C,Y), (D,Y), (F,Y)
3	(F,C), (C,P), (P,Y), (F,D), (D,P), (C,D), (F,P), (C,Y), (D,Y), (F,Y)

Semi−naive Auswertung:

i	P_i	Δ_i
0	(F,C), (C,P), (P,Y), (F,D), (D,P), (C,D)	(F,C), (C,P), (P,Y), (F,D), (D,P), (C,D)
1	(F,C), (C,P), (P,Y), (F,D), (D,P), (C,D), (F,P), (C,Y), (D,Y)	(F,P), (C,Y), (D,Y)
2	(F,C), (C,P), (P,Y), (F,D), (D,P), (C,D), (F,P), (C,Y), (D,Y), (F,Y)	(F,Y)
3	(F,C), (C,P), (P,Y), (F,D), (D,P), (C,D), (F,P), (C,Y), (D,Y), (F,Y)	−

Ergänzende Literatur zu Kapitel 2:

S. Abiteboul, R. Hull, V. Vianu, Foundation of Databases, Addison−Wesley, 1995

P. Kandzia, H.−J. Klein, Theoretische Grundlagen relationaler Datenbanksysteme, BI−Verlag, Reihe Informatik, Band 79, 1993

P.C. Kanellakis, Elements of Relational Database Theory, in: J. van Leeuwen (Ed.), Handbook of Theoretical Computer Science, Elsevier, Amsterdam, 1991

D. Maier, The Theory of Relational Databases, Computer Science Press, 1983

E.F. Codd, A Relational Model of Data for Large Shared Data Banks, Communications of the ACM Vol.13 No.6, 1970, pp. 377−387

J.D. Ullman, Principles of Database and Knowledge−Based Systems Vol. 1 and Vol.2, Computer Science Press, 1989

J. Biskup, Grundlagen von Informationssystemen, Vieweg−Verlag, 1995

S. Ceri, G. Gottlob, L. Tanca, Logic Programming and Databases, Springer−Verlag, 1990

A.B. Cremers, U. Griefahn, R. Hinze, Deduktive Datenbanken, Vieweg−Verlag, 1994