

# Datenhaltung

Sommersemester 2008

## Contents

<b>1</b>	<b>Relationale Algebra</b>	<b>1</b>
1.1	Join . . . . .	1
<b>2</b>	<b>Entity Relationship Model</b>	<b>2</b>
2.1	Kardinalitäten . . . . .	2
<b>3</b>	<b>Relationaler Entwurf</b>	<b>2</b>
3.1	Schlüssel . . . . .	2
3.2	RAP-Algorithmus . . . . .	3
3.3	Normalisierung . . . . .	5
3.4	Syntheseverfahren . . . . .	5
3.5	Dekompositions-Verfahren . . . . .	6
<b>4</b>	<b>Transaktionsverwaltung</b>	<b>6</b>
4.1	Begriffe . . . . .	6
4.2	Anomalien . . . . .	7
4.3	Eigenschaften von Histories . . . . .	7
4.4	Konfliktserialisierbarkeit . . . . .	7
4.5	Locking . . . . .	8
<b>5</b>	<b>Logische Anfrageoptimierung</b>	<b>9</b>

## 1 Relationale Algebra

### 1.1 Join

**allgemeiner Verbund.** Für zwei Relationen  $R$  und  $S$  und eine Selektionsbedingung  $c$  ist der allgemeine Verbund definiert als

$$R \bowtie_c S := \{r \cup s : r \in R \wedge s \in S \wedge c\}$$

Das ist äquivalent zu

$$\sigma_c(R \times S)$$

**Equijoin.** In diesem Spezialfall bestimmt die Selektionsbedingung die Gleichheit eines Attributes  $A$  von  $R$  und eines Attributes  $B$  von  $S$ .

$$R \bowtie_{A=B} S := \{r \cup s : r \in R \wedge s \in S \wedge r[A] = s[B]\}$$

Das ist äquivalent zu

$$\sigma_{[A=B]}(R \times S)$$

**Natural Join.** Ein Natural Join setzt sich zusammen aus einem Equijoin und dem Ausblenden gleicher Spalten. Für zwei Relationen  $R(A_1, \dots, A_n, B_1, \dots, B_n)$  und  $S(B_1, \dots, B_n, C_1, \dots, C_n)$  ist

$$R \bowtie S := \{r \cup s_{[C_1, \dots, C_n]} : r \in R \wedge s \in S \wedge r_{[B_1, \dots, B_n]} = s_{[B_1, \dots, B_n]}\}$$

## 2 Entity Relationship Model

### 2.1 Kardinalitäten

**Teilnehmerkardinalitäten.**

- $E1$  steht in Relation zu 0 oder 1  $E2$
- $E2$  steht in Relation zu 1 bis  $n$   $E1$

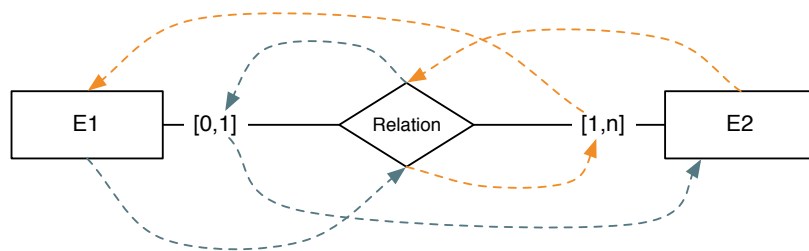


Figure 1: Leserichtung für Teilnehmerkardinalitäten

## 3 Relationaler Entwurf

**Mehrwertige Abhängigkeit (Multi-Valued Dependency).**

**Universalrelation** Die Universalrelation einer Menge von Relationen ist

$$R = R_1 \bowtie R_2 \bowtie \dots R_n$$

### 3.1 Schlüssel

**Superschlüssel.** Die Attributmenge  $K$  ist ein Superschlüssel, falls sie die Tupel einer Relation eindeutig identifiziert, d.h es gilt die funktionale Abhängigkeit  $K \rightarrow R$

**Schlüsselkandidat.** Die Attributmenge  $K$  ist ein Schlüsselkandidat, falls für das Relationenschema  $R$  die funktionale Abhängigkeit  $K \rightarrow R$  gilt und  $K$  minimal ist.

**Primärschlüssel.** Aus der Menge aller Schlüsselkandidaten wird ein Primärschlüssel ausgewählt, um die Tupel der Relation eindeutig zu identifizieren.

---

**Algorithm 1:** Schlüssel finden

---

**Input:** Relation  $R = (A_1, \dots, A_n)$ , funktionale Abhängigkeiten  $F$

```

1  $K \leftarrow \{\}$ 
2 for  $X \rightarrow Y$  in  $F$  do
3    $K \leftarrow K \cup X \setminus Y$ 
4 if  $K^+ = R$  then
5   if  $\forall K' \subset K : K'^+ \neq R$  then
6     return  $K$ 

```

---

**Hüllen.** Die transitive Hülle  $F_R^+$  einer Menge von funktionalen Abhängigkeiten  $F$  über der Relation  $R$  ist die Menge der funktionalen Abhängigkeiten, die von  $F$  impliziert werden:

$$F_R^+ := \{f : F \mid = f\}$$

Die Hülle einer Attributmenge  $X$  bezüglich einer Menge von funktionalen Abhängigkeiten  $F$  ist

$$X_F^* := \{A : X \rightarrow A \in F^+\}$$

### Überdeckung

$$F \equiv G \Leftrightarrow F^+ = G^+$$

## 3.2 RAP-Algorithmus

**Membership-Problem.** Kann eine bestimmte funktionale Abhängigkeit  $X \rightarrow Y$  aus einer Menge  $F$  abgeleitet werden? Gilt also

$$X \rightarrow Y \in F^+ \quad ?$$

Das modifizierte Membership-Problem

$$Y \subseteq X_F^*$$

kann durch den RAP-Algorithmus in Linearzeit (in der Anzahl der Attribute) gelöst werden.

### RAP-Regeln

**Reflexivität**  $\{\} \Rightarrow X \rightarrow X$

**Akkumulation**  $\{X \rightarrow YZ, Z \rightarrow VW\} \Rightarrow X \rightarrow YZV, X \rightarrow YZW, \dots$

**Projektivität**  $\{X \rightarrow YZ\} \Rightarrow X \rightarrow Y, X \rightarrow Z$

---

**Algorithm 2:** RAP-Algorithmus

---

**Input:** Attributmenge  $X$ , Attributmenge  $Y$

```
1  $X^* \leftarrow X$ 
2 while  $X^*$  nicht stabil do
3   if  $\exists f_1 = X_1 \rightarrow Y_1 \in F, X_1 \subseteq X^*$  then
4      $X^* \leftarrow X^* \cup Y_1$ 
5 if  $Y \subseteq X^*$  then
6   return wahr
7 else
8   return falsch
```

---

**Anomalien.** Ein Relationenschema mit Redundanzen kann die Entstehung von Anomalien begünstigen, z.B.:

**Einfügeanomalie** Durch die Schlüsseldefinition muss zum Einfügen einer bestimmten Information mehr Information bzw. Null-Werte eingefügt werden.

**Updateanomalie** Ändert sich eine Information, so müssen mehrere Tupel aktualisiert werden, was aufwändig und fehleranfällig ist.

**Löschanomalie** Durch Löschen einer bestimmten Information geht mehr Information verloren als erwünscht.

### Erwünschte Schemaeigenschaften

- **Redundanzen** vermeiden
- **Abhängigkeitstreue** besteht dann, wenn alle funktionalen Abhängigkeiten der Originalrelation auch in der zerlegten Relation noch gelten. Ein Relationenschema  $S$  ist abhängigkeitsreu bezüglich  $F$  wenn

$$F \equiv \{K \rightarrow R : (R, K) \in S, K \in \mathcal{K}\}$$

- **Verbundtreue** bezeichnet die Möglichkeit, die Originalrelation aus der zerlegten Relation mittels Natural Joins wiederherstellen zu können.

**Verbundtreue** Die Dekomposition der Relation  $R$  in  $R_1$  und  $R_2$  ist verbundtreu, falls

$$R_1 \cap R_2 \rightarrow R_1 \in F^+$$

oder

$$R_1 \cap R_2 \rightarrow R_2 \in F^+$$

**partielle Abhängigkeit** liegt vor, wenn ein Nichtschlüsselattribut funktional schon von einem Teil des Schlüssels abhängt.

### 3.3 Normalisierung

**1NF** Jedes Attribut der Relation muss einen atomaren Wertebereich haben. Verbieta Mengenwertige, geschachtelte oder zusammengesetzte Attribute.

**2NF** Jedes Nichtschlüsselattribut ist von jedem Schlüsselkandidaten voll funktional abhängig, d.h. abhängig vom ganzen Schlüssel, nicht nur von Teilen des Schlüssels.

**3NF** Kein Nichtschlüsselattribut hängt von einem Schlüsselkandidaten transitiv ab.

**Boyce-Codd NF** In allen Relationenschemata gehen die funktionalen Abhängigkeiten nur vom Primärschlüssel aus.

**4NF** Alle nicht-trivialen mehrwertigen Abhängigkeiten gehen vom Schlüsselkandidaten aus.

**5NF**

**2NF: Eliminierung von partiellen Abhängigkeiten**  $(\underline{AB}CD) A \rightarrow CD$   $(\underline{ACD})$   $(\underline{AB})$

### 3.4 Syntheseverfahren

**Ziel.** Das Syntheseverfahren zerlegt eine Relation so, dass die 3NF erreicht wird bei gleichzeitiger Abhängigkeitstreue und Minimalität.

---

**Algorithm 3:** Syntheseverfahren

---

**Input:** Relation  $R = (A_1, \dots, A_n)$ , funktionale Abhängigkeiten  $F$

```
1 // führe weitere FD ein für Verbundtreue:
2  $F \leftarrow F \cup \{A_1 \dots A_n \rightarrow \delta\}$ 
3 // zerlege FDs sodass rechte Seite atomar
4 for  $X \rightarrow A_1 \dots A_k$  in  $F$  do
5    $F \leftarrow F \setminus \{X \rightarrow A_1 \dots A_k\} \cup \{X \rightarrow A_1, \dots, X \rightarrow A_k\}$ 
6 // eliminiere redundante FDs
7 for  $f$  in  $F$  do
8   if  $F \setminus \{f\} \equiv F$  then
9      $F \leftarrow F \setminus \{f\}$ 
10 // entferne überflüssige Attribute auf der linken Seite
11 for  $X \rightarrow Y$  in  $F$  do
12   if  $X' \rightarrow Y \in F, X' \subset X$  then
13      $F \leftarrow F \setminus \{X \rightarrow Y\} \cup \{?\}$ 
14 // fasse FDs mit gleicher linker Seite zusammen
15 while  $\exists X \rightarrow Y \wedge \exists X \rightarrow Z \in F$  do
16    $F \leftarrow F \setminus \{X \rightarrow Y, X \rightarrow Z\} \cup \{X \rightarrow YZ\}$ 
17 //
```

---

### 3.5 Dekompositions-Verfahren

## 4 Transaktionsverwaltung

### 4.1 Begriffe

**Transaktion.** Als Transaktion bezeichnet man die Ausführung eines Programmes, das Lese- und Schreibzugriffe auf die Datenbank durchführt.

**Konflikt.** Konfliktär sind zwei Operationen, deren Reihenfolge nicht vertauscht werden kann, ohne dass sich ihr Ergebnis ändert. Zwei Operationen  $o_1$  und  $o_2$  konfliktieren, wenn sie auf das gleiche Datenobjekt zugreifen, und  $o_1$  oder  $o_2$  eine Schreiboperation ist.

$$o_1[x] \not\parallel o_2[x] \Leftrightarrow o_1[x] = w[x] \vee o_2[x] = w[x]$$

Eine Ausnahme bilden hier Inkrement- und Dekrement-Operationen, die gegenseitig kompatibel sind.

**History.** Eine History ist eine Menge von Transaktionen, deren Operationen nebenläufig ablaufen.

$$H = \{T_1, \dots, T_n\}$$

Eine vollständige History Zu Scheduling-Zwecken wird als History ein Präfix einer vollständigen History bezeichnet.

**Reads-From-Beziehung.**

$$T_i \leftarrow T_j$$

Eine Transaktion  $T_i$  liest von einer Transaktion  $T_j$  falls

1.  $T_i$  liest  $x$ , nachdem  $T_j$   $x$  geschrieben hat;
2.  $T_j$  abortet nicht, bevor  $T_i$   $x$  liest;
3. jede andere Transaktion, die  $x$  in der Zeit zwischen  $w_j[x]$  und  $r_i[x]$  schreibt, abortet vor  $r_i[x]$ ;

**Committed Projection.** Die committed projection einer History  $C(H)$  resultiert aus  $H$  durch Löschen aller Operationen, die nicht committed sind.

**Konfliktrelation.** Die Konfliktrelation einer History  $H$  ist die Menge der nach Ausführungsreihenfolge geordneten Paare von konfliktierenden Operationen.

$$KR(H) = \{(o <_H p) : o, p \in H, o \not\parallel p\}$$

**Konfliktäquivalenz.** Die Histories  $H$  und  $H'$  sind konfliktäquivalent, falls sie die gleichen Operationen enthalten und die Konfliktrelationen von  $C(H)$  und  $C(H')$  identisch sind.

**Cascading Abort**

## 4.2 Anomalien

**Lost Update** Update geht verloren, da es von einer anderen Transaktion überschrieben wird

$$r_1[x] < r_2[x] < w_2[x] < w_1[x]$$

**Dirty Read** Datenobjekt wird in einem inkonsistenten Zustand gelesen

$$r_1[x] < w_1[x] < r_2[x] < w_2[x] < c_2 < a_1$$

**Non-Repeatable Read** Leseergebnis nicht wiederholbar, weil andere Transaktion das Datenobjekt zwischenzeitlich geändert hat.

$$r_1[x] < r_2[x] < w_2[x] < r_1[x]$$

**Phantom Read** entspricht Non-Repeatable Reads auf Mengen statt Werten: Während einer Transaktion wiederholte gleiche Anfragen ergeben unterschiedliche Ergebnismengen, da andere Transaktion die Relation geändert haben

## 4.3 Eigenschaften von Histories

**Prefix Commit-Closed** Eine Eigenschaft  $\alpha$  einer History  $H = o_1 \dots o_n$  heißt prefix-commit closed, falls  $\alpha$  auch für jedes Präfix  $H' = o_1 \dots o_k, k < n$  von  $H$  gilt.

## 4.4 Konfliktserialisierbarkeit

**Konfliktgraph.** Zu einer History  $H$ , an der mehrere Transaktionen  $\mathcal{T} = \{T_1, \dots, T_n\}$  beteiligt sind, gibt es einen Konfliktgraphen  $G_K(H) = (V \subseteq \mathcal{T}, E \subseteq \mathcal{T} \times \mathcal{T})$ . Zur Erstellung des Konfliktgraphen betrachtet man den Konfliktrelation von  $H$ , eingeschränkt auf diejenigen Konflikte, die zwischen Operationen aus verschiedenen Transaktionen bestehen.

$$KRT(H) = \{(o <_H p) \in KR(H) : o \in T_i, p \in T_j, i \neq j\}$$

Für jeden Konflikt  $(o_i <_H p_j)$  mit  $o \in T_i$  und  $p \in T_j$  wird in den Konfliktgraph eine gerichtete Kante  $(T_j, T_i)$  eingefügt. Diese Kante kann als die Beziehung “ $T_j$  hängt ab von  $T_i$ ” verstanden werden.

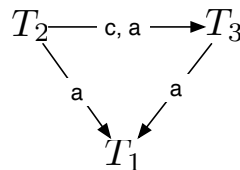


Figure 2: Beispiel eines Konfliktgraphen

**konfliktserialisierbar** ist eine History  $H$

- wenn der Konfliktgraph  $G_k(H)$  azyklisch ist *oder*
- wenn für eine serielle History  $H_s$  gilt:  $C(H)$  und  $C(H_s)$  sind konfliktäquivalent.

**sichtserialisierbar** wenn der zugehörige Konfliktgraph azyklisch ist

**Recoverability.** Um Recoverability zu gewährleisten darf eine Transaktion erst dann committet werden, wenn alle Transaktionen, von denen sie gelesen hat, bereits committet sind. Es muss gelten

$$T_i \leftarrow T_j \wedge c_i \in H \Rightarrow c_j <_H c_i$$

Wenn dies für eine History  $H$  zutrifft schreibt man  $H \in RC$ .

**Cascadelessness / Avoids Cascading Aborts.** Um Cascadelessness zu gewährleisten und Cascading Aborts zu vermeiden, darf jede Transaktion nur von zuvor committeten Transaktionen lesen. Damit  $H \in ACA$  ist muss gelten

$$T_i \leftarrow T_j \Rightarrow c_j < r_i[x]$$

Cascadelessness ist eine Einschränkung von Recoverability:

$$ACA \subset RC$$

**Strictness.** Um Strictness zu gewährleisten dürfen geschriebene Daten einer noch laufenden Transaktion nicht geschrieben oder gelesen werden. Damit  $H \in ST$  ist muss gelten

$$w_j[x] < o_i[x] (i \neq j) \Rightarrow c_j < o_i[x] \wedge a_j < o_i[x]$$

Strictness ist eine Einschränkung von Cascadelessness:

$$ST \subset ACA$$

	$H \in RC$	$H \notin RC$
$H$ konfliktserialisierbar	ja	nein
$H$ nicht konfliktserialisierbar		nein

Table 1: Korrektheit

**Korrektheit.**

**ACID-Eigenschaften**

**Atomicity**

## 4.5 Locking

**Serielle Ausführung.**



## 5 Logische Anfrageoptimierung

### Grundsätze

- Selektion so früh wie möglich
- Entfernung redundanter Operationen, Idempotenzen und leerer Zwischenrelationen
- Zusammenfassung gleicher Teilausdrücke
- Basisoperationen zusammenfassen