Datenbanksysteme WS 1999/2000

Übung 7 — Beispiellösung zu Aufgabe 4

1. - Anzahl der Indizes über die konkatenierten Attribute $Ai_1, \ldots, Ai_j$:
     Wir können annehmen, daß der Parser der Query die spezifizierten Attribute jeweils in eine "kanonische" (z.B. sortierte) Reihenfolge überführen kann. Daher brauchen wir für eine Partial–Match–Query über j–Attribute nur alle j–elementigen Teilmengen zu betrachten.
     - Im Spezialfall $(k = 3, m = 2)$ erhalten wir somit die Indizes:

       $$A_1 \ , \ A_2 \ , \ A_3 \ , \ A_1 \circ A_2 \ , \ A_1 \circ A_3 \ , \ A_2 \circ A_3 \ , \ A_1 \circ A_2 \circ A_3$$

       $\Rightarrow$ Es sind 7 verschiedene Indizes möglich.
     - Und für den allgemeinen Fall:

       $$\#Indizes = \sum_{i=1}^{k} \binom{k}{i} = \sum_{i=0}^{k} \binom{k}{i} - \binom{k}{0} = 2^k - 1$$

   - Gesamtanzahl der TID-Listen:
     - Für den Spezialfall gilt:

Index	#(TID-Listen)
A_1	$2 = 2^1$
A_2	$2 = 2^1$
A_3	$2 = 2^1$
$A_1 \circ A_2$	$4 = 2^2$
$A_1 \circ A_3$	$4 = 2^2$
$A_2 \circ A_3$	$4 = 2^2$
$A_1 \circ A_2 \circ A_3$	$8 = 2^3$
$\sum$	26

     - Dies läßt sich verallgemeinern zu:

       $$\#(TID - Listen) = \sum_{i=1}^{k} \binom{k}{i} \cdot m^i = \sum_{i=0}^{k} \binom{k}{i} \cdot m^i \cdot 1^{k-i} - \binom{k}{0} = (m+1)^k - 1$$

   - Durchschnittliche Länge einer TID-Liste:

– Im Spezialfall:

Index	*Länge(TID – Liste)*	*#(TID – Listen)*
A_1	$10 + 4 \cdot 6 = 34$	2
A_2	$10 + 4 \cdot 6 = 34$	2
A_3	$10 + 4 \cdot 6 = 34$	2
$A_1 \circ A_2$	$20 + 2 \cdot 6 = 32$	4
$A_1 \circ A_3$	$20 + 2 \cdot 6 = 32$	4
$A_2 \circ A_3$	$20 + 2 \cdot 6 = 32$	4
$A_1 \circ A_2 \circ A_3$	$30 + 1 \cdot 6 = 36$	8
$\sum$	876	26

Durchschnittliche Länge(TID-Liste) $= \frac{876}{26} \approx 33,69$ Bytes

– Im allgemeinen:

$$\frac{\sum_{i=1}^{k} \binom{k}{i} m^i (10i + 6m^{k-i})}{\sum_{i=1}^{k} \binom{k}{i} m^i} = \frac{1}{(m+1)^k - 1} \sum_{i=1}^{k} \binom{k}{i} m^i (10i + 6m^{k-i})$$

2. Betrachten wir eine Anfrage der Form

$$Ai_1 = wi_1 \text{ AND } \ldots Ai_j = wi_j \text{ mit } \{i_1, \ldots, i_j\} \subseteq \{1, \ldots, k\}$$

so muß ein Index über $Al_1 Al_2 \ldots Al_j Al_{j+1} \ldots Al_k$ existieren, der die folgenden Bedingungen erfüllt:

$$\begin{aligned} \{Ai_1, \ldots Ai_j\} &= \{Al_1, \ldots Al_j\} \\ \{Ai_{j+1}, \ldots Ai_k\} &= \{Al_1, \ldots Al_k\} - \{Al_1, \ldots Al_j\} \end{aligned}$$

Wir sehen also, daß die Reihenfolge der Attribute nur in soweit wichtig ist, daß alle "spezifizierten" Attribute vor allen "nicht spezifizierten" kommen müssen. Daher kommen wir mit weniger als $k!$ vielen Indizes (alle möglichen Permutationen) aus.

- Für unseren Spezialfall erhalten wir somit:

Index	neu erreichte Prädikate
$A_1 \circ A_2 \circ A_3$	$A_1, A_1 \wedge A_2, A_1 \wedge A_2 \wedge A_3$
$A_2 \circ A_3 \circ A_1$	$A_2, A_2 \wedge A_3$
$A_3 \circ A_1 \circ A_2$	$A_3, A_3 \wedge A_1$

- Und im allgemeinen:

$$\#Indizes = \max_j \left\{ \binom{k}{j} \right\} = \binom{k}{\lfloor \frac{k}{2} \rfloor}$$

Nähere Informationen zur Berechnung der Anzahl der Indizes für den allgemeinen Fall findet man in:
Multi-attribute Retrieval with Combined Indexes
V.Y. Lum, IBM Research Labaratory, San Jose, California
Communications of the ACM, Volume 13, Number 11, November 1970

3. Aus der Tabelle in der 2. Teilaufgabe können wir erkennen, daß wir lediglich die folgenden Indizes benötigen, um alle Partial–Match–Queries über den Attributen A_1, A_2 und A_3 ausführen zu können:

$$A1 \circ A_2 \circ A_3, A_2 \circ A_3 \text{ und } A_3 \circ A_1$$

Index	$Länge(TID - Liste)$	$\#(TID - Listen)$
$A_1 \circ A_2 \circ A_3$	$30 + 1 \cdot 6$	8
$A_2 \circ A_3$	$20 + 2 \cdot 6$	4
$A_3 \circ A_1$	$20 + 2 \cdot 6$	4
$\sum$	544	16

Speicherplatzbedarf für Indizes aus Teilaufgabe 2:

$$3 \cdot 8 \cdot (3 \cdot 10 + 1 \cdot 6) = \mathbf{864\ Bytes}$$

Wir erkennen also, daß wir durch die Modifikation der Indizes eine Speicherplatzeinsparung von $(100 * (864 - 544))/864 \approx 37\%$ erreichen können.

4. • Anhand des folgenden Beispiels können wir erkennen, daß es nicht ausreicht variable lange Attribute zu konkatenieren und die resultierenden Strings zu vergleichen:

Beispiel 1:

	Attribut A_1	Attribut A_2
Tupel 1	a	baa
Tupel 2	aa	a

Betrachten wir nämlich nur die Werte für das Attribut A_1 so gilt, daß Tupel 1 lexikographisch kleiner ist als Tupel 2, während bei den konkatenierten Strings gilt, daß Tupel 1 größer ist als Tupel 2.

• Gehen wir nun davon aus, daß wir ein ausgezeichnetes Zeichen ('\0') besitzen, das kleiner ist als alle anderen Zeichen unseres Alphabets, und das selbst nicht als Zeichen in Attributen vorkommen darf, so können wir folgende Methode benutzen um die Teilstrings zu konkatenieren:

Zwischen die Teilstrings der verschiedenen Attribute fügen wir jeweils das Zeichen '\0' als Trennzeichen ein.

Für das Beispiel 1 erhalten wir dann:

Tupel 1: a \0 baa
Tupel 2: aa \0 a

und wir erkennen, daß nach dem Konkatenieren der Teilstrings die lexikographische Ordnung erhalten bleibt.

Besitzen wir jedoch kein Zeichen, daß obige Anforderungen erfüllt (d.h. jedes Zeichen darf in den Attributen vorkommen), so können wir leicht ein Gegenbeispiel konstruieren:

Beispiel 2:

	Attribut A_1	Attribut A_2
Tupel 1	a\0	a
Tupel 2	a	\0ba

- Nehmen wir nun wieder an, daß \0 kein Element unseres Alphabets ist, so können wir eine weitere Methode zur Kodierung beschreiben:

  > *Wir bestimmen jeweils die maximale Länge für jedes Attribut und füllen kürzere Attribute jeweils mit dem* low-value *'\0' auf diese maximale Länge auf.*

  Betrachten wir wiederum Beispiel 1, so erhalten wir jetzt unter der Annahme, daß $\max|A_1| = 2$ und $\max|A_2| = 3$:

Tupel 1:	a\0	baa
Tupel 2:	aa	a\0\0

  Auch hier ist es wieder essentiell, daß der benutzte low–value nicht zum Alphabet gehört, wie wir uns an folgendem Beispiel klar machen können:

Beispiel 3:

	Attribut A_1	Attribut A_2
Tupel 1	a\0	aa
Tupel 2	a	ba

- *Kombinieren* wir die beiden obigen Verfahren, so erhalten wir eine Methode, die wir auch ohne ein separates Trennzeichen benutzen können.

  > *Zuerst füllen wir wiederum die einzelnen Attribute mit dem kleinsten Wert unseres Alphabets bis zur jeweils maximalen Länge auf. Sei diese maximale Länge für Attribut A_i gleich L_i und sei die Anzahl der eingefügten low–values für Tupel j gleich n_{ij}, so hängen wir an dieses Attribut die Zahl $L_i - n_{ij}$ an.*

  Für Beispiel 2 erhalten wir dann:

Tupel 1:	a\0	2	a\0\0	1
Tupel 2:	a\0	1	\0ba	3

  und für Beispiel 3:

Tupel 1:	a\0	2	aa\0	2
Tupel 2:	a\0	1	ba\0	2

  Variieren die Attributlängen sehr stark, so verbrauchen wir durch das Auffüllen sehr viel unnötigen Platz. Eine Möglichkeit das abzuschwächen wird in:

  Theo Härder: *Implementierung von Datenbanksystemen*,
  Hanser–Verlag, S. 94-98

  beschrieben.