Prof. Dr.-Ing. Dr. h. c. T. Härder
Fachbereich Informatik
Arbeitsgruppe Datenbanken und Informationssysteme
Universität Kaiserslautern

Übungsblatt 6 – Lösungsvorschläge

Unterlagen zur Vorlesung: „wwwdvs.informatik.uni-kl.de/courses/DBSREAL/“

Aufgabe 1: Darstellung von Sätzen - Extremlösung

Es soll eine Speicherungsstruktur für eine Satzmenge mit ≥ 200 Feldern/Satz entwickelt werden, von denen ggf. viele NULL-Werte besitzen können. Da der Software-Hersteller seine Anwendungssoftware weltweit vertreibt und den Kunden eine freie Wahl aus k ≤ 6 relationalen DBMS gestattet, muss die Struktur in allen DBMS gleichermaßen abbildbar sein. Jeder Satz oder mindestens jedes Satzfragment muss in einer Seite speicherbar sein, d.h. $S_l \leq L_S - L_{SK}$. Außerdem sollen sich die Felder indexieren lassen. Beliebige dynamische Erweiterungen mit neuen Feldern sind wünschenswert.

Als Beispiel wählen wir die Tabelle Pers mit den herkömmlichen Spalten (Attributen) Pnr, Vorname, Name, Beruf, Alter usw. Auf die einzelnen Felder wird mit unterschiedlicher Häufigkeit zugegriffen. Auf der DB-Schemaebene kann Pers natürlich vertikal partitioniert werden. Dann entstehen aber mehrere Tabellen, die bei DB-Operationen explizit angesprochen werden müssen.

a) Es werde eine Tabellenstruktur für die Sätze von Pers gewählt. Welche Speicherungsoptionen sind sinnvoll, wenn auch darauf geachtet werden muss, dass ein Satz (Satzfragment) immer in eine Seite passt?

   Einführung von Satzfragmenten, d. h. Aufspaltung der gespeicherten Sätze, und deren Kettung auf Feldebene; Zuordnung der Felder nach Zugriffshäufigkeiten zu den Satzfragmenten (Primary, Secondary, ...); wird möglicherweise nicht von allen k DBMS unterstützt.

b) Wenn eine Tabellenstruktur gewählt wurde, wie kann man dann auf die Programmierer zugreifen, die älter als 50 Jahre sind und den Vornamen Xaver haben?

```
Select   *
From     Pers
Where    Vorname = 'Xaver'
And      Beruf = 'Programmierer'
And      Alter > 50
```

c) Welche Systemunterstützung ist möglich, um diesen Anfragetyp so schnell wie möglich auswerten zu können?

Indexdefinition für Vorname, Beruf und Alter

d) Da Erweiterbarkeit, NULL-Werte und Abbildbarkeit auf k DBMS ein großes Problem bleiben, untersuchen Sie die extreme Form einer Speicherungsstruktur mit AOW (Attribut von Objekt ist Wert), wobei in einer AOW-Tabelle prinzipiell nur solchermaßen aufgebaute Tripel gespeichert werden. Die O-Spalte enthalte systemweite eindeutige OIDs als Werte.
Wie sieht die Speicherungsstruktur für einen Satz dieser Tabelle aus?

A	O	W
Vorname	XYZ0815	Xaver

e) Formulieren Sie die vereinfachte obige Anfrage auf der AOW-Tabelle, wobei nur das Prädikat (Vorname = ‘Xaver’) verwendet wird. Die Ausgabe kann zunächst als Sub-Tabelle von AOW spezifiziert werden.

```
Select      T1.A, T1.O, T1.W
From        AOW   T1
Where       T1.O IN (
   Select       T2.O
   From         AOW T2
   Where        T2.A = ‘Vorname’
   And          T2.W = ‘Xaver’
   And          T2.O = T1.O
)
```

oder

```
Select      *
From        AOW   T1
Where       T1.O IN (
   Select       T2.O
   From         AOW T2
   Where        T2.A = ‘Vorname’
   And          T2.W = ‘Xaver’
)
```

oder

```
Select      T1.A, T1.O, T1.W
From        AOW   T1, AOW T2,
Where       T2.A = ‘Vorname’
   And          T2.W = ‘Xaver’
   And          T2.O = T1.O
```

oder

```
Select      T1.A, T1.O, T1.W
From        AOW   T1
```

```
Where    Exists (
  Select     *
  From       AOW T2
  Where      T2.A = 'Vorname'
  And        T2.W = 'Xaver'
  And        T2.O = T1.O
)
```

In der Ergebnis-Tabelle stehen die Tripel, die zum selben Xaver-Satz (von potentiell mehreren Xaver-Sätzen) gehören, noch untereinander. Mit einem OID-Join müssten diese Werte zu langen Sätzen (≥ 200 Feldern) verknüpft werden.

f) Wie lautet die Anfrage auf AOW mit (Vorname = 'Xaver' And Beruf = 'Programmierer' And Alter > 50), die das äquivalente Ergebnis zur Anfrage auf Pers erzielt? Da das in SQL sehr aufwendig ist, schränken Sie die Ausgabe auf die explizit aufgelisteten Attribute Pnr, Vorname, Name, Beruf und Alter ein.

```
Select   P.W, Vor.W, Nam.W, Ber.W, Alt.W, ...
From     AOW P,
         AOW Vor,
         AOW Nam,
         AOW Ber,
         AOW Alt,
         . . .
Where    P.O = Vor.O      And P.A = 'Pnr'        And Vor.A = 'Vorname'
  And    P.O = Nam.O                             And Nam.A = 'Name'
  And    P.O = Ber.O                             And Ber.A = 'Beruf'
  And    P.O = Alt.=O              And Alt.A = 'Alter'
         . . .
  And    Vor.W = 'Xaver'
  And    Ber.W = 'Programmierer'
  And    Alter.W > 50
```

g) Wie wird dynamische Erweiterbarkeit gelöst?

Dynamische Erweiterbarkeit (Schemaevolution) ist eingebaut. Es ist also nur ein Insert mit dem entsprechenden AOW-Satz erforderlich und dies kann für jeden Pers-Satz zum Modifikationszeitpunkt erfolgen.

h) Die bisherige Darstellung der W-Spalte war stark vereinfacht, da ja Werte verschiedenen Typs in dieser Spalte auftreten. Das wird u. a. auch zum Problem, wenn man auf den Werten (vom gleichen Typ) Indexstrukturen einführen will. Wie kann dieses Problem gelöst werden?

Die Darstellung von W wird künstlich auf W1 bis W5 erweitert, wobei den Wi die Typen Integer, Float, Money, Varchar und Own zugewiesen werden (in der Realisierung des erwähnten Software-Herstellers). In einem Satz ist nur der zutreffende Wert belegt. Die restlichen Wi enthalten NULL-Werte.

In der A-Spalte werden nicht die vollen Attributnamen gespeichert, sondern systeminterne Kürzel, also beispielsweise

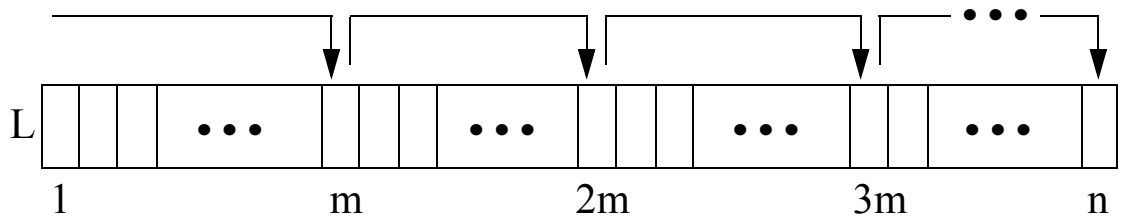
A	O	W1	W2	W3	W4	W5
A15	XYZ0815	NULL	NULL	NULL	Xaver	NULL

Jetzt sind die Wi-Spalten homogen und es können Indexstrukturen darauf definiert werden.

Diese Technik wird bei einem großen Software-Hersteller angewendet, um Generalisierungshierarchien (mit multipler Vererbung) in relationalen DBMS zu repräsentieren. Dabei kommen sehr viele Attribute vor, wobei häufig dynamische Erweiterungen anfallen. Die Struktur der Generalisierungshierarchie ist in den (sehr langen) Werten der OIDs codiert.

Aufgabe 2: Suche in einer Liste - Sprungsuche

Gegeben ist eine Liste L mit n Einträgen fester Länge. Die Sprungsuche soll auf Liste L optimiert werden, wobei folgendes Schema angenommen wird:



a) Bei der einfachen Sprungsuche erfolgen konstante Sprünge zu den Positionen m, 2m, 3m usw. Sobald der Abschnitt, in den der gesuchte Schlüssel fällt, lokalisiert ist, wird sequentiell gesucht. Welche mittleren Suchkosten $C_{avg}(n)$ ergeben sich, wenn ein Sprung a und ein sequentieller Vergleich b Einheiten kostet?

$$C_{avg}(n) = \frac{1}{2}a \cdot \frac{n}{m} + \frac{1}{2}b(m-1)$$

b) Was sind die Kosten $C_{avg}(n)$ bei optimaler Sprungweite m?

$$C_{avg}(n) = \quad a\sqrt{n} - a/2$$

mit optimaler Sprungweite $m = \sqrt{(a/b)n}$ bzw. $m = \sqrt{n}$ falls a=b

c) Bei der Zwei-Ebenen-Sprungsuche wird statt sequentieller Suche im lokalisierten Abschnitt wiederum eine Quadratwurzel-Sprungsuche angewendet, bevor dann sequentiell gesucht wird. Welches $C_{avg}(n)$ ergibt sich mit

a = Kosten eines Sprungs auf der ersten Ebene,
b = Kosten eines Sprungs auf der zweiten Ebene,
c = Kosten für einen sequentiellen Vergleich?

$$C_{avg}(n) \le \frac{1}{2} \cdot a \cdot \sqrt{n} + \frac{1}{2} \cdot b \cdot n^{\frac{1}{4}} + \frac{1}{2} \cdot c \cdot n^{\frac{1}{4}}$$

Für a = b = c ergibt sich: $C_{avg}(n) \le a\left(\frac{1}{2}\sqrt{n} + n^{\frac{1}{4}}\right)$

d) Welche Verbesserungen ergeben sich durch optimale Abstimmung der Sprungweiten m_1 und m_2 der beiden Ebenen, wenn a = b = c gesetzt wird?

$$m_1 = n^{\frac{2}{3}} \quad \text{und} \quad m_2 = n^{\frac{1}{3}}$$

$$C_{avg}(n) = \frac{3}{2} \cdot a \cdot n^{\frac{1}{3}}$$

e) Lässt sich die Effizienz der Suche steigern, wenn das Verfahren zu einem n-Ebenen-Verfahren verallgemeinert wird?

Verallgemeinerung zu n-Ebenen-Verfahren ergibt ähnlich günstige Kosten wie Binärsuche (Übereinstimmung bei $\log_2 n$ Ebenen)

Aufgabe 3: Aufbau eines TRIE

a) Erstellen Sie einen Trie mit variablem Knotenformat für die Schlüsselfolge

OTTO, HUGO, OTTI, OLLI, EDDI, ELLI, HASI, HANS, ELSE, ULLI, ILSE.

b) Ergänzen Sie den Trie aus a) mit den Schlüsseln
OTT, HU, OTTILIE, OTTOKAR, ILSEBILL.

selbsterklärend (siehe Folien)

Aufgabe 4: PATRICIA-Baum

Erstellen Sie einen PATRICIA-Baum für die Schlüsselfolge aus Aufgabe 3a. Verwenden Sie folgende Kodierung der Buchstaben (ASCII):

A =	0100 0001	D =	0100 0100	E =	0100 0101
G =	0100 0111	H =	0100 1000	I =	0100 1001
L =	0100 1100	N =	0100 1110	O =	0100 1111
S =	0101 0011	T =	0101 0100	U =	0101 0101

selbsterklärend (siehe Folien)

Aufgabe 5: Analyse einer Hash-Funktion

Die Schlüssel zur Identifikation der Datensätze einer Anwendung seien als CHAR(8) im EBCDIC-Code definiert. Sie werden typischerweise als Zahlen in festen Abständen, also 10, 20, 30, ..., vergeben. Vor Verwendung in einer Hash-Funktion werden diese Schlüssel erst nach folgendem Schema gefaltet und dabei mit EXOR verknüpft. In hexadezimaler Darstellung (im EBCDIC-Code) ergibt sich beispielsweise nach Faltung für den Schlüssel S_1 mit 0000 0010 der Wert von K_1 mit

$$\begin{array}{rcl} & \begin{array}{cccccccc} F & 0 & F & 0 & F & 0 & F & 0 \\ F & 0 & F & 0 & F & 1 & F & 0 \end{array} & \oplus \\ \hline K_1 = & 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0_{16} & \triangleq\ 256_{10} \end{array}$$

Danach werde das Divisionsrest-Verfahren mit

$$h(K_i) = K_i \bmod n$$

angewendet.

Analysieren Sie das Kollisionsverhalten dieser Funktion bei obiger Schlüsselvergabe und einem Hash-Bereich von n = 576 Buckets.

a) Nach welchen Abständen von j vergebenen Schlüsseln, die sich jeweils mit einem Inkrement von Δ k (was dem Wert von K_1 entspricht) erhöhen, treten Kollisionen auf?

Kollision, wenn

$$K_i \bmod n = K_j \bmod n = (K_i + t \cdot n) \bmod n, \qquad t = 1, 2, 3, ...$$

Schlüsselvergabe sei $K_j = K_i + j \cdot \Delta k$, $j = 1, 2, 3, ...$

Kritisch, wenn

$j \cdot \Delta k = t \cdot n$

Nach welchem Abstand $j \cdot \Delta k$ ergibt sich eine Kollision?

$j = (t \cdot n)/\Delta k$, $n = 576_{10} = 2^6 \cdot 3^2$

$\Delta k = 100_{16} = 256_{10} = 2^8$

$j = (t \cdot 576)/256 = (t \cdot 2^6 \cdot 3^2)/2^8 = (t \cdot 3^2)/ 2^4$

$j = 9$ mit bei einem minimalen $t = 4$, da j ganzzahlig ist.

b) Was ist eine effektive Abhilfemaßnahme?

Für die Größe des Hash-Bereichs sollte eine Primzahl gewählt werden. Dann ergeben sich keine gemeinsamen Teiler von Schlüsselinkrement Δk und n, so dass die Kollisionsabstände maximal werden.

Aufgabe 6: Erweiterbares Hashing 18

Die Sätze mit den Schlüsseln K10, J84, K35, A12, E77, K12, B22, K08 sollen in der angegebenen Reihenfolge mit dem Verfahren des Erweiterbaren Hashing in Buckets abgespeichert werden, die jeweils 2 Sätze aufnehmen können. Den Pseudoschlüssel des jeweiligen Satzes erhält man durch Verknüpfung der in EBCDIC codierten Zeichen mittels XOR. Zeichnen Sie die Belegung der Buckets und das Directory nach jeder Einfügung.

Schlüsselfolge: K10, J84, K35, A12, E77, K12, B22, K08

EBCDIC-Codierung: $J \equiv D1_{16}$ $B \equiv C2_{16}$ $K \equiv D2_{16}$ $E \equiv C5_{16}$ $A \equiv C1_{16}$

K10	J84	K35	A12
F0	F4	F5	F2
F1	F8	F3	F1
01	0C	06	03
D2	D1	D2	C1
D3	DD	D4	C2
1101 0011	1101 1101	1101 0100	1100 0010

E77	K12	B22	K08
F7	F2	F2	F8
F7	F1	F2	F0
00	03	00	08
C5	D2	C2	D2
C5	D1	C2	DA
1100 0101	1101 0001	1100 0010	1101 1010