UNIVERSITÄT DES SAARLANDES
Fachbereich 14– Informatik
Prof. Dr.–Ing. G. Weikum
Dipl.–Inform. R. Schenkel

Datenbanksysteme WS 1999/2000

Übung 7 — Abgabe: 20.12.1999

Aufgabe 1: Zugriffsstrukturen

Gegeben sei die Musikdatenbank aus Übung 1. Die Schemadefinition, die angenommenen Kardinalitäten der einzelnen Relationen sowie die mittleren Tupellängen in Bytes sind in der folgenden Tabelle aufgeführt.

Relation	**Schema**		**Anzahl von Tupeln**	**Mittlere Tupellänge (Bytes)**
Disk	DiskID DiskTitel	NUMBER PRIMARY KEY NOT NULL VARCHAR(255)	1000	100
Musik-stueck	DiskID StueckID Titel	NUMBER NUMBER VARCHAR(255)	10000	50
Person	PID Name Nationalitaet	NUMBER PRIMARY KEY NOT NULL VARCHAR(40) NOT NULL CHAR(5)	6000	40
Autor	PID DiskID StueckID Funktion	NUMBER NOT NULL NUMBER NOT NULL NUMBER NOT NULL VARCHAR(20) NOT NULL	30000	40

Zur Effizienzsteigerung von Anfragen sind unter anderem die folgenden mittelbaren Indizes angelegt, die im Datenbanksystem jeweils als B*–Baum implementiert seien:

```
CREATE UNIQUE INDEX DiskIdIndex ON Disk(DiskID);
CREATE INDEX DiskTitelIndex ON Disk(DiskTitel);
CREATE INDEX PidIndex ON Autor (PID);
```

Außerdem sind folgende Parameter/Einstellungen bekannt:

- Jede Relation und jeder Index belegt jeweils einen eigenen Tablespace.
- Die Seitengröße sei 4 KBytes, die Länge des Seiten–Headers sei 96 Bytes.
- Der Füllgrad von Datenseiten sei 90%, von Indexseiten 70%.
- Der Datentyp NUMBER belege 10 Bytes.
- *Disktitel* sei Schlüsselkandidat der Relation *Disk* und habe eine durchschnittliche Länge von 90 Bytes.

- Ein TID oder Zeiger im B*–Baum habe die Länge 6 Bytes.
- Alle Werte eines Fremdschlüsselattributes kommen jeweils gleichhäufig vor. (Gleichverteilungsannahme).

Analysieren Sie die Speicherplatz– und Indexzugriffskosten dieser Datenbank. Vernachlässigen Sie dabei den Platzbedarf für die Slotlisten und für die Längenfelder des Datentyps VARCHAR.

Aufgabe 2: B*–Baum

(a) Fügen Sie in einen zu Beginn leeren B*–Baum vom Typ *m=2, m*=2* der Reihe nach die folgenden Schlüssel ein: *13, 97, 45, 23, 103, 84, 100, 99, 50, 70, 55, 47, 52, 54*. Welches Aussehen hat der Baum nach Ausführung dieser Einfüge–Operationen ? Löschen Sie anschließend die Schlüssel *70* und *84*. Überlegen Sie sich dazu eine geeignete Reorganisation beim Löschen.

(b) Die Speicherplatzausnutzung eines B*–Baums beträgt – von der Wurzel abgesehen – mindestens 50%. Bei zufälligen Einfügungen beträgt sie im Mittel $\ln(2) \approx 69\%$.

- Unter welchen Bedingungen wird – auch bei einer großen Anzahl von Einfügungen (also auch asymtotisch) – die schlechteste Speicherplatzausnutzung von 50% erreicht?
- Wie kann man die Mindestspeicherplatzauslastung (und damit auch die mittlere Speicherplatzauslastung) generell verbessern, d.h. auch für den normalen Betrieb des B*–Baumes?

(c) Bei der Entwicklung eines neuen Datenbanksystems hat der Chef des Designteams (der leider nicht in Saarbrücken studiert hat) die Idee, zur Unterstützung von Operationen auf B*–Bäumen für jeden Knoten einen Verweis zum Vater einzuführen.

- Bei welchen Operationen auf B*–Bäumen scheint dieser zusätzliche Verweis sinnvoll zu sein?
- Warum ist die Idee des Systementwicklers letztlich aber doch alles andere als genial?

(d) Die inneren Knoten eines B*–Baumes sind darauf ausgelegt, möglichst viele Einträge zu enthalten, um den Verzweigungsgrad zu maximieren. Sie erreichen oft Größenordnungen von mehreren Hundert Einträgen pro Knoten und damit pro Diskblock. Der CPU–Aufwand, der nötig ist, um den Zeiger zu finden, dem der Suchalgorithmus bei gegebenem Suchschlüssel folgen muß, ist deshalb nicht unbedingt vernachlässigbar. Welche Verfahren würden Sie vorschlagen, um diese Suche zu implementieren? Unterscheiden Sie dabei zwischen Schlüsseln fester Länge (z.B. Typ NUMBER) und Schlüsseln variabler Länge (z.B. VARCHAR(n)).

Aufgabe 3: Kostenschätzung für Indizes

Betrachten Sie einen als B*–Baum realisierten Textindex über 10 Mio. Dokumente. Pro Dokument gebe es 100 Suchwörter, die zu indizieren sind; insgesamt gebe es 1 Mio. verschiedene Suchwörter, die alle gleichhäufig vorkommen.

(a) Wie groß (Anzahl der Blattseiten und der Blattknoten, Höhe) wird der B*–Baum? Wieviele Indexseitenzugriffe sind notwendig, um alle Dokumente zu bestimmen, die ein bestimmtes Suchwort enthalten?
Gehen Sie von einer Nettoseitengröße von 4000 Bytes aus, und nehmen Sie einen Seitenfüllgrad von 70 Prozent an. Die Länge eines Suchworts sei konstant 30 Bytes, die Länge

eines TIDs (bzw. Zeigers auf ein Dokument, z.B. auch eine URL) sei 10 Bytes, und die Länge eines Sohnzeigers innerhalb des Baums sei 4 Bytes.

Hinweise:

– Ein Blattknoten kann aus mehreren miteinander verketteten Blattseiten bestehen.

– Ein Baum der Höhe 1 besteht nur aus der Wurzel.

(b) Wie verändert sich die Lösung zu a), wenn Sie in den Nichtblattseiten des Index eine Schlüsselkomprimierung durchführen, bei der die Suchwörter innerhalb der Wegweiser (konstant) um den Faktor 2 verkürzt werden?

Aufgabe 4: Indizes für Partial–Match–Queries

Gegeben ist eine Relation $R(A_1, ... ,A_k)$, deren Attribute alle denselben Domain D mit $|D| = m$ haben. Ein Attributwert benötige jeweils $b=10$ Bytes Speicherplatz. R enthalte alle m^k möglichen Tupel.

Bearbeiten Sie die folgenden Teilaufgaben mindestens für den Spezialfall $k = 3, m = 2$.

1. Stellen Sie sich vor, es gäbe eine formularorientierte Schnittstelle, in der der Benutzer für jedes beliebige Attribut aus $A_1, ... ,A_k$ einen entsprechenden Wert $w_1, ... ,w_k$ eintragen kann. Nach Fertigstellung des 'Formulars' wird für die j eingegebenen Werte eine Partial–Match–Query der Form

   $A_{i1} = w_{i1}$ AND $A_{i2} = w_{i2}$ AND ... AND $A_{ij} = w_{ij}$ mit $\{i1, ..., ij\} \subseteq \{1, ..., k\}$

   an das Datenbanksystem gestellt. Sie möchten jede mögliche Query durch einen Index unterstützen. Die Indizes seien alle mittelbar, d.h. für jeden Schlüsselwert gibt es eine Liste von TIDs, die alle entsprechenden Tupel referenzieren.

   – Wieviele Indizes benötigen Sie für $k=3$ Attribute bzw. für den allgemeinen Fall, wenn für jede mögliche Konkatenation der Attribute ein Index existieren soll ?

   – Wieviele TID–Listen werden insgesamt benötigt? Überlegen Sie sich dazu, wieviele TID–Listen es in den entsprechenden Indizes geben wird. Betrachten Sie wieder den Spezialfall *(k=3, m=2)* und den allgemeinen Fall.

   – Wie lange sind die TID–Listen im Mittel, wenn die Länge einer TID 6 Bytes betrage? Betrachten Sie wieder den Spezialfall *(k=3, m=2)* und den allgemeinen Fall.

2. Nutzen Sie nun aus, daß ein Index auch für beliebige Präfixe der Schlüsselwerte benutzt werden kann. Sofern jeder einzelne Attributwert innerhalb einer Konkatenation eindeutig identifiziert werden kann, kann man diese Eigenschaft dazu ausnutzen, um nicht für jede mögliche Konkatenation der Attribute einen eigenen Index erzeugen zu müssen. Wieviele Indizes benötigen Sie nun für alle k Attribute?

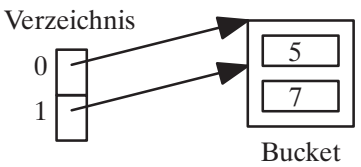
3. Welche Indizes müssen mindestens erzeugt werden, um alle möglichen Partial–Match–Queries beantworten zu können und um im Mittel einen möglichst minimalen Speicherplatzbedarf für die TID–Listen zu haben?

4. Bei einem Index für z.B. die konkatenierten Attribute $A_1 \circ A_2 \circ A_3$ muß bei der Suche eines Wertes $w_1 \circ w_2$ eindeutig identifiziert werden können, welcher Teil eines Eintrages im Index zu A_1 und welcher Teil zu A_2 gehört. Bei Attributen konstanter Länge kann ein Attribut einfach über die Position identifiziert werden. Überlegen Sie sich Möglichkeiten, die Konkatenation variabel langer Attributwerte zu realisieren.

Aufgabe 5: Erweiterbares Hashing

In der Vorlesung wurde das Erweiterbare Hashing (Extendible Hashing) vorgestellt. Ziel dynamischer Hash–Verfahren ist die dynamische Anpassung des Bildbereiches an wechselndes Datenvolumen, um die gute Suchperformance zu erhalten. Das Rehashing von Einträgen darf dabei jedoch den laufenden Betrieb nicht zu lange aufhalten.

Gegeben sei der folgende erweiterbare Hash–Index der globalen Tiefe *d=1*. Die Hash–Funktion sei *h(k)= k*. Die Anzahl der Buckets sei *N=1* und ein Bucket beinhalte maximal *2 Tupel*, die hier nur durch ihren Schlüsselwert angedeutet sind.



(a) Welches ist die lokale Tiefe der Einträge des Hash–Index?

(b) Fügen Sie die Werte *1, 10, 5 und (wieder) 5* in den Hash–Index ein.

(c) Warum ist es im Allgemeinen sinnvoller, für die Zuweisung eines Buckets die hinteren Bits des Hash–Ergebnisses *h(k)* zu verwenden, und nicht die vorderen Bits?