Datenbanken
Wintersemester 09/10
Prof. Dr. W. May

4. Übungsblatt: Physischer Entwurf, Interna, Optimierung

Besprechung voraussichtlich am 26.1.2010

Aufgabe 1 (Join-Algorithmen) Gegeben sind zwei Tabellenschemata R und S:

R		
A VARCHAR2(3)	B NUMBER	C VARCHAR2(20)

S		
E VARCHAR2(3)	F VARCHAR2(25)	G VARCHAR2(25)

Die Aufgabe behandelt das Join $R \bowtie_{R.A=S.E} S$.

Physikalisches Datenbankschema:

- Schlüssel von R ist (A, B), Schlüssel von S ist (E, F).
- Der Wertebereich von $R.A$ ist gleich dem Wertebereich von $S.E$, d.h. $\pi[A](R) = \pi[E](S)$.
- Die Tupel von R sind in beliebiger Reihenfolge gespeichert.
- Die Tupel von S sind so gespeichert, dass Tupel mit demselben E-Wert benachbart gespeichert sind. Eine weitergehende Ordnung existiert nicht.

Daten:

- R enthält 2.000.000 Tupel [Hinweis: diese Schreibweise bedeutet in Deutschland "2 Millionen"].
- S enthält 4.000.000 Tupel.
- Die Tabellen sind wie üblich seitenweise abgelegt. Eine Seite (4KB) von R enthält 100 Tupel, eine Seite von S enthält 60 Tupel.
- Zu jedem Wert a von A gibt es genau 2 Tupel $\mu \in R$, so dass $\mu[A] = a$ ist.
- Zu jedem Wert e von E gibt es durchschnittlich 4 Tupel $\mu \in S$, so dass $\mu[E] = e$ ist.
- Im Hauptspeicher können 1000 Seiten gleichzeitig gehalten werden.

Die Anfrage $\pi[A, B, F](R \bowtie_{R.A=S.E} S)$ soll berechnet und das Ergebnis ausgegeben werden.

A. Join unter Verwendung eines Indexes

- Nehmen Sie an, dass für das Attribut $S.E$ ein B*-Baumindex angelegt ist, dessen Einträge jeweils auf das erste Tupel mit dem entsprechenden Wert von $S.E$ zeigen.
- Für den B*-Baum über $S.E$ gilt folgendes: Jedes Blatt enthält durchschnittlich 100 Einträge. Jeder innere Knoten enthält 200 Einträge.

Fragen:

a) Wieviele Werte enthält $\pi[A](R)$?

b) Wieviele Blätter enthält der B*-Baum über $S.E$?

c) Wie viele Ebenen hat der Baum?
Wieviele Knoten enthält der Baum insgesamt?

d) Die Anfrage $\pi[A, B, F](R \bowtie_{R.A=S.E} S)$ soll berechnet und am Bildschirm ausgegeben werden. Wieviele Ergebnistupel bekommt man?

e) Beschreiben Sie, wie die Auswertung (unter Verwendung des Indexes) am besten vorgeht.
Wieviele Hintergrundspeicherzugriffe auf die Datenbank werden benötigt?
(beschreiben Sie am besten zuerst grob, wie Sie vorgehen, und analysieren Sie dann die Zugriffe)
(ingesamt 10P)

---

a) Es sind 1.000.000 (jeder Wert von $R.A$ kommt genau zweimal vor) Werte. (2P)

b) Aus $\pi[A](R) = \pi[E](S)$ folgt, dass es insgesamt 1.000.000 verschiedene Werte sind, also ebensoviele Einträge im Index. (wie in der Aufgabenbeschreibung angegeben soll jeweils auf das erste der hintereinander abgelegten Tupel mit demselben Wert von $S.E$ verwiesen werden). Damit benötigt man (100/Blatt) 10.000 Blätter.

c) 10.000 Blätter $\rightarrow$ $10.000/200 = 50$ Knoten auf erster Ebene, die unter einer gemeinsamen Wurzel hängen.
Der Baum hat damit 2 innere Ebenen sowie eine Ebene mit Blättern.
Insgesamt sind es 10000+50+1=10051 Knoten.

d) Jedes R-Tupel wird mit durchschnittlich 4 S-Tupeln kombiniert. Das Ergebnis enthält 8.000.000 Tupel.
Man benötigt keine Duplikateliminierung, da (A, B) Schlüssel von R ist und $(A = E, F)$ Schlüssel von S ist und somit keine Duplikate bei der Auswertung entstehen können.

e) Grober Ablauf: R durchgehen, für jeden Wert über den Index auf S zugreifen, Ergebnisse (durchschnittlich 4) nehmen, $\pi[A, B, F]$ darauf anwenden und dem Ergebnis hinzufügen.
Feinablauf:
Man liest erstmal die inneren Knoten des Index ein. 51 Zugriffe. Dieser wird immer wieder benötigt, also wird er im Hauptspeicher gehalten. Dann geht man R seitenweise durch (insgesamt 2.000.000/100=20.000). Für jedes Tupel wird auf den Baum über $S.E$ (im Hauptspeicher) zugegriffen, das entsprechende Blatt aus dem Hintergrundspeicher geholt, und damit dann auf S zugegriffen. Dort liegen die durchschnittlich 4 benötigten Tupel hintereinander. Bei 60 Tupeln pro Seite hat man durchschnittlich in jedem zwanzigsten Zugriff (Beginn bei Tupel 58,59,60) einen Sprung auf die nächste Seite. Damit muss man insgesamt $2,000,000 \cdot (1 + 1 + 0.05) = 4.100.000$ mal auf S-Seiten zugreifen. (Hinweis: da $\pi[A](R) = \pi[E](S)$ gilt, ist jeder gesuchte Wert mindestens einmal in S vorhanden.)
(Da S insgesamt 4.000.000/60=66.667 Seiten hat, kann man nicht davon ausgehen, dass "zufällig" die benötigte Seite mit den Baumblättern oder den Tupeln aus S gerade noch im Hauptspeicher liegt, in dem sich neben den permanenten 51 Index-Seiten noch die gerade verarbeiteten R-Seite befindet.
Da die Ergebnisse direkt ausgegeben werden sollen (und keine Duplikateliminierung notwendig ist), ist kein weiterer Zugriff zum Ablegen notwendig (Seitenweise Abspeicherung: Annahme dass 40 Tupel auf eine Seite passen, würde man weitere 200.000 Zugriffe haben).
Insgesamt also 51+20.000+4.100.000 = 4.120.051 Zugriffe.

B. Hash-Join Untersuchen Sie die Anwendung eines Hash-Joins über $R.A$ bzw. $S.E$ um $\pi[A, B, F](R \bowtie_{R.A=S.E} S)$ zu berechnen.

a) Untersuchen Sie zuerst den naiven Ansatz, dies zu tun.

b) Optimieren Sie die Anfrage zuerst algebraisch und überlegen Sie sich dann eine naheliegende Optimierung.

---

a) Hashen von R nach einer Hashfunktion $f(A)$.
f so gewählt, dass jede Partition 200 Seiten umfasst – also hat man ca. 100 Hashwerte (S muss genauso gehasht werden, und ist mehr als doppelt so gross. Nachher will man zusammenpassende Partitionen im Hauptspeicher halten koennen; bei erwarteten 200+666 Seiten (s.u.) ist man auf der sicheren Seite).
Man fängt also zu allen 100 Hashwerten eine Seite im Hauptspeicher an, und wenn diese voll ist, wird sie in den Hintergrundspeicher uebertragen und eine neue Seite begonnen. Da die letzte Seite jeder Hash-Partition durchschnittlich zu 50% voll ist, muss man damit rechnen, maximal 20100 Seiten zu benötigen.
Für R: 2.000.000 Tupel von 20.000 Seiten auf 20100 Seiten verteilen: 40100 Zugriffe. Dasselbe für S (60 Tupel pro Seite, insg. 66.667 Seiten), mit $f(E)$ ergibt 66766 Seiten : 133.434 Seitenzugriffe.
Dann die Buckets paarweise durchgehen, im Hauptspeicher miteinander verknüpfen, und Ergebnisse ausgeben. 20.100+66.766=86.866 Zugriffe. Ergebnis wird wieder sofort ausgegeben.
Summe: 40100 + 144343 + 86.866 = 271309 Zugriffe.

b) Die Anfrage kann zu $\pi[A,B,F](\pi[A,B](R) \bowtie_{R.A=S.E} S)$ umgeformt werden.
Die Projektion kann man beim Hashen durchführen, und so mehr Tupel pro Bucket-Seite ablegen.
Schätzung der Tupelgrößen:
Bei $R(ABC)$ passen 100 Tupel in 4096 KB. Platzbedarf eines Tupels ist maximal $1+3+1+6+1+20 = 32$ Byte (jeweils 1 Byte für die Länge des Varchar, dann die Zeichenkette selber). Der Rest ist Verwaltungsinformation der Seite [800 Byte].
Bei $S(EFG)$ passen 60 Tupel in 4096 KB. Platzbedarf eines Tupels ist maximal $1+3+1+25+1+25 = 56$ Byte, der Rest ist wieder Verwaltungsinformation der Seite [700 Byte].
Ein auf $R(A,B)$ projiziertes Tupel benötigt nur noch 11 Byte, womit man ca. 400 projizierte Tupel auf einer Seite speichern kann. Ein $S(E,F)$-Tupel benötigt 30 Byte, und man kann ca. 120 projizierte Tupel auf einer Seite speichern.
Beim Hashing werden also $5.000+100$ (Verschnitt) R_{hash}-Seiten und $33.333+100$ S_{hash}-Seiten gefüllt.
Dann die Buckets paarweise durchgehen, im Hauptspeicher miteinander verknüpfen, und Ergebnisse ausgeben. $5.100+33.433 = 38533$ Zugriffe. Ergebnis wird wieder sofort ausgegeben.
Summe: $20.000+5.100+66.667+33.433+(5.100+33.433) = 163733$ Zugriffe.

Aufgabe 2 (Optimierung, Aufwandsabschätzung (SQL)) Gegeben sei folgendes Datenbankschema (Auszug aus Mondial)

```
Country(Name, Code, Capital, Province, Area, Population)
ismember(Organization, Country, Type)
```

Formulieren Sie die folgenden Anfragen in SQL:

a) Welche Länder sind Mitglied in mehr als 60 Organisationen?

b) Welche Länder mit einer Fläche von mehr als 500000 km^2 sind Mitglied in mehr als 60 Organisationen?

Untersuchen Sie bei der 2. Teilaufgabe den Aufwand (Anzahl Hintergrundspeicherzugriffe, Vergleiche zur Ausführung des Join, Vergleiche zum Test der Fläche). An welchen Stellen wäre ein (Baum- oder anderer) Index nützlich? Untersuchen Sie auch für diesen Fall den Aufwand.

Können Sie Ihre Anfrage optimieren, um damit den Aufwand weiter zu reduzieren?

---

a) Welche Länder sind Mitglied in mehr als 60 Organisationen?

```
SELECT country
FROM ismember
GROUP BY country
HAVING count(*) > 60
```

b) Welche Länder mit einer Fläche von mehr als 500000 km^2 sind Mitglied in mehr als 60 Organisationen?

Naheliegendste Lösung: WHERE-Klausel, um vor der Gruppierung zu selektieren.

```
SELECT country
FROM ismember
WHERE (select area
       from country
       where code=ismember.country) > 500000
GROUP BY country
HAVING count(*) > 60
```

Aufwandsbetrachtung: Für jede der 7000 Mitgliedschaften wird überprüft, ob das betreffende Land grösser als 500000 km^2 ist.

Fall 1: kein Index auf country. Also wird linear die gesamte Tabelle country durchsucht. Aufwand: $8.000 \cdot 240 = 1.920.000$ Vergleiche von `code=ismember.country` und 8000 Flächentests.

Fall 2: Baum-Index auf country.code [Tafelbild]. Dann wird für jede der Mitgliedschaften über den Baum zielsicher auf das entsprechende country-Tupel zugegriffen und die Fläche getestet (8000 Tests) (Dasselbe gilt auch für einen Hash-Index).

Optimierung: der Flächentest wird z.B. für Deutschland 68 mal durchgeführt - für jede Mitgliedschaft einmal. Es wäre effizienter, *erst* nach "country" zu gruppieren, und den Test dann nur für jede Gruppe einmal durchzuführen:

```
SELECT country
FROM ismember
GROUP BY country
HAVING count(*) > 60
 and (select area
      from country
      where code=ismember.country) > 500000
```

Damit: 8000 Tupel aus ismember verhashen, 240 Gruppen, bei dann sequentieller Suche in country noch $240 \cdot 240 = 57600$ Vergleiche von `code=ismember.country` und 240 Flächentests, bei Vorhandensein eines Index nur noch Kosten fuer die Suche (Baum oder Hash) sowie nur noch 240 Flächentests.

Auswertung der Count-Bedingung:
Natürlich zuerst, wenn man die Gruppen gebildet hat. Es bleiben dann maximal $8000/60 = 133.33$, also 133 Gruppen (=Countries) übrig. Also bei Index auf `country.code` nur noch Kosten fuer die Suche sowie 133 Flächentests.

Weitere interne algorithmische Optimierung:
Bei der Auswertung von GROUP BY wird der Inhalt der ismember-Tabelle sowieso gruppiert (sortiert oder gehasht, oder zumindest ein passender Index erstellt).

Wenn man einen Index (Baum oder Hash) über country.code hat, führt man das GROUP BY über ismember passend durch und kann dann den area-Test auch als Merge ablaufen lassen, wobei parallel die Gruppen und die Blätter des Indexbaumes bzw. die Hash-Kacheln durchlaufen werden.

Und noch ein Schritt besser:
`ismember.country` ist ein foreign key, man kann also annehmen, dass darüber bereits ein Index

existiert, der die Gruppierung bereits übernimmt. Also kann man die Blätter des Index linear durchgehen, und erstmal für jedes Land schauen, wieviele Referenzen in die Basistabelle (= Mitgliedschaftseinträge) dort angelegt sind. Sind es mindestens 60, so behält man den Landescode, und greift damit direkt üeber den primary-key-Index von Country auf das entsprechende Tupel zu, schaut ob area > 500000 ist, und gibt den Landescode ggf. aus.

Aufwand: Der Index `ismember.country` besteht nur aus einem Wurzelknoten und vielleicht 20 Blätter-Seiten (die zusammen die 8000 Referenzen verteilt auf 240 Country-Einträge beinhalten). Die 240 Country-Einträge durchlaufen, davon bleiben < 133 (8000/60) übrig, für diese den Country.Code-Index durchsuchen (max. 2 Ebenen, 3 Seiten), schlimmstenfalls 133 Zugriffe auf die Basistabelle machen (wobei dies vollkommen unrealistisch ist, da die Country-Tabelle nur ein paar Seiten benötigt).

Man muss also insbesondere garnicht auf die ismember-Tabelle zugreifen.

Hinweise:

- Üblicherweise wird automatisch ein Index über alle Schlüssel sowie Fremdschlüssel erzeugt, da diese häufig für Joins und Selektionen benötigt werden.
- Bei der Auswertung kann ein DBS auch temporär einen Index erstellen.

Aufgabe 3 (Mengenoperationen) Seien $R(\bar{X}), S(\bar{X})$ Schemata und $\bar{Y} \subseteq \bar{X}$. Zeigen Sie oder widerlegen Sie:

a) $\pi[\bar{Y}](R \cup S) = \pi[\bar{Y}](R) \cup \pi[\bar{Y}](S)$

b) $\pi[\bar{Y}](R \cap S) = \pi[\bar{Y}](R) \cap \pi[\bar{Y}](S)$

c) (falls Anfrageoptimierung und interne Auswertung in der Vorlesung bereits besprochen wurden): wie werden die jeweiligen Ausdrücke am effizientesten ausgewertet?

---

a) Es gilt Gleichheit:

$$\begin{aligned}
\pi[\bar{Y}](R \cup S) &= \{\mu \in \text{Tup}(\bar{Y}) | \text{ es gibt } \nu \in R \cup S \text{ so dass } \mu = \pi[\bar{Y}](\nu)\} \\
&= \{\mu \in \text{Tup}(\bar{Y}) | \text{ es gibt } \nu_1 \in R \text{ so dass } \mu = \pi[\bar{Y}](\nu_1)\} \\
&\qquad \text{oder es gibt } \nu_2 \in S \text{ so dass } \mu = \pi[\bar{Y}](\nu_2)\} \\
&= \{\mu \in \text{Tup}(\bar{Y}) | \mu \in \pi[\bar{Y}](R) \text{ oder } \mu \in \pi[\bar{Y}](S)\} \\
&= \pi[\bar{Y}](R) \cup \pi[\bar{Y}](S)
\end{aligned}$$

b) Diese Gleichung gilt nicht: Betrachte

R	
A	B
a	1
a	2
b	4
c	5
d	7

S	
A	B
a	2
a	3
b	4
c	6
e	8

$R \cap S$	
A	B
a	2
b	4

$\pi[A](R)$
A
a
b
c
d

$\pi[A](S)$
A
a
b
c
e

$\pi[A](R) \cap \pi[A](S)$
A
a
b
c

c ist im Ergebnis enthalten, obwohl es in R und in S mit unterschiedlichen Werten für B auftritt.

Sei $\bar{Z} := \bar{X} \setminus \bar{Y}$.

$$\begin{aligned}
\pi[\bar{Y}](R \cap S) &= \{\mu \in \mathrm{Tup}(\bar{Y}) | \text{ es gibt } \nu \in \mathrm{Tup}(\bar{Z}) \text{ so dass } \mu\nu \in R \cap S\} \\
&= \{\mu \in \mathrm{Tup}(\bar{Y}) | \text{ es gibt } \nu \in \mathrm{Tup}(\bar{Z}) \text{ so dass } \mu\nu \in R \text{ und } \mu\nu \in S\}. \\
\pi[\bar{Y}](R) \cap \pi[\bar{Y}](S) &= \{\mu \in \mathrm{Tup}(\bar{Y}) | \mu \in \pi[\bar{Y}](R) \text{ und } \mu \in \pi[\bar{Y}](S)\} \\
&= \{\mu \in \mathrm{Tup}(\bar{Y}) | \text{ es gibt } \nu_1 \in \mathrm{Tup}(\bar{Z}) \text{ so dass } \mu\nu_1 \in R \text{ und} \\
&\qquad \text{es gibt } \nu_2 \in \mathrm{Tup}(\bar{Z}) \text{ so dass } \mu\nu_2 \in S\}
\end{aligned}$$

Hierbei kann $\nu_1 \neq \nu_2$ sein.

c) Bei Mengenoperationen (Duplikateliminierung, bei Schnittmenge und Mengendifferenz auch Suche) bietet es sich an, einen Index (Baum oder Hash) zur Laufzeit anzulegen.

a) über R iterieren, jedes Tupel projizieren und Ergebnis in Index einfügen. Dann über S iterieren, jedes Tupel projizieren und Ergebnis einfügen. Alles zusammen auslesen, indem die Blätter der Indexe sequentiell wie bei Merge-Sort parallel durchlaufen werden (wenn der Index als Hash realisiert ist: die Partitionen ungeordnet durchlaufen - gleiches muss wie beim Merge-Join in gegenüberliegenden Partitionen liegen). Falls bereits ein Index über $R[\bar{Y}]$ oder $S[\bar{Y}]$ in der Datenbank existiert, kann man statt Iteration über die Tupel direkt über die Indexeinträge iterieren.

$|R| \cdot (1 + \text{indextiefe}) + |S| \cdot (1 + \text{indextiefe}) + |\text{Ergebnis}|$

b) über R iterieren, jedes Tupel in Index einfügen; über S iterieren, jedes Tupel im R-Index suchen, gefundene Tupel projizieren und das Ergebnis in einen Ergebnis-Index einfügen. Zuletzt diesen ausgeben (Duplikate dabei ggf. entfernen).

$|R| \cdot (1 + \text{indextiefe}) + |S| \cdot (1 + \text{indextiefe}) + |R \cap S| \cdot (1 + \text{ergebnisindextiefe}) + |\text{Ergebnis}|$