

PD Dr. rer. nat. habil. Sven Groppe

Übungen zur Vorlesung

Mobile und Verteilte Datenbanken

WS 2013/2014

Übung 1 – Anfrageoptimierung in zentralisierten Datenbanksystemen

Aufgabe 1:

Folgende Relationen seien gegeben:

lieferant	Wer	Ort
	vobis	ulm
	escom	ulm
	quelle	nuernberg

liefert	Wer	Teil	Preis	Lieferzeit
	vobis	pc386	2000	4
	quelle	pc386	1900	9
	vobis	pc486	2900	4
	escom	pc486	3000	5
	vobis	pc586	5000	7
	escom	pc586	5900	9
	vobis	hp41	1400	6
	vobis	hddisk	13	0
	escom	hddisk	12	0
	quelle	cdrom	400	4

auftrag	Auftrag	Teil
	meier	pc486
	meier	hddisk
	reich	pc586
	reich	hp41
	reich	hddisk
	arm	pc386
	arm	hddisk

Die Anfrage „Wer hat ein Teil bestellt, das von einem Nürnberger Lieferanten geliefert wird?“ hat in SQL die Form:

select Auftrag **from** auftrag A, liefert L, lieferant LN

where LN.ort = “nuernberg“ **and**

LN.Wer=L.Wer **and**

L.Teil=A.Teil

- a) Zeichne den logischen Anfragebaum (ohne Iteratoren, mit Knoten zu algebraischen Operationen) und notiere an jedem Knoten die Kosten der Operation, d. h. die Anzahl der Tupel des Zwischenergebnisses anhand der gegebenen Tabellen.
- b) Berechne den optimalen Anfragebaum anhand der Optimierungsregeln zur logischen Anfrageoptimierung (siehe Skript) und zeichne ihn mit den entsprechenden Kosten an den Knoten.

Aufgabe 2:

Widerlegt die folgenden Gleichungen:

$$\Pi_m(R1 \cap R2) = \Pi_m(R1) \cap \Pi_m(R2)$$

$$\Pi_m(R1 - R2) = \Pi_m(R1) - \Pi_m(R2)$$

Aufgabe 3:

In dieser Aufgabe soll der *Nested-Loop-Join* zweier Relationen R und S näher betrachtet werden.

- a) Entwickle eine Formel, die die Anzahl der Seitenzugriffe bei einer einfachen Nested-Loop-Join-Implementierung (ohne Optimierung wie z.B. Wiederverwendung von gelesenen Seiten der inneren Schleife) repräsentiert. Benutze dabei die folgenden Variablen:

b_R : Anzahl Seiten der Relation R

b_S : Anzahl Seiten der Relation S

m : Anzahl Seiten, die im Hauptspeicher passen

k : Anzahl Seiten des Hauptspeichers, die für die innere Schleife benutzt werden, also

$$1 \leq k \leq m-1$$

- b) Wie viele Seitenzugriffe werden bei den folgenden Beispieldaten zur Berechnung eines Joins von A und B nötig? Welches k sollte dabei gewählt werden und welche Relation sollte für die äußere und welche für die innere Schleife benutzt werden (also $A \triangleright \triangleleft B$ oder $B \triangleright \triangleleft A$)?

- Die Relation A enthält 800.000 Tupel, die Relation B enthält 3.000.000 Tupel.
- Tupel von Relation A haben eine Größe von 100 Bytes und Tupel von Relation B eine Größe von 50 Bytes.
- Eine Seite umfasst 8 KBytes. Ein einzelnes Tupel muss immer auf eine Seite, d.h. pro Seite bleibt evtl. ein kleiner Rest unbenutzt.

- Der für den Join zur Verfügung stehende Hauptspeicher hat eine Größe von 2 MByte.

Berechne zuerst die Parameter b_A , b_B und benutze dann die Formel aus Teil a).

Aufgabe 4:

Betrachte nun den *Merge-Join*, der davon ausgeht, dass die beidem Relationen bereits nach den zu verbindenden Attributen sortiert sind. Wie viele Seitezugriffe sind dann im Beispiel von Aufgabe 3 nötig, falls wir von dem optimalen Fall ausgehen, dass es keine doppelten Werte gibt?

Aufgabe 5:

In dieser Aufgabe soll der Hash-Join näher untersucht werden. Der Hash-Join versucht, einen ähnlichen Vorteil wie den des Merge-Joins gegenüber dem Nested-Loop-Join zu erreichen ohne den gesamten zusätzlichen Aufwand des Sortierens leisten zu müssen.

Die Grundidee ist dabei, die Relationen mit Hilfe von Hash-Funktionen in kleine Partitionen zu zerteilen. Bei dem anschließenden Join muss dann nicht jedes Tupel mit jedem verglichen werden, sondern nur die Tupel in den jeweils entsprechenden Partitionen der beiden Relationen müssen verglichen werden. Indem zumindest einen der beiden zu vergleichenden Partitionen so klein gemacht wird, dass sie in $m-1$ von m zur Verfügung stehenden Hauptspeicherseiten passt, kann man den Vergleich zweier Partitionen so realisieren, dass dazu jede Partition insgesamt nur einmal vom Sekundärspeicher geladen werden muss.

Der Hash-Join läuft nach dem folgenden Schema ab:

Zuerst wird die kleinere der beiden Relationen partitioniert. Das Partitionieren geschieht in Runden, wobei in jeder Runde aus jeder bestehenden Partition $m-1$ entsprechend kleinere Partitionen entstehen. Das Partitionieren einer bestehenden Partition wird ausgeführt, indem eine der m Hauptspeicherseiten jeweils einer neuen Partition entsprechen. Die Tupel der bestehenden Partition werden durch eine Hash-Funktion einer der $m-1$ neuen Partitionen zugewiesen und in die entsprechende Hauptspeicherseite kopiert. Läuft eine Hauptspeicherseite leer bzw. voll, wird die nächste Seite nachgeladen bzw. wird die Seite auf den Sekundärspeicher geschrieben. Insgesamt werden die Runden solange wiederholt, bis jede der entstandenen Partitionen in $m-1$ Hauptspeicherseiten passt.

- Anschließend wird die zweite Relation partitioniert. Dies geschieht nach demselben Schema unter Benutzung derselben Hash-Funktion. Es werden genau so viele Runden gemacht wie bei der Partitionierung der ersten Relation.

- Schließlich kommt die Join-Phase. Dazu werden die beiden sich entsprechenden Partitionen beider Relationen verglichen. Da eine der beiden Partitionen in $m-1$ Hauptspeicherseiten passt, kann die verbliebene Hauptspeicherseite für die andere Partition benutzt werden.

Zur Analyse des Hash-Joins beantworten Sie die folgenden Fragen:

- a) Sei x_i die Größe (Anzahl Seiten) einer Partition nach i Partitionierungsrunden. Wie groß ist x_i bei m Hauptspeicherseiten und einer Relation R mit b_R Seiten? (Dabei soll hier von einer *perfekten* Hash-Funktion ausgegangen werden, d.h. jede der entstehenden Partitionen ist gleich groß. Diese Perfektion wird in der Praxis nie erreicht werden. Aber bei hinreichend großen Relationen und einer guten Streuung des Hash-Attributs ist die Abweichung von der perfekten Hash-Funktion sehr klein.)
- b) Wie viele Partitionsrunden sind dann nötig?
- c) Wie viele Sekundärspeicherseiten müssen pro Partitionierungsrunde gelesen oder geschrieben werden? Wie groß ist dann der Gesamtaufwand für das Partitionieren einer Relation?
- d) Wie groß ist der Gesamtaufwand für das Hash-Join zweier Relationen R und S mit b_R bzw. b_S Seiten und m Hauptspeicherseiten?