PD Dr. rer. nat. habil. Sven Groppe

Übungen zur Vorlesung

Mobile und Verteilte Datenbanken

WS 2011/2012

Übung 1 – Anfrageoptimierung in zentralisierten Datenbanksystemen

LÖSUNG

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	hp4l	1400	6
	vobis	hddisk	13	0
	escom	hddisk	12	0
	quelle	cdrom	400	4

auftrag	**Auftrag**	**Teil**
	meier	pc486
	meier	hddisk
	reich	pc586
	reich	hp4l
	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.

Lösung:

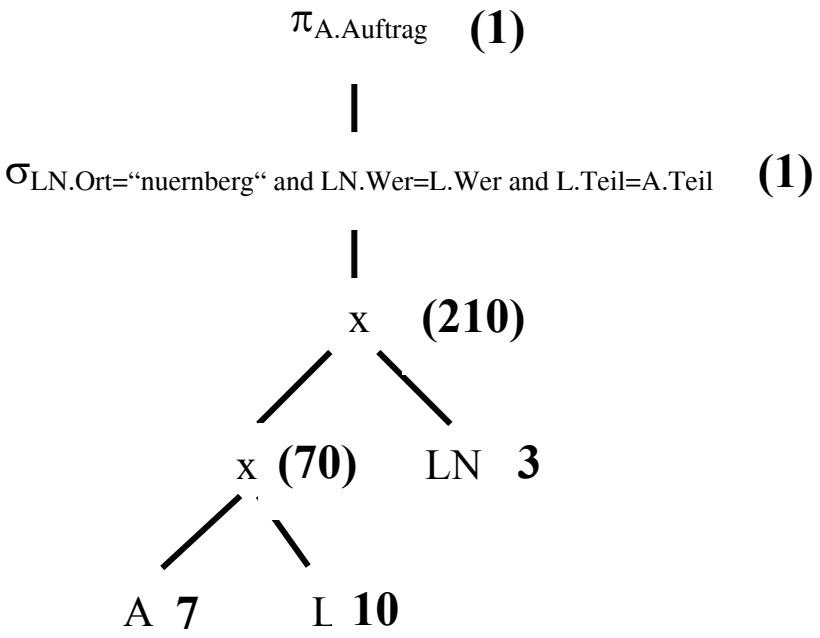
Die folgenden Operationen der relationalen Algebra werden benutzt:

x für kartesisches Produkt

σ_B für Selektion nach der Bedingung B

π_A für Projektion nach dem Attribut A

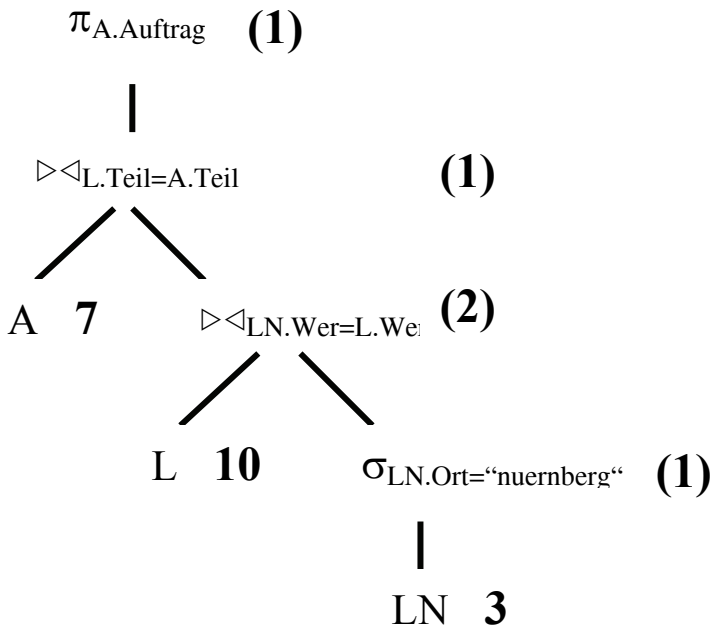
Der folgende Baum entspricht der nichtoptimierten Evaluierung des select-Befehls:



Die Nummer in Klammern bedeuten Größe der Zwischenergebnisse (d. h., Anzahl der Zeilen der erzeugten Tabellen). Insgesamt ergeben sich bei diesem Baum 282 Tupel (Zeilen) in den Zwischenergebnissen.

b) Berechne den optimalen Anfragebaum anhand der Optimierungsregeln zur logischen Anfrageoptimierung (siehe Skript) und zeichne ihn mit den entsprechenden Kosten an den Knoten.

Lösung:



In diesem Baum sind nur 5 Tupel als Zwischenergebnis notwendig. Entsprechend schneller kann dieser Anfragebaum bearbeitet werden.

Die Selektion *LN.Ort="nuernberg"* wurde hier ganz nach unten verschoben, die anderen Selektionen wurden mit den kartesischen Produkten in Join-Operationen (▷◁) zusammengefasst. (Auch wenn man die Join-Operation nicht verwendet – d. h., die Joins in dem optimalen Baum durch ein kartesisches Produkt mit der anschließenden Selektion ersetzt - wird die Größe der Zwischenergebnisse wesentlich kleiner als bei dem nicht-optimierten Baum.)

Aufgabe 2:

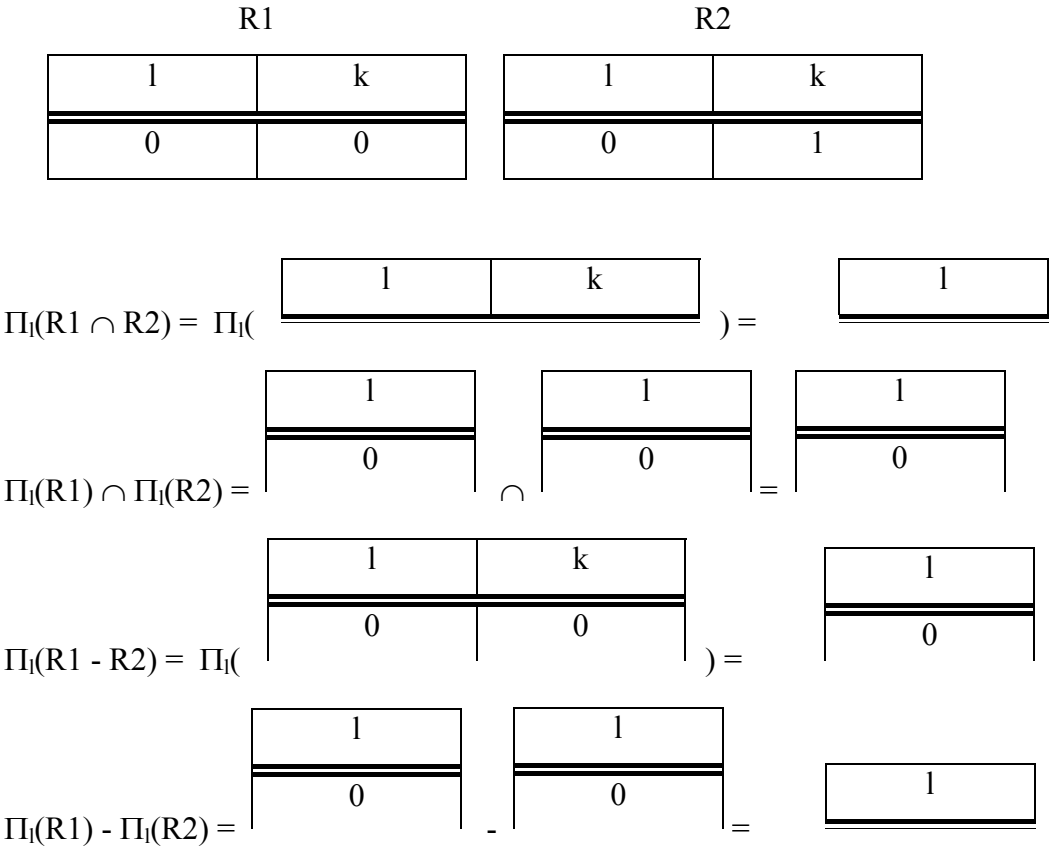
Widerlegt die folgenden Gleichungen:

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

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

Lösung:

Beweis durch Gegenbeispiel. Betrachte dazu die folgenden Relationen R1 und R2 mit jeweils einem Tupel (die Relationen sind strukturell äquivalent, d. h., sie enthalten die gleichen Attribute):



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 Variabeln:

   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 $1 \leq k \leq m-1$

Lösung:

Die gesamte äußere Relation R wird einmal gelesen: b_R

In der inneren Schleife wird in einer Runde (d.h. für den jetzigen „Hauptspeicherkontext“ der Relation R) die gesamte Relation S einmal gelesen: b_S

Es werden so viele Runden benötigt, wie viel mal der „Hauptspeicherkontext“ der Relation R gewechselt werden muss: $\lceil b_R/(m-k) \rceil$

Also, die gesamte Anzahl der Platten(lese)zugriffe zu Seiten der Relation R wird b_R und die gesamte Anzahl der Plattenzugriffe zu Seiten der Relation S wird $\lceil b_R/(m-k) \rceil b_S$

Insgesamt werden $\mathbf{b_R + \lceil b_R/(m-k) \rceil b_S}$ benötigt.

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▷◁B oder B▷◁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).

Lösung:

Die Anzahl der Platten(lese)zugriffe wird minimiert, wenn die äußere Relation *R* die kleinere ist und wenn k=1 gewählt wird.

In diesem Beispiel passen in eine Seite $\lfloor 8192 / 100 \rfloor = 81$ Tupel der Relation *A*, d.h.

$b_A = \lceil 800.000 / 81 \rceil$

Für die Relation *B* passen in eine Seite $\lfloor 8192 / 50 \rfloor = 163$ Tupel, d. h.

$b_B = \lceil 3.000.000 / 163 \rceil = 18.405$

Für *m* gilt

$m = \lfloor 2048 / 8 \rfloor = 256$

Die Anzahl der Platten(lese)zugriffe wird (Relation *A* wird die äußere)

$9877 + \lceil 9877 / 255 \rceil * 18.405 =$ **727.672**

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?

Lösung:

Im optimalen Fall, **wenn es keine doppelten Werte gibt**, sind maximal $\mathbf{b_R + b_S}$ Platten(lese)zugriffe notwendig, weil beide Relationen einmal gelesen werden (bei Equi-Join). Die Anzahl erhöht sich normalerweise nur gering, wenn es doppelte Werte gibt. Kommt es allerdings zu dem Extremfall, dass alle Werte gleich sind, ist der Aufwand gleich dem Aufwand des Nested-Loop-Joins. Der Grund des zusätzlichen Aufwands bei Duplikaten ist, dass der Zeiger (Cursor) einer Relation mehrmals zurückgesetzt werden muss, so dass er auf den ersten duplizierten Wert zeigt.

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 bitte die folgenden Fragen:

a) Sei x_i die Größe (Anzahl Seiten) einer Partition nach i Partitionierungs-Runden. 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.)

Lösung:

Vor der ersten Runde ist die ganze Relation in einer Partition gespeichert, also $x_0 = b_R$. In jeder Runde verkleinert sich die Größe jeder Partition um einen Faktor $m - 1$. Also

$$x_i = b_R / (m - 1)^i$$

(Dieser Wert müsste eigentlich immer wieder aufgerundet werden, wodurch sich die Anzahl der Seiten von Runde zu Runde etwas erhöht. Diesen kleinen Fehler werden wir aber ignorieren.)

b) Wie viele Partitionsrunden sind dann nötig?

Lösung:

Es muss solange partitioniert werden, bis $x_i \leq m - 1$. Daraus folgt

$$i \geq \log_{m-1}(b_R) - 1$$

Es sind also insgesamt $\lceil \log_{m-1}(b_R) - 1 \rceil$ Runden nötig.

c) Wie viele Sekundärspeicherseiten müssen pro Partitionierungsrunde gelesen oder geschrieben werden? Wie groß ist dann der Gesamtaufwand für das Partitionieren einer Relation?

Lösung:

Pro Runde muss jede Partition einmal gelesen und einmal geschrieben werden. Da alle Partitionen zusammen die gesamte Relation darstellen, wird pro Runde jede Seite der Relation einmal gelesen und einmal geschrieben. Das bedeutet $2\ b_R$ Plattenzugriffe pro Runde. Insgesamt sind für das Partitionieren der Relation R

$$2\ b_R \lceil \log_{m-1}(b_R) - 1 \rceil$$

Plattenzugriffe notwendig.

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?

Lösung:

Insgesamt müssen beide Relationen partitioniert werden (die Anzahl von Runden ist für beide Partitionen gleich, siehe Aufgaben a) und b)) und anschließend muss der Join ausgeführt werden. Das bedeutet

$$2\ b_R \lceil \log_{m-1}(b_R) - 1 \rceil + 2\ b_S \lceil \log_{m-1}(b_R) - 1 \rceil + b_R + b_S = (b_R + b_S)\ (2 \lceil \log_{m-1}(b_R) \rceil - 1)$$

Plattenzugriffe.