ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ

Συστήματα Διαχείρισης Βάσεων Δεδομένων

Άσκηση 2

Δημήτρης Πλεξουσάκης

Τμήμα Επιστήμης Υπολογιστών

HY460 – Συστήματα Διαχείρισης Βάσεων Δεδομένων

Δημήτρης Πλεξουσάκης
2η Σειρά Ασκήσεων

Άσκηση 1 (25 μονάδες) – Εξωτερική ταξινόμηση

Έστω αρχείο 10.000 σελίδων. Επιπλέον έστω 3 διαθέσιμες σελίδες ενδιάμεσης μνήμης.
α) Απαντήστε στις παρακάτω ερωτήσεις θεωρώντας ότι χρησιμοποιούμε τον αλγόριθμο εξωτερικής ταξινόμησης multi-way external sort.

i. *Πόσα ταξινομημένα runs θα παραχθούν στο πρώτο πέρασμα;*
ii. *Πόσα περάσματα θα χρειασθούν για την πλήρη ταξινόμηση του αρχείου;*
iii. *Ποιο είναι το συνολικό κόστος της ταξινόμησης του αρχείου;*
iv. *Πόσες σελίδες ενδιάμεσης μνήμης χρειάζονται για την πλήρη ταξινόμηση του αρχείου σε 2 μόνο περάσματα;*

Λύση

i. Στο πρώτο πέρασμα (run 0), διαβάζονται τα δεδομένα εισόδου στις 3 σελίδες τη φορά και ταξινομούνται εσωτερικά ώστε να παραχθούν 10000/3=3334 ταξινομημένα περάσματα (το τελευταίο πέρασμα μπορεί να έχει λιγότερες σελίδες)

ii. Ξέρουμε από την θεωρία ότι ο αριθμός των αρχικών περασμάτων καθορίζει τον αριθμό των συνολικών περασμάτων για την ταξινόμηση του αρχείου. Χρειαζόμαστε logB-1[N/B]+1=13 περάσματα

iii. Αφού κάθε σελίδα διαβάζεται και γράφεται μια φορά ανα πέρασμα, το συνολικό κόστος ταξινόμησης του αρχείου θα είναι 2*N*(# of passes) = 2*10.000*13 = 260.000 IOs

iv. Χρειαζόμαστε B = 101, buffers μια και θα πρέπει να ισχύει $\log_{x-1}(10000/x)+2=2$

β) Απαντήστε στις ερωτήσεις του ερωτήματος α) για την περίπτωση που έχουμε αρχείο 20.000 σελίδων να ταξινομήσουμε και διαθέτουμε 5 διαθέσιμες σελίδες ενδιάμεσης μνήμης.

Λύση

i. 20.000/5=4.000 runs
ii. logB-1[N/B]+1=7 περάσματα
iii. 280.000 IOs
iv. B = 142, buffer pages

γ) Απαντήστε στις ερωτήσεις του ερωτήματος α) για την περίπτωση που έχουμε αρχείο 2.000.000 σελίδων να ταξινομήσουμε και διαθέτουμε 17 σελίδες ενδιάμεσης μνήμης.

Λύση

i. 2.000.000/17=117.648 runs
ii. logB-1[N/B]+1=6 περάσματα
iii. 24.000.000 IOs
iv. B = 1415 buffer pages

δ) Απαντήστε στις ερωτήσεις του ερωτήματος α) θεωρώντας ότι έχουμε αρχείο 20.000 σελίδων να ταξινομήσουμε και χρησιμοποιούμε εξωτερική ταξινόμηση με 2 σελίδες ενδιάμεσης μνήμης (two-way external sort)

Λύση

i. 20.000 runs
ii. logB-1[N/B]+1=16 περάσματα
iii. 640.000 IOs
iv. Δεν μπορούμε να το ταξινομήσουμε σε 2 περάσματα.

Άσκηση 2 (25 μονάδες) – Αλγόριθμοι Σύζευξης

Έστω οι σχέσεις Orders και Customers αποθηκευμένες ως αταξινόμητα αρχεία σωρού με τα παρακάτω χαρακτηριστικά

Orders	***Customers***
10.000 πλειάδες, 10 πλειάδες ανά σελίδα	*2.000 πλειάδες, 10 πλειάδες ανά σελίδα*
Πρωτεύον κλειδί είναι το πεδίο orderID *Το πεδίο cID της σχέσης είναι ξένο κλειδί από την σχέση Customers*	*Πρωτεύον κλειδί είναι το πεδίο customerID*

Επιπλέον έστω 52 σελίδες ενδιάμεσης μνήμης και επιπλέον δεν έχουμε διαθέσιμα ευρετήρια

α) Ποιο είναι το κόστος της σύζευξης $Orders_{Orders.cID=Customers.customerID}$ *Customers με χρήση block nested loops; Υπολογίστε το ελάχιστο πλήθος σελίδων ενδιάμεσης μνήμης που απαιτούνται ώστε το κόστος αυτό να παραμένει σταθερό.*

Λύση

Για την σχέση Orders, έχουμε ότι αποθηκεύονται 10 πλειάδες ανά σελίδα. Εφόσον έχουμε 10.000 πλειάδες χρειαζόμαστε Pages_Order = 1.000 σελίδες.

Για την σχέση Customers, έχουμε ότι αποθηκεύονται 10 πλειάδες ανά σελίδα. Εφόσον έχουμε 2.000 πλειάδες χρειαζόμαστε Pages_Customers = 200 σελίδες.

Η Customers είναι η μικρότερη σχέση και θα παίζει το ρόλο της εξωτερικής σχέσης ενώ ι η Orders θα είναι η εσωτερική σχέση. Θα τεμαχίσουμε την Customers σε blocks, και θα σαρώσουμε την Orders για ταιριαστές πλειάδες. Συνεπώς, η εξωτερική σχέση θα διαβαστεί 1 φορά και η εσωτερική θα διαβαστεί 1 φορά για κάθε block.

Ο αριθμός των block που θα έχουμε θα είναι : #pages in outer relation/ #buffer pages – 2 = 200 / 52-2= 4

Συνεπώς, το κόστος θα είναι Total Cost = Pages_Customers + blocks * Pages_Orders = 200 + 4*1.000 = **4.200 I/O.**

Για να βρούμε το ελάχιστο πλήθος σελίδων ενδιάμεσης μνήμης που απαιτούνται ώστε το κόστος αυτό να παραμείνει σταθερό πρέπει να σκεφτούμε ότι ο αριθμός των block θα είναι μεγαλύτερος αν αρχίσουμε να αφαιρούμε σελίδες ενδιάμεσης μνήμης. Για παράδειγμα, αν το πλήθος των ενδιάμεσων σελίδων είναι 51 και όχι 52, τότε ο αριθμός των block θα είναι : Αριθμός σελίδων στην εξωτερική σχέση / αριθμός ενδιάμεσων σελίδων – 2 = 200/49 = 5. Συνεπώς, με την αφαίρεση μιας και μόνο σελίδας θα αυξήσουμε το κόστος. Οπότε, το ελάχιστο πλήθος σελίδων ενδιάμεσης μνήμης για αυτό το κόστος είναι **52**.

β) Ποιο είναι το κόστος της παραπάνω σύζευξης με ταξινόμηση και συγχώνευση (sort merge join); Υπολογίστε το ελάχιστο πλήθος σελίδων ενδιάμεσης μνήμης (buffer) που απαιτούνται ώστε το κόστος αυτό να παραμένει σταθερό.

Λύση

Η ιδέα της σύζευξης με ταξινόμηση και συγχώνευση, είναι η ταξινόμηση και των δύο σχέσεων ως προς το γνώρισμα σύζευξης και η αναζήτηση των κατάλληλων πλειάδων μέσω της συγχώνευσης τους.

Στο πρώτο πέρασμα (run 0), τα παραγόμενα περάσματα για κάθε σχέση θα είναι N/2*B= 1000/2*52=10 και N/2*B=200/2*52=2. Αφού 10 + 2 = 12 < 52 μπορούμε να χρησιμοποιήσουμε την βελτιστοποίηση του αλγορίθμου ταξινόμησης & συγχώνευσης και δεσμεύουμε 1 buffer page για κάθε πέρασμα για της κάθε σχέσης. Το κόστος όπως έχουμε δει από τις διαφάνειες θα είναι Total Cost = 3* (Pages_Order + Pages_Customer) = 3* (1000+200) = 3 *1.200 =**3.600 I/Os** (Διάλεξη query processing, σελ 58)

Για τον υπολογισμό του ελάχιστου πλήθους σελίδων ενδιάμεσης μνήμης για το κόστος, υπολογίζουμε βασικά το ελάχιστο πλήθος σελίδων ενδιάμεσης μνήμης που απαιτούνται για την ταξινόμηση σε 2 περάσματα της orders. Έτσι θέλουμε $\log_{B-1}1000/B +1 = 2$ οπότε πρέπει B>=33. Άρα απαιτούνται κατ' ελάχιστον **33** σελίδες ενδιάμεσης μνήμης.

γ) Ποιο είναι το κόστος της παραπάνω σύζευξης με κατακερματισμό (hash join); Υπολογίστε το ελάχιστο πλήθος σελίδων ενδιάμεσης μνήμης (buffer) που απαιτούνται ώστε το κόστος αυτό να παραμένει σταθερό.

Λύση

Στη φάση διαμερισμού, πρέπει να σαρώσουμε και να γράψουμε στο δίσκο μια φορά και την Customers και την Orders, πράγμα που σημαίνει ότι το κόστος αυτής της φάσης θα είναι 2* (Pages_Customers + Pages_Orders) = 2* (1.000 + 200) = 2* 1.200 = 2.400. Στη δεύτερη φάση, και

υποθέτοντας ότι δεν υπάρχουν υπερχειλίσεις των διαμερισμάτων, σαρώνουμε κάθε διαμέρισμα μια φορά με συνολικό κόστος Pages_Customers + Pages_Orders = 1.200 I/O.

Άρα το συνολικό κόστος θα είναι 3* (Pages_Customers + Pages_Orders) = 3* (1.000 + 200) = 3* 1.200 = **3.600 I/O.**

Το ελάχιστο πλήθος σελίδων ενδιάμεσης μνήμης που χρειάζονται είναι περίπου ίσο με το πλήθος σελίδων ενδιάμεσης μνήμης που απαιτούνται για την κατασκευή ενός πίνακα κατακερματισμού μέσα στη μνήμη ενός partition οποιασδήποτε από τις δύο σχέσεις. Καθώς η Customers είναι η μικρότερη, μικρότερη θα είναι και κάθε partition της, οπότε η Customers αποτελεί καλύτερη επιλογή. Χρειαζόμαστε περίπου σελίδες ενδιάμεσης μνήμης όπου f είναι ο προσεγγιστικός τελεστής fudge.

δ) Θεωρείστε την ύπαρξη ενός ευρετηρίου κατακερματισμού στο γνώρισμα customerID της Customers, υλοποιημένου με την εναλλακτική 2. Ποιο είναι το κόστος της εν λόγω σύζευξης με χρήση φωλιασμένων βρόγχων με ευρετήριο (index nested loop join);

Λύση

Αφού το CustomersID είναι κλειδί της Customers, έχουμε το πολύ μια ταιριαστή πλειάδα. Το cID της Orders είναι ξένο κλειδί που αναφέρεται στην Customers, και έτσι έχουμε ακριβώς μια ταιριαστή πλειάδα της Customers για κάθε πλειάδα της Orders.

Η Orders έχει 1.000 σελίδες, άρα το κόστος σάρωσης της είναι 1.000. Υπάρχουν 10.000 πλειάδες στην Orders. Για κάθε μια από αυτές τις πλειάδες, το κόστος ανάκτησης της σελίδας του ευρετηρίου που περιέχει τον προσδιοριστή εγγραφής της ταιριαστής πλειάδας της Customers είναι 1-2 I/O -- 1.2είναι ένα τυπικό κόστος για ευρετήριο που βασίζονται στον κατακερματισμό. Επιπλέον, πρέπει να ανακτήσουμε τη σελίδα της Customers που περιέχει την κατάλληλη πλειάδα. Έτσι, χρειαζόμαστε: 1.000 + 10.000 *(1 + 1.2) = **23.000 I/O.**

ε) Θεωρείστε την ύπαρξη ενός συσταδοποιημένου ευρετηρίου τύπου B+-tree στο γνώρισμα cID της Orders, υλοποιημένου με την εναλλακτική 1, αγνοώντας την ύπαρξη του ευρετηρίου του ερωτήματος (δ). Ποιο είναι το κόστος της εν λόγω σύζευξης με χρήση φωλιασμένων βρόγχων με ευρετήριο (index nested loop join); Υποθέστε ότι το B+-tree που θα χρησιμοποιήσετε έχει fanout 100.

Λύση

Τώρα η εξωτερική σχέση είναι η Customers. Υποθέτουμε ότι το δέντρο έχει 3 επίπεδα, το πρώτο περιλαμβάνει μόνο τη ρίζα, το δεύτερο έχει 10 κόμβους και το τρίτο έχει 1000 φύλλα καθένα από τα οποία περιέχει 10 πλειάδες. Τα 2 πρώτα επίπεδα (11 σελίδες) χωράνε στην μνήμη, οπότε για την ανάκτηση κάθε πλειάδας της Orders χρειαζόμαστε μόνο 1 πρόσβαση για να μεταβούμε στο φύλλο που την περιέχει.

Αφού το γνώρισμα cID της Orders δεν είναι κλειδί, είναι πιθανό να υπάρχουν πολλαπλά ταιριάσματα για κάθε πλειάδα της Customers. Εφόσον το ευρετήριο της Orders είναι συσταδοποιημένο υποθέτουμε ότι για κάθε πλειάδα της Customers η ανάκτηση όλων των αντίστοιχων πλειάδων της Orders κοστίζει το πολύ 2 I/Os.

Οπότε κόστος = 200 +2.000*2+11=**4211 I/Os**

Άσκηση 3 (25 μονάδες) – Βελτιστοποίηση Επερωτήσεων και Στατιστικά

Υποθέστε ότι έχετε τις δύο σχέσεις Orders και Customers της άσκησης 2, μαζί με τα ευρετήρια των ερωτημάτων (δ) και (ε). Έστω, επίσης, μια σχέση Sales με τις ίδιες ιδιότητες όπως η Orders, αλλά χωρίς κανένα ευρετήριο. Τέλος, θεωρήστε ότι το ευρετήριο B+-tree στο γνώρισμα cID της Orders περιέχει 1.000 διακριτές τιμές κλειδιού, ενώ το γνώρισμα cID της Sales έχει εύρος 3.000 τιμών.

Εκτελέστε τον αλγόριθμο δυναμικού προγραμματισμού για την εύρεση ενός βέλτιστου πλάνου με βαθιά αριστερά δένδρα (left-deep trees) συζεύξεων για την αποτίμηση της ακόλουθης έκφρασης:

$$(Orders \underset{Orders.cID=Customers.customerID}{} Customers) \underset{Customers.customerID=Sales.cID}{} Sales$$

Λύση

Πέρασμα 1:

Εξετάζουμε τις μεθόδους πρόσβασης για τους πίνακες ξεχωριστά

Customers: 2 access methods:

- hash index,
- file scan

Επιλέγουμε να χρησιμοποιήσουμε hash index, αφού έχει μικρότερο κόστος σε σχέση με το File scan.

Orders: 2 access methods:

- B+ tree index,
- file scan

Επιλέγουμε να χρησιμοποιήσουμε B+ tree index, αφού έχει μικρότερο κόστος σε σχέση με το File scan.

Sales: 1 access method:

- file scan

Πέρασμα 2:

Εξετάζουμε καθένα από τα πλάνα του περάσματος 1, θεωρώντας τη σχέση που περιέχει ως εξωτερική σε μια σύζευξη με άλλη σχέση ως εσωτερική. Για κάθε μέθοδο σύζευξης, χρειάζεται να εξετάσουμε και αν η έξοδος είναι ταξινομημένη καθώς αυτό επηρεάζει το κόστος της επόμενης φάσης

Orders και Customers				
Μέθοδος join	καλύτερη εξωτερική	Κόστος Ε/Ε	Ταξινομημένη έξοδος	Μέγεθος εξόδου
block nested loop	Customers	4.200	Όχι	2.000^2
sort merge	**-**	**1.600[1]**	**Ναι**	
hash	-	3.600	Όχι	
indexed nested loop	Orders	23.000	Όχι	
	Customers	4.211	Όχι	

[1] Στον παραπάνω πίνακα χρησιμοποιήσαμε τα αποτελέσματα της Άσκησης 2. Για την sort-merge χρησιμοποιήσαμε τον refined sort-merge join. Παρατηρούμε επίσης ότι η Orders είναι ήδη ταξινομημένη εξαιτίας του συσταδοποιημένου ευρετηρίου B+-tree τύπου Alternative 1. Οπότε, εξοικονομούμε 1 ανάγνωση και 1 εγγραφή της R, άρα το κόστος γίνεται M+3 N = 1.000+3200 = 1.600

[2] Η έξοδος περιλαμβάνει 10.000 πλειάδες, καθώς για κάθε πλειάδα της Orders, υπάρχει ακριβώς μία πλειάδα της Customers που ικανοποιεί τη συνθήκη *Orders.cID=Customers.customerID*. Όμως μία πλειάδα της O*rders Customers* έχει σχεδόν διπλάσιο μέγεθος από μια πλειάδα της Orders, και γι' αυτό υποθέτουμε ότι το μέγεθος εξόδου θα είναι 2.000 σελίδες.
Συνεπώς, καλύτερη επιλογή αποτελεί το sort merge, καθώς όχι μόνο έχει το μικρότερο δυνατό κόστος Ε/Ε, αλλά είναι και το μοναδικό που μας δίνει στην έξοδο ταξινομημένο αποτέλεσμα.

Sales και Customers				
Μέθοδος join	καλύτερη εξωτερική	Κόστος Ε/Ε	Ταξινομημένη έξοδος	Μέγεθος εξόδου
block nested loop	Customers	4.200	Όχι	
sort merge	-	**3.600**	**Ναι**	
hash	-	**3.600**	Όχι	2.000
indexed nested loop	Sales [3]	23.000	Όχι	

[3] Εδώ μόνο η Sales μπορεί να χρησιμοποιηθεί ως εξωτερική καθώς μόνο η Customers είναι ευρετηριασμένη.
Ο εκλεπτυσμένος αλγόριθμος σύζευξης με ταξινόμηση και συγχώνευση (refined sort-merge join) εξακολουθεί να αποτελεί την βέλτιστη επιλογή, καθώς έχει το πλεονέκτημα έναντι της ιδίου κόστους Ε/Ε σύζευξης με κατακερματισμό (hash join) να παράγει ταξινομημένη έξοδο.

Orders και Sales				
Μέθοδος join	καλύτερη εξωτερική	Κόστος Ε/Ε	Ταξινομημένη έξοδος	Μέγεθος εξόδου
block nested loop	Οποιαδήποτε	1.000+[1.000/(52-2)]· 1.000 = 21.000	Όχι	6.600[4]
sort merge	-	**4.000[1]**	**Ναι**	
hash	-	3(1.000+1.000) = 6.000	Όχι	
indexed nested loop	Sales[3]	1.000+10.000·(1+1)+11 = 21.011	Όχι	

[4] Υποθέτουμε ομοιόμορφη κατανομή των τιμών του Sales.cID, συνεπώς, για κάθε τιμή του Sales. cID θα υπάρχουν κατά μέσο όρο (10.000/3.000) = 3,3 πλειάδες.

Επιπλέον, υποθέτουμε ότι κάθε τιμή του Orders.cID εμφανίζεται στο σύνολο τιμών του Sales.cID, συνεπώς η έξοδος θα έχει μέγεθος 3,310.000 = 33.000 tuples. Θεωρώντας, τέλος, το μέγεθος των πλειάδων της εξόδου περίπου διπλάσιο, καταλήγουμε σε μέγεθος εξόδου 6.600 σελίδες.

Για μια ακόμη φορά, ο εκλεπτυσμένος αλγόριθμος σύζευξης με ταξινόμηση και συγχώνευση (refined sort-merge join) παραμένει βέλτιστη επιλογή.

Πέρασμα 3: Εδώ πρέπει να εξετάσουμε κατά πόσον οι πλειάδες που παράγονται από το εξωτερικό πλάνο μπορούν να διοχετευθούν με pipeline στη σύζευξη (join). Αν όχι, τότε έχουμε επιπλέον κόστος για την υλοποίηση (materialization), εφόσον πρέπει να γράφουμε και εν συνεχεία να διαβάζουμε ενδιάμεσα αποτελέσματα.

Orders Customers με Sales, όπου η Orders Customers έχει υλοποιηθεί με sort merge		
Μέθοδος join	Κόστος Ε/Ε	pipelined
sort merge	**3·1.000 = 3.000[5]**	**Ναι[6]**
block nested loop	1.000[2.000/50] = 40.000 [7, 8]	Περιττό [9]
hash	2·2.000+3·1.000 = 7.000[7]	Περιττό
indexed nested loop	-[10]	-

[5] Χρησιμοποιούμε και πάλι τον εκλεπτυσμένο αλγόριθμο σύζευξης με ταξινόμηση και συγχώνευση (refined sort-merge join). Αφού η σχέση Orders Customers είναι ταξινομημένη και διοχετεύεται με pipeline στη σύζευξη (join), μπορούμε ακόμη και να εξοικονομήσουμε αναγνώσεις από το δίσκο γι' αυτήν.

[6] Η φάση της συγχώνευσης για τον υπολογισμό της Orders Customers χρειάζεται μόνο 2 σελίδες ενδιάμεσης μνήμης (buffer) για την είσοδο και 1 για την έξοδο. Δεσμεύοντας 1 σελίδα ενδιάμεσης μνήμης (buffer) εξόδου για την ταξινόμηση της Sales, έχουμε στη διάθεση μας 48 σελίδες ενδιάμεσης μνήμης (buffer) για είσοδο στην ταξινόμηση της Sales, που είναι υπεραρκετές. Συνεπώς, η ταξινόμηση της Sales με τη συγχώνευση για το join της Sales με την Orders Customers, μαζί με την ταξινόμηση και συγχώνευση (sort merge) για τον υπολογισμό της Orders Customers, μπορεί να γίνει με pipelining.

[7] Εξοικονομούμε 1 ανάγνωση για την Orders Customers θεωρώντας ότι εφαρμόζεται pipeline.

[8] Εφόσον θεωρούμε μόνο πλάνα με βαθιά αριστερά δέντρα συζεύξεων, η Orders Customers πρέπει υποχρεωτικά να είναι η εξωτερική σχέση.

[9] Είναι περιττό να την εξετάσουμε καθώς είναι ήδη χειρότερη από το sort-merge, ακόμη και με την υπόθεση ότι θα μπορούσε να γίνει υπό σωλήνωση (pipeline).

[10] Βάσει του (8) και της απουσίας ευρετηρίου για την Sales.

Συνεπώς, για την Orders Customers με την Sales, ο καλύτερος τρόπος είναι να προηγηθεί η σύζευξη μεταξύ Orders και Customers με τη μέθοδο sort merge, και να ακολουθήσει η σύζευξη μεταξύ της Orders Customers και της Sales με την ίδια μέθοδο, συνολικού κόστους 1.600+3.000 = **4.600 Ε/Ε.**

Customers Sales **με Orders, όπου η** Customers Sales **έχει υλοποιηθεί με sort merge**		
Μέθοδος join	Κόστος Ε/Ε	pipelined

sort merge	**1.000[11]**	**Ναι**
block nested loop	1.000[2.000/50] = 40.000[7], [8]	Περιττό[9]
hash	2·2.000+3·1.000 = 7.000[7]	Περιττό
indexed nested loop	10.000·(1+1)+11 = 20.011[7]	Περιττό

[11] Χρησιμοποιούμε και πάλι τον εκλεπτυσμένο αλγόριθμο σύζευξης με ταξινόμηση και συγχώνευση (refined sort-merge join). Αφού η σχέση Customers Sales είναι ταξινομημένη και διοχετεύεται με σωλήνωση (pipeline) στη σύζευξη (join), μπορούμε ακόμη και να εξοικονομήσουμε το read γι' αυτήν. Η Orders είναι επίσης ταξινομημένη.
Συνεπώς, για την Customers Sales με την Orders, ο καλύτερος τρόπος είναι να προηγηθεί η σύζευξη μεταξύ Customers και Sales με τη μέθοδο sort merge, και να ακολουθήσει η σύζευξη μεταξύ της Customers Sales και της Orders με την ίδια μέθοδο, συνολικού κόστους 3.600+1.000 = **4.600 Ε/Ε.**

Orders Sales **με Customers, όπου η** Orders Sales **έχει υλοποιηθεί με sort merge**		
Μέθοδος join	Κόστος Ε/Ε	pipelined
sort merge	**600[12]**	**Ναι[6]**
block nested loop	200[6.600/50] = 26.400[7], [8]	Περιττό[9]
hash	2·6.600+3·200 = 13.800[7]	Περιττό
indexed nested loop	33.000(1,2+1) = 72.600[7]	Περιττό

[12] Χρησιμοποιούμε και πάλι τον εκλεπτυσμένο αλγόριθμος σύζευξης με ταξινόμηση και συγχώνευση (refined sort-merge join). Αφού η σχέση Orders Sales είναι ταξινομημένη και διοχετεύεται με pipeline στη σύζευξη (join), μπορούμε ακόμη και να εξοικονομήσουμε το read γι' αυτήν.
Συνεπώς, για την Orders Sales με την Customers, ο καλύτερος τρόπος είναι να προηγηθεί η σύζευξη μεταξύ Orders και Sales με τη μέθοδο sort merge, και να ακολουθήσει η σύζευξη μεταξύ της Orders Sales και της Customers με την ίδια μέθοδο, συνολικού κόστους 4.000+600 = 4.600 Ε/Ε.
Συνεπώς οι βέλτιστες επιλογές είναι οι ακόλουθες τρεις μέθοδοι:
Πρώτα join Orders Customers με sort merge, και μετά join (Orders Customers) Sales με sort merge.
Πρώτα join Customers Sales με sort merge, και μετά join (Customers Sales) Orders με sort merge.
Πρώτα join Orders Sales με sort merge, και μετά join (Orders Sales) Customers με sort merge.

Άσκηση 4 (25 μονάδες) – Βελτιστοποίηση Επερωτήσεων και Στατιστικά

Έστω η παρακάτω σχέση Phones

Manufacturer	***Model***	***GBs***
Apple	*iPhone 4*	*8*

Apple	*iPhone 6*	*8*
Apple	*G3*	*8*
LG	*Nexus 4*	*16*
LG	*Nexus 5*	*16*
LG	*G3*	*16*
SAMSUNG	*Galaxy S*	*8*
SAMSUNG	*Galaxy Note*	*8*
SAMSUNG	*G3*	*8*

α) Τί τιμή έχουν οι παρακάτω μεταβλητές;

i. *T(Phones)*
ii. *V(Phones, Manufacturer)*
iii. *V(Phones, GBs)*

Λύση

i. T(Phones)= 9
ii. V(Phones, Manufacturer)= 3
iii. V(Phones, GBs)=2

β) Υποθέστε ότι οι τιμές για τις επερωτήσεις επιλέγονται από τις τιμές που υπάρχουν στην βάση μας. Τι θα υπολογίσει ο βελτιστοποιητής των επερωτήσεων (optimizer) για:

i. *Τον αριθμό των πλειάδων στο* $\sigma_{Manufacturer=Apple}(R)$
ii. *Τον αριθμό των πλειάδων στο* $\sigma_{Model=G3}(R)$
iii. *Τον αριθμό των πλειάδων στο* $\sigma_{Manufacturer=Apple \wedge Model=G3}(R)$
iv. *Τον αριθμό των πλειάδων στο* $\sigma_{Manufacturer=Apple \wedge GBs=8}(R)$
v. *Τον αριθμό των πλειάδων στο* $\Pi_{Manufacturer, GBs}(R)$

Λύση

i. T(Phones)/V(Phones, Manufacturer) = 9/3 = 3
ii. T(R) / V (R, Model) = 9/7
iii. V(Phones, Manufacturer)) = 9/(7*3) = 3/7 = 0.42
iv. T(Phones)/(V(Phones, GBs) * V(Phones, Manufacturer)) = 9/(2*3) = 3/2 = 1.5
v. Η σχέση Phones έχει 9 διακριτές πλειάδες. Το projection απαιτεί να απορρίψουμε μερικά πεδία και η διασφάλιση ότι δεν υπάρχουν διπλότυπα. Αν δεν χρειάζεται να εξαλειφθούν τα διπλότυπα (δεν υπάρχει το distinct στην επερώτηση στην sql) τότε η προβολή αποτελείται από την απλή ανάκτηση ενός υποσυνόλου των πεδίων του πίνακα. Άρα ο αναμενόμενος αριθμός πλειάδων από τον βελτιστοποιητή θα είναι T(Phones) = 9.

γ) Εξηγείστε σύντομα γιατί η εκτίμηση για την επερώτηση Manufacturer=Apple ^ Model=G3 μπορεί να έχει μεγαλύτερη ακρίβεια από την επερώτηση Manufacturer = Apple ^ GBs=8.

Λύση

Το γνώρισμα Model έχει 7 διακριτές τιμές, το οποίο είναι πολύ κοντά στο σύνολο των πλειάδων μας, πράγμα που σημαίνει ότι το πολύ 3 πλειάδες μπορούν να έχουν την ίδια τιμή. Το γνώρισμα GBs έχει 2 διακριτές τιμές, πράγμα που σημαίνει ότι

αρκετές πλειάδες έχουν την ίδια τιμή. Ο βελτιστοποιητής θεωρεί ότι τα γνωρίσματα είναι ανεξάρτητα το ένα από το άλλο και δεν ξέρει αν το γνώρισμα GBs κατανέμονται ομοιόμορφα ή όχι στις πλειάδες του Manufacturer = Apple. Πράγμα που σημαίνει ότι όταν θα υπολογιστεί η Manufacturer = Apple ^ GBs=8 δε μπορεί να γνωρίζει ότι υπάρχει συσχέτιση μεταξύ των 2 γνωρισμάτων, δηλαδή όλα τα Apple έχουν 8 GBs.

δ) Για να χειριστούμε το πρόβλημα που προέκυψε στο υποερώτημα γ, κάποιος θα μπορούσε να προτείνει να κρατάμε στατιστικά για το V(Phones, Manufacturer-GBs), όπου το Manufacturer-GBs χρησιμοποιείται σαν ένα ενιαίο πεδίο που σχηματίζεται από την συνένωση των τιμών του Manufacturer και του GBs. Εξηγήστε σύντομα το πόσο καλή είναι η παραπάνω ιδέα και αν θα μπορούσε η όχι να υλοποιηθεί σε ένα DBMS.

Λύση

Αν κρατάμε στατιστικά για το V(Phones, Manufacturer-GBs), δηλαδή να το βλέπουμε σαν ενιαίο πεδίο τότε θα έχουμε ότι οι διακριτές του πλειάδες είναι 3, Άρα , ο αναμενόμενος αριθμός πλειάδων από τον βελτιστοποιητή θα είναι T(Phones)/ V(Phones, Manufacturer-GBs) = 9/3 = 3. Βλέπουμε η εκτίμηση είναι ακριβής και καλύτερη του Manufacturer=Apple ^ Model=G3. Για το αν θα μπορούσε να υλοποιηθεί σε ένα DBMS πρέπει να σκεφτούμε ότι η κράτηση στατιστικών στοιχείων και η ενημέρωση τους είναι ένα επιπλέον κόστος και θα ήταν μια εξαιρετικά ακριβή διαδικασία αν θα έπρεπε να το κάνουμε σε όλα τα ζεύγη γνωρισμάτων. Για αυτό το λόγο αυτή η υλοποίηση της παραπάνω ιδέας θα πρέπει να γίνει στα ζεύγη που υπάρχει κάποια συσχέτιση μεταξύ τους. Όμως, στη σχετικά μικρή σχέση που έχουμε είναι εύκολο να καταλάβουμε αν υπάρχουν συσχετίσεις μεταξύ των γνωρισμάτων. Σε μια γενική περίπτωση βάσης δεδομένων αυτό είναι σχεδόν αδύνατο να συμβεί.

Σημειώματα

Σημείωμα αναφοράς

Σημείωμα Αδειοδότησης

Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Κρήτης» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.