Πληροφορική & Τηλεπικοινωνίες

Κ23α - Ανάπτυξη Λογισμικού Για Πληροφοριακά Συστήματα

Χειμερινό Εξάμηνο 2018 – 2019

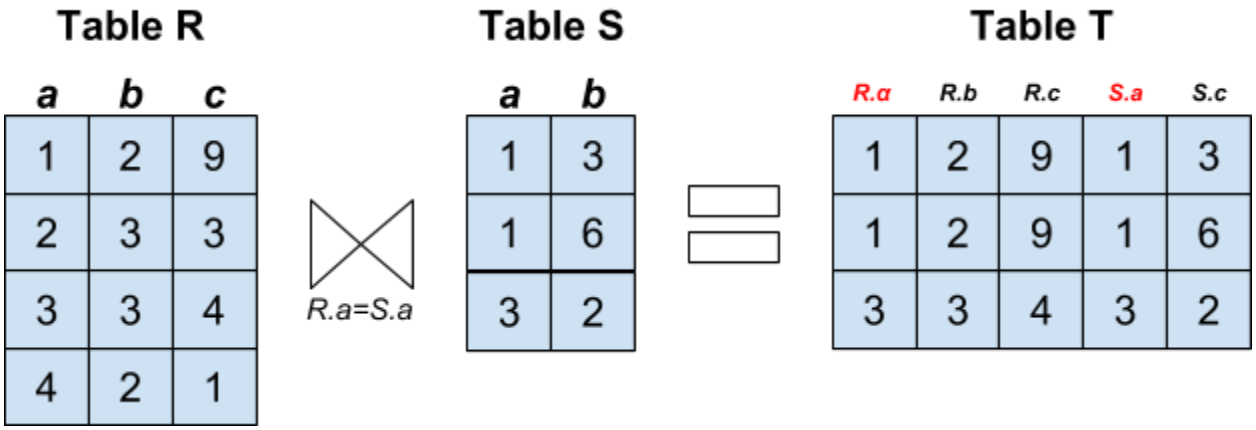
Καθηγητής Ι. Ιωαννίδης

Άσκηση 1 – Παράδοση: Δευτέρα 05 Νοεμβρίου 2018

Με την εξέλιξη των υπολογιστών τα τελευταία χρόνια, η σχέση τιμής ανά GB για μνήμες RAM είναι πολύ μικρή ενώ ταυτόχρονα αυξάνονται ολοένα και περισσότερο ο αριθμός των πυρήνων σε κάθε CPU. Σήμερα είναι σύνηθες να συναντάμε Servers με 256 GB RAM και 20 επεξεργαστικούς πυρήνες. Σε αυτά τα νέα δεδομένα hardware προσπαθούν να προσαρμοστούν οι σύγχρονες σχεσιακές βάσεις δεδομένων. Σε αυτο το εξάμηνο θα προσπαθήσουμε να δημιουργήσουμε ένα υποσύνολο μιας βάσης δεδομένων που διαχειρίζεται δεδομένα που βρίσκονται εξ ολοκλήρου στη RAM. Στο πρώτο μέρος θα ασχοληθούμε με την υλοποίηση του βασικό σχεσιακού τελεστή που θα χρειαστούμε για τα μετέπειτα κομμάτια της άσκησης που είναι ο τελεστής ζεύξης ισότητας.

Τελεστής ζεύξης

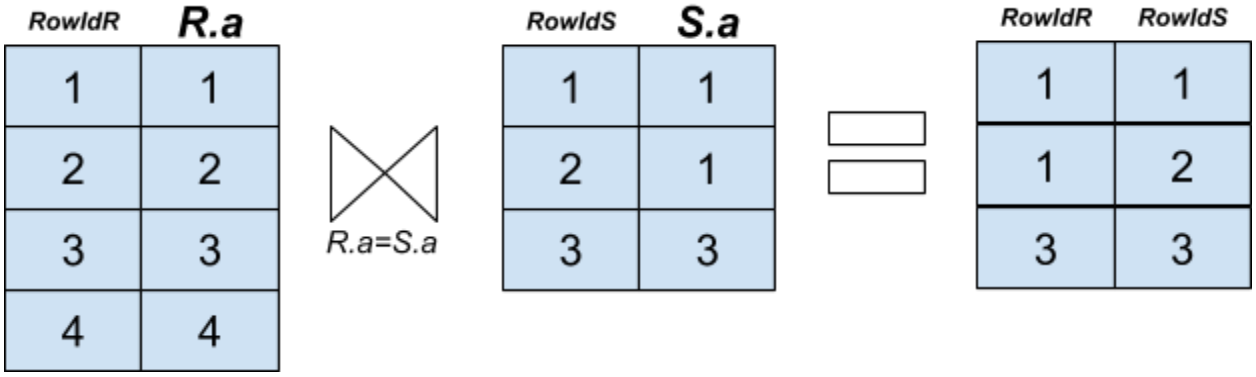
Πρίν προχωρήσουμε στις δομές που θα χρειαστείτε να φτιάξετε για την πρώτη άσκηση θα δούμε ένα παράδειγμα με το πώς δουλεύει ένας τελεστής ζεύξης ισότητας. Στην πιο κάτω εικόνα, φαίνεται το αποτέλεσμα της πράξεις $R.\alpha = S.\alpha$ μεταξύ δύο σχεσιακών πινάκων (R, S):



Η πιο πάνω απεικόνιση θεωρεί ότι οι πίνακες αποθηκεύονται στην μνήμη κατα σειρές (row store). Αυτό πρακτικά σημαίνει, ότι το κάθε στοιχείο μιας στήλης θα απέχει από το επόμενο του, τόσα στοιχεία μακριά όσες είναι και ο αριθμός των στηλών της αντίστοιχης σχέσης. Αυτο σε αρκετες περιπτωσεις είναι χρονοβόρο, γιατί καλούμαστε να διαχειριζόμαστε πολύ περισσότερη

μνήμη, από όση στην πράξη χρειαζόμαστε. Μια άλλη τεχνική, αποθηκεύει τις σχέσεις ανά στήλες. Αυτό σημαίνει, ότι στη μνήμη τα στοιχεία μιας στήλης είναι σειριακά το ένα δίπλα στο άλλο. Με αυτόν τον τρόπο, αν θέλουμε να διαβάσουμε μια στήλη, “ακουμπάμε” πολύ λιγότερη μνήμη.

Για να εκμεταλλευτούμε την αποθήκευση ανά στήλες θα θέλαμε ο τελεστής ζεύξης να χρησιμοποιεί μόνο τις στήλες που γίνεται η πράξη και το αποτέλεσμα να είναι σε τέτοια μορφή ώστε να μπορούμε να ανακτήσουμε όλες τις στήλες από τους πίνακες που παίρνουν μέρος. Για τον λόγο αυτό εισάγουμε την έννοια του rowId. Το rowId είναι μια ένδειξη του αριθμού της γραμμής που ανήκει ένα συγκεκριμένο στοιχείο μιας στήλης.. Με αυτόν τον τρόπο, μπορούμε να αποθηκεύσουμε κάθε στήλη μιας σχέσης, σαν μια ξεχωριστή σχέση με δύο στήλες, όπου η πρώτη στήλη είναι το rowId και η δεύτερη οι πραγματικές τιμές της στήλης. Το αποτέλεσμα της ζεύξης δύο τέτοιων σχέσεων, είναι μια σχέση με δύο στήλες όπου αναφέρονται στα ζευγάρια από rowIds τις πρώτης και της δεύτερης σχέσης που ταιριαζουν με βάση την συνθήκη της ισότητας. Η πιο κάτω εικόνα, απεικονίζει την εκτέλεση του τελεστή της ζεύξης του προηγούμενου παραδείγματος, χρησιμοποιώντας αποθήκευση ανα στήλες.

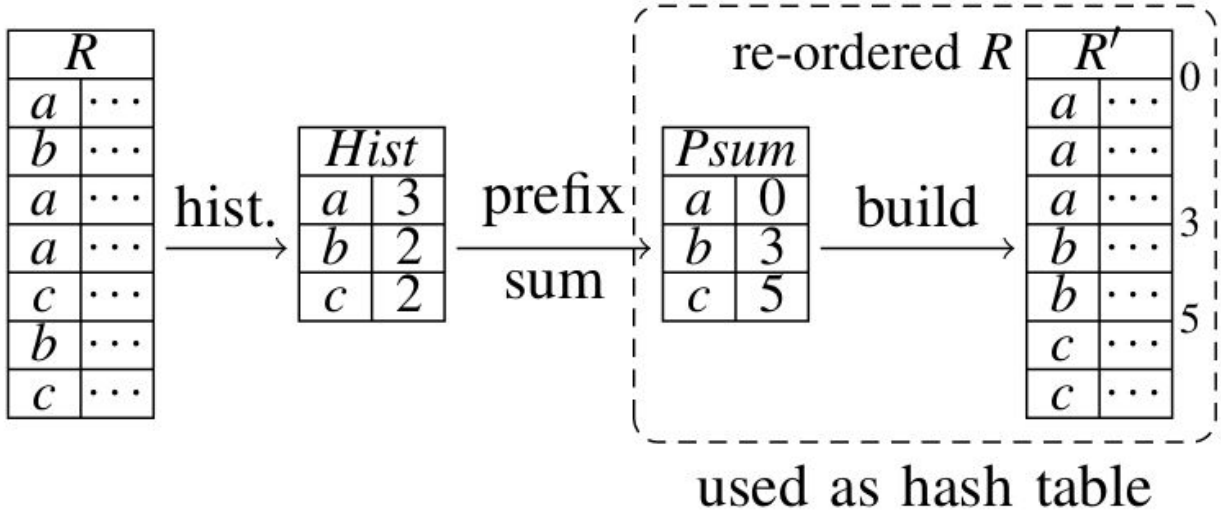


Radix Hash Join

Στο πρώτο μέρος της άσκησης καλείστε να υλοποιήσετε των τελεστή ζεύξης ισότητας υποθέτοντας ότι τα δεδομένα σας είναι αποθηκευμένα ανά στήλες. Πιο συγκεκριμένα καλείστε να υλοποιήσετε τον Radix Hash Join (RHJ) αλγόριθμο. Η ιδέα του RHJ είναι να κομματιάσει τα δεδομένα από τις δύο σχέσεις σε τόσους κάδους, έτσι ώστε ο μεγαλύτερος σε μέγεθος κάδος να μπορεί να χωράει στην Cache του επεξεργαστή. Οι κάδοι προκύπτουν εφαρμόζοντας στα δεδομένα των δύο σχέσεων την ίδια συνάρτηση κατακερματισμού H1. Εφόσον έχουμε κομματιάσει τα δεδομένα μας σε κάδους εφαρμόζοντας την ίδια συνάρτηση κατακερματισμού μπορούμε να συγκρίνουμε μόνο τα δεδομένα που ανήκουν στους ίδιους κάδους.

Πιο αναλυτικά, ο RHJ χωρίζεται σε τρεις φάσεις, τη φάση της τμηματοποίησης, τη φάση του χτισίματος ευρετηρίων σε κάθε κάδο και τη φάση της σύγκρισης. Κατά τη φάση της τμηματοποίησης χρησιμοποιούμε σαν συνάρτηση κατακερματισμού (H1), τον αριθμό που δίνουν τα *v* λιγότερο σημαντικά bits κάθε στοιχείου. Για παράδειγμα, αν έχουμε τον δυαδικό

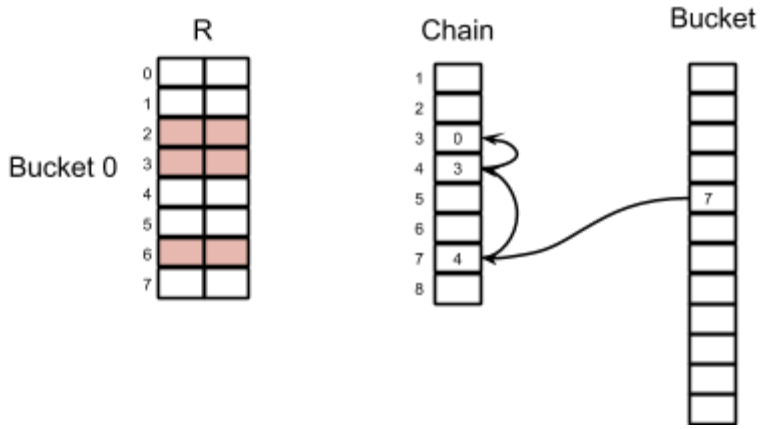
αριθμό 10010100, και το ν ισούται με 3, τοτε H1(10010100) = 100 δηλαδή 4. Κατα αυτόν τον τρόπο μπορούμε να κοματιάσουμε ένα πίνακα σε 2^{v} κάδους. Διαλέγοντας κάποιο ν αρκετά μεγάλο μπορούμε να σπάσουμε τον πίνακα σε αρκετα μεγάλο πλήθος κάδων έτσι ώστε τα δεδομενα κάθε κάδου να χωράνε στην Cache του επεξεργαστή. Για να γίνει αυτή η διαδικασία αποδοτικά χρειάζεται πρώτα να δεσμεύσουμε ένα καινούργιο πίνακα με μέγεθος όσο ο αρχικός. Στον πίνακα αυτό θα αποθηκεύσουμε σε σειρα τα δεδομένα του κάθε κάδου. Για να το κάνουμε αυτό χρειάζεται να ξέρουμε σε ποια θέση του νέου πίνακα ξεκινάνε τα δεδομένα του κάθε κάδου. Για τον λόγο αυτό δημιουργούμε ένα πίνακα (ιστόγραμμα) 2^{v} θέσεων όπου στην κάθε θέση του κρατάμε το πλήθος των στοιχείων που υπάρχουν σε αυτόν τον κάδο. Στη συνέχεια θα δημιουργήσουμε το αθροιστικό του ιστόγραμμα που θα δείχνει στη θέση από όπου θα ξεκινάει ο κάθε καδος. Έχοντας αυτό το ιστόγραμμα μπορούμε με ένα πέρασμα του αρχικού πίνακα να γράψουμε τα δεδομένα στη σωστή θέση του νέου πίνακα. Η φάση της τμηματοποίησης φαίνεται στο πιο κάτω περιληπτικό σχήμα:



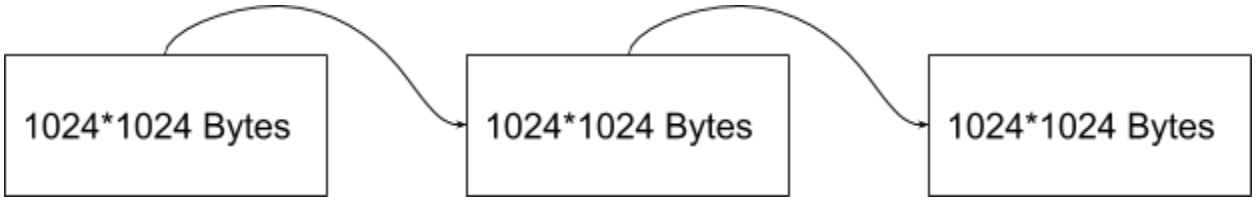
Εφόσον έχουμε δημιουργήσει τους κατάλληλους κάδους, και στις δύο σχέσεις, με την ίδια συνάρτηση κατακερματισμού, αυτό που μένει, είναι να συνδυάσουμε τους κάδους που κρατάνε δεδομένα, στα οποία η συνάρτηση κατακερματισμού έδωσε την ίδια τιμή. Αυτο δηλαδή που μένει, είναι να επιλέξουμε τα ζευγάρια κάδων που ταιριάζουν από την σχέση R και S, να χτίσουμε τα κατάλληλα ευρετήρια σε έναν από τους δύο κάδους και να δούμε ποια δεδομένα της μιας σχέσης ταιριάζουν με τα δεδομένα της άλλης.

Το ευρετήριο που θα χτίσουμε σε ένα κάδο έχει τη μορφή ενός πίνακα κατακερματισμού. Πιο αναλυτικά, εκτός από τον πίνακα που έχει τα δεδομένα ενός bucket θα χρησιμοποιήσουμε άλλους δύο εξωτερικούς πίνακες. Ό ένας πίνακας, που θα τον ονομάσουμε bucket, θα έχει τόσες θέσεις όσες και το εύρος της συνάρτησης κατακερματισμού που θα επιλεγεί, και ο δεύτερος (chain) θα έχει μέγεθος ακριβώς όσα και τα δεδομένα του εκάστοτε κάδου. Η συνάρτηση κατακερματισμού που θα επιλεγεί (H2) θα πρέπει να είναι διαφορετική από εκείνη που χρησιμοποιήθηκε κατα την διάρκεια της τμηματοποίησης. Ο πίνακας bucket θα κρατάει στη θέση i τη θέση του τελευταίου στοιχείου που ανήκει στο bucket i, ενώ ο πίνακας chain θα

κρατάει στη θέση bucket[i] τη θέση του αμέσως προηγούμενου στοιχείου που ανήκει στη θέση bucket[i] κ.ο.κ.. Το πιο κάτω σχήμα δείχνει τη δομή που μόλις περιγράφηκε:



Για να πάρουμε τα τελικά αποτελέσματα, θα πρέπει να σαρώσουμε τα δεδομένα του κάδου που δεν έχει ευρετήριο και να ρωτήσουμε το ευρετήριο του άλλου κάδου. Επειδή δεν ξερουμε εκ τον προτέρων πόσα θα είναι τα αποτελέσματα, τα αποτελέσματα της ζεύξης θα γραφτούν σε μια νέα δομή η οποία έχει τη μορφή λίστας. Κάθε κάδος της λίστας θα κρατάει έναν buffer με μέγεθος 1MB και έναν δείκτη στο επόμενο bucket. Η λογική είναι πως γράφουμε δεδομένα σε έναν buffer. Μόλις ο buffer γεμίσει τότε θα εισάγουμε έναν καινούριο κάδο στην λίστα κ.ο.κ.. Η δομή αυτή θα έχει την μορφή που φαίνεται στο πιο κάτω σχήμα:



Πρότυπα συναρτησεων και δομών

Πιο κάτω δίνονται οι ορισμοί των συναρτήσεων και των βασικών δομών τους που καλείστε να υλοποιήσετε:

```
/** Type definition for a tuple */
struct tuple {
        int32_t key;
        Int32_t payload;
};

/**
 * Type definition for a relation.
```

```
 * It consists of an array of tuples and a size of the relation.
 */
struct relation {
  tuple  *tuples;
  uint32_t num_tuples;
};

/**
 * Type definition for a relation.
 * It consists of an array of tuples and a size of the relation.
 */
struct result {
 ...
};

/** Radix Hash Join**/
result* RadixHashJoin(relation *relR, relation *relS);
```

Οι πιο πάνω ορισμοί είναι ενδεικτικοί. Αν θεωρείτε ότι σας περιορίζουν μπορείτε να τους αλλάξετε.

Παράδοση εργασίας

Η εργασία είναι ομαδική, **2 ή 3 ατόμων**.

Προθεσμία παράδοσης: 05/11/2017.

Γλώσσα υλοποίησης: C / C++ χωρίς χρήση stl.

Περιβάλλον υλοποίησης: Linux (gcc 5.4+).

Παραδοτέα: Η παράδοση της εργασίας θα γίνει με βάση το τελευταιο commit πριν την προθεσμία υποβολής στο git repository σας. **Η χρήση git είναι υποχρεωτική**.

Αναφορά

- Cagri Balkesen, Jens Teubner, Gustavo Alonso, and M. Tamer Özsu. Main-Memory Hash Joins on Multi-Core CPUs: Tuning to the Underlying Hardware. *Proc. of the 29th International Conference on Data Engineering (ICDE 2013)*, Brisbane, Australia, April 2013.