K23α- Ανάπτυξη Λογισμικού Για Πληροφοριακά Συστήματα

Τρίτο Μέρος

Δανάη Ρουμελιώτη, ΑΜ: sdi1600145

Ιωαννης Βάιος Δημοπουλος, ΑΜ: sdi1600043

- **Επιλογή Δομών:**

  - **Tuple:** Αυτή η δομή περιέχει το στοιχείο και σε ποιά γραμμή βρίσκεται (row_id).
  - **Shell:** Είναι μία δομή που “κρατάει” τον αριθμό των γραμμων και στηλών των στοιχείων και έναν πίνακα με δείκτες σε Tuple (καθε αρχειο αντιστοιχεί σε ένα Shell).
  - **Table:** Είναι μία δομή που “κρατάει” τον αριθμό των Shells που περιέχει και έναν πίνακα με δείκτες σε Shell..

  - **Histogram:** Είναι μία δομή που περιέχει έναν πίνακα με 256 θέσεις (όσες και οι πιθανές τιμές ενός byte) που χρησιμοποιείται ως map. Δηλαδή, η ποσότητα κάθε τιμής βρίσκεται στη θέση με index την τιμή αυτη (έτσι έχουμε άμεση προσβαση).

  - **Prefix_Sum:** Ομοίως είναι μία δομή που περιέχει έναν πίνακα με 256 θέσεις που χρησιμοποιείται ως map. Δηλαδή, το offset κάθε τιμής βρίσκεται στη θέση με index την τιμή αυτη.

  - **List Node:** Περιέχει έναν δισδιάστατο πίνακα LIST_SIZE γραμμών και 2 στηλών (μία για κάθε row_id ενός Relation), έναν ακέραιο counter

(ώστε να ξέρουμε πόσες εγγραφές έχει μέσα ο κόμβος) και έναν δείκτη next για τον επόμενο κόμβο.
- **Result_List:** Έχει έναν δείκτη start που δείχνει στον 1ο κόμβο της και έναν last που δείχνει στον τελευταίο της (έτσι αποφεύγουμε την προσπέλαση όλης της λίστας όταν θέλουμε να προσθέσουμε έναν κόμβο).

- **Query:** Είναι μία δομή που περιέχει 3 strings (relations, predicates, projections) και εναν δείκτη σε επόμενο query.
- **Batch:** Είναι μία δομή που “κρατάει” έναν δείκτη στο πρώτο query του batch (δηλαδή στον 1ο κόμβο της ουράς), εναν στο τελευταιο (έτσι αποφεύγουμε την προσπέλαση, όταν θέλουμε να προσθέσουμε έναν κόμβο) και το πόσα queries εχει αυτο το batch.

- **Parsed_Query:** Είναι μία δομή που περιέχει έναν πίνακα με relations, εναν με joins, εναν με filters κι έναν με projections.

- **Filter_Result:** Είναι μία δομή που “κρατάει” έναν πίνακα με row_ids, την σχέση που φιλτραραμε και τον αριθμό των αποτελεσμάτων.
- **Filter_Outcome:** Είναι μία δομή που “κρατάει” έναν πίνακα δείκτες σε Filter_Result και τον αριθμό των φιλτρων που εκτελέσαμε.

- **Execution_Queue_Node:** Είναι μία δομή που περιέχει έναν δείκτη σε Join και εναν δείκτη σε επόμενο κόμβο-join.

    - **Execution_Queue:** Έχει έναν δείκτη head που δείχνει στον 1ο κόμβο και έναν tail που δείχνει στον τελευταίο της.

- **Initializer:**
- **Argument_Manager:**
    - *Check_Arguments_Number:*
        - Ελέγχει για τον αριθμό των args.
    - *Go_Through_Argv_And_Get_Filenames:*
        - Ελέγχει ότι τα flags ότι έχουν δοθεί σωστά και αποθηκεύει τα ονόματα των αρχείων για διάβασμα.
    - *Get_Argument_Data:*
        - Αφού κληθεί η Check_Arguments_Number και η Go_Through_Argv_And_Get_Filenames, δημιουργείται μια δομή, ArgumentData, που περιέχει τα ονόματα των input-files.
- **Table_Allocator:**
    - *Create_Table_Allocator:*
        - Δημιουργεί μια δομή που αποθηκεύει το directory και το file που περιεχει τα ονοματα των αρχείων-σχεσεων.
    - *Allocate_Table:*
        - Καλεί την Count_Shells για να ξέρει πόσα είναι τα αρχεία-σχέσεις και δημιουργεί μια δομή Table.
    - *Fill_Table:*
        - Ανοίγει το Init αρχείο για διάβασμα και για καθε αρχειο-σχεση καλεί την FIll_Shell, η οποία κάνει allocate και γεμίζει το shell με τα δεδομένα.
        - Ανοίγει το αρχείο για διάβασμα, καλεί την Read_from_File για να διαβασει ποσες γραμμες και στήλες …για να “γεμίσει” την σχέση απο το αρχειο και κλείνει τον fp που είχε ανοίξει.

- **Work_Reader:**

    - *Read_next_Batch:*
        - Δημιουργεί ένα Batch, διαβάζει γραμμή-γραμμή τα queries και τα εισάγει στην λιστα.
- **Work_Executor:**
    - *Start_Work:*
        - Ανοίγει το work αρχειο για διαβάζει batch-batch, καλώντας την Read_next_Batch, οταν διαβασει ενα batch αρχίζει κ κάνει pop ενα-ενα τα queries για να τα εκτελέσει.

- **<u>Query_execution:</u>**

- **Query_parser:**
    - *Parse_Query:*
        - Δημιουργεί μια Parsed_Query δομή και την γεμίζει καλώντας τις Setup_Relations, Setup_Joins_and_FIlters, Setup_Projections.
- **Preparator:**
    - *Prepare_Execution_Queue:*
        - Δημιουργεί μια Execution_Queue, ελέγχει για Self_Joins (και τα βάζει πρώτα αν βρει).

- **Query_executor:**
    - *Execute_Query:*
        - Κανει parse το Query και αποθηκεύει με ποιες σχέσεις θα δουλέψουμε στο συγκεκριμενο query, καλεί την Prepare_Execution_Queue για να ετοιμάσει με ποια σειρά θα εκτελεστούν τα joins, καλεί την Execute_Filters (καθως τα φιλτρα θα εκτελεστουν πρωτα και δεν μας νοιαζει η σειρά) και τελος.

- **Filter_executor:**
    - *Execute:*
        - Προσπελαύνει μια φορα τον πίνακα με τα δεδομένα της σχέσης που θελουμε και κραταμε το row_id οποιας γραμμής πληρεί το φίλτρο (στην στήλη που θέλουμε).

- *Execute_Filters:*
  - Κανει allocate εναν πινακα με δείκτες σε Filter_Result και για καθε φιλτρο σε ένα query: βρισκει την σχέση που πρέπει και καλεί την Execute. Τελος, κανει allocate μια δομη Filter_Outcome, βαζει τον δείκτη Filter_Outcome->Filter_Result να δείχνει στον πίνακα με τα απολεσματα μας και αποθηκεύει στο Filter_Outcome->num_of_filters ποσα φιλτρα εκτελέσαμε.

● <u>Join_execution:</u>

- **Histogram:**
  - *Fill_Histogram:*
    - Αποθηκεύει, προσπελαύνοντας το Relation και υπολογίζοντας την τιμή του byte που θέλουμε, ένα-ένα τα tuples στον πίνακα του ιστογράμματος.
  - *Create_Histogram:*
    - Κάνει allocate μνήμη για ένα Histogram ανάλογα με το όρισμα (num_of_tuples) που της έχει δοθεί.
  - *Get_Histogram:*
    - Αφού καλέσει την Count_*Histogram_Rows* (για να γνωρίζουμε τον αριθμό των tuples) και την Create_Histogram, καλεί την Fill_Histogram για να αποθηκεύσει τα tuples στο ιστόγραμμα.

- **Prefix_Sum:**
  - *Fill_Psum:*
  - Αποθηκεύει, προσπελαύνοντας το ιστόγραμμα και προσθέτοντας τις προηγούμενες ποσότητες, ένα-ένα τα tuples στον πίνακα του Psum.
- *Create_Psum:*
  - Κάνει allocate μνήμη για ένα Psum ανάλογα με το όρισμα (num_of_tuples) που της έχει δοθεί.

- *Get_Psum:*
    - Αφού καλέσει την Create_Psum, καλεί την Fill_Psum για να αποθηκεύσει τα tuples στο Psum.

- **Relation_Sorting:**
    - *Find_Byte_Value:*
        - Παίρνει μια τιμή 8 byte και το byte που θέλουμε και με την χρήση shift και μάσκας υπολογίζει την τιμή του byte αυτού.
    - *Get_index_map:*
        - Δημιουργεί ένα αντίγραφο του Prefix Sum το οποίο θα χρειαστούμε καθώς θέλουμε να αλλάξουμε τις τιμές του χωρίς να πειράξουμε το αρχικό.
    - *Copy_Relation:*
        - Αρχικά, καλεί την Get_index_map ώστε να έχει ένα αντίγραφο του Prefix Sum. Στην συνέχεια, προσπελαύνοντας στο αρχικό Relation και βρίσκοντας την τιμή του byte που θέλουμε (Find_Byte_Value) και το offset (απο το το Index map), τοποθετεί κάθε εγγραφή στο temporary Relation στην σωστή θέση (ανάλογα με το 1ο ή 2ο κλπ byte). Τέλος, κάθε φορα αυξάνει την τιμή (δηλαδή το offset) του συγκεκριμένου byte στο Index map.
    - *Sort_Relation:*
        - Αρχικά, ελέγχει αν το Relation χωράει σε 64 kb.
        - Αν ναι τότε επιστρέφει και απο την main καλείται η quicksort.
        - Αν όχι τότε δημιουργείται το ιστόγραμμα και το Prefix Sum για αυτό το Relation και καλεί την Copy_Relation. Όταν επιστρέψει η Copy_Relation έχει βάλει τις εγγραφές του αρχικού Relation στο temporary, ταξινομημένες με το 1ο byte (την πρωτη φορα, την 2η φορα θα ναι με το 2ο byte κλπ). Έτσι, με τη χρήση της memcopy αντιγράφουμε το ταξινομημένο περιεχόμενο της temp-Relation στην αρχική. Μετά για κάθε bucket της Relation καλείται αναδρομικά η Sort_Relation ξανά. Τέλος,

αποδεσμεύουμε τα ιστόγραμμα και Prefix Sum καθώς η χρησιμότητα τους τελείωσε.

- *Sort:*
  - Δημιουργεί μία νέα Relation (κενη, την οποία θα χρησιμοποιήσουμε ως temporary Relation για να κανουμε την ταξινόμηση ανα byte), καλεί την Sort_Relation και στο τέλος την αποδεσμεύει καθώς δεν τη χρειαζόμαστε πλέον.

- **Join:**
  - *Join:*
    - Δέχεται δύο πλήρως ταξινομημένες σχέσεις, A και B, και με την βοήθεια δύο δεικτών για κάθε σχέση εκτελούμε την πράξη join. Επίσης, η while loop τερματίζει όταν ο cntA (μετρητής για την σχέση A) γίνει ίσος με τον αριθμό εγγραφών στο Relation A, όταν δηλαδή έχουμε προσπελαύνει όλη την σχεση.
    - *Σημείωση: χρησιμοποιούμε τους δύο δείκτες, για κάθε σχέση, ώστε να αποφύγουμε να προσπελαύνουμε την σχέση B για κάθε στοιχείο της σχέσης A.*
    - Κάθε φορά που έχουμε ένα αποτέλεσμα της πράξης join, εισάγουμε τα δύο row_ids στην λίστα.
    - Τέλος, όταν έχουμε πλέον βγει απο την while loop, και έχουμε όλα τα αποτελέσματα στην λίστα, καλούμε την Print_List η οποία γράφει αυτά τα αποτελέσματα σε ένα output αρχείο και αποδεσμεύουμε την λίστα.

-Εκτέλεση Join ενός query:

- **Ενδιάμεση Δομή Αποτελεσμάτων**: Η ενδιάμεση δομή που κρατούνται τα αποτελέσματα μετα απο κάθε join είναι η Intermediate Result. Αναλυτικά τα πεδία της δομής είναι τα εξής:
    - relations_in_result: Αποτελεί έναν πίνακα 4 θέσεων (όσος και ο μέγιστος αριθμός σχέσεων μέσα σε ένα query). Ρόλος αυτου του πίνακα ειναι να μας δίνει μια αναπαράσταση που να αντιπροσωπεύει το ποιες σχέσεις συμπεριλαμβάνονται στο αποτέλεσμα (πχ άμα η σχέση 1 υπάρχει μέσα τότε η τιμή relation_in_result[0] θα έχει την τιμή 1 ενώ σε αντιθετη περίπτωση την τιμή 0).
    - num_of_results: Μεταβλητή στην οποία κρατάμε τον αριθμό τον αποτελεσμάτων που υπάρχουν στο ενδιάμεσο αποτέλεσμα.
    - num_of_relations: Μεταβλητήσ την οποία κρατάμε τον αριθμό των σχέσεων που συμμετέχουν στο ενδιάμεσο αποτέλεσμα.
    - row_ids: Ουσιαστικά αποτελεί τον βασικό πίνακα με τα αποτελέσματα. Πρόκειται για έναν δισδιάστατο πίνακα οπου σε κάθε γραμμή του θα έχουμε όλες τις ν-αδες αποτελεσμάτων. Θα εξηγηθεί παρακάτω πως ακριβώς επιτυγχάνεται αυτό.
    - Used_Columns: Προκειται για μια δομή λίστας στην οποια κάθε κομβος κραταει ολα τα ζευγάρια σχεσης-στήλης που έχουν ιδι χρησιμοποιηθεί σε κάποιο προηγούμενο join. Με τον τρόπο αυτό αμα σε καποιο join ξαναχρειαστει να χρησιμοποιήσουμε μια στηλη ξέρουμε ότι μπορούμε εύκολα να την ανακτήσουμε ταξινομημένη μεσω της ενδιαμεσης δομής.(Θα δοθει αναλυτικότερη επεξήγηση στη συνέχεια).

- **Εκτέλεση Join:** Στο ανώτερο επίπεδο της διαδικασίας έχουμε την εκτέλεση της Execute_Join. Επαναληπτικά εξάγουμε από την ουρά με που έχουμε βάλει πριν τα joins και ελέγχουμε σε ποιά απο τις 3 περιπτώσεις που μπορούν να υπάρξουν υπάγεται, δηλαδή να είναι join που αποφορα στήλες από την ίδια σχέση, να είναι join ανάμεσα σε σχέσεις που υπάρχουν και οι 2 στον ενδιάμεσο αποτέλεσμα είτε να είναι join στο οποίο μόνο η μια απο τις 2 σχέσεις θα υπάρχει στον ενδιάμεσο αποτέλεσμα.
    - Execute_Self_Join: Τα self joins σε κάθε περίπτωση θα είναι τα πρώτα joins που θα εκτελεστούν μέσα στο query. Για το λόγο αυτό δεν υπάρχει ανάγκη να εμπλέξουμε το ενδιάμεσο αποτέλεσμα στη διαδικασία αυτή. Απλός αφαιρούμε οσα αποτελέσματα δεν υπακούν στο join απο τον πινακα που έχει προκύψει από τα φίλτρα
    - Execute_Scan_Join: Η συνάρτηση αυτή θα δημιουργήσει δύο πίνακες με tuples της μορφής (row_id, value) οπου row_id είναι το row_id σε σχέση με το ενδιάμεσο αποτέλεσμα και value η τιμη στον αρχικό πίνακα. Στη συνέχεια διατρέχουμε τους 2 πινακαες παραλληλα και κρατάμε τα αποτελέσματα σε εναν νεο πινακα. Τελος ξαναδημιουργούμε τον πινακα της ενδιάμεσης δομής βάση των αποτελεσμάτων.
    - Execute_Normal_join: Η συνάρτηση αυτή αναλαμβάνει να εκτελέση οποιαδήποτε άλλη περίπτωση join.Οι περιπτώσεις που προκύπτουν είναι είτε να υπαρχει η σχέση Α στο αποτέλεσμα είτε να υπάρχει η σχέση Β είτε να μην υπάρχει καποια απο τις δυο. Στις πρώτες δύο περιπτώσεις η εκτέλεση είναι πανομοιότυπη. Αρχικά ελέγχουμε ποια σχέση πρέπει να δημιουργηθεί από τον αρχικό μας πίνακα και ποια απο το ενδιάμεσο αποτέλεσμα. Και στις δύο περιπτώσεις κατασκευάζουμε μια δομή Relation_Ptr την οποία και δέχεται η join που είχε υλοποιηθεί στο 1ο μέρος. Μια σημαντική λεπτομέρεια ειναι το οτι τα row_ids στο Relation που αφορά τη νέα σχέση είναι τα row_ids που έχει και στον αρχικό πίνακα. Αντιθέτως για τη σχεση που δημιουργούμε βάση του ενδιάμεσου αποτελέσματος τα row_ids είναι τα

row_ids του ενδιάμεσου αποτελέσματος. Για παράδειγμα έστω ότι έχουμε σαν ενδιάμεσο αποτέλεσμα το παρακάτω

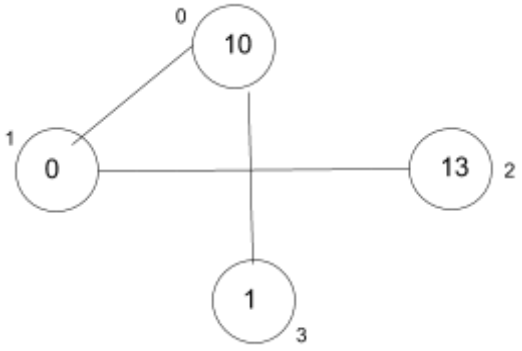
- 

Έστω ότι στο join που ακολουθεί γίνεται join h σχέση 0 με τη σχέση 2. Στη συνάρτηση Make_Relation_From_Intermediate_Array αυτό που θα γίνει για κάθε row_id της σχέσης 0 να εισάγουμε το value του column που γίνεται join και να δώσουμε σαν row_id τη γραμμή που βρίσκεται αυτή η πληροφορία στον ενδιάμεσο αποτέλεσμα. Ετσι για παράδειγμα αν γινεται join 0.1 = 2.0 και η σχέση 0 στη στήλη 1 και στη γραμμη 22884 έχει σαν value το 5 αυτο που θα εισαχθεί στο relation είναι το (0,5). Αφου ταξινομηθούν τα δύο relations και γίνει το join στη λίστα με τα αποτελέσματα θα έχω μια αντιστοίχιση γραμμής στο ενδιάμεσο αποτέλεσμα με γραμμή στην σχέση που πρεπει να εισαχθεί. Με την πληροφορία αυτή καλείται η Setup_Intermediate_Array η οποια δεσμευει εναν νεο πινακα για το ενδιάμεσο αποτέλεσμα βάση του πλήθους αποτελεσμάτων που είχε η join με μια παραπάνω στήλη για τα row_ids της νέας σχέσης. Αφου ο πινακας γεμίσει διαγράφουμε τον παλιο και θέτουμε την τιμή row_ids της δομής Intermediate_result στον νέο πίνακα. Στην περίπτωση που δεν έχουμε καμία σχέση στο ενδιάμεσο αποτέλεσμα(είναι δηλαδη το πρώτο join), γινεται μια παρόμοια διαδικασια απλά δεν χρειάζεται να υπάρξει η αντιστοίχιση που εξηγήθηκε παραπάνω καθώς τα δυο Relations έχουν σαν row_ids αυτά του αρχικού πίνακα.

Optimizer:

- Ξεκινάμε εκτελώντας τα φίλτρα που εχουμε στο query, αφού γίνει αυτο καλούμε την Update_Filter_stats η οποία αναθέτει στις σχέσεις (που συμμετεχουν στο query) τα νέα στατιστικα.
- Ομοίως, εκτελούμε τα Self_Joins και καλούμε την Update_Self_Join_stats η οποία αναθέτει τα νέα στατιστικα.
- Στην συνέχεια, αρχικοποιούμε εναν πινακα, με τοσες θέσεις οσα τα joins του query, με δείκτες σε Execution_Queue και στο head της καθε μιας βαζουμε ενα join και τα στατιστικα του. Για κάθε join, βρίσκουμε το επομενο join που:
    - αρχικα να συνδέεται με αυτο
    - και να εχει το μικροτερο κόστος
- Στο τέλος, ελέγχουμε απο τους συνδυασμούς που “καταφεραν” να συμπεριλάβουν όλα τα joins την ουρα που ξεκιναει με το “φθηνοτερο” join.

  “Καταφεραν” εννοουμε το εξης:
  πχ.
  10 0 13 1| 0.2=1.0&1.0=2.2&0.1=3.0&0.1=209|0.2 2.5 2.2
  αυτο σαν γραφος απεικονίζεται ετσι



  Παρατηρούμε οτι δεν μπορουμε να ξεκινησουμε την εκτελεση των joins απο το 0.2=1.0 καθώς μετα θα πρεπει να παμε στο 1.0=2.2 ή 0.1=3.0.
  Αν παμε στο 1.0=2.2, τοτε το 0.1=3.0 δεν θα εκτελεστεί εφοσον δεν υπαρχει καποια ακμη (join) να το ενωσει με εναν απο του 2 κομβους του τελευταιου join.

  Ομοίως αν παμε στο 0.1=3.0, τοτε το 1.0=2.2 δεν θα εκτελεστεί εφοσον δεν υπαρχει καποια ακμη (join) να το ενωσει με εναν απο του 2 κομβους του τελευταιου join.

Συνεπώς, ακομα και αν το 0.2=1.0 εχει το μικροτερο κοστος δεν θα επιλέξουμε να αρχισουμε απο αυτο.

Χρονοι Εκτέλεσης:

OS: Ubuntu18.04
CPU: IntelCore i7-8550U 16GB
RAM: GDDR 2400Mhz

	Small_workload	Medium_workload
1 thread	11 sec	283 sec
2 thread	6.4 sec	191 sec
3 thread	5.31 sec	173 sec
4 thread	4.89 sec	-