MDE515:Database Management Systems Programming Project2

Konstantopoulos MIchalis, M1362
email: mkonstan@di.uoa.gr

Tsitsigos Dimitrios, M1283
email: dtsitsigk@di.uoa.gr

Εκτέλεση

Για να εκτελεστεί το πρόγραμμα δέχεται ως είσοδο με την ακόλουθη σειρά:

1. Το αρχείο με το catalog της βάσης
2. Το αρχείο με τα system requirements
3. Το αρχείο με τα query εκτέλεσης

Στην τέλος εκτέλεσης εμφανίζεται:

1. Το πλάνο με το materialization του query
2. Το annotated πλάνο του αρχικού materialization με το κόστος του
3. Το best annotated πλάνο που προκείπτει μετά τα optimizations

Για να εμφανιστεί κάθε πλάνο που προκείπτει μετά την εφαρμογή κάθε transformation υπάρχει η προαιρετική επιλογή “-all true” που δίνεται ως είσοδο μετά τα αρχεία, η οποία είναι by default σε false.

Parser

Για την δημιουργία του Parser χρησιμοποιήθηκαν τα εργαλεία javacc και java tree builder. Λαμβάνοντας ως είσοδο την γραμματική στην κατάλληλη μορφή(αρχείο Query.jj) δημιουργούν το AST, το οποίο το διατρέχουμε μία φορά για την αποθήκευση των operations με την μορφή που χρειαζόμαστε για την συνέχεια τις διαδικασίας. Αυτό επιτυγχάνεται μέσω του visitor pattern.

Δομή Αρχείου catalog

Το αρχείο αυτό περιέχει ολες τις πληροφορίες της βάσης μας(relations,attributes,indexes κτλ). Η δομή αυτού του αρχείου ειναι πολύ συγκεκριμένη, για αυτό θα σας δώσουμε κάποιες οδηγίες, έτσι ώστε να μην σας χτυπήσει σφάλμα στο διαβασμα του αρχείου catalog. Ενα παράδειγμα του αρχείου είναι αυτό :

dbname:exampledb

table:savingaccounts
attr:account_number int 2 1000 100,branch_name varchar(50) aaa www 50,branch_city varchar(50) aaa www 50,customer_id int 2 1000 100
primaryKey:customer_id,account_number
primaryIndex:customer_id hashing 200 3
secondaryIndex:account_number,customer_id B+tree 200 3
branch_name B+tree 200 3
foreignIndex:account_number chekingaccounts(customer_id) staticHashing 200 5
cardinality:4
sizeOfTuple:104
numberOfTuples:1000

table:chekingaccounts
attr:account_number int 2 1000 100,branch_name varchar(50) aaa www 50,branch_city varchar(50) aaa www 50,customer_id int 2 1000 100
primaryKey:account_number
primaryIndex:account_number B+tree 200 3
secondaryIndex:account_number staticHashing 200 3
foreignIndex:
cardinality:4
sizeOfTuple:104
numberOfTuples:100

Οδηγίες :

1. Θα πρεπει να υπάρχουν πάντα η πρωτη στηλη(που αναφερει τί ειναι η κάθε γραμμή)ανεξαρτήτου εάν έχει τιμή η όχι. Και θα πρεπει να ειναι μαζί με το ‘:’.
2. Όταν θέλετε να γράψετε μετα το ‘:’ δεν θα πρέπει να αφήνετε κενο, θα γράφετε αμέσως μετά.
3. Στην πρώτη γραμμή γράφουμε το ονομα της βασης και αφήνουμε μια κενή γραμμή. Ουσιαστικά από εκει και κάτω, ειναι τα relations της βάσης, και χωρίζονται παντα με κενή γραμμή
4. Τα attributes του relation χωριζονται με κόμμα, και η δομή του attribute είναι :

ονομα τύπος μικρότερη_τιμή μεγαλύτερη_τιμή distinct_values
με αυτη την σειρά και χωρισμένα από κομματα.

5. Το primary key μπορεί να είναι πολλά attributes και τα χωριζετε με κομματα, και χωρις κενά μεταξύ τους.
6. Η δομή του PrimaryIndex και secondaryIndex είναι ιδια :

όνομα_attribute(εάν έχει πολλα attribute, τα χωρίζουμε με κόμματα, χωρίς όμως κενά) τύπος distinct_values cost_factor(για trees είναι το ύψος, για hash είναι ο αριθμός τον overflow buckets).
Στην περίπτωση του secondaryIndexe, μπορούμε να έχουμε παραπάνω από ενα index, οποτε για καθε έναν, αλλάζουμε γραμμή(οπως φαίνεται και στο παραπάνω παράδειγμα).

7. Τα foreign indexes έχουν την δομή :

όνομα_attribute(εαν είναι πολλά βάζουμε κόμματα) relation(attributes) τύπος distinct_values cost_factor
΄ολα με κενά μεταξύ τους. Επειδή μπορούμε να εχουμε πολλά foreign indexes, ακολουθούμε την λογικη που είχαμε και για τα secondary indexes.

Δομη Αρχείου system requirements

Το αρχειο αυτό περιέχει της πληροφορίες του συστήματος που χρειαζόμαστε για να υπολογίσουμε τα διαφορα κόστοι. Ένα παράδειγμα του αρχείου είναι αυτό :
numberOfBuffers:100
sizeOfBuffers:1000
averageLatency:5.4
transferTime:3.4
timePenaltiesForWritingPages:4.2

Το μόνο που θα πρεπει να προσέξετε είναι να μην αφήσετε πουθενά κενά, ούτε κενή γραμμη.

Παραδοχές

1. Κάθε φορά που έχουμε ένα attribute της μορφής student.name πρέπει το πρόθεμα να συμφωνεί με την relation στην οποία δουλέυουμε.
Για παράδειγμα αν έχουμε sel[relation1.attribute>0](relation2) πρέπει relation1==relation2.
Αν έχουμε απλά sel[attribute>0](relation) ξέρουμε ότι το attribute αναφέρεται στην relation.

2. Στην περίπτωση του join, στα conditions το 1ο μερος της πράξης αναφέρεται παντα στην 1η relation και το 2ο πάντα στην 2η relation. Για παράδειγμα,
join[attribute1=attribute2 and attribute3=attribute4](relation1)(relation2)
Εδώ θα πρέπει να ισχύει ότι attribute1,attribute3 ανήκουν στην relation1 και attribute2,attribute4 ανήκουν στην reation2.

3. Το αποτέλσμα του join θεωρούμαι ότι έχει όλα τα non-join attributes και από τα join attributes κρατάμε την ονομάσία του 1ου. Για παράδειγμα έχουμε relation1 με attribute1,attribute3 και relation2 με attribute2,attribute4. Το output της join[attribute1=attribute2](relation1)(relation2) θα περιέχει τα attribute1,attribute3,attribute4.

Υπολογισμός κόστους

- ***Projection:***

Το κόστος έχει να κάνει με την ανάγνωση όλων των tuples ενός relations και την εξάλειψη duplicates. Duplicates elimination γίνεται με :

1. relation sort
2. relation hash

Δεν έχουμε κόστος για duplicate elimination όταν:

1. Κάποιο/α από τα projection attributes είναι primary key, οπότε δεν έχουμε duplicates
2. Η relation είναι ήδη sorted σε κάποιο από τα projection attributes.

- ***Set operations (union, diff, inter):***

Για να εφαρμοστεί κάποιος από αυτούς τους operators, οι δύο relations πρέπει να έχουν ακριβώς τα ίδια attributes. Στην συνέχεια οι 2 relations πρέπει είτε να είναι sorted είτε hashed στα ίδια attributes. Επιλέγεται το χαμηλότερο κόστος βάση των indexes που υπάρχουν σε κάθε relation.

Τα relations είναι sorted όταν:

1. Έχουν B+tree primary index
2. Είναι output άλλου operator που έχει γίνει sorted, π.χ. από ένα projection για duplicate elimination

Τα relations είναι hashed όταν:

1. Έχουν είτε primary είτε secondary index staticHashing ή extentibleHashing

- ***GroupBy:***

Για να εφαρμοσθεί αυτός ο operator πρέπει:

1. Όλα τα attributes της relation να εμφανίζονται ή στο groupby part ή στην aggregate function.
2. Αν υπάρχει having clause, όλα τα attributes που περιέχει στο condition να εμπεριέχονται είτε στο groupby part ή στην aggregate function.

3. Το relation να είναι sorted ή hashed στα groupby attributes. Αν δεν είναι συμπεριλαμβάνεται το σχετικό κόστος. Για το αν είναι ήδη sorted ή hashed ισχύει ότι και στις προηγούμενες περιπτώσεις.

Επιλέγεται το χαμηλότερο κόστος βάση των indexes που μπορεί να υπάρχουν στα groupby attributes ή της διαδικασίας sort/hash.

- **Selection:**

Για να υπολογίζουμε το κόστος του selection αρχικά πρέπει να δούμε εάν έχουμε ένα ή πολλά conditions. Στην πρώτη περίπτωση είναι απλά τα πράγματα και παίρνουμε το μικρότερο κόστος, στην δεύτερη περιπτωση, εάν έχουμε 'and' παίρνουμε το μικρότερο κόστος από όλα τα conditions, και εάν έχουμε 'or' παίρνουμε το μικρότερο κόστος για κάθε condition και τα αθροίζουμε. Για υπολογίζουμε το κόστος του condition θα πρέπει να γνωρίζουμε εάν έχουμε ισότητα ή ανισότητα για να ξέρουμε τι index θα χρησιμοποιησουμε. Αναλυτικότα οι περιπτώσεις (τις έχουμε πάρει από το βιβλιο 'DATABASE SYSTEM CONCEPTS):

1. ισότητα:
    a. linear Search
    b. linear Search Equality on Key
    c. tree Primaru Equality on Key
    d. tree Primary Non Equality on Key
    e. hashing Primary Equality on Key
    f. hashing Primary Non Equality on Key
    g. tree Secondary Equality on Key
    h. tree Secondary Non Equality on Key
    i. hashing Secondary Equality on Key
    j. hashing Secondary Non Equality on Key
2. ανισότητα :
    a. linear Search
    b. linear Search Equality on Key
    c. tree Primary Compare
    d. tree Secondary Compare

Ουσιαστικά η ιδέα είναι ότι για κάθε condition προσπαθούμε να βρούμε εάν κάποιο attribute έχει καποιο index (σε πολλά attributes και σε περίπτωση 'or' προσπαθούμε να βρούμε εάν υπάρχει κάποιο Index(primary ή secondary) που να περιέχει όλα τα attributes των conditions), έτσι ώστε να μπορεσουμε να μικρύνουμε το κόστος.

- **Joins:**

Για να υπολογίζουμε το κόστος του selection αρχικά πρέπει να δούμε εάν έχουμε ένα ή πολλά conditions. Στην πρώτη περίπτωση είναι απλά τα πράγματα και παίρνουμε το μικρότερο κόστος, στην δεύτερη περιπτωση, εάν έχουμε 'and' παίρνουμε το μικρότερο κόστος από όλα τα conditions, και εάν έχουμε 'or' παίρνουμε το μικρότερο κόστος για κάθε condition και τα αθροίζουμε. Οι περιπτώσεις που έχουμε είναι :

- block Nested Join
- merge Join
- hash Join
- indexed Block Nested Join(υπάρχει η συνάρτηση, απλως δεν χρησιμοποιείται)

Ουσιαστικά η ιδέα είναι ότι για κάθε condition προσπαθούμε να βρούμε εάν κάποιο attribute έχει καποιο index (σε πολλά attributes και σε περίπτωση 'or' προσπαθούμε να βρούμε εάν υπάρχει κάποιο Index(primary ή secondary) που να περιέχει όλα τα attributes των conditions), έτσι ώστε να μπορεσουμε να μικρύνουμε το κόστος.

Transformations

Τα παρακάτω transformations έχουν υλοποιηθεί και εφαρμόζονται για την εύρεση του καλύτερου πλάνου εκτέλεση:

1. Eliminate multiple projections :
   proj1(proj2(...(projN(relation))) = proj1(relation)
2. Push projections in set operations :
   proj(union(relation1)(relation2)) = union(proj(relation1))(proj(relation2))
   union could be diff or intersection
3. Push projections in join operations :
   proj(join(relation1)(relation2)) = join(proj(relation1))(proj(relation2))
4. Selection rearrange:
   sel[A1>0](sel[A2>0](relation)) = sel[A2>0](sel[A1>0](relation))
5. Push selection in set operations:
   a. sel[A1>0](union(relation1)(relation2))=
      union(sel[A1>0](relation1))(sel[A1>0](relation2), union could be diff or intersection
   b. sel[A1>0](diff(relation1)(relation2)) = diff(sel[A1>0](relation1))(relation2), diff could be intersection (no union in this case)
6. Push selection in join operations:
   sel(join(relation1)(relation2)) = join(sel(relation1))(sel(relation2))
7. Set rearrange:
   union(union(relation1)(relation2))(relation3)=
   union(union(relation3)(relation2))(relation1) ,union could be diff or intersection
8. Join rearrange:
9. join(join(relation1)(relation2))(relation3) = join(relation1)(join(relation2)(relation3))

Παρατήρηση: Τα tranformations 3,6 και 9 σπάνε σε πολλές περιπτώσεις ανάλογα με τα join attributes,τα projection attributes και τα selection conditions