International Journal of Computer Applications (0975 – 8887)
Volume 70– No.7, May 2013

An Efficient Methodology for the Evaluations of Select Operation in Query Optimization

Syed Naimatullah Hussain
Taif University,KSA

Sultan Aljahdali
Taif University,KSA

Ashfaq Ahmed K
Taif University,KSA

ABSTRACT

The advancement of 4th Generation Languages (4GLs) brought the mutiny in the Information technology by making users free from the burden of developing logic for solution of the application problem. At the same time, it enhanced the need for implementation of ameliorated strategies to evaluate the intermediate operations. The development of 4GLs freed the users from the strained design of 'how to do it' for solutions of their applications. At the back end, it increased the complexity of transformation from non-procedural query to procedural query. The query processing and optimization manages these transformations, by transforming 4GLs query into appropriate relational algebra/calculus expression in the first phase, minimizing the operations in the next phase and then evaluating the expression through the efficient strategies for such operations in the third phase. The transformation to relational algebra/calculus expression and minimizing operations are straightforward processes. The development of ameliorated strategies depends on the file organizations used in the storing relations of the database (Secondary storage structures). This paper discusses the strategies designed for select operation, when tuples of operand relations are stored in one of various file organizations in the database. Here, an attempt is made to design ameliorated algorithms to evaluate select operation when the tuples of the operand relations are stored in any of the file organizations like (i) sequential file (ii) indexed sequential files with B-tree file having number of key field indexes in each node and B+ tree file organizations for multiple indexes where data pointers are stored in each leaf node of the multilevel indexes of the file organization. The literature available suggests that, there are good methods available when key fields are primary keys. The authors have not noticed any clear cut strategies, when key fields for indexing are non primary keys (non candidate keys). Here, an attempt is made to develop the strategies, when key fields are non-candidate keys.

General Terms

Tuples, Algebra/Calculus, Algorithms, Sequential file and Indexed Sequential files

Keywords

4GL 4th Generation Languages, RA regular expression, B Trees and B+ Trees

1. INTRODUCTION

The development of 4th generation languages paved the way for abstracting needful information, through only referencing the needs without bothering about the hassles of how to procure it. The efficiency of the query evaluation depends on minimization of number of operations and memory requirements [2, 3]. The query requirement may spread over wide range of concatenation of these operations to be performed on operand relations in any ordering sequence. The efficiency of such query evaluation can be obtained in two different phases. Initially, first by developing a methodology to sequence the operations to minimize the memory space utilization and number of I/O operations. The query processing can be done in three phases i.e. translation of the 4GL query into relational algebra/calculus expression, transforming it into bare minimum number of operations with minimal memory and then evaluating the expression using an ameliorated strategy.

The figure 1 in the following depicts the flow diagram, describing different phases of the query evaluation [5, 6].

- Conversions of 4GL query into relation algebra (RA) expression.
  The expression:

  SELECT attributes

  FROM tables

  WHERE predicate

can be directly converted into RA expression as

πattributes (σp (r1'x r2 x….x rm)

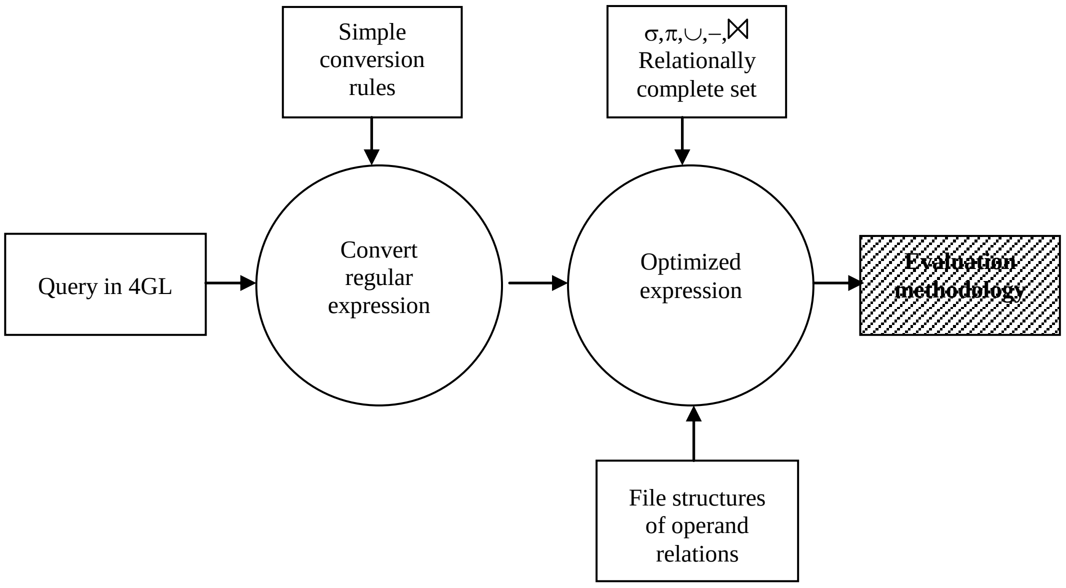
Simple conversion rules
σ,π,∪,–,⋈ Relationally complete set
Query in 4GL
Convert regular expression
Optimized expression
Evaluation methodology
File structures of operand relations

Fig 1: Query evaluation phases.

This can be obtained by performing the following steps:

1. Form cross product corresponding to the relations FROM CLAUSE.
2. Predicate corresponding to the WHERE clause will be the predicate P for SELECT operation.
3. Attributes in SELECT clause correspond to attributes of PROJECT clause.

The first phase is straight forward process, the second phase depends on structure and length of records present in operand relations and the last phase needs the design of ameliorated strategies for each operation along with more ameliorated methodology for each sequencing these strategies.

A query expression may involve number of operations [5, 6] (mathematical as well as relational). Out of these operations, the operations <select, project, join, intersection & difference > make an expression relationally complete. Thus, the major role in the query evaluation is to develop the optimal techniques for the third phase concentrating only on the above said five operations i.e. the major task in query evaluation is to design ameliorated strategies with minimizing memory requirements and I/O operations for these individual relationally complete operations and interleaving them through an efficient methodology for the evaluation.

Our project of developing a natural query language involves the design of ameliorated strategies for the evaluation of relational algebra/calculus operations i.e. unraveling the black box shown in the figure 1. This paper discuses the design of strategies for the evaluation of select operation. Here, we have considered the database comprising of number of data files with same or different file organizations and with constraints specified in the predicate P.

2. SEQUENTIAL FILE

The operand relation of select operation may have been stored in any of the file organizations like sequential, B-tree or B+ tree etc. within the database [3,4]. The operand relation stored in sequential file comprises of n records / tuples (cardinality) stored in increasing / lexicographical order of key field values. The predicate P contains

field name (rel.op.) value

The typical select operation is of the type:

R1: = σ fieldname (rel.op.)value (R)

where rel.op. ∈ { <,<=,=,≠,>,>=}∈

Here, the tuples {records) of operand relation R are selected, base on the satisfaction of the predicate conditions.

An attempt is made to develop an effective strategy through the following algorithm.

Algorithm (seqential.selection)

/* P ←field (rel.op.) value. rel.op. is one of the { <, <= ,=, ≠, >, >=}operators R is operand relation. ri is the ith tuples of R, n is the cardinality of R. 'a' is the field name specified in the predicate P and R1 is the result operand relation.

```
Read P
Begin
while {i<n) do
{Read R.ri
{If rel.op ∈{ <, < = } then
   { if ri.a. {rel.op} value
   R1 ← {R1} U ri ;  i ←i + 1
           Else i ←  n + 1}}
           else { if rel.op {  >, >= } then
                     { if ri.a ( <=, < ) value
                     i←i+1
                      else R1 ←R1 U {ri} ; i ←i + 1}}
                    else {if relop ∈ { = }
  {if ri.a. <  value; i ← i + 1
           else if  ri.a. = value
          R1 ← {R1} U ri
           else i ∈ {n+1}}
           else {{ if ri.a .(not =)   value
          R1 ← {R1} U ri ; i ←i + 1
           else i ←i+1}
end (while)
```

Here, the average access of records from sequential file is of O(n/2). The output operation of selected records remains the same for a specified file type. The optimality is unaffected by this. So it is not accounted in computing the efficiency.

3. B-TREE

The database may contain different types of data files [2, 3]. One of the file types may by indexed sequential file with index stored in B-tree file. The B-tree structure of index may be of the type as explained below. Here, B-tree is balanced tree with all leaf nodes are at the same level. A node of the B-tree of order P contains:

P_1	K_1	Pr_1	P_2	K_2	Pr_2	….	P_i	K_i	Pr_i	….	P_{q-1}	K_{q-1}	Pr_{q-1}	P_q

Fig 2: XXXXXXXXX

- K1 < K2 < ….. < Ki < … Kq-1 < Kq are key field values of the records.
- P1, P2 ….. Pq, q < p are tree pointers i.e. they contain the addresses of sub trees containing index for x : ki < x < Ki+1 i ∈ [1,q]; The sub tree pointed by P1 contains index for key field values X < K1 and similarly for Pq X > Kq.
- Pr1, Pr2, ….. Prq-1 are data pointers i.e. each contain the address of the beginning of the block of records whose last record is with key field value pointed out in the corresponding index.
- Key field value of one record in each block has an entry in the index of B-tree.
- The number of indexes q in each node is restricted by 2 < q < p for root node and ceil (p/2) < q < p for other nodes.

When the operand relation records are stored in B-tree file organization, an attempt is made to design the ameliorated strategy through the following algorithm. The predicate p is of the type key field (rel.op:) value and rel.op may be one of {<, < =, ≠, >, > = } operators. The records of tree sites in the equation order of key field values. ri is the ith record in the operand relation R. The predicate referenced in the key field need not be primary key. n is the cardinality of the relation.

3.1 Algorithm (B-tree selection)

```
/* The order of B-tree is p, q is the number of tree pointers in
the node at the jth level [4, 7].
Pij is the tree pointer to ith index at jth level. Prij is the data
pointer pointing to the block of records
Brij for the ith index value at jth level R1 is the result operand
relation.
i is the position of the node in a level
Brij is the block pointed by Prij*/
i = 1; j = 1
while (i < q and Pij  = null)
{If relope ∈ { <, < = }  and
          Kij (relop) value or i = 1
          {R1 ←  {R1} U Brij; j  ←  j+1
return}
if  Kij = value
   { ru} ← □Brij
    for u = 1 to n
   R1□← { R1} U ru  until ru.a (relop) value }
{if relop ← {>, >=}and Kij < value
        { i = i+1 until Kij> value
         j← j +1; return}
if Kij > = value
              {{ru} ←□Brij
              for u = 1 to n
                u ← u + 1 until  ru.a  > = value
               R1 ← {R1} U ru ; i□i+1}
              j← j + 1 until i=q, Pij  = null
{If relop  ←□{=}
      If {Kij < value i ←i+1 return }
            j  ←j+1
     else { if Kij = value
              {ru} ←Brij
            for u = 1 to n
            if ru .a  =  value
            R1 ← {R1} U{ru} until
             ru.a ≠value}
{IF relop← { ≠ } and
   if Kij<value
        R1 ←{R1} U {Brij};
              i ←i+1 ; j ←j+1  until Kij >  value
            {ru} ←Brij
             for u = 1 to n
              R1 ←{R1} U ru until ru.a. < value
               u ←u + 1 until  (ru.a) .value
               R1 ←{R1} U ru
              R1 ←{R1} U Bij, until i = q and Pij  = null }
```

Here, the number of I/O accesses is of the order O (1/2 log p n).

4. B+ TREE

The operand relation might also be stored in B+ tree multilevel dynamic index and can be implemented [1, 4]. The data pointers for key field values appear only at leaf nodes. The internal node of B+ tree is of the type.

P_1	K_1	P_2	K_2	…	Pi	K_i	…	K_{q-1}	Pq

and the leaf node of B+ tree is of the type

K 1	Pr 1	K_2	Pr_2	..	Ki	Pr_i	..	K_{q-1}	Pr_{q-1}	P_{next}

Fig 3: The nodes of B- tree.

- The order of B+ tree is p the value of q is 2 < q < p for root node, ceil (p/2) < q < p for internal nodes and root node.
- K1 < K2 < ….. < Kq are index field values.P1, P2 ….. Pq, q <= p are tree pointers those exists only in the internal nodes.
- Pr1 Pr2 …. Prq-1 are data pointers existing in leaf nodes those are the addresses of beginning of memory blocks containing the specified field value at the last record of the block.
- P_{next} points to the next leaf node.

4.1 Algorithm (B+ tree select)

```
Read P
While (i<=q; j =j last
{
      if rel.op ∈ { <,<=} then
          for i = 1 to q
               for j = j to jlast –1
                  if  Kij(rel.op.) value
                    j = j + 1
                    R1 ← {R1} U Bij
                     else
                   for i = i+1 to q
                         for j =  j to jlast –1
                           {ru} ← Bij
                            R1 ←{R1} U ru  until ru .a  > value
                            j to jlast +1;  i ← q+1 }
{ if rel.op ∈ { = } then
for i = 1 to q
         for j = j to jlast –1
         if Kij  <= value
         i=i+1; j=j+1 until j= jlast –1
       else {ru}← Bij
           for u = i to  n
               if ru .a < value
                u=u+1
               else if ru.a=value
               R1 ← {R1} U {ru};
                   Pj← Pnext   until ru.a≠value
                                 u←n+1}
{ if rel.op. ∈ {>} then
     for i = 1 to q
          for j = 1to jlast-1
            if Kij < value
i = i + 1; j = j + 1
        else
          j←jlast
         {ru}← Bij
           for u= 1= to n
           u=u+1 until ru.a>value
             R1 = { R1 }U ru
             u=u+1 until u=n
                  Pj←Pnext until Pnext=null}
{
if rel.op. ∈ {≠}then
for j =1to q
for j =1 to jlast-1
if Kij <>  value
i = i + 1  j = j+1 until j =jlast
R1←R1U { Bij}
 Else
     j←jlast
     {ru} ← Bij
      if ru.a  ≠  value
  R1← R1 U { ru }
  u ← u + 1 until u = n
Pj ← Pnext until Pnext= null
```

Here, the number of I/O accesses is of the order O (1/2 log p n).

5. ACKNOWLEDGEMENT

I sincerely acknowledge to Dr Shivanand M Handigund for his constant support and encouragement for successful completion of this paper.

6. CONCLUSION

The strategies developed through the design of the three algorithms are ameliorated strategies as these algorithms minimized the input operations with

- accessing only the first block which contains mixed records, both satisfying and not satisfying conditions.
- avoiding the access of blocks which contains all the records not satisfying the conditions

Thus, we have made attempts to develop efficient strategies. This project also brought clues for the design of new efficient data structures which is our current study

7. REFERENCES

[1] A paper entitled "Skew Handling Techniques in sort merge join." By Weifi-et- al, Oracle Corporation, ACM SIGMOD, June 2002

[2] A Guide to the SQL Standard third edition C J Date with Hugh Darwen Fundamentals of Database Systems by Ramez Elmansri Shamkanth Navathe the Benjamim/Cummings Publishing Company Inc. Redword CA(USA)

[3] Ramakrishna and Gehrke "Database Management System" third edition, Me-Graw Hill, 2003

[4] Goetz Graefe et al "query evaluation techniques for large databases" ACM computing surveys, 24(1), 63-113, march 1992.

[5] "Query Processing and Optimization "module 12 of Database Management Systems Impact learning material series prepared by Indian Institute of Technology, Bombay.

[6] "SQL2 and Application Programming "Lecture series by S. Seshadri Computer Science and Engineering IIT Bombay.