Industrial Engineering Journal
ISSN: 0970-2555
Volume : 52, Issue 3, March : 2023

QUERY OPTIMIZATION PLAN ALGORITHM FOR SUB-QUERIES

Amiya Kumar Sahoo, Prakash Dehury, Harapriya Rout, Manas Kumar Mallick, Chitta Ranjan Sahoo, Department of Computer Science & Engineering, Aryan Institute of Engg. & Technology Bhubaneswar, India
amiya79@rediffmail.com, dehuryprakash1982@gmail.com, harapriya172rout@gmail.com, manasnarayan@gmail.com, ranjan.chitta@gmail.com

Abstract

The SQL language allows users to express the queries that have nested sub-queries in them. Optimization of nested queries has received considerable attention over the last few years. Most of the previous optimization work has assumed that at most one block is nested within any given block. The two main contributions of this report are:

1. Optimization strategies for queries that have an arbitrary number of blocks nested within any given block, and
2. a new algorithm for the execution of nested queries, involving one or more other joins, in a multi- processor environment. The new algorithm cuts down the processing costs over conventional algorithms.

Keywords: subquery, join, SQL, query processing

I. INTRODUCTION

Traditionally, database systems have executed nested SQL queries using Tuple Iteration Semantics (TIS). It was analytically shown in that executing queries by TIS can be very inefficient. It was first pointed out and then in that nested queries can be evaluated very efficiently using relational algebra or set-oriented operators. The process of obtaining set-oriented operators to evaluate nested queries is known as unnesting. It was later pointed out in and that the unnesting techniques do not always yield the correct results for nested queries that have non equi-joincorrelation predicates or for queries that have the COUNT aggregate between nested blocks. These queries have correlation join predicates and an aggregate (AVG, SUM, MIN, MAX, or COUNT) between the nested blocks. The reason for focusing on JA type queries is that many other nesting predicates (such as EXISTS, NOT EXISTS, ALL, ANY) can be reduced to JAtype queries. In this paper we focus our attention uninvesting Join Aggregate type queries (JA).These queries have correlation join predicates and an aggregate function between the nested blocks.

II. PROCESSING A GENERAL NESTED QUERY

The recursive version of algorithm NEST-G is described in procedure nest_g (query-block), where the parameter query block is a pointer to a SQL query block, possibly with descendant inner query blocks nested with in it. The procedure is usually called with a pointer to the outermost query block of the query.
Procedure nest_g(query_block)
{
inner queryblock) for each predicate in the WHERE clause of query_blockif predicate is a nested predicate(i.e contains
nest_g(inner_query_block)
*Determine type of nesting, and call appropriate

```
*Transformation Procedure
if SELECT clause of inner_query_block containsaggregate functionif inner_query_block contains joinpredicate referencing a relation which isnot in it's FROM clause
Nesting is type_JA
*/
nest_JA2(inner_query_block)
nest_N_J(query_block,
inner_query_block)
else
*nesting is type_A
nest_A(inner_query_block)
else
nest_N_J(query_block, inner_query_block)
return
Procedure nest_g(query_block)
{
for each predicate in the WHERE clause of query_blockif predicate is a nested predicate(i.e contains inner query block)
nest_g(inner_query_block)
*Determine type of nesting, and call appropriate
*Transformation Procedure
*/
if SELECT clause of inner_query_block containsaggregate functionif inner_query_block contains joinpredicate referencing a relation which isnot in it's FROM clause
/*
Nesting is type_JA
*/
nest_JA2(inner_query_block)
nest_N_J(query_block,
inner_query_block)
else
/*
*nesting is type_A
*/
nest_A(inner_query_block)
else
nest_N_J(query_block, inner_query_block)
return
}
```

and the innermost query blocks are the leaves. Procedurenest_g () searches down through the levels of a nestedquery from the outermost query block until it finds the innermost query blocks. It then examines the leaf block to determine the type ofnesting present, and transforms the parent to canonical form by calling the appropriate transformation procedures. After this is done for all nested predicates in query_block,the recursion then nwinds one level and the query blockimmediately above is processed in the same way, continuing the unwinding until lastly the outermost, orroot, query block is transformed. The algorithm represented in procedure nest_g () solves the problem of correctly transforming a type-JA query with multiple levels of nesting. To demonstrate this, let us assume the following query tree (fig. 1).

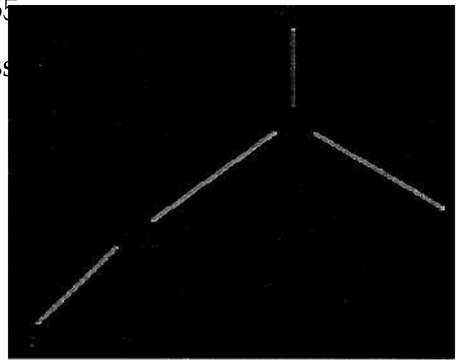

Fig. 1 Query Tree

The edges of the tree are labelled with the kind of nesting present at that level. Query block B contains anaggregate function in its SELECT clause, and both C andE contains JOIN predicates referencing tables in query blocks at a higher level. So far, the most important featurewith regard to processing the query has not been mentioned does C or E contain a reference to a table in theFROM clause of A? This is important because it indicates whether there is a type-JA nesting present in the query. Ifone of the inner blocks, including B, contains a reference to a table in A, then type-JA nesting is present. In otherwords, a Join predicates reference must span a query block containing an aggregate function for type-JA nesting to be present. For example, assume the example query tree contains a reference in B, C, or E to a table in the FROM clause of A.

Let us assume that E contains this reference, in a Join predicate Procedure nest_g () will travel down to E,unwind and apply algorithm NEST-N-J, combing C and E. This moves the reference to the table in A to block C.Then blocks C and B are combined, and then blocks D and B. Now query block B has inherited the Join predicate inblock E, so that it contains both an aggregate function and a JOIN predicate which references a table not found in theFROM clause of B. Thus, procedure nest_JA2() is called, which creates a temporary table with a GROUP BY clauseas specified in algorithm NEST-JA2, and removes the aggregate function, replacing it with a reference to thecolumn in the temporary table which results from the application of the aggregate function. This reduces thetype-JA nesting to type-J nesting, and procedure nesr_N_J() is immediately called to finish the Job of reducing thequery to canonical form .Thus type-JA nesting of deeperthan one level can be detected by examining a single queryblock, which has inherited the 'trans-aggregate' JOIN predicate by the recursive transformation of inner queryblocks, and the type-JA nested query can be transformed to canonical form by applying the single-level algorithmNEST-JA2.

From this example it can be seen that the advantage of the recursive algorithm presented in procedure nest_g() is simplicity the information needed to transform a queryblock containing a nested predicate is confined to two levels of the query the outer level and the inner.

III. MODIFYING KIM'S ALGORITHM

In this section we describe how kim's algorithm maybe modified to avoid the COUNT bug. The motivationin trying to modify Kim's approaches that it may be moreefficient than Ganski's solution.

A. Queries with two blocks

We return to Example query that created the temporary relation TEMP, remains unchanged. However, query hasto be modified. We know that the COUNT associated with a tuple of R that does not join with any tuple of S is 0. Thus, a tuple of r belongs to R that does not join with any tuple of TEMPI will be a result tuple if (r.b OP1 0) is true. For a tuple r belongs to R that joins with a tuple ofTemp1, r will be a result tuple if (r.b OPITEMP1.count) is true.

Notationally, we write this as shown below:
Example :
SELECT R.a FROM R, TEMPI
WHERE R.C = TEMPI.c --- OJ
[R.bOPITEMP1.count :R.b OP1 0]

Industrial Engineering Journal
ISSN: 0970-2555
Volume : 52, Issue 3, March : 2023

The square brackets, in the last line of the above query, enclose the two predicates which are separated by a colon. The first predicate is applied to the joining tuples while the second tuple is applied to the anti-join tuples. There is currently no way of expressing the above query in SQL.
We now show that under certain circumstances, the modified Kim's method may be more efficient than Ganski's method. The heuristic argument is based on

1) The number of tuples that flow from each node in the query plans corresponding to the two methods and
2) The number of tuples that have to be processed at each group-by and outer join node.

The query plans for the two methods are shown in the figure. The edges in Figures are labelled by the number of tuples flowing through those edges. Both methods involve accessing relations R and S. Clearly |TEMP|<=1S 1and IR 1<= IROJS I. Assume that ISI<IRI. The number of tuples flowing from the group-by node to the outer join node in Kim's method is equal to I TEMPII. The number of tuples flowing from the outer join node to the group by nodein Ganski's method is equal to IR OJ S I. Clearly, I TEMP1 I<IR OJ S I. The number of tuples processed by the group-by node and the outer join nodein Kim's method is each less than the corresponding number of tuples in Ganski's method. Kim's method should perform better than Ganski's method.
In the above discussion we have ignored the fact that Ganski's method joins two base relations, whereas in Kim’s method, we join a base relation with a temporary relation. As a result, Ganski's method might be able toemploy more join methods. Clearly, the optimizer has topick the cheaper method more carefully than as outlined above. The important point is that we can use Kim's method even in the presence of the COUNT aggregate when the correlation predicates are all equi-joins.

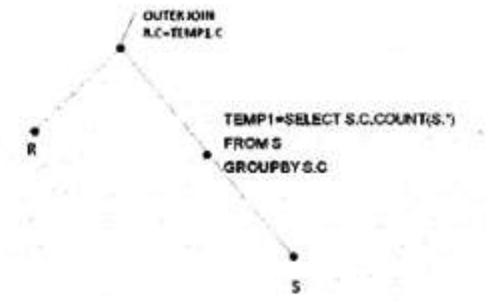

Fig. 2 Modified Kim's Method

B. Queries with three blocks

We now extend the modified Kim's algorithm toqueries with three blocks. An equi-join correlation predicate is called a neighbour predicate if it references the relation in its own block and the relation from the immediately enclosing block.
Consider the following example in which all the joinpredicates are neighbor predicates.
Example:
SELECT R.a FROM R WHERE R.b OP1 (SELECT
COUNT (S.*) FROM S WHERE R.c = S.C AND S.d OP2
(SELECT COUNT (T.*) FROM T WHERE S.e= T.e))
The algorithm given by Kim worked bottom up. We follow the same approach here. The result of the query is obtained by evaluating the following three unnested query
Query:
TEMP1 (e, count) = SELECT T.e, COUNT (T.*)
FROM T GROUP BY T.e;
TEMP2 (c, count) = SELECT S.c, COUNT (S.*)
FROM S, TEMP1 WHERE S.e= TEMP1.e --- OJ

Industrial Engineering Journal
ISSN: 0970-2555
Volume : 52, Issue 3, March : 2023
[S.d OP2 TEMP1.count: S.d OP2 0]
GROUPBY S.c, TEMP1.f;
SELECT R.a FROM R, TEMP2
WHERE R.C= TEMP2.c --- OJ
[R.b OPI TEMP2.count: R.b OP1 0]
Thus, we were able to extend the same principle to athree block query of Example and avoid the COUNTbug. It is easy to see how we can extend the abovesolution to a query with more than three blocks as long asthe correlation predicates are neighbor predicates. Thenatural question then is: what happens when we have non-neighbor predicates.
C. Queries with non neighbor predicates
We start with the query shown in Example. Thisquery is obtained by adding the non neighbor predicate, R.f= T.f, in the third block of the query inExample. Surprisingly, the query becomes very hard to
un-nest in the presence of the COUNT aggregates.
Example:
SELECT R.a FROM R WHERE R.b OPI (SELECT COUNT (S.*) FROM S WHERE R.c = S.C AND S.d OP2
(SELECT COUNT (T.*) FROM T WHERE S.e=T.e
AND R.f=T.f));
Evaluating bottom up, we would expect the three unnestedqueries to be as follows:
Query:
TEMPI (e, f, count) = SELECT T.e, T.f, COUNT (T.*)
FROM T GROUPBY T.e, T.f;
TEMP2(c, f, count) = SELECT S.C, TEMP1.f,
COUNT (S.*) FROM S, TEMP1 WHERE S.e= TEMP1.e --- OJ
[S.d OP2TEMP1.count: S.d OP2, 0]
GROUPBY S.c, TEMP1.f;
SELECT R.a FROM R, TEMP2
WHERE (R.c = TEMP2.CAND R.f= TEMP2.f) - OJ
[R.b OP1 TEMP2.count: R.b OP1 0]

There are no surprises in Queries. Here, each tuple of R joins with at most one tuple of TEMP2. The middlequery is incorrect. Notice that we are selecting attributesfrom both S and TEMP1. We are also grouping byattributes from both the relations. In case an S tuple does not join with any TEMP1 tuples, we cannot meaningfully evaluate the query. Let us try to understand what happens when an S tuple does not join with any tuple of TEMP1. It is clear from the query that if an S tuple does not join with any T tuple, then COUNT (T.*) is 0, irrespective of the value of R.f. Therefore, such an S tuple will contribute to COUNT (S.*) if (S.d OP2 0) is true. There is another subtlety that we need to focus on. Assume that a tuple s belongs to S joins with one or moreTEMP1 tuples. Let (TEMP1.f) denote the set of f values in the joining TEMP1 tuples. We need to decide if s will contribute to COUNT (S.*). If a tuple r belongs to R has as an f value that is in (TEMP1.f), we know that COUNT (T.*) associated with this (r, s) pair will be greater than 0. Then s will contribute to COUNT (S.*) if
(s.d OP2 TEMP1.count) is true. On the other hand, for any tuple r belongs to R that has an f value that is notin (TEMP1.f), the corresponding COUNT (T.*) will be 0. If (s.d OP2 0) is true, then s will contribute to COUNT (s.*). Using these observations, we now describe what the outer join operator of query must accomplish using the following pseudo-code.
1. if no tuple of TEMP1 satisfies (s.e = TEMP1.e)
2. then output (s.c, all)
3. else for each tuple of TEMP1 satisfying (s.e=
TEMP1.e)

4. {
5. if (s.d OP, TEMP1.count)
6. then output (s.c, TEMP1.f)
7. else if (s.d OP2 0)
8. then output (s.c,~ {TEMP1.f})
9. }

IV. DATAFLOW DIAGRAM OF QUERY

Let us explain the dataflow diagram (Fig. 3) and the algorithm with the help of the following query:
SELECT R1.a FROM R1 WHERE F1 (R1) AND R1,b
OPI (SELECT COUNT (R2.*) FROM R2 WHERE F2
(R2) AND F2 (R2, R1) AND R2.c OP2 (SELECT
COUNT (R3.*) FROM R3 WHERE F3 (R3) AND F3 (R3, R2)
AND F3 (R3, R1)));

V. ALGORITHM FOR QUERY EXECUTION

```
Unoptimized_algorithm()
{
i=2, k=1, n, j;
While (n>2)
{
Repeat for i=2 to n
{
Repeat for k=1 to i-1
While (k>=2)
Temptable,table;
temptempi.
Repeat for j=2 to k-1
temp,temptemp
z=n;
if (grouping is needed)
{
While (z>1)
{
If (having condition)
Then group-by temp, with attributesof z-1 tables.
Z--;
Return result tuples;
}
```

VI. OPTIMIZED ALGORITHM FOR QUERY EXECUTION

We can reduce the query execution time and thenumber of calculations by using the following algorithm:

```
Optimized_algorithm()
i=2, k=1, n;
if (n>2)
{
Repeat for i=2 to n
Repeat for k=1 to i-1
```

Industrial Engineering Journal
ISSN: 0970-2555
Volume : 52, Issue 3, March : 2023

←
X
temp1table¡table¡-k.
While (k>=2)
{
<←temp, temp1
←
}
Repeat for j=2 to k-1
attrselection(i, tempp).
tempptempptemp¡\
ResultResult U tempp;
}
}
{
attrselection (n, tempp)
Repeat for q=n to 2
{
← Tempselect attributes from temppwithoutconsidering the attributes of the table q. If neededthen perform the group-by operation with thetable 1, table2...tableq-2, tableq-1.
←temp,temp.
q=2? returntempp: return attrselection(q, tempp);
q--;
}
{

VII. CONCLUSION

This paper contains the un-nesting of the nested correlated sub-queries and it also explains the data flow diagram of the un-nested query. Finally it contains an optimized algorithm over the dataflow diagram which involves the query plan for the efficient query processing.

REFERENCES

[1] U. Dayal, "Of Nests and Trees: A Unified approach to Processing Queries That Contain Nested Subqueries, Aggregates, and Quantifiers," Proc. VLDB Conf., pp. 197-202, September 1987.
[2] Richard A. Ganski and Harry K. T. Long, "Optimization of nested SQL Queries Revisited", Proc. SIGMOD Conf., pp. 23-33, May 1987.
[3] W. Kiessling, "SQL-like and Quel-like correlation queries with aggregates revisited", UCB-ERL Memorandum No. 84/75, Univ. of California at Berkeley, Sept. 1984.
[4] W. Kim, "On Optimizing an SQL-like Nested Query", Trans. On Database Systems, Vol 9, No. 3, 1982.
[5] M. Muralikrishna, "Optimization and Dataflow Algorithms for Nested Tree Queries", Proc. VLDB Conf., pp.77-85, August 1989.
[6]M.Muralikrishna, "Improved unnesting algorithms for join aggregate sql queries," Proceedings of the 18th International Conference on Very Large Data Bases, pp. 91- 102, 1992.