Volume 2, Issue 7 July 2016 ISSN NO: 2454 - 7514

North Asian International Research Journal Consortium

North Asian International Research Journal

Of

Science, Engineering and Information Technology

IRJIF IMPACT FACTOR: 3.821

NAIRJC JOURNAL PUBLICATION

North Asian International Research Journal Consortium

Welcome to NAIRJC

ISSN NO: 2454 -7514

North Asian International Research Journal of Science, Engineering & Information Technology is a research journal, published monthly in English, Hindi. All research papers submitted to the journal will be double-blind peer reviewed referred by members of the editorial board. Readers will include investigator in Universities, Research Institutes Government and Industry with research interest in the general subjects

Editorial Board

Address: -North Asian International Research Journal Consortium (NAIRJC) 221 Gangoo, Pulwama, Jammu and Kashmir, India - 192301, Cell: 09086405302, 09906662570, Ph. No: 01933-212815, Email: nairjc5@gmail.com, nairjc@nairjc.com, info@nairjc.com Website: www.nairjc.com

IRJIF IMPACT FACTOR: 3.821

OPTIMIZATION OF UNNESTING METHODOLOGIES FOR SQL NESTED QUERIES

MOHIT CHHABRA*

*CBS Group of Institutions

ABSTRACT

The SQL language allows users to express queries that have nested sub-queries in them. Optimization of nested queries has received considerable attention over the last few years. The first algorithm for unnesting nested queries was Kim's algorithm, but this technique had a COUNT bug for JA type queries. Later few researchers gave more general strategies to avoid the COUNT bug. Finally to all this M. Muralikrishna modified Kim's algorithm so that it avoids the COUNT bug. The modified algorithm may be used when it is more efficient than the general strategy. In addition, he presented a couple of enhancements that pre-compute aggregates and evaluate joins and outer joins in a top down order. These enhancements eliminated Cartesian products when certain correlation predicates are absent and enabled us to employ Kim's method for more blocks. Apart from this he proposed the Integrated algorithm for generating query plans for a given input query.

In this paper, we have given a new solution for implementing the Kim's modi-fied algorithm of unnesting nested queries and this also avoids the COUNT bug convincingly. Integrated algorithm generates flaws query plans, which has been modified in this thesis. We have also shown experimental results proving one query plan among the all other as computationally better one. These computa-tions are in terms of elapsed time. We have carried out experiments for different data sets of varying sizes from 100 to 1000 tuples in each relation. These re-sults are taken as average of some possible iterative execution of each query plan. Finally, we incorporate the above improved merits into a new unnesting algorithm.

I. INTRODUCTION TO NESTED QUERIES

SQL is a block-structured query language for data retrieval and manipulation developed at the IBM Research Laboratory in San Jose, California. SQL was incorporated into System R, the relational data base management

system, also developed at the IBM San Jose Research Laboratory. One of the most powerful features of SQL is the nesting of query blocks. 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 in 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 presented in do not always yield the correct results for nested queries that have non equi-join correlation predicates or for queries that have the COUNT aggregate between nested blocks. Unnesting solutions for these types of queries were pro-vided in . These solutions were further refined and extended in . An important contribution of the current thesis is a successful implementation for Kim’s modified algorithm that avoids the COUNT bug. Under certain conditions, Kim’s approach may be more efficient than the general solution and hence worth considering.

In this paper, we focus our attention on unnesting Join-Aggregate (JA) type of SQL queries. 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 JA type queries.

An example of a 2 block JA type query is:
SELECT
FROM
WHERE
AND
R1.a
R1
F1(R1)
R1.b OP1 (SELECT COUNT (R2.*)
FROM R2
WHERE F2(R2) AND F2(R2,R1))
F1(R1) and F2(R2) are selection predicates on R1 and R2 respectively, while
F2(R2, R1) is a correlation join predicate between R1 and R2.
A run time system that would execute the above query using TIS would pro-ceed as follows: A tuple r1 from R1 would be fetched. If F1(R1) is false for r1, tuple r1 will not be present in the result. Assuming F1(R1) is true, the

values of the relevant attributes of r1 would be substituted into predicates at deeper levels (F2(R2, R1)). The two block query now becomes a single block query:

SELECT COUNT (R2.*)

FROM R2

WHERE F2'(R2)

F2'(R2) is a predicate on R2 and is equivalent to F2(R2) AND F2(R2, R1) after values of r1's attributes have been substituted in F2(R2, R1).

Let the COUNT value returned by this block be C ($C \leq 0$). C represents the number of tuples of R2 that satisfy F2'(R2). If (r1.b OP1 C) is true, rl will be in the result. Notice that each tuple of R1 can occur in the result at most once. Us- ing TIS, the system executes a query on R2 (the inner relation) for every tuple of R1 (the outer relation) leading to a very inefficient execution strategy. Blocks in the above nested query may be nested within each other to any arbitrary depth.

There are two types of nested queries and they are defined as follows:

Nested Linear Query: is a JA type query in which at most one block is nested within any block.

Nested Tree Query: is a JA type query in which there is at least one block which has two or more blocks nested within it at the same level.

In this thesis, we focus our attention on linear queries only.The techniques for unnesting tree queries presented in were not as general as the ones we are 3developing in the current thesis. For example, did not consider Kim's algo- rithm at all. For ease of notation, we shall assume that there is only one relation in the FROM clause of each block. The algorithms presented in this thesis can be easily extended to the case when there are multiple relations in any FROM clause.

The reader is advised that we shall not adhere to strict SQL syntax when writing queries in this thesis. The SQL syntax for expressing outer joins is fairly cumbersome. Instead, we shall write queries in a syntax that is fairly intuitive.

II. RELATED WORK

A. Kim's algorithm and the COUNT bug:

The first algorithm for unnesting nested queries was Kim's algorithm but this technique had a COUNT bug. Consider the following example of a 2 block JA type query:

SELECT R.a
FROM R, S
WHERE R.b OP1 (SELECT COUNT (S.*)
FROM S
WHERE R.c = S.c)
Kim's algorithm transforms the above query into the following two unnested queries.
Query 1:
TEMP1(c, count) = SELECT S.c, COUNT (S.*)
FROM S GROUP BY S.c

Query 2: SELECT R.a
FROM R, TEMP1
WHERE R.c = TEMP1.c AND R.b OP1 TEMP1.count

The database engine computes the result from the first query and feeds it as an input to the next query. Query 1 calculates the COUNT for each distinct value in the c attribute of S. Each tuple of the relation R only joins with at most one tuple of TEMP1. We assessed that the aggregate functions can be incorporated in the query. Kim's algorithm works correctly if the function is not COUNT, if it is so, then the COUNT bug is seen. The result will only include the tuples from R which join with the tuples of S, so a tuple r from R to be in the result where the COUNT for r is 0 and having (r.b OP1 0) as true will not be there. To avoid this, we can do an outer join, to include these tuples from R, which makes the query as:

Query 3: SELECT R.a
FROM R, S

WHERE R.c = S.c - - - OJ GROUP BY R. #

HAVING R.b OP1 COUNT (S.*)

Using the outer join, we get all the tuples from R, avoiding the COUNT bug. Kim's method can be used if somehow we can compute that R.b is never 0. The outer join is evaluated before the group operation in Query 3. The solution by Ganski proves to be more gereric than Kim's, it is also applicable while using an eui-join.

B. The modified algorithm:

Kim's algorithm was modified by Muralikrishna, which now averts the COUNT bug. Take a look at the nested query from above. The modified algorithm can be applied to demonstrate it's application on the 2 block nested queries. Temporary relation created by the Query 1 stays the same and the Query 2 gets changed. It is known that a tuples from R that don't join with S have a COUNT 0. So, any tuple r from R that joins with TEMP1 will be in result only if (r.b OP1 0) holds true, evidently any such tuple r will be in result if (r.b OP1 TEMP1.count) holds true. We replace the join from Query 2 to an outer join like so:

Query 5: SELECT R.a
FROM R, TEMP1
WHERE R.c = TEMP1.c - - - - - - OJ [R.b OP1 TEMP1.count: R.b OP1 0]
The expression from the first pair of brackets is applied to the join tuples and the second to anti join ones. The Integrated algorithm can express this query in a new way, as in Section III.

Here's the implementation of the modified algorithm to avoid the count bug, we can re-write Query 5 in SQL as:

SELECT R.a
FROM R LEFT OUTER JOIN TEMP1
ON R.c = TEMP1.c WHERE R.b = TEMP1.count
OR TEMP1.count IS NULL)

III. OBJECTIVES

Kim’s modified algorithm is an extension to original Kim’s algorithm of unnesting the nested queries. This algorithm avoids the COUNT bug successfully in all cases of nested queries. And also this shows the better performance as compared to the later advancements so far after Kim’s original unnesting algorithm. In terms of

the execution time, Kim's modified algorithm is the computationally better unnesting algorithm over the techniques.

To implement the routing methods we used the temporary tables for the propagation of intermediate data to the upper layers query execution plan. The data set of tables R, S, T and U is used to implement the unnesting query plans for the given 4 block nested query. Here also we used temporary tables for the propagation of intermediate data to the upper layers in the implementation of query plans. We have carried out these experiments for different data sets of varying sizes from 100 to 1000 tuples in each relation. These results are taken as average of some possible iterative execution of each query plan.

IV. AN INTEGRATED ALGORITHM

The Integrated algorithm given by M. Muralikrishna can generate execution plans for the unnesting of nested SQL queries. It proceeds by generating the execution plan by combining multiple ways to unnest such queries. We propose the very least expensive query plan from all. The algorithm takes the graph of the query, G and outputs a set of query plans. Here's the pseudocode for the algorithm.

```
unnest(G)
{
    if (the BMA can be precomputed)
    {
        compute the aggregate. unnest(G');
    }
    if (Kims method can be applied to the remaining blocks)
    {
        apply Kims method return;
    }
    if (J--J-- ....---J---OJ---...) is encountered
    {
        for (i = 1; i <= m; i++)
```

```
        evaluate first i joins using the cheapest join order. unnest (G');
    }
    if (OJ--J---J--.....---J---OJ---...) is encountered
    {
for (i = 1; i <= m; i++)
        evaluate first I joins using the cheapest join order. unnest (G');
        evaluate the first OJ; replace the first J by OJ. unnest (G');
    }
}
```

We illustrate the working of the above algorithm on the following query.

```
Query 6: SELECT R.a
FROM R WHERE R.b OP1
            (SELECT COUNT (S.*)
            FROM S
            WHERE R.c = S.c AND S.d OP2 (SELECT AVG (T.e)
                        FROM T
                        WHERE S.e = T.e AND R.f = T.f AND T.g OP3
                            (SELECT SUM (U.g)
                                    FROM U WHERE S.h = U.h AND T.i = U.i)))
```

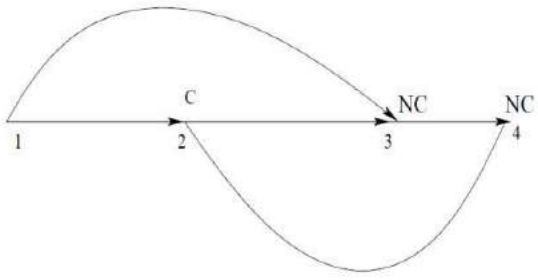


Figure 1. Graphical Representation of Query 6

The J/OJ expression is R---OJ---S---J---T---J---U.

The following query plans, as shown in Figures 2-7 are possible: for Query 6:

(a) Apply Kim's method to blocks 2, 3, and 4.

(b) Join R and S and apply Kim's method to blocks 3, and 4. Since the outer join between R and S is performed before the join, the first join is now evaluated as an outer join.

(c) Join R, S, and T and apply Kim's method to block 4. Both joins are then replaced by outer joins.

(d) Replaced all joins by outer joins, followed by three group by operations.

(e) Join relations S, T, and U first, followed by the outer join. This amounts to applying the general solution for the entire query.

(f) Join relations S, T, and U first. Since the BMA depends only on relations S and T, the BMA is computed before the outer join with R.

(g) About the entire outer join nodes in fig 2 have 2 outgoing edges. The vertical ones are for the anti-join rows, whereas the horizontal ones for the join tuples. Also, the GROUP BY HAVING nodes have 2 output edges, the vertical for the groups not fulfilling the having clause, which have a few portions nulled out. For instance, in fig 5, all the groups that lie between the group by having nodes on the vertical edge are in the form (R,NULL). In Fig 3 we can see edges routing tuples to nodes much higher than their immidiate parent in the tree. This is optional but saves a huge deal in communication costs, as we show in VI.

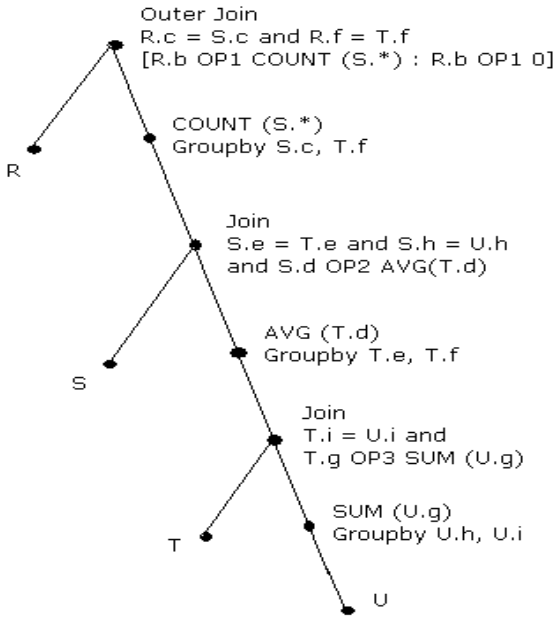


Figure 2. Query plan (a)

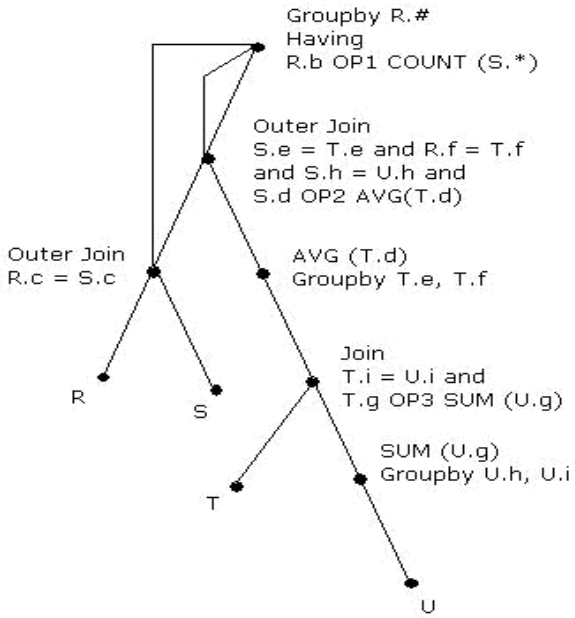


Figure 3. Query plan (b)

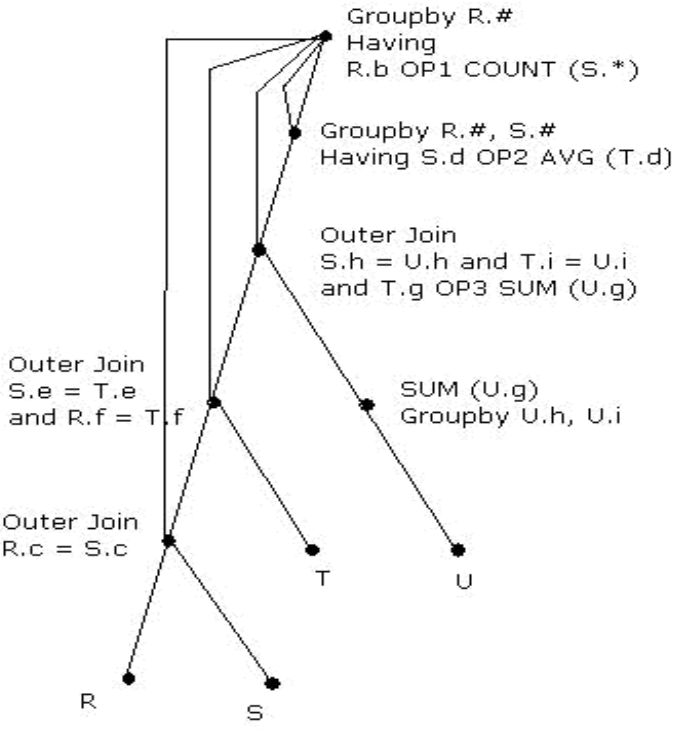


Figure 4. Query Plan (c)

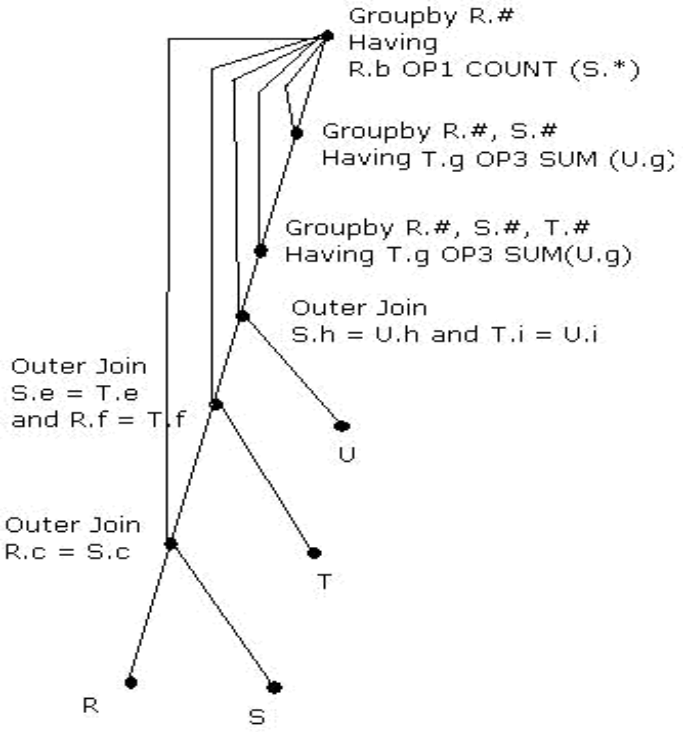


Figure 5. Query Plan (d)

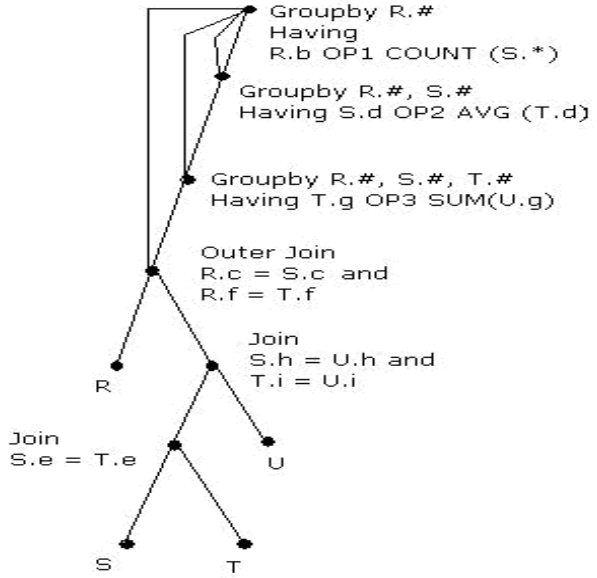


Figure 6. Query Plan(e)

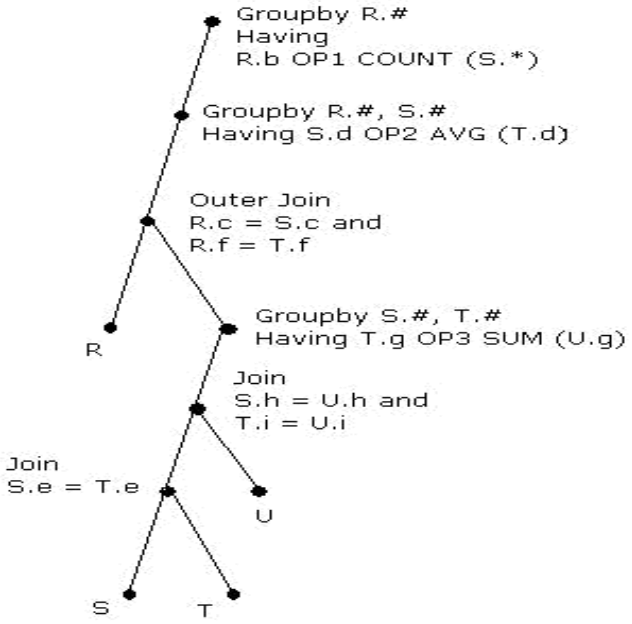


Figure 7. Query Plan (f)

V. FLAWS IN INTEGRATED ALGORITHM

The above integrated algorithm contains some flaws in Query plans (b), (c) and (d). They are described as follows:

- Query (b) has a flaw in it, when some specific join tuple from the outer join operation (R OJ S) won't have any match in an OJ operation with the immediate higher layer, it still has to traverse through the path of the anti join layers to reach the top layer. Here, if R.b is 0, then this anti join tuple will certainly be in the result. But it can be avoided with query plan (b). This tuple is referenced by the primary key of S, S.srn for instance to the top layer and as S.srn will never be null, so the COUNT(S.srn) will never be zero. So, the expression (R.b OP1 COUNT(S.srn)) evaluates false.

- A comparative issue exists in Query arrangement (c) and (d) moreover. The external join operation between the join tuples of external join (R OJ S) and table T proliferates the essential key estimation of table S to the top generally layer. There these hostile to join tuples are sure to show up in the last result of the question if R.b is 0. Be that as it may, it doesn't happen that way on the off chance that we execute our question arranges (c) and (d). In these query plans, this hostile to join tuple conveys the essential key estimation of table S (Ex: S.srn here) to the top generally layer. Since S.srn is not invalid, the estimation of COUNT (S.srn) is non zero. Along these lines the predicate (R.b OP1 COUNT (S.srn)) turns out to be false.

- This is a typical issue for a wide range of question arrangements which join R and S tables pair-wise. The general issue lies when joining two tables pair-wise which has a COUNT total capacity between the two. In the above illustration, the issue lies in the query plans when we join tables R and S pair-wise as there is a COUNT total capacity between them. The arrangement is clarified in Section 6.

VI. SOLUTIONS TO THE FLAWS IN INTEGRATED ALGORITHM

We propose the accompanying answer for the blemishes we have found in the incorporated calculation. The general arrangement is not to join the two tables when there exists a COUNT total capacity amongst them, and to join these two tables with another table together in an unnesting inquiry arrangement.

1. Join the tables R, S and the resultant table of the right wing together.

2. For the join tuples, there are no changes. For the outer join tuples, propagate the primary key value of 3rd table to the top most layer as it is the primary key value of the 2nd table.

For Query plan(b): Join(Outer join) the tables R, S and the resultant table of the conservative together. Pass the essential key estimation of the third table to the top generally layer. We do it in light of the fact that the essential key estimation of the third table is the essential key estimation of table S. For Query arranges (c) and (d): Join (Outer join) tables R, S and T together. Proliferate the essential key estimation of table T to beat generally layer.

Case: For this situation, S.srn would be invalid for all against join tuples and afterward (R.b OP1 COUNT(S.srn)) turns out to be valid if R.b is zero. At that point this tuple shows up in the last result of the inquiry as the predicate (R.b OP1 COUNT(S.srn)) turns out to be valid. In this manner, it has been demonstrated that the arrangement works fine for all Query arranges. Utilizing the enhanced calculation, Query arranges (b), (c) and (d) are changed to as appeared in Figures 8.

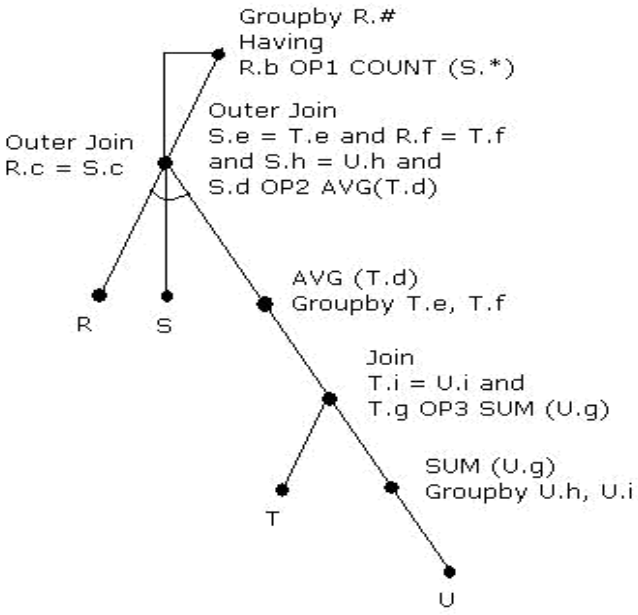


Figure 8. Modified Query Plan (b)

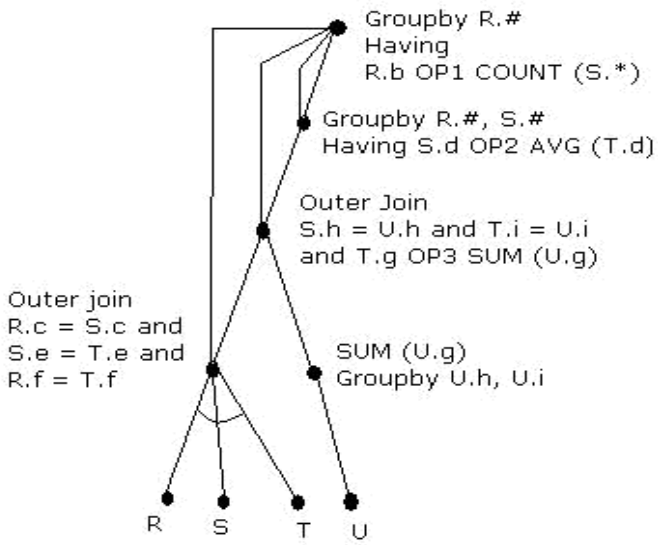


Figure 9. Modified Query Plan (c)

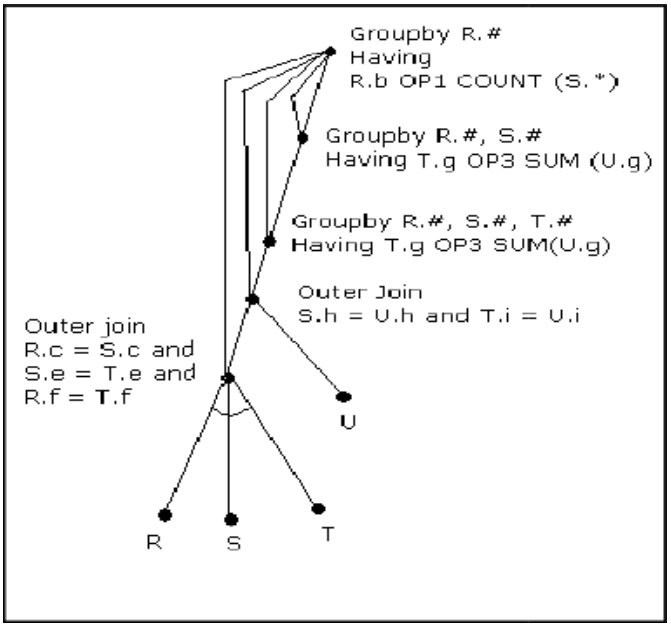
VII. EXPERIMENTAL RESU LTS

A. Implementation

To actualize the proposed steering methods we utilized the temp tables for the transfering of intermediate information to the upper layers query execution plan. The information set of tables R, S, T and U is utilized to realize the unnesting query gets ready for the given four square settled query (Query 6). We additionally utilized temp tables for the propagation of intermediate information to the upper layers in the execution of query plans. We have done these tests for various information sets of shifting sizes from 100 to 1000 tuples in every connection. These results are taken as normal of a few iterations execution of every query plan. We used the DB2 database for our execution. The execution investigation of every one of those query plan is portrayed underneath.

B. Performance analysis

To check the effectiveness of the changed algorithm, we have taken 3 square settled query and produced the question arranges utilizing Nested cycle approach, Ganski's a pproach and the altered calculation. For every query, we quantified the normal execution time of different keeps running of the qu ery as essential execution metric. The charts of the outcomes plot the passed time on Y-hub and the measure of the information tables. The span of the tables means the quantity of tuples in every connection table in the inquiry. Here we have taken almost same n umber of tuples in every table. The extent of the table picked as an essential parameter for execution assessment because of the way that it specifically identifies with the middle of the road result, which thus, identifies with the overhead comparing to fetching tuples from the SQL engine. We have thought about best three existing calculations to be specific Modified calculation, Ganski's methodology and Nested cycle technique and demonstrated the outcomes in Figure 11. It can be effectively seen from the diagram that the slipped by time of the altered calculation is not exactly the ot her two. Along these lines it can be inferred that our proposed calculation performs superior to anything Ganski's methodology and nested iteration approach.



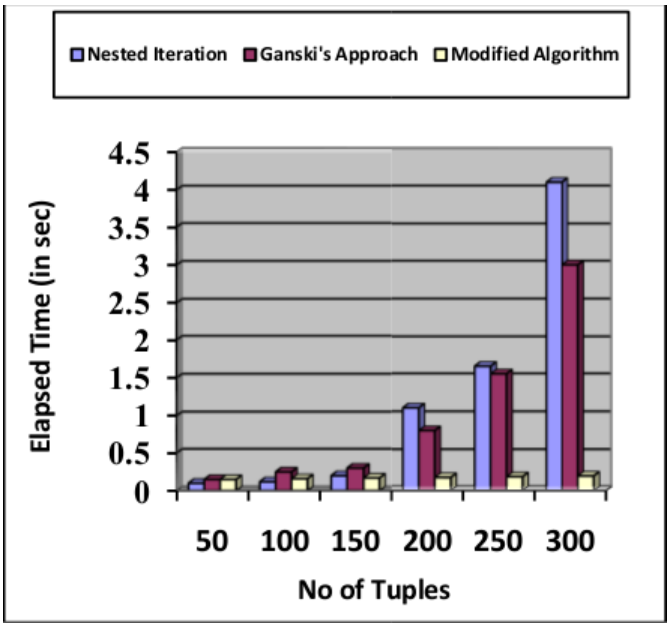
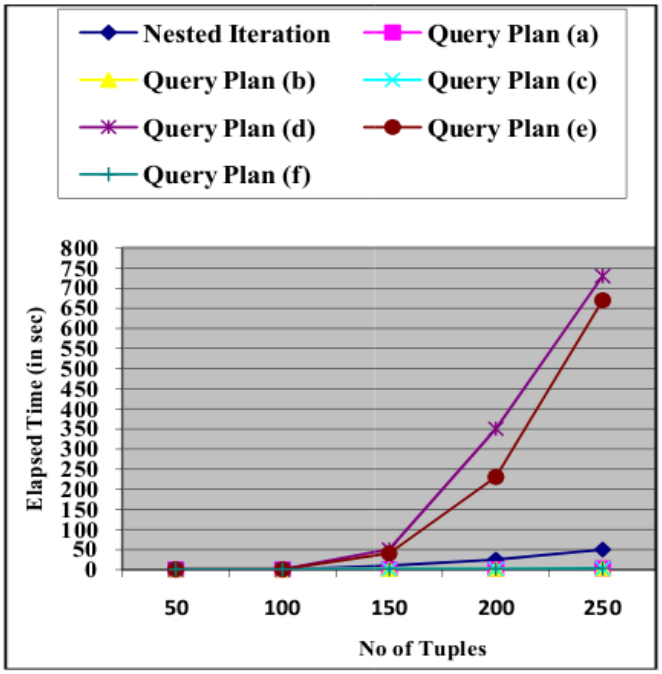


Figure 10. Modified Query Plan (d) **Figure 11. Performance compariso n of the three algorithms**

The execution examination of all query plans created by the incorporated calculation for the given four pieces settled inquiry (Query 6) is appeared in Figures 12. We have supplanted question arranges (b), (c) and (d) with altered inquiry arrangement (b), adjusted inquiry plan(c) and changed inquiry arrangement (d) individually. These changed arrangements are acquired in the wake of rectifying the defects as clarified in Section 6.

In figure 12, we look at the execution time of the first settled question and our six query plans. From the figure, we can see that query plan (d) and (e) take a great deal of time contrasted with all other query plans. In fact, the execution time is significantly more than the execution time for the nested query. As the quantity of tuples increase, the gap in the execution time of query plans (d) and (e) with that of alternate plans continues expanding.



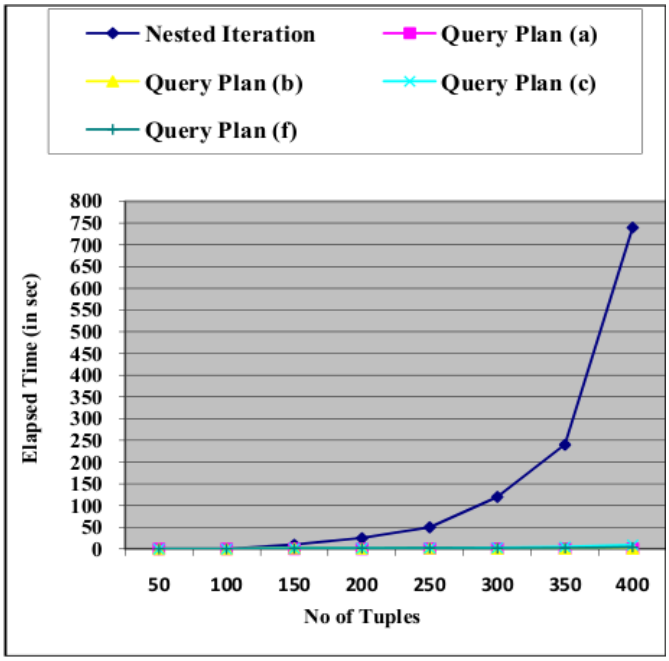


Figure 12. Execution time of nested itera tion and unnesting query plans

Figure 13. Execution time of nested iteration and unnesting query plans (a), (b), (c) and (f)

As the execution time of query plans (d) and (e) are much higher, we are not ready to look at the execution of other query plans with the nested iteration. To analyze alternate plans, we plot the execution time of just query plans (a), (b), (c) and (e) alongside the settled cycle. This correlation is appeared in figure 13. From figure 13, it can be seen that nested iteration takes much higher execution time than our proposed plans. The distinction in execution time increments exponentially as the quantity of iterations increment.

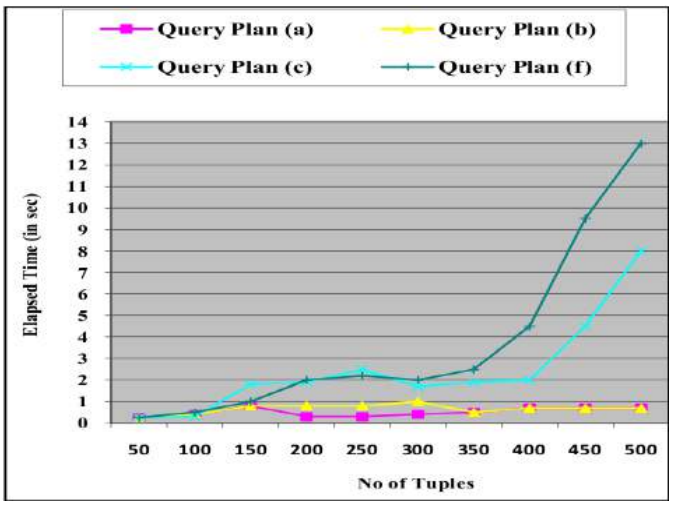


Figure14.Executiontime of unnesting query plans

To think about the execution of the four nesting query plans, we plot a chart wherein just these query plans are incorporated and nested iteration is excluded. This examination is appeared in Figure 14. It can be seen from the chart that the query plans (an) and (b) are computationally superior to anything query plans (c) and (f) regarding execution time.

VIII. CONCLUSIONS AND FUTURE WORK.

An imperative commitment of this paper is an effective execution of Kim's changed calculation acquired subsequent to expelling its imperfections. We have tentatively demonstrated that Muralikrishna's adjusted calculation of unnesting settled inquiries is greatly improved than Ganski's methodology or customary Nested iteration approach. We have given another answer for actualizing the adjusted calculation created by Muralikrishna. Notwithstanding this we have likewise enhanced the question arranges produced by the coordinated calculation, which improves and consolidates the beforehand known strategies for unnesting JA sort inquiries. Imperfections were resolved in the current calculation and alterations were proposed to wipe out these blemishes. We have effectively executed all the query plans in their current structure furthermore with our proposed adjustments. The execution times of all these query plans were measured and their execution was contrasted and that of nested iteration. Query plans (a), (b),(c) and (f) have been found to perform much better contrasted with nested iteration. Among these four query plans, query plans (a) and (b) have been observed to be computationally superior to the next query plans.

REFERENCES

[1] Astrahan, M. M, and Chamberlin, “Implementation of a structured english query language,” ACM, vol. 18, no. 10, pp. 580–588, October 1975.

[2] Astrahn, M. M, B. M. W, Chamberlin, D. D, Eswaran, K. P, Gray, J. N, G. P. P, K. W. F, Lone.RA, McJones, PR.Mehl, J. W, P. R, T. L, Wade, B. W, and Watscit, “System r relational approach to database management,” ACM Trans Database Syst, vol. 1, no. 2, pp. 97–137, June 1976.

[3] Codd and E. F, “Extending the database relational model to capture more meaning,” ACM Trans Database Syst, vol. 4, no. 4, pp. 397–434, December 1979.

[4] 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.

[5] R. Epstein, "Techniques for processing of aggregates in relational database systems," ERL/UCB Memo M79/8, Electronics Research Laboratory, Univ. of California, Berkeley, February 1979.

[6] R. A. Ganski, "Optimization of nested sql queries revisited," Proc. SIGMOD Conf., vol. 16, pp. 22–33, may 1987.

[7] W. Kiessling, "Sql-like and quel-like correlation queries with aggregates revisited," UCB-ERL Memorandum No. 84/75. University of California, Berkley, September 1984.

[8] W. Kim, "On optimizing an sql-like nested query," ACM Transactions on Database Systems (TODS), vol. 7, no. 3, pp. 443 – 469, 1982.

[9] 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.

[10] M. Muralikrishna, "Optimization and dataflow algorithms for nested tree queries," Very Large Data Bases Proceedings of the 15th international con- ference on Very Large data bases, pp. 77–89, August 1989.

[11] A. Rosenthal and C. Galindo-Legaria, "Query graphs, implementing tress, and freelyreorderable outerjoins," Proc. SIGMOD Conf., pp. 291–299, May 1990.

Publish Research Article

Dear Sir/Mam,

We invite unpublished Research Paper,Summary of Research Project,Theses,Books and Book Review for publication.

Address:- North Asian International Research Journal Consortium (NAIRJC)
221, Gangoo Pulwama - 192301
Jammu & Kashmir, India
Cell: 09086405302, 09906662570,
Ph No: 01933212815
Email:- nairjc5@gmail.com, nairjc@nairjc.com , info@nairjc.com
Website: www.nairjc.com