Improved query plans for unnesting nested SQL queries

Pranav Khaitan
Dept of Computer Science, Stanford University
pranavkh@cs.stanford.edu

Satish Kumar M, Korra Sathya Babu, Sanjay Kumar Jena
Department of Computer Science and Engg, NIT Rourkela
{ ksathyababu, skjena}@nitrkl.ac.in

Abstract - The SQL language allows users to express queries that have nested subqueries in them. Optimization of nested queries has received considerable attention over the last few years. Executing a query after unnesting it leads to lesser processing compared to executing it in the nested form itself. Kim's algorithm for unnesting nested queries has a COUNT bug for Join Aggregate type queries. Richard A. Ganski and U. Dayal gave general strategies to avoid the COUNT bug. M. Muralikrishna modified Kim's algorithm so that it avoids the COUNT bug. Apart from this, he proposed an integrated algorithm for generating query plans for a given input query. In this paper, we have given a solution for implementing the Kim's modified algorithm of unnesting nested queries and this also avoids the COUNT bug convincingly. We observed that the integrated algorithm holds flaws and modifications for correcting these flaws have been proposed in this paper. All existing and proposed query plans have been implemented and the experimental results have determined those query plans which are computationally better than others generated by integrated algorithm. The above merits have been incorporated into a new unnesting algorithm.

Keywords- query, optimization, sql, nested, unnesting

I. INTRODUCTION.

SQL is a block-structured query language for data retrieval and manipulation developed at the IBM Research Laboratory in San Jose, California. Query nesting is a very powerful feature of SQL that enhances the practical programming technique. Traditionally, database systems have executed nested SQL queries using Tuple Iteration Semantics (TIS). It has been shown analytically that executing queries by TIS can be very inefficient. It was pointed in [1, 2] 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. Further observation in [3, 4] stated that the unnesting techniques presented in [2] 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. This problem is popularly known as COUNT bug. Ganski [4] gave a solution that COUNT bug can be eliminated by using outer join. These solutions were further refined and extended in [5]. Muralikrishna came out with a modified Kim's algorithm with a good theoretical solution to avoid a COUNT bug.

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
```

When the above query is executed by the database engine, the result from the first query is given as an input to the next query. Query 1 computes the COUNT value associated with every distinct value in the c attribute of S. Notice that a tuple of R may djoin with at most one tuple of TEMP1. We observe that aggregate functions can be embedded in the query. Kim's algorithm works correctly if the aggregate is not a COUNT. However, in the presence of the COUNT aggregate, the algorithm gives rise to the COUNT bug [3, 4]. A tuple r of R would be lost after the join if it does not join with any tuples of S. However, the COUNT associated with r is 0 and if (r.b OP1 0) is true, r should appear in the result. In order to preserve tuples in R that have no joining tuples in S, an outer join (OJ) is performed when the COUNT aggregate is present between two blocks [4]. In this case, the unnested query becomes:

```
Query 3:
SELECT R.a
FROM   R, S
WHERE R.c = S.c - - - OJ
GROUP BY R. #
```

HAVING R.b OP1 COUNT (S.*)

The outer join preserves every tuple of R and hence the COUNT bug is avoided. If it can be determined at compile time that R.b can never equal 0, we could still use Kim's method. Notice that the outer join precedes the group by operation in Query 3. Ganski's solution is more general than Kim's solution as the former may be applied even in the presence of non equi-join correlation predicates [4].

B. The modified algorithm

Muralikrishna modified Kim's algorithm to avoid the COUNT bug. Consider the nested query used in the previous example. We apply the modified algorithm to the nested query to explain how it is applied to 2 block nested queries. Query 1, which creates the temporary relation TEMP1, remains unchanged. However, Query 2 is 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 r of R that does not join with any tuple of TEMPl will be a result tuple if (r.b OP1 0) is true. For a tuple r of R that joins with a tuple of TEMPl, r will be a result tuple if (r.b OP1 TEMP1.count) is true. The join operator in Query 2 is replaced by an outer join. The modified query is given below:

```
Query 5:
SELECT R.a
FROM R, TEMP1
WHERE R.c = TEMP1.c - - - - - - OJ
    [R.b OP1 TEMP1.count: R.b OP1 0]
```

The first predicate in square brackets is applied to the join tuples while the second predicate is applied to the anti-join tuples. A new way of expressing the above query using SQL is described in Section 3.

We show an implementation technique for the modified algorithm which successfully avoids COUNT bug. Query 5 can be written 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. An Integrated Algorithm

M. Muralikrishna [7] gave an Integrated Algorithm, which generates execution plans for unnesting nested queries. This algorithm generates execution plans by combining multiple ideas for unnesting queries. We have proposed the cheapest query plan among all others. The input to the algorithm is the graph G of the query and the output is a set of query plans. The pseudocode of our integrated algorithm is given below:

```
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)))
```

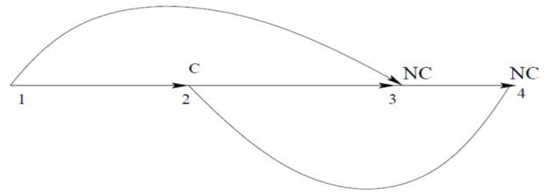

Figure 1. Graphical Representation of Query 6

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.

Most of the outer join nodes in Figure 2-7 have two output edges. The vertical edge represents the anti-join tuples, while the other edge represents the join tuples. Similarly, the GROUP BY HAVING nodes has two output edges. The vertical edge represents the groups that did not satisfy that condition in the HAVING clause. These groups have certain portions nulled out. For example, in Figure 5, groups flowing from the first group by having node to the topmost group by having node along the vertical edge are of the form (R, NULL) [6]. Also, Figures 3 - 7 have edges that route tuples to a node much higher in the tree than the immediate parent. As pointed out in [6], this is optional but leads to savings in communication costs.

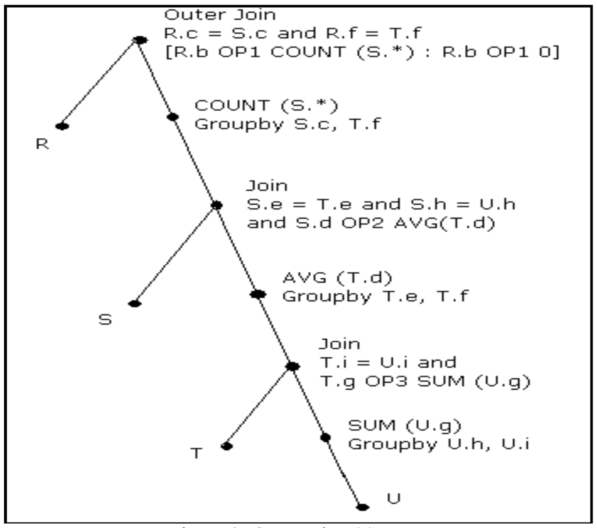

Figure 2. Query plan (a)

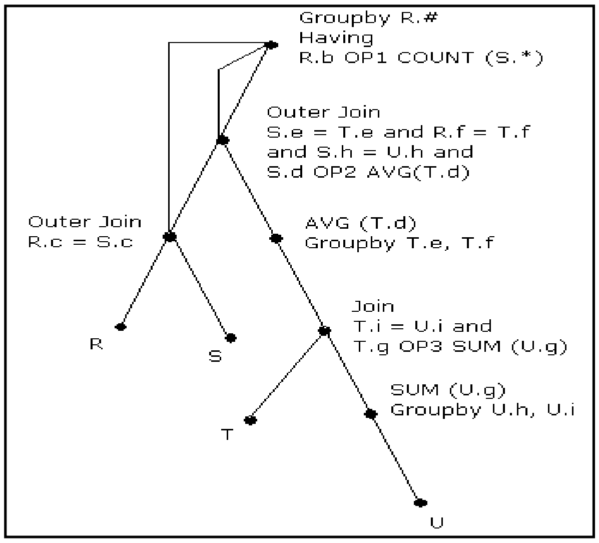

Figure 3. Query plan (b)

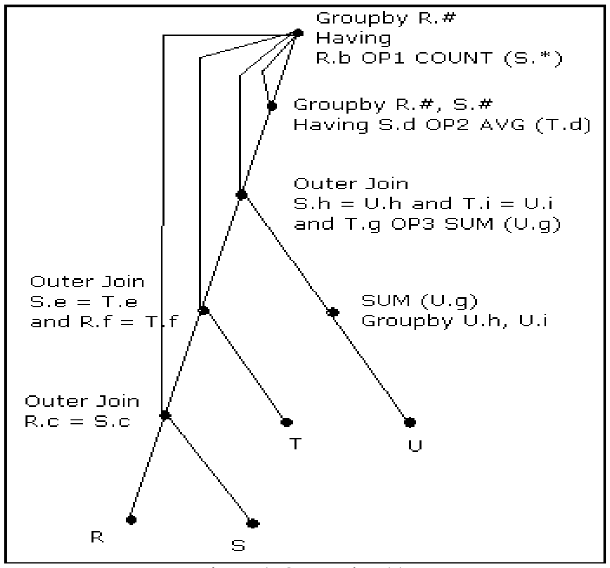

Figure 4. Query Plan (c)

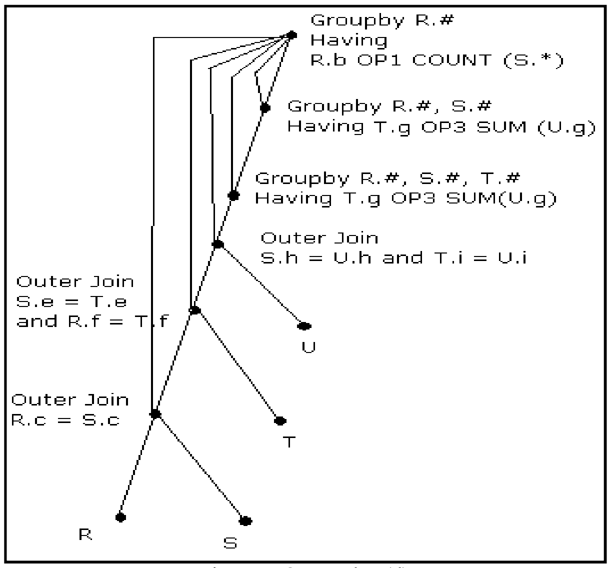

Figure 5. Query Plan (d)

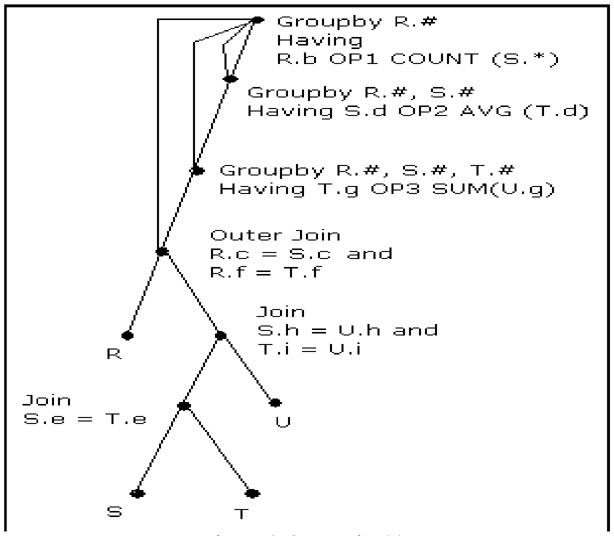

Figure 6. Query Plan(e)

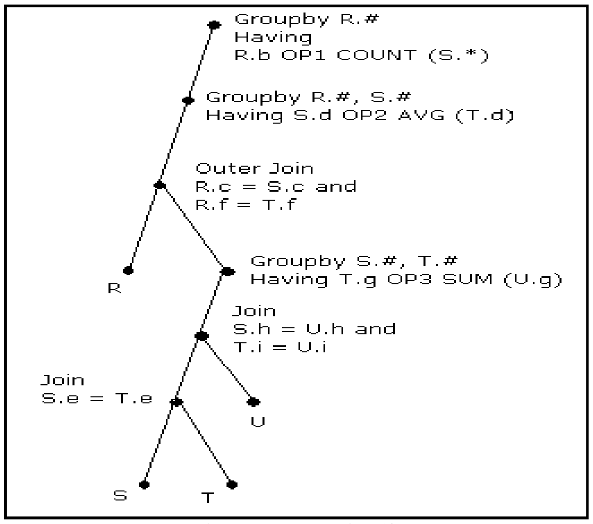

Figure 7. Query Plan (f)

IV. Flaws in Integrated Algorithm

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

- We first explain the flaw in Query (b). Consider a case when certain join tuple from the outer join operation (R OJ S) doesn't find any match in outer join operation in the next higher layer. However it goes through the path of anti-join tuples to reach the top most layer. In this case this anti join tuple is certain to come in the final outcome of the query if R.b is 0. But it doesn't happen like that if we execute our query plan (b). In query plan (b), this anti-join tuple carries the primary key value of table S (Ex: S.srn here) to the top most layer. Since S.srn is not null, the value of COUNT (S.srn) is non zero. Thus the predicate (R.b OP1 COUNT (S.srn)) becomes false.
- A similar problem exists in Query plan (c) and (d) also. The outer join operation between the join tuples of outer join (R OJ S) and table T propagates the primary key value of table S to the top most layer. There these anti join tuples are certain to appear in the final outcome of the query if R.b is 0. But it doesn't happen like that if we execute our query plans (c) and (d). In these query plans, this anti-join tuple carries the primary key value of table S (Ex: S.srn here) to the top most layer. Since S.srn is not null, the value of COUNT (S.srn) is non zero. Thus the predicate (R.b OP1 COUNT (S.srn)) becomes false.
  This is a common problem for all types of query plans which join R and S tables pair-wise. The general problem lies when joining two tables pair-wise which has a COUNT aggregate function between the two. In the above example, the problem lies in the query plans when we join tables R and S pair-wise as there is a COUNT aggregate function between them. The solution is explained in Section 6.

V. Solutions to the Flaws in Integrated algorithm

We propose the following solution to the flaws we have found in the integrated algorithm. The general solution is not to join the two tables when there exists a COUNT aggregate function between them, and to join these two tables with another table together in an unnesting query plan.

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 right wing together. Pass the primary key value of the 3rd table to the top most layer. We do it because the primary key value of the 3rd table is the primary key value of table S. For Query plans (c) and (d): Join (Outer join) tables R, S and T together. Propagate the primary key value of table T to top most layer.

Example: In this case, S.srn would be null for all anti join tuples and then (R.b OP1 COUNT(S.srn)) becomes true if R.b is zero. Then this tuple appears in the final outcome of the query as the predicate (R.b OP1 COUNT(S.srn)) becomes true. Thus, it has been shown that the solution works fine for all Query plans. Using the improved algorithm, Query plans (b), (c) and (d) are changed to as shown in Figures 8-11.

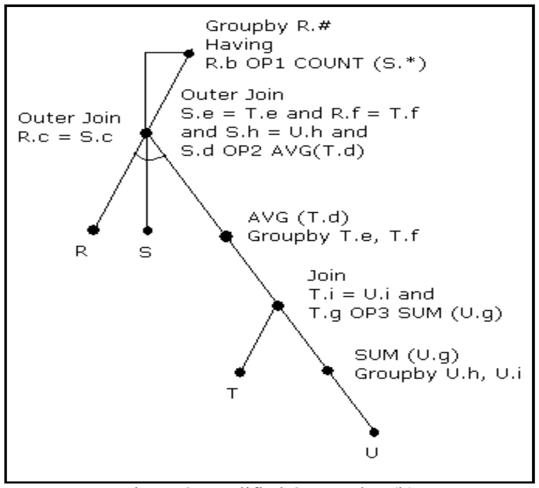

Figure 8. Modified Query Plan (b)

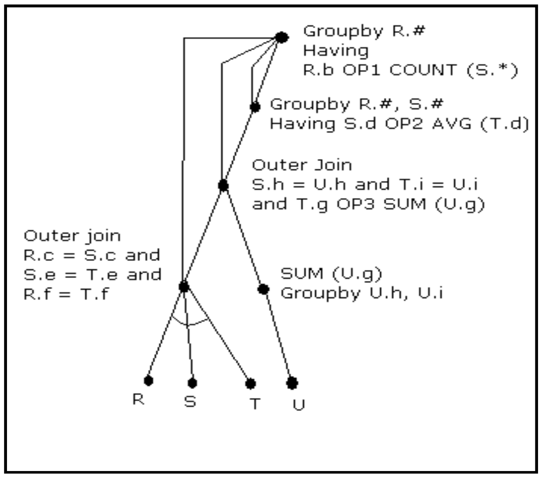

Figure 9. Modified Query Plan (c)

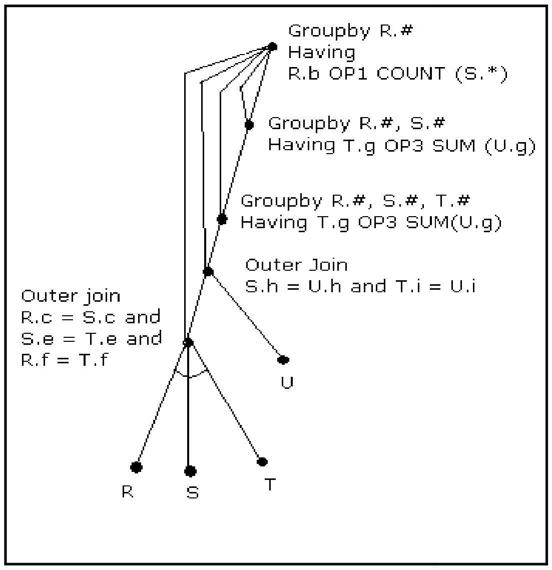

Figure 10. Modified Query Plan (d)

VI. EXPERIMENTAL RESULTS

A. Implementation

To implement the proposed 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 four block nested query (Query 6). We also 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 several iterations execution of each query plan. We deployed the DB2 database for our implementation. The performance analysis of all those query plans is described below.

B. Performance analysis

To verify the efficiency of the modified algorithm, we have taken 3 block nested query and generated the query plans using Nested iteration approach, Ganski's approach and the modified algorithm. For each query, we measured the average execution time of multiple runs of the query as primary performance metric. The graphs of the results plot the elapsed time on Y-axis and the size of the data tables. The size of the tables denotes the number of tuples in each relation table in the query. Here we have taken nearly same number of tuples in each table. The size of the table chosen as a primary parameter for performance evaluation due to the fact that it directly relates to the intermediate result, which in turn, relates to the overhead corresponding to fetching tuples from the SQL engine. We have compared best three existing algorithms namely Modified algorithm, Ganski's approach and Nested iteration method and shown the results in Figure 11. It can be easily seen from the graph that the elapsed time of the modified algorithm is less than the other two. Thus it can be concluded that our proposed algorithm performs better than Ganski's approach and nested iteration approach.

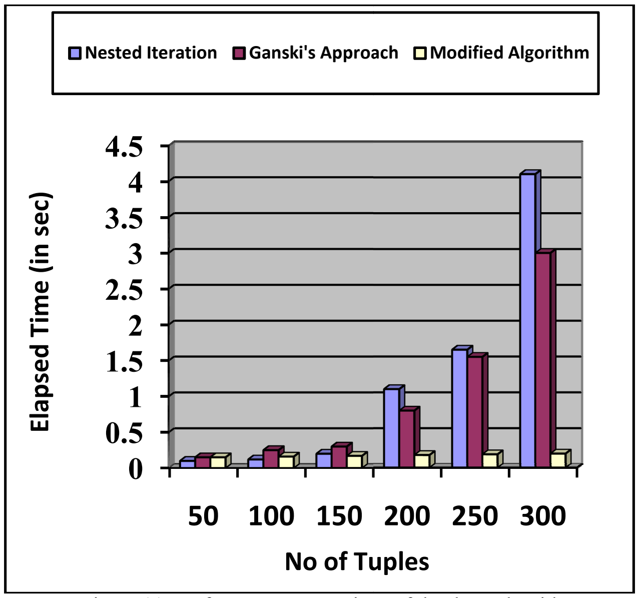

Figure 11. Performance comparison of the three algorithms

The performance analysis of all query plans generated by the integrated algorithm for the given four blocks nested query (Query 6) is shown in Figures 12 - 14. We have replaced query plans (b), (c) and (d) with modified query plan (b), modified query plan(c) and modified query plan (d) respectively. These modified plans are obtained after correcting the flaws as explained in Section 6.

In figure 12, we compare the execution time of the original nested query and our six query plans. From the figure, we can see that query plan (d) and (e) take a lot of time compared to all other query plans. In fact, the execution time is even more than the execution time for the nested query. As the number of tuples increase, the difference in the execution time of query plans (d) and (e) with that of the other plans goes on increasing.

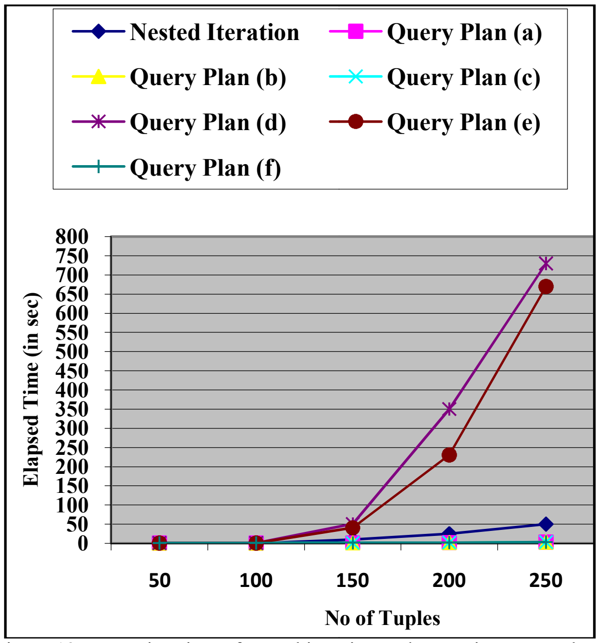

Figure 12. Execution time of nested iteration and unnesting query plans

Since the execution time of query plans (d) and (e) are much higher, we are not able to compare the performance of other query plans with the nested iteration. To compare the other plans, we plot the execution time of only query plans (a), (b), (c) and (e) along with the nested iteration. This comparison is shown in figure 13. From figure 13, it can be seen that nested iteration takes much higher execution time than our proposed plans. The difference in execution time increases exponentially as the number of iterations increase.

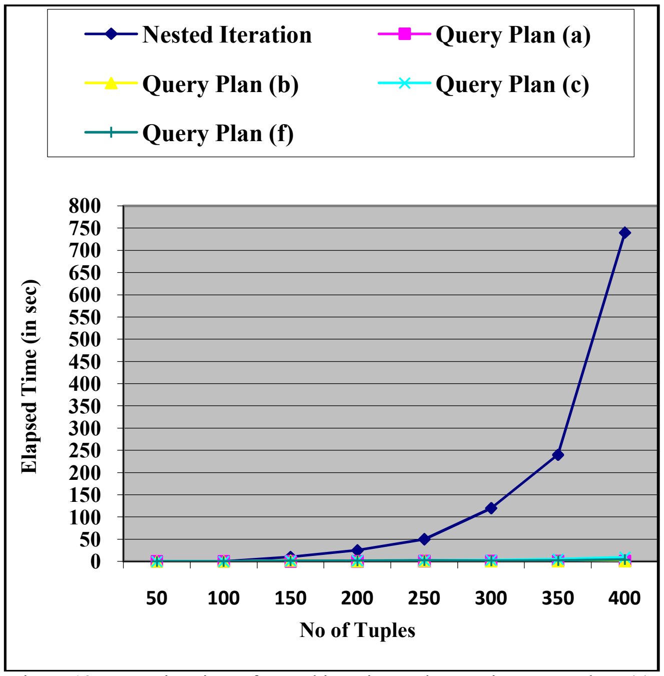

Figure 13. Execution time of nested iteration and unnesting query plans (a), (b), (c) and (f)

To compare the performance of the four nnesting query plans., we plot a graph wherein only these query plans are included and nested iteration is not included. This comparison is shown in Figure 14. It can be observed from the graph that the query plans (a) and (b) are computationally better than query plans (c) and (f) in terms of execution time.

VII. Conclusions and Future work.

An important contribution of this paper is a successful implementation of Kim's modified algorithm obtained after removing its flaws. We have experimentally proved that Muralikrishna's modified algorithm of unnesting nested queries is much better than Ganski's approach or traditional Nested iteration approach. We have given a new solution for implementing the modified algorithm developed by Muralikrishna. In addition to this we have also improved the query plans generated by the integrated algorithm, which enhances and incorporates the previously known techniques for unnesting JA type queries. Flaws were determined in the existing algorithm and modifications were proposed to eliminate these flaws. We have successfully implemented all the query plans in their existing form and also with our proposed modifications. The execution times of all these query plans were measured and their performance was compared with that of nested iteration. Query plans (a), (b), (c) and (f) have been found to perform much better compared to nested iteration. Among these four query plans, query plans (a) and (b) have been found to be computationally better than the other query plans.

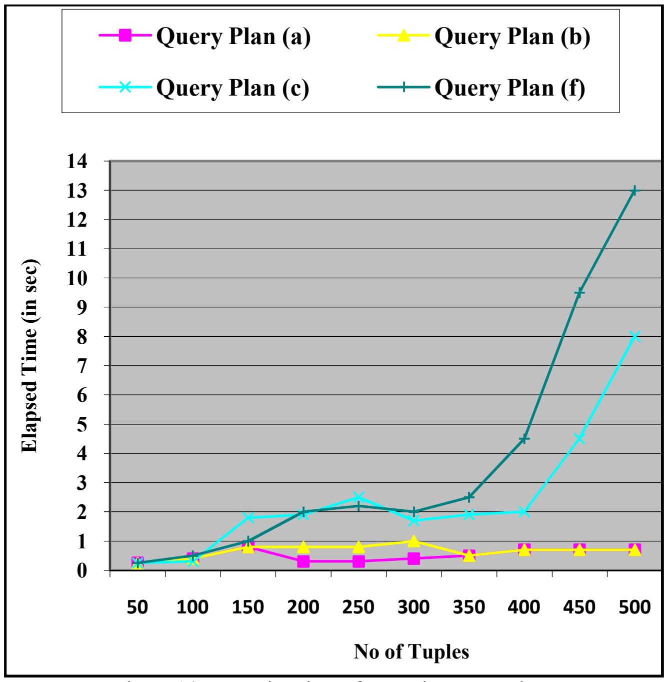

Figure 14. Execution time of unnesting query plans

VIII. References

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

[2] W. Kim, "On Optimizing an SQL-like Nested Query", Trans. on Database Systems, Vol 9, No. 3, (1982)

[3] W. Kiessling, "SQL-like and Quel-like correlation queries with aggregates revisited", UCB/ERL Memo 84/75, Univ. of California at Berkeley, (Sept. 1984).

[4] Richard A. Ganski and Harry K. T. Long, "Optimization of nested SQL Queries Revisited", Proc. SIGMOD Conf., pp. 23-33, (May 1987).

[5] U. Dayal, "Of Nests and Trees: A Unified Approach to Processing Queries That Contain Nested Sub queries, Aggregates, and Quantifiers," Proc. VLDB Conf., pp.197-202, (September 1987).

[6] M. Muralikrishna, "Optimization and Dataflow Algorithms for Nested Tree Queries", Proc. VLDB Conf., pp.77-85, (August 1989).

[7] 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).