Nested Query Optimization in Relational Database

Mr. Mahesh Singh
mahesh100nucs@gmail.com
Asst. Professor (Computer Science Department)
Advanced Institute of Technology and Management (Palwal)

Abstract

Many database applications perform complex data retrieval and update tasks. Nested queries, and queries that invoke user-defined functions, which are written using a mix of procedural and SQL constructs, are often used in such applications. A straightforward evaluation of such queries involves repeated execution of parameterized sub-queries or blocks containing queries.

This work discusses important aspects related to optimization and execution of these queries, which can be a rather complex task, which in turn reduces the overall response time of the application as it reduces the database access time for query evaluation.

Due to the flexible structures of SQL, no general approach works efficiently for all kinds of queries. Some special kinds of queries can be further optimized for better performance. In this paper we study two kinds of such queries: one is queries having non-aggregate sub-queries and the other is queries having redundancy. To deal uniformly with non-aggregate sub-queries in SQL, we propose the nested relational approach based on the nested relational model. To deal with redundancy in complex SQL queries, we propose the redundancy awareness method.

1. INTRODUCTION

Optimization has always been, and still is, a central topic in database research. The last 30 years have seen a tremendous amount of work in the topic, and that work still continues strong today. This is especially true in the context of Decision Support, where the amount of data keeps growing, queries keep on getting more and more complex, and answers are still expected quickly.

SQL, the de facto standard for relational query languages, supports more operations now than they did just a few years ago and this also adds to the complexity. Hence, query optimization continues to be an important topic. Being a declarative language, SQL relies on optimization techniques to provide responses in a timely manner. However, it is known that complex queries (with sub-queries, grouping, and large number of joins) generate so many choices for execution (i.e., possible query plans), that most optimization algorithms cannot guarantee that an optimal plan is chosen.

1.1 Nested Query

A nested query is a query that has another query embedded within it. An embedded query may appear in the FROM clause, or in the WHERE or HAVING clause (called a sub-query). The query that contains a sub-query is called a main query or an outer query.

1.2 Non Aggregate Sub-query

Depending on whether there is an aggregate function in the SELECT clause or not, sub-queries can be simply classified into two types: aggregate and non-aggregate.

An aggregate sub-query has an aggregate function in its SELECT clause; it always returns a single value as the result. A non-aggregate sub-query is linked to the outer query by one of the following operators: EXISTS, NOT EXISTS, IN, NOT IN, SOME/ANY, and ALL, where; the result is either a set of values or empty. An example query is shown below

Query 1:

\[
\text{select p_partkey, p_name from part }
\text{where p_size>=x1 and p_size<=x2 and }
\text{p_retailprice < all }
\text{(select ps_supplycost from partsupp }
\text{where ps_partkey=p_partkey and
}\]
1.3 Redundant Sub-query

Queries having aggregate sub-queries are widely used in data warehousing and decision support systems. One important thing to notice is that such queries usually show a great deal of redundancy, that is, the outer query and the sub-query share common tables and conditions. The following query is an example.

Query 2:

```sql
select sum(l_extendedprice)/7.0 from lineitem, part, orders where p_partkey=l_partkey and p_size=15 and p_type like '%BRASS' and l_shipdate>='1994-01-01' and l_shipdate<l_commitdate and l_orderkey=o_orderkey and o_orderdate>='1994-01-01' and o_orderdate<'1994-02-01' and l_quantity < (select 0.2*avg(l_quantity) from lineitem, partsupp where l_partkey=p_partkey and l_commitdate<l_receiptdate and l_shipdate<l_commitdate and l_suppkey=ps_suppkey and ps_availqty>5000)
```

2. Related Work and Motivations

Although significant research efforts have been devoted to optimization of nested queries, most proposed approaches concentrate on aggregate sub-queries. The solutions proposed for non-aggregate sub-queries still have some limitations. Generally, there are two methods for evaluating non-aggregate sub-queries. The first is derived from the approaches for aggregate sub-queries, that is, non-aggregate sub-queries are transformed into aggregate Sub queries first, and then the transformed queries are evaluated by the existing approaches. However, transformation of the NOT IN or ALL sub-query may not preserve semantics when null values are present. The second method is proposed directly to non-aggregate sub-queries. They all involve extending the standard relational algebra.

3. Proposed Work

Sub-queries are nested when they appear in the WHERE clause of the parent statement. When a database evaluates a statement with a nested sub-query, it must evaluate the sub-query portion multiple times and may overlook some efficient access paths or joins.

Sub-query un-nesting un-nests and merges the body of the sub-query into the body of the statement that contains it, allowing the optimizer to consider them together when evaluating access paths and joins.

To efficiently and uniformly evaluate queries having non-aggregate sub-queries, we propose the nested relational approach. Our approach is based on the nested relational algebra. The nested relational approach not only allows unnesting non-aggregate sub-queries directly without transformation, but also allows each sub-query to be evaluated in a uniform manner.

Redundancy is present because of the structure of SQL, which necessitates a sub-query in order to declaratively state the aggregation to be computed. With the addition of user-defined methods to SQL, detecting and dealing with redundancy is even more important, as many time such methods are expensive to compute and it is hard for the optimizer to decide whether to push them down or not.

To deal with redundancy in SQL queries, we propose the redundancy awareness method. We attack the redundancy problem directly, by identifying tables and conditions common to query and sub-query and executing this common part only once, and then compute aggregates and conditions involving them with one pass over the common part.

4. Nested Relational Approach

The basic idea of the nested relational approach is straightforward: we un-nest the query from top-down using existing techniques, and then we compute linking predicates(a linking predicate refers to the predicate that connects a sub-query and an outer query) from bottom-up. The second step requires the sub-query result to be a set, and a comparison between a single value and a set of values. Such operations are not supported by relational algebra. Thus we propose to use the nested relational algebra.

Our algorithm proceeds in three steps:
1) First, we reduce each query block to one relation by doing all operations in the WHERE clause except linking predicate and correlated predicate(s).

2) Second, we create a tree expression for the query as follows:

a) Walk through the query in Depth-First, Left-to-Right order; create one node for each query block.

b) We label each node with the corresponding Ti.

c) Between any two adjacent nodes Ti and Ti+1, we add an edge directed from Ti to Ti+1 labeled with the linking predicate Li.

d) If Ti+1 is correlated to Ti, we add the correlated predicate C(i+1)i to the edge.

e) If Ti is correlated to a non-adjacent node Tj (i > j), we add the correlated predicate Cij to the edge between Ti and Ti-1.

f) If all edges between Tj and Ti have been labeled with correlated predicates.

g) Otherwise, we add an edge directed from Tj to Ti labeled with the correlated predicate Cij.

h) The root is labeled by the name of the outermost query block, leaves are labeled by the name of innermost query blocks, and other nodes are labeled by the name of the middle query blocks.

i) A node is called a sub-root if it has more than one child. All nodes under a sub-root are called a sub-tree of the sub-root.

j) For a given node n, let name(n) be the Ti that serves as name of the node; linkC(n;m) be the Cij (if one exists) and link L(n;m) be the Li, which label the link between n and one of his children m.

3) Third, we compute (root, T1)

Then we process this query graph to get the final execution plan which gives us the query result.

5. Redundancy Awareness Method

For queries having redundancy, the conditions and tables in the outer query and the sub-query can be roughly divided into three parts: one that is common in both the outer query and the sub-query, one that belongs only to the outer query, and one that belongs only to the sub-query.

Based on these three parts, a query can be evaluated as follows: first, we create a base relation based on common tables and common conditions; second, starting from the base relation, we compute the aggregation in the sub-query based on the tables and conditions belonging only to the sub-query; finally, we generate the desired result based on the tables and conditions belonging only to the outer query and the sub-query result.

If the sub-query has extra tables, the base relation should be extended to include these tables; such a base relation is called the extended base relation.

6. Implementation

To verify the efficiency of the nested relational approach and the redundancy awareness method, we have implemented them on Oracle. First of all we have created the query graph and execution plans for our queries and then we use stored procedure in procedural SQL to get the desired results.

7. Conclusions and Future Work

This work proposes the nested relational approach and redundancy awareness method to efficiently evaluate queries having non-aggregate sub-queries and queries having redundancy respectively. Currently our work faces problem in case we have null values in our database. In near future we wish to expand this work in order to find effective solution for this problem also.

8. References

[3] Dandan Li, Lu Han, Yi Ding. SQL Query Optimization Methods of Relational Database System