"Multi Query Optimization Using Heuristic Apporach And Flow Of Optimizer To Evaluate Cost With The Use Of Greedy Technique"

Miss. Isha K. Gayki
P.R.Patil College of Engg & Technology,Amravati.

Prof. P. D. Soni
P.R.Patil College of Engg & Technology, Amravati.

Abstract

Today, it is very common that complex queries are being greatly used in the real time database applications. These complex queries have a lot of common sub-expressions, either within a single query or across multiple such queries run as a group or batch. Multi-query optimization used for common sub-expressions to reduce evaluation cost. Generally Multi-query optimization has been viewed as impractical, because previous algorithms were exhaustive, and require large search space.

But we observe that multi-query optimization using heuristics approach is practical, and it provides significant benefits. The optimization implements three cost-based heuristic algorithms: basic Volcano-SH and Volcano-RU, which are based on simple modifications applied to the Volcano algorithm, and a greedy heuristic. The greedy heuristic approach incorporates novel optimizations that improve efficiency of algorithm to optimize multiple queries. The algorithms are designed so that it can be easily added to existing optimizers.

1. Introduction

Complex queries are becoming commonplace, due to the advancement in tools that help to analyze information from large data warehouses. These complex queries often have a lot of common sub expressions since they make extensive use of views which are referred number of times in the query and most of them are correlated nested queries in which parts of the inner sub query may not depend on the outer query variables[13]. The scope for finding common sub-expressions increases greatly if we consider a set of queries executed as a batch or group.

Analysis of data or reporting often requires a batch of queries to be executed. A set of related materialized views also generates related queries with common sub expressions. Materialized views are increasingly being supported by commercial database systems, and are used to speed up query processing. Data warehouses, which store large volumes of data, depend on materialized views for efficient query processing, and the materialized views as well as expressions have significant amounts of common sub expressions. Sharing of common sub-expressions is also have importance when the expressions access remote data, and hence it becomes expensive. Here we address the problem of optimizing sets of queries which may have common sub expressions; this problem is referred to as *multi-query optimization*. It is also possible that common sub expressions are present even within a single query; the techniques developed are used to deal with such intra-query common sub expressions[14].

Traditional query optimizers are not beneficial for optimizing queries with common sub expressions, since they make locally optimal choices.

Let us take an example, Where we have seen that, in SQL, a query could be expressed in several different ways. Each SQL query can be translated into a relation-algebra expression in one of the several ways. The relational algebra representation of a query specifies only how to evaluate a query; there are several ways to evaluate relational algebra expressions. Let us consider the query for selecting the marks of student more than 60 columns from student database.

SELECT *marks*
FROM *student*
WHERE *marks>60;*

This query can be translated into either of the following relational algebra expressions:-

1. σ *marks>60* (Π *marks*(*student*))
2. Π *marks*(σ *marks* >60(*student*))

- Cost estimation in query optimization
- Calculation of cost
- Query processing

2. Cost estimation in query optimization

The cost can be evaluated by considering the properties of hard disk with respect to time.

2.1. Cost parameters

The three main disk scheduling parameters are:

- Seek time
- Latency time
- Transfer time

- Seek Time

Seek time is the time required to move a read-write head from one track to another.

- Rotational Latency

At the movement when read-write head is positioned onto the correct cylinder, it waits for the requested sector to rotate beneath it. This waiting time is called as rotational latency.

- Transfer Rates

It is a time for data to be read by the disk head, i.e., time required to move the head across the particular block of a sector.

Transfer time = amount of data per track / time per rotation

2.2. Calculation of cost

The nested loop algorithm reads one record from one relation, and passes each record of the outer relation to the inner relation; also join the record of the outer relation with suitable records of the inner relation. The next record from the outer relation is again read and entire inner relation is again scanned, and so on. The nested block algorithm works by reading a block of records from the outer relation and passing over each record of the inner relation, joining the records of the outer relation with those of the inner relation. If there are B pages in the memory, B-2 pages are usually allocated to the outer relation, one to the inner relation, and one to the result relation. For cost estimation required parameters are as shown in the table.

Notation	Meaning
V1	Number of pages in relation R1
V2	Number of pages in relation R2
V_r	Number of pages in result of joining relation R1 and R2
B	Number of pages in memory for the use in buffers
B1	Number of pages in memory for relation R1
B2	Number of pages in memory for relation R2
B_R	Number of pages in memory for result

Table 1: Notation to evaluate cost

The time taken to perform an operation x as Tx. Table below shows the default values used to calculate the results

Notation	Meaning	Values
T_C	Cost of constructing a hash table per page in memory	0.015
T_K	Cost of moving the disk head to the page on disk	0.0243
T_J	Cost of joining a page with a hash table in memory	0.015
T_T	Cost of transferring a page from disk to memory	0.013

Table 2: Parameter for cost estimation

Cost of transferring a set V1 pages through buffer of size B1:

$$C_{Read\ R1} = C_{I/O}(V1,B1)$$

Cost of creating Hashed Pages from V1 pages:

$$C_{Create} = V1T_C$$

Cost of transferring a ser V2 pages through buffer of size B2:

$$C_{Read\ R2} = C_{I/O}(V2,B2)$$

Cost of Joining each hashed page with V2 pages

$$C_{Join} = V_2T_J$$

Cost of Writing back the result into the disk drive:

$$C_{Write\ RR} = C_{I/O}(V_R,B_R)$$

Total Cost of Operation:

$$C_{NB} = C_{Read\ R1} + C_{Create} + C_{Read\ R2} + C_{Join} + C_{Write\ RR}$$

3. Query processing

How optimizer works?

Phases of optimization are implemented in the execution of a query. Query processing refers to the range of activities involved in extracting the data from database. The activities include transformation of queries in high level database languages into expressions that can be used at physical level of the file system, a variety of query optimizing transformations and actual evaluation of queries[15].

The basic steps involved in the query processing are

1) Parsing and translation
2) Optimization
3) Evaluation

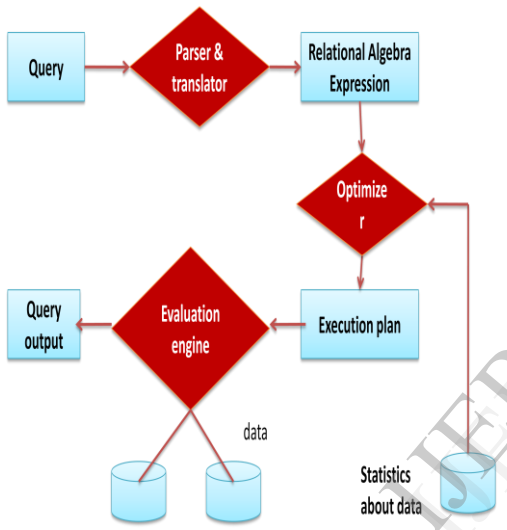

Figure 1: Query processing

4. Operations in Query-Processing

- Selection
- Join Operation
  a) Nested –Loop Join
  b) Hash Join

5. Technique: greedy

Algorithm
Procedure GREEDY
Input: Expanded Directed acyclic graph nodes
Output: Set of nodes obtained to form corresponding best plan
X = *NULL*
Y = set of equivalence nodes in the DAG
while (Y ≠ *NULL*)
L1: Pick the node x ∈ Y with the smallest value for bestplan(Q, {x} ∪ X)
if (bestplan(Q, {x} ∪ X) < bestplan(Q, X))
Y = Y - x; X = X ∪ {x}
else Y = *NULL* // benefit < 0, so break out of loop
return X

6. Flow of Optimizer

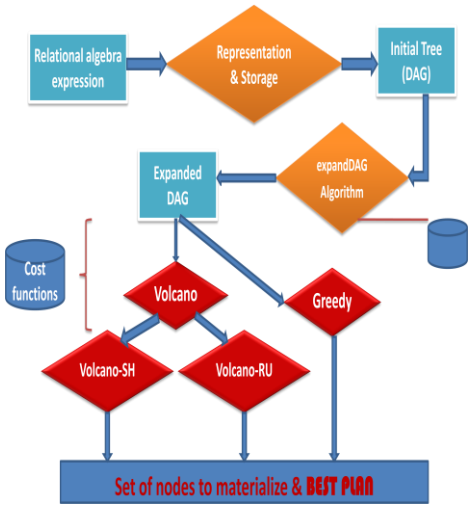

Figure 2: Flow of optimizer

7. Advantages

7.1. Sharing

The optimization based on that the nodes materialized forms optimal plan are subset and shared by other queries. Therefore, it is essential to initialize Y, with nodes that are shared by another query. Called as sharable nodes[15].

7.2. Incrementing Cost Update

The second optimization is based on there are many calls to bestplan , with different parameters. Symmetric difference in the sets passed as parameters to successive calls to bestplan is very small – successive calls take parameters of the form bestplan(Q, {x} ∪ X), where only x varies.

7.3. Monotonic

The third optimization, which we call the monotonic avoids having to invoke *bestplan(Q,{x} ∪ X)*, for every x ∈ Y .

8. Future Work

The results in this paper form the basis for a significant amount of future work. The algorithms or techniques can be extended to deal with space constraints on materialized results. A more challenging problem is how to schedule computations so that temporary storage space can be reused during computation.

Another important area of future work lies in dealing with large sets of queries (large workloads); the size of the workload can be reduced by abstracting queries, for instance by replacing queries that differ in just selection constants by a parameterized query, invoked multiple times.

Applying multi-query optimization to incremental update expressions for materialized views, it also apply to these results to the problem of materialized view or index selection, where update costs need to be taken into account.

9. Conclusion

There are number of techniques to greatly speed up the execution time for multiple queries. Techniques depend upon AND-OR DAG representation of queries, are thereby can be easily extended to handle new operators. Implementation demonstrated that the algorithms can be added to an existing optimizer with a reasonably small amount of effort. Multi-query optimization is practical and gives significant benefits at a reasonable cost. The benefits of multi-query optimization were also demonstrated on a real database system. In conclusion, we believe we have laid the groundwork for practical use of multi-query optimization, and multi-query optimization will form a critical part of all query optimizers in the future.

10. References

[1] [CKPS95] Surajit Chaudhuri, Ravi Krishnamurthy, Spyros Potamianos, and Kyuseok Shim. Optimizing queries with materialized views. In *Intl. Conf. on Data Engineering*, Taipei, Taiwan, 1995.

[2] [CLS93] Ahmet Cosar, Ee-Peng Lim, and Jaideep Srivastava. Multiple query optimization with depth-first branch and bound and dynamic query ordering. In *Intl. Conf. on Information and Knowledge Management (CIKM)*, 1993.

[3] [CR94] C. M. Chen and N. Roussopolous. The implementation and performance evaluation of the ADMS query optimizer: Integrating query result caching and matching. In *Extending Database Technology (EDBT)*, Cambridge, UK, March 1994.

[4] [Fin82]S. Finkelstein. Common expression analysis in database applications. In *SIGMOD Intl. Conf. on Management of Data*, pages 235–245, Orlando,FL, 1982.

[5] [FLMS99] Daniela Florescu, Alon Levy, Ioana Manolescu, and Dan Suciu. Query optimization in the presence of limited access patterns. In *SIGMOD Intl. Conf. on Management of Data*, 1999.

[6] [Gup97] H. Gupta. Selection of views to materialize in a data warehouse. In *Intl. Conf. on Database Theory*, 1997.

[7] [LQA97] W.L. Labio, D. Quass, and B. Adelberg. Physical database design for data warehouses. In *Intl. Conf. onData Engineering*, 1997.

[8] [LY85] P. A. Larson and H. Z. Yang. Computing queries from derived relations. In *Intl. Conf. Very Large Databases*, pages 259–269, Stockholm, 1985.

[9] [PGLK97] Arjan Pellenkoft, Cesar A. Galindo-Legaria, and Martin Kersten. The Complexity of Transformation-Based Join Enumeration. In *Intl. Conf. Very Large Databases*, pages 306–315, Athens,Greece, 1997.

[10] [PS88]Jooseok Park and Arie Segev. Using common sub-expressions to optimize multiple queries. In *Proc.IEEE CS Intl.Conf. on Data Engineering 4, Los Angeles.*, February 1988.

[11] [RR82] A. Rosenthal and D. Reiner. An architecture for query optimization. In *SIGMOD Intl. Conf. on Management of Data*, Orlando, FL, 1982.

[12] [RR98] Jun Rao and Ken Ross. Reusing invariants: A new strategy for correlated queries. In *SIGMOD Intl. Conf. on Management of Data*, Seattle, WA, 1998.

[13] [RSR+99] Prasan Roy, Pradeep Shenoy, Krithi Ramamritham, S. Seshadri, and S. Sudarshan. Don't trash your intermediate results, cache 'em. Submitted for publication, October 1999.

[14] [RSS96] Kenneth Ross, Divesh Srivastava, and S. Sudarshan. Materialized view maintenance and integrity constraint checking: Trading space for time. In *SIGMOD Intl. Conf. on Management of Data*, May 1996.

[15] [RSSB98] Prasan Roy, S. Seshadri, S. Sudarshan, and Siddhesh Bhobe. Practical algorithms for multi query optimization. Technical report, Indian Institute of Technology, Bombay, October 1998.