Design of Optimization in Object–Oriented Queries

M.C. Nikose[1], S.S. Dhande[2] and G. R. Bamnote[3]

**[1] Student, Computer Science & Engineering,
Sipna's College of Engg & Tech. Amravati, Maharashtra. India.
monalinikose18@gmail.com
[2]Asst. Prof. , Computer Science & Engineering,
Sipna's College of Engg & Tech. Amravati, Maharashtra. India.
dhande_123@rediffmail.com
[3]Prof., Computer Science & Engineering,
P.R.Meghe Institute of Tech. & Research, Badnera, Maharashtra. India.
grbamnote@rediffmail.com**

Abstract

Query optimization is the process of finding the best or rather a reasonably efficient execution plan, thus by minimizing the time of query evaluation & the cost of evaluation to the level accepted by user. When a query jointly addresses very large and small collections, the iteration caused by query operator is driven by large collection and in each cycle a subquery which depends on an element of small collection is evaluated. The result return by subquery for such each element is same. Moreover, such a subquery is unnecessarily evaluated many times. The underlying idea used here is to rewrites such a query so that the loop is performed on small collection and inside each its cycle a subquery addressing a large collection is evaluated.

Keywords: Query optimization, Subquery, Execution plan, Rewriting.

I. INTRODUCTION

Nowadays, the necessity to support complex data in databases is intensified. Models trying to answer to these needs appeared as the object-oriented and the object relational model. Relational languages are amplified to a big extent by the idea of declarative query languages, notably SQL. However, the relational model only permits the alphanumeric data management. A similar role in object-oriented database is fulfilled by object query languages. The usefulness of these languages strongly depends on query optimization [5]. The data model of a DBMS lays down the possible structure of the data; to provide easy access to the user, a high-level query language is supported. The implementation of such a high-level query language requires an enormous effort; it is the task of the query optimizer to ensure fast access to the data stored in the database.

In recent years, database research has concentrated on object-oriented data models, which allow to store highly structured data. With regard to the data structuring concepts offered, an object-oriented data model can be looked upon as an extension of the nested relational model, which allows to store relations as attribute values. The nested relational model, in turn, is an extension of the relational model, which allows for flat table structure only. The relational model permits only the alphanumeric data management. A similar role in object-oriented database is fulfilled by object query languages. The usefulness of these languages strongly depends on query optimization. With growing complexity of data structuring concepts, the complexity of the accompanying query language grows as well and thus also the complexity of query processing and optimization.

Query processing and its optimization have been two of the most popular areas of research in the database community. Query processing is the sequence of actions that takes as input a query formulated in the user language and delivers as result the data asked for. Query processing involves query transformation and query execution. Query transformation is the mapping of queries and query results back and forth through the different levels of the DBMS. Query execution is the actual data retrieval according to some access plan. An important task in query processing is query optimization. Usually, user languages are high-level, declarative languages allowing to state what data should be retrieved, not how to retrieve them. For each user query, many different execution plans exist, each having its own associated costs. The task of query optimization ideally is to find the best execution plan, i.e. the execution plan that costs the least, according to some performance measure. Usually, one has to accept just feasible execution plans, because the number of semantically equivalent plans is too large to allow for enumerative search.

II. OBJECTIVES

Query optimization is important from various point of view, some of these are discuss here. First, it provides the user with faster results, which makes the application seem faster to the user. Secondly, it allows the system to service more queries in the same amount of time, because each request takes less time than unoptimized queries. Thirdly, query optimization ultimately reduces the amount of wear on the hardware (e.g. disk drives), and allows the server to run more efficiently (e.g. lower power consumption, less memory usage).

Objectives of query optimization are,

1. **To retrieve data quickly:** Query optimization is the refining process in database administration and it helps to bring down speed of execution. Most of the databases after are built and filled with data, and used come down on speed. The time taken to execute a query and return results exponentially grows as the amount of data increases in the database leading to more waiting times on the user, and application sides. Sometimes the wait times could range from minutes, to hours, and days as well in worst cases.

   Technically one of the reasons of slow speeds of executions could be excess normalization leading to multiple tables. More the number of tables, more is the complex nature of joins, and thus leading to more execution times. Sometimes, the complex nature of joins could worse the situation by bring the execution into deadlock.
2. **To reduce the system resources**: The resources required to fulfill a query are reduces, and ultimately provide the user with the correct result set faster.
3. **The goal of the query optimizer is the best throughput:** This means that it chooses the least amount of resources necessary to process all rows accessed by the statement.

III. LITERATURE REVIEW & RELATED WORK

The task of a Database Management System (DBMS) is to safely store (usually large amounts of) consistent data, and to provide easy and fast access to these data, either for retrieval or update purposes. The data model of a DBMS lays down the possible structure of the data; to provide easy access to the user, a high-level query language is supported. The implementation of such a high-level query language [1] requires an enormous effort; it is the task of the query optimizer to ensure fast access to the data stored in the database.

2.1 When to Perform Optimization

Generally approaches to query optimization are subdivided into two main categories with regard to when it is performed [5].

- **Static Optimization**: If Optimization is performed once at compile time. For each invocation at run time, the plan is activated and then executed. If, however, the state of a DBMS changes frequently, it is usually useful to optimize the query a new for each invocation. This compile time optimization is called as static, in that it cannot use the information available at run time, such as current statistics. A common solution is to periodically re-optimize queries with new statistics, based on the pace at which the relevant statistics change. For instance, [14] describe a schema in which query execution plans generated by an optimizer are re-optimized just before query execution time if they are believed to be sub-optimal.

- **Dynamic Optimization:** Dynamic optimization is a mix of compile time and run time optimization i.e., the plan produced at compile time includes plan alternatives, and the actual plan to be executed is chosen at run time. If only a part of it is rendered before query execution and the rest is made during evaluation (i.e., at run time), then it is referred to as dynamic. Due to the incomparability of costs at compile time alternative plans are ordered only partially, and the final choice is delayed until start-up-time, when all run time bindings can be instantiated. Then cost calculations and comparisons become feasible and the optimal plan can be chosen and evaluated.

Query optimization is an engineering art that seeks for any possible invention aiming at reducing query evaluation time. Although query optimization is supported by some theories e.g. Relational model, monoid calculus, in general this support concerns only few methods some of these are given below. There is lot of specific cases in a database environment and in a query language that can be the subject of method aiming at radical improvement of the query evaluation time. The major group of methods concerns the redundant access support data structure such as indices [6]. Other methods concern caching query results in order to reuse them. Another class of method includes physical data organization that is design to support processing of queries. The general strategies of query optimization are

1. Avoid Evaluating Cartesian Products
2. Perform selection as Early as Possible
3. Perform Projection as Early as Possible
4. Combine Sequences of Unary Operations
5. Identify Common Subexpressions in an Expression
6. Evaluate Options
7. Preprocess Data Files
8. Indexing
9. Calculate Constant Expression

The above methods don’t gives satisfactory result so I proposed a new method for query optimization.

IV. ANALYSIS OF PROBLEM

While analyzing query processing in the optimization model, it observed that not only some sub-queries are evaluated many times in the loops implied by the non-algebraic operators but also the result of these subquiries is same in subsequent loop cycle. Such a sub-queries unnecessary evaluated many times, thus by increasing the cost of execution and the time required to execute the queries. In spite, such sub-queries can be processed only once and the result can be reused in next loop cycles [1].

V. PROPOSED WORK

There are various methods available for query optimization discuss in [5]. In this project we are proposing one of the methods of query optimization as method of independent subqueries. The idea of this method is based on the observation that if none of the names in a subquery is bound by the algebraic operator currently being evaluated, then this subquery is independent of this operator [1][3]. The method modifies the textual form of query so that all its subqueries are evaluated as soon as possible.

This method is based on query rewriting that deals with optimization method at textual level. Rewriting requires performing a special phase called static analysis [1]. During the static analysis we simulate run time query evaluation to gather all the information that we need to optimize queries. The advantage of rewriting is that algorithms are fast, optimization is made before a query is executed, and the resulting performance improvement is very significant.

The entire process of query optimization is proposed as follows:

1. **First take a query as an input:** A query is an expression that describes information that one wants to search for in a database. The result of a query is a relation described with the simple attributes.
2. **Rewrite step:** Rewriting means transforming a query q1 into a semantically equivalent query q2 promising much better performance. It consists in detecting parts of a query matching pattern. When it is recognized, a query is rewritten according to the predefine rewritten rule.
3. **Execution step:** Permits to choose the optimal execution plan and to execute it.
4. **Comparative study of proposed method:** Here we are comparing our proposed method with previously available method in term of time required to execute the query.

The method of independent subqueries covers a more general case than e.g. pushing selection before join. The method is independent of the kind of operator connecting an independent subquery to the outer subquery.

VI. CONCLUSION

Finding the optimal strategy is usually too time-consuming except for the simplest of queries and may require information on how the files are implemented and even on the contents of the files, information that may not be fully available in the DBMS catalog. Hence, planning of an execution strategy may be a more accurate description than query optimization. Here we are trying to reduce the time and the cost required to execute an query.

REFERENCES

[1]**[JPlod00]** J. Płodzień, A. Kraken, "Object Query Optimization through Detecting Independent Subqueries", *Information Systems*, Elsevier Science, 25(8), 2000, pp. 467-490.

[2]**[Mich09]** Michel Bleja, Krzysztof Stencel, Kazimierz Subeita, "Optimization of Object-Oriented Queries Addressing Large and Small Collections", Proc. Of the IMCSIT, 2009, ISBN 978-83-60810-22-4, Vol. 4, pp. 643-680.

[3]**[PlSu99]** J. Plodzien, "Optimization of Object-Oriented Queries by Factoring out Independent Subqueries". Institute of Computer Science PAS, Report 889, Warsaw, November 1999.

[4]**[Subi95]** K.Subieta, C.Beeri, F.Matthes, J.W.Schmidt. "*A Stack-Based Approach to Query Languages"*. Proc.2nd East-West Database Workshop, 1994, Springer Workshops in Computing, 1995, 159-180.

[5]**[Plod00]** J. Plodzien, "Optimization Methods in Object query Languages", Ph. D. Thesis, Institute of Computer Science, Polish Academy of Sciences, 2000.

[6][Adam08] R.Adamus, M.Daczkowski, "Overview of the Project ODRA". Proceedings of the First International Conference on Object Databases, ICOODB 2008, Berlin 13-14 March 2008, ISBN 078-7399-412-9, pp.179-197.

[7][CIDe92]Cluet, C. Delobel. "A General Framework for the Optimization of Queries". Proc. Of SIGMOD Conference, 383-392,1992.