OBJECT QUERY OPTIMIZATION THROUGH DETECTING INDEPENDENT SUBQURIES

Amol U Ingle#1, Dr. A. S. Alvi\*2

#Department Information Technology, P.R.M.I.T & R, Badnera. M.S. India.

1samyaastulkar@hotmail.com

\*H.O.D. Department Information Technology, P.R.M.I.T & R, Badnera, M.S. India.

2abrar\_alvi@rediffmail.com

***Abstract*— In recent years, database research has concentrated on object-oriented data models, which allow to store highly structured data. 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 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.**

**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, Query Evaluator, Parser, Query Processor, Optimizers.**

1. 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. 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. 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 processingis 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 transformationis 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 howto 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 to large to allow for enumerative search.

# RELATED WORK

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 [5]. 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 Sub expressions in an Expression
6. Evaluate Options
7. Preprocess Data Files
8. Indexing
9. Calculate Constant Expression

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

# III. 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.

# IV. Proposed Work and Objectives

Fig 1: General Schema of query processing.

1. The text of a query is parsed and syntax tree is constructed.

2. Query is optimized by rewriting.

3. The static analysis involves a metabase (a data structure obtained from a database scheme), a structure simulating the query result stack (static ES) and a structure simulating the query result stack.

4. After optimization the query is evaluated, the evaluation involves the run time object store, ES and QRES.

How the Query Optimizer Works

At the core of the SQL Server Database Engine are two major components the **Storage Engine** and the **Query Processor** also called the Relational Engine. The Storage Engine is responsible for reading data between the disk and memory in a manner that optimizes concurrency while maintaining data integrity. The Query Processor, as the name suggests, accepts all queries submitted to SQL Server, devises a plan for their optimal execution, and then executes the plan and delivers the required results.

Queries are submitted to SQL Server using the SQL language (or T-SQL, the Microsoft SQL Server extension to SQL). Since SQL is a high-level declarative language, it only defines what data to get from the database, not the steps required to retrieve that data, or any of the algorithms for processing the request. Thus, for each query it receives, the first job of the query processor is to devise a plan, as quickly as possible, which describes the best possible way to execute said query (or, at the very least, an efficient way). Its second job is to execute the query according to that plan.[1]

Each of these tasks is delegated to a separate component within the query processor; the **Query Optimizer** devises the plan and then passes it along to the **Execution Engine**, which will actually execute the plan and get the results from the database.

In order to arrive at what it believes to be the best plan for executing a query, the Query Processor performs a number of different steps, the entire query processing process is shown on figure 2.

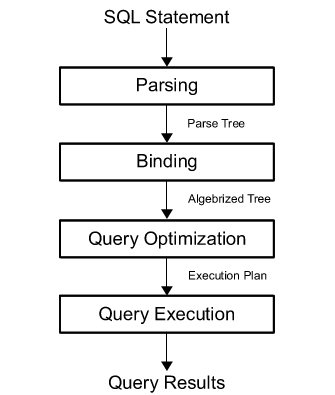


Fig 2 - The Query Processing Process

Parsing and binding: - the query is parsed and bound. Assuming the query is valid, the output of this phase is a logical tree, with each node in the tree representing a logical operation that the query must perform, such as reading a particular table, or performing an inner join. This logical tree is then used to run the query optimization process, which roughly consists of the following two steps;

Generate possible execution plans: – using the logical tree, the Query Optimizer devises a number of possible ways to execute the query i.e. a number of possible execution plans. An execution plan is, in essence, a set of physical operations (an index seek, a nested loop join, and so on), that can be performed to produce the required result, as described by the logical tree;

Cost-assessment of each plan: – While the Query Optimizer does not generate every possible execution plan, it assesses the resource and time cost of each plan it does generate. The plan that the Query Optimizer deems to have the lowest cost of those it’s assessed is selected, and passed along to the Execution Engine;

Query execution, plan caching: – the query is executed by the Execution Engine, according to the selected plan. The plan may be stored in memory, in the plan cache.

Parsing and binding are the first operations performed when a query is submitted to a SQL Server instance. Parsing makes sure that the T-SQL query has a valid syntax, and translates the SQL query into an initial tree representation: specifically, a tree of logical operators representing the high-level steps required to execute the query in question. Initially, these logical operators will be closely related to the original syntax of the query, and will include such logical operations as “get data from the Customer table”, “get data from the Contact table”, “perform an inner join”, and so on. Different tree representations of the query will be used throughout the optimization process, and this logical tree will receive different names until it is finally used to initialize the Memo structure, as will be discussed later.

Binding is mostly concerned with name resolution. During the binding operation, SQL Server makes sure that all the object names do exist, and associates every table and column name on the parse tree with their corresponding object in the system catalog. The output of this second process is called an algebrized tree, which is then sent to the Query Optimizer.

The next step is the optimization process, which is basically the generation of candidate execution plans and the selection of the best of these plans according to their cost. As has already been mentioned, SQL Server uses a cost-based optimizer, and uses a cost estimation model to estimate the cost of each of the candidate plans.[4]

In essence, query optimization is the process of mapping the logical query operations expressed in the tree representation to physical operations, which can be carried out by the execution engine. So it's actually the functionality of the execution engine that is being implemented in the execution plans being created by the Query Optimizer, that is, the execution engine implements a certain number of different algorithms and it is from these algorithms that the Query Optimizer must choose, when formulating its execution plans. It does this by translating the original logical operations into the physical operations that the execution engine is capable of performing, and execution plans show both the logical and physical operations. Some logical operations, such as a Sort, translate to the same physical operation, whereas other logical operations map to several possible physical operations. For example, a logical join can be mapped to a Nested Loops Join, Merge Join, or Hash Join physical operator.

Thus, the end product of the query optimization process is an execution plan: a tree consisting of a number of physical operators, which contain the algorithms to be performed by the execution engine in order to obtain the desired results from the database.

Table No.1

|  |  |
| --- | --- |
| **The Run-Time Structures** | **The Static Analysis Structures** |
| The object store | The Metabase |
| Run-time ES | Static ES |
| Run-time QRES | Static QRES |

Query Evaluation:

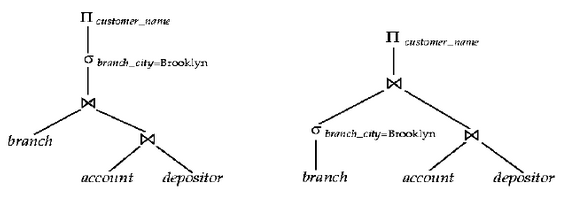
Alternative ways of evaluating a given query

* Equivalent expressions
* Different algorithm for each operation

Query Processing

Query is processed in three phases, as below

Parsing: DBMS parses the SQL query and chooses the most efficient access and execution plan.

Fig 3. Query Evaluation

Execution: The DBMS executes the SQL query using the chosen execution plan.

Fetching: The DBMS fetches the data and sends the result sets back to the client.

-The processing of DDL is different from DML

-For DDL, DBMS actually updates the data dictionary tables or system catalog while DML manipulates end user data.

In this section we will follow the definitions of metabase, static environment stack (S\_ES) and static query result stack (S\_QRES). We can shortly describe their meaning by such a simplistic assignment, which however gives general overview of used concepts:

From optimization perspective static environment stack and metabase are especially important. The former is responsible (among other things) for providing the optimizer with information about the relative position in the stack, where the name is binded. The latter comprises data about estimated size of searched data and maximal and minimal values of attributes. In our optimization techniques we are also using metabase to store some information like indices or static navigations.

Short overview of query optimization

There are many approaches to query optimization that can boost object DBMS performance. They can be divided into various subcategories. Main criterion that subdivides query optimizations is the time when they are applied to the query[9].

Static optimization

First group of optimizations is called "static", because operations are performed before the query is executed. Static optimizations can be done for instance by rewriting queries. During static analysis optimizer doesn't have access to run-time data like current statistics describing state of database. It also cannot take advantage of the knowledge about the state of environment stack. Instead optimizer can make use of their static counterparts - static environment stack and metabase. Thus we can subdivide static optimizations into two groups.

Optimizations that don't need metabase

* Factoring out independent subqueries
* Pushing selection
* Factoring out common path sub expressions
* Removing unnecessary auxiliary names
* Replacing a navigational join with a dot

Optimizations using metabase

* Textual indices
* Direct navigations
* Access support relations
* Removing dead subqueries

Dynamic optimization

Second group of optimizations consists of operations based on statistics about the state of environment during query evaluation. Such optimizations are also called cost-based, because they can evaluate the exact time of various execution plans. Dynamic optimizations generally are more powerful, but also have one important flaw. It is the constraint of the time of execution. The cost of choosing the best execution plan cannot be greater than the cost of executing naive and straightforward strategy. Author of query optimizer, which uses dynamic optimizations must take it into consideration and should rather make use of heuristics than trying to find optimal solution.

Object oriented store Model

Any approach of formalization of query language must be proceeding by formalization of data structure to be queried. To eliminate secondary features of data structure, assume an unification of record, tuples, array and all bulk structure.

An object store is formed of the structure of objects, starting point of query and constraints.

The object has three features.s

1. Internal identifier

2. External name

3. Contents

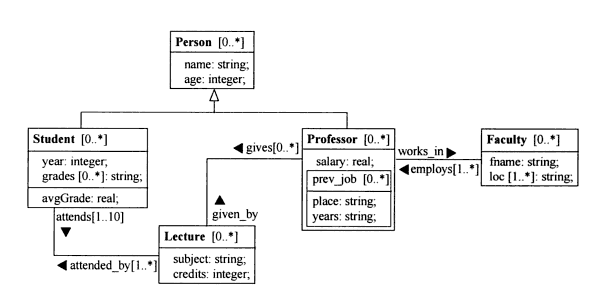
Our query never return objects, but some structures depend on references, values and names.

Fig 4: The Class diagram of the example database.

An important aspect of query optimization is a cost model. However, because removing dead subqueries always improves performance, it should be applied whenever possible. As a consequence, there is no need to assess performance improvement during optimization even for cost based optimization. Therefore, we do not consider a cost model.

The queries are defined for an example database whose schema (the class diagram in a little modified UML) is shown in fig . The classes Lecture, Student, Professor and faculty model lectures attended by students and given by professors working in faculties respectively. Professor object can contain multiple complex prev\_job sub objects (previous jobs). The name of class, attribute, etc is followed by its cardinality, unless it is 1. All the properties are public.[1]

# VI. CONCLUSION

The above propose system has both an object oriented data model deals with static query optimization & special optimization method concerning queries or we say subqueries. The reason is the fact that in the stack based approach to query language the semantics of all nonalgebraic operators in the construction of final result. More powerful variants of the method were received on the assumption concerning the distributive property of selection. Due to above reason we can develop extended version of query rewriting methods known from the relational model in particular pushing a selection before a join.

# REFERENCES

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

[2] 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] Venkata Krishna Suhas Nerella, Swetha Surapaneni, Sanjay Kumar Mardria, Thomos Weigert, Department of Computer Science, Missouri Univarsity of Science & Tech, Rolla, MO, 34 th Annual IEEE Computer Software & Applications Conference,2010.

[4] 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] R.Adamus, M.Daczkowski, P.Habela, K.Kaczmarski, T.Kowalski, M.Lentner, T.Pieciukiewicz, K.Stencel, K.Subieta, M.Trzaska, T.Wardziak, J.Wiślicki: 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.

[6] Semi-strong Type Checking in Database Programming Languages (in Polish), PJIIT - Publishing House, 2006, 207 pages.

[7]K.Subieta, Y.Kambayashi, J.Leszczylowski. Procedures in Object-Oriented Query Languages.Proc. 21-st VLDB Conf., Zurich, 1995, pp.182-193

[8] K.Subieta. Stack-Based Architecture (SBA) and Stack-Based Query Language(SBQL). http://www.sbql.pl/, 2010.

[9] Yannis E. Ioannidis. Query Optimization. Y. Ioannidis. Universality of serial histograms. In Proc. 19th Int. VLDB Conference,

pages 256{267, Dublin, Ireland, August 1993..