ABSTRACT

As the industry continues to expand its horizons, there is a greater need of handling humongous amount of data. Searching for information from terabytes of data is like finding a needle in a haystack unless the search is optimized. The queries applied of structured and unstructured data stored in databases combined with information retrieval techniques can lead to faster and efficient processing of data. When a database is queried, it generates results using one of the multiple available plans. The difference in each plan is the amount of time that is called cost it takes to reach to the answer. Selecting the best plan is of high importance since it determines the entire system performance. A query optimizer is a module present in a database that compares all the available plans and finds the one that has the least cost. The process of generating these query execution plans is nothing but query optimization. Thus, query optimization is mainly determining ways to search through all the query execution plans and comparing results of each plan to understand which is the best. Using information retrieval techniques for query optimization can thus generate the most efficient results in terms of cost.
INTRODUCTION

Query optimization is a continued topic of research in the fields of academics and commerce. The optimization in relational database has become an issue due to the flexibility given to the users by the modern database interfaces. It becomes more and more important to optimize and reformat a query given by the user to make it computationally cost effective and efficient since the interfaces facilitate users to provide complex queries. Query optimization can be defined as a process of transforming a query into an equivalent form that produces the same result as the original one for all database states which can be evaluated more efficiently. Conventionally optimization is mainly based on the syntactic knowledge of the operations and storage details of the relations. A query is a language expression describing data to be retrieved from a database. The query is represented in a content-based manner and then given to the query optimizer so that the optimizer is presented with multiple choices for evaluation of the most efficient way. Query optimization can be done either by maximizing the output for the given number of resources or by minimizing the resource usage for a given input. In most cases, response time for a given query language is tried to be minimized. This general goal allows several different operational objective functions. The response time goal is reasonable only under the assumption that user time is the most important bottleneck resource. Otherwise, direct cost minimization of technical resource usage can be attempted [4]. On a higher level, query optimization applies equivalence rules to rewrite the tree of operators evoked in a query and produce an optimal plan. The plan is considered to be an optimal one if provides the answer to the given query in the least time or by using the least space. For optimization purpose, there are specific equivalence rules developed that consider the syntax, logic, and semantics of the query. These rules internally
associate cost with every semantically equivalent plan to find out the one with the least cost. Cardinality, number of result tuples in the output of each operator, the cost of accessing a source and obtaining results from that source are the few things considered while calculating the cost of the query [5]. The complexity of producing an optimal, low-cost plan for a relational query is NP-complete. However, ongoing research work is focusing on developing reasonable heuristics to solve this problem. In this paper, we will discuss in depth the process of optimizing a user query and compare the different approaches of the optimization process.

SURVEY OF RELEVANT WORK

The optimizer attempts to generate the most cost-effective plan for any given SQL statement. For a specific query in each environment, the cost computation accounts for factors of query execution such as I/O, CPU, and communication. For example, a user is requesting for information of all managers in a company’s database. The optimizer will attempt a full table scan if the optimizer statistics show that around 70% employees are managers i.e., a majority. However, reading an index followed by a table access by row-id in case when there are only a few manager records in the database is more efficient. These decisions are based on the statistics gathered using the internal tools of the database available to the optimizer. Hence, an optimizer is better suited to take these decisions than the external user. An execution plan gives a detailed description of the steps taken to execute the user’s query. The following figure show how the optimizer generates two possible execution plans for an input SQL statement, uses statistics to estimate their costs, compares their costs, and then chooses the plan with the lowest cost [6].
The query optimizer of comprised of the three main components:

- **Query Transformer**

  Some queries when rewritten into a semantically different form may present better results in terms of cost. The query transformer is used to determine if a particular can be rewritten in a different form to be more advantageous. When a better alternative is available, the database calculates the cost of the alternatives separately and chooses the lowest-cost alternative.

- **Estimator**

  The overall cost of the query is determined by the estimator. This overall cost is calculated on the basis of selectivity, cardinality and cost. Selectivity is the percentage of rows in the row set that the query selects, with 0 meaning no rows and 1 meaning all rows [6]. Cardinality is nothing but the number of rows returned by each operation in an execution
plan. Cost represents units of work or resource used. The query optimizer uses disk I/O, CPU usage, and memory usage as units of work.

- Plan generator

This component explores various plans for a query block by trying out different access paths, join methods, and join orders. Many plans are possible because of the various combinations that the database can use to produce the same result. The optimizer picks the plan with the lowest cost [6].

The following figure gives the overall components described above

![Diagram](image)

The query optimization process can be divided logically into 2 stages, rewriting stage and a planning stage.
• **Rewriting stage**

In this stage, the rewriter module mainly uses the transformer and component mentioned above. The main task of this module is to apply transformations to the user’s query and produce all different possible queries that are more efficient. Some of the common processes followed by the rewriting stage are standardization of the query form, replacement of views by their definition and flattening out of nested queries [1]. All these transformations do not consider the actual cost required for the query. They only consider the static characteristics of the given query and perform transformations.

• **Planning stage**

There are multiple execution plans developed internally using Algebraic Space and the Method-Structure Space modules for a given query. In this stage, all these execution plans are explored for ordering them based on their cost. All the executions plans are compared to each other based on estimates of their cost derived by the Cost Model and the Size-Distribution Estimator modules. On comparison, the plan having the least cost is finally used to generate results for the given query. Different search strategies are used in this stage for exploring the possible plans. One of the search strategies mentioned in [1] uses the dynamic programming approaches. This strategy iterates the relations joined so far to generate the execution plans and simultaneously keeps pruning the sub-optimal plans. The time and space complexity goes on increasing exponentially as the query size increases, in the worst-case scenario. This approach is effective in case when there are less than 10 joins in a single query. Larger queries can be optimized using different approaches.
Types of query optimization

The different execution plans are compared by the optimizer based on a single cost metric which in most cases is the execution time. However, there are can more things to consider which comprise of the total cost. This leads to different types of query optimization listed in multiple papers that can be clubbed into two main categories:

- **Parametric query optimization**

  In this type, the cost of the query depends on parameters whose values are not known at the time of optimization. Examples can be the select predicates that are fully provided at the execution time but won’t be available during the optimization time. Parametric query optimization therefore associates each query plan with a cost function that maps from a multi-dimensional parameter space to a one-dimensional cost space [7].

- **Multi-objective query optimization**

  This type comes in picture when other metrics along with the execution time are used to compare the execution plans. The cost of a query is modelled as a cost vector and each vector component represents cost given by the different cost metrics. In scenarios where cloud platforms are used, the cost vectors can be the execution time and amount of money required for execution. Both these can together give the overall cost of the query. Here the goal is not only to minimize all the cost metrics but to find an execution plan that realizes the best compromise between different cost metrics [7].
DIFFERENT APPROACHES OF QUERY OPTIMIZATION

- **Index Scans**

  To access data using an index, either a scan is performed or a seek operation. The scan operation searches for the entire index to return the results. A table with 1 million rows will be traversed entirely to produce the results. The seek operation filters rows based on the selection criteria. A seek of a table uses index’s binary tree to retrieve the required data and hence, does not scan the entire table. The choice of using scan vs seek depends on the size of the data that needs to be returned. More the size of the result set better are the results using the index.

- **Functions wrapped around WHERE clauses**

  Cost of a query mainly comprises of the IO operations needed to be performed the fetch the result set. Lesser the IO operations, lesser is the query cost. In queries with where clauses, the conditions of the where clause are resolved first. If there is a use of functions like SUBSTRING or CONVERT around a particular column in the given query, then these functions also need to be resolved. The entire table needs to be scanned if this function needs to be evolved prior to execution to determine a result set. This increases the cost and hence, needs to be avoided. Thus, in situation where functions are wrapped around the where clause, the query can be transformed into a form that uses a scalar variable such as the LIKE operator.

- **Implicit Conversations**

  All data types in any SQL management systems are assigned a precedence. The database internally reconciles the data types when values are to be compared. Data types with
lower precedence are automatically converted to higher precedence data types. Conversion between some data types occurs without any performance impact. However, certain data types are completely different from each other and cannot be converted automatically. The database converts are the values of the table prior to applying any filter. This can again be avoided by the usage of scalar variables.

CONCLUSION

The paper presents a detailed analysis of performing query optimization. Different approaches that can be followed for efficient query optimization are listed above with their pros and cons. There are still many questions regarding to the query optimization components that are not been answered. Query optimization for parallel, distributed, semantic, global, parametric, dynamic, nested, rule-based, object-oriented, heterogeneous, recursive systems is an open topic of research [1]. Optimization has an immediate impact on the overall system performance. Inefficiency in the system while executing users’ queries directly impacts the integrity of the system and reduces the overall user satisfaction. Hence, query optimization remains an important and exciting field for research.

REFERENCES


[6] https://docs.oracle.com/database/121/TGSQL/tgsql_optcncpt.htm#TGSQL196