International Journal of Modern Trends in Engineering and Research
www.ijmter.com
e-ISSN: 2349-9745
p-ISSN: 2393-8161

An Analysis on Query Optimization in Distributed Database

Joshi Janki[1]
[1]*R&D Department, Infitrix Software, Delhi*

Abstract: The query optimizer is a significant element in today's relational database management system. This element is responsible for translating a user-submitted query commonly written in a non-procedural language-into an efficient query evaluation program that can be executed against the database. This research paper describes architecture steps of query process and optimization time and memory usage. Key goal of this paper is to understand the basic query optimization process and its architecture.

Keywords –Query Optimization, Distributed Database System, Query Processing

I. INTRODUCTION

Query optimization is a function of much relational database management system. Generally, the query optimizer cannot be accessed directly by users, once queries are submitted to database server, and parsed by the parser, they are then passed to the query optimizer where optimization take place. Queries results are generated by accessing relevant database data and manipulating it in a way that yields the requested information.[1] Since database structures are complex, in most cases, and especially for not-very-simple queries, the needed data for a query can be collected from a database by accessing it in different ways, through different data-structures, and in different orders. It determines the lowest cost plan for executing queries. By "lowest cost plan," we mean an access path to the data that takes the least amount of time.[2]

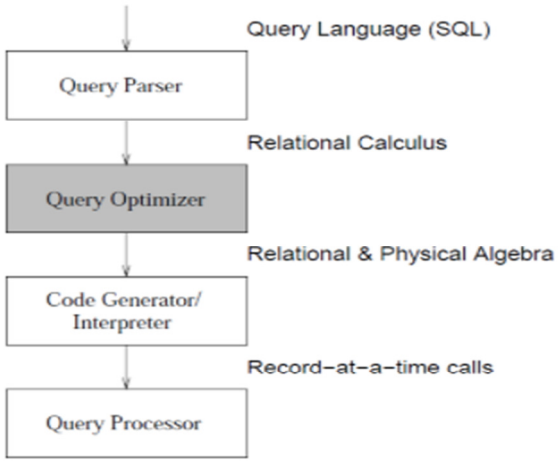

Figure1: Query Optimization Concept Through above figure

e-ISSN: 2349-9745
p-ISSN: 2393-8161

The description of the above figure is as below:

The Query Parser checks the validity of the query and then translates it into an internal Form usually a relational calculus expression or something equivalent. The Query Optimizer examines all algebraic expressions that are equivalent to the given query and chooses the one that is estimated to be the cheapest. The Code Generator or the Interpreter transforms the access plan generated by the optimizer into calls to the query processor. The Query Processor actually executes the query.[3]

Queries are posed to a DBMS by interactive users or by programs written in general-purpose programming languages (e.g., Fortran, PL-1) that have queries embedded in them. An interactive (ad hoc) query goes through the entire path shown in Figure 1. On the other hand, an embedded query goes through the three steps only once, when the program in which it is embedded is compiled. The code produced by the Code Generator is stored in the database and is simply invoked and executed by the Query Processor whenever control reaches that query during the program execution (run time). Thus, independent of the number of times an embedded query needs to be executed, optimization is not repeated until database updates make the access plan invalid (e.g., index deletion) or highly suboptimal (e.g., extensive changes in database contents).[5]

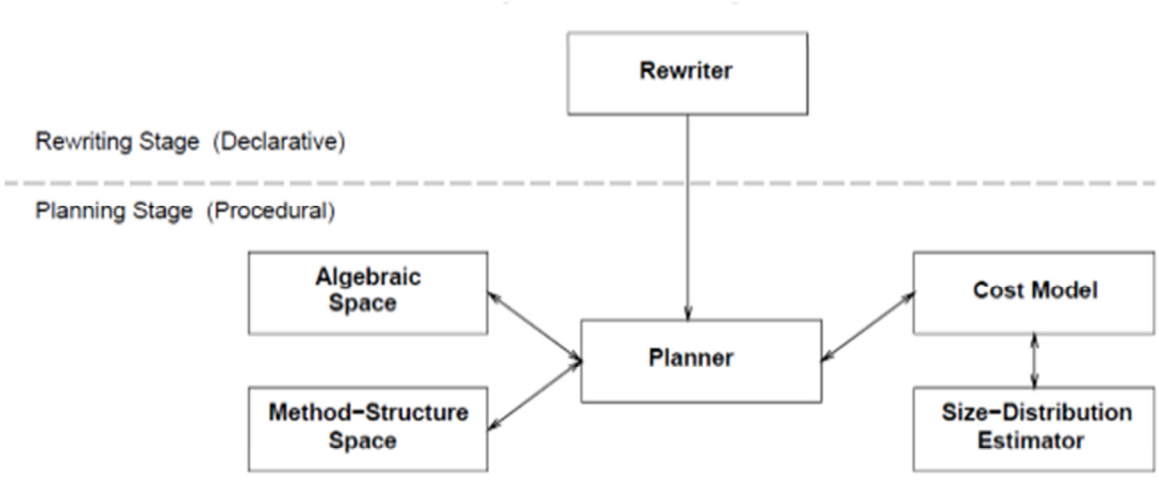

Figure2: Query Optimizer Architecture [7]

The entire query optimization process can be seen as having two stages: rewriting and planning. There is only one module in the first stage, the Rewriter, whereas all other modules are in the second stage.

e-ISSN: 2349-9745
p-ISSN: 2393-8161

II. Functionality of Query Optimizer Architecture

Rewriter: Module applies transformations to a given query and produces equivalent queries that are hopefully more efficient, e.g., replacement of views with their definition, attending out of nested queries, etc. The transformations performed by the Rewriter depend only on the declarative i.e., static, characteristics of queries and do not take into account the actual query costs for the specific DBMS and database concerned. If the rewriting is known or assumed to always be beneficial, the original query is discarded; otherwise, it is sent to the next stage as well. By the nature of the rewriting transformations, this stage operates at the declarative level.[7]

Planner: This is the main module of the ordering stage. It examines all possible execution plans for each query produced in the previous stage and selects the overall cheapest one to be used to generate the answer of the original query. It employs a search strategy, which examines the space of execution plans in a particular fashion. This space is determined by two other modules of the optimizer, the Algebraic Space and the Method-Structure Space. For the most part, these two modules and the search strategy determine the cost, i.e., running time, of the optimizer itself, which should be as low as possible. The execution plans examined by the Planner are compared based on estimates of their cost so that the cheapest may be chosen. These costs are derived by the last two modules of the optimizer, the Cost Model and the Size-Distribution Estimator.[7]

Method-Structure Space: This module determines the implementation choices that exist for the execution of each ordered series of actions specified by the Algebraic Space. This choice is related to the available join methods for each join (e.g., nested loops, merge scan, and hash join), if supporting data structures are built on the y, if/when duplicates are eliminated, and other implementation characteristics of this sort, which are predetermined by the DBMS implementation. This choice is also related to the available indices for accessing each relation, which is determined by the physical schema of each database stored in its catalogs. Given an algebraic formula or tree from the Algebraic Space, this module produces all corresponding complete execution plans, which specify the implementation of each algebraic operator and the use of any indices.[7]

Cost Model: This module specifies the arithmetic formulas that are used to estimate the cost of execution plans. For every different join method, for every different index type access, and in general for every distinct kind of step that can be found in an execution plan, there is a formula that gives its cost. Given the complexity of many of these steps, most of these formulas are simple approximations of what the system actually does and are based on certain assumptions regarding issues like buffer management, disk-cpu overlap, sequential vs. random I/O, etc. The most important input parameters to a formula are the size of the buffer pool used by the corresponding step, the sizes of relations or indices accessed, and possibly various distributions

e-ISSN: 2349-9745
p-ISSN: 2393-8161

of values in these relations. While the first one is determined by the DBMS for each query, the other two are estimated by the Size-Distribution Estimator.[7]

Size-Distribution Estimator: This module specifies how the sizes (and possibly frequency distributions of attribute values) of database relations and indices as well as (sub)query results are estimated. As mentioned above, these estimates are needed by the Cost Model. The specific estimation approach adopted in this module also determines the form of statistics that need to be maintained in the catalogs of each database, if any.[7]

Algebraic Space: This module determines the action execution orders that are to be considered by the Planner for each query sent to it. All such series of actions produce the same query answer, but usually differ in performance. They are usually represented in relational algebra as formulas or in tree form. Because of the algorithmic nature of the objects generated by this module and sent to the Planner, the overall planning stage is characterized as operating at the procedural level.[7]

III. Examples of Optimization Time and Memory

- To find item price and customer name from two tables Orders and Customers Original:

Original:
Select O.ItemPrice, C.Name
From Orders O, Customers C

Corrected:
Select O.ItemPrice, C.Name
From Orders O, Customers C
Where O.CustomerID = C.CustomerID

In the first example we can see two query one is original and second one is corrected query. Here join query was not used and also not used for all the keys, so that it would return so many records and it's takes a hours to find result.[4]

- To find out employees salary based on their ID

Original:
For i = 1 to 20000
Select salary From Employees Where EmpID = Parameter(i)

Corrected:
Select salary From Employees Where EmpID >= 1 and EmpID <= 20000

e-ISSN: 2349-9745
p-ISSN: 2393-8161

The original Query involves a lot of time and memory consumption and will make your entire system slow.[8]

VI. CONCLUSION

The paper gives brief concept of query optimization along with its architecture and module functionality. It also describes working of its query flow methods (step by step) execution. With the help of example it shows optimization time and memory based on record extraction.

References

[1]M. M. Astrahan et al. System R: A relational approach to data management. ACM Transactions on Database Systems, 1(2):97{137, June 1976.
[2] G. Antoshenkov. Dynamic query optimization in Rdb/VMS. In Proc. IEEE Int. Conference on Data engineering, pages 538{547, Vienna, Austria, March 1993.
[3] K. Bennett, M. C. Ferris, and Y. Ioannidis. A genetic algorithm for database query optimization. In Proc. 4th Int. Conference on Genetic Algorithms, pages 400{407, San Diego, CA, July 1991.
[4] P. A. Bernstein, N. Goodman, E. Wong, C. L. Reeve, and J. B. Rothnie. Queryprocessing in a system for distributed databases (SDD-1). ACM TODS, 6(4):602{625,December 1981.
[5]R. Cole and G. Graefe. Optimization of dynamic query evaluation plans. In Proc.ACM-SIGMOD Conference on the Management of Data, pages 150{160, Minneapolis,MN, June 1994.
[6] S. Christodoulakis. Implications of certain assumptions in database performance evaluation. uation. ACM TODS, 9(2):163{186, June 1984.
[7]S. Christodoulakis. On the estimation and use of selectivities in database performance evaluation. Research Report CS-89-24, Dept. of Computer Science, University ofWa-terloo, June 1989.
[8]http://www.serverwatch.com/tutorials.

