International Journal of Technology Research and Management
ISSN (Online): 2348-9006
Vol 1 Issue 1 March 2014

Framework for Query optimization

Pawan Meena
Department of Computer Science and Engineering Patel college of science & Technology
Bhopal,M.P,INDIA
pawanmeena75@yahoo.com

ABSTRACT:- *Modern database systems use a query optimizer to identify the most efficient strategy, called "plan", to execute declarative SQL queries. Optimization is much more than transformations and query equivalence. The infrastructure for optimization is significant. Designing effective and correct SQL transformations is hard. Optimization is a mandatory exercise since the difference between the cost of the best plan and a random choice could be in orders of magnitude. The role of query optimizers is especially critical for the decision-support queries featured in data warehousing and data mining applications. This paper presented an abstraction of the architecture of a query optimizer and focused on the techniques currently used by most commercial systems for its various modules. In aaddition, provide technical constraint of advanced issues in query optimization.*

Keywords:- *Query optimizer ,Operator tree, Query analyzer, Query optimization*

1. INTRODUCTION

For significantly improve application development and user productivity, relational database technology growing success in the treatment of data is appropriate in part to the availability of non-procedural languages. By hiding the low-level details about the physical organization of the data, relational database languages allow the expression of complex queries in a concise and simple fashion. In particular, to build the answer to the query, the user does not exactly specify the procedure. This procedure is in fact designed by a DBMS module, known as query processor. This relieves the user to query optimization, a tedious task that is managed correctly by the query processor. Modern databases can provide tools for the effective treatment of large amounts of complex scientific data involving the application of specific analysis [1, 2]. Scientific analysis can be specified as high-level requests user-defined functions (UDFs) in an extensible DBMS. The query optimization provides scalability and high performance without the need for researchers to spend time on low-level programming. Moreover, as the queries are specified and easily changed, new theories, for example implemented as filters, can be tested quickly.

Queries about events are complex, because the cuts are complex with many predicates applied to the properties of each event. The conditions of the query involving selections, arithmetic operators, aggregates, UDF, and joins. The aggregates compute complex derived event properties. For example, a complex query is to look for event production Higgs bosons [1, 3] by applying scientific theories expressed cuts. These complex queries need to be optimized for the efficient and scalable. However, the optimization of complex queries is a challenge because:

• The queries contain many joins.

• The size of the queries makes optimization slow.

• The cut definitions contain many more or less complex aggregates.

• The filters defining the cuts use many numerical UDFs.

• There are dependencies between event properties that are difficult to find or model.

• The UDFs cause dependencies between query variables.

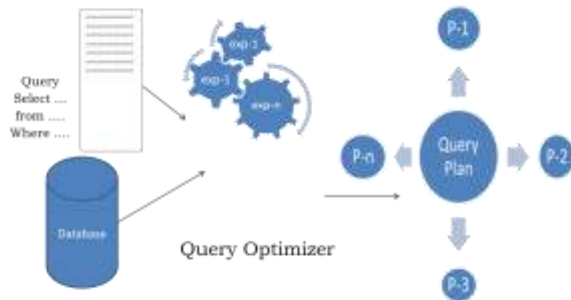

Figure 1: Query Optimizer

Relational query languages provide a high level "declarative" interface to access data stored in relational databases. Over time, SQL [1,4] has emerged as the standard for relational query languages. Two key elements of the component of the evaluation of a system for querying SQL databases are the query optimizer and execution engine queries. The query execution engine implements a set of physical operators. An operator takes as input one or more data streams and produces an output data stream. Examples of operators are physical (external) sorting, sequential analysis, index analysis, nested loop join and sort-merge join. We refer to operators such as physical operators since they are not necessarily related one by one with the relational operators. The easiest way to think of physical operators is like pieces of code that are used as building blocks to enable the execution of SQL queries. An abstract representation of such a performance is a physical operator tree, as shown in

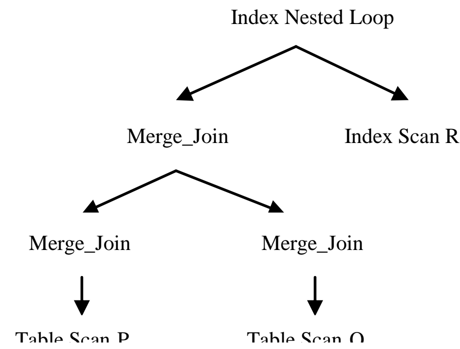

Figure 2: Physical Operator Tree

Figure 2. The edges in an operator tree represent the flow of data between the physical operators.

We use the terms physical operator tree and execution plan (or simply plan) interchangeably. The execution engine is responsible for implementing the plan resulting generate responses to the request. Therefore, the Capabilities of the query execution engine to determine the structure of the operator trees that are practicable. We refer the reader to [5] for an overview of the technical evaluation of the query. The query optimizer is responsible for producing the input for the execution engine. It takes a parsed representation of an SQL query as input and is responsible for producing an efficient execution plan for the given SQL query in the space of possible execution plans. The task of an optimizer is nontrivial since for a given SQL query, there may be many operator trees possible:

• The algebraic representation of the data query can be transformed into many other logically equivalent algebraic representations: for example,

Join (Join (P, Q), R) = Join (Join (Q, R), P)

• For a given algebra representation, there can be many operator trees that the operator algebraic expression to perform, for example, in general, there are several algorithms supported them in a system database. In addition, the current or the response time for the implementation of these plans is very different. Therefore, a choice of execution by the optimization program is crucial. For instance, query

optimizations are regarded as difficult search. To solve this problem, we need:

• A space of plans (search space).

• A cost estimation technique so that a cost may be assigned to each plan in the search space. Intuitively, this is an estimation of the resources needed for the execution of the plan.

• An enumeration algorithm that can search through the execution space A desirable optimizer is one where the search space includes plans to lower costs, the costing technique is correct and the enumeration algorithm eff- icient. Each of these tasks is nontrivial and that is why building a good optimizer is a huge undertaking.

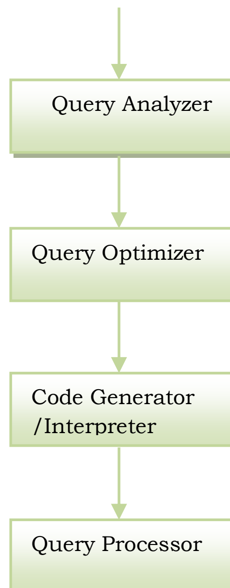

Figure 3: Query traverses through DBMS

The path through a query to a DBMS is generated by its reaction is shown in Figure 3.The modules of the system, allowing it to move the following functions.

The Query Analyzer checks the validity of the query; it creates an internal form, usually an expression of the relational calculus or something similar. The query optimizer considers all algebraic expressions that are equivalent to the given query and choose one that is estimated to be less expensive. The code generator or interpreter changes the map generated by the optimizer calls the query processor.

2. QUERY OPTIMIZATION ARCHITECTURE

In this section, we provide an abstraction of the query optimization process in a DBMS. Given a database and a query on it, several execution plans exist that can be employed to answer the query. In principle, all the alternatives need to be considered so that the one with the best estimated performance is chosen. An abstraction of the process of generating and testing these alternatives is shown in Figure 4, which is essentially a modular architecture of a query optimizer. Although one could build an optimizer based on this architecture, in real systems, the modules shown do not always have so clear-cut boundaries as in Figure 4. Based on Figure 4, the entire query optimization process can be seen as having two stages: rewriting and planning [6]. There is only one module in the first stage, the Rewriter, whereas all other modules are in the second stage. The functionality of each of the modules in Figure 4 is analyzed below

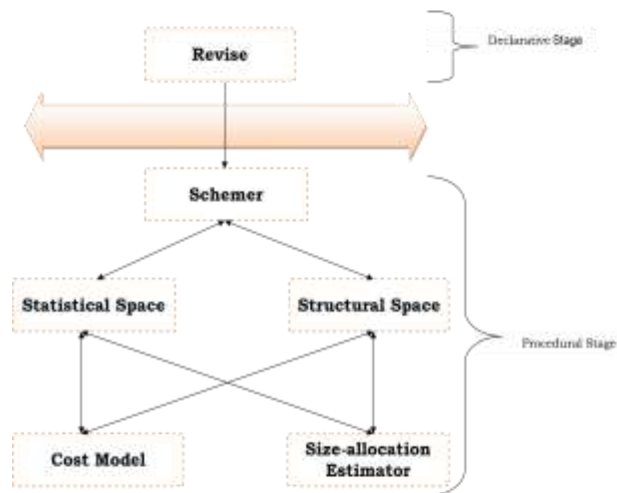

Figure 4: Query optimizer architecture

2.1 Revise

This module applies transformations to a given query and produces similar questions that are hopefully more effective, for example, replacement of thought with their definition, to attend nested queries, etc. The processing is done by the author only on the declarative, that is, static the characteristics of requests and do not take into account the actual

cost for the specific question DBMS and the database in question. If rewriting is known or assumed always positive, the initial request is ignored, otherwise sent to the next as well. The nature of the transformations to rewrite this step occurs in declarative level [6].

2.2 Schemer

This is the main module of the ordering stage. Examine all possible execution plans for each query generated in the previous step and selects the best global market to be used for the reaction to generate the original query. It employs a research strategy that examines the space of execution plans in a particular fashion. This is determined by two other modules of the optimizer, space and space-mode algebraic structure. Most of these modules and the search strategy to the cost, i.e., work time, the optimizer itself, which should be as low as possible to determine. The implementations of the plans reviewed by the planner are compared in terms of their cost estimates so that the cheapest may be chosen. These costs are calculated by the last two modules of the optimizer, the cost model and the estimator-Size allocation.

2.3 Statistical 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.

2.4 Structural Space

This module determines the choice of performance that exists for the execution of each set of actions ordered by the field of statistics. This choice is related to the join methods are available for each joint (eg, nested loop, scan and hash them together), as supporting data structures are built on them if / when duplicates are eliminated, and the characteristics of other implementation of this kind, which are determined by the performance of the DBMS. This choice is also linked to evidence any relationship, which is determined by the physical schema of each database stored in its catalog entry Given a Statistical formula or tree from the Statistical Space, this module produces all corresponding complete execution plans, which specify the implementation of each algebraic operator and the use of any indices [6].

2.5 Cost Model

This module specify the mathematical formulas that are used to approximate the cost of execution plans. For every different join method, for every different index type access, and in general for every different 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 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- allocation Estimator.

2.6 Size- Allocation 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 [6]

3. ADVANCED TYPES OF OPTIMIZATION

In this section, we attempt to provide a concise sight of advanced types of optimization that researchers have proposed over the past few years. The descriptions are based on examples only; further details may be found in the references provided. Furthermore, there are several issues that are not discussed at all due to lack of space, although much

interesting work has been done on them, e.g., nested query optimization, rule-based query optimization, query optimizer generators ,object-oriented query optimization, optimization with materialized views, heterogeneous query optimization, recursive query optimization, aggregate query optimization, optimization with expensive selection predicates, and query optimizer validation. Before presenting specific technique consider the following simple relation EMP (empid ,salary, job, department, dno) , DEPT(dno, budget,)

3.1 Semantic Query Optimization

Semantic query optimization is a form of optimization mostly related to the Rewriter module. The basic idea lies in using integrity constraints defined in the database to rewrite a given query into semantically equivalent ones [7]. These can then be optimized by the Planner as regular queries and the most efficient plan among all can be used to answer the original query. As a simple example, using a hypothetical SQL-like syntax, consider the following integrity constraint:

assert sal-constraint **on** emp:

salary>200K **where** job = “Assistant professor"

In addition consider the following query:

select empid, subject

from emp, dept

where emp.dno = dept.dno and job = “Assistant professor".

Using the above integrity constraint, the query can be rewritten into a semantically equivalent one to include a selection on sal:

select empid, subject

from emp, dept

where emp.dno = dept.dno and job = “Assistant professor" and salary>200K.

Having the extra selection could help extremely in discovery a fast plan to answer the query if the only index in the database is a B+-tree on emp.sal. On the other hand, it would certainly be a waste if no such index exists. For such reasons, all proposals for semantic query optimization present various heuristics or rules on which rewritings have the potential of being beneficial and should be applied and which not.

3.2 Global Query Optimization

So far, we have focused our attention to optimizing individual queries. Quite often, however, multiple queries become available for optimization at the same time, e.g., queries with unions, queries from multiple concurrent users, queries embedded in a single program, or queries in a deductive system. Instead of optimizing each query separately, one may be able to obtain a global plan that, although possibly suboptimal for each individual query, is optimal for the execution of all of them as a group. Several techniques have been proposed for global query optimization [8].

As a simple example of the problem of global optimization consider the following two queries:

select empid, subject

from emp, dept

where emp.dno = dept.dno and job = “Assistant professor ",

select empid

from emp, dept

where emp.dno = dept.dno and budget > 1M

Depending on the sizes of the emp and dept relations and the selectivity’s of the selections, it may well be that computing the entire join once and then applying separately the two selections to obtain the results of the two queries is more efficient than doing the join twice, each time taking into account the corresponding selection. Developing Planner modules that would examine all the available global plans and identify the optimal one is the goal of global/multiple query optimizers.

3.4 Parametric Query Optimization

As mentioned earlier, embedded queries are typically optimized once at compile time and are executed multiple times at run time. Because of this temporal separation between optimization and execution, the values of various parameters that are used during optimization may be very different during execution. This may make the chosen plan invalid (e.g., if indices used in the plan are no longer available) or simply not optimal (e.g., if the number of available buffer pages or operator selectivity’s have changed, or if new indices have become available). To address this issue, 31several techniques [9,10,11] have been

proposed that use various search strategies (e.g., randomized algorithms [10] or the strategy of Volcano [11]) to optimize queries as much as possible at compile time taking into account all possible values that interesting parameters may have at run time. These techniques use the actual parameter values at run time, and simply pick the plan that was found optimal for them with little or no overhead. Of a drastically different flavor is the technique of Rdb/VMS [12], where by dynamically monitoring how the probability distribution of plan costs changes, plan switching may actually occur during query execution**.**

4. CONCLUSION

To a large extent, the success of a DBMS lies in the quality, functionality, and sophistication of its query optimizer, since that determines much of the system's performance. In this paper, we have given a bird's eye view of query optimization. We have presented an abstraction of the architecture of a query optimizer and focused on the techniques currently used by most commercial systems for its various modules. In addition, we have provided a glimpse of advanced issues in query optimization, whose solutions have not yet found their way into practical systems, but could certainly do so in the future.

REFERENCES

[1] J. Gray, D.T. Liu, M.A. Nieto-Santisteban, A. Szalay, D.J. DeWitt, and G. Heber, "Scientific data management in the coming decade", SIGMOD Record 34(4), pp. 34-41, 2005.

[2] Ruslan Fomkin and Tore Risch 1997 "Cost-based Optimization of Complex Scientific Queries", Department of Information Technology, Uppsala University

[3] C. Hansen, N. Gollub, K.Assamagan, and T. Ekelöf, "Discovery potential for a charged Higgs boson decaying in the chargino-neutralino channel of the ATLAS detector at the LHC", Eur.Phys.J. C44S2, pp. 1-9, 2005.

[4] Melton, J., Simon A. Understanding The New SQL: A Complete

[5] Graefe G. Query Evaluation Techniques for Large Databases. In ACM Computing Surveys: Vol 25, No 2., June 1993.

[6] Yannis E. Ioannidis," Query optimization" Computer Sciences Department,University of Wisconsin Madison, WI 53706

[7] J. J. King. Quits: A system for semantic query optimization in relational databases. In Proc. of the 7th Int. VLDB Conference , pages 510{517, Cannes, France, August 1981.

[8] T. Cells. Multiple query optimization. ACM-TODS, 13(1):23{52, March 1988.

[9] G. Graefe and K. Ward. Dynamic query evaluation plans. In Proc. ACM-SIGMOD Conference on the Management of Data, pages 358-366, Portland, OR, May 1989.

[10] Y. Ioannidis, RNg, K. Shim, and T. K. Sellis. Parametric query optimization. In Proc. 18th Int. VLDB Conference, pages 103{114, Vancouver, BC, August 1992.

[11] 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.

[12] G. Antoshenkov. Dynamic query optimization in Rdb/VMS. In Proc. IEEE Int. Coference on Data Engineering, pages 538{547, Vienna, Austria, March 1993.