Query Optimization Based on Heuristic Rules

Vishal Hatmode
Department of Information Technology, Siddhant
College of Engineering, Pune, India.

Professor Sonali Rangdale
Department of Information Technology, Siddhant
College of Engineering, Pune, India.

Abstract: **In this paper, we will Rules based on Heuristic approach for query optimization. It is often observed in many database industries that a lot of time is spent in executing inefficient SQL queries. The problem here is that although the DBA knows that the query red are inefficient, the large sections of people who are actually running these queries are unable to write efficient queries. As a result, the performance of the entire system affects because of the major fall in the system performance i.e. the number of transactions performed per unit time is reduced. Typically, to overcome this problem, most of the people frequently consult the DBA for writing efficient and optimized queries. This approach results in a lot of time and money loss. A better solution is using a Query Optimizer Tool. A Query Optimizer will accept the user query and automatically generate an equivalent but highly optimized and effective query. This will save a lot of time, cost and effort. This in turn improves the system performance and its overall throughput capability. The Query Optimizer in this paper is based on Heuristic Optimizer. It tries to minimize the number of accesses by reducing the number of tuples and number of columns to be select operation.**

Keywords: Heuristic Approach, Query Optimization, tuples

I. INTRODUCTION

The main function of many relational database management systems is query optimization in which multiple query plans are to be prepared for satisfying a query are examined, and based on result of examined queries good query plan is identified. This may or may not be the absolute best of all strategies because there are many ways of creating plans. There is a trade-off between the amount of time spent figuring out the best plan and the amount running the plan. Different database management systems have different quality and ways of balancing these two. The main performance advantage is achieved in modern database system is by means of query optimizers. [1] Given a request for data retrieval; an Optimizer will select an optimal plan for evaluating with the given request from among the manifold different strategies. The largest and most complex module of database systems is query optimizers and it’s been proved its difficulty to modify and extend to accommodate these areas. Query optimization is very large area within the database field. It’s been studied in a great variety of contexts and from many divergent angles, giving rise to several different solutions in each case. Over time, SQL has become as the standard for relational query languages. Query optimization and query execution are the two key components for query evaluation of an SQL database system [1][6].Heuristic Optimization is less expensive than that of cost based optimization. It is based on some heuristic rules by which optimizer can decide optimized query execution plan [6].

Backround Need

Speed of execution is main factor in huge databases of biological, physical or chemical projects. Optimization needed for saving energy and resources [2]. The query optimizer is one of the important components in today’s database management systems. This component is responsible for transform user submitted queries usually written in non procedural language into efficient query evaluation program that can be executed against database. In this paper, we will gain a feel for how query processing works in a database management system, specifically focusing on a simple Select-Project-Join query engine. We will also see how various query execution plans have different performance results, which will provide some motivation for query optimization techniques [5].

Today all corporations, companies and organizations like banks maintain their own databases. These databases are used for holding many different kinds of information like customer details, employee details, and so on and so forth. The size of these databases is too huge i.e. containing large amount of records. Retrieval of information is done by querying the database using SQL. Now if the query used is an inefficient one then it consumes a lot of time before producing the result [6]. This is because an inefficient query requires a lot of memory operations i.e. the number of I/O operations is very high. Since in any computer system the main factor that hampers high speed performance is the I/O operation, an inefficient query is very slow in performance which consumes lot of time. Thus an inefficient query reduces the system performance by reducing the throughput greatly. Hence being able to write an efficient query is a important. An efficient query reduces the number of disk operations and hence reduces the number of I/O operations disk access rate. As a result an efficient query produces the results much more quickly than its inefficient counterpart. Now obviously the question that arises is how to write such an efficient query? This is where Query Optimizer steps in. This allows you to write your own queries and will in turn generate an equivalent but highly and efficient optimized query. Thus the Query Optimizer helps in enhancing the system performance by increasing the throughput.

II. LITERATURE SURVEY

Different Query Optimization Approaches

Cost Based Optimization

A cost-based query optimizer works as follows: First, it generates all possible query execution plans. Next, the cost of each plan is estimated. Finally, based on the estimation, the plan with the lowest estimated cost is chosen. Since the decision is made using estimated cost values, the plan chosen may not be optimal. [1] The quality of optimizer decisions depends on the complexity and accuracy of cost functions used. It includes different techniques such as the use of dynamic programming for deciding best plan. Their main disadvantage is that it is very costly. As a result most of the optimizers do not employ this strategy [2] [3]

1. Generates all possible query execution plans and then based on it cost is calculated.
2. Quality depends on complexity and accuracy of the cost Function.

Algebraic Expressions for following query-
SELECT p.pname, d.dname FROM Patients p, Doctors d Where p.doctor = d.dname AND d.gender ='M'

Advantages:
1. Rather than considering with the time constraints, it adapts to client requirements.
2. Speed of query retrieval increase

Disadvantages:
1. Uses cost based optimization hence expensive.

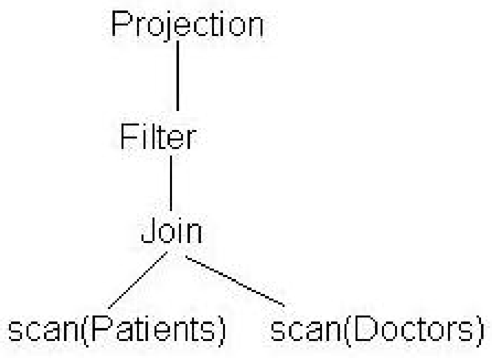

Figure 1: Relational Algebra Expression for Query

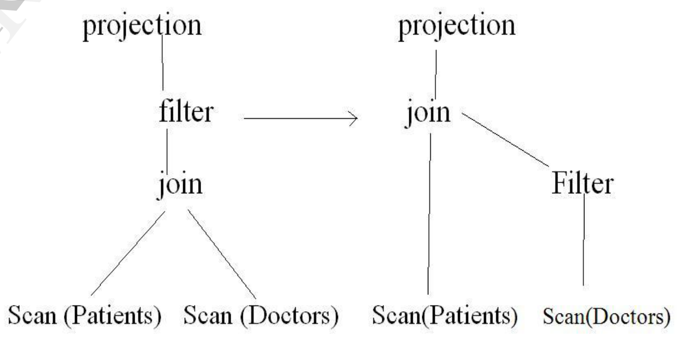

Figure 2: Execution Plan

Semantic Query Optimization

Two queries can be called as a semantically equivalent if they return the same answer for a database. For this purpose, it uses integrity constraints to match results. Semantic query optimization is known as process of determining the set of semantic conversion that result in a semantically equivalent query with a low execution cost. ODB-Optimizer defines more specialized classes to be accessed and reduces the number of factors by applying different integrity constraint rules. [2][4]

Advantages
1. Supports recursive queries and queries having negation, disjunction.
Disadvantages:
1. Suitable for only simple prototypes.
2. No commercial implementations exist.

III.PROPOSED ARCHITECTURE

Proposed System

The Query Optimizer in this project is a Heuristic Optimiser. It tries to minimize the number of accesses by reducing the number of tuples and number of columns to be searched. Heuristic Optimization is less expensive than that of cost based optimization. It is based on some heuristic rules by which optimiser can decide optimized query execution plan. [1][3][6] Important Heuristic Rules used are:

1. Performing selection as early as possible.
2. Perform projections as early as possible.

It's been found that Cost-based optimization is more expensive, even with dynamic programming. Systems can use heuristics to decrease the number of choices that have to be made in a cost-based fashion. Heuristic optimization transforms the query into query-tree by using a set of rules that (but not in all cases) improves execution performance [2][6].
1. Perform selection early (reduces the number of tuples)
2. Perform projection early (reduces the number of attributes)
3. Performing most restrictive selection and join operations (i.e. with smaller result size) before other equivalent operations. [4][6]

Few systems use only heuristics; while others combine both heuristics with partial cost-based optimization.

Example of two rules
Perform selection as early as possible.
Original Query: Select * from branch, customer where branch.name = 'Brooklyn' and customer. City = 'Brooklyn';
Transformed Query: Select * from (select * from branch where branch.name = 'Brooklyn'), (select * from customer where customer. City= 'Brooklyn');
Performance enhancement: Suppose there are branch and customer tables each has 100 and 100 tuples respectively.

Original query: 100 * 100 tuples fetched
Optimized Query: Selection performed early hence say only 10 and 20 tuples selected so 10*20 tuples fetched.

Perform projection as early as possible**:**
Original Query: Select branch.id, customer.cid from branch, customer where branch.name='Brooklyn';

Optimized Query: Select * from (select branch.id from branch) t, (select customer.cid from customer) where t.name='Brooklyn';

Performance Enhancement:
1) Projection operations reduce size of relations.
2) Reduces the number of columns in relation and hence relation size reduces.
3) Better technique is to use selection rule.

Constraints

1. The optimizer is a Heuristic optimizer only. It does not contain anything related to cost based optimization.
2. Parser has certain constraints like it takes only DML queries (select queries) and not any DDL queries.
3. Also some of the clauses of SQL such as EXISTS, NOT EXIST and ORDER BY is not taken into consideration.
4. Before changing the backend database the corresponding database schema has to be specified before operating on it.
5. Transparency for user application is not possible

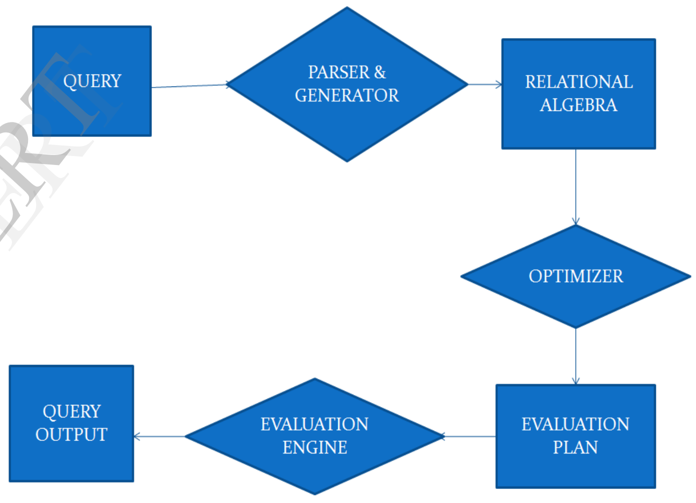

Figure 3: Query Optimization Process Flow

IV. DESIGN OF THE PROPOSED SYSTEM

Design Specifications
Query processing refers to the range of activities involved in extracting data from a database. The cost of any processing a query is usually dominated by disk access, which is slow compared to memory access
Main tasks involved are:
1. Parsing the given SQL query
2. Transforming it in the form of GLL

3. Convert GLL query into relational algebra,
4. Optimization
5. Regeneration of Queries in SQL format.

The steps involved in processing a query are as follows:
1. Parsing and translation
2. Optimization and Evaluation

Transforming query into internal form is the first action must be taken by system. The second action is query optimization that is it will generate a variety of equivalent plans for a query, and choose the least expensive one. And the third action is to evaluate this query [3].

Need for GLL:

Single GLL statement can have many nested statements in inside and, therefore, it is necessary that every individual level the nested query be optimized independently of the other levels by using GLL format the different levels of nesting can be represented by different levels in GLL. Thus, each level can be translated separately into relational algebra expression and can be processed independently.

For example if we have query as

Select a from b where c<d;

The corresponding GLL is:

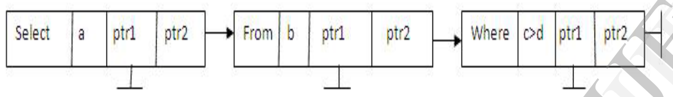

Figure 4: GLL Representation of Query

Relational algebra conversion

Transforming an SQL Queries into Relational Algebra in SQL query can itself be translated into a relational algebra expression on one of the several ways:
1. SQL query is first translated into an equivalent extended relational algebra expression.
2. SQL queries are divided into query blocks, which form the basic units that can be transformed into the algebraic operators and optimized.
3. Each query blocks contains a single SELECT-FROM-WHERE expression, along with the GROUP BY and HAVING clauses.
4. Nested queries within a query are identified as separate query blocks.

Translating SQL Queries into Relational Algebra Example:

SELECT LNAME, FNAME FROM EMPLOYEE WHERE SAL < (SELECT MAX (SAL) FROM EMPLOYEE WHERE DNO=15);

The inner block (SELECT MAX (SAL) FROM EMPLOYEE WHERE DNO=15)

Translated in:
MAXSALARY (σDNO = 5(EMPLOYEE))
The Outer block
SELECT LNAME; FNAME FROM EMPLOYEE WHERE SALARY < C

Translated in:
Π (LNAME; FNAME (σSALARY > C (EMPLOY EE)))
C will represent the result returned by the inner block.
1. The query optimizer would then select an execution plan for each block.
2. The inner block needs to be evaluated only once. (Uncorrelated nested query).
3. It is much harder to optimize the more complex correlated nested queries.

Example 2:
SELECT BALANCE FROM ACCOUNT WHERE BALANCE < 2500;
Corresponding Relational Expressions are:
σbalance < 2500 (π balance (account))or
Πbalance (σbalance < 2500(account))

A relational algebra expression annotated with instructions on how to evaluate it is called as evaluation primitive. Several primitives may be grouped together into a pipeline, in which several operations are performed in parallel. The Query execution engine picks up a query evaluation plan, executes that plan, and returns the result to the query. The various evaluation plans for a given query can have different costs. User will write a query and optimizer executes the most e client evaluation plan [1][2].

Equivalence of expressions

This phase includes matching of relational algebra with one of the forms in equivalence rules. An equivalence rule states that expressions of two forms are equivalent. We can transform either to the other while preserving equivalence. Some important equivalence rules on relational algebra are as follows:

Rule 1:
$\sigma\theta 1 \wedge \theta 2 = \sigma\theta 1 (\sigma \theta 2(E))$
Sample query:
Select * from loan where lid < 100 and bid > 1200;
Optimized query is:
Select * from (select * from loan where lid < 100) where bid > 1200;

Rule 2:
$\Pi L1 (\Pi L1; L2; (\Pi L1 \ldots\ldots Ln (E)) ..) = \Pi L1 (E)$
Sample query:
Select lid from (select lid, bid from loan);
Optimized query is:
Select lid from loan;

Rule 3:
$\sigma\theta 1 (E1 \infty E2) = (\sigma\theta 1 (E)) \infty (E2)$
Sample query:
Select from loan; branch where loan:lid < 100 and branch: bid = loan:lid;
Optimized query is:
Select * from (select * from loan1 where loan1:lid < 100) t; branch t1 where t1:bid = t:bid;

Rule 4:
$\sigma\theta 1 \wedge \theta 2 (E1 \infty E2) = (\sigma\theta 1 (E1)) \infty (\sigma\theta 2 (E2))$
Sample query:
Select * from loan, branch where loan.lid=100 and branch.name=PUNE and branch.bid=loan1.lid;
Optimized query is:
Select * from (select * from loan where loan.lid=100 and branch.name=PUNE) t1 where t.lid=t1.bid;

Rule 5:
$\Pi L1L2 (E1 \infty E2) = \Pi L1 (E1)) \infty (\Pi L2 (E2))$
Sample query:
Select lid, name from loan1, branch where branch.bid=loan.bid;
Optimized query is:
Select * from (select lid from loan1) t, (select name from branch) t1 where t.bid=t1.bid;

Rule 6:
$\sigma\theta 1 (E1) - (E2) = \sigma\theta 1 (E1) - \sigma\theta 2 (E2)$
Sample query:
(Select bid from branch where bid < 1000) minus (select bid from loan);
Optimized query for RHS is:
(Select bid from branch where bid < 1000) minus (select bid from loan where bid < 1000);

Rule 7:
$\sigma\theta 1 (E1) \cap E2 = \sigma\theta 1 (E1) \cap \sigma\theta 1 (E1))$
Sample query:
(Select bid from branch where bid < 1000) intersect (select bid from loan);
Optimized query for RHS is:
(Select bid from branch where bid < 1000) intersect (select bid from loan where bid < 1000);

Rule 8:
$\Pi L (E1 \cup E2) = \Pi L1 (E1) \cup \Pi L (E2)$
Sample query:
Select bid from (select from branch where bid   1000) union (select from loan1);
Optimized query for RHS is:
(Select bid from branch where bid   1000) union (select bid from loan1 where bid < 1000);

Optimization

In this stage, the query processor applies rules to the internal data block structures of the query to translate these structures into equivalent, but more efficient client representations. The rules are mostly based upon mathematical models of the relational algebra [2][4]. Expression and tree (heuristics), upon cost estimates of various algorithms, applied for operations or upon the semantics within the query and the relations it has. Selecting the correct rules to apply, when to apply them and how should be they applied is the function of the query optimizer [3] [6]. This phase includes the use of some heuristic rules such as performing selections and projection operations as early as possible.

Regeneration

Finally integrate all the intermediate SQL statements that will be passing to backend database.

V. CONCLUSION

We have proposed a new approach to translating an SQL queries into equivalent highly optimized SQL queries found in many commercial databases. A test database is built consisting of several lacks records. This test database will then be used to time the execution speeds of "identical" queries in the existing and new built query optimizer. It proves that like to the large amount of data, data structure, complex transaction logic and request for high data integrity and security in DBS query optimization is of at most importance. Ability to process queries in a timely manner is one of the most critical functional requirements of a DBMS. This is particularly proves for very large, mission critical applications such as banking systems, weather forecasting and Stoke market applications, which can contain millions or even more than millions of records. The need for faster and faster results never ceases. Thus, a great amount of research and resources are spent on creating highly efficient query optimization engines.

VI. REFERENCES

[1] Jayant R.Haritsa "Query Optimizer plan Diagram: Production, Reduction and Application, Data Engineering (ICDE)", *2011 IEEE 27th International conference*

[2] G. R. Bamnote and S. S. Agrawal "*Introduction to Query Processing and Optimization*" IJRCSSE Volume 3, Issue 7, July 2013.

[3] Yannis E.Ioannidis paper on "Query Optimization" Computer Sciences Department University of Wisconsin Madison, *WI 53706 in 2011*

[4] Surajit Chaudhuri "*An Overview of Query Optimization in Relational Systems*" Microsoft Research

[5] Anuja K Gaikwad, Rupali A Davalgaonkar and Seema Vaidya. Article: "*Pathfinder of X Query for a Relational Database System*." International Journal of Computer Applications NTSACT (1):22-24, August 2011.

[6] Classle.net "*Query Optimization - Heuristic Approach*" Available through: <https://www.classle.net/book/query-optimization-heuristic-approach> [Accessed 3 March 2014]