ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 3, Issue 6, June 2014

HEURISTIC BASED QUERY OPTIMIZATION

Vishal Hatmode[1], Prof. Sonali Rangdale[2]

Department of Information Technology, Siddhant College of Engineering, Pune, India[1,2]

Abstract: In this paper, we will enlist the process of SQL query optimization based on Heuristic approach. It is often found in the database industry that a lot of time is consumed in executing inefficient SQL queries. The problem here is that although the DBA knows that the queries red are inefficient, the large sections of people who are actually ring these queries are unable to write efficient queries. As a result, the performance of the entire system degrades because of the drastic fall in the system throughput i.e. the number of transactions performed per unit time is reduced. Typically, to overcome this problematic situation, most of the people frequently consult the DBA for writing efficient queries. This approach results in a lot of time loss. A better solution is using a Query Optimizer. A Query Optimizer will accept the inputted user query and automatically generate a equivalent but highly optimized query. This will save a lot of time and e ort. This in turn improves the system throughput and its overall performance. The Query Optimizer in this project is a Heuristic Optimizer. It basically tries to minimize the number of accesses by reducing the number of tuples and number of columns to be searched.

Keywords: Query Optimization, Heuristic Approach, tuples

I. INTRODUCTION

Query optimization is a function of many relational database management systems in which multiple query plans for satisfying a query are examined and a good query plan is identified. This may or not be the absolute best strategy because there are many ways of doing plans. There is a trade-off between the amount of time spent figuring out the best plan and the amount running the plan. Different qualities of database management systems have different ways of balancing these two. Query Optimizers are one of the main means by which modern database systems achieve their performance advantages. [1] Given a request for data retrieval, an Optimizer will choose an optimal plan for evaluating the request from among the manifold alternative strategies. Query Optimizers already among the largest and most complex modules of database systems, and they have proven difficult to modify and extend to accommodate these areas. The area of query optimization is very large within the database field. It has been studied in a great variety of contexts and from many divergent angles, giving rise to several diverse solutions in each case. Over time, SQL has emerged as the standard for relational query languages. Two key components of the query evaluation component of a SQL database system are the query optimizer and the query execution engine [1].

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 actually 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 use of dynamic programming for deciding best plan. Its main drawback is that it is very costly. As a result most of the optimizers do not employ this strategy. A cost estimation technique is 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.[2]

1. Generates all possible query execution plans and then cost is
   Calculate

2. Quality depends on complexity and accuracy of cost
   Function.

Cost-based query Optimization:

Algebraic Expressions for following query-

SELECT p.pname, d.dname FROM Patients p, Doctors d WHERE p.doctor = d.dname AND d.dgender ='M'

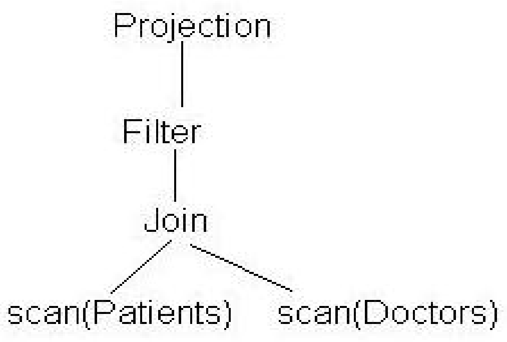

Figure 1: Relational Algebra Expression for Query

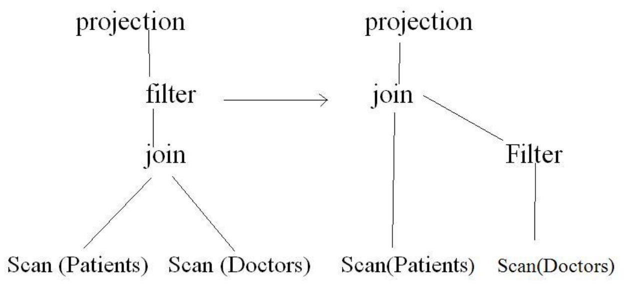

Figure 2: Execution Plan

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

Disadvantages:
1. Uses cost based optimization hence expensive.

Semantic Query Optimization

Two queries are 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 the process of determining the set of semantic transformations that results in a semantically equivalent query with a lower execution cost. ODB-Optimizer determines more specialized classes to be accessed and reduces the number of factors by applying the Integrity Constraint Rules.[2]

Advantages

1. Supports recursive queries and queries having negation, disjunction.

Disadvantages:

1. Suitable for only simple prototypes.

2. No commercial implementations exist.

II.PROPOSED ARCHITECTURE

Existing System:

Oracle currently uses cost based optimization, and rule based optimization. The Oracle database provides query optimization. You can influence the optimizer's choices by setting the optimizer goal, and by gathering representative statistics for the query optimizer. The optimizer goal is either throughput or response time. Oracle 7 introduced concept of cost based query optimization. Oracle 10g contained both cost based and rule based optimization. As rule based optimization proved to inefficient oracle has removed rule based optimization and current version of oracle i.e. 11g uses cost based optimization only [2].

Proposed System

The Query Optimizer in this project is a Heuristic Optimizer. It basically 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 optimizer can decide optimized query execution plan. [2]

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. Important Heuristic Rules used are: [2][3]

1. Perform selection as early as possible.

2. Perform projections as early as possible.

Cost-based optimization is expensive, even with dynamic programming. Systems may use heuristics to reduce the number of choices that must be made in a cost-based fashion. Heuristic optimization transforms the query-tree by using a set of rules that typically (but not in all cases) improves execution performance.

1. Perform selection early (reduces the number of tuples)

2. Perform projection early (reduces the number of attributes)

3. Perform most restrictive selection and join operations (i.e. with smallest result size) before other similar operations.

Some systems use only heuristics; others combine 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 have 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.

2. 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.

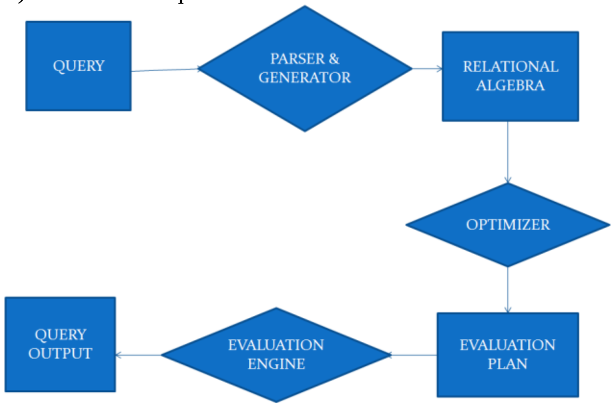

Figure 3: Query Optimization Process Flow

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

III. DESIGN OF THE PROPOSED SYSTEM

Design Specifications

Query processing refers to range of activities involved in extracting data from a database. The cost of 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
3. Evaluation

The first action system must take in query processing is to translate a given query into its internal form. 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.

Javacc and Parser Generator:

JavaCC is a parser generator and a lexical analyser generator. Parsers and lexical analysers are software components for dealing with input of character sequences. Compilers and Interpreters incorporate lexical analysers and parsers to decipher les containing programs, However lexical analysers and parsers can be used in a wide variety of other applications as well. The parser and translator is the second module in the project. It has been developed using the parser generator JavaCC. JavaCC constructs a recursive descent top down parser when provided with necessary grammar for SQL.

Javacc Working:

The grammar was also provided with necessary semantic actions which for translating the select statements into the corresponding GLL (Generalized Linked List) format. Thus the parser and translator have been merged into single module. Now to be able to understand this module working we will first have to examine GLL. Generalized Linked List (GLL): Each select statement will be mapped onto a GLL format so that it can be easily translated into a corresponding relational expression. The GLL structure consists of the following four fields:

1. Type of node.
2. Contents of node.
3. Pointer to next node on same level.

Need for GLL:

Single GLL statement can have many nested statements inside and therefore it is necessary that each individual level In the nested query be optimized independently of the other levels by using GLL format the different levels of nesting can be represented by different

ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940
International Journal of Advanced Research in Computer and Communication Engineering
Vol. 3, Issue 6, June 2014

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:

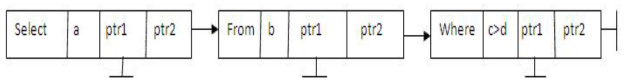

Figure 4: GLL Representation of Query

Relational algebra conversion

Translating SQL Queries into Relational Algebra in SQL a 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 decomposed into query blocks, which form the basic units that can be translated into the algebraic operators and optimized.

3. Query block contains a single SELECT-FROM-WHERE expression, as well as 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 SALARY < (SELECT MAX (SALARY) FROM EMPLOYEE WHERE DNO=5);

The inner block

(SELECT MAX (SALARY) FROM EMPLOYEE WHERE DNO=5)

Translated in:

MAXSALARY (σDNO = 5(EMPLOY EE))

The Outer block

SELECT LNAME; FNAME FROM EMPLOY EE WHERE SALARY < C

Translated in:

Π (LNAME; FNAME (σSALARY > C (EMPLOY EE)))

C represents the result returned from the inner block.

1. The query optimizer would then choose 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. A sequence of primitive operations that can be used to evaluate a query is a query evaluation plan or query execution plan. The Query execution engine takes a query evaluation plan, executes that plan, and returns the answer to the query. The different evaluation plans for a given query can have different costs. User will write a query and optimizer executes the most e client evaluation plan.

Equivalence of expressions

This phase includes matching of relational algebra with one of the forms in equivalence rules. An equivalence rule says that expressions of two forms are equivalent: We can transform either to the other while preserving equivalence. By preserve equivalence we mean that relations generated by the two expressions have the same set of attributes may be ordered differently. Equivalence rules are used by the optimizer to transform expressions into other logically equivalent expressions. 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:

LHS un-optimized query:

Select * from loan where lid < 100 and bid > 1200;

ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 3, Issue 6, June 2014

Optimized query for RHS is:

Select * from (select * from loan where lid < 100) where bid > 1200;

Rule 2:

ΠL1 (ΠL1; L2; (ΠL1…..Ln (E)) ..) = ΠL1 (E)

Sample query:

LHS un-optimized query:

Select lid from (select lid, bid from loan);

Optimized query for RHS is:

Select lid from loan;

Rule 3:

σθ1 (E1∞E2) = (σθ1 (E)) ∞ (E2)

Sample query:

LHS un-optimized query:

Select from loan; branch where loan:lid < 100 and branch: bid = loan:lid; Optimized query for RHS is :

Select * from (select * from loan1 where loan1:lid < 100) t; branch t1 where t1:bid = t:bid;

Rule 4:

σθ1 ^ θ2 (E1∞E2) = (σθ1 (E1)) ∞ (σθ2 (E2))

Sample query:

LHS un-optimized query:

Select * from loan, branch where loan.lid=100 and branch.name=PUNE and branch.bid=loan1.lid;

Optimized query for RHS is:

Select * from (select * from loan where loan.lid=100 and branch.name=PUNE) t1 where t.lid=t1.bid;

Rule 5:

ΠL1L2 (E1∞E2) = ΠL1 (E1)) ∞ (ΠL2 (E2))

Sample query:

LHS un-optimized query:

Select lid, name from loan1, branch where branch.bid=loan.bid;

Optimized query for RHS is:

Select * from (select lid from loan1) t, (select name from branch) t1 where t.bid=t1.bid;

Rule 6:

σθ1 (E1) - (E2) = σθ1 (E1)-σθ2 (E2)

Sample query:

LHS un-optimized 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:

σθ1 (E1) ∩ E2 = σθ1 (E1) ∩ σθ1 (E1))

Sample query:

LHS un-optimized 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:

ΠL (E1 U E2) = ΠL1 (E1) U ΠL (E2)

Sample query:

LHS un-optimized 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 structures of the query to transform these structures into equivalent, but more e client representations. The rules can be based upon mathematical models of the relational algebra. Expression and tree (heuristics), upon cost estimates of different algorithms applied to operations or upon the semantics within the query and the relations it involves. Selecting the proper rules to apply, when to apply them and how they are applied is the function of the query optimizer. This phase includes use of some heuristic rules such as performing selections and projection operations as early as possible.

Regeneration

This phase includes regenerating queries in SQL format from relational algebra expression and ring resultant query to database using corresponding drivers. For each level of nesting use individual relational algebra expression of each level and for each of them follow the same process. Finally integrate all the intermediate SQL statements that will b passed to backend database. Consider one example which will describe actual relational process. Suppose user has inputted following query:

Select Customer-name where Customer-name=awt and Customer-id in (select * from Customer where Customer-name=awt and Customer-SSN=200);

The following relational algebra given to regeneration phase: $\Pi L\sigma\theta 1$ ($\sigma\theta 2$ ($\sigma\theta 3$ (E)))

IV. CONCLUSION

We have proposed a new approach to translating a 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 as 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. One of the most critical functional requirements of a DBMS is its ability to process queries in a timely manner. This is particularly true for very large, mission critical applications such as weather forecasting, banking systems and aeronautical applications, which can contain millions and even trillions of records. The need for faster and faster, "immediate" results never ceases. Thus, a great deal of research and resources is spent on creating smarter, highly e client query optimization engines. Some of the basic techniques of query processing and optimization will be presented in this project.

REFERENCES

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

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

[3]Leo Giakoumakis, Cesar Galindo-Legaria paper on "Testing SQL Server's Query Optimizer: Challenges, Techniques and Experiences" . IEEE Data Eng. Bull. 31(1): 36-43 (2010)

[4]Maier, Leonard Shapiro paper on "The Columbia Query Optimization Project" Port-land State University (NSF IRI-9610013) and to the Oregon Graduate Institute (NSF IRI-9619977)