International Journal of Emerging Trends & Technology in Computer Science (IJETTCS)
Web Site: www.ijettcs.org Email: editor@ijettcs.org, editorijettcs@gmail.com
Volume 7, Issue 6, November - December 2018 ISSN 2278-6856

Dynamic Query Optimization in Relational Database

Mr. Omkar Singh[1], Dr. S. K. Singh[2], Mr. Ajeet Pal[3], Ms. Vibha Mishra[4]

[1,2,3,4] Thakur College of Science and Commerce, Thakur Village, Kandivali – E, Mumbai – 400101

Abstract : Query optimizer is an important component in the architecture of relational data base management system. This component is responsible for translating user submitted query into an efficient query evolution program which can be executed against the database. The present query evolution existing algorithm tries to find the best possible plan to execute a query with a minimum amount of time using mostly semi accurate statistical information (e.g. sizes of temporary relations, selectivity factors, and availability of resources). It is a static approach for generating optimal or close to optimal execution plan. Which in turn increases the execution cost of the query to reduce the execution cost of the query; I propose a new dynamic query optimization algorithm which is based on greedy dynamic programming algorithm uses randomized strategies and reduces the execution cost of the queries and system resources and also it works efficiently with distributed and centralized databases.

Keywords- Query optimizer, relational database, static query optimization, dynamic query optimization.

1 INTRODUCTION

The query optimizer is the component of a database management system that attempts to determine the most efficient way to execute a query. The optimizer considers the possible query plans for a given input query, and attempts to determine which of those plans will be the most efficient. Cost-based query optimizers assign an estimated "cost" to each possible query plan, and choose the plan with the smallest cost. Costs are used to estimate the runtime cost of evaluating the query, in terms of the number of I/O operations required, the CPU requirements, and other factors determined from the data dictionary. The set of query plans examined is formed by examining the possible access paths (e.g. index scan, sequential scan) and join algorithms (e.g. sort-merge join, hash join, nested loop join). The search space can become quite large depending on the complexity of the SQL query.

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 occurs. However, some database engines allow guiding the query optimizer with hints. Not every aspect of SQL execution can be optimally planned ahead of time. Oracle thus makes dynamic adjustments to its query-processing strategies based on the current database workload. The goal of dynamic optimizations is to achieve optimal performance even when each query may not be able to obtain the ideal amount of CPU or memory resources. Three steps are involved for query processing: decomposition, optimization and execution.

The first step decomposes a relational query using logical schema into an algebraic query. During this step syntactic, semantic and authorization are done.

The second step is responsible for generating an efficient execution plan for the given SQL query from the considered search space.

The third step consists in implementing the efficient execution plan.

Most of the DBMSs have used the static optimization approach which consists of generating an optimal (or close to the optimal) execution plan, then executing it until the termination. All the methods, using this approach, suppose that the values of the parameters used (e.g. sizes of temporary relations, selectivity factors, availability of resources) to generate the execution plan are always valid during its execution. However, this hypothesis is often unwarranted. Indeed, the values of these parameters can become invalid during the execution due to several causes. Estimation errors: the estimation on the sizes of the temporary relations and the relational operator costs of an execution plan can be erroneous because of the absence, the obsolescence, and the inaccuracy of the statistics describing the data, or the errors on the hypotheses made by the cost model. For instance, the dependence or the independence between the attributes member of a selective clause(e.g. town='Paris' and country = 'France').

These estimation errors are propagated in the rest of the execution plan. Moreover, showed that the propagation of these errors is exponential with the number of joins. Unavailability of resources: at compile-time, the optimizer does not have any information about the system state when the query will run, in particular, about the availability of resources to allocate (e.g. available memory, CPU load).Because of reasons quoted previously, the execution plans generated by a static optimizer can be sub-optimal. To correct this sub-optimality, some recent research suggest improving the accuracy of parameter values used during the choice of the execution plan. A first solution consists in improving the quality of the statistics on the data by using the previous executions. This solution was used by to improve the estimation accuracy of the operator

selectivity factors and by to estimate the correlation between predicates

2 Related Work

The optimization algorithm is based on relational algebra equivalence transformation. The basic idea of proposed algorithm is: To convert the query problem into relational algebra expression, analyse the obtained query syntax tree, and optimize according to equivalence rules. The algorithm first uses relational algebra equivalence transformation to raise the connecting and merging operations in the query tree as much as possible, while moving the selection and projection operations down to the fragment definition. Then to determine the horizontal or vertical slice, if the horizontal slice, to compare with the slice and the selection conditions to remove the contradictory fragments, if only a fragment left then to remove one parallel operation.

If the vertical slice, then to compare with the fragment property set and the attribute set involved in projection operation to remove all the unrelated fragments. If only one vertical segment left then to remove one connection operation, thus to achieve the purpose of optimization query. It is to use heuristic optimization method to optimize the relational algebra expression. And in the relational algebra expression, the most time and space operation is Cartesian product and join operations, so, the implementation of selection and projection operations as early as possible, to avoid the direct Cartesian product operation, then combines a series of selection and projection before and after it together to reduce the size of intermediate relations, thus to achieve optimization. The Proposed algorithm utilizes some of the rules of transform an initial query tree into an optimized tree that is efficient to execute during heuristic optimization.

INPUT: SQL Statement.
OUTPUT: Best evaluation plan

Algorithm:
Step 1: Design the initial canonical tree of the query.
Step 2: Move the SELECT Operation down the query tree.
Step 3: Apply more restrictive SELECT operation first. If the two relations are residing at same site, they will be handled first.
Step 4: Replace CARTESIAN PRODUCT and SELECT with JOIN operation.
Step 5: Move PROJECT operations down the query tree.

3 Methodology

The main motivations to introduce 'dynamicity' into query optimization, in particular to reduce the query execution time or to speed up the query response time of the Database management system. We propose a new algorithm which will work efficiently with simple, nested, co-related queries. The algorithm will exhibit the dynamism in calculating query execution plan for the last inner nested query. Same technique is again applied for outer layer nested query.

Simultaneously algorithm is generating the optimal query execution plan and executing the plan for each nested query. It works in bottom-up way by building more complex sub-plans from simpler sub-plans until the complete plan is constructed.

SELECT {DISTINCT} <list of
column> FROM <list of Table>
{WHERE <list of "Boolean Factor"
(Predicates)} {GROUP BY <list of column>}
{HAVING <list of Boolean Factor>}
{ORDER BY <list of column>};

3.1 The algorithm generates optimal query execution plan based on the following semantics:

3.1.1 Take Cartesian product (cross-product) of tables in FROM clause, projecting to-
3.1.2 Only those columns that appear in other clauses.
3.1.3 If there's a WHERE clause, apply all filters in it.
3.1.4 If there's a GROUP BY clause, form groups on the result.

3.1.5 If there's a HAVING clause, filter groups with it.
3.1.6 If there's an ORDER BY clause, make sure output is in the right order.
3.1.7 If there's a DISTINCT modifier,
3.1.8 Remove duplicates

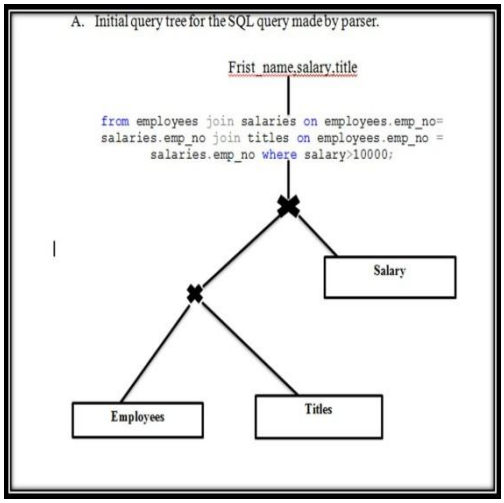
3.2 Heuristic Based Algorithm for Query Optimization.

There are following steps in converting a query tree during heuristic optimization by using various rules. First we have query in SQL and convert it into relational algebra query.

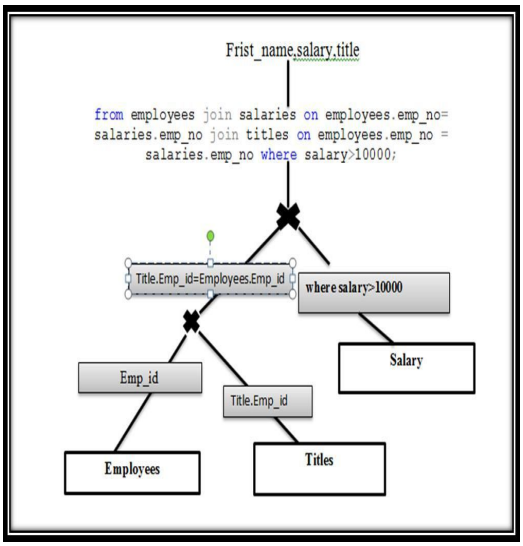
Example:

```
select first_name,salary,dept_no
from employees
full outer join dept_emp on employees.emp_no = dept_emp.emp_no
full outer join salaries on employees.emp_no = salaries.emp_no;
```

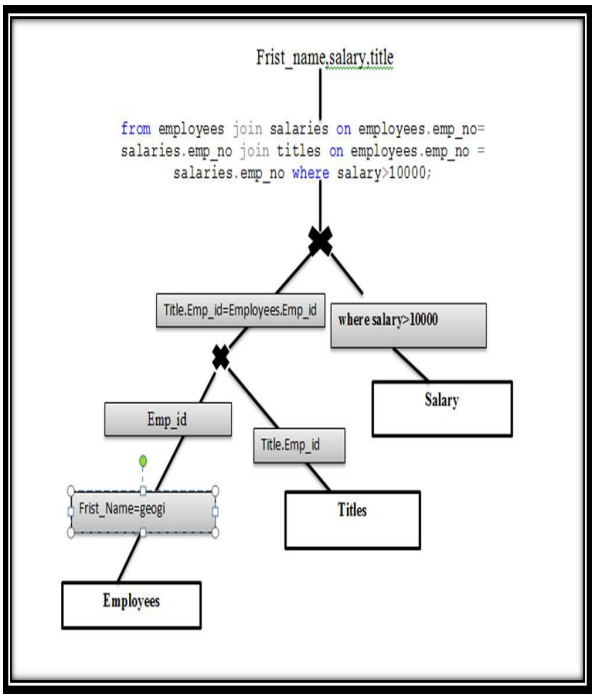
3.2.1 Initial query tree for the SQL query made by parser.



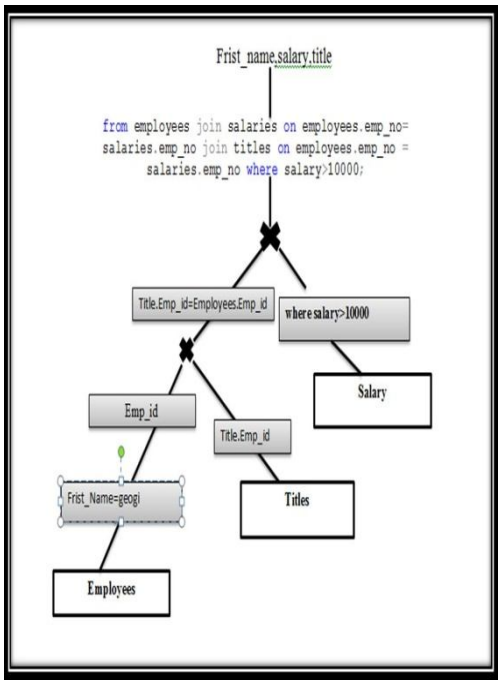
3.2.2 Moving SELECT operations down the query tree



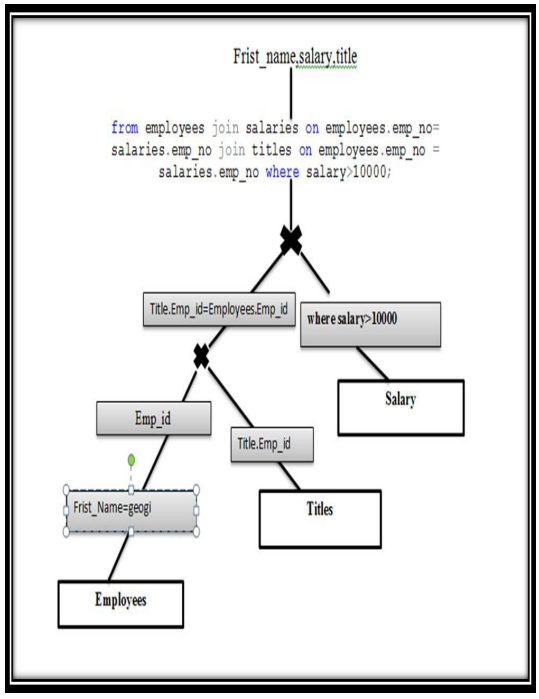
3.2.3Applying the more restrictive SELECT operation first.



3.2.4 Replacing CARTESIAN PRODUCT and SELECT with JOIN operations.



3.2.5Moving PROJECT operations down the query tree



Conclusion

Most of the query optimizers of database management systems are using static approaches to generate optimal execution plan for executing queries against the databases. it increases the execution cost of the queries and also consumes more system resources like CPU& Memory. The proposed algorithm which is based on greedy dynamic programming algorithm uses randomized strategies and reduces the execution cost of the queries and system resources.

Future Enhancement

Optimization only on select-project-join queries also need to handle complex queries (e.g. Top-k query, Group by,

aggregations, materialized view, recursive query, dynamic query).Optimize query on centralized system can be extended to implement on parallel database and Distributed Database.

REFERENCES

[1] SQL SERVER 20012 Query Optimization Of Join Function.

[2] Vishal P. Patel, Hardik R. Kadiya "Optimization of Large Join Query using Heuristic Greedy Algorithm" IJCAT - International Journal of Computing and Technology Volume 1, Issue 1, February 2014.

[3] N. Satyanarayan, SK Sharfuddin, SK Jan Bhasha "New Dynamic Query Optimization Technique in Relational Database Management Systems" International Journal of Communication Network Security, ISSN: 2231 – 1882, Volume-2, Issue-2, 2013.