*Journal of Institute of Science and Technology 2019, **24(1)**: 42-46*
ISSN: 2469-9062 (print), 2467-9240 (e)
Doi: http://doi.org/10.3126/jist.v24i1.24627

Research Article

COST-BASED QUERY OPTIMIZATION IN CENTRALIZED RELATIONAL DATABASES

Nawaraj Paudel*, Jagdish Bhatta

Central Department of Computer Science and Information Technology
Tribhuvan University, Kathmandu, Nepal
**Corresponding author: nawarajpaudel@cdcsit.edu.np*

*(**Received:** May 20, 2019; **Revised**: June 17, 2019; **Accepted:** June 18, 2019)*

ABSTRACT

Query optimization is the most significant factor for any centralized relational database management system (RDBMS) that reduces the total execution time of a query. Query optimization is the process of executing a SQL (Structured Query Language) query in relational databases to determine the most efficient way to execute a given query by considering the possible query plans. The goal of query optimization is to optimize the given query for the sake of efficiency. Cost-based query optimization compares different strategies based on relative costs (amount of time that the query needs to run) and selects and executes one that minimizes the cost. The cost of a strategy is just an estimate based on how many estimated CPU and I/O resources that the query will use. In this paper, cost is considered by counting number of disk accesses for each query plan because disk access tends to be the dominant cost in query processing for centralized relational databases.

Keywords: Query-optimization, Query-processing, Cost-based query optimization, SQL database

INTRODUCTION

A centralized relational database is the database that stores and maintains data in a single location using tables (or relations). Because of widespread use of such databases, it is important to reduce system resources required to fulfill a query and ultimately provide correct information faster. Query optimization in database has gained significant importance as it helps to reduce the size, memory usage and time required for any query to be processed. The main objective of any query optimization is to determine the best strategy for executing each query. It identifies an efficient way to execute the query with less time complexity to produce better results. This process can be formally defined as transforming a query into an equivalent form which can be evaluated more efficiently (Tejy 2016). Query optimization is the process of determining the most efficient way to execute a given query by considering the possible query plans. Wong and Youssefi (1976) and Selinger *et al.* (1979) have done work related to the relational query optimization. Several approaches, methods and techniques of query optimization have been proposed for various DBMS (i. e., relational, deductive, distributed, object, parallel). The quality of query optimization methods depends strongly on the accuracy on the efficiency of cost models (Hussein *et al.* 2005, Naacke *et al.* 1998, Zhu *et al.* 2003, Adali *et al.* 1996, Ganguly *et al.* 1996, Gardarin *et al.* 1996).

Query optimization is an important part of query processing. The four main phases of query processing are decomposition (consisting of parsing and validation), optimization, code generation and execution. Query processing is the activities involved in parsing, validating, optimizing and executing a query. Query processing transforms a query written in a high-level language, typically SQL, into a correct and efficient execution strategy expressed in a low-level language (implementing the relational algebra), and executes the strategy to retrieve the required data. The activity of choosing an efficient executing strategy for processing query is query optimization (Thomas & Carolyn 2015). It is generally a process of reducing total execution time of the query, which is the sum of the execution times of all individual operations that make up the query (Selinger *et al.* 1979).

Query optimization needs database statistics to evaluate different execution strategies properly. The statistics cover information about relations, attributes and indexes. The accuracy and currency of these statistics have a significant bearing on the efficiency of the execution strategy chosen. Keeping the statistics current can be problematic. If the statistics is updated every time, the database is updated there is significant impact on performance during peak periods. An alternative and generally preferable approach is to update the statistics on a periodic basis, for example, nightly or whenever the system is idle.

The cost-based query optimization technique compares different strategies based on their relative costs and selects the one that minimizes resource usage. Because disk access is slow compared to memory access, disk access tends to be the dominant cost in query processing for a centralized relational DBMS. The main objective of this study is to compare cost of different execution strategies of a SQL query represented as relational algebra expressions in centralized relational databases to choose most efficient execution strategy.

As given in Fig.1, the query processing has four phases as query decomposition, query optimization, code generation

and runtime query execution. The important aspect of query processing is query decomposition. Query decomposition is the first phase of query processing. This phase transforms a high-level query into a relational algebra query and to check whether the query is syntactically and semantically correct. The query optimization phase chooses an efficient execution strategy that minimizes the overall cost of execution. This phase uses information from database statistics to find cost of each execution strategy. The code generation phase receives an optimal execution strategy from the optimization phase and produces an iterative execution plan that is usable by the rest of the database. Runtime query execution is the last phase of query processing and runs the query and displays the required result.

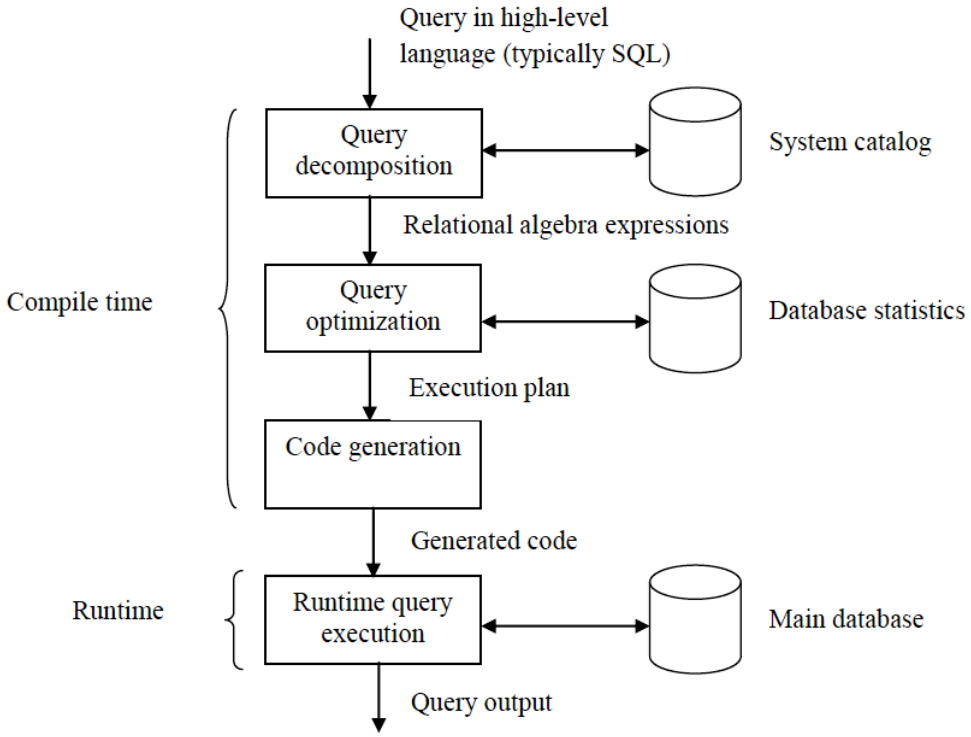

Fig. 1. Phases of query optimization (Thomas & Carolyn 2015)

MATERIALS AND METHODS

Cost-based query optimization

Cost-based query optimization is an overall process of choosing the most efficient means of executing a SQL statement based on overall cost of the query. The efficient execution is the execution with minimum cost. To find the cost of query execution plan, the optimization technique uses database statistics.

In this optimization technique, all of the possible ways or scenarios in which a query can be executed will be assigned a 'cost', which indicates how efficiently that query can be run. Then, the optimizer will pick the scenario that has the least cost and execute the query using that scenario, because the query with least cost is the most efficient way to run the query. The dominant cost in query processing for centralized relational databases is disk access because disk access is slower than memory access. So the optimization technique counts the number of disk accesses of each scenario and execute the scenario with minimum number of disk accesses. In centralized systems, the costs are dominated by the time for secondary storage access although the CPU costs may be quite high for complex queries (Gotlieb 1975).

During decomposition phase, high level query (SQL) is transformed into some internal representation typically using query tree (relational algebra tree). Thomas and Carolyn (2015) have devised a rule for constructing query tree as follows:

- A leaf node is created for each base relation in the query.
- A non leaf node is created for each intermediate relation produced by a relational algebra operation.
- The root of the tree represents the result of thee query.
- The sequence of operations is directed from the leaves to the root.

RESULTS AND DISCUSSION

Consider two relations employee (emp_no, emp_name, emp_address, position, salary, branch_no) and branch

(branck_no, branch_city, branch_address, city) with a member of employee can only work at one branch.

Consider an SQL query as given below.

SELECT*

FROM Employee, Branch

WHERE Employee.branch_no = Branch.branch_no **AND** Employee.position = 'Manager' **AND** Branch.city = 'Kathmandu';

We can write three different relational algebra queries for the above SQL query as given below.

Query 1: $\sigma_{(position = 'Manager') \wedge (city = 'Kathmandu') \wedge (Employee.branch\_no = Branch.branch\_no)}$ (Employee × Branch)

Query 2: $\sigma_{(position = 'Manager') \wedge (city = 'Kathmandu')}$ (Employee ⋈ Branch)

Query 3: ($\sigma_{position = 'Manager'}$(Employee)) ⋈ ($\sigma_{city = 'Kathmandu'}$ (Branch))

The relational algebra trees for each of above queries are listed below as shown in Figs 2-4.

Suppose there are 2000 tuples in Employee, 20 tuples in Branch, 20 Managers (one for each branch), and 10 Kathmandu branches. To compare these three queries, we assume number of disk accesses. We also assume that there are no indexes or sort keys on either relation. The results of any intermediate operations are stored on disk. The cost of the final write is ignored because it is the same in each query. We further assume that tuples are accessed one at a time (although in practice disk accesses would be based on blocks, which would typically contain several tuples), and main memory is large enough to process entire relations for each relational algebra operation.

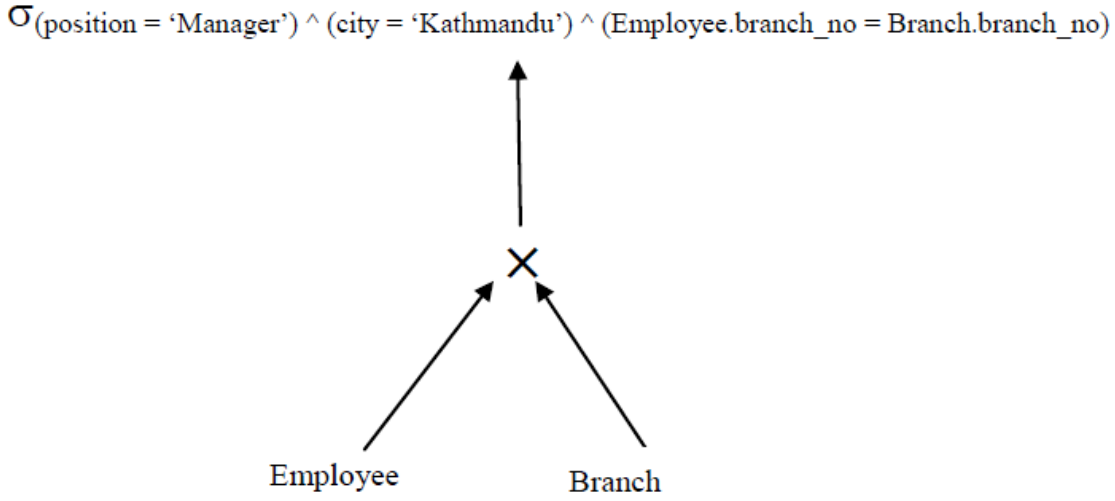

Fig. 2. Relational algebra tree for query 1

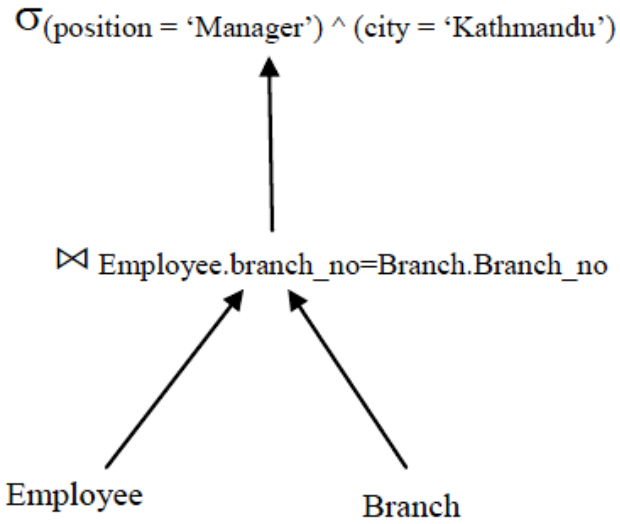

Fig. 3. Relational algebra tree for query 2

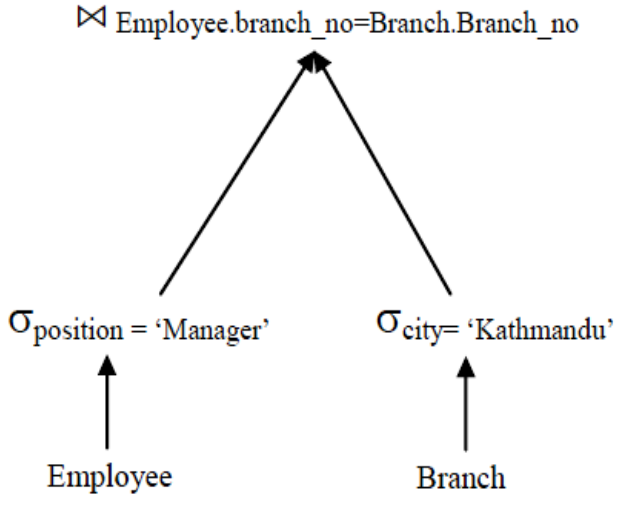

Fig. 4. Relational algebra tree for query 3

The query1 calculates the Cartesian product of Employee and Branch, which requires (2000 + 20) disk accesses to read these two relations, and creates a relation with (2000 × 20) tuples. We then have to read each of these tuples again to test them against the selection predicate at a cost of another (2000 × 20) disk accesses, giving a total cost of (2000 + 20) + 2 × (2000 × 20) = 82020 disk accesses.

The query 2 joins Employee and Branch which again requires (2000 + 20) disk accesses to read each of the relations. The Join of these two relations has 2000 tuples, one for each member of Employee. Consequently, the Selection operation requires 2000 disk accesses to read the result of the join, giving a total cost of (2000 + 20) + 2 × (2000) = 6020 disk accesses.

The query 3 first reads each Employee tuple to determine the Manager tuples, which requires 2000 disk accesses and produces a relation with 20 tuples. Similarly, the second Selection operation reads each Branch tuple to determine the Kathmandu branches, which requires 20 disk accesses and produces a relation with 10 tuples. The final operation is the join of the reduced Employee and Branch relations, which requires (20 + 10) disk accesses, giving a total cost of (2000 + 20) + (20 + 10) + (20 + 10) = 2080 disk accesses. From the calculations above, it is clear that query 3 is the most efficient query and is 2.89 times faster than query 2 and 39.43 times faster than the query1. Figure 5 shows the cost comparisons of each relational algebra query. It can be seen clearly that that the cost of query 3 is minimum as compared to the cost of other two queries.

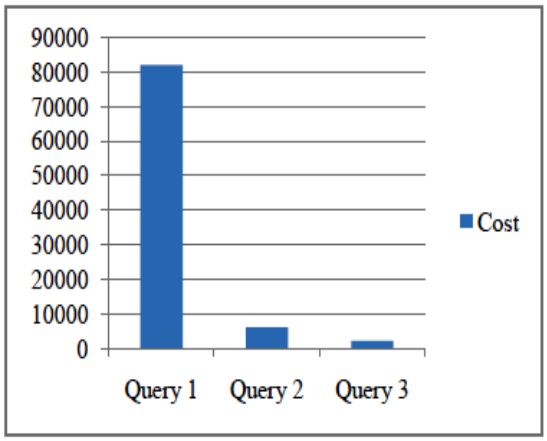

Fig. 5. Cost comparison of each query

If we considered 10000 tuples in Employee and 1000 tuples in Branch query 3 would be 2.38 times faster than query 2 and 1536.84 times faster than the query1. Since, Cartesian product and Join operations are much more expensive than Selection operation, query3 significantly reduces the size of the relations that are being joined together.

CONCLUSION

Query optimization is the process of determining the most efficient method for a SQL statement to access requested data. A SQL query can have different query execution strategies and the cost-based query optimization technique selects and executes the query execution strategy with least cost among all the execution strategies. To find the cost of each execution strategy, the optimization technique uses database statistics, because disk access is slower as compared with memory access and disk access tends to be the dominant cost in query processing for centralized relational databases. Using database statistics, the optimization technique counts the number of disk accesses for each execution strategy in centralized relational databases.

REFERENCES

Adali, S., Candan, K.S., Papakonstantinou, Y. and Subrahmanian, V.S. 1996. Query caching and optimization in distributed mediator systems. In: *Proceedings of ACM SIGMOD International Conference on Management of Data*, ACM Press, New York, pp. 137-148.

Ganguly, S., Goel, A. and Silberschatz, A. 1996. Efficient and accurate cost models for parallel query optimization. In: *Symposium in Principles of Database Systems PODS*, ACM Press, New York, pp. 172-182.

Gardarin, G., Sha, F. and Tang, Z.-H. 1996. Calibrating the query optimizer cost model of IRO-DB, an object-oriented federated database system. In: *Proceedings of 22nd VLDB*, Morgan Kaufmann, San Francisco, pp. 378-389.

Gotlieb, L.R. 1975. Computing joins of relations. In: *Proceedings of the ACM-SIGMOD International Conference of Management of Data*, ACM, New York, pp. 55-63.

Selinger, P.G., Astrahan, M.M., Chamberlin, D.D., Lorie, R.A. and Price, T.G. 1979. Access path selection in a relational database management system. In: *Proceeding of the 1979 ACM SIGMOD International Conference on Managment of Data*, Massachusetts, USA, pp. 23-34.

Hussein, M., Morvan, F. and Hameurlain, A. 2005. Embedded cost model in mobile agents for large scale query optimization. In: *Proceedings of the 4th International Symposium on Parallel and Distributed Computing, IEEE CS*, Los Alamitos, pp. 199-206.

Naacke, H., Gardarin, G. and Tomasic, A. 1998. Leveraging mediator cost models with heterogeneous data sources. In: *Proceedings of the*

14th International Conference on Data Engineering, IEEE CS, Los Alamitos, pp. 351-360.

Selinger, P.G., Astrashan, M., Chamberlin, D., Lorie, R. and Price, T. 1979. Access path selection in a relational database management system. In: *Proceedings of the 1979 ACM SIGMOD Conference on Management of Data*, ACM Press, New York, pp. 23-34.

Tejy, K.K. 2016. *Query optimization in database systems*. PhD Thesis, Department of Computer Applications, Faculty of Computer Applications, Dr. M.G.R. Educational and Research Institute University, Maduravoyal, Chennai, India.

Thomas, C.M. and Carolyn, B.E. 2015. *Database systems: a practical approach to design, implementation and management*. 6th Edition, Pearson, pp. 727-782.

Wong, E. and Youssefi, K. 1976. Decomposition- a strategy for query processing. *ACM Transactions on Database Systems* **1**, 223-241.

Zhu, Q., Motheramgari, S. and Sun, Y. 2003. Cost estimation for queries experiencing multiple contention states in dynamic multi-database environments. *Journal of Knowledge and Information Systems Publishers* **5**(1): 26-49.