International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 3 Issue 11, November-2014

A Novel Evaluation of Query Processing and Optimization in DBMS

Mohd Muntjir
College of Computers and Information Technology
Tail University, Taif,
Saudi Arabia

***Abstract*- Query Processing is the systematic method of accessing the require information from a database system in an expected and reliable trend. Database systems must be agile to respond to requests for information from the user i.e. process queries. In huge database systems that may be running on unreliable and elusive domain it is no easy to outcome to dynamic database query plans based on information available exclusively at compile time. Obtaining and finding the database results in a prompt manner deals with the method of Query Optimization. Adequate processing of queries is a major requirement in various interactive environments that associates huge amounts of data. Dynamic query processing in environments such as the multimedia search, Web, and distributed systems has shown a main impact on performance and optimization. This paper will suggest and propose the main concepts of query processing and query optimization in the relational database systems. It is also describing and differentiating query-processing method in relational database systems.**

Keywords: Query Processing, Query Optimization, and Database

I. INTRODUCTION

The basic part of any Database Management Systems is query processing and optimization. The outcomes of queries must be accessible in the timeframe required by the complying user [2]. Query processing techniques based on various design dimensions can be defined as [1]:

A. Query model:

Processing techniques are defined according to the query model they consider. Few techniques recognize a selection query model, where outcomes are attached basically to base tuples. Alternative techniques speculate a join query model, where final outcomes are calculated over join results. A third section considers an aggregate query model, where we are responsive in ranking groups of tuples.

B. Implementation level:

These processing techniques are defined according to their level of association with database systems. E.g., some techniques are designed in an application layer on top of the database systems, although others are implemented as query operators.

C. Data access methods:

Processing method is classified according to the data access technique they consider to be accessible in the fundamentals data sources. For instance, some techniques define the availability of random access, although others are controlled to only classified access.

D. Ranking function:

Processing techniques are classified based on the limitations they establish on the latent ranking (scoring) functions. Best suggested techniques expect monotone scoring functions.

E. Data and query uncertainty:

Processing techniques are defined based on the ambiguity elaborated in their data and query models. Many techniques establish exact results, whilst others order for proximate answers, or manage indefinite data.

II. QUERY PROCESSING

Query processing specifies to the range of activities integrated in extracting data from a database system. The activities comprise translation of queries in high-level database languages into expressions that can be used at the physical level of the file systems, and a variation of query-optimizing conversion, and real interpretation of queries. Furthermore, a database query is the vehicle for instructing a DBMS to update or fetch specific data to/from the physically stored intermediate. The real updating and fetching of data is established through different low- level procedures [10]. Instance of such operations for a relational DBMS can be relational algebra operations such as select, project, join, Cartesian product. [11]. As long as the DBMS is created and designed to process these low -level operations purposely, it can be quite the burden to a user to create requests to the DBMS in these designs.

There are three phases [12] that a query passes through during the DBMS' processing of that query: 1. parsing and translation 2. Optimization3. Evaluation

Furthermore, the first step in processing a query referred to a Database Management System is to convert the query into a form accessible by the query-processing engines. High- level query languages such as SQL defined a query as a sequence or string, of characters.

Actual sequences of characters represent different types of tokens such as literal strings operators, keywords, operands, etc. Similar to all languages, there are rules (syntax and grammar) that control how the tokens can be integrated into (i. valid statements.

The major job of the parser is to extract the tokens from the raw string of characters and translate them into the equivalent internal data elements and structures (i.e. query graph, query tree). The last task of the parser is to authenticate the validity and syntax of the real query strings. In second phase, the query processor implements rules to the internal data structures of the query to transform these structures into similar, but more adequate demonstrations. The standard can be based upon

mathematical models of the relational algebra expression and tree, upon cost calculates of various algorithms used to operations or upon the semantics within the query and the relations it integrates. Electing the proper rules to implement, when to apply them and how they are implemented is the function of the query optimization engine.

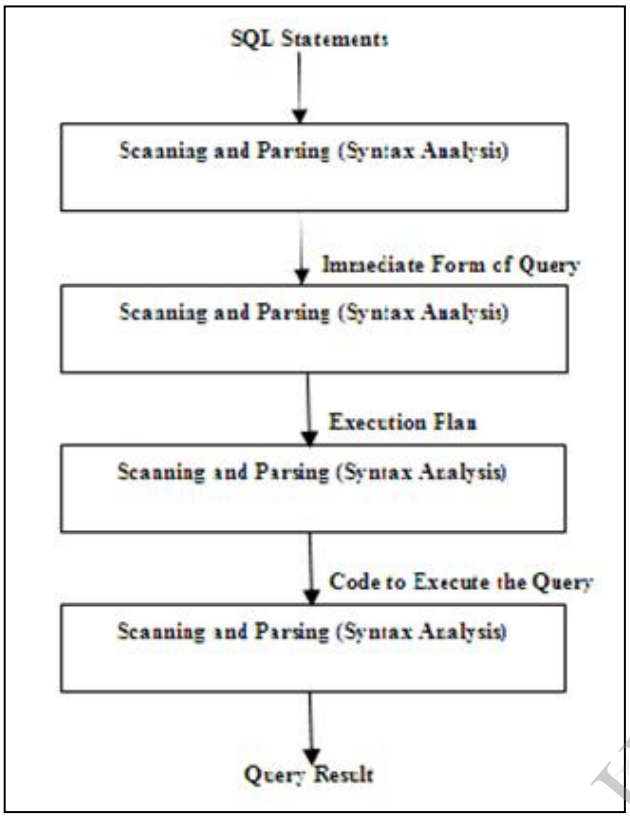

Fig. 1. Query processing

The last step in processing a query is the evaluation phase. The best evaluation plan candidate developed by the optimization engine is selection and then execution. Note that there can stand various methods of executing a query. Beyond processing a query in easy consecutive methods, many of a query's individual operations can be oppressed in parallel either as autonomous processes or threads or as interdependent pipelines of processes. Unconcerned of the method selected, the permanent results should be same.

Consider for example in Fig.2 :

Select salary
From bank
Where salary< 2500
This can be interpreted into either of the following relational algebra expressions:

- σsalary<2500 (Πsalary(bank))
- Πsalary(σsalary< 2500 (bank))

Which can also be represented as either of the following query trees?

σsalary <2500 Πsalary
Πsalaryσsalary<2500
bank bank

Fig. 2. A query-evaluation plan

III. MEASURES OF QUERY COST

Cost of query is basically measured as total overdue time for answering query in a database. Furthermore, expanse of query evaluation can be assumed in terms of a number of various assets, and CPU time to execute a query, along with disk accesses, and, the expanse of communication and broadcasting, in a distributed or parallel database system. The response time for a query-evaluation plan, assuming no other action is going on the computer systems, would scheduled for all these costs, and could be used as a better scope of the cost of the methods. In some database systems, although, disk approaches are usually the most extensive expanse, since disk accesses are low associated to in-memory operations. Further, the speed of CPU has been reestablishing much faster than have disk speed. Henceforth, it is more similar that the time spent in disk activity will continue to control and maintain the total time to execute queries in database systems. Decisively, manipulating and estimating the CPU time is basically conventional compared to calculating the disk-access expanses? Mostly the people consider that the disk-access cost feasible estimation of the cost of a query-evaluation strategy in databases.

IV. QUERY ALGORITHMS

A. *Selection Algorithms*

According to selection algorithm, select operation must search through the data files for records meeting the selection prototype. Following are some examples of simple and basic (one attribute) selection algorithms [13].

1) Linear search:

The Database must use a linear search when no database index exists on the selection condition. Each record from the file is read and compared to the selection method. The execution expanse for searching on a non-key attribute is b_r, where b_r is the number of blocks in the file describing relation r. On a key attribute, the average cost is $b_r/2$, with a worst case of b_r. This is basically the type of operation that used to impede with proper indexing.

2) Binary search:

The DBMS might apply a binary search on an index with non-unique entries. A binary search, on equality, performed on a primary key attribute has a worst-case cost of □log (b$_r$)□. It can be significantly more effective than the linear search, for a generous number of records.

3) Search using a primary key index on equality:

In a DBMS With a B$^+$-tree index, an similarity comparison on a key attribute will have a worse -case cost of the height of the tree plus one to fetch the record from the data files. Similarity comparison on a non-key attribute will be the same excluding that various records may meet the condition, in which matter, we add the number of blocks including the records to the expanses.

4) Search using a primary index on comparison:

In a DBMS When the comparison operators (<, □, >, □) are used to fetching various records from a file sorted by the search attributes, the first record satisfying the condition is placed and the total blocks before (<, □) or after (>, □) is added to the cost of locating the first records.

5) Search using a secondary index on equality:

Fetching one record with an equality comparison on a key attributes or retrieves a set of records on a non-key attribute [6]. Furthermore, for a single record, the cost will be equal to the cost of locating the search key in the index file plus one for fetching the data record. For different multiple records, in a database the cost will be equal to the cost of locating the search key in the index file plus one block approaches for each data record retrieval, since the data file is not numbered on the search attributes.

B. Join Algorithms

The join algorithm can be developed in a different way. In terms of disk accesses, the joint operations can be very costly, so implementing and utilizing sufficient join algorithms is more important in minimizing a query's execution time [8]. Following are four types of join algorithms:

1) Nested-Loop Join:

This join also called nested iteration that uses one join input as the outer input table and one as the inner (bottom) input table. The outer loop accesses the outer input table row by row. Furthermore, the inner loop executed for each outer row and searches for matching rows in the inner input table. Hence, It abides of an inner for loop nested within an outer for loop [12].

2) Index Nested-Loop Join:

If the search ventures an index, it is called an index nested loops join. This algorithm is almost same as the Nested-Loop Join, besides an index file on the inner relation's join attribute is used versus a data-file scan on each index lookup in the inner loop is basically an equality selection for maintaining one of the selection algorithms. Let c be the cost for the lookup then the worst - case cost for joining rand sis $b_r + n_r * c$.

3) Sort –Merge Join:

The sort-merge join algorithm depends on the actual join condition, and it will depend on whether join attributes are keys or not. This algorithm can be used to process natural joins and equi-joins and requires that each relation be sorted by the common attributes between natural join and equi-joins [5].

4) Hash Join:

Hash joins require an equijoin predicate to comparing values from one table with values from the other table using the equals' operator. The hash join algorithm can be used to access natural joins and equi-joins. Further, the hash join maintains two hash table file structures to partition each relation's records into sets assuming identical hash values on the join attributes. Each and every relation is scanned and its relate hash table on the join attribute values is create.

C. Indexes Role

The execution time of many operations, such as select and join can be decrease by using indexes [7]. Let us see some of the types of index file structures and the roles they play in consuming execution time:

1) Dense Index:

A dense index is a file with pairs of keys and pointers for every record in the data file. Each key in this file is related with a particular pointer to *a record* in the sorted data files. The search key orders Data-file and each and every search key value have a different index record. This type of structure needs only a single seek to find the first occurrence of a set of contiguous records with the aspired search values [9].

2) Sparse Index:

A sparse index is a file with pairs of keys and pointers for every block in the data files. Every key in this file is attached with a particular pointer *to the block* in the sorted data files. The index search key orders Data-file and only some of the search key values have related index records. For each index record's, data-file pointer indicates to the first data-file record with the search key values. Further, while this structure can be less sufficient than a dense index to find the valued record, it needs less storage space and fewer expenses during insertion and deletion operations.

3) Primary Index:

If an index is built on ordering key-field of file, it is called Primary Index. Primary index defined on an ordered data file. The data file is ordered on a key field. The data file is sequenced by the attribute that is also the search key in the index file. Any primary index can be dense or sparse. A primary index is also considered to as an Index-Sequential File [5].

4) Secondary Index:

If an index is built on non-ordering field of file, it is called Secondary Index. Secondary index must be dense. The data file is sequenced by an attribute that is not same as the search key in the index file.

5) Multi-Level Index:

A Multilevel Index is an alteration of the secondary level index system. An index structure have 2 or more tiers of records where an upper tier's records indicate to related index records of the tier down. The bottom tier's index records possess the pointers to the data-file records. Furthermore, multi-level indices can be used to reduce the number of disk block reads required during a binary search in a DBMS.

6) Clustering Index:

If an index is creating on ordering non-key field of file it is called Clustering Index. Further, a two-level index structure where the records in the first level contain the clustering field value in one field and in the second level, a second field indicating to a block [of 2^{nd} level records]. The records in the second level have one field that indicates to areal data file record or to another 2^{nd} level block.

4.3.7. B^+-tree Index: B^+-tree Indices are an alternative to indexed sequential files. B^+-tree Index means multi- level index with a balanced-tree structure. Creating a search key value in a B^+-tree is proportional to the height of the tree maximum number of seeks required is □log(height)□. Although, this, on average is more than a single -level, dense index that requires only one seek. The B^+-tree structure has a unique advantage in that it does not need rearrangement; it is self-developing because the tree is kept arranged during insertions and deletions.

V. CHOICE OF EVALUATION PLANS

In a DBMS, the query optimization engine originates a set of candidate evaluation plans. Although, some will, in heuristic theory, creates faster, more sufficient executions. On the contrary, by previous historical summary, be more efficient than the theoretical model; this can very well be the case for queries dependent on the semantic nature of the data to be managed. Whereas, still others can be more efficient due to "outside agencies" such as competing applications, network congestion, on the same CPU, etc.

VI. CONCLUSIONS

In a DBMS, One of the major functional needs of a database system is its ability to process queries in convenient manner. It is basically true for huge, mission critical applications such as aeronautical applications, banking systems and weather forecasting, which can possess millions and even trillions of records. The basic need for faster and faster, "immediate" results never conclude. Hence, a big deal of research and resources is spent on creating and generating smarter, highly efficient query optimization engine for query optimization. Among them, some of the basic techniques of query processing and optimization have been presented and redefine in this paper.

REFERENCES

[1] D. Calvanese, G. DeGiacomo, M. Lenzerini and M. Y. Vardi. Reasoning on Regular Path Queries. ACM SIGMOD Record, Vol. 32, No. 4, December 2003.

[2] Henk Ernst Blok, DjoerdHiemstra and Sunil Choenni, Franciska de Jong, Henk M. Blanken and Peter M.G. Apers. Predicting the cost-quality trade-off for information retrieval queries: Facilitating database design and query optimization. Proceedings of the tenth international conference on Information and knowledge management, Pages 207 - 214.

[3] Andrew Eisenberg and Jim Melton. Advancements in SQL/XML. ACM SIGMOD Record, Vol. 33, No. 3, September 2004..

[4] AndrewEisenberg and Jim Melton. An Early Look at XQuery API for JavaTM (XQJ). ACM SIGMOD Record, Vol. 33, No. 2

[5] RamezElmasri and Shamkant B. Navathe. Fundamentals of Database Systems, second edition. Addison-Wesley Publishing Company.

[6] DonaldKossmann and Konrad Stocker. Iterative Dynamic Programming: A new Class ofQuery Optimization Algorithms. ACM Transactions on Database Systems, Vol. 25, No. 1, March 2000, Pages 43- 82.

[7] Chiang Lee, Chi - Sheng Shih and Yaw - Huei Chen. A Graph-theoritic model for optimizing queries involving methods. The VLDB Journal — The International Journal on Very Large Data Bases, Vol. 9,Issue 4, Pages327 - 343.

[8] Hsiao-Fei Liu, Ya - Hui Chang and Kun-Mao Chao. An Optimal Algorithm for Querying Tree Structures and its Applications in Bioinformatics. ACM SIGMOD Record Vol. 33, No. 2, June 2004.

[9] Reza Sadri, Carlo Zaniolo, Amir Zarkesh and JafarAdibi. Expressing and Optimizing Transactions on Database Systems, Vol. 29, Issue 2, Pages 282 - 318.

[10] Reza Sadri, Carlo Zaniolo, Amir Zarkesh and JafarAdibi. Optimization of Sequence Queries in Database Systems. In Proceedings of the twentieth ACM SIGMOD -SIGACT-SIGART symposium on Principles of database systems, May 2001, Pages 71 -81.

[11] Thomas Schwentick. XPath Query Containment. ACM SIGMOD Record, Vol. 33, No. 1, March 2004.

[12] AviSilbershatz, Hank Korth and S. Sudarshan.Database Systems Concepts,7th Editions.McGraw – Hill.

[13] Dimitri Theodoratos and WugangXu. Constructing Search Spaces for Materialized View Selection. Proceedings of the 7th ACM international workshop on Data warehousing and OLAP, Pages 112 - 121.

[14] Jingren Zhou and Kenneth A. Ross. Buffering Database Operations for Enhanced Instruction Cache Performance. Proceedings of the 2004 ACM SIGMOD international conference on Management of data, June 2004, Pages 191 202.