International Journal of Innovative Research in Advanced Engineering (IJIRAE) ISSN: 2349-2163
Issue 3, Volume 2 (March 2015) www.ijirae.com

Introduction to Techniques of Query Processing and Optimization

Dr.K. Kiran Kumar[1], T.M. Santhi Sri [2], Voruganti Vamshi priya [3],
Professor, Department of Electronics and Computers, K L University, Guntur, AP
BTech Student, dept. of Electronics and Computers. K L University, Guntur, AP.
BTech Student, dept. of Electronics and Computers. K L University, Guntur, AP.

Abstract: Query processing is a technique for getting information from the database in a reliable way. The performance of the database system depends on the query processing techniques we used in database systems. Normally, Databases must be able to respond to the users request in getting information. Whereas in large databases we see that that they may run on un-predictable and volatile environment then it becomes hard to produce database queries effectively based on the information that is available at the compile time. Then we introduce Query Optimization where it is used to get the results from the database in a timely manner. These efficient processing of results from queries is a main requirement in interactive environment where that involves large amounts of data. Query processing is very efficient in domains like distributed databases, Web, Global information System, Sims information media or which shows a great impact on their performance. This Paper gives a brief introduction and the techniques that are used in query processing and query optimization in the relational Databases.

Keywords used: Query Processing, Query Optimization, Database.

1.INTRODUCTION:

As we know that the basic part of database management system is query processing and query optimization. Now, the query processing methods for multiple dimensions are divided as follows:

1.Selection Query Model: Selection query model is a technique where tuples are directly assigned to the scores. And the other such technique is join query model and this model is used to compute over the attached results. The another such technique is aggregate query model where it is used mostly in ranking the tuples group.

2.Data access model: these methods are available in underlying data sources and are classified accordingly by the processing techniques. example: Some techniques are used for the availability of random access whereas the others are permitted to only sorted access.

3.Implementation-Level: These implementation techniques are divided to their level of integration with their DBMS. For example: some of the implementation techniques are implemented in top of the database systems of application layer, the others are in query operations.

4. Query and Data uncertainty: query processing involved in data and query models based on the division of their un-certainty. For Example: some may give the accurate answers while the other may give approximate answers with uncertain data.

5.Ranking Function: Here the techniques they impose on underlying score functions is based on the restrictions they impose on scores.

The steps involved to transform high- level to low-level queries are as follows:

The three phases where a query passes through DBMS processing is as follows:
1.Parsing and translation
2.Optimization
3.Evaluation
4.Execution

1.Parsing and translation:

This phase is used for translating the query into internal form. After translating to internal form it is then translated into relational algebra. It is used for checking syntax and also used for relations verification.

(or)

This phase job is to extract the raw tokens from strings of characters and they translate them to the data elements internally. It also helps to check the validity and also checks the syntax.

2.Optimization:

Here for query planning we used to generate an evaluation plan of lowest cost.

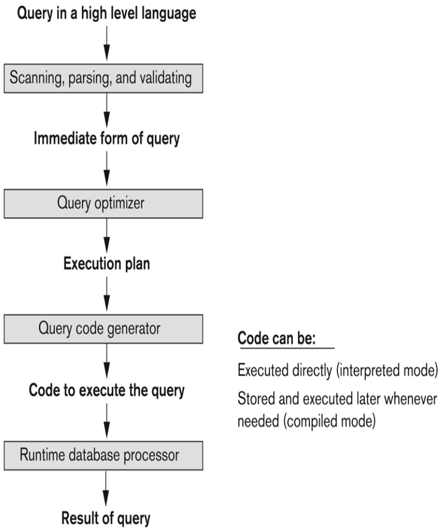

Query in a high level language
Scanning, parsing, and validating
Immediate form of query
Query optimizer
Execution plan
Query code generator
Code to execute the query
Runtime database processor
Result of query
Code can be:
Executed directly (interpreted mode)
Stored and executed later whenever needed (compiled mode)

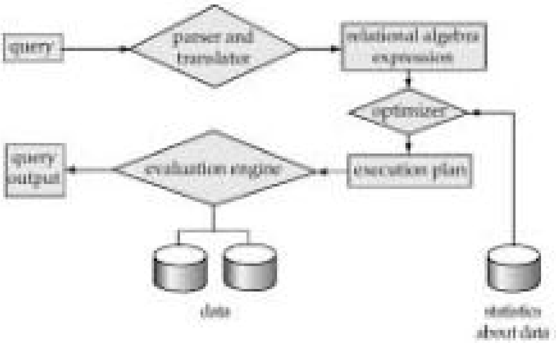
Figure 15.1
Typical steps when processing a high-level query.

3.Evaluation:

Here for executing the query we the query execution engine do a evaluation plan and executes that plan and also give back its answer to the query.

4.Execution:

Here after evaluation it return its answer to the query.



We have multiple methods for executing a query. In a sequential manner for processing a query the individual operations can be in parallel mode as a independent processor or as a pipelines or threads.

2.Query Optimizer:

Query optimizer used to give the most efficient results after executing the query using query plans.

Principles of query optimization:

To know the principles of optimization we should mainly identify the "basic building block of the system" which undergoes all the algorithms. Based on this way we have three aspects independently, They are:
1.QEP Generation
2.Search strategy
3.Cost Function

1.QEP Generation: This describes about the transformations, target language and also the source languages and define how to create the target language from initial query from source language. The target language normally reflect the aspects of run time where QEP is evaluated.

Example: Physical representation of hash tables, indexes which determines the usage of varies varieties of access operators. Operators implementing various join methods and indexes the QEP usage.

2.Search Strategy: User submitted query be evaluated by many other different QEP's which are used to create different choices to find a good candidate. So, we define how to search different set of possibilities by QEP's. This type is known as Search Strategy.

3.Cost Function: This is used for comparing of different QEP's and choosing the best which gives the best and accurate results.

Consider for example:
select salary
from balance
where salary<3000
This is translated to the following relational algebraic way:
σ salary < 3000 (Π salary (balance))
Π balance (σ salary<3000 (balance))
It is also represented in the following tree method:

```
σ salary <3000     Π salary
        |                  |
 Π salary            σ salary<3000
        |                  |
 balance              balance
```

3.General Aspects of Optimization

To provide a better understanding of what we mean by the term query optimization, the first subsection briefly outlines the difference between query optimization and query modification. The second subsection the discusses the three major aspects of query optimization that can be found in all optimization algorithms described in the literature.

Two Kinds of Query Optimization

The term query optimization has been used in the literature to describe different aspects of query processing. In general, we can distinguish two major phases of optimization that can be performs during the translation of the user-submitted query into an executable program. The first is to rewrite the modifications of the initial query such that we can expect an improved efficiency during the evaluation of the query.

The following is the example:
Example 1:
The following shows the examples of database

OFFC (Offc# , Name , Salary ,Dept#)
DEPT (Dept# , DeptName ,Mgr)

Each tuple in the relation OFFC describe an officer details by his on her officer number, name, and manager. The query is

```
SELECT Name , Mgr FROM OFFC, DEPT
WHERE OFFC .Dept# = Dept.Dept#
               And OFFC.Dept# < 1000
```

Can be modified as

```
SELECT Name, Mgr FROM OFFC, DEPT
 WHERE OFFC , Dept# = DEPT.Dept#
     And DEPT.Dept# < 1000
```

by showing Dept# of the DEPT relation instead in the OFFC relation.

By moving the restriction on the Dept# attribute from the EMP relation to the DEPT relation, the joining tuples of that relation are immediately restricted to those with Dept# <1000. The research literature reports on a wide variety of query modification schemes that range from syntactically rewriting a query to including semantic knowledge to simplify the initial query.

It is important to notice that query modification does not change the non-procedurally of the query. It is the responsibility of the second phase to translate the non-procedural to procedural. We call the output plan a query evaluation plan(QEP). Besides being procedural , the QEP also incorporates knowledge about the physical representation of relations in terms of base tables and indexes that can speed up the access to data ,and might include operators such as sorting or creating temporary tables. Furthermore, for operations like join, the QEP specifies what methods to use among different alternatives. Example 2 shows one possible QEP for the modified query of Example 1 assuming that an index is present on the attribute DEPT.Dept#.

Example 2:

The following QEP is one possibility for the evaluation of the query in Example 1. We use algebraic (i.e, relational-algebra like) operators as introduced in [Fre87] to express the QEP as follows:

```
 (PROJECT (OFFC .Name , DEPT .Mgr)
   (NLJOIN
    (FILTER (DEPT .Dept# <1000)
            (ACCESS DEPT))
      (GET OFFC
        (ACCESS (Dept# = DEPT .Dept#)
                 D_INDEX))))
```

Table DEPT is accessed before filtering out all tuples with a department value more or equal to 1000. The resulting set of tuples is the outer input stream for the nested-loop join operator. The inner input stream is generated b first accessing the D_INDEX using the join predicate before retrieving the tuple from the OFFC data table. Finally, the result is promoted into the attribute tuple.

Throughout this paper we concentrate on the second king of optimization and refer to it as query optimization, i,e. the translation of the non-procedural query specification into a procedural one. Furthermore it is important to notice that there exists a fundamental difference on how both phases are usually performed. Query modification rewrites the initial query in a straight forward manner without considering alternatives. On the other hand, query optimization explores different alternative QEPs for the same query and chooses one the best candidate for further execution process. To creating different alternatives and to compare with them and to select one from them in an efficient way, makes query optimization complicated.

It is worth nothing some additional differences between query modification which many researchers consider as “high-level optimization”. As the latter kind has been studied effectively from the time relational systems were built is understood well. The former kind of optimization has been studied less extensively: despite many results there is no agreed way how to structure or to perform query modification. On the other hand, because of it’s well defined nature and its well understood structure we undertake this task for query translation in the rest of this paper.

4. Measures of Query Cost

Firstly evaluating a query can be measured using the number of different resources we used, disk accesses, execution time of CPU to execute a query and a parallel or distributed database systems we use and also depends on the communication cost.
The time taken to respond to the query-evaluation plan and assuming that no activity is running on the computer would account for these costs and it is also used as a best method of cost of plan.

If we see in large database systems, accessing a disk are of most important cost, where accessing a disk is of slow compared to the operations which are done inside the memory.

Also now the speeds of CPU are also growing much faster than the speeds of disks. Thus the time taken on disk activity will continue to cross the total time taken to execute a query. Finally, estimating the time taken by the CPU is relatively hard when compared with the estimating disk-access cost. Therefore, most of the people take into account that the accessing disk cost takes a reasonable measure of the cost for a query-evaluation plan.

5. Query Algorithms

Queries are finally reduced to a number of file scan operations on the physical file structures [3, 4]. For each and every relational operation, there exist a different access paths where the particular records are needed. The execution engine for every query have a multitude of specialized algorithms which are designed to process a particular relational operation and path access combinations.

A. Selection Algorithms:

The Select operation must search the data files for records which meets the selection criteria. Some examples of simple i.e one attribute selection algorithms are as follows:

1. Linear search: Here each and every record from the files are read and compared with the selection criteria. For these searching the execution cost for non-key attribute is bk, where bk contains number of blocks in the file which represents the relations r. The average cost of key-attribute is bk/2,with a worst case of bk.

2. Binary search: A binary search equality is performed on a primary key attribute which is having a worst-case cost of [log (bk)]This is most efficient than linear search which is used for large number of data records.

3. Search using a primary index on equality:
An equality comparison with B+-index on a key attribute will have a worst -case cost of height of the tree and also includes retrieving the record from the data file. An equality comparison on a non-key attribute will be same except that to meet the condition we need multiple records, in that case, we need to add the number of blocks which are containing the records to the cost.

4. Primary index on comparison using search:
When the comparison operators (<,>) are used to retrieve different multiple records from a file which are sorted by search attribute, the first record which is satisfying the condition is located and the total blocks before (<, □) or after (>, □) is locating the first record and the cost is added to it.

B. Join query Algorithms:

This join query algorithm can be implemented in a many ways. Where in terms of disk accesses, the join operations can be very expensive and for implementing and utilizing efficient join query algorithms it is important in minimizing a execution time of query's . There are 4 well - known types of join query algorithms which are as follows:

1. Nested-Loop Join: It is having a inner for loop which is nested within an outer for loop.

2. Index Nested-Loop Join: This algorithm is also same as the Nested-Loop Join except an index file which is on the inner relation's join attribute is used with a data-file scan on each index lookup in the inner loop which is essentially an equality selection for utilizing one of the selection algorithms. Let cs be the cost for the lookup, then the worst -case cost for joining rand sis bk + nr * cs.

3. Sort –Merge Join: This algorithm is used to perform natural joins, equi-joins and it requires that each relation to be sorted by the common attributes between each relation.

4. Hash Join: This algorithm is used to perform natural joins and equi-joins. This hash join algorithm utilizes two hash tabled file structures i.e one for each relation which are partition to each relation's records into sets that containing identical hash values which are placed on the join attributes. Each relation is scanned and then its corresponding hash table are built on the join attribute values.

C. Indexes Role

Here we use the execution time of various operations such as select and join which are reduced by using indexes role. Now, Let us see some types of a structures of index file and the roles they used to play in reducing the execution time and overhead:

1. Dense Index: Here the Data-files are ordered by the search key and every and each search key worth features a separate index record. This structures needs one request to search out the set of contiguous records with desired search values among the primary prevalence itself.

2. Sparse Index: Here Data-file is ordered by the index search key and some of the search key values have corresponding index records. Here with the search key value each index record's data-file pointer points to the first data-file record. While this structure is less efficient than a dense index to find the desired records, then it requires less storage space, less overhead during insertion and deletion operations.

3. Primary Index: Here the data files are ordered by the attribute which is also a search key in the index file. Primary indices can be dense or sparse. This type is also referred to as an Index-Sequential File.

4. Secondary Index: Here the data file is ordered by attributes that are different from the search key in the index file. Here Secondary indices must be dense.

5. Clustering Index: A two-level index structure where it contains the records in the first level having clustering field value in one field and a second field pointing to a block i.e of 2nd level records in the second level. The records in the second level is having only one field that points to an actual data file record or to another 2nd level block.

6. B+-tree Index: Here Multi- level index is having a balanced-tree structure. Finding a search key value in a B+-tree is proportional to the height of the tree which are having maximum number of seeks and the number of seeks required is log(height). While having this on average it is more than a single -level, dense index that requires only one seek, the B-tree structure which has a distinct advantage in that and it does not require reorganization. It is kept balanced because it is self-optimizing during insertions and deletion operations.

6. Choice of Evaluation Plans

The query optimization engine is used to generates a set of candidate evaluation plans. Some will use the heuristic theory which produces a faster and more efficient execution. Then the others may produced by the prior historical results which are more efficient than the theoretical models, this can be used very well in case of queries dependent on the semantic nature of the data to be processed. Still others can be more efficient due to "outside agencies" such as network congestion, competing applications on the same CPU, etc.

7. Conclusion

The most important functional requirements of a database system is to process queries in a timely manner. This is particularly true in case of very large and mission critical applications such as weather forecasting, banking systems and aeronautical applications where they have millions and even trillions of records containing data and it becomes hard to store and to retrieve data from them. The need for faster and faster, "immediate" results never ceases. Some of the basic techniques and principles with examples of query processing and optimization is mentioned here in this paper.

References:

1. Cost-quality trade-off for information retrieving queries by Henk Ernst Blok, Sunil Choenni, Franciska de Jong, and Peter M.G. Apers.
2. Regular path queries and reasoning by D. Calvanese, G. DeGiacomo, M. Lenzerini and M. Y. Vardi published on December 2003.
3. Advancement in the SQL/XML by Andrew Eisenberg and Jim Melton published on September 2004.
4. Lookup at X-Query API for java by Andrew Eis enberg and Jim Melton.
5. Fundamentals of Database Systems by Ramez Elmasri and Shamkant.
6. Class of Query Optimization Algorithms by Donald Kossmann and Konrad Stocker published on March 2000.
7. Graph-theoritic model for optimizing queries methods by Chiang Lee, Chi - Sheng Shih and Yaw - Huei Chen.
8. An Optimal Algorithm for Querying Tree Structures by Hsiao-Fei Liu, Ya - Hui Chang published on june 2004.
9. Optimizing and expressing transaction mathods on database by Reza Sadri, Carlo Zaniolo, Amir Zarkesh and Jafar Adibi.
10. Optimizing of sequential queries in database systems by Reza Sadri, Carlo Zaniolo, Amir Zarkesh and Jafar Adibi published on may 2001.