Analysis of Execution Plans in Query Optimization

Dr. Sunita M. Mahajan, Mrs. Vaishali P. Jadhav

Abstract— The sequence in which the source tables are accessed during query execution is called a query execution plan. The process of selecting one execution plan from potentially many possible plans is referred to as query optimization. The query optimizer is one of the most important components of a query processor. The input to the optimizer consists of the query, the database schema (table and index definitions), and the database statistics. The output of the optimizer is a query execution plan, sometimes referred to as a query plan or just a plan. The goal is to eliminate as many unneeded tuples, or rows as possible. The paper describes the design of query optimizer with four basic phases such as Query Analysis, Index Selection Join Selection and Plan Selection.

Index Terms— Execution plan, index selection, join selection, query analysis, query optimization.

1 INTRODUCTION

There are three phases that a query passes through during the
DBMS' processing of that query:

- Parsing and translation
- Optimization
- Evaluation

Most queries submitted to a DBMS are in a high-level language such as SQL. During the parsing and translation stage, the human readable form of the query is translated into forms usable by the DBMS. These can be in the forms of a relational algebra expression, query tree and query graph.

The first step in processing a query submitted to a DBMS is to convert the query into a form usable by the query processing engine. High-level query languages such as SQL represent a query as a string, or sequence, or characters. Certain sequences of characters represent various types of tokens such as keywords, operators, operands, literal strings, etc. Like all languages, there are rules (syntax and grammar) that govern how the tokens can be combined into understandable (i.e. valid) statements. The primary job of the parser is to extract the tokens from the raw string of characters and translate them into the corresponding internal data elements (i.e. relational algebra operations and operands) and structures (i.e. query tree, query graph). The last job of the parser is to verify the validity and syntax of the original query string.

- *Dr. Sunita M. Mahajan is currently Principal, Institute of Computer Science, MET, Mumbai. E-mail: sunitamm@gmail.com*
- *Mrs. Vaishali P Jadhav is an Assistant professor in St Francis Instituteof Technology, Borivali. She is a research scholar in NMIMS University, E-mail:vaishaliwadghare@gmail.com*

In optimization phase, the query processor applies rules to the internal data structures of the query to transform these structures into equivalent, but more efficient 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 optimization engine.

The final step in processing a query is the evaluation phase. The best evaluation plan candidate generated by the optimization engine is selected and then executed. There can exist multiple methods of executing a query. Besides processing a query in a simple sequential manner, some of a query's individual operations can be processed in parallel—either as independent processes or as interdependent pipelines of processes or threads. Regardless of the method chosen, the actual results should be same [1][2].

2. QUERY OPTIMIZATION

The paper focuses on query optimization based on cost, which determines the query plan that will access the data with the least amount of processing time. The input to the optimizer consists of the query, the database schema (table and index definitions), and the database statistics. The output of the optimizer is a query execution plan, sometimes referred to as a query plan or just a plan.

The primary goal of the query optimizer is to find the cheapest access path to minimize the total time to process the query. To achieve this goal, the optimizer analyses the query and searches for access paths and techniques to minimize logical page access and physical page access. Disk I/O is the most significant factor in query cost. Therefore fewer physical and logical I/O, speeds up the query [3][4][5].

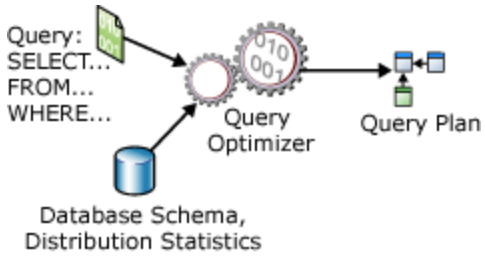

Fig. 1 Input and output to query optimizer

Query optimization has four basic steps:

- Query Analysis
- Index Selection
- Join Selection
- Plan Selection

2.1 Query Analysis

Query analysis phase look at each clause of the query and determine whether it can be useful in limiting how much data to be scanned. It checks whether the clause is useful as a search argument (SARG) or as a part of join criteria. A clause that can be used as a search argument is referred to as Surgable or Optimizable argument. It can make use of index for faster retrieval.

A SARG limits a search because it specifies an exact match, a range of values or a conjunction of two or more items joined by AND. It also contains a constant expression that acts on a column by using an operator. SURGABLE operators includes <, >, <=, >=, BETWEEN and sometimes LIKE operator. LIKE is surgable depending upon the wildcard used.

Consider the following surgable query:

Select emp_name, emp_designation, emp_salary from Employee where emp_name like 'JOHN%'

Consider the following non-surgable query:

Select emp_name, emp_designation, emp_salary from Employee where emp_name like '_JOHN'

The second query is non-surgable because the wildcard (_) at the beginning of JOHN prevents the use of index.

An expression that is not surgable can not limit the search. Every row has to be checked to determine whether it meets the conditions in the where clause. So index is not useful to non- surgable expressions. Typical non-surgable expressions includes NOT, !=, < >, ! >, NOT EXISTS, NOT IN and NOT LIKE etc.Surgable expressions use Clustered Index Scan and Non-Surgable expressions use Index Seek. Index scan is a vertical traversal of an index and index seek is a horizontal traversal of an index [6][7].

2.2 Index Selection

Index is a database object, which can be created on one or more columns (16 Max columns combination).When creating the index , optimizer will read the column(s) and forms a relevant data structure to minimize the number of data comparisons. The index will improve the performance of data retrieval and adds some overhead on data modification.

Conditions specified in where clause, i.e. the column used in where clause should have an index for searching as well as fast retrieval of data. If index covers all the referenced columns then index is called covering index and is one of the way to access data. Consider the following table:

Table 1: Customer Table

Cust_id	Cust_name	Cust_category
121	Johnson	Rich
122	Sachin	Rich
123	Amanat	Middle
124	Jenet	Poor
125	Sandeep	Rich
126	Ashish	Poor
127	Gaurav	Middle

Clustered Index:

The primary key created for the particular column will create a clustered index for that column. A table can have only one clustered index on it. When creating the clustered index, SQL server 2005 reads the column and forms a Binary tree on it. This binary tree information is then stored separately in the disc. With the use of the binary tree, the search for the particular row based on the indexed column decreases the number of comparisons to a large amount.

Consider the following clustered index:

Create clustered index CLUS_CUST_ID on customer (Cust_id)

The use of index on Cust_id will generate binary tree and reduce the number of comparisons while searching for a particular Cust_id in customer table[3][4].

Non-Clustered Index:

A non-clustered index is useful for columns that have some repeated values. A clustered index is automatically created when we create the primary key for the table. The non-clustered index has to be created separately. A table can have more than one Non-Clustered index. But, it should have only one clustered index that works based on the binary tree concept. Non-Clustered column always depends on the Clustered column in the database.Consider the following non-clustered index:

Create non-clustered index NCLUS_CUST_CATEGORY on customer (cust_category)

In the above scenario, the cust_category is non-clustered index and the cust_id is clustered index arranged in a binary

tree. Here, the Cust_category column with distinct values Rich, Middle, Poor will store the clustered index columns values along with it.

For example; Let us take only class value of Rich. The Index goes like this:

Rich:
121,122,125

Index Cost
For determining the selectivity of a SARG, the estimate cost of the access methods that can be used is calculated. Even if a useful index is present, it might not be used if the optimizer determines that it is not the cheapest access method. One of the main components of the cost calculation is the amount of logical I/O, which is the number of page accesses that are needed. Logical I/O is counted whether the pages are already in the memory cache or must be read from disk and brought into the cache. Pages are always retrieved from cache via a request to the buffer manager, so all reads are logical. If the pages are not already in the cache, they must be brought in by buffer manager.

In those cases, the read is also physical. Whether the access method uses a clustered index, one or more non-clustered indexes, a table scan, or another option determines the estimate number of logical reads[5]. The estimated cost in logical reads for different index structures is given below:

Table 2: The analysis of cost of data access for tables with different index structures

Access Method	Estimated Cost(Logical Reads)
Table Scan	Total number of data pages in the table
Clustered Index	Number of levels in the index + number of pages to scan
Non-Clustered index on a heap	Number of levels in the index + number of leaf pages + Number of qualifying rows The same data pages are often retrieved many times, so number of logical reads can be much higher than the number of pages in the table.
Non-Clustered index on a table	The number of levels in the index + with a clustered index number of leaf pages + the number of qualifying rows times the cost of searching for a clustered index key
Covering Non-clustered index	Number of level in the index + number of leaf index pages (qualifying row/rows per leaf page) The data page need not be accessed because all necessary information is in the index key.

2.3 Join Selection

Join Selection is the third step in query optimization. If the query is multi-table query or a self- join, the query optimizer evaluates join selection and selects join strategy with the lowest cost. It determines the cost using a number of factors, including expected number of reads and the amount of memory required. It can choose between three basic strategies for processing joins: nested loop joins, merge joins, and hash joins[5][6].

Nested Loop Join
Iteratively scanning the rows of one table and trying to match with second table using index is called nested loop join. If index is not available then hash join is used.

Consider the following join

```
Select dept_no, emp_id, emp_salary
from department dept join employee
empl on dept.dept_no = empl.dept_no
```

Pseudo-Code

```
Do(Until no more dept ows);
GET NEXT dept row
{
        begin
                // Scan empl, using an index empl.deptno
                GET NEXT empl row for given
        dept end

}
```

Merge Join
When two inputs are sorted on the join column then merge join is used. If the inputs are already sorted then less I/O is required to process a merge join if the join is one to many. A many to many merge join uses a temporary table to store rows instead of discarding them. If there are duplicate values from each input one of the inputs must rewind to start of the duplicates as each duplicate from other input is processed.

Pseudo- Code

```
GET one dept row and one empl row
DO( until one input is empty)
        IF dept_no values are equal
                Return values from both
                        rows
                GET next employee row
        ELSE IF empl.dept_no < dept.dept_no
                GET next employee row
        ELSE GET next department row
```

Hash Join
Hashing determines whether a particular data item matches an already existing value by dividing the existing data into groups based on some property. Hash Bucket contains the data with the same value. For finding the match of new data, hash bucket with existing data is to be checked. While processing hash joins, the smaller input is taken as build input. The buckets are actually stored as linked lists, in which each entry contains only columns from build input that are

needed. The collection of these linked lists is called the hash table [6][7]. Table 3.shows the analysis of basic strategies for processing joins.

Pseudo-Code

Allocate an Empty Hash Table
For Each Row in the Build Input
Determine the hash bucket by applying the hash function
Insert the relevant fields of the record into the bucket
For Each Record in the Probe Input
Determine the hash bucket
Scan the hash bucket for matches
Output matches
De-allocate the Hash Table

Table 3: Analysis of basic strategies for processing joins

Parameters	Nested Loop Joins	Merge Joins	Hash Joins
Name of Inputs	Join Inputs	Sorted Inputs	Build Input Probe Input
Need of Sorting	Not Necessary	Sorting saves merging time	Not Necessary
Need of Indexing	Indexing saves the searching time	Not Necessary	Not Necessary
Situation when it is used	Joining the small number of rows	Two join inputs are sorted on the join	Joining the large number of rows
Need of Equality	Not Necessary	Not Necessary	Must
Best use of the method	When index is on the join inputs	When join inputs are sorted	When smaller table fits entirely in the available memory.

2.4 Plan Selection

The plan selection is the fourth step in query optimization. It is based on the cost of a given plan, in terms of required CPU preprocessing and I/O, and how fast query will execute. Hence it is known as Cost-Based plan.The optimizer will generate and evaluate many plans and will choose the lowest cost plan. It is the plan which will execute the query as fast as possible and use the least amount of resources, CPU and I/O. The optimizer may choose a less efficient plan if it thinks it will take more time to evaluate many plans than to run a less efficient plan.

Suppose a query has a single table with no indexes and with no aggregates or calculations within these the query then rather than spending the time in calculating the optimal plan, optimizer will simply apply single, trivial plan to these types of queries. If the query is non-trivial, the optimizer will perform cost-based calculation to select a plan. For this purpose, optimizer relies on statistics of the execution plan [7].

3. CONCLUSION

Since SQL is declarative, there are typically a large number of alternative ways to execute a given query, with widely varying performance. When query is submitted to the database, the query optimizer evaluates different possible plans for executing the query and returns what it considers the best alternative.

Paper gives the overall phase-wise working of the query optimizer. In Query Analysis phase, optimizer will find search arguments, OR clauses and Joins. In Index Selection phase, optimizer chooses the best index for SARGs, ORs, Join Clauses and the best index to use for each table. In join reordering phase, optimizer evaluates join orders, computes cost and evaluates other server options for resolving joins. In plan selection, optimizer will select a plan suitable for given query.

So while processing the query, a cost based query optimizer has to find the cheapest access path to minimize the total time of execution of the query.

REFERENCES

[1] Ramez Elmasri and Shamkant B. Navathe. Fundamentals of Database Systems, second edition. Addison-Wesley Publishing Company, 1994.

[2] Avi Silbershatz, Hank Korth and S. Sudarshan. Database System Concepts, 4th Edition. McGraw-Hill,2002

[3] Henk Ernst Blok, Djoerd Hiemstra and Sunil Choenni, Franciska de Jong, Henk M. Blanken and Peter M.G. Apers.Predicting the cost-quality trade-off for information retrieval queries: Facilitatiing database design and query optimization. Proceedings of the tenth international conference on Information and knowledge management, October 2001, Pages 207-214.

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

[5] G. Antoshenkov, "Dynamic Query Optimization in RdbIVMS", Proc. IEEE Int'1. Conf on Data Eng., Vienna, Austria, April 1993,538.

[6] Inside Microsoft SQL Server 2000. http://flylib.com/books/en/2.257.1.137/1/

[7] K. Ono and G, M, Lehman, "Measuring the Complexity of Join Enumeration in Query Optimization", Proc. Int'1. Conf$ on Ve~ Large Data Bases, Brisbane, Australia, August 1990,31