2012 International Conference on Communication, Information & Computing Technology (ICCICT), Oct. 19-20, Mumbai, India

An Analysis of Execution Plans in Query Optimization

Dr. Sunita M. Mahajan
Principal, Institute of Computer Science
Mumbai Education Trust, Reclamation,
Bandra, India
sunitamm@gmail.com

Ms. Vaishali P. Jadhav
Research Scholar, NMIMS University.
Assistant Professor, St. Francis Institute of Technology,Borivali,India
vaishaliwadghare@gmail.com

Abstract—**A major task of query processor is to produce query results, as efficiently as possible. For efficient query result, query has to follow proper execution plan which minimizes the cost of query execution. The sequence in which source tables are accessed during query execution is very important. Query optimizer which is the core component of query processor gives the best execution plan from many possible plans for the same query. The paper is the extended version of previous work [12]. Paper describes four phases of query optimization along with their execution plan. It gives the analysis of access methods and different types of joins used in execution plan of the query.**

Keywords-Execution plan, query analysis, index selection, join selection.

I. Introduction

There are various phases of query processing .The query processor accepts SQL syntax, selects a plan for executing the syntax, and then executes the chosen plan. Following diagram gives working of query processor[1-5].

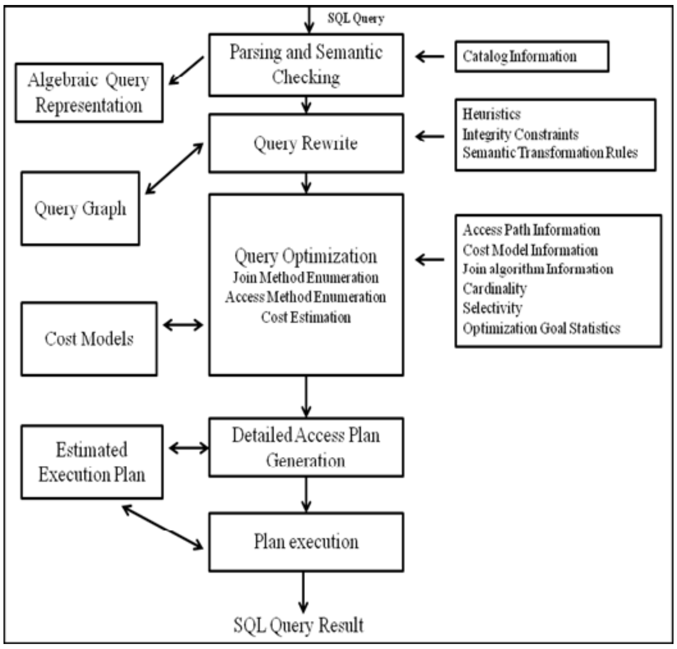


Figure1. Query Processing

Parsing and Semantic Checking:
When SQL query is to be processed, the database server uses parser to parse the query. It is used for syntactic and semantic checking of the query. It transforms the query into a parse tree which is the algebraic representation of the query [6-8].

Query Rewrite:
Simple queries skip this step. This step applies to the complex query which involves joins, predicates, grouping etc. This phase also represent the query as annotated parse tree. During this step, query undergoes iterative transformations according to heuristics used. Join elimination, predicate normalization, select operation before project operation and many transformation rules are applied in this phase [9].

Query Optimization
It consists of pre-optimization phase and enumeration phase. In pre-optimization phase, query is analyzed to find all predicates, indexes, joins used in the access plan. This phase also builds alternative plans for the respective query. In Enumeration phase, the optimizer enumerates the different possibilities of the access plans for the given query. Based on the use of join algorithm, cost estimation method, access method, selectivity used etc. the best access plan is determined[6-9].

Detailed Access Plan execution:
This phase takes the best plan and builds the graphical view of the corresponding query. Graphical view of the plan is available in Interactive SQL. Graphical View has tree like structure and each node is a physical operator implementing specific relational algebra operation [10-12].

Plan Execution:
The result of the query is computed using best execution plan built in previous phase [12].

In this paper, we are mainly concerned about query optimizer. The input to the optimizer consists of the query, the database schema (table and index definitions), and the database statistics. Execution plan is the output of query optimizer. Paper describes the analysis of execution plans of various queries. Execution plan of the optimizer is determined by query optimizer based on the following components.

- **Access Method**: It is the manner in which data is being accessed. It can be Table Scan or Index Scan etc.
- **Join Method**: It is the manner in which tables are joined with each other. It can be hash join or merge join etc.
- **Order of Join:** It is the sequence in which tables are joined.
- **Cardinality:** Depending upon the number of rows, the access method, or join algorithm is selected.
- **Join Type:** Execution plan also considers the join type such anti-join, semi-join etc.
- **Partition Pruning:** Query optimizer checks the partitions required for obtaining the query result.
- **Parallel Execution:** Query Optimizer checks whether each operation in the plan is being conducted in parallel or not. Data redistribution method used is right or not.

When dealing with query optimization, it would be necessary to test the execution plan for different execution strategies. The selection of best execution plan involves various factors such as selection of join algorithm, use of indexes, the order of executing relational algebra operators etc. The paper will describe all factors in detai[5-7]l.

The remaining paper is organized with basic steps of query optimization. Section II gives the execution plan for different queries in query analysis. Section III describes the next step of query optimization as Index Selection. Section IV gives different join algorithms used. Section V describes plan selection which is the last step of query optimization. Section VI gives the conclusion of the paper.

II. QUERY ANALYSIS

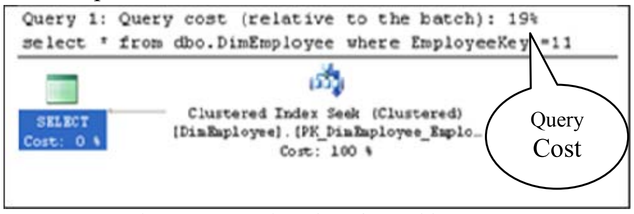
For query analysis we are using AdventureWorks2008R2 database in SQL Server 2008.We have seen the use of surgable or optimizable argument in previous version of this paper [12]. Query Cost of surgable query is much les than the query cost of non-surgable queries[6-8]. In the following example, the cost of surgable query is 19% and cost of non-surgable query is 81%.

Following table gives the expressions of surgable and non-surgable arguments.

Table 1. Suragble and Non-Surgable arguments

Surgable	Non-Surgable
X= 45	X<>45
x>65	X= 3 OR x=2
X IN(7,8,9)	X NOT IN(1,2,3)
X=z	X=y
X LIKE ‘Sat%’	X LIKE ‘%Sat’
X= 10-8	X+8 =10
X IS NULL	X IS NOT NULL

Consider the following surgable query,
select * from dbo.DimEmployee where EmployeeKey =11
Execution plan :



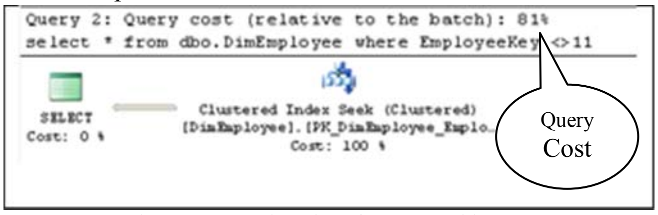
```
Query 1: Query cost (relative to the batch): 19%
select * from dbo.DimEmployee where EmployeeKey =11
```

SELECT
Cost: 0 %
Clustered Index Seek (Clustered)
[DimEmployee].[PK_DimEmployee_Emplo...
Cost: 100 %
Query Cost

Figure 2. Execution Plan of Surgable Query

Query Cost for executing a query = 19%

Consider the following non-surgable query,
select * from dbo.DimEmployee where EmployeeKey <>11
Execution plan:



```
Query 2: Query cost (relative to the batch): 81%
select * from dbo.DimEmployee where EmployeeKey <>11
```

SELECT
Cost: 0 %
Clustered Index Seek (Clustered)
[DimEmployee].[PK_DimEmployee_Emplo...
Cost: 100 %
Query Cost

Figure 3. Execution plan of Non-Surgable Query

Query Cost for executing a query = 81%

Query cost for executing a surgable query is much less than that of non-surgable query.

III. INDEX SELECTION

An index associated with table increases the speed of retrieval of rows from table[9-11]. There are two types of indexes: Clustered index and Non-Clustered index. Clustered Indexes sort and stores the data rows in the table based on the column selected as the key. When the table has a clustered index, the table is called clustered table. If a table has no clustered index, it is called as a heap which is an unordered structure. Consider the table customer and the following query which creates an index on Cust_id of Customer table.

Table 2.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

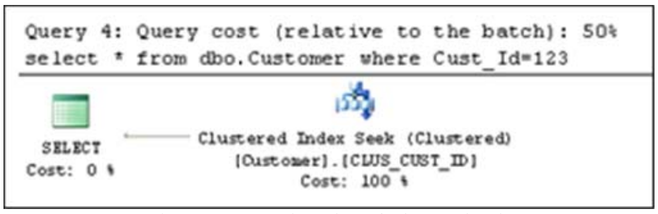
Consider the following queries,

create clustered index CLUS_CUST_ID on dbo.customer (Cust_Id)

Non-clustered index contains the non-clustered index key values and each key value entry has a pointer to the data row that contains key value. The following query creates non-clustered index on Cust_Category of Customer table.

create nonclustered index NCLUS_CUST_CATEGORY on dbo.customer (Cust_Category)

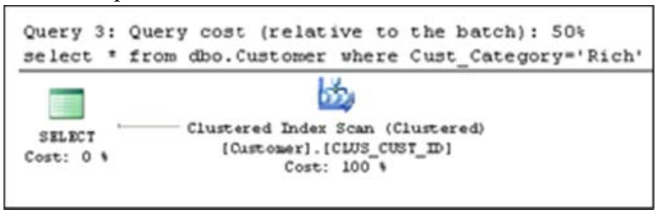
select * from dbo.Customer where Cust_Id=123
Execution Plan:



```
Query 4: Query cost (relative to the batch): 50%
select * from dbo.Customer where Cust_Id=123
```

Figure 4. Execution plan of Clustered Index

select * from dbo.Customer where Cust_Category='Rich'
Execution plan :



```
Query 3: Query cost (relative to the batch): 50%
select * from dbo.Customer where Cust_Category='Rich'
```

Figure 5. Execution plan of Non-Clustered Index

Non-Clustered column always depends on the Clustered column in the database. The Cust_category column with distinct values *Rich, Middle, Poor* will store the clusteredindex columns values along with it. So the Index goes like this, *Rich*: 121,122,125.

Following table gives the analysis of cost of data access for tables with different index structures.

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.

IV. Join Selection

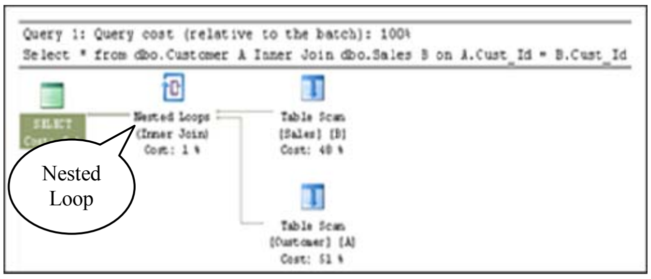
Join Selection is the third step in query optimization. If the query is multi-table query or a self- join, the query optimizer evaluates join selection and selects join strategy with the lowest cost. It determines the cost using a number of factors, including expected number of reads and the amount of memory required. It can choose between three basic strategies for processing joins: nested loop joins, merge joins, and hash joins [6][7].

A.Nested Loop Joins

It is mainly used for smaller tables. 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. There are 3 types of nested Join. The search scans an entire table or index is called a naive nested loops join. If the search exploits an index, it is called an index nested loops join. If the index is built as part of the query plan it is called a temporary index nested loops join. In temporary index nested loop join, index is destroyed after execution of the query.

Consider the following query,
Select * from dbo.Customer A Inner Join dbo.Sales B on A.Cust_Id = B.Cust_Id

Execution plan:



```
Query 1: Query cost (relative to the batch): 100%
Select * from dbo.Customer A Inner Join dbo.Sales B on A.Cust_Id = B.Cust_Id
```

Figure 6. Use of Nested Loop Join in Execution Plan

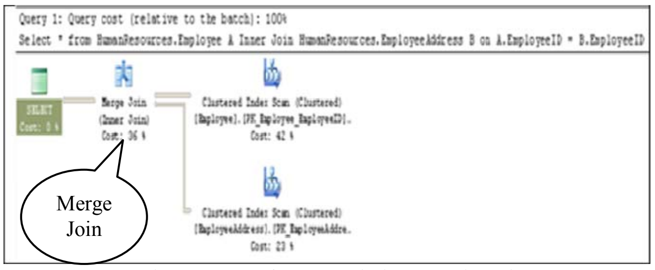
Both tables Customer (4 Rows)and Sales (3 Rows) are small, so they are entirely scanned and used nested loop join for performing join operation.

B. Merge Joins

Merge Join is considered to be more efficient for large tables with the keys columns sorted. 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. It is a fast joining method, but it can be expensive if sort operation is required.

Consider the following query,
Select * from HumanResources.Employee A Inner Join HumanResources.EmployeeAddress B on A.EmployeeID = B.EmployeeID

Execution Plan:



```
Query 1: Query cost (relative to the batch): 100%
Select * from HumanResources.Employee A Inner Join HumanResources.EmployeeAddress B on A.EmployeeID = B.EmployeeID
```

SELECT
Cost: 0 %
Merge Join
(Inner Join)
Cost: 36 %
Clustered Index Scan (Clustered)
[Employee].[PK_Employee_EmployeeID]..
Cost: 42 %
Clustered Index Scan (Clustered)
[EmployeeAddress].[PK_EmployeeAddre..
Cost: 23 %
Merge Join

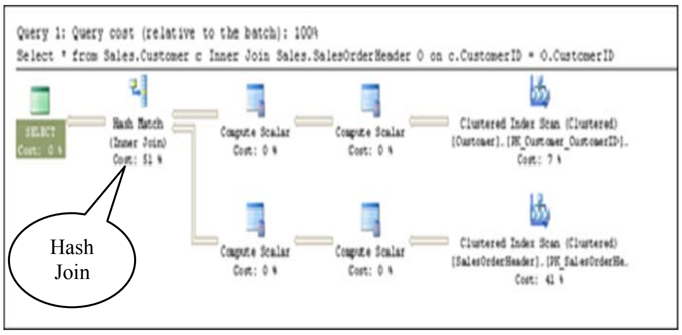
Figure7. Use of Merge Join in Execution Plan

Two tables Employee (290 Rows) and EmployeeAddress (290 Rows) are large. Both tables have clustered index scan and used merge join for implementing join operation.

C.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* [10][11].Hash joins are mainly useful in set-matching operations such as intersection, union, semi-join, full outer join. The set of columns in the equality predicate is called hash key which contributes in hash function. We have used AdventureWorksdatabase. From database we have used Sales.Customer and Sales.SalesOrderHeader tables for demonstrating hash join.

Consider the following query,
Select * from Sales.Customer c Inner Join Sales.SalesOrderHeader O on c.CustomerID = O.CustomerID
Execution plan:



```
Query 1: Query cost (relative to the batch): 100%
Select * from Sales.Customer c Inner Join Sales.SalesOrderHeader O on c.CustomerID = O.CustomerID
```

Figure 7. Use of Hash Join in Execution Plan

Hash Join is also suitable for large tables.Tables customer(19185 Rows) and SalesOrderHeader(Rows 31465)are large and used Hash key as CustomerID for join operation.

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 columns	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.

V. 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 [11].

VI. 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] AviSilbershatz, Hank Korth and S. Sudarshan. Database System Concepts, 4th Edition. McGraw-Hill, 2002

[3] 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: 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 JafarAdibi. 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 RdblVMS”, Proc. IEEE Int ‘1. Conf on Data Eng., Vienna, Austria, April 1993,538.

[6] C. Mohrm, D. Haderle, Y. Wang, and J. Cheng, “Single Table Access Using Multiple Indexes: Optimization, Execution and Concurrency Control Techniques”, Lecture Notes in Comp. Sci. 416 (March 1990), 29, Springer Verlag,

[7] K. Ono and G, M, Lehman, “Measuring the Complexity of Join Enumeration in Query Optimization”, Proc. Int ‘1. Con$ on Ve~ Large Data Bases, Brisbane, Australia, August 1990,314.

[8] G. Graefe and W. J. McKenna, “The Volcano Optimizer Generato~ Extensibility and Efficient Search”, Proc. IEEE Int ‘1. Con$ on Data Eng., Vienna, Austria, April 1993,209.

[9] W. Hasan and H. Pirahesh, “Query Rewrite Optimization in Starburst”, Comp. Sci. Res. Rep., SanJose, CA, August 1988.

[10] T.Sellis, ”Multiple query optimization”, IEEE transactions on knowledge and data Engineering , Vol-2, June-1990

[11] K. Shim, T.Sellis ,D.Nau, “Improvements on algorithms for multiple query optimization“ ,IEEEtransactions on knowledge and data Engineering , 12, 1994, pp.197-222.

[12] Sunita M. Mahajan, Vaishali P. Jadhav,”Analysis of Execution Plans in query optimization”, International Journal of Scientific & Engineering Research, Volume 3, Issue 2, February-2012 , ISSN 2229-5518

Dr. SunitaMahajan is currently Principal, Institute of Computer Science, MET, Mumbai. She worked in Bhabha Atomic Research Centre for 31 years. She obtained Ph D in Parallel Processing in 1997 from SNDT Women University and M Sc degree from Mumbai University in Physics in 1966. She is a member of Indian Women Scientists Association, Vashi. Her research areas are Parallel Processing, Distributed Computing, Data mining and Grid Computing.

Vaishali P Jadhav is an Assistant professor in St Francis Institute of Technology, Borivali. She is a research scholar in NMIMS University, Mumbai. She obtained Master’s degree in Computer Engineering from ThadomalShahani College of Engineering, Mumbai. She is a member of IEEE and ISTE. Her research areas are Database Management, Advanced Databases, Distributed Computing, Operating Systems, and Artificial Intelligence.