iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 7 **Issue:** III **Month of publication:** March 2019

DOI: http://doi.org/10.22214/ijraset.2019.3021

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 6.887
Volume 7 Issue III, Mar 2019- Available at www.ijraset.com

A Review on Query Optimization in Generic RDBMS

Purushottam Patel
Ph.D Research Scholar, Kalinga University, Raipur INDIA

Abstract: ***In Database system, user makes query will be responded by the DBMS. Generally, there are varities of method for computing which response of the given query. It is the responsibility of the query processor to transform the query as entered by the user into an equivalent query that can be computed more efficiently. Query optimization is the process to find a good strategy or best strategy evaluation plan for processing a query. The Objective of this research is to discuss about technique used by the oracle for query and to present a comparative study of the various costs involve executing join and the LIKE predicate based queries. To present comparative study an empirical study is done on Oracle 9i and MS Access 2007.***
Keywords: Database, optimization, command, generic, RDMS

I. INTRODUCTION

Microsoft SQL Server is a relational database management system, having MS-SQL and Transact-SQL as chief structured programming languages. They depend on relational algebra which is primarily worn for data insertion, modifying, deletion and retrieval, as well as for data access operations. The issue with receiving the probable output is controlled by the management system which has the objective of searching the best implementation plan, this procedure being known as optimization.

The most frequently worn queries are those of data retrieval through SELECT command. We have to take into concern that not only the select queries need optimization, but also other objects, like: index, study or figures.We consider the following issues as being liable for the stumpy performance of a Microsoft SQL Server system. After optimizing the hardware, the operating system and then the SQL server settings, the main factors which influence the tempo of implementation are:

1) Missing indexes;
2) Inexact figures;
3) Badly written queries;
4) Deadlocks;
5) T-SQL operations which do not depend on a single group of output (cursors);
6) Excessive fragmentation of indexes;
7) Frequent recompilation of queries.

These are only a few of the factors which can negatively manipulate the performance of a database.

A. Missing Indexes

This particular factor influences the most SQL Server's performance. When missing indexing of a table, the system has to go step by step through the whole table in order to get the desired value. This takes to congestion RAM memory and CPU, thus significantly enhancing the time implementation of a query.

More than that, deadlocks can be formed when for instance, session number 1 is running, and session number 2 queries the same table as the first session. Let's consider a table with 10 000 lines and 4 columns, among which a column named ID is automatically incremented one by one.

Table 1.1. Running a simple query to retrieve a row in a table

With clustered index (execution time / query plan)			Without clustered index (execution time / query plan)		
Time Statistics			Time Statistics		
Client processing time	5	→ 5.0000	Client processing time	5	→ 5.0000
Total execution time	327	→ 327.0000	Total execution time	327	→ 327.0000
Wait time on server replies	322	→ 322.0000	Wait time on server replies	322	→ 322.0000

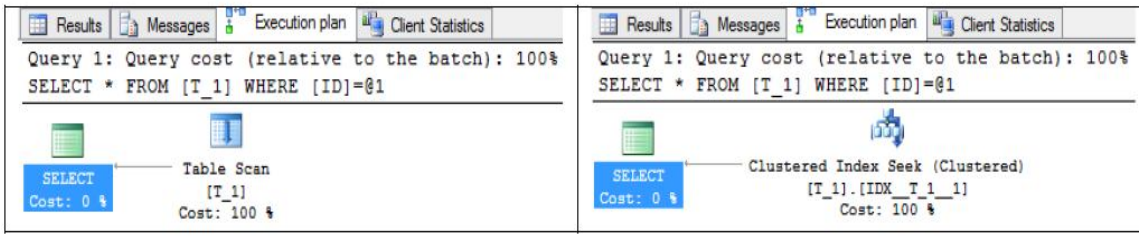


Table 1.2. Running a 2 table join query

With clustered index (execution time / query plan)			Without clustered index (execution time / query plan)		
Time Statistics			Time Statistics		
Client processing time	6	→ 6.0000	Client processing time	6	→ 6.0000
Total execution time	16	→ 16.0000	Total execution time	16	→ 16.0000
Wait time on server replies	10	→ 10.0000	Wait time on server replies	10	→ 10.0000

II. QUERY OPTIMIZATION IN GENERIC RDBMS

The query optimizer is responsible for generating the input for the execution engine. It takes a parsed representation of a SQL query as input and is responsible for generating an efficient execution plan for the given SQL query from the space of possible execution plans.

Relational database management systems (RDBMSs) let users specify queries using high level declarative languages such as SQL. Users define their desired results without detailing how such results must be obtained. The query optimizer, a component of the RDBMS, is then responsible for finding an efficient procedure, or execution plan, to evaluate the input query. For that purpose, the optimizer searches a large space of alternative execution plans, and chooses the one that is expected to be evaluated in the least amount of time.

Once the optimizer produces the execution plan estimated to be the best plan, the execution engine of the RDBMS provides an environment in which this plan is evaluated. State-of-the-art query optimizers search the space of alternative execution plans in a cost-based manner. Conceptually, modern optimizers assign each candidate plan its estimated cost (i.e., the expected amount of resources that the execution engine would require to evaluate the candidate plan), and choose the plan with the least expected cost for execution.

Query optimization is of great importance for the performance of a relational database, especially for the execution of complex SQL statements. A query optimizer determines the best strategy for performing each query. The query optimizer chooses, for example, whether or not to use indexes for a given query, and which join techniques to use when joining multiple tables. These decisions have a tremendous effect on SQL performance, and query optimization is a key technology for every application, from operational systems to data warehouse and analysis systems to content-management systems.

The query optimizer is entirely transparent to the application and the end-user. Because applications may generate very complex SQL, query optimizers must be extremely sophisticated and robust to ensure good performance. For example, query optimizers transform SQL statements, so that these complex statements can be transformed into equivalent, but better performing, SQL statements. Query optimizers are typically ‘cost-based’. In a cost-based optimization strategy, multiple execution plans are generated for a given query, and then an estimated cost is computed for each plan. The query optimizer chooses the plan with the lowest estimated cost.

Oracle’s optimizer is perhaps the most proven optimizer in the industry. Introduced in 1992 with Oracle7, the cost-based optimizer has been continually enhanced and improved through almost a decade’s worth of real-world customer experiences. A good query optimizer is not developed in a laboratory based on purely theoretical conjectures and assumptions; instead, it is developed and honed by adapting to actual customer requirements.

Oracle’s query optimizer has been used in more database applications than any other query optimizer, and Oracle’s optimizer has continually benefited from real-world input.

Oracle’s optimizer consists of four major components:

A. SQL Transformations

Oracle transforms SQL statements using a variety of sophisticated techniques during query optimization. The purpose of this phase of query optimization is to transform the original SQL statement into a semantically equivalent SQL statement that can be processed more efficiently.

B. Execution Plan Selection

For each SQL statements, the optimizer chooses an execution plan (which can be viewed using Oracle's EXPLAIN PLAN facility or via Oracle's "v$sql_plan" views). The execution plan describes all of the steps when the SQL is processed, such as the order in which tables are accessed, how the tables are joined together and whether tables are accessed via indexes. The optimizer considers many possible execution plans for each SQL statement, and chooses the best one.[1]

C. Cost Model and Statistics

Oracle's optimizer relies upon cost estimates for the individual operations that make up the execution of a SQL statement. In order for the optimizer to choose the best execution plans, the optimizer needs the best possible cost estimates. The cost estimates are based upon in-depth knowledge about the I/O, CPU, and memory resources required by each query operation, statistical information about the database objects (tables, indexes, and materialized views), and performance information regarding the hardware server platform.

The process for gathering these statistics and performance information needs to be both highly efficient and highly automated.

D. Dynamic Runtime Optimization

Not every aspect of SQL execution can be optimally planned ahead of time. Oracle thus makes dynamic adjustments to its query-processing strategies based on the current database workload. The goal of dynamic optimizations is to achieve optimal performance even when each query may not be able to obtain the ideal amount of CPU or memory resources.

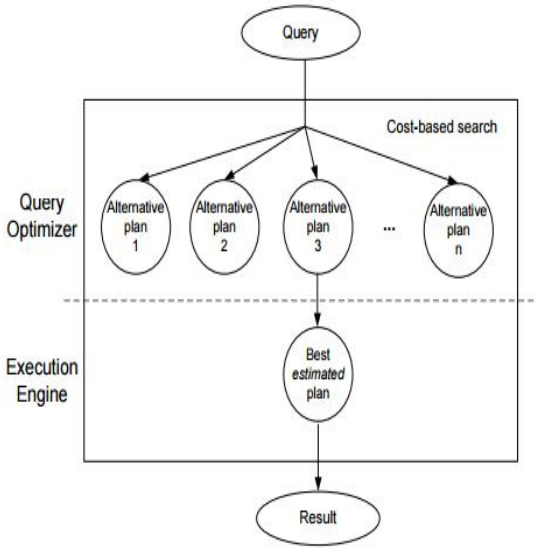


Fig. 3 The RDBMS chooses efficient execution plans to evaluate input queries.

Oracle additionally has a legacy optimizer, the rule-based optimizer (RBO). This optimizer exists in Oracle Database 10g Release 2 solely for backwards compatibility. Beginning with Oracle Database 10g Release 1, the RBO is no longer supported.

The vast majority of Oracle's customers today use the cost based optimizer. All major applications vendors (Oracle Applications, SAP, and People soft, to name a few) and the vast majority of recently built custom applications utilize the cost-based optimizer for enhanced performance, and this paper describes only the cost-based optimizer.

III.EXECUTING SQL QUERIES

A. The input query, treated as a string of characters, is parsed and transformed into an algebraic tree that represents the structure of the query. This step performs both syntactic and semantic checks over the input query, rejecting all invalid requests.

B. The algebraic tree is optimized and turned into a query execution plan. A query execution plan indicates not only the operations required to evaluate the input query, but also the order in which they are performed, the algorithm used to perform each step, and the way in which stored data is obtained and processed.

C. The query execution plan is evaluated and results are passed back to the user in the form of a relational table.

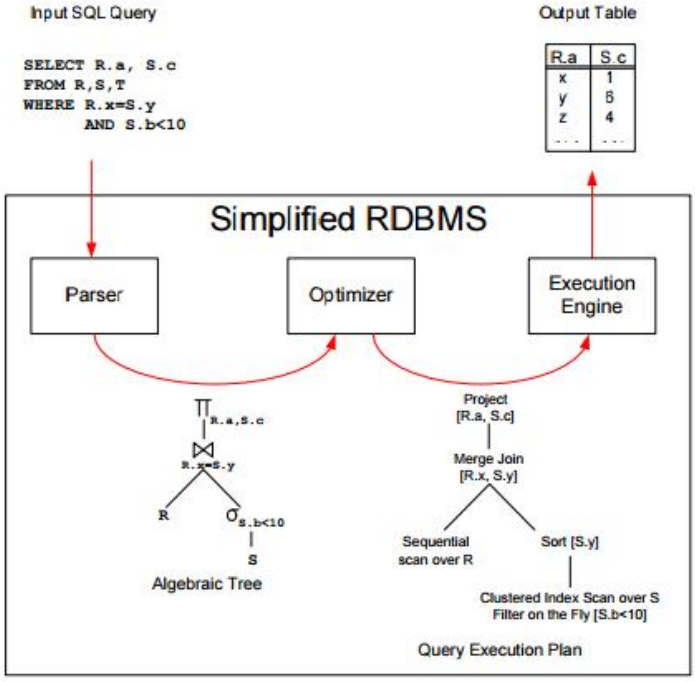
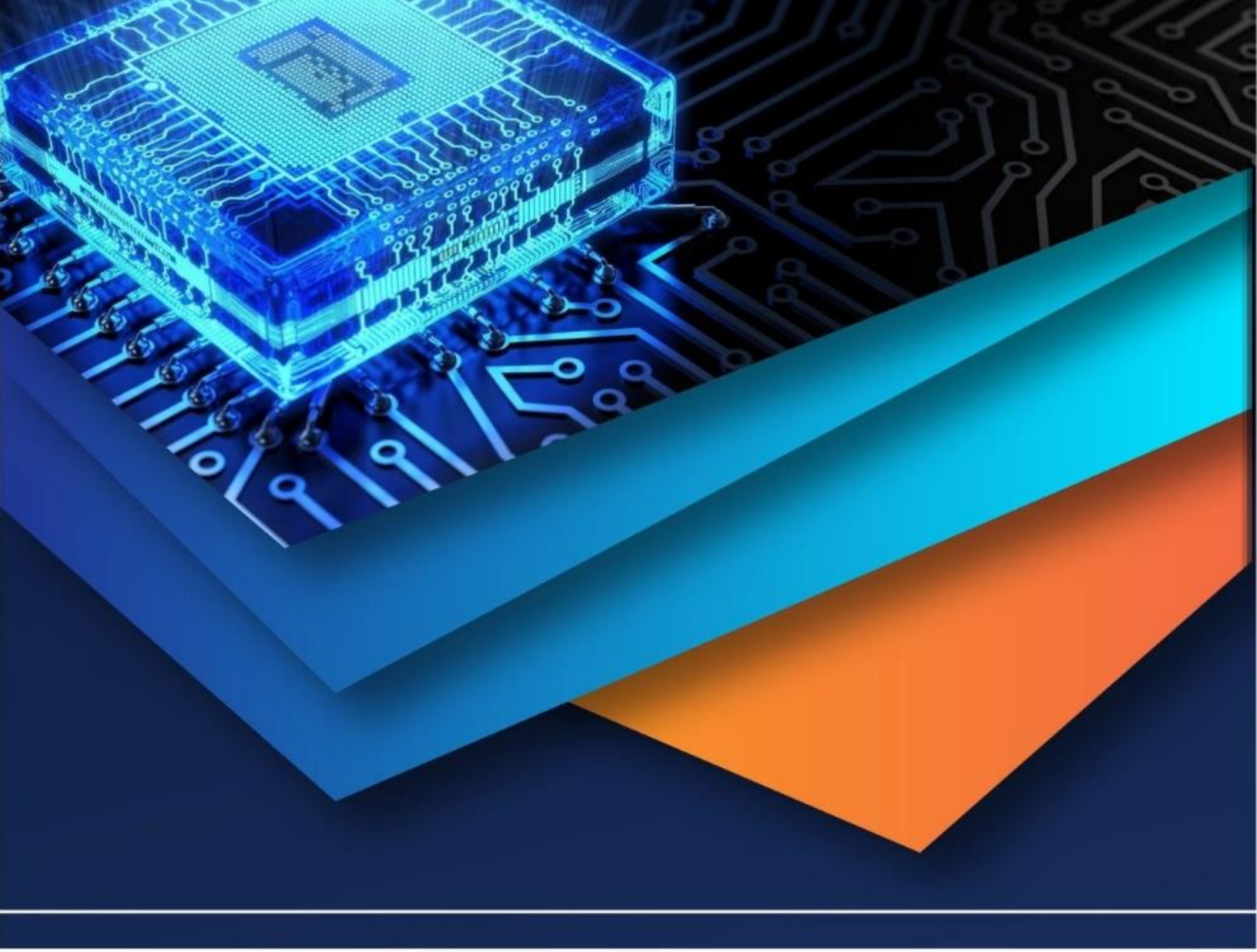


Fig. 4: Executing SQL queries in a relational database system

REFERENCES

[1] M. M. Astrahan et al. System R: A relational approach to data management. ACM Transactions on Database Systems, 1(2):97{137, June 2011.
[2] G. Antoshenkov. Dynamic query optimization in Rdb/VMS. In Proc. IEEE Int. Conference on Data Engineering, pages 538{547, Vienna, Austria, March 2013.
[3] K. Bennett, M. C. Ferris, and Y. Ioannidis. A genetic algorithm for database query optimization. In Proc. 4th Int. Conference on Genetic Algorithms, pages 400{407, San Diego, CA, July 2011.
[4] P. A. Bernstein, N. Goodman, E. Wong, C. L. Reeve, and J. B. Rothnie. Query processing in a system for distributed databases (SDD-1). ACM TODS, 6(4):602{625, December 2011.
[5] R. Cole and G. Graefe. Optimization of dynamic query evaluation plans. In Proc. ACM-SIGMOD Conference on the Management of Data, pages 150{160, Minneapolis, MN, June 2010.
[6] S. Christodoulakis. Implications of certain assumptions in database performance evaluation. ACM TODS, 9(2):163{186, June 2010.
[7] S. Christodoulakis. On the estimation and use of selectivities in database performance evaluation. Research Report CS-89-24, Dept. of Computer Science, University of Waterloo, June 2009.
[8] Graefe and D. DeWitt. The exodus optimizer generator. In Proc. ACM-SIGMOD Conf. on the Management of Data, pages 160{172, San Francisco, CA, May 2012.
[9] C. Galindo-Legaria, A. Pellenkoft, and M. Kersten. Fast, randomized join-order selection - why use transformations? In Proc. 20th Int. VLDB Conference, pages 85{95, Santiago, Chile, September 2010.
[10] G. Graefe and B. McKenna. The Volcano optimizer generator: Extensibility and efficient search. In Proc. IEEE Data Engineering Conf., Vienna, Austria, March 2013.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY