Exploring Query Optimization Techniques in Relational Databases

Majid Khan and M. N. A. Khan

SZABIST, Islamabad, Pakistan
engrmajidkhan@gmail.com,mnak2010@gmail.com

Abstract

In the modern era, digital data is considered as the more valuable asset of an organization, and the organizations assign more significance to it than the software and hardware assets. Database systems are computer-based record keeping systems, which have been developed to store data for efficient retrieval and processing. One particular approach is the relational databases in which all the information is stored in rows and columns in a series of interconnected tables, and a snippet of structured code called query is used to interact with these database tables. Database management involves indexing of data by tagging information based on some common factors and corresponding criteria. Performance of SQL query against a production database eventually becomes an issue sooner or later. The time-intensive queries not only degrade performance of servers and applications by consuming substantial system resources, but can also result in table locking and data corruption. Therefore, query optimization becomes necessary to prevent performance degradation. Query optimization primarily means selection, followed by sequencing in specific order, of the different SQL clauses to formulate an efficient query from the multiple query plans by drawing a comparison of the query plans based on the cost of the resources involved and the response time. The objective of query optimization is to provide minimum response time and maximum throughput (i.e., the efficient use of resources). This paper reviews different query optimization techniques and approaches discussed in the contemporary literature for both centralized and distributed databases. The paper also highlights merits of these techniques by critically analyzing them.

Keywords: *SGA, PGA, Advanced Database Management System, DBCC, Automatic Storage Management, VDSI, Query Optimization*

1. Introduction

Query evaluation and its optimization belong to a broader class of research, and much work has been performed in this field since the early 1970s [7]. Query processing passes through three steps: decomposition of query, its optimization and execution. During the decomposition process, parsing and binding are performed on the SQL query. During parsing, the query validation in terms of syntax, semantic and authorization is checked. Binding makes sure that all objects used in the query do exist and bind every table and column name in the parse tree to their corresponding objects of the system catalogue. The output of the first step is an algebraic tree. This tree is then passed to query optimizer to generate a cost-effective plan for the specified SQL query from the possible search spaces which have many candidate execution plans. In the third step, the query with the efficient plan is executed making a tree called *operator tree* or *physical operator tree*. The main purpose of query optimization is to optimize cost in terms of minimizing the response time and maximizing the throughput by the proficient use of resources [8].

There are two main components of DBMS to evaluate and optimize the query: the query optimizer and the query execution engine. The execution engine uses physical operators like SORT, NESTED LOOP JOIN, MERGE JOIN and INDEX SCAN, etc. to take input and produce the required output. These physical operators are the building blocks which make the SQL query execution possible. These operators construct a tree called parsed tree, which represents the flow of data from one operator to the others in the form of edges moving back and forth to the nodes. Query optimizer takes a parsed tree of the SQL query as an input from the execution engine and produces an best possible or close to optimal execution plan out of the possible execution plans for the given query based on the least resource consumption. For a given query, there are many logical algebraic representations and there are many choices of physical operators to implement these logical representations in addition to the variation of response time of these plans. Therefore, it is obviously not an easy task for an optimizer to generate an optimal plan [7].

Query optimizer being an important module of a DBMS has a greater impact on the database performance. It analyzes a number of candidate plans generated for a given query which have equivalent output but having different resource costs. It selects an efficient plan out of these candidates' plans having the least cost [1]. Although much work has been performed in query optimization, but cardinality estimation to manage optimization time and effective cost estimation with respect to the server state are still the challenging problems [11].

The optimizer should be capable to handle complex and large data by adopting some specific search strategy to counter the problems that an optimizer may face. Genetic strategies can be used to solve the optimization issues such as joins which is a challenging part for an optimizer in making efficient plan. Genetic strategies are mainly used to focus these problems to improve the query optimization [3].

Different methods for query processing and optimization are used according to the data size and the complexity of queries [10]. As the amount of data increases, queries also become complex to interact with silos of huge data, thus optimization of these queries becomes essential due to the involvement of different sub-queries, joining and grouping [2]. Besides, it also becomes more difficult to cache the result at client side when there is a massive volume of data. In such scenarios, efficient paging query concept is used to handle the situation when data size becomes huge (*e.g.*, in terabytes) [5].

In relational databases, the result of some queries depends on some unknown condition to be known first. To deal with such situations, the sub-query concept is used. Sub-query enhances the expressive and declarative capabilities of SQL. It has much importance to deal with the optimization of sub-queries, particularly, when they are in the form of correlated sub-queries, *i.e.*, when sub-queries have reference from outer query. In decision support systems and OLAP, sub-queries are widely used and different query transformation techniques are used to optimize these queries [4].

Distributed databases which are the logical collection of multiple databases ensure high performance and availability. User can access the system as a single unique system from anywhere without any concern of the data location [10]. In distributed systems, computers are physically distributed and are connected to a centralized database system. The data is distributed among different computers having their own local application. User simply queries the data without knowing its actual physical location. The main purpose of the distributed query processing and optimization is to devise query processing strategy and to select a least expensive query. There are two basics objective of query optimization in distributed systems. First one is to achieve minimum total cost including CPU, I/O and data

transmission cost. Second is to achieve the minimum response time by increasing the possibility of parallel processing which speeds up the overall query processing [9].

2. Literature Review

Ioannidis [1] focused the optimization of a query in centralized database management systems. According to the author, the process of query optimizations goes by two key stages: the rewriting stage and the planning stage. Various query optimizer components are then explored in these stages. The rewriter module in the rewriting stage performs transformations for a given query and produces an efficient query. Planner which is the basic module of planning strategy performs various search strategies mechanism. It explores plans identified by the algebraic space. Method-structure space modules evaluate these plans using the cost derived from the Size-Distribution Estimator and Cost-Model module. Out of which, plan with the least cost is selected. Algebraic space module operators considering the plan of a query and identifies their order in form of trees or relational algebraic formulas. Logical operator trees are then related to physical available join methods by Method-Structure Space module. Cost model determine cost for the access plans using an arithmetic formula. Sampling or statistical approximations can be done to get query result and their frequency. Author has focused on histogram method where each attributes values are distributed into chunks or buckets. However, there are several issues in the field of query optimizations that necessitates making the query optimizers architecture in a generalized way to handle every type of query either simple or complex. Also, some other advanced issues to handle in future are dynamic, parallel, distributed, semantic, object oriented and aggregate query optimization.

Li *et al.*, [2] discussed optimization methods to enhance the query optimization to get an optimal plan. The main issue using sub-queries is the intra query redundancy in which the sub-query has the same tables and conditions that are for the outer query. However, it degrades the query performance if the query has correlated nested queries. One solution to improve the performance is the query un-nesting, *i.e.*, writing the nested queries into flat forms to reduce the number of sub-queries. The authors proposed some heuristic strategies for enhancing query processing. The first strategy proposed is to perform the selection operations first in order to limit the number of rows/tuples. The second proposed method is to limit the number of columns by performing projection operations. Thirdly, perform the operations with the smaller or simple join first if there are consecutive joins in the query. And finally save the result for the same expression for future use.

Chande *et al.*, [3] focused the join ordering problem in relational database and used genetic algorithm (GA) to face the problem of efficient selection of join ordering for making an optimal plan by an optimizer. The queries used for genetic query optimizer (GQO) are executed in different environment to compare their performance. The execution time for all queries is compared with the proposed GQO. The authors performed experiments to compare GQO with PostgreSQL, DB2 and MySQL. The experimental results show that genetic strategy for query optimization is a good approach for complex queries optimization and generates better result than standard RDBMS optimizers for large figure of joins e.g., DB2 or MySQL. GQO is more suitable for fewer joins but it is little expensive in case number of joins exceed the figure of 20.

Bellamokanda *et al.*, [4] presented different query transformations techniques in Oracle relational database. Oracle uses many transformation techniques, *e.g.*, sub-query un-nesting, group by, view merging, common sub-expression elimination, join predicate pushdown, semi-joins, anti-joins, star transformation and OR expression. Generally, sub-query coalescing, sub-query removal using windows functions and NULL-AWARE ANTI-JOIN methods are used to improve execution time of queries. Query transformations in Oracle use either cost

based approach or heuristic approach to select an optimal execution plan. To avoid self-joins in multiple query blocks, Oracle uses windows function for efficient execution and optimization. In addition, Oracle uses PARTITION BY key or ORDER BY key for sorting data to compute window functions. In sub-query coalescing technique, two sub-queries are coalesced into one single sub-query and is used to reduce multiple table access and multi joins operations into single table access and single join operation. Sub-query coalescing works like a filter on the tables of the outer query. In Oracle, coalescing sub-queries appear in conjunction or disjunction. When two sub-queries are of the same type, *e.g.*, both use either EXIST or NOT EXIST then sub-query coalescing result in the removal of one query. Sub-query removal using windows function technique replaces the sub-queries with windows functions to reduce the number of table access and joins to improve query efficiency. A regular Anti-join is exactly opposite to inner join. Since in SQL, any relational comparison with null always results in a null value, so there should be some strategy to deal this situation. NULL-AWARE ANTI-JOIN concept is used to handle null values in anti-join operations.

Sun *et al.*, [5] proposed a paging query solution to for a large-scale data to improve query efficiency and overall application performance. Through this method, all the data satisfying all the query conditions are first placed at server's memory and then only the part of the data needed by the client is passed to the client. The proposed method is in contrast to the traditional method of paging query in which all the data is stored at client memory which is obviously not an efficient way while handling massive data as it may lead to choke the client system resources. To achieve little paging or swapping, server's memory should be configured properly. The authors also suggest that data files, table data and indexes should be distributed properly into different table spaces and disks to reduce disk interaction and to perform load balancing. To improve the query efficiency, the authors focused the indexes by explaining when to use and where to use the indexes. The authors have optimized the paging query statements by avoiding scanning the full tables in all sub-queries. The proposed solution has huge effect on performance by utilizing the server and client resources efficiently, particularly to reduce the consumption of client's resources.

Mateen *et al.*, [6] proposed developing an Automatic Database Management Systems (ADBMS) to automate most of the database activities in order to reduce burden over the DBA. SQL server has different components which are used to implement autonomic behavior. Self-optimization is an important key factor for ADBMS which can be achieved by query optimizer, SQL server automatic statistics management component and performance monitor. Self-configuration being another important factor for ADBMS can be achieved by configuration manager, DB tuning advisor and self-tuning. Factor self-healing can be achieved through maintenance plan providing automatic recovery when SQL server is started. Self-protection can be achieved by single sign in, encryption mechanism and other security features. Self-inspection can be achieved by monitoring tools and Database Consistency Check (DBCC). Self-organization can be achieved through Automatic Storage Management (ASM) and reorganization the indexes. In addition the authors reveal the auto nature of different SQL server components and their degree of human intervention that may lead us to those changes that should make the system fully autonomic in future. Main advantages of the proposed solution are: less DBA interaction with the system, efficient use of system resources, recovery and protection. But sometime the auto mechanism does not fulfill the user requirement; in that case, the DBA involvement is much necessary. To make the system fully autonomic, all these features of SQL server should be compared with other database management systems like Oracle and DB2.

Chaudhuri [7] discussed the fundamental requirements such as search spaces, an accurate cost estimation technique, an efficient algorithm and an optimizer to generate the best

execution plan. The query passes through many representations when it is submitted to database server. The first phase is called parse tree, the intermediate phase is called logical operation tree and the final representation is called the operator tree. There are many logical trees possible for a given query submitted to database and to implement these trees, many combinations of physical operators are possible. An optimal plan is generated according to the operator tree having least resource consumption. For selecting best plan, the statistical information and execution cost are gathered and analyzed. Statistical information includes the number of rows, joins, memory requirement and the number of pages used by a table. Statistical information of column also has importance particularly when they have indexes. Sampling data is used to estimate the statistics/histograms accurately and efficiently. In order to select an inexpensive plan, an enumeration algorithm is needed to build an optimizer and its nature should be to adapt changes in the search spaces dues to the addition of new transformation or new physical operator. Such optimizers are called extensible optimizers, *e.g.*, Starburst and Volcano/Cascade.

Hameurlain [8] explored the growth of query optimization techniques from centralized DB system to data grid systems. The optimization is discussed in uni-processor, distributed, parallel processing and large-scale environments. Optimization methods and their special characteristics are described for each environment. In static optimization, sub-optimal plans are generated due to lack of resources. Then to detect and modify these plans at runtime, dynamic method is presented in various environments. The query optimization in uni-processor systems is of two types: logical and physical optimization. In the proposed solution of scheduling problem of uni-processor relational systems, two search strategies enumerative and random are focused. Enumerative strategy follows the dynamic programming, but it cannot handle complex queries optimization due to their large execution plans. To pick an optimal plan definitely becomes a difficult task. To focus this problem, the concept of random strategies is used. For parallel relational systems, a dynamic optimization algorithm is proposed, which have the basic idea based on collected statistics. In case of distributed environment, the static query optimization is used to optimize the communication costs between the nodes by reducing data transferred between them. Centralized approach could not be scaled up due to the network bandwidth and latency issues. Grid systems are large-scale systems having massive data with many users, sources and other computing resources. These are also dynamic in nature. Flexibility and power make grid systems a good platform for distributed query processing. The adaptive query processing approach is also proposed by the author to optimize query in grid environments.

Lin [9] presented the query optimization flow consisting of multiple modules for distributed databases. The user module in distributed system analyzes the user query request. The System Analysis module examines sentence of the query, where its semantics, syntax and spells are checked followed by converting the query into its corresponding tree. This corresponding tree is passed to query tree conversion module which converts it into the global query tree according to data structure described in the query tree. The global query tree received from query tree conversion module is mapped to the corresponding physical operators' trees by the optimizer module. Then the optimizer module selects a physical operator tree with lowest cost. The order processing module sends the whole process to respective server which gives response to user. The local data dictionary is added with sentence table to store mostly used results to avoid the transmission of large data, which greatly improves query efficiency. But when the size of the corresponding table become large, the CPU processing time will be large and the memory consumption will be more which may be consider the limitation of the proposed system.

In contrast to the old methods, Zafarani *et al.*, [10] proposed a new method for the optimization of heterogeneous distributed databases by introducing a new agent to reduce calculation of join orders resulting in better response time. Some old methods for like decisive, genetic and contingent techniques are discussed for join ordering optimization. The authors have extend the system by adding a new agent which act as an adapter and its main purpose is to reduce the calculations in join ordering. The proposed algorithm avoids the joins operations stages which are frequently sent to database during query processing. The algorithm has three parts: joins order separator, substitution politics and query similarity recognizer. The responsibility of the join order separator is to separate queries with and without join operations. Query similarity recognizer compares the queries and identifies queries of same structure and then makes an execution plan for them. If the sent query is already in the database then its weight is increased. The queries with high score i.e., frequently used queries are then saved into the database. The performance is measured in terms of its execution time between the submitting query and receiving the reply.

Chaudhuri [11] addressed the problems of optimizer like its cardinality estimation. The main focus of the paper is on the characteristics of optimizer so that its core components i.e., cardinality, cost and search estimation control the application input and produce better output. In cardinality estimation, it is easy and effective to work with single dimensional histogram but it may face many challenges when dealing with multi-dimensional histograms because of its space and the combination of columns. For ad-hoc queries, cost estimation is also a challenge for optimizers to improve the optimization time. Keeping in mind the mentioned challenges and issues, the authors propose to revisit the work of optimizer to make it capable to leverage information for analysis. The authors have generalized the statistical module architecture to make it open for the larger set of statistical directives, *e.g.*, cardinality injection and cardinality constraints. Execution plan is simpler type of search directive to control the nature of execution plan selected by an optimizer. Plan-space directives help the application developers and administrator to control the behavior of search algorithm. Plan-space directive helps the DBA to deal with parameterized queries to select a good plan.

Sun *et al.*, [12] focus to optimize the query to speedily retrieve data from the running database by means of indices. To gain the best performance, the database should have a good design, *i.e.*, all the basics tables are in 3NF. The 3NF eliminate data redundancy, support fast transaction and ensure data integrity. After having the design in 3NF, the logic structure of the database is optimized by using indexes techniques. Index is a database object used to retrieve the required data rapidly by reducing the amount of data. It also reduces the I/O operation, improves the response time of the query Since the indexes store addresses of all the rows related the column on which indexing is applied, therefore, they should be applied very carefully as it sometime degrades the performance. Also, there are some cases which restrict the use of index, *e.g.*, the use of NULL and NOT NULL may interrupt the normal use of indexing. Index cannot be utilized in case of incorrect use of LIKE statement. The use of IN or NOT IN operations may also lead to the full table scan hence degrading the performance.

Gupta *et al.*, [13] addressed the techniques used by Oracle for query optimization and analyzed the performance of LIKE operator. Oracle uses cost based as well rule based optimizer to handle query optimization. The main objective of cost based optimizer is to provide high throughput and best response time. Rule based optimizer apply equivalence rules to generate an optimal query evaluation plan for an algebraic query. Different queries are analyzed and their costs are computed by measuring CPU time, elapsed time, disk characteristics, *i.e.*, the number of physical reads, number of buffers for consistent read and current mode. Queries are analyzed with indexed and without indexes. The queries on tables without indexes took much time as compared to the indexed one. It was analyzed that LIKE

operator required a full scan of the table, hence degrading the performance. Therefore, some extra improvement is required to use '%%' in LIKE operator to avoid full scan.

Herodotou *et al.*, [14] focused the optimization of SQL queries running over partitioned tables by presenting a technique to optimally select the best execution plan. Partitioned tables provide variety of advantages to the database systems including query pruning, *i.e.*, fast query processing, access of data in parallel fashion, efficient mechanism to load data, to backup data, to maintain statistics in case of DML operations, better cardinality evaluation and to avoid fragmentation. Particularly, query optimization is not an easy task for a large amount of data,. The authors proposed a partitioned aware technique for the PostgreSQL optimizer which generates plans much better than the current optimizer through better cardinality estimation and improved search space. The type of partition like List, Range *etc.* is not mentioned in the paper.

Antoshenkov *et al.*, [15] focused the query processing and its optimization in Oracle relational databases by reworking the query optimizer to get improved compilation time, handling large amount of data, reduced computational complexity and manual plan management. The authors have developed bitmap compression and dictionary compression methods to achieve efficiency in case of large objects and indexes.

O'Neil *et al.*, [16] focused the query performance in decision support systems and OLAP environments by introducing a method to execute the common multi-table joins. The author presented *Star Join* method with bitmap indexes to improve the performance by avoiding full table scan and hence better evaluation plans.

Bruno *et al.*, [17] looked into ways to improve the poor plan selected by the optimizer through query hints. Through suggestive hints, optimizer can keep on improving the plan until a best plan is picked up. The authors proposed *Phints* to capture possible hints for optimizer to get better plan. Query hints are a non-trivial complex methods to achieve better plan. A summary of the critical evaluation of the proposed techniques in the contemporary literature is provided in Table 1.

Table 1. Summary of Query Optimization Techniques in Relational Databases: Problems/Solutions/Practices/Techniques

Research Topic	Author(s)	Problem Discussed and Proposed Solution	Strengths	Limitation/Scope
Query Optimization	Ioannidis [1]	Described the structure of the optimizer and explained the main issues handle by each optimizer module.	Understanding of optimization concepts and main modules of query optimizer.	Dynamic, parallel and distributed optimizations are not discussed.
Query Optimization	Li *et al*. [2]	Discussed intra-query redundancy in sub-queries. Suggest un-nesting and some other heuristics strategies like selection, projection and joining.	Optimizing query by removing intra-query redundancy.	The experiments performed are not related to the techniques mentioned in the paper.
Genetic optimization for join ordering problem	Sun *et al*.[3]	Proposed genetic algorithm to face the problem of efficient selection of join ordering for making an optimal plan by an optimizer.	Better execution time for the query by proper join ordering.	In terms of cost, the proposed algorithm is little expensive for more than 20 joins.

Sub-query optimization in Oracle	Bellamokanda *et al.* [4]	Avoiding self joins, multi-joins, multi table access and handling NULL values. Proposed sub-query coalescing, sub-query removal using windows functions and NULL-AWARE ANTI-JOIN transformation techniques.	Improved query execution time.	Simulation results are achieved using parallel CPUs and other high level hardware which may be affected if degree of parallelism becomes low.
Paging Query Optimization of Massive Data	Sun *et al*.[5]	Paging query in large scale databases. Proposed a sharable stored procedure at server side and applying Oracle techniques like memory management, indexes, distribution of data and its files.	The proposed solution has huge effect on performance by utilizing the server and client resources efficiently.	The paper mainly focused on Oracle user guides.
Autonomic Computing in SQL Server	Mateen *et al*.[6]	Development of an Automatic Database Management Systems (ADBMS) Introduced self-optimization, self-healing, self-protection, self-inspection and self-organization key characteristics for ADBMS.	Main advantages of the proposed solution are: less DBA interaction with the system, efficient use of system resources, recovery and protection.	No simulation is performed to show the autonomic nature of the system.
An overview of query optimization	Chaudhuri [7]	Selecting the best execution plan for a query by an optimizer. Discussed the fundamental requirements for search spaces, accurate cost estimation technique and optimizer.	Best execution plan out of many candidates plans.	No optimization technique is discussed pertaining to memory issues linked with optimization.
Query optimization in centralized, distributed systems	Hameurlain [8]	The optimization in uni-processor, distributed and parallel processing environments. Discussed enumerative and random techniques, dynamic optimization algorithm, decentralized approach and adaptive query processing.	Enhance optimization of query in all the environments	The paper lacks algorithmic or prototype support.
Query optimization strategies for distributed databases	Lin [9]	Query optimization in distributed databases. The local data dictionary is added with sentence table to	These strategies enhance the query efficiency and reduce data transmission costs.	As size of the corresponding table becomes large, it takes more CPU time and occupies more

		store mostly-used results to avoid the transmission of large data.		memory.
Optimizing Join queries	Zafarani *et al*.[10]	Join ordering in heterogeneous distributed databases.	Reduced calculation of join orders result in better response.	The simulation results shown in the paper are complex.
Query optimizers	Chaudhuri [11]	Cardinality estimation and effective cost estimation.	Best response time for a query.	No interface for the optimizer is discussed.
Query optimization strategy of the VDSI system database	Sun *et al*. [12]	Query optimization of large scale VDSI systems. Proposed indexes algorithm.	Reduces I/O operations and improves the response time of the query.	No cache technique is discussed.
Empirical evaluation of *LIKE* operator in Oracle	Gupta *et al*. [13]	Performance issues with *LIKE* operator, DB design and indexes	High throughput and best response time is achieved.	The paper only discusses *LIKE* operator, but the experimental diagrams and results are about indexes.
Query Optimization Techniques for Partitioned Tables	Herodotou *et al.[14]*	Optimization of SQL queries which are running over partitioned tables. Proposed partitioned aware technique for the optimizer.	Fast query processing, access of data in parallel fashion, efficient mechanism to load data and to maintain statistics.	Ambiguity about the type of partition for which the proposed technique can perform better.
Query processing and optimization in Oracle RDB	Antoshenkov *et al.[15]*	Query processing and its optimization in Oracle relational databases by reworking the query optimizer.	Improved compilation time to reduce complexity and to handle large amount of data.	Partitioning can be introduced while dealing with large amount of data.
Multi-Table Joins Through Bitmapped Join Indices	O'Neil *et al.* [16]	Query performance in decision support systems and OLAP environments by introducing a method to execute the common multi-table joins (Star join).	Better evaluation plans.	Indexes seriously affect performance when DML operations are performed.
Power Hints for Query Optimization	Bruno *et al.* [17]	To improve the poor plan selected by optimizer by query hints.	Better query plan.	Query hints are a non-trivial complex method to achieve better plan.

3. Future Work

Majority of research conducted in this area highlights the techniques and models used to enhance the query performance. I intend to propose a technique which will improve the performance issues pointed the in existing research regarding Query Optimizations. My focus will be to improve the query performance through optimization technique in distributed environment which have massive amount of data located at different locations.

4. Conclusions

In this paper, we have made an attempt to present a detailed review of query optimization techniques. The main idea behind this research is to review various techniques to implement query optimization in an effective manner. Query optimization techniques and approaches primarily focus centralized and distributed databases. The paper also highlighted merits of these techniques by critically analyzing them with respect to their utility and efficacy. We have discussed the existing techniques and their implementation for the sake of optimizing query. We have identified some of the proposed techniques that lead towards achieving the key benefits of an optimized query as compare to an un-optimized query in terms of its throughput and response time.

References

[1] Y. Ioannidis, "Query Optimization", Journal ACM Computing Surveys (CSUR), **(1996)**.
[2] D. Li, L. Han and Y. Ding, "SQL Query Optimization Methods of Relation Database System", Computer Engineering and Applications (ICCEA), **(2010)**.
[3] S. Chande and M. Sinha, "Genetic optimization for the join ordering problem of database Queries", India Conference (INDICON), **(2011)**.
[4] S. Bellamokanda, R. Ahmand and A. Witkowski, "Enhanced Subquery Optimizations in oracle", Proceeding of the VLDB Endowment, **(2009)**.
[5] F. Sun and L. Wing, "Paging Query Optimization of Massive Data in Oracle 10g Database", Computer and Information Science and Service System (CSSS), IEEE International Conference, **(2011)**.
[6] A. Mateen, B. Raza and T. Hussain, "Autonomic Computing in SQL Server", Computer and Information Science, Seventh IEEE/ACIS International Conference, **(2008)**.
[7] S. Chaudhri, "An overview of query optimization in relational systems", Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems, **(1998)**.
[8] A. Hameurlain, "Evolution of Query Optimization Methods: From Centralized Database Systems to Data Grid Systems", Proceedings of the 20th International Conference on Database and Expert Systems Applications, **(2009)**.
[9] X. Lin, "Query Optimization Strategies and Implementation Based on Distributed Database", Computer Science and Information Technology, 2nd IEEE International conference, **(2009)**.
[10] E. Zafarani, M. Reza, H. Asil and A. Asil, "Presenting a New Method for Optimizing Join Queries Processing in Heterogeneous Distributed Databases", In Knowledge Discovery and Data Mining,WKDD '10, **(2010)**.
[11] S. Chaudhuri, "Query optimizers: time to rethink the contract?", Proceedings of the 35th SIGMOD international conference on Management of data, **(2009)**.
[12] P. Sun, Z. Zhao and Z. Ge, "The research on the query optimization strategy of the VDSI system database", Computational Intelligence and Industrial Applications, PACIIA 2009, Asica Pacific Conference, **(2009)**.
[13] M. Gupta and P. Chandra, "An Empirical Evaluation of LIKE Operator in Oracle", BVICAM'S International Journal of Information Technology (BIJIT), **(2011)**.
[14] H. Herodotou, N. Borisov and S. Babu, "Query Optimization Techniques for Partitioned Tables", ACM SIGMOD International Conference on Management of data, **(2011)**.
[15] G. Antoshenkov and M. Ziauddin, "Query processing and optimization in Oracle Rdb", The VLDB Journal The International Journal on Very Large Data Bases, **(1996)**.
[16] P. O'Neil and G. Graefe, "Multi-Table Joins Through Bitmapped Join Indices", ACM SIGMOD, **(1995)**.
[17] N. Bruno, S. Chaudhuri and R. Ramamurthy, "Power Hints for Query Optimization", IEEE International Conference on Data Engineering, **(2009)**.