Efficient Processing of Queries with Set Predicates using Reduced Function based Approach

[1] A.Regita Thangam, [2] Dr. S.John Peter
[1] Research Scholar, Department of Computer Science, St.Xavier's College, Palayamkottai, Affiliated to Manonmaniam Sundaranar University, Abishekapatti, Tirunelveli, Tamil Nadu, India
[2] Associate Professor & Head, Department of Computer Science, St.Xavier's College, Palayamkottai, Tamil Nadu, India

Abstract: Query Processing is process of obtaining the desired information from a database system in a predictable and reliable manner. Efficient processing of query is an important requirement in many interactive environments that involve massive amounts of data. Efficient query processing in domains such as Data Warehousing, multimedia search and distributed systems has shown a great impact on performance. The complex SQL queries consist of many scalar-level operations to support the comparison between group of tuples with their attributes and values. Such queries are challenging for a database engine to optimize. But they can be easily implemented with very simple set-level semantics. In this paper, we have developed an efficient algorithm using Reduced Function based Approach for processing queries with set predicates. The efficiency of our technique is demonstrated by running experiments using the WorldCup98 data set.

Index Terms- Data warehousing, optimize, Set predicates, SQL, Reduced Function based Approach, query processing, efficiency.

Introduction

A query is a request to get required data from a database. It can be as simple or more complex. Since database structures are complex in most cases, the query needed to get a data also very complex. The results are generated by accessing relevant database and manipulating it in a way that gives the requested information. Each different query typically requires different processing time. Processing times of the same query may have large variance, from a fraction of a second to hours, depending on the way selected. So the purpose of query optimization is to find the efficient way to process a given query in minimum time.

Since everything is computerised in our environment, large amount of data is maintained in the database. Thus the users have to deal with an enormous amount of data. Therefore, it is necessary to store this information in such a way that it can be retrieved from the database in the fastest possible manner to satisfy the request from a user. Databases are most useful in representing data in an organized manner. It provides the user with the ability to acquire accurate, reliable and timely data for effective decision making process. Thus, the significance of database systems is increasing day by day. At the same time, data queries are becoming more and more complex. As a result, query optimization has emerged as the most significant factor in reducing this complexity. During the past decade, many research and development works have been carried out in the area of query optimization. Query optimization in database has gained significant importance as it helps to reduce the size, memory usage and time required for any query to be processed. The main objective of any query optimization is to determine the best strategy for executing each query. It identifies an efficient way to execute the query with less time complexity to produce better results. This process can be formally defined as transforming a query into an equivalent form which can be evaluated more efficiently.

In recent days, demand of querying the data in larger databases with the semantics of set-level comparison is very high. Suppose we want to find the clients who watched the match on the set of particular days on the given world cup match database. Dates of each candidate that is set of values are compared against the dates in the query condition. Such sets are dynamically formed. Such process of set level comparisons can be performed using currently available SQL syntax and semantics without proposed system [25]. If the set level comparisons performed using currently available SQL syntax, resulting query may be more and more complex. Such complex query becomes a difficult for the user to formulate, which results in too much costly evaluation [17].

The SQL query to find the candidates with extra-curricular activity "singing" and "dancing", as follows:

SELECT id FROM Profiles GROUP BY id HAVING SET(extraactivity) CONTAIN {' singing', 'dancing'}

Given the above query, after grouping, a dynamic set of values on the attribute extraactivity is formed for each unique id, and the groups whose corresponding SET (extraactivity) contain both "singing" and "dancing" are returned as query answers.

The SQL query to find the articles with the authors Raja and Maria only. For this query, the EQUAL operator can be used as below:

SELECT id, articlename FROM Articles GROUP BY id HAVING SET(author) EQUAL {'Raja', 'Maria'}

For the decision making example, suppose we have a table Ratings (department, avg_rating, month, year). The following exemplary query finds the departments whose monthly average ratings in 2018 have always been poor (assuming the rating is from 1 to 5):

SELECT department FROM Ratings WHERE year = 2018 GROUP BY department HAVING SET(avg_rating) CONTAINED BY {1, 2} In this query, CONTAINED BY is used to capture the set-level condition. The CONTAINED BY operator is a logical operator that allows you to compare a value against a set of values. This operator returns true if the value is within the set of values. Otherwise, it returns false or unknown.

Without the explicit notion of set predicates, the query semantics may be captured by using sub-queries connected by SQL set operations (UNION, INTERSECT, EXCEPT), in coordination with join and GROUP BY. Such queries may be quite complex for users to formulate which results in too much costly evaluation. On the contrary, the set predicate construct according to embodiments of the present invention explicitly enables set-level comparisons. The concise syntax makes query formulation simple and also facilitates the efficient native support of such queries in a query engine. This paper is organized as follows. Section I contains the introduction of query processing, the related work laying the stage for our approach is discussed in section II. The proposed REDUCED FUNCTION BASED algorithm is explained in section III. The experimental results and comparison of the proposed algorithm with the state-of-the-art algorithm is described in section IV. Eventually, section V concludes this paper and highlights some future directions.

Related Work

Query processing is the retrieval of information from a database based on the set of retrieval criteria and the database itself remaining unchanged. It is a translation of high-level queries into low-level expression. In query processing, we will actually understand how these queries are processed and how they are optimized. It refers to the range of activities that are involved in extracting data from the database. It is used to extract the information from a large amount of data without actually change the underlying database where the data are organized. . There is a high demand of querying data with the semantics of set-level comparisons. Users can dynamically form set level comparisons without any limitation caused by database schema for set predicates.

Early aggregation is a technique for speeding up the processing of GROUP BY queries by reducing the amount of intermediate data transferred between main memory and disk. Larso [1] described the early aggregation into six algorithms. Three of the algorithms, namely sorting with replacement selection and the two algorithms based on hash partitioning, achieve much higher data reduction than the other three. All six algorithms benefit from early aggregation with grouping by hash partitioning producing the least amount of intermediate data. If the group size distribution is skewed, the overall reduction can be very significant, even with a modest amount of additional main memory.

The importance of Multi-Query Optimization in the context of relational database query processing is explained by J.Chen et al. [4]. Panos Kalnis et al. [10] proposed an efficient way for concurrent execution of multiple queries to increase the throughput.

N. Mamoulis[11] proposed the join of two relations on their set-valued attributes. He considered the various join types, namely the set containment, set equality and set overlap joins. He described existing signature-based algorithms for the set containment join and presents new techniques that employ inverted files. The signature-based methods are only appropriate for set equality joins. For the other join types, a version of Block Nested Loops algorithm was the most suitable algorithm. The set containment join can also be evaluated by joining the two inverted files. On the other hand, a method that joins two inverted files was found inappropriate for all join types.

An efficient algorithm Filtered bitmap index based approach for processing queries with set predicates was proposed in [28]. This algorithm has the benefits of saving disk access and the computation time was reduced by reducing the number of iterations. In this algorithm the groups and the corresponding sets are formed according to the query needs which results in speeds up the query processing.

Chengkai Li et al. [25] proposed Aggregate function based technique and Bitmap index based technique to process query with set predicates. Aggregate function based technique processes set predicates in the normal way as processing conventional aggregate function. Second technique focuses on bitmaps of appropriate columns. Such index structure is applicable on many different types of attributes. This technique processes queries such as selections, joins, multi-attribute grouping etc.

Jayant Rajurkar and T. Khan [26] developed a bitmap pruning strategy by using Word Aligned Hybrid (WAH) compression for processing queries which eliminates the necessity of scanning and processing the entire data set. This technique is used for optimizing queries with set predicates. The set predicates have several advantages than the set-valued attributes together with set containment joins which can support set-level comparisons.

This paper focuses on relational data model and architecture. Some data analytics systems today are built on top of massive parallel computing architecture. The query languages for such systems deal with complex data models such as set-valued attributes, maps, and nested data.

Filtered Bitmap Index Approach

A. Set Predicates

The SQL syntax is extended to support set predicates. Since a set predicate compares a group of tuples to a set of values, it fits well into GROUP BY and HAVING clauses[25]. Specifically in a HAVING clause there is a Boolean expression over multiple regular aggregate predicates and set predicates connected by logic operators AND, OR and NOT.

The syntax of a set predicate is

SET($v_1, . . . , v_m$)
CONTAIN | CONTAINED BY | EQUAL
{($v_1^1, . . . , v_m^1$), …. , ($v_1^n, . . . , v_m^n$) }where $v_i^j \in$ Dom(v_i).

The set predicates allow sets to be dynamically formed through GROUP BY and supports CONTAIN, CONTAINED BY and EQUAL. Below are several example queries with set predicates over the Cust_Sales data table. The sample data for user bank balance is specified in Table I. Each tuple records information such as the CustId, Product, Quantity, Rate and Amount.

Table I Cust_Sales Information

CustId	Product	Quantity	Rate	Amount
1	Pencil	10	5	50
1	Pen	12	10	120
2	Pen	15	10	150
2	Eraser	20	2	40
3	Pencil	16	5	80
3	Pen	18	10	180
3	sketch	12	10	120

Example: 1

To find the total Amount of customers who have bought the products Pen, Pencil.

SELECT CustId, SUM(Amount) FROM Cust_Sales GROUP BY CustId HAVING SET(Product) CONTAIN {‘ Pen’,’ Pencil’}

It identifies the users who have accounts in both banks KVB, TMB. The results are userid 1 and 3. The keyword CONTAIN represents a superset relationship, i.e., the set variable SET(Product) is a superset of {‘ Pen’,’ Pencil’}

Example: 2

To find the total Amount of customers who have bought only Pen, Pencil or Eraser.

SELECT CustId, SUM(Amount) FROM Cust_Sales GROUP BY CustId HAVING SET(Product) CONTAINED BY {‘ Pen’,’ Pencil’,’ Eraser’}

We use CONTAINED BY for the reverse of CONTAIN, i.e., the subset relationship. It selects all the customers who bought the products are only within Pen, Pencil and Eraser. The results are customerid 1and 2.

Example: 3

To find the total Amount of customers who have bought Pen and Pencil, but nothing else.

SELECT CustId, SUM(Amount) FROM Cust_Sales GROUP BY CustId HAVING SET(Product) EQUAL {‘ Pen’,’ Pencil’}

We use EQUAL to represent the equal relationship in set theory. It selects all the customers who bought the products Pen and Pencil. Its result contains only customerid 1.

In many cases, semantics of group-level comparisons may be stated by using the current available given SQL syntax without suggested extension. But the queries resulting will be increasingly complicated than required. A significance to be noted is that complicated queries can be difficult to create for users. More important, such complicated queries may prove to be tough for DBMS in optimizing, and this leads to expensive evaluation that is unnecessary [21]. The query plans resulting from these may involve certain multiple inner queries involving grouping as well as set processes. The suggested syntax with set predicates empowers direct stating of group-level comparisons within SQL, and this makes formulation of query easy. It also fosters effective support to such queries.

B. Proposed Approach

This paper focuses on relational data model and architecture. Reduced Function-Based approach is proposed for processing queries with set predicates. Performance of previously available algorithms suffers from processing unwanted query conditions. In our proposed algorithm the groups and corresponding sets are dynamically formed according to query needs. It supports the set predicate operators CONTAIN, CONTAINED BY and EQUAL. In our algorithm, we first assign the groupid for each record. Then during query processing some filtered conditions are applied for equal and contained by operators to skip the unnecessary checking which helps us to reduce the iterations.

Our approach is based on Function-Based technique. Complex selection queries can be efficiently answered by this approach. A set predicate-aware query plan could potentially be much more efficient by just scanning each group of records and processing its tuples sequentially. The key to such a direct approach is to perform grouping and perform set-level comparison for each tuple in the group. The idea resembles how regular aggregate functions can be processed together with grouping. Hence, we design a method that handles set predicates as aggregate functions.

In the existing Function-Based algorithm [25], the process of matching techniques is applied for all the records and it is a very time consuming process. But in our algorithm, Reduced Function-Based conditions are applied to reduce the number of records to be compared and it will do the process very efficiently with reduced time complexity.

The sketch of the algorithm is as below. It is used to evaluate the queries with set predicates containing the three kinds of set operators {⊇, ⊆, =}. In the first step, it assigns the groupid GID for each tuple based on the group by column. The groupid for each tuple is stored in the hash table GID and the validity of the group is stored in the hash table G. The aggregate values are calculated and stored in the array A for the qualified groups. The set predicate value for each tuple is checked with the condition values specified in the query. If any one of the condition field doesn't match, then further checking will be skipped for the current record for the operators EQUAL (=) and CONTAINED BY (⊆) and the time complexity will be reduced. In detail, a tuple is skipped if the corresponding group is already disqualified and the operator is or = (Line 9).

Algorithm: Reduced Function-Based approach
Input:
Table R(g,a,v) with t tuples,
Query Q = $\Upsilon_{g,\oplus a}$ V op {CV_1,……CV_K}

Output:
Qualified groups g and their aggregate values ⊕a
/*
V – Set predicate column
CV – Condition Values
k – Number of condition values
G – Hash table for storing group validity
A - Aggregate value array
GID – Hash table of size t, storing groupId of each tuple */

Steps:

```
    /* Step 1. Set the groupid for each tuple based on the group by column*/
1.  Assign groupid GID for each record based on groupby fields /* Step 2. Find the qualified groups */
2.  For each distinct groupid g from GID do
3.  For each record in group g do
    /* set predicate value is checked with the condition values specified in the query */
4.  If  V = CVj for some j then
5.  count = count + 1
6.  A[g]=A[g] ⊕ a
7.  Else If  op ∈ {=, ⊆}then
8.  G[g] = false
9.  goto Nextgroup
10. End if
11. End For
12. If op ∈ {⊇} and count >= k then
13. G[g] = true
14. Else if op ∈ {⊆} and (count >0 and                    count <= k) then
15. G[g] = true
16. Else if op ∈ {=} and count = k then
17. G[g] = true
18. Else
19. G[g] = false
20. End if
21. Nextgroup:
22. End for
    /* Step 3. Output qualified groups and their aggregate values */
23. For every group g in hash table G do
24. If G[g] = true then
25. Output  (g, A[g])
26. End for
```

In Step 1, assign the groupId for all the tuples in R, based on the group by column in the given query and store it in the hash table GID. The algorithm outline covers all three set operators, although the details differ, as explained below. In Step 2, it finds the qualified groups by applying Reduced Function-Based approach and it calculates the aggregate value of attribute ($\oplus$ a) from each tuple. The aggregate value of attribute ′a′ is calculated in line 6.

CONTAIN (⊇): It is the superset condition operator in set predicates. Set predicate column value each record is matched against the condition values specified in the query and the count is incremented if the match occurs. If the count is greater than or equal to the number of condition values k specified in the query then we set this current group as valid (Line 12). We use the hash table G to record the Boolean indicators for qualified groups.

CONTAINED BY (⊆): Each record is matched against the condition values specified in the query and the count is incremented if the match occurs. Since it is the subset condition operator, the count value should be within the count of the condition values k specified in the query (Line 14). If the condition fails then skip the current group and mark it as invalid (Line 8). Thus the unnecessary looping is avoided and the time complexity is reduced.

EQUAL (=): In equal operator, all the conditions should be matched (Line 16). Apply the reduced function based approach to exit from the loop if any one condition fails (Line 7). Thus reduces the time complexity of the process.

Results and Discussion

The proposed algorithm uses Reduced Function-Based technique. The experiments are performed on the Intel I3 processor with 4GB RAM memory. The efficiency of this algorithm is proved by using benchmark dataset worldcup-98 which is collected from the website http://ita.ee.lbl.gov/html/contrib/WorldCup.html. The WorldCup98 data set contains 1,352,804,107 tuples, which correspond to all the access requests made to the 1998 World Cup website between April 30, 1998 and July 26, 1998. Each tuple has information such as the time of the request, the type of the requested file, the file size, the server that handled the request, the client identifier (which maps to an IP address) and so on. The Reduced Function-Based algorithm is implemented using Matlab.

Queries: We designed two types of queries on this dataset as follows:

Query Type: 1

To find the total traffics for clients who had visited in two consecutive days- July 24th, July 25th.

SELECT clientID, SUM(Bytes) GROUP BY clientId HAVING SET(date) CONTAIN {0724,0725}

It identifies the clients who visited in both days July 24th, July 25th. The keyword CONTAIN represents a superset relationship, i.e., the set variable SET(date) is a superset of {0724,0725}

Query Type: 2

To find the total traffics of clients who had accessed file types JPG(2), and GIF(3), but nothing else.

SELECT clientID, SUM(Bytes) GROUP BY clientId HAVING SET(type) EQUAL {2,3}

We use EQUAL to represent the equal relationship in set theory. It selects all the clients whose file types are equal to 2 and 3.

Results: The results on the WorldCup98 data set under different data sizes. The number of tuples is changed as 25, 50, 75 and 100 percent of the original data set. The results with various data sizes are shown in Table 2 and Table 3 with Fig. 1. From Fig.1, it shows that Reduced Function-Based technique approach is more efficient than existing Function-Based algorithm [25].

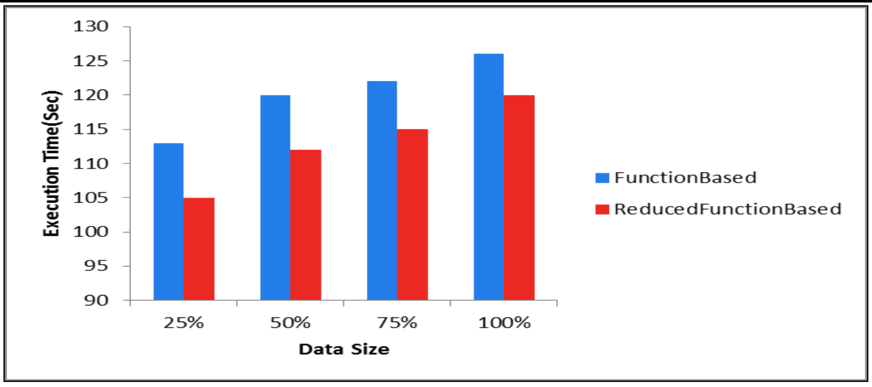
Table 2 Execution time with different data sizes for querytype1

QTYPE-1		
Data Size	**FunctionBased**	**ReducedFunctionBased**
25%	113 secs	105 secs
50%	120 secs	112 secs
75%	122 secs	115 secs
100%	126 secs	120 secs

QTYPE-2

Table 3 Execution Time With Different Data Sizes For Querytype2

QTYPE-2		
Data Size	**FunctionBased**	**ReducedFunctionBased**
25%	505 secs	485 secs
50%	520 secs	513 secs
75%	533 secs	522 secs
100%	545 secs	520 secs



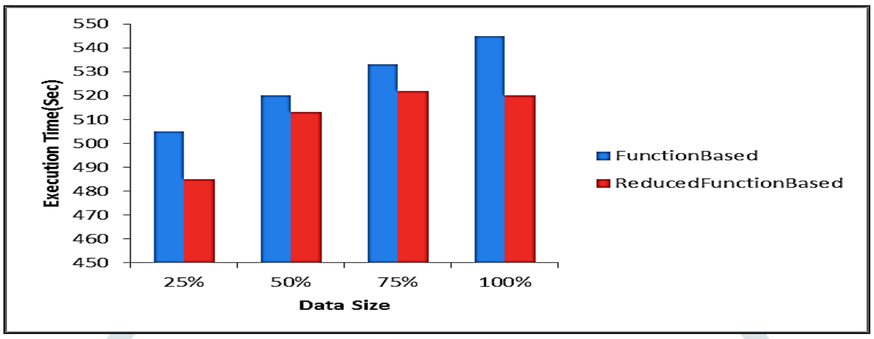


Fig. 1 Execution time on the WorldCup98 dataset under different data sizes for querytype1 and querytype2.

Conclusion

This paper presents an efficient algorithm Reduced Function-Based approach for processing queries with set predicates. This proposed algorithm has the benefits of saving disk access and the computation time was reduced by reducing the number of iterations. In this algorithm the groups and the corresponding sets are formed according to the query needs which results in speeds up the query processing. Today many query processing experts agree that standard query optimization has reached its limits and needs to be re-thought that many sophisticated query optimization techniques must be enabled. Because the query optimizer cost models are extremely inaccurate for complex queries. For handling the growing number of large data warehouses for decision support applications, efficiently executing aggregate queries are becoming increasingly important. In future study another enhanced algorithm to be proposed to tackle the problems of processing the complex queries in Data warehouse environment.

Acknowledgment

We would like to express our deepest thanks to all those who provided the possibility to complete this work. We thank the Manonmaniam Sundaranar University, Abishekapatti, Tirunelveli, Tamil Nadu, India for the Motivation and Encouragement to make this work as successful one. We also like to thank the anonymous reviewers for their valuable comments and suggestions to improve the quality of the paper. We find no words to express our gratitude towards those who were constantly involved with us throughout our work.

References

[1] P.-A. Larso, “Grouping and Duplicate Elimination: Benefits of Early Aggregation,” Technical Report MSR-TR-97-36, Microsoft Research, 1997.

[2] K. Ramasamy, J. Patel, R. Kaushik, and J. Naughton, “Set Containment Joins: The Good the Bad and the Ugly,” Proc. 26th Int’l Conf. Very Large Data Bases (VLDB), 2000.

[3] Surajit Chaudhuri, Kyuseok Shim, “Query optimization in the presence of Foreign functions”, Published in the Proceedings of the 19th International Conference on Very Large Data Bases 03/2000;

[4] J. Chen, D. J. DeWitt, F. Tian, and Y. Wang. NiagaraCQ, “A scalable continuous query system for internet databases”, Published in Proc. SIGMOD, pages 379–390, 2000.

[5] J. Albrecht, W. Hümmer, W. Lehner, L. Schlesinger, “Query Optimization By Using Derivability In a Data Warehouse Environment”, Published in the Proceedings of the 3rd ACM international workshop on Data warehousing and OLAP, DOLAP - 2000, pages 49-56.

[6] Ying Wah Teh, A. B. Zaitun, “Query Processing Techniques in Data Warehousing Using Cost Model”, Published in the Electronic Journal of Information Systems in developing Countries, Volume 3, 2000.

[7] D. Rinfret, P. O’Neil, and E. O’Neil, “Bit-Sliced Index Arithmetic,” Proc. ACM SIGMOD Int’l Conf. Management of Data,

pp. 47-57,2001.

[8] Ralf Rantzaua, Leonard D,. Shapirob, Bernhard Mitschanga and Quan Wangc, “Algorithms and Applications for Universal Quantification in Relational Databases”, Published in Information Systems, Special issue: Best papers from EDBT 2002, Volume 28, Issue 1-2, 01 March 2003.

[9] R. Fagin, A. Lotem, and M. Naor, “Optimal Aggregation Algorithms for Middleware”, Published in Computer and System Sciences, vol. 66, no. 4, pp. 614-656, 2003.

[10] M. A. Hammad, M. J. Franklin, W. G. Aref, and A. K.Elmagarmid, “Scheduling for shared window joins over datastreams”, published in Proc. VLDB, pages 297–308, 2003.

[11] N. Mamoulis, “Efficient Processing of Joins on Set-Valued Attributes,” Proc. ACM SIGMOD Int’l Conf. Management of Data,pp. 157-168, 2003.

[12] S. Melnik and H. Garcia-Molina, “Adaptive Algorithms for Set Containment Joins,” ACM Trans. Database Systems, vol. 28, no. 1,pp. 56-99, 2003.

[13] I.F. Ilyas, W.G. Aref, and A.K. Elmagarmid, “Supporting Top-k Join Queries in Relational Databases”, Published in VLDB J., vol. 13, no. 3, pp. 207-221, 2004.

[14] Bernd Hafenrichter, Werner Kießling, “Optimization of Relational Preference Queries”, published in Proc. ADC '05 Proceedings of the 16th Australasian database conference - Volume 39.

[15] Alaa Aljanaby, Emad Abuelrub, Jordan and Mohammed Odeh, “A Survey of Distributed Query Optimization”, published in The International Arab Journal of Information Technology, Vol. 2, No. 1, January 2005.

[16] Giovanni Maria Sacco, “Truly Adaptive Optimization: The Basic Ideas”, published in Database and Expert Systems Applications(DEXA), volume 4080 of Lecture Notes in Computer Science, page 751-760, Springer, 2006.

[17] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins, “Pig Latin: A Not-so-Foreign Language for Data Processing,” Proc. ACM SIGMOD Int‟l Conf. Management of Data, pp. 1099-1110, 2008.

[18] Mingsheng Hong, Mirek Riedewald, Christoph Koch, Johannes Gehrke, Alan Demers, “Rule-Based Multi-Query Optimization”, Published in Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology, EDBT - March 24–26, 2009.

[19] Pawan Meena, Arun Jhapate & Parmalik Kumar, "Framework for Query Optimization", published in the International Journal of Computer Science and Information Security, Vol. 9, No. 10, October 2011.

[20] Hui Zhao, Shuqiang Yang, Zhikun Chen, Songcang Jin, Hong Yin and Long Li, ”MapReduce model-based optimization of range queries”, Published in 2012, 9th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD 2012).

[21] Bin He,Hui-l Hsiao, Member IEEE, Ziyang Liu ,Yu Huang,and Yi Chen,Member,IEEE, “Efficient Iceberg Query Evaluation Using Comressed Bitmap Index”, IEEE Transactionson K1nowledge and Data Engineering , Vol. 24, No. 9, SEPTEMBER 2012.

[22] Davide Martinenghi and Marco Tagliasacchi, ” Cost-Aware Rank Join with Random and Sorted Access”, Published in the IEEE Transactions On Knowledge And Data Engineering, VOL. 24, NO. 12, DECEMBER 2012.

[23] Christian Politz and Ralf Schenkel,” Ranking under tight budgets”, Published in 2012 23rd International Workshop on Database and Expert Sytems Applications.

[24] Swati Jain and Paras Nath Barwal, “Performance Analysis of Optimization Techniques for SQL Multi Query Expressions over Text Databases in RDBMS”, Published in the International Journal of Information & Computation Technology, Volume 4, no. 8, 2014.

[25] Chengkai Li, Member,IEEE, Bin He, Ning Yan, Muhammad Assad Safiullah ”Set Predicates in SQL: Enabling Set-Level Comparisons for Dynamically Formed Groups” IEEE Transactions on Knowledge and Data Engineering , Vol. 26, No. 2, FEBRYARY 2014.

[26] Jayant Rajurkar1 T. Khan2, "A System for Query Processing and Optimization in SQL for Set Predicates using Compressed Bitmap Index", International Journal for Scientific Research & Development Vol. 3, Issue 02, 2015.

Authors Profile

Mrs. A.Regita Thangam is working as an Assistant Professor in St.Xavier's College, Palayamkottai, She earned his M.C.A. degree from M.S. University, Tirunelveli. She also earned his M.Phil from Alagappa University, Karaikudi. Now She is doing Ph.D. (Reg.No: 12203) in Computer Applications at St.Xavier‟s College, Palayamkottai, Affiliated to Manonmaniam Sundaranar University, Abishekapatti, Tirunelveli, Tamil Nadu, India. She has published research papers in International and National journals.

Dr. S.John Peter earned his M.Sc. and M.Phil. from Bhradhidasan University, Trichirappli. The M.S University, Tirunelveli awarded his Ph.D. degree in Computer Science for his research in Data Mining. He is the Head of the department of computer science, and the Director of the computer science research center, St. Xavier‟s College (Autonomous), Palayamkottai, Tirunelveli. The M.S. University, Tirunelveli has recognized him as a research guide. He has published research papers in International, National journals and conference proceedings. He has organized Conferences and Seminars at the National level.