IJCSE International Journal of Computer Sciences and Engineering Open Access
Research Paper Volume-5, Issue-11 E-ISSN: 2347-2693

Efficient Processing and Optimization of Queries with Set Predicates using Filtered Bitmap Index

A.Regita Thangam[1*], S.John Peter[2]

[1*]Department of Computer Science, St.Xavier's College, Palayamkottai, India
[2] Department of Computer Science, St.Xavier's College, Palayamkottai, Tamil Nadu, India

[*]*Corresponding Author:* regitaraja@gmail.com, *Tel.: +91-98426-08882*

Available online at: www.ijcseonline.org

Received: 17/Oct/2017, Revised: 29/Oct/2017, Accepted: 20/Nov/2017, Published: 30/Nov/2017

***Abstract*—** Query optimization is a common task performed by database administrators and application designers in order to tune the overall performance of the database system. In several applications, the currently available Database Management System is inadequate to support the comparison between the group of tuples with their attributes and values. Currently, databases are used in almost all corporate and business applications that handle a huge amount of data. The complex SQL queries consist of scalar-level operations are often formed to obtain even very simple set-level semantics. Such queries are not only difficult to write but also challenging for a database engine to optimize. To overcome this problem, in this paper we developed an effective algorithm using Filtered Bitmap Index Approach for processing queries with set predicates. It eliminates the necessity of processing the entire Bitmap array index for the required tables and speeds up the query processing significantly. Experimental results show that our approach outperforms the existing algorithm to process queries with set predicates.

Keywords: Bitmap array Index, Set predicates, Set-level semantics, SQL, Filtered Bitmap index, Processing queries, Optimizing queries.

I. INTRODUCTION

Query processing is the process of translating a query expressed in a high-level language such as SQL into low-level data manipulation operations. Query Optimization refers to the process by which the best execution strategy for a given query is found from a set of alternatives.

In recent days, the demand of querying the data in the data warehouse and OLAP applications 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 scalar-level implication in SQL becomes progressively important to support a new group of operation that needs set-level contrast semantics. That is matching a tuples group with multiple values. Complicated queries of SQL constructed using scalar-level operation are frequently formed to get even simplest set-grade semantics. These queries are not only challenging to write but also difficult with regard to database engine optimization. So they may result in the expensive evaluation.

The query syntax also allows comparing the sets defined on multiple attributes. A query with multiple set predicates can be supported for Boolean Operators such as AND, OR and NOT and the aggregate functions that are defined by the database server, such as AVG, SUM, and COUNT.

The SQL query to find the candidates with skills "Java programming" and "Web services", as follows:

SELECT id FROM Resumes GROUP BY id HAVING SET(skill) CONTAIN {'Java', 'Web services'}

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

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

SELECT id, articlename FROM Articles GROUP BY id HAVING SET(author) EQUAL {'Mary', 'James'}
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 2016 have always been poor (assuming the rating is from 1 to 5):

SELECT department FROM Ratings WHERE year = 2016
GROUP BY department HAVING SET(avg_rating)
CONTAINED BY {1, 2}

In this query, CONTAINED BY is used to capture the set-level condition. 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 the too much costly evaluation. On the contrary, the set predicate constructs according to embodiments of the present invention explicitly enable 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 FILTERED BITMAP INDEX 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.

II. RELATED WORK

Query Optimization refers to the process by which the best execution strategy for a given query is found from a set of alternatives. There is a high demand for 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. In many applications, there is a need to integrate data and operations that are external to the database. Access to such external data is provided by a set of interface routines. Surajit Chaudhuri and Kyuseok Shim [3] described a comprehensive approach for optimization in the presence of foreign functions. They provided a declarative rewrite rule system which can be used to express the semantics of foreign functions and also provided an algorithm to enumerate the equivalent queries.

A Rule-based Multi-Query Optimization framework, called RUMOR was presented by Mingsheng Hong et al. [18]. It extends the rule-based query optimization and query-plan-based processing model used by the current RDBMS and stream systems. RUMOR provided a modular and extensible framework, enabling new optimization techniques to be developed and incorporated incrementally into the system. It also integrated the new and existing Multi-Query Optimization techniques for relational stream engines and for event engines. The importance of Multi-Query Optimization in the context of relational database query processing is explained by J.Chen et al. [4].

An extension of traditional query rewrite techniques was proposed by Albrecht et al. [5]. Derivability of multidimensional aggregates is the condition that has to be fulfilled to compute the result of an aggregate query based on the values of one or more aggregate views. They presented the conditions for derivability in a large number of relevant cases which go beyond previous approaches.

Ying Wah Teh et al. [6] introduced the various query processing techniques that are used in Data Warehousing queries. They compared the performance of the different query processing techniques and proposed a recommendation for Database Management Systems to select the most cost-effective query processing techniques based on the cost model. In relational databases, universal quantification is implemented by the division operator (represented by ÷) of the relational algebra. The optimal algorithm for the division operator with all possible inputs was identified by Ralf Rantzaua et al. [8].

Rank join operators combine objects of two or more relations and output the k- combinations with the highest aggregate score. The rank join problem [13] has been dealt in the literature by extending rank aggregation algorithms [9] to the case of join in the setting of relational databases. The Cost-Aware with Random and Sorted access (CARS) pulling strategy was proposed by Davide Martinenghi et al. [22] for retrieving the k- combinations with the highest aggregate score that can be formed by joining the results of heterogeneous search engines. They optimized such a strategy with respect to an additive cost model that considers both sorted access and random access.

The efficient integration of preference querying into standard database technology is an important issue. Bernd Hafenrichter et al. [14] proposed a novel approach for relational preference query optimization based on algebraic transformations. The preference queries can be evaluated by preference relational algebra, extending classical relational algebra by two new preference operators. They have provided a series of novel transformation laws for preference relational algebra that are the key to algebraic optimization.

The distributed query optimization is one of the hardest problems in the database area [15]. For a given SQL query, there is more than one possible algebraic query. Some of these algebraic queries are better than others. The quality of an algebraic query is defined in terms of expected performance.

B.Sujatha et al. [27] proposed a search application that enables keyword-based search in the available relational database. She employed several techniques to be able to retrieve meaningful answers to queries consisting of multiple keywords.

Truly Adaptive Optimization (TAO) is a new approach to query optimization proposed by Giovanni Maria Sacco [16].

TAO is a unifying framework for query optimization problems. This method was applied to query optimization for databases distributed over a broadcast network and provided better performance.

Modern database systems use a query optimizer to identify the most efficient plan to execute declarative SQL queries. The role of query optimizers is critical for the decision-support queries featured in data warehousing and data mining applications. Pawan Meena et al. [19] proposed an abstraction of the architecture of a query optimizer and the technical constraints of advanced issues in query optimization.

A global index based optimization strategy for range query and analysis was proposed by Hui Zhao et al. [20] and they do some tests to evaluate the correctness and efficiency at the end. The strategy was first checking whether user requests can be optimized by using the global index knowledge. Christian Politz et al. [23] explained the problem of ranking under budgets on loading and computational costs and introduced a budget-aware learning to rank approach that limits the cost for evaluating a ranking model. The evaluation of their proposed solution of the optimization task showed better results compared with state-of-the-art budget aware ranking methods.

The optimization issues in distributed databases were addressed by Swati Jain et al. [24]. They explored the major principles of query optimization process with volcano query optimization. In order to examine the role of query optimization process in RDBMS, they proposed both static and dynamic process of optimization as well as all the general aspects of query optimization.

Aggregate function based technique and Bitmap index based technique was proposed by Chengkai Li et al. [25] to process the query with set predicates. Aggregate function based technique processes set predicates in the normal way as processing conventional aggregate function. The second technique is more efficient because it focuses on only those tuples which satisfy query condition and bitmaps of appropriate columns. Such index structure is applicable to 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.

III. 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. 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

$SET(v_1, \ldots, v_m)$

CONTAIN | CONTAINED BY | EQUAL

$\{(v_1^1, \ldots, v_m^1), \ldots, (v_1^n, \ldots, v_m^n)\}$ where $v_i^j \in Dom(v_i)$.

The set predicates allow sets to be dynamically formed through GROUP BY and support CONTAIN, CONTAINED BY and EQUAL. Below are several example queries with set predicates over the UserAccounts data table. The sample data for user bank balance is specified in Table-1. Each tuple records information such as the UserId, Bank, BalanceAmount.

Table-1 User Bank Balance information

UserId	Bank	BalanceAmount
1	KVB	10000
1	TMB	12000
2	TMB	20000
2	ICICI	40000
3	KVB	18000
3	TMB	12000
3	LVB	11000

Example: 1

To find the total BalanceAmount for the users who have accounts in the banks KVB, TMB.

SELECT UserId, SUM(BalanceAmount) FROM UserAccounts GROUP BY UserId HAVING SET(Bank) CONTAIN {‘KVB’,’TMB’}

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(Bank) is a superset of {‘KVB’,’TMB’}

Example: 2

To find the total BalanceAmount of users who have only accounts in KVB, TMB and ICICI.

SELECT UserId, SUM(BalanceAmount) FROM UserAccounts GROUP BY UserId HAVING SET(Bank) CONTAINED BY {‘KVB’,’TMB’,’ICICI’}

We use CONTAINED BY for the reverse of CONTAIN, i.e., the subset relationship. It selects all the users whose accounts are only within KVB and TMB. The results are userid 1and 2.

Example: 3

To find the total BalanceAmount for the users who have accounts in the banks KVB, TMB, but nothing else.

SELECT UserId, SUM(BalanceAmount) FROM UserAccounts GROUP BY UserId HAVING SET(Bank) EQUAL {'KVB','TMB'}

We use EQUAL to represent the equal relationship in set theory. It selects all the users whose banks are equal to KVB and TMB. Its result contains only userid 1.

In many cases, the 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 the formulation of query easy. It also fosters effective support to such queries.

B. Proposed Approach

This paper focuses on relational data model and architecture. The Filtered bitmap index 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 find the search pattern bitmap index of the given query. 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 bitmap index based technique. There exists a bitmap for each unique attribute value. The vector length equals the number of tuples in the indexed relation. In the vector for value x of attribute v, its ith bit is set to 1 if the ith tuple has value x on attribute v. Complex selection queries can be efficiently answered by bitwise operations over bit vectors. Moreover, bitmap indices enable efficient computation of aggregates (e.g., SUM and COUNT).

The idea of using the bitmap index to process set predicates is in line with the aforementioned intuition of processing set level comparison by a one-pass iteration of tuples (i.e., their corresponding bits in bit vectors).

This method brings several advantages by leveraging the distinguishing characteristics of bitmap index:

1. Bitmap index vector is created only for the attributes specified in the query.
2. Bitmap index gives us the ability to skip irrelevant tuples.
3. The filtered Bitmap index conditions are applied to reduce the iterations.
4. Results are computed using our algorithm and these results are compared according to time to access those records from the tables.

Table-2 Bitmap index and the Bit Slice Index

Userid		**Bank**			
B1	***B0***	***KVB***	***TMB***	***ICICI***	***LVB***
0	1	1	0	0	0
0	1	0	1	0	0
1	0	0	1	0	0
1	0	0	0	1	0
1	1	1	0	0	0
1	1	0	1	0	0
1	1	0	0	0	1

The Bitmap index and the Bit Slice Index for the sample data in Table-1 are represented in Table-2 which is used in our algorithm. The bitmap index-based approach only needs bitmap indices on individual attributes. Based on single-attribute indices, it copes with general queries, dynamic groups, joins, selection conditions, multiattribute grouping, and multiple set predicates. It does not require the precomputed index for join/selection results or combination of attributes. The particular type of bitmap index we use is the bit-sliced index (BSI) [7] for numeric fields and Bitmap index (BI) for character type fields. Given a numeric attribute on integers or floating-point numbers, BSI directly captures the binary representations of attribute values. The tuples' values on an attribute are represented in binary format and kept in bit vectors. The advantage of BSI is that it indexes high-cardinality attributes with a small number of bit vectors, thus improves query performance if grouping or aggregation is on such high-cardinality attributes.

In the existing Bitmap index 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, Filtered bitmap index 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 finds the search pattern SP of the condition values CV1, CV2 ,… CVn on attribute V specified in the query. Given each value Vi in condition attribute V, it obtains a bit vector SP[i] , where the ith bit is set (i.e., having value 1) if the value Vi is present in

the condition values. The aggregate values are calculated and stored in the array A for the qualified groups. 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.

All the set bits in the search pattern for the condition specified in the given query should be matched with the corresponding values of the searched field of each record for the operators EQUAL (=) and CONTAIN (⊇). If anyone of the condition fields doesn't match, then further checking will be skipped for the current record and the time complexity will be reduced.

Algorithm: Filtered Bitmap Index-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
/* SP - search pattern array
G – Hash table for storing group validity
A - Aggregate value array
GID – Hash table of size t, storing groupId of each tuple
BI_V - Bitmap Index for the set predicate column */

Steps:

```
    /* Step 1. Find the search pattern in the predicates */
1.  For each value of the set predicate column V, i from
    1 to n do
2.  SP[i] = 0
3.  For each condition value CV_i in CV, i from 1 to k do
4.      SP[FieldPosition(CV_i)] = 1
    /* Step 2. Get the groupid for each tuple */
5.  For each bit slice B_i in BSI(g), i from 0 to s-1 do
6.      For each set bit b_k in bit vector B_i do
7.          GID[k] = GID[k] + 2^i
    /* Step 3. Find the qualified groups by applying
    Filtered BI */
8.  For each distinct groupid g from GID do
9.      For each row in BI_V belongs to
            group g do
10.         For each bit in SP, j from 1 to n do
11.             If SP[j] = 1 and BI_V[j] = 1 then
12.                 count = count + 1
13.                 A[g]=A[g] ⊕ a
14.             Else If (SP[j] <> BI_V[j]) and
    (op ∈ {=} or op ∈ {⊇}) then
15.                 G[g] = false
16.                 goto Label1
17.             End if
18.         End for
19.     End for
20.     If op ∈ {⊇} and count >= length(SP=1)
21.         G[g] = true
22.     Else if op ∈ {⊆} and (count >0 and
            count <= length(SP=1))
23.         G[g] = true
24.     Else if op ∈ {=} and (count =
                length(SP=1))
25.         G[g] = true
26.     Else
27.         G[g] = false
28.     End if
29. Label1:
30. End for
    /* Step 4. Output qualified groups and their
    aggregate values */
31. For every group g in hash table G do
32.         If G[g] = true then
33.         Output (g, A[g])
34. End for
```

In Step 2, it gets the groupIds for tuples in R, by querying BSI(g). The groupIds are calculated by iterating through the slices of BSI(g) and summing up the corresponding values for tuples with bits set in these vectors. i.e., BSI(clientid) in our example 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 3, it finds the qualified groups by applying Filtered Bitmap Index conditions and it calculates the aggregate value of the attribute (⊕ a) from each tuple. The aggregate value of attribute a is calculated in line number 13.

CONTAIN (⊇): It is the superset condition operator in set predicates. Since the condition values are bind in the search pattern, each record is matched against the search pattern bits and the count is incremented if the match occurs. If the count is greater than or equal to the no of set bits in search pattern string then we set this current group as valid (Line 20). We use the hash table G to record the Boolean indicators for qualified groups.
CONTAINED BY (⊆): Each record is matched against the search pattern bits and the count is incremented if the match occurs. Since it is the subset condition operator, apply the filtered bitmap index condition, i.e. the count value should be within the length of the set bits in the search pattern (Line 22). If the condition fails then the unnecessary looping is avoided and the time complexity is reduced.
EQUAL (=): In equal operator, all the conditions should be matched. Apply the filtered bitmap index condition to exit from the loop if anyone condition fails (Line 24). Thus reduces the time complexity of the process.

IV. RESULTS AND DISCUSSION

The experiments are performed on the Intel I3 processor with 4GB RAM memory. The proposed algorithm uses the Filtered bitmap index based technique. 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 dataset 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 Filtered bitmap index based method, denoted as the FilteredBI 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 18th, July 19th.

SELECT clientID, SUM(Bytes) GROUP BY clientId HAVING SET(date) CONTAIN {0718,0719}

It identifies the clients who visited in both days July 18th, July 19th. The keyword CONTAIN represents a superset relationship, i.e., the set variable SET(date) is a superset of {0718,0719}

Query Type: 2

To find the total traffics of clients who had accessed file types HTML(1), JPG(2), and GIF(3), but nothing else.

SELECT clientID, SUM(Bytes) GROUP BY clientId HAVING SET(type) EQUAL {1,2,3}

We use EQUAL to represent the equal relationship in set theory. It selects all the clients whose file types are equal to 1, 2 and 3.

Results: The results on the WorldCup98 dataset under different query complexities for querytype1 and querrytype2 are shown in Table 3 and Table 4. We observed that the significant performance gains of the proposed Filtered Bitmap index method on the billion-tuple dataset. The number of values in the set predicate is changed as 25, 75 and 100 percent of the distinct attribute values in the original dataset. The results are shown in Fig. 1.

Table-3 Execution time with different query complexities for querytype1

QTYPE-1		
QC	***Bitmap***	***FBitmap***
25%	1289 secs	675 secs
75%	1260 secs	701 secs
100%	1272 secs	758 secs

Table-4 Execution time with different query complexities for querytype2

QTYPE-2		
QC	***Bitmap***	***FBitmap***
25%	1274 secs	741 secs
75%	1272 secs	786 secs
100%	1297 secs	1170 secs

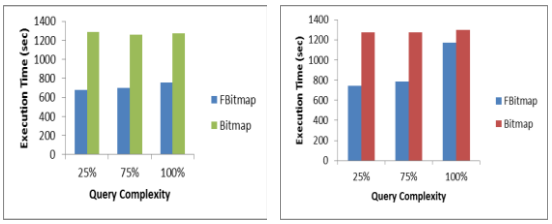

Fig.1 Execution time of bitmap on the WorldCup98 dataset under different query complexities for querytype1 and querytype2.

It is further investigated with different data sizes. The number of tuples is changed as 25, 50, 75 and 100 percent of the original dataset. The results with various data sizes are shown in Table 5 and Table 6 with Fig. 2. From Fig.1 and Fig.2, it shows that Filtered Bitmap Index-Based approach is more efficient than existing Bitmap algorithm [25].

Table-5 Execution time with different data sizes for querytype1

QTYPE-1		
Data Size	***Bitmap***	***FBitmap***
25%	158 secs	96 secs
50%	162 secs	91 secs
75%	158 secs	94 secs
100%	154 secs	93 secs

Table-6 Execution time with different data sizes for querytype2

QTYPE-2		
Data Size	***Bitmap***	***FBitmap***
25%	1269 secs	1155 secs
50%	1271 secs	1166 secs
75%	1273 secs	1162 secs
100%	1274 secs	1161 secs

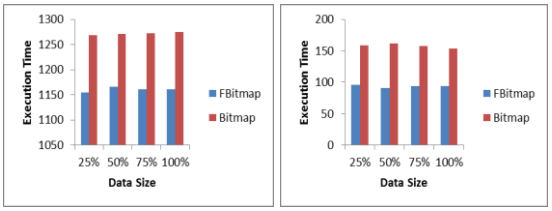

Fig.2 Execution time of bitmap on the WorldCup98 dataset under different data sizes for querytype1 and querytype2.

V. CONCLUSIONS

This paper presents an efficient algorithm Filtered bitmap index 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. In future study another enhanced algorithm to be proposed to tackle the problems of processing the query with multiple set predicates in Data warehouse environment. For handling the growing number of large data warehouses for decision support applications, efficiently executing aggregate queries are becoming increasingly important.

REFERENCES

[1] S. Helmer and G. Moerkotte, "*Evaluation of Main Memory Join Algorithms for Joins with Set Comparison Join Predicates*," Proc. Int'l Conf. Very Large Databases (VLDB), 1996.

[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 Proce..

[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, 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.

[27] A.K. Dwivedi1, A.K. Sharma,"*A Framework For Processing Keyword-Based Queries In Relational Databases For Exact Record*", International Journal of Computer Sciences and Engineering, Vol. 2, issue 8, 2014.

Authors Profile

Mrs. A.Regita Thangam is working as Assistant Professor in Centre for Information Technology & Engineering, M.S. University, Tirunelveli. 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. in Computer Applications at St.Xavier's College, Palayamkottai, Tirunelveli. 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.

