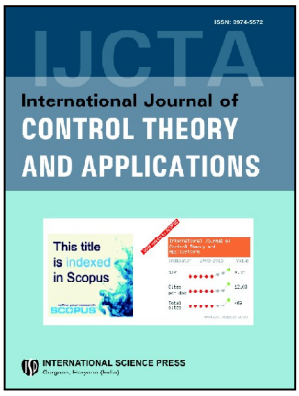

International Journal of Control Theory and Applications

ISSN : 0974-5572

Volume 10 • Number 11 • 2017

Efficient Iceberg Query Evaluation on Multiple Attributes using Set Representation

V. Chandra Shekhar Rao[1] and P. Sammulal[2]

[1] Research Scalar, JNTUH, Associate Professor of Computer Science & Engg., Kakatiya Institute of Technology & Science Warangal-15, Telangana, India, E-mail: vcsrao.kitswgl@gmail.com
[2] Assistant Professor of Computer Science & Engg., JNTUH College of Engineering,, Hyderabad, Telangana, India E-mail: sammulalporika@gmail.com

Abstract: Iceberg query (IBQ) is a special class of aggregation query which compute aggregations upon user provided threshold. In this paper, we extend IBQ evaluation using set approach for multiple attributes. We observed the proposed approach on multiple attributes is efficient as intermediate results contains intersection of attributes, which gets further decreased in their set size in every iteration. An exhaustive experimentation was conducted on synthetic data set and the results are documented.

Keywords: Database, Iceberg query, Set operations, Threshold.

1. INTRODUCTION

The word iceberg query (IBQ) was first introduced by scientist Min Fang et.al [1]. This query is especially designed for extracting the most aggregated data from large database. This is very much suitable for applications such as network monitoring, intrusion detection, finding frequent item sets and all data mining tasks. Most of our current applications are adopting this type of queries for processing them because it allows the user interested threshold for further filtering of aggregated values. The format of an iceberg query on a relation S (C_1, C_2… C_n) is given below:

SELECT C_i, C_j, …, C_m, AGG(*),
FROM S,
GROUP BY C_i, C_j…, C_m,
HAVING AGG (*) > = T.

Figure 1: The format of IBQ

In figure (1), C_i, C_j….C_m represents a subset of attributes in S and is referred as aggregate attributes. AGG represents an aggregation function such as COUNT, SUM, MIN, MAX, AVG and RANK. The greater than or equal to (>=) is a symbol used as a comparison predicate. The HAVING clause filters the query on the given threshold T.

Bin He et. al. in [2] proposed an efficient dynamic pruning algorithm to answer Iceberg queries using compressed bitmap indices, which uses vector alignment algorithm, which guarantees that any bitwise-AND operation will not generate empty result. IBQ evaluation using set representation [3] stores the positions of occurrence of attribute values in a column as a set and performs IBQ evaluation efficiently using set operations.

Many real time applications involve multiple attributes. For example, when we are interested in finding, how much number of cars are sold in the year of 2015 of white color from Maruti, which involves four attributes.

The existing approaches focuses on two attributes, which can be extended further to work with multiple attributes, which do not focus directly on multiple attributes.

In this paper we extended to multiple attributes over two attributes [3]. The results guarantee improved efficiency of the IBQ evaluation as it gets advantage of set intersection operation and store only the existing positions.

Section 2 reviews the related research work on computing iceberg queries. The proposed research work is described in section 3. The experiments conducted on the data sets are explained in section 4. Section 5 analyses the results obtained in experimentation. Section 6 demonstrates the results. The research work is concluded with future scope in section 7.

2. REVIEW OF RELATED RESEARCH

This section reviews an existing work for iceberg query evaluation in two sub sections. The first sub section reviews the techniques before indexing and the second sub section reviews after indexing.

2.1. IBQ evaluation before index

All the traditional methods existing in the literature were evaluated the iceberg queries by scanning the database at least once. The tuples-scan method is the first method that answers an iceberg query by scanning the databases from top to bottom. This method considers two attributes for processing the iceberg query with two columns only. Processing of iceberg query was first studied by Fang et.al [1] by extending the probabilistic techniques [4] and suggested hybrid and multi buckets algorithms. The sampling and multiple hash function techniques were used as basic building blocks of probabilistic techniques such as scaled-sampling and coarse-count algorithms. They estimated the sizes of query results in order to predict the valid iceberg results. This improves query performance and reduces memory requirements greatly. However, these techniques erroneously resulted in false positives and false negatives. To correct these errors, efficient strategies are designed by hybridizing the sampling and coarse-count techniques. To optimize the query execution time of hybrid strategies by extending the linear counting probabilistic algorithm for counting the number of unique values in the presence of duplicates. The linear counting algorithm is based on hashing technique which allocates a bitmap (hash table) of size m in main memory. All entries are initialized to “0”s. The algorithm then scans the relation and applies a hash function to each data value in the column of interest. The hash function generates a bitmap address and the algorithm sets this addressed bit to “1”. The algorithm first counts the number of empty bitmap entries. Then estimates the column cardinality by dividing this count by the bitmap size m and plugging the result.

2.2. Bitmap Indices

The concept of bitmap index was first introduced by professor Israel Spiegler et al [5]. Bitmap indices are known to be efficient in order to accelerate the iceberg queries especially used in the data warehousing applications of

column stores. Model 204 [6] was the first commercial product making extensive use of the bitmap index. This implementation was a hybrid between the basic bitmap index (without compression) and the list of row identifiers (RID list). Overall performance of Model 204 was similar to the index organized as a B+ tree. early bitmap indices are used to implement inverted files [7]. In data warehouse applications, bitmap indices are shown to perform better than tree based index scheme, such as the variants of B-tree or R-tree [6], [8], [9]. Compressed bitmap indices are widely used in column oriented data bases, such as C-store [10] to improve the performance over row oriented data bases. Word-Aligned Hybrid (WAH) [11] and Byte-aligned Bitmap Code (BBC) [12] are two important compression schemes mostly used in query processing with little effort. More importantly, bitmaps are compressed with BBC and WAH can directly participate in bitwise operations without decompression. BBC is effective in both reducing index sizes and query performance. BBC encodes the bitmaps in bytes, while WAH encodes in words. The new word aligned schemes use only 50% more space, but perform logical operations on compressed data 12 times faster than BBC. The development of bitmap compression methods [11], [12] and encoding strategies [13] further broaden the applicability of bitmap index. Nowadays it can be applied on all types attributes such as high cardinality categorical attributes [4], numeric attributes [4], [13], text attributes [14], Compressed bitmap index[15], and it is very efficient for OLAP and warehouse query processing [11], [12].

Bin He et. al. in [2] proposed an efficient dynamic pruning algorithm to answer Iceberg queries using compressed bitmap indices, which uses vector alignment algorithm, which guarantees that any bitwise-AND operation will not generate empty result. Shankar V. et. al in [16], [17] proposed similar strategy by deferring bitwise-XOR operations. A Framework to process iceberg queries using set-intersection and set-difference operation was implemented by Chaitanya Bharati. et. al. [18].

The upcoming section presents a proposal to effectively performing the set operations on multiple columns which computes iceberg query efficiently stated in above sections of this paper.

3. PROPOSED RESEARCH WORK

This section proposes the research work to be carried out on the topic under investigation in the following three sub sections. The block diagram of the proposed work is shown in first sub section 3.1, the algorithm is shown is sub section 3.2 and the description of algorithm and validation discussed in sections 4.

3.1. Proposed block diagram

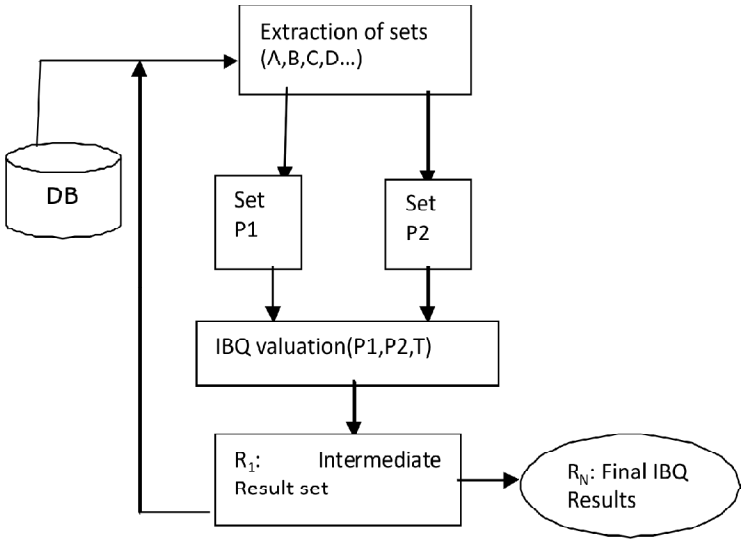

Figure 2: The DB indicates database, R_1 indicates intermediate results and R_N indicates final results

3.2. Algorithm 1: for IBQ Evaluation on multiple columns using Sets

1. Scan the Table
2. Let N as number of columns
3. Set T as Threshold
4. Generate Sets

   Repeat for all the columns in the table

   4.1 Fetch Each column

   4.2 Generate Sets for each attribute storing its position in the column

   4.3 Store the Sets
5. Let P1 = 1 (First Column)
6. Let P2 = 2 (Second Column)
7. Let R = N +1 (Result Column)
8. Repeat following steps while P2 <= N

   8.1 Let T_b time before execution

   8.2 IBQ Evaluation(P1, P2, R, T)

   8.3 Let Te time after execution

   8.4 Let $T_t = T_e - T_b$ (Time taken)

   8.5 Display intermediate results from location R

   8.6 If R = N+1 Begin

   Set P1 = R

   Set R = N+2

   Else

   Set P1 = R

   Set R = N + 1

   End.

   8.7 P2 = P2 +1

   8.8 Display the T_t for each iteration
9. Display the Result From Location R

Algorithm 2 : IBQ Evaluation(P1, P2, R,T)

Consider two attributes P1 and P2 and Place Result in R location

1. Calculate count for each set of attribute P1

   Ex. A1.count = size of set A1.
2. Push sets of attribute P1into Priority Queue based on its first occurrence, if their size is greater than given threshold

Ex. if A1.count>= T then

Sp1.push(A1)

3. Calculate count for each set of attribute P2

   Ex. B1.count = size of set B1.

4. Push sets of attribute P2 into Priority Queue based on its first occurrence if their size is greater than given threshold

   Ex. if B1.count>= T then

   SP2.push(B1)

5. Repeat following steps while both Priority Queues are notempty

   5.1 Fetch aligned sets S1 and S2 from queues SP1, SP2

   5.2 Let S3:=S1

   5.3 Compute S:= S1)" S2

   5.4 Compute S1:=S1 - S2

   5.5 Compute S2:=S2 - S3

   5.6 Compute c:=size of S

   5.7 If c>T

add attribute values with count c and set S into location R.

   5.8 Push sets into corresponding Priority Queues if their set size is greater than given threshold.

6. Return result at location R

4. DESCRIPTION OF ALGORITHMS

The above algorithms provide steps for evaluating IBQ on multiple columns using set operations. Algorithm 1: Accepts database tuples with N columns as input and generates sets for each attribute values by storing its occurrences and invokes Algorithm 2 -IBQ evaluation by passing two attributes at a time. The result will be stored in a location N+1, which will become as next input for the next pass. In the next passes the result will be stored in N+2, N+1, N+2, so on. The algorithm 2 will be repeatedly called until all the specified number of columns are computed. In Algorithm2:The corresponding attribute values sets of P1 and P2 columns will be pushed into corresponding attribute priority queues based on their first occurrence, if their size is greater than given threshold T. The algorithm repeatedly fetches aligned sets from priority queues and computes iceberg result by performing set intersection operation among them until either of the priority queue becomes empty.

In implementation of algorithm we used the following modules, whose implementation is resembles the algorithms defined by Bin He[2].

1. Fetch first attribute position
2. NextAlignedSets.

4.1. Algorithm validation on sample database

This subsection demonstrates the validity of the above proposed algorithm and evaluates the following iceberg query having multiple aggregate attributes with COUNT function on sample database. We show IBQ evaluation

on sample data base is in Table 1, and corresponding attribute value sets are in Table 2 and iceberg result is (A1, B2, C1, D1).

Table 1
Sample Database S.

A	A1	A2	A1	A2	A1	A2	A1	A2	A1
B	B2	B3	B2	B2	B2	B1	B1	B2	B3
C	C1	C2	C1	C2	C1	C2	C2	C2	C2
D	D1	D2	D1	D2	D1	D2	D1	D2	D1

Table 2
Sets for attribute values A1, A2, B1, B2 and B3

S_{A1}	0,2,4,6,8
S_{A2}	1,3,5,7
S_{B1}	5,6
S_{B2}	0,2,3,4,7
S_{B3}	1,8
S_{C1}	0,2,4
S_{C2}	1,3,5,6,7,8,
S_{D1}	0,2,4,6,8
S_{D2}	1,3,5,7

Validation: The sample database S consists of two attributes A, B, C and D with 9 rows. Now the iceberg query in fig 1 is fired against the database table 1. First the set elements are fetched for all the distinct values of aggregate attributes listed in SELECT clause of the query. The sets A1, A2 of attribute A are S_{A1}= {0,2,4,6,8}, S_{A2} = {1,3,5,7}, S_{B1} = {5,6},and S_{B2} = {0,2,3,4,7}, S_{B3} = {1,8}. Here all the sets are sets pushed into corresponding priority queues as their size is greater than given threshold 2.

The aligned sets returned from vector alignment algorithm from priority queues are S_{A1} and $S_{B2.}$

The iceberg result is computed as:

S_T=S_{A1}={0,2,4,6,8}

S_{A1}=S_{A1}-S_{B2}={0,2,4,6,8}-{0,2,3,4,7}={6,8}

S_{B2}=S_{B2}-S_T={0,2,3,4,7}-{0,2,4,6,8}={3,7}

C=size of S_T - size of S_{A1} that is 5-2=3

As the value of C is greater than T the A1,B2 with count 3 is added into result R and $S_{A1}$and S_{B2} are pushed back into priority queues as their new sizes are above 2(threshold)..

A	B	COUNT(*)
A1	B2	3

The above result is sent again with 3rd column produces result (A1, B2, C1) which will applied with 4th column produces final result (A1, B2, C1, D1).

5. EXPERIMENTATION

This section describes three experiments carried out on the implementation described in the previous section under specified iceberg threshold values 100, 300 and 500 on a synthetic database consisting of 1-10 lakhs records with four attributes.

The next section lists the results obtained from the above experimentation and perform the analysis on obtained results through a graph. The result table and corresponding graph is discussed with all necessary comparisons.

6. RESULTS

Experiment 1 with threshold 100 (Data Set in Lakhs):

Table 3
Comparison of execution time by varying data set and attributes for threshold 100

Data Set Attributes	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
2	0.39	0.74	1.54	2.37	2.33	3.20	3.89	6.51	9.30	14.9
3	0.14	0.32	0.69	1.08	1.76	2.19	2.89	3.90	4.95	6.08
4	0.12	0.21	0.33	0.54	0.78	1.14	1.45	2.05	2.69	3.20

The above result table 3 consists of 3 rows and 10 columns. The columns describes the database size by ranging from 1 lakh to 10 lakhs, The first row shows execution times of first two attributes resulting intermediate result which will be sent with third attribute and so on. The results demonstrate the fall in the execution time first two columns with next two columns and so on.

The performance of proposed algorithm demonstrated with graphical representation.

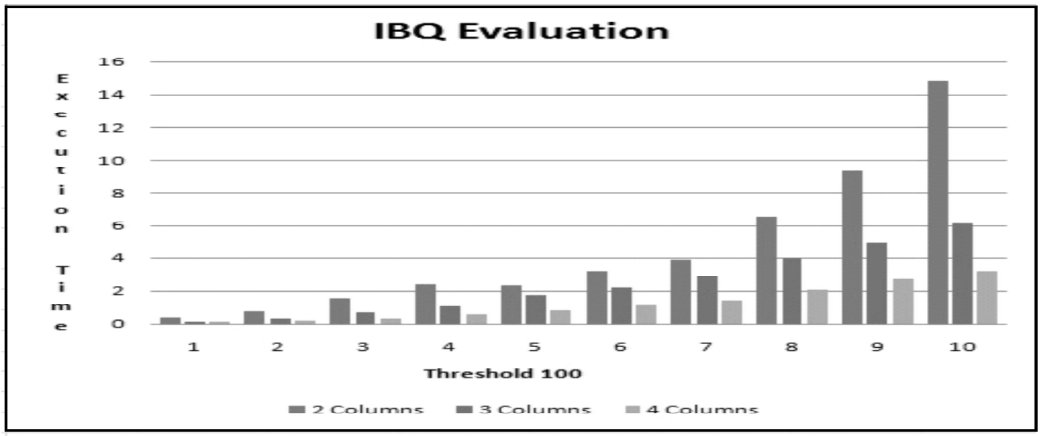

Figure 3: Graph with threshold 100

The above graph fig 3 is showing an execution time of algorithms with threshold 100. The graph is having two axis titles horizontal and vertical on which database size varying from 1 lakh to 10 lakhs and execution time are shown respectively. The blue bar shows execution time on first two columns, the red bar shows the intermediate result with third column and then green bar shows the intermediate results with fourth column. We can also observe the execution time is decreasing. The experiment 2 and 3 are carried with threshold 300 and 400 on database, whose size varying 1 lakh to 10 lakhs demonstrates clearly the fall in the execution time.

Experiment 2 with threshold 300 100 (Data Set in Lakhs):

Table 4
Comparison of execution time by varying data set and attributes for threshold 300

Data Set Attributes	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
2	0.29	0.51	1.72	1.75	2.91	3.12	4.24	5.87	6.45	6.65
3	0.15	0.27	0.44	0.72	1.16	1.49	1.91	3.17	3.29	3.5
4	0.09	0.32	0.39	0.47	0.67	0.78	1.17	1.34	3.08	3.2

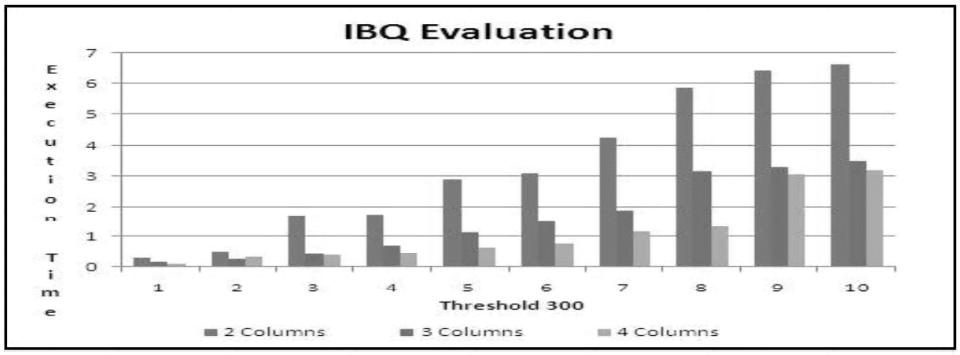

Figure 4: Graph with threshold 300

The above graph figure 4 is showing an execution time of algorithms with threshold 300. The graph is having two axis titles horizontal and vertical on which database size varying from 1 lakh to 10 lakhs and execution time are shown respectively. The blue bar shows execution time on first two columns, the red bar shows the intermediate result with third column and then green bar shows the intermediate results with fourth column. We can also observe the execution time is decreasing.

Experiment 3 with threshold 500 100 (Data Set in Lakhs):

Table 5
Comparison of execution time by varying data set and attributes for threshold 500

Data Set Attributes	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
1-2	0.25	0.43	0.65	1.91	2.15	2.76	3.34	3.75	4.93	5.79
2-3	0.10	0.22	0.34	0.76	0.81	1.10	1.45	2.65	2.75	4.04
3-4	0.09	0.16	0.25	0.36	0.58	0.67	1.21	1.30	1.36	1.42

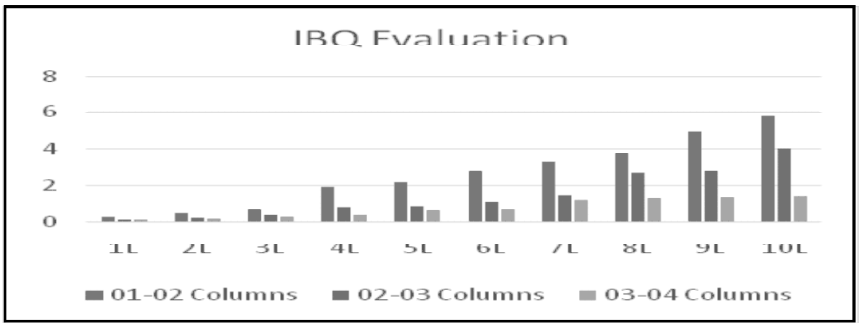

Figure 5: Graph with threshold 500

The above graph figure 5 is showing an execution time of algorithms with threshold 500. The graph is having two axis titles horizontal and vertical on which database size varying from 1 lakh to 10 lakhs and execution time are shown respectively. The blue bar shows execution time on first two columns, the red bar shows the intermediate result with third column and then green bar shows the intermediate results with fourth column. We can also observe the execution time is decreasing.

7. CONCLUSION & FUTURE SCOPE

This paper presents a new IBQ evaluation for processing of multiple columns using set representation method. The sets are used for processing of IBQ by conducting set intersection operation between aligned sets only. The experimental results are demonstrated and observed that IBQ evaluation time from first two attributes to the next two and so on. The future research direction of this work may be reduction of number of set operations and applying dynamic approach in choosing the attributes which may further optimizes the execution time of evaluation of iceberg queries.

REFERENCES

[1] M. Fang, N. Shivakumar, H.Garcia- Molina, R.Motwani and J.D.Ullman."Computing Iceberg Queries Efûciently". In VLDB, pages 299–310, 1998.

[2] Bin He, Hui-I Hsiao, Ziyang Liu, Yu Huang and Yi Chen, "Efficient Iceberg Query Evaluation Using Compressed Bitmap Index", *IEEE Transactions on Knowledge and Data Engineering*, vol. 24, issue 9, Sept 2011, pp.1570-1589.

[3] Rao V.C.S, Sammulal P, "Efficient Iceberg Query Evaluation Using Set Representation", IEEE Explorer, INDICON, Pune, INDIA, ISBN 978-1-4799-5362-2, December 2014, pp. 1-5.

[4] K. Y.Whang, B.T.V.Zanden and H.M.Taylor. "A Linear-Time Probabilistic Counting Algorithm for Database Applications". ACM Trans. Database Syst., 15(2):208–229, 1990.

[5] Spiegler I; Maayan R. "Storage and retrieval considerations of binary databases". *Information Processing and Management: an International Journal* 21 (3): pages 233-54, 1985.

[6] P.E.O'Neil. "Model204 Architecture and Performance". In HPTS Pages 40–59, 1987.

[7] D.E. Knuth, "The Art of Computer Programming : A Foundation for computer mathematics" Addison-Wesley Professional, second edition, ISBN NO: 0-201-89684-2, January 10, 1973.

[8] M. Jrgens "Tree Based Indexes vs. Bitmap Indexes: A Performance Study" In DMDW, 1999.

[9] P.E.O'Neil and G.Graefe. "Multi-Table Joins Through Bitmapped Join Indices". SIGMOD Record, 24(3): 8–11, 1995.

[10] M. Stonebraker, D.J.Abadi, A.Batkin, X.Chen, M.Cherniack, M.Ferreira, E.Lau,A.Lin, S.Madden, E.J.O'Neil, P.E.O'Neil, A.Rasin, N.Tran and S.B.Zdonik.C-Store: "A Column-oriented DBMS". In VLDB, pages 553– 564, 2005.

[11] K.Wu, E. J. Otoo and A. Shoshani. "Optimizing Bitmap Indices with Efûcient Compression", ACM Transactions on Database System, 31(1):1–38, 2006.

[12] G.Antoshenkov, "Byte-aligned Bitmap Compression", Proceedings of the Conference on Data Compression, IEEE Computer Society, Washington, DC, USA, Mar28-30,1995, pp. 476.

[13] P.E.O'Neiland D.Quass. "Improved Query Performance with Variant Indexes". In SIGMOD Conference, pages 38–49, 1997.

[14] K. Stockinger, J.Cieslewicz, K.Wu, D.Rotem and A.Shoshani. "Using Bitmap Index for Joint Queries on Structured and Text Data", *Annals of Information Systems*, 2009, pp: 1–23.

[15] Hsiao H, Liu Z, Huang Y, Chen Y, "Efficient Iceberg Query Evaluation using Compressed Bitmap Index", in Knowledge and Data Engineering, IEEE, Issue: 99, 2011, pp:1.

[16] C.V. Guru Rao and V. Shankar. "Efficient Iceberg Query Evaluation Using Compressed Bitmap Index by Deferring bitwise-XOR Operations", ISBN 978-1-4673-4529-3, IEEE, pages 1306-1311. 2012.

[17] V. Shankar and C. V. Guru Rao. "An Algorithm to Evaluate Iceberg Query using Compacted Bitmap Vector", IJCA(0975-8887), Vol. 60-No. 15, December pages 1-7. 2012.

[18] Chaitanya Bharathi, V. Shankar and B. Hanmanthu " A Framework to Process Iceberg Queries using Set-Intersection and Set-Difference Operations" IJCA(0975-8887), Vol 81-No. 7, November 2013, Pages 22-27.