An Efficient I-MINE Algorithm for Materialized Views in a Data Warehouse Environment

[1]T.Nalini, , [2]Dr. A.Kumaravel, [3]Dr.K.Rangarajan
Dept of CSE, Bharath University,
173, Agaram Road, Selaiyur, Chennai – 600 073, India.

***Abstract*—**The ability to afford decision makers with both accurate and timely consolidated information as well as rapid query response times is the fundamental requirement for the success of a Data Warehouse. Selecting views to materialize for the purpose of supporting the decision making efficiently is one of the most significant decisions in designing Data Warehouse. Selecting a set of derived views to materialize which minimizes the sum of total query response time & maintenance of the selected views is defined as view selection problem. Therefore, to select an appropriate set of a view is the major target that diminishes the entire query response time and also maintains the selected views. Selecting a suitable set of views that minimizes the total cost associated with the materialized views is the key objective of data warehousing. However, these views have maintenance cost, so materialization of all views is not possible. In this paper we are taking into consideration of query frequency, query processing cost and space requirement. In order to find the frequent queries, we make use of I-mine mining techniques from which the frequently user accessible queries will be generated. Then, an appropriate set of views can be selected to materialize by minimizing the total query response time and/or the storage space along with maximizing the query frequency. These can be utilized by the users to obtain the quicker results once a set of views is materialized for the data warehouse.

Keywords : materialization view, data warehousing, selection cost, I-mine item set index, FP growth

I.INTRODUCTION

Data warehouse (DW) can be defined as subject-oriented, integrated, nonvolatile, and time-variant collection of data in support of management's decision [4]. It can bring together selected data from multiple database or other information sources into a single repository [6]. To avoid accessing from base table and increase the speed of queries posed to a DW, we can use some intermediate results from the query processing stored in the DW called materialized views. Therefore, materialized view selection involved query processing cost and materialized view maintenance cost. Selecting views to materialize for the purpose of supporting the decision making efficiently is one of the most significant decisions in designing Data Warehouse [5]. Selecting a set of derived views to materialize which minimizes the sum of total query response time & maintenance of the selected views is defined as view selection problem. Therefore, to select an appropriate set of a view is the major target that diminishes the entire query response time and also maintains the selected views. So, many literatures try to make the sum of that cost minimal.[3-15]

In order to find the frequent queries, we make use of I-Mine techniques from which the frequently user accessible queries will be generated. Then, an appropriate set of views can be selected to materialize by minimizing the total query response time

and/or the storage space along with maximizing the query frequency. These can be utilized by the users to obtain the quicker results once a set of views is materialized for the data warehouse. Given a set of queries *Q* and a quantity *S* (available storage space), the view selection problem is to select a set of views *M* to materialize, that under the multiple objectives constraint that the total space occupied by *M* is less than *S*. [1-2]
In this paper we are consideration three things to improve the query response time, space constraints, query frequency. First we describe I-mine index for materialized view to finding query frequency. Second we are finding query response time and space constraint.[1] The threshold is frequency of query is high and query cost and space constraint is low which query is meet the threshold that particular query is create the materialize view. [2]

The paper is organized as follows. In Section 2, we describe a related work of materialized view and propose work of selection view. In section 3, we describe Terminology and methods using in selection of materialized view query. In section 4, w explain experimental setup and results. In section 5, we describe concluded the paper and section 6 will provide the references.

2. RELATED WORKS

The problem of finding views to materialize to answer queries has traditionally been studied under the name of view selection. Its original motivation comes up in the context of data warehousing.

Harinarayan et al. [21] presented a greedy algorithm for the selection of materialized views so that query evaluation costs can be optimized in the special case of "data cubes". However, the costs for view maintenance and storage were not addressed in this piece of work. Yang et al. [8] proposed a heuristic algorithm which utilizes a Multiple View Processing Plan (MVPP) to obtain an optimal materialized view selection, such that the best combination of good performance and low maintenance cost can be achieved. However, this algorithm did not consider the system storage constraints.
Himanshu Gupta and Inderpal Singh Mumick [9] developed a greedy algorithm to incorporate the maintenance cost and storage constraint in the selection of data warehouse materialized views.
Amit Shukla et al. [12] proposed a simple and fast heuristic algorithm, PBS, to select aggregates for precomputation. PBS runs several orders of magnitude faster than BPUS, and is fast enough to make the exploration of the time-space tradeoff feasible during system configuration.
Himanshu Gupta and Inderpal Singh Mumick [6] developed algorithms to select a set of views to materialize in a data warehouse in order to minimize the total query response time under the constraint of a given total view maintenance time. They have designed approximation algorithms for the special case of OR view graphs.
Chuan Zhang and Jian Yang [4] proposed a completely different approach, Genetic Algorithm, to choose materialized views and demonstrate that it is practical and effective compared with heuristic approaches.
Sanjay Agrawal et al. [11] proposed an end-to-end solution to the problem of selecting materialized views and indexes. Their solution was implemented as part of a tuning wizard that ships with Microsoft SQL Server 2000.
Chuan Zhang et al. [4] explored the use of an evolutionary algorithm for materialized view selection based on multiple global processing plans for queries. They have applied a hybrid evolutionary algorithm to solve problems.
Elena Baralis, Tania Cerquitelli, and Silvia Chiusano,developed a the I-Mine index, a general and compact structure which

provides tight integration of item set extraction in a relational DBMS.[1]

The primary intent of this research is to develop a framework for selecting views to materialize so as to achieve finer query response in low time by reducing the total cost associated with the materialized views. The proposed framework exploits materialize the candidate views by taking into consideration of query frequency, query processing cost and space requirement. In order to find the frequent queries, we make use of I-Mine techniques from which the frequently user accessible queries will be generated. [11] Then, an appropriate set of views can be selected to materialize by minimizing the total query response time and/or the storage space along with maximizing the query frequency. The outcome can be directly utilized by the users to obtain the quicker results once a set of views is materialized for the data warehouse.[11-14]

3. TERMINOLOGY AND METHODS

This section explains the proposed cost effective framework for materialized view selection. We materialize the candidate views by taking into consideration of query frequency, query processing cost and space requirement. In order to find the frequent queries, we make use of I-mine techniques which generates the frequently user accessible queries.[11,12]

3.1 I-Mine(Item set-Mine index) indexes in materialized view

The I-Mine index [1] is a general and compact structure which provides tight integration of itemset extraction in a relational DBMS. Since no constraint is enforced during the index creation phase, I-Mine provides a complete representation of the original database. Data access as well as itemset extraction go in parallel in to reduce the I/O cost. The I-Mine index structure can be efficiently exploited by different itemset extraction algorithms. In particular, I-Mine methods currently support the (FP-growth and LCM v.2 algorithms), but they can straightforwardly support the enforcement of various constraint categories. [16]

Experiments, run for both sparse and dense data distributions, show the efficiency of the proposed index and its linear scalability also for large datasets. Itemset mining supported by the I-Mine index shows performance always comparable with, and often (especially for low supports) better than, state of the art algorithms accessing data on flat file.

The I-Mine index (is a novel data structure that provides a compact and complete representation of transactional data supporting efficient item set extraction from a relational DBMS. It is characterized by the following properties:

1. It is a covering index. No constraint (e.g., support constraint) is enforced during the index creation phase. Hence, the extraction can be performed by means of the index alone, without accessing the original database.
2. The I-Mine index is a general structure which can be efficiently exploited by various item set extraction algorithms.
3. The I-Mine physical organization supports efficient data access during item set extraction.
4. I-Mine supports item set extraction in large data Sets

The index performance has been evaluated by means of a wide range of experiments with data sets characterized by different size and data distribution. The execution time of frequent item set extraction based on I-Mine is always comparable with, and often (especially for low supports).

Mapping relational queries into an item set

A materialized view is a table on disk that contains the result set of a query.

Materialized views are most often used in data warehousing / business intelligence applications where querying large fact tables with thousands of millions of rows would result in query response times that resulted in an unusable application.

Keeping the materialized views under control we need to create materialized views as forms of aggregate tables, or as copies of frequently executed queries, this can greatly speed up the response time of any end user application.
Let Q = { q_1 , q_2 ,.. q_n } be a set of finite number of queries accessing T = { T_1 , T_2 ,.. T_m }, set of finite number of tables having attributes a_i belongs to any table in T. An P-Tree associated to relation R is actually a forest of prefix-trees, where each tree represents a group of transactions all sharing one or more items. In order to make 'item sets' based on queries, we consider the user log for query usage and construct the relation R with the transaction of occurrences of subset of Q as a row in R. Hence any algorithm meant for index selection can be dealt with Q for better performance.

An effective way to compactly store transactional records is to use a prefix-tree. Trees and prefix-trees have been frequently used in data mining and data warehousing indices, including cube forest, FP-tree, H-tree, Inverted Matrix , and Patricia-Tries . Our current implementation of the I-Tree is based on the FP-tree data structure, which is very effective in providing a compact and lossless representation of relation R.

3.2 Fp-Growth

The initial phase of FP-growth is the construction of a memory structure called FP-tree. FP-tree is a highly compact representation of the original database, which is assumed to fit into the main memory (a scalable, disk-based version of FP-tree has also been proposed). FP-tree contains only frequent items, each transaction has a corresponding path in the tree, and transactions having a common prefix share the common starting fragment of their paths. The procedure of creating an FP-tree requires two database scans: one to discover frequent items and their counts, and second to build the tree by adding transactions to it one by one.

After an FP-tree is built, the actual FP-growth procedure is recursively applied to it, which discovers all frequent itemsets in a depth-first manner by exploring projections (conditional FP-trees) of the tree with respect to frequent prefixes found so far. It should be noted that after the FP-tree is created, the original database is not scanned anymore, and therefore the whole mining process requires exactly two database scans.[22]

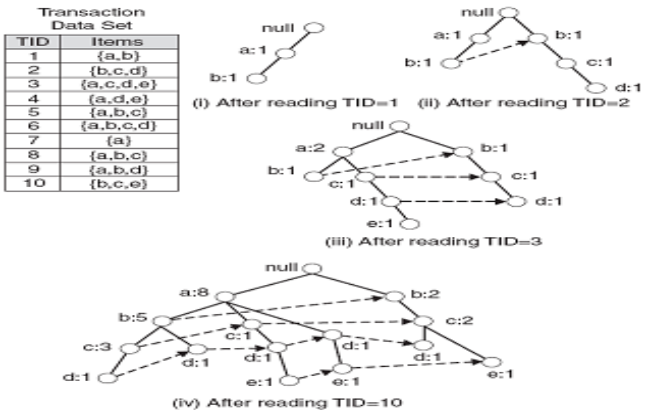


Figure : 1

- Nodes correspond to items and have a counter
- FP-Growth reads 1 transaction at a time and maps it to a path
- Fixed order is used, so paths can overlap when transactions share items (when they have the same prefix).
- In this case, counters are incremented
- Pointers are maintained between nodes containing the same item, creating singly linked lists (dotted lines)

- The more paths that overlap, the higher the compression. FP-tree may fit in memory.
- Frequent itemsets extracted from the FP-Tree.

Algorithm

Input: database *D*, minimum support threshold *minsup*

Output: the complete set of frequent patterns

Method:

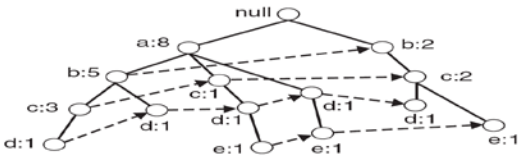
1. scan *D* to discover frequent items and their counts
2. create the root of *FP-tree* labeled as *null*
3. scan *D* and add each transaction to *FP-tree* (omitting non-frequent items)
4. call *FP-growth*(*FP-tree*, *null*)

procedure *FP-growth*(*FP-tree*, α) {
if *FP-tree* contains a single path *P*
then for each combination β of nodes in *P* **do**
generate frequent itemset βUα
with *support*(βUα,*D*)= min support of nodes in β;
else for each a_i in header table of *FP-tree* **do** {
generate frequent itemset β = a_iUα
with *support*(β,*D*) = *support*(a_i,*D*);
construct β's conditional pattern base and
β's conditional *FP-tree*β;
if *FP-tree*≠Ø**then** *FP-growth*(*FP-tree*β);
}

3.3 Frequent Itemset Generation

FP-Growth extracts frequent itemsets from the FP-tree.

- Bottom-up algorithm _ from the leaves towards the root
- Divide and conquer: first look for frequent itemsets ending in e, then de, etc. . . then d, then cd, etc. . .
- First, extract prefix path sub-trees ending in an item(set). using the linked lists.



↑ Complete FP-tree

Example: prefix path sub-trees

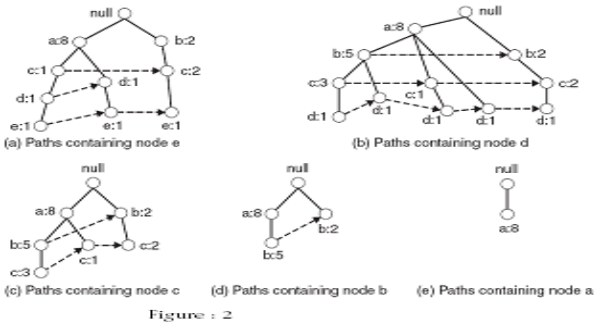


Figure : 2

Each prefix path sub-tree is processed recursively to extract the frequent itemsets. Solutions are then merged.

E.g. the prefix path sub-tree for e will be used to extract frequent itemsets ending in e, then in de, ce, be and ae, then in cde, bde, cde, etc.

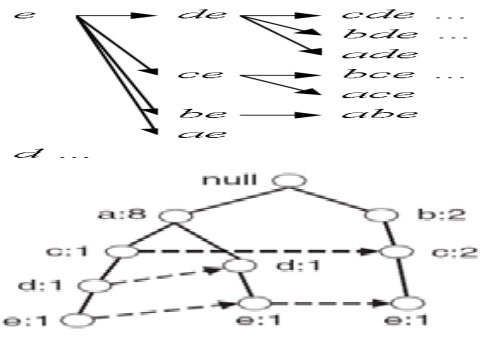


Figure : 3

Prefix path sub-tree ending in e.

3.4 Illustration of frequent itemset Generation

Let minSup = 2 and extract all frequent itemsets containing e.

1. Obtain the prefix path sub-tree for e:
2. Check if e is a frequent item by adding the counts along the linked list (dotted line). If so, extract it.

Yes, count =3 so {e} is extracted as a frequent itemset.

3. As e is frequent, find frequent itemsets ending in e. i.e. de,ce, be and ae. i.e. decompose the problem recursively.

To do this, we must first to obtain the conditional FP-tree for e.

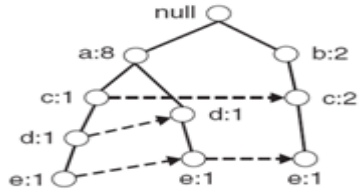


Figure : 4

3.4 Mining rules from the generated frequent itemsets

While association rule mining over FP-Growth without constraints is trivial, when constraints are in play, it is *not* trivial.

To calculate the confidence of a rule {A, B} ⇒ {C} (where {A, B} is called the rule *antecedent* and {C}is called the rule *consequent*), one must use the following formula:

a = rule.antecedent
c = rule.consequent
f = a UNION c = frequent itemset
confidence(a ⇒ c) = support(a UNION c) / support(a) = support(f) / support(a)

So to calculate the confidence of a rule, one needs two values: the support of the entire (frequent) itemset of which the rule consists (support(f)) and the support of the antecedent (support(a)). When the resulting confidence is smaller than minConf, the candidate association rule is dropped, otherwise it is added to the result. [22]. This procedure is implemented based on the following steps.

1) Step through header table from end to start (least common single attribute to most common single attribute). For each item.
a) Count support by following node links and add to linked list of supported sets.
b) Determine the "ancestor trails" connected to the nodes linked to the current item in the header table.
c) Treat the list of ancestor itemSets as a new set of input data and create a new header table based on the accumulated supported counts of the single items in the ancestor itemSets
d) Prune the ancestor itemSets so as to remove unsupported items.
e) Repeat (1) with local header table and list of pruned ancestor itemSets as input.

3.5 Computation of cost selecting queries constructing materialized views

Given space restrictions and, if available, a set of frequently users' queries, these algorithms select an appropriated set of views to materialize in order to achieve a good performance in the query processing of data warehousing environments.

For finding the selection cost S_Q of the every query, the query frequency cost Q_f, query storage cost Q_s and Query processing cost Q_p are computed using the following formulae,

$$Q_f = \frac{f_Q}{\underset{i}{\text{Max}}\ f_Q^{(i)}}$$

$$Q_p = \frac{P_Q}{\underset{i}{\text{Max}}\ P_Q^{(i)}}$$

$$Q_s = \frac{S_Q}{\underset{i}{\text{Max}}\ S_Q^{(i)}}$$

Where the parameters are defined as,
$f_Q \rightarrow$ frequency of query Q
$P_Q \rightarrow$ Processing cost of query Q
$S_Q \rightarrow$ Storage of cost S_Q

Using these parameters such as, Q_f, Q_s and Q_p, the selection cost S_Q is computed using the designed formulae that maximize the query frequency and minimize the spatial cost and query processing cost.

$$S_Q = \alpha * Q_f + \beta * (1 - Q_p) + \delta * (1 - Q_s)$$

Where, α, β *and* δ are Weights such that sum of α, β *and* δ equals 1. Moreover,

Q_f represent Query frequency cost,

Q_s represent query storage cost, and

Q_p represent Query processing cost respectively.

Then, the set of queries whose cost is implemented in less than the minimum threshold (T_M) is selected to build the materialized views.

$$T_M = \sum_{i=1}^{M} \frac{S_Q}{M}$$

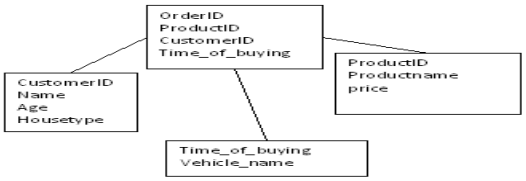
Thus, the selected views to materialize can be achieved the best combination of good query response, low query processing cost and low storage space.

4. EXPERIMENTIAL SETUP AND RESULT

We set up the environment with a purchase-order processing context having four physical table and applying fifty sample queries. The program has been written in Java and the backend is used SQL server8.Then we consolidate the user profile for using those queries by one thousand users as follows:

The input to the proposed approach is data warehouse model, D_W and a user's table (U_T) that contains the list of queries used by the number of users. For the constructing materialized view, the queries that are mostly used by the users should be selected but, at the same time, the query processing cost should be less as w discussed in previous section.

The schema of the data warehouse used in the proposed approach is represented with four various tables such as *customer* (T_1), *order* (T_2), *product* (T_3) and *vehicle* (T_4). Here, '*order*' (T_2) is a target table, which consists of four field records such as *OrderID, ProductID, CustomerID* and *Time of buying* where, *ProductID* and *CustomerID* are two foreign key relations. The *order* table contains one tuple for each new order, and its key is *OrderID*. The *customer* table contains details about the customer and its field records are *customerID, Name, Age, Housetype* and *City*.



Schema design

U_T is used to find the frequency of every queries for computing the query frequency cost. U_T- consisting of 'm' columns signifies the set of queries used by the corresponding users and 'n' rows signifies the number of users who are used the data warehouse to find the important information by posing the queries.

we apply I-Mine algorithm to user's query table U_T for finding the frequent queries and their corresponding support value. Details are given in section 3. After mining the frequent queries we find selection cost for materialized views. Details of Methodology are shown in section 4.

By considering these multi-objective, at first we sort the queries in a descending order based on frequency and at the same time, for other objectives, the queries are sorted in a ascending order according to the storage cost and query processing cost. Then, we select the top 'k' queries from the every sorted list so that the queries that are satisfying multiple objectives can be possibly selected. [13,14]

Sample queries

1. select customer.cid,customer.name from dbo.Customer where cid in (select vehicle.cid from dbo.Vehicle where Vehicle.model='model-19' and Vehicle.color='color-1')
2. SELECT Customer.name, Orders.oid from dbo.Customer INNER JOIN dbo.Orders ON Customer.cid = Orders.cid
3. SELECT Customer.name, Customer.age from dbo.Customer WHERE (((Customer.age)>30))
4.SELECT Customer.age, Vehicle.model from dbo.Customer INNER JOIN dbo.Vehicle ON customer.cid = Vehicle.cid WHERE (((Customer.age)='DISTINCT') AND ((Vehicle.model)='DISTINCT'))
5.SELECT Product.price, Product.quantity from dbo.Product WHERE (((Product.price)>7000) AND ((Product.quantity)>20))

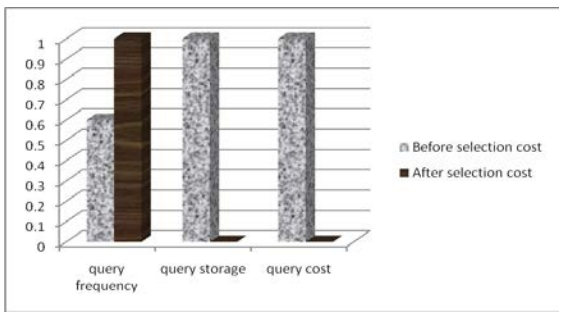
Sample U_T

Query	Qs	Qf	Qc	sq
1	0	0.7	0.2	8.2
2	0	0.8	0.2	8.4
3	0	0.7	0.2	8.2
4	0	0.7	0.2	8.2
5	0	0.8	0.2	8.4
6	0	0.6	0.2	8
7	0	0.6	0.2	8
8	0	0.6	0.2	8
...n (168)	0	0.6	0	8.7

1000 users are access 46 queries.
FP tree generate 2355 nodes.
FP mining generate from 2355 nodes to 168 frequent sets.

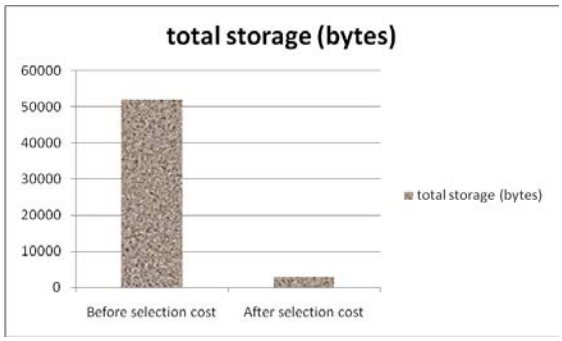
Query	Qs	Qf	Qc	sq
85	0	1	0	9.9
57	0	0.9	0	9.7
49	0	0.9	0	9.7
44	0	1	0.1	9.7
159	0	0.9	0.1	9.5
105	0	0.9	0	9.5
131	0	0.9	0	9.5
86	0	0.9	0	9.4
63	0	0.8	0	9.3
101	0	0.8	0	9.3
167	0	0.8	0	9.3
28	0	0.8	0	9.3
140	0	0.8	0	9.3
125	0	0.8	0	9.3
75	0	0.8	0	9.3
50	0	0.8	0	9.3
150	0	0.8	0	9.2
55	0	0.8	0	9.1
78	0	0.8	0	9.1
130	0	0.8	0	9.1
142	0	0.8	0	9.1
141	0	0.8	0	9.1
88	0	0.8	0	9.1
89	0	0.8	0	9.1
116	0	0.8	0	9.1
41	0	0.8	0	9.1
27	0	0.8	0	9.1
31	0	0.8	0	9.1
29	0	0.8	0	9.1
132	0	0.8	0	9.1
90	0	0.8	0	9.1
25	0	0.8	0	9.1
26	0	0.8	0	9.1
21	0	0.8	0	9.1
16	0	0.9	0.2	9.1
24	0	0.8	0	9.1
96	0	0.8	0	9.1

From the frequent set we apply the multiple objective to find the selection cost of queries. From that we select top 'k' queries.



Before selected the query to materialize storage cost and query cost are high

After satisfy multiple objectives selected queries are materialized.



Total storage is taken less space after select cost of query**.**

7. CONCLUSIONS

The selection of views to materialize is one of the most important issues in designing a data warehouse. The view selection problem has been addressed in this paper by means of taking into account the essential constraints for selecting views to materialize so as to achieve the best combination of low storage cost, low query processing cost and high frequency of query. It can be utilized by the users to obtain the quicker results once a set of views is materialized for the data warehouse. For experimentation, the proposed approach is executed on the simulated data warehouse model and the query list to find the efficiency of the proposed approach in maintaining of materialized view.

In addition to, the choice of algorithm is a major concern in finding the frequent queries for further reducing the time complexity. By considering these, we make use of the I-Mine algorithm, Index Support for Item Set Mining to mine the frequent queries. The advantage of the I-Mine algorithm is that it can mine the frequent queries with less computation time due to its I-Mine index structure compared with the traditional algorithms like, Apriori and FP-Growth.

As further extensions of this work, Incremental update of the index. Currently, when the transactional database is updated, the I-Mine index needs to be rematerialized. A different approach would be to incrementally update the index when new data become available. Since no support threshold is enforced during the index creation phase, the incremental update is feasible without accessing the original transactional database. Also work can be extended with various granular sizes of query i.e. a query can be charactered with the tables, records, attributes levels to meet accurate requirements instead considering only the queries as items in our itemset.

8. REFERENCES

[1] Elena Baralis, Tania Cerquitelli, and Silvia Chiusano," I-Mine: Index Support for Item Set Mining" IEEE Transactions on Knowledge and Data Engineering, vol. 21, no. 4, april 2009

[2] B.Ashadevi, R.Balasubramanian," Cost Effective Approach for Materialized Views Selection in Data Warehousing Environment", IJCSNS International Journal of Computer Science and Network Security, VOL.8 No.10, October 2008

[3] Y. Zhuge, H. Garcia-Molina, J. Hammer, and J. Widom, "View Maintenance in a Warehousing Environment." In Proceedings of the ACM SIGMOD Conference, San Jose, California, May 1995.

[4] C. Zhang, X. Yao, and J. Yang. An evolutionary Approach to Materialized View Selection in a Data Warehouse Environment. IEEE Transactions on Systems, Man and Cybernetics, vol. 31, no.3, pp. 282–293, 2001.

[5] H. Gupta, I.S. Mumick," Selection of views to materialize under a maintenance cost constraint", In Proc. 7th International Conference on Database Theory (ICDT'99), Jerusalem, Israel, pp. 453–470, 1999.

[6] J.Yang, K. Karlapalem, and Q. Li. "A framework for designing materialized views in data warehousing environment". Proceedings of 17th IEEE International conference on Distributed Computing Systems, Maryland, U.S.A., May 1997.

[7] S. Agrawal, S. Chaudhuri, and V. Narasayya, "Automated Selection of Materialized Views and Indexes in SQL Databases," Proceedings of International Conference on Very Large Database Systems, 2000.

[8] P. Kalnis, N. Mamoulis, and D. Papadias, "View Selection Using Randomized Search," Data and Knowledge Eng., vol. 42, no. 1, 2002.
[9] Gupta, H. & Mumick, I., Selection of Views to Materialize in a Data Warehouse. IEEE Transactions on Knowledge and Data Engineering, 17(1), 24-43, 2005.
[10] M. Lee and J. Hammer, Speeding up materialized view selection in data warehouses using a randomized algorithm,
International Journal of Cooperative Information Systems, 10(3): 327–353, 2001.
[11] Gang Gou; Yu, J.X.; Hongjun Lu., "A* search: an efficient and flexible approach to materialized view selection Systems," IEEE Transactions on Man, and Cybernetics, Part C: Applications and Reviews, Vol. 36, no. 3, May 2006 pp: 411 - 425.
[12] A. Shukla, P. Deshpande, and J. F. Naughton, "Materialized view selection for multidimensional datasets," in Proc. 24th Int. Conf. Very Large Data Bases, 1998, pp. 488–499.
[13] T.Nalini,S.K.Srivatsa,K.Rangarajan," International Journal of Advanced Research in Computer Engineering(IJARCE),"Method of ranking in indexes on materialized view for database workload" Vol.4,No.1,pp 157-162
[14] T.Nalini,S.K.Srivatsa,K.Rangarajan," International journal of computer science, systems engineering and information technology(IJCSSEIT)," Efficient methods for selecting materialized views in a data warehouse"Vol.3,No.2, pp 305-310
[15] R. Agrawal and R. Srikant, "Fast Algorithm for Mining Association Rules," Proc. 20th Int'l Conf. Very Large Data Bases (VLDB '94), Sept. 1994.
[16] R. Agrawal, T. Imilienski, and A. Swami, "Mining Association Rules between Sets of Items in Large Databases," Proc. ACM SIGMOD '93, May 1993.
[17] J. Han, J. Pei, and Y. Yin, "Mining Frequent Patterns without Candidate Generation," Proc. ACM SIGMOD, 2000.
[18]A. Savasere, E. Omiecinski, and S.B. Navathe, "An Efficient Algorithm for Mining Association Rules in Large Databases," Proc. 21st Int'l Conf. Very Large Data Bases (VLDB '95), pp. 432-444, 1995.
[19] H. Toivonen, "Sampling Large Databases for Association Rules," Proc. 22nd Int'l Conf. Very Large Data Bases (VLDB '96), pp. 134-145, 1996.
[20] M. El-Hajj and O.R. Zaiane, "Inverted Matrix: Efficient Discovery of Frequent Items in Large Datasets in the Context of Interactive Mining," Proc. Ninth ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (SIGKDD), 2003.
[21] V. Harinarayan, A. Rajaraman, and J. Ullman. "Implementing data cubes efficiently". Proceedings of ACM SIGMOD 1996 International Conference on Management of Data, Montreal, Canada, pages 205--216, 1996.
[22] Frequent Pattern Growth (FP-Growth) Algorithm An Introduction, Florian Verhein , January 2008.

Mrs. T.Nalini received M.Sc from the Karanataka university, M.Tech from Bharath University in 2000, 2007 respectively. Now, she is pursuing Ph.D. in Bharath University. She was a Lecturer between 2000 and 2006. Currently she is an Assistant Professor in the Department of CSE. She has published more than 4 research papers in international journals. She also presented the paper in 15 national conferences and 2 international conferences. She is a life member of many professional bodies like ISTE, CSI, IEEE.

Dr. A.Kumaravel received PG in Computer Science in Applied Sciences from the MIT Chennai , Ph.D in theoretical computer science from Anna university in 1988,1992 respectively. He was worked as a professor in CS for more than 20 years in Singapore. He has published more than 10 papers, and he also presented 20 papers in national and international conferences. He has organized workshops, national and international conferences, seminars in various organizations. Now, he is working as Dean of Computing Studies. He had received senior fellowship in UGC. He is a life member of many professional bodies like ISTE, CSI, IEEE.

Dr.K.Rangarajan is a Professor and Dean of Science & Humanities, Bharath University, TN, India. He has about 60 research publications and guiding many research scholars. His research areas include Automata Theory, Formal Languages, Petri Nets and Graph theory.