A NOVEL ALGORITHM WITH IM-LSI INDEX FOR INCREMENTAL MAINTENANCE OF MATERIALIZED VIEW

[1]Dr.T.Nalini, , [2]Dr. A.Kumaravel, [3]Dr.K.Rangarajan
Dept of CSE, Bharath University
Email-id: nalinicha2002@yahoo.co.in
drkumarvelappavoo@gmail.com
krranagajan@yahoo.com

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. To provide fast access, a data warehouse stores materialized views of the sources of its data. As a result, a data warehouse needs to be maintained to keep its contents consistent with the contents of its data sources. Incremental maintenance is generally regarded as a more efficient way to maintain materialized views in a data warehouse The view has to be maintained to reflect the updates done against the base relations stored at the various distributed data sources. The proposed approach contains two modules namely, (1) materialized view selection(MVS) and (2) maintenance of materialized view. (MMV). In recent times, several algorithms have been proposed for keeping the views up-to-date in response to the changes in the source data. Therefore, we present an improved algorithm for MVS and MMV using IM-LSI(Itemset Mining using Latent Semantic Index) algorithm. selection of views to materialize using the IM(Itemset Mining) algorithm method to overcome the problem resulting from conventional view selection algorithms and then we consider the maintenance of materialized views using LSI. For the justification of the proposed algorithm, we reveal the experimental results in which both time and space costs better than conventional algorithms.

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

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 [2]. It can bring together selected data from multiple database or other information sources into a single repository [3]. 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. Materialized views are the derived relations, which are stored as relations in the database. When a base relation is update, all its dependant materialized views have to be updated in order to maintain the consistency and integrity of the database. The process of updating a materialized view in response to the changes in the base relation is called 'View Maintenance' that incurs a 'View Maintenance Cost'. Because of maintenance cost, it is impossible to make all views materialized under the limited space and time. This need to select an appropriate set of views to materialize for answering queries, this was denoted Materialized View Selection (MVS) and maintenance the selected view denoted Maintenance of Materialized View(MMV). [1-3]

The paper is organized as follows. In Section 2, we describe a related work of materialized view selection and materialized view maintenance and also explain in section 3 and 4 propose work of MVS and MMV. In section 5, we shown experimental setup, section 6, display and its discussion and section 7, we describe concluded the paper and section 8 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. [5] 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 [8] 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 [3] 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 [5] 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. [6] 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. [2] 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.[9]

The primary intent of this research is to 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 work 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 Item set Mining (IM) techniques from which the frequently user accessible queries will be generated. [11]. For the item set mining we are using FP TREE algorithm to find the frequency queries. Then, an appropriate set of views can be selected to materialize by minimizing the total query response time and 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]. After selecting a top k queries are materialized. These queries are maintenance when base table updated without re-computation using LSI (latent Semantic Index).

3 APPROACHES TO MATERIALIZED VIEW SELECTION (MVS)

The challenge behind the first phase is to 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 Item set 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.

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 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. According to, we have used the data ware house, D_W that contains four tables. The schema of the data ware house 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*. The relationship among the multiple tables presented in the example is represented as: $T_2 \rightarrow T_1$; $T_2 \rightarrow T_3$ and $T_4 \rightarrow T_1$, where $T_i \rightarrow T_j$ means that the foreign key of table T_i is the primary key of T_j.

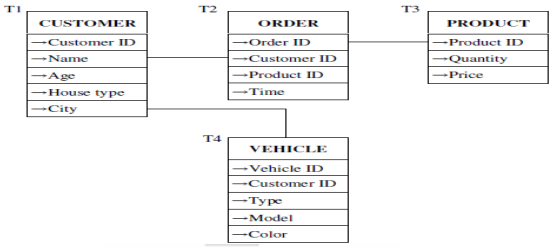

Figure 1: Structure of Database

3.1 Finding the parameters of view selection cost

Then, we have built one user's table, U_T to find the frequency of every query for computing the query frequency cost. The user's table is denoted as, U_T consisting of '*m*' columns and 'n' rows. Every row signifies the number of users who are used the data ware house to find the important information by posing the queries. Every column signifies the set of queries used by the corresponding users. Here, the users table is maintained for the input data ware house model so that the *query frequency* computation can be possible. Once a user's table is built, we can select a set of views for materialization. The frequency computation is not an easy task if the user's table contains large number of attribute columns as well as user rows. So, there is need a standard algorithm to mine the frequent queries from the user's table, U_T. 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 IMine algorithm, Index Support for Item Set Mining to mine the frequent queries. The advantage of the IMine algorithm is that it can mine the frequent queries with less computation time due to its IMine index structure compared with the traditional algorithms like, Apriori and FP-Growth.
So, we have applied IMine algorithm to user's query table U_T for finding the frequent queries and their corresponding support value. Then, for all the queries, we maintain a table, *T* that contains the frequency obtained from the IMine algorithm, the query processing cost and spatial cost required. Using this table, the selection cost S_Q of every query *Q* is computed by combining the above three values. The main objective is that the spatial cost and query processing cost should be minimized but, the frequency-based cost should be maximized. The reason behind is that, if the query is to be materialized, then the query should be frequently used by the number of users. On the other hand, the storage cost should be minimum in order to reduce the space require to store the results. By considering this 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. After that, the queries that are presented in the three sorted lists are selected to find the selection cost, S_Q [14-20].

3.2 Designed formulae to compute the selection cost

For finding the selection cost 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,

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

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

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

Where, $f_Q \rightarrow$ 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, are Weights such that sum of equals 1. Moreover, δβα ,*and* δβα *and* represent Query frequency cost, Q_f represent query storage cost, and *Qs* and *Qp* represent Query processing and cost respectively. Then, the set of queries whose cost is implemented in less than the minimum threshold () is selected to build the materialized views. T_M
Where, α, β, and δ → Weightage constants, Q_f → Query frequency cost, Q_s → query storage cost, and Q_p → Query processing cost. Then, the set of queries that are satisfied 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.

3.3 Experiment for designed formula:

This section presents the running example of the designed formulae utilized in computing the selecting cost. Table 1 gives the users and their queries representing in the matrix format that is given to IMine algorithm to find the frequent queries. Table 2 represents the queries and their relevant frequency, processing cost and storage cost. Then, the queries are sorted in an ascending order for the frequency column and descending order for the processing and storage cost column. Then, top *k*-queries

selected from table 3 are used to find the selection cost. The selection cost of the queries is computed based on the above equation and the values computed are shown in table 3. From the table 3, Q1and Q4 can be selected for materialized view selection based on the threshold value (0.65).

User	Queries		
U1	Q1	Q2	Q3
U2	Q1	Q3	Q2
U3	Q1	Q3	Q5
U4	Q2	Q1	Q4

Table 1: Users and their corresponding queries

Query	f_Q	P_Q	S_Q
Q1	4	2	2
Q2	3	4	5
Q3	2	2	1
Q4	2	4	2
Q5	1	5	4

Table 2: Queries with their costs

query	Qf	Qp	Qs	SQ
q1	1	0.4	0.4	1.1
q4	0.4	0.4	0.4	0.8
q3	0.4	0.8	0.2	0.7
q2	0.75	0.8	1	0.475
q5	0.2	1	0.8	0.2
			Threshold=	0.655

Table 3: Queries sorted based on the computed cost

To satisfy the multiple constraints we are selection only two queries from the total queries.

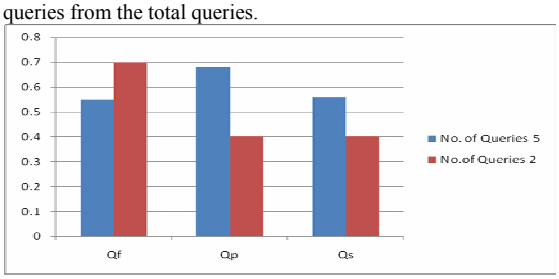

Fig2 : Before and After Materialized view

4 APPROACHES TO MATERIALIZED VIEW MAINTENANCE (MVM)

This section describes the detailed procedure of the designed approach to view maintenance. The principle behind the second module is to handle the maintenance problem without re-computing the materialized views. For example, if the data warehouse gets updated (Addition and deletion of data source) after selecting materialized view, the corresponding updating data source should be reflected in the view. In order to deal with the updating and deletion of data source, the output of the query should be given by considering the updated data records without re-computing the whole process. Accordingly, we have designed an approach to view maintenance without accessing the data warehouse or view. The process of updation and deletion can be happened whenever the data sources are updating the records to the original data warehouse. The diagram given in figure 2 describes the data warehouse updation from the data sources and figure 3 describes the overall procedure of the proposed approach.

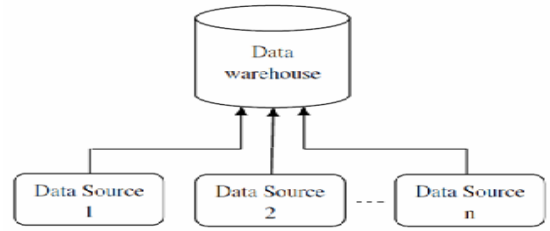

Fig3: Data warehouse updation from the multiple sources,

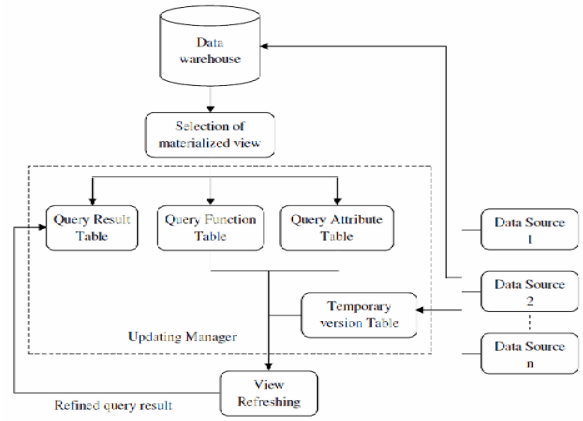

Fig 4: View Maintenance process

4.1. Representation of changes

Once we generate the materialized view for the specific data records, the maintenance of materialized view is important. In order to maintain the information about the materialized view, the following types should be handled. Let, V = $R1 \infty R2 \infty R3$ be the set of relations in the materialized view and R be the relations denoted as, $R = (A, B, C)$. Here, the data warehouse updation especially data record changes can be done in three different ways such as, (1) insertion, (2) deletion and, (3) modification of data record.

(1) Insertion: Let $<D_W>$ be the original data warehouse house and if new record R_i is added into the original data warehouse, the data warehouse will be changed to $< D_W + R_i >$.
(2) Deletion: Let the data record, R_i be defined in the original data warehouse and $< D_W - R_i >$ is denoted like the data record deleted from the original data warehouse $< D_W >$.
(3) Modification of data record: Let R_i be the data record defined in the $< D_W >$ and the specified data record R_i is changed to R_i'. But, there is no addition or deletion in the data ware house and there is a change as $< D_W - R_i' - R_i >$.

4.2 Maintaining tables in updating manager

The ultimate aim of this phase is to build the approach that should reflect the changes done in the updation phase by considering the maintenance cost. Actually, the original data warehouse obtains the data from the multiple data sources that may be in different places. So, the data warehouse can be updated from the multiple data sources that are connected with the different data sources. The view maintenance process is initiated by the updating manager when the data gets added or deleted in 'n' number of times. Once the 'n' updates occurred, the corresponding updates should be reflected in the query output using the depicted procedure. In the updating manager, four tables are maintained about to query attributes, function, query result table and temporary table using LSI index. After constructing the materialized view, the three tables are constructed from the view definition. These three tables are necessary to update the materialized view without accessing the original data warehouse and materialized view.

1) Query attribute table A_T: This table contain $N*M$ matrix, where N is the number of queries materialized and M is the number of attributes within the queries materialized. The values within the matrix may be zero or one, based on whether the attribute is defined in the query or not. The binary values only defined within the query attribute table so that it can be named as binary matrix. This table is used to relate the updated record with the attributes of the query materialized. This table is formed to identify the tables which are relevant to the query.

2) Query function table F_T: This function table maintains the functions of the queries materialized so that the relevant function

of the queries can be performed on the updated record. The query function table is represented with the matrix *N*K*, where '*N*' is the number of queries materialized and '*K*' is the function utilized in the query. This table is necessary to find out the comparison predicate, which restricts the rows to be added to the materialized view.

3) Temporary version table T_T: This table maintains the detailed information of the updated record. Here, the table contains whether the data is inserted, deleted or updated along with the version id. The detailed information of the updated record is located in the temporary version table after the view maintenance process finished. Once the view maintenance process finished for the particular updates, the relevant data will be deleted from the temporary version table that will help to reduce the space complexity.

4) Query result table R_T: This table may be represented as, *N**1 matrix, where, *N* represents the number of queries materialized, Here, the query results of every materialized queries are maintained so that the refreshing the query is easy.

5. EXPERIMENT

The data warehouse schema of the chosen example is given in table 6 in which four tables such as customer, product, order and vehicle are used.

Customer	Product	Order	vehicle
cid	Pid	oid	vid
name	pname	cid	cid
House type	price	Pid	Type
city	quantity	Time	Model

Table 4: Data warehouse schema

Based on the above example, the following three queries, Q1, Q2 and Q4 are considered as the query materialized shown in table 5.

Sample selected materialized view
Q1. SELECT Product.quantity, Product.price from dbo.Product WHERE (((Product.quantity)<50) AND ((Product.price)>500))
Q4. SELECT COUNT(customer.cname) AS Expr1 from dbo.customer

Table 5: Sample selected materialized view

Once the query has selected for materialized, query attribute table, query function table and temporary version will be constructed. For the given example, the attributes needed to execute the queries are cid, cname, price and quantity that are stored as attributes in query attribute table. The query attribute table for the chosen queries is given in table 6, in which "binary one" indicates the attributes presented in the query. Query function table maintains the functions of all materialized queries in table 7.

Table 6: Query attribute table

Index	C1	C2	C3	C4
Q1	0	0	1	1
Q4	0	1	1	1

Table 7: Query function table

Index	Count	Where
Q1	0	Quantity > 50 and price >500
Q4	Count(customer name)	0

Table 8: Query results table

Index	Query Results
Q1	55,400 57,450 60,250
Q4	1000

Whenever the data are inserted into the original data warehouse, the same data is maintained into the temporary version table. Similarly, we consider five updates were done in the original data warehouse so that those data are also updated into the temporary version table. The sample data considered as updated to warehouse is maintained in the following table 9.

Version no	Operation mode	Table	NEW ARRIVEL							
			C1 cid	C2 Name	C3 House type	C4 city	C5 Pid	C6 Pname	C7 Price	C8 Quanity
V1	Insert	Customer	cid-101	A	H1	Chennai				
V2	Insert	Product					Pid-175	Q	125000	25
V3	Delete	Product					Pid-174	P	50000	25
V4	Delete	Customer	cid-101	A	H1	Chennai				
V5	Modify	product					Pid-175	R	30000	25

Table 9: Temporary version table

5.1 Finding relevant queries to view adaptation

When the temporary version table contains '*n*' versions, we have decided to use the batching technique to refresh the view extent rather than the sequential method. In general, sequential and batch maintenance methods are used to maintain the materialized views. Here, we decided to use the batch strategy for updating the result of the particular query defined within materialized view. In this method, whenever '*n*' versions are updated in the temporary version table, the view maintenance process will be started. Here, at first, new arrival column of version table is converted into the binary matrix. If the corresponding attribute contains the data entry, then the binary matrix will have 'binary one' in their relevant field. If the attribute does not contain any entry, the corresponding value would be 'binary zero'.

Version no	Operation Mode	Table	NEW ARRIVEL							
			C1 cid	C2 name	C3 House type	C4 city	C5 Pid	C6 Pname	C7 Price	C8 Quanity
V1	Insert	Customer	1	1	1	1	0	0	0	0
V2	Insert	Product	0	0	0	0	1	1	1	1
V3	Delete	Product	0	0	0	0	1	1	1	1
V4	Delete	Customer	1	1	1	1	0	0	0	0
V5	Modify	product	0	0	0	0	1	1	1	1

Table 10: Temporary Binary version table

After obtaining this matrix, every row matrix is matched with the every row of the query attribute table to find the difference value. The matching should be done with the row matrix of query attribute table that contains the value one. In this set of elements, we find out the number of matches. If the matches will be zero, there is no need to update the materialized view of this query based on the updated record. If the significant matches are found out, then the function defined in the query function table will be performed on the updated data record and the final output of the query will be automatically updated. This procedure is repeated for the every data record in the query temporary table and the results get updated.

Version no	cid	Cname	Price	Qty
V1 (I)	1	1	0	0
V2 (I)	0	0	1	1
V3 (D)	1	1	0	0
V4 (D)	0	0	1	1
V5(M)	0	0	1	1

Table 11: Temporary Binary Matched Matrix

Version no	cid	Cname	Price	Qty
V1 (I)	1	1	1	1
V2 (I)	0	0	1	1
V3 (D)	1	1	0	0
V4 (M)	0	0	1	1

Table 12: Binary matrix from temporary version table

Once the relevant changes are identified, the corresponding view is refreshed based on the function defined within the query function table.

Refreshing the view extent based on insertion changes: Suppose the data record, < *R*+ > is newly added in the data warehouse and assume that this data is related to the query function $F_T(i)$ of query $Q(i)$. Then, the query output will be refined by adding this data so that the refreshing view can be known as, self maintainable.
$R_T^{new}(i) = R_T^{old}(i)$ is replaced by if TBVT(i) belong to $A_T(i)$ and $F_T(i)$

$$R_T^{new}(i) = F_T(R_T^{old}(i), R+) \; if \; < R+ > belongsto \; A_T(i)$$

Function	Formula
Min	If $\max(R_T^{old}(i) < TVT(a_i)$ then $R_T^{old}(i) = R_T^{new}(i)$
Max	If $\max(R_T^{old}(i) > TVT(a_i)$ then $R_T^{old}(i) = R_T^{new}(i)$
Count	$R_T^{new} = R_T^{old} + 1$
Avg	$R_T^{new} = R_T^{old} + TVT(a_i)/n+1$
Sum	$R_T^{new} = R_T^{old} + TVT(a_i)$

Refreshing the view extent based on deletion changes: Suppose the data record, < *R*+ > is newly deleted in the data warehouse and this data are related to the query function $F_T(i)$ of query $Q(i)$. Then, the query output stored in query result table will be refined by performing the query function over this data.

Function	Formula
Min	If $\min(R_T^{old}(i) < TVT(a_i)$ then $R_T^{old}(i)$ else $(R_T^{old}(i), R_T^{new}(i))$ which minimum value is stored , that value is deleted means deletion update will not work
Max	If $\max(R_T^{old}(i) > TVT(a_i)$ then $R_T^{old}(i)$ else $(R_T^{old}(i) = R_T^{new}(i))$ which maximum value is stored , that value is deleted means deletion update will not work
Count	$R_T^{new} = R_T^{old} - 1$
Avg	$R_T^{new} = R_T^{old} - TVT(a_i)/n-1$
Sum	$R_T^{new} = R_T^{old} - TVT(a_i)$

$$R_T^{new}(i) = F_T(R_T^{old}(i), R-) \; if \; < R- > belongsto \; A_T(i)$$

Refreshing the view extent based on modification: Let the data record < *R* > from the data warehouse be modified to another value < *R** >. Then, refreshing the view is carried out by performing the query function to the modified data record and the query output is updated without accessing the data warehouse.

Function	Formula
Min	If $\min(R_T^{old}(i) < TVT(a_i)$ then $R_T^{old}(i)$ else $(R_T^{old}(i), R_T^{new}(i))$
Max	If $\max(R_T^{old}(i) > TVT(a_i)$ then $R_T^{old}(i)$ else $(R_T^{old}(i) = R_T^{new}(i))$
Avg	$R_T^{new} = R_T^{old} - R_T^{old}{}_{(i)} + TVT(a_i)/n$
sum	$R_T^{new} = R_T^{old} - R_T^{old}{}_{(i)} + TVT(a_i)$

$$R_T^{new}(i) = F_T(R_T^{old}(i), R^*) \; if \; < R^* > belongsto \; A_T(i)$$

After the five updates, the binary matrix is generated based on the attributes updated. The binary matrix generated for the above temporary version table is given in table 9. Then, query attribute table is matched with the binary matrix in which the query Q1 is not match with Version no and the query Q2 is matched with V1(I) and V4(I). So, for all queries materialized, the updates are found out such a way it leads to the modification of query result table.

Index	Query Results
Q1	55,400 57,450 60,250
Q4	1000

Table 13: Refresh Result

After updation is completed all temporary tables removed from the memory space. Again temporary version tables are create when new batch of updation forming in the update buffer.

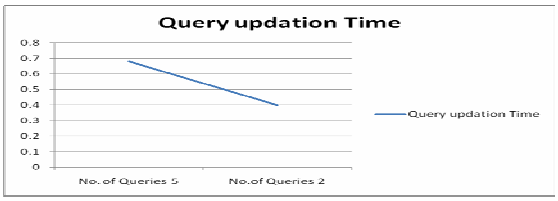

Fig 5: Query updation time

6. RESULTS AND DISCUSSION

This section presents the experimentation of the proposed approach and the detailed analysis over the proposed approach with the previous algorithm.

6.1 Experimental set up and database description

The proposed approach to incremental maintenance of materialized view is implemented using JAVA. The database taken for experimentation is customer transaction database that contains four tables such as *customer* (T_1), *order* (T_2), *product* (T_3) and *vehicle* (T_4). The description of the database is given in figure 1. The experimentation is carried out using the core2duo processor with 1 GB RAM. The total number of data available in customer table is 500 and the order table contains 10000. Similar way, the product table contains 900 and vehicle table consists of 400.

The performance of the proposed approach is compared with the previous algorithms such as IRVSA (Incremental Re-computation Strategy View Selection Algorithm) [25], IVSA (Incremental Strategy View Selection Algorithm) [25], RVSA (Re-computation Strategy View Selection Algorithm) [25] and YKL algorithm [26]. Here, total Cost measured in seconds is used as the performance measure for evaluating and comparing the performance of the proposed approach.

6.2 Performance Experiments

The performance of the proposed approach in maintenance of materialized view is analyzed with the help of three different experiments. 1) The effect of percentage of updates 2) The effect of query load 3) Scalability analysis. In the first set of experiment, the updates are varied in different percentage and total cost needed to maintain the materialized view is computed for different algorithm as well as proposed approach. In the second set of experiment, query load is varied significantly such a way that, the total query cost in sec is computed to find the performance of the algorithms. Similar way, the scalability analysis is also carried out by varying the size of the data warehouse.

6.3 Comparative analysis

This section presents the comparative analysis of the proposed approach with the three algorithms presented in the paper [25] as well as the algorithm described in [26]. For analyzing the effect of updates, the data records are continuously updated to the original data warehouse and the computation time is computed. For doing this experiment, initially, we have taken 500 records in customer table, 10000 records in order table, 900 records product table and 400 records in the vehicle table. Then, the similar number of records is updated to the original data warehouse to find the computation time needed to view maintenance process. Similar way, the process is repeated and the values are plotted as graph shown in figure 4 and figure 5. The figure shows that the performance of the proposed approach is significantly improved in terms of computation time compared with the previous algorithms.

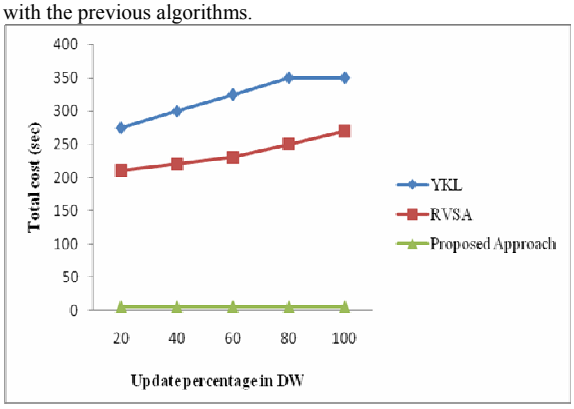

Fig. Effect of percentage of updates

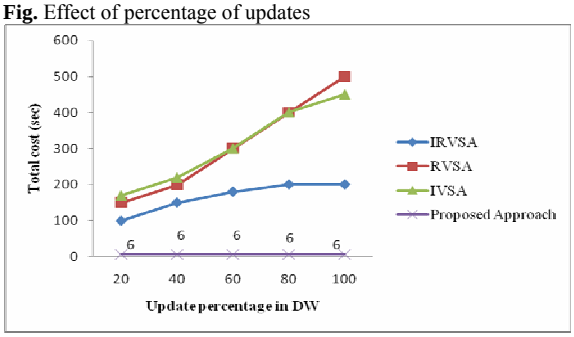

Fig.5. Effect of percentage of updates

In the second set of experiment, we have taken 100 queries for materialized view selection and then the records are updated continuously. The computation cost needed to update these queries is computed and the experiment is repeated for different set of queries. Finally, the values are plotted as graph shown in figure 6 and figure 7. From the figure, when the queries are increased, the total cost is also increased. But, the increasing rate of the proposed approach is less compared with previous algorithms. So, the performance of the proposed approach is improved better in terms of computation time compared with the previous algorithms.

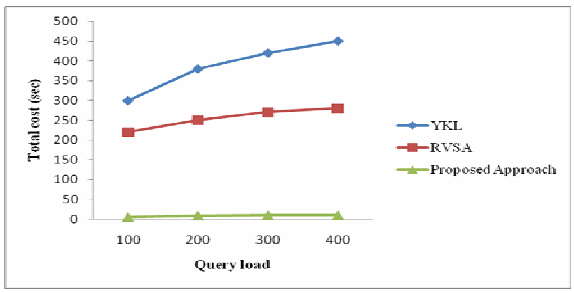

Fig.6. Effect of query load

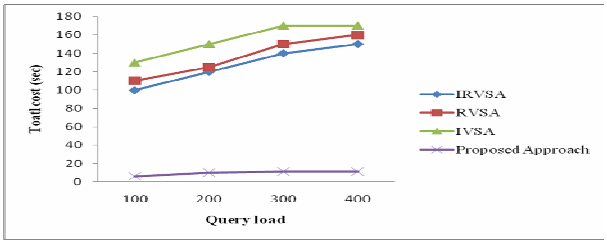

Fig.7. Effect of query load

In the scalability analysis, the experiments are performed for different size of data warehouse. Here, for the change factor 1, the data warehouse is taken as same as that of the first set of experiment and then the computation time needed to update the query is computed. For the change factor 2, the data warehouse size is increased two times than the first one and the experimentation is again continued to obtain the computation time. The results obtained from the experimentation are plotted as a graph shown in figure 8 and figure 9. From the figures, the performance of the proposed approach is significantly improved in terms of computation time compared with the previous algorithms.

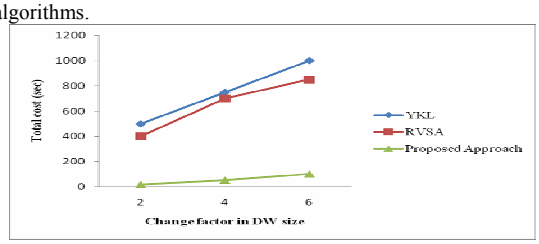

Fig. 8. Scalability analysis

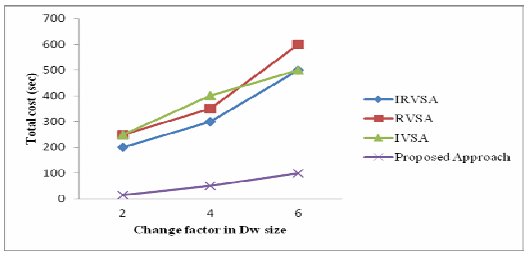

Fig. 9. Scalability analysis

7. CONCLUSIONS

The maintenance of views to materialize is one of the most important issues in designing a data warehouse. The view selection problem and materialized view maintenance problem have 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 and updation of materialized view using LSI. In the first approach, a mathematical model was designed to select materialized view by considering the frequency, processing cost and spatial cost. 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. For the second approach of view maintenance we are using LSI index for maintaining the materialized view without re-computation. 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. As further extensions of this work, improve the index for updating the materialized view. For future research in this area could focus on validating this model against some real-world data warehouse systems and also concentrate on join queries for maintenance.

8. REFERENCES

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] P. Kalnis, N. Mamoulis, and D. Papadias, "View Selection Using Randomized Search," Data and Knowledge Eng., vol. 42, no. 1, 2002.

[8] 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.

[9] 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

[10] 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

[11]T.Nalini,Dr.A.Kumaravel,Dr.K.Rangarajan ," An Efficient I-Mine Algorithm For Materialized Views In A Data Warehouse Environment**",** Ijcsi International Journal Of Computer Science Issues, Vol. 8, Issue 5, No 1, September 2011 Issn (Online): 1694-0814

[12] 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.

[13] 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.

[14] 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.

[15] T.Nalini, S.K.Srivatsa ,K.Rangarajan" Method of anking in indexes on materialized view for database workload" , International Journal of Advanced Research in Computer Engineering(IJARCE), vol.4, No.1,pp 157-162

[16] 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

[17] R. Agrawal and R. Srikant, "Fast Algorithm for Mining Association Rules," Proc. 20th Int'l Conf. Very Large Data Bases (VLDB '94), Sept. 1994.

[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] 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.

[20] Frequent Pattern Growth (FP-Growth) Algorithm An Introduction,Florian Verhein ,January 2008

[21] Jian Yang, Kamalakar Karlapalem, Qing Li, "Algorithms for Materialized View Design in Data Warehousing Environment", in Proceedingof the 23rd International Conference on Very Large Data Bases, San Francisco, CA, 1997.

[22] Yousri, N.A.R., Ahmed, K.M., El-Makky, N.M., "Algorithms for selecting materialized views in a data warehouse", in proceedings of 3rd ACS/IEEE International Conference on Computer Systems and Applications, 2005.

Received: November 2011. Accepted: March 2012.