"AN APPROACH FOR EFFICIENT QUERY PROCESSING FOR MATERIALIZED VIEW SELECTION AND MAINTENANCE"

[1]SARVESH ANASANE
[1]NUVA College of Engineering & Technology, Nagpur
sarvesh.anasane@gmail.com

[2]PROF. SHYAM P. DUBEY
[2]NUVA College of Engineering & Technology, Nagpur
shyam.dubey06@gmail.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. To improve the affection of OLAP queries is an important aspect of data warehouse domain. It affects the efficiency of queries in data warehouse directly. Base on the PBUS algorithm, a novel method is proposed to select materialized views of multidimensional data called dynamic selection Strategy. Another technique is the hybrid mediator is an integration system where one part of data is queried on demand as in the virtual approach, while another part is extracted, filtered and stored in a local database. Statistical analysis on existing query set help to predict the attributes likely to be used for future queries. The materialized views are generated accordingly.*

Keywords: materialization view, data warehousing, hybrid integration system, Dynamic Selection Strategy, OLAP, data warehousing

1. 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. Materialized views are created over existing tables to maintain the set of data which are likely to be accessed frequently by the users. It is pre-computed and summarized data set where from the user queries are answered. This enables fast query execution, as the result set is constructed from summarized data set instead of large tables. This result construction process is started at first, by accessing the materialized views. However, if the desired data is not present in the materialized views then the original tables are accessed to complete the result set construction process.

Availability of desired data in the materialized views is termed as hit and the non-availability of desired data is termed as miss. The ratio of hit and (miss + hit) is termed as hit ratio. A better hit-ratio is an indication of well performing materialized view. This materialized view construction process is guided by this quantitative metric. Based on the given constraints or specifications the knowledge of quantitative metric is applied to finally generate the materialized views.
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]

Data warehouse is to provide information about online analytical processing like decision support and data mining to decision makers. The father of the data warehouse, W.H.Inmon [1], first expounded the thought and theory about data warehouse systematically. He defined the data warehouse as a collection of subject-oriented, integrated, non-volatile and time-variant data with the purpose to support managers' decision-making. data warehouse would consume a great deal of time when it stores data with many dimensions, make an inquiry used OLAP, and conduct aggregation algorithm, Sum, Count, Max, Min,Average etc., which reduces the use efficiency of the data warehouse.

A lot of work has been done to solve this problem such as the PBS algorithm [2] and PBUS algorithm [3]. PBUS algorithm and further adjustment of PBUS algorithm during the specific usage in order to get better results. For this reason, method of materialized view was used to improve the speed of OLAP queries. This information is generally heterogeneous, stored in autonomous and distributed sources. Thus, it becomes necessary to introduce an intermediate and intelligent system. This one should satisfy the following requirements: on one hand it should provide a single point of access to these sources, on the other hand, it should make the aspects of autonomy, distribution and heterogeneity transparent.

2. RELATED WORK

Harinarayan et al. [21] presented a greedy algorithm for the selection of materialized views so that query evaluation costs canbe 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.The requirement of creating materialized views has been also found useful in the other large data centric applications such as data warehouse or data mining. The importance of materialized view is that it is stored permanently in storage elements. Whereas, ordinary views are loaded with data every time it is called. Thus in real life applications materialized views are found to be more suitable to reduce query execution time. Materialized view creation involves several issues to consider. However, the main concern is to ensure availability of higher amount of user requested data directly from materialized views. Automated selection [13] of materialized views in large data oriented application is desirable for dynamic changes. A survey work is carried out here to give the idea of how the different methodologies have been applied over the years to generate materialized views. V. Harinarayan et. al. applied greedy algorithm [1] to select materialized views to optimize query evaluation costs of "data cubes". This work does not address view maintenance and storage issues. A heuristic algorithm [2] was described to utilize Multiple View Processing Plan (MVPP) to obtain an optimal materialized view selection.

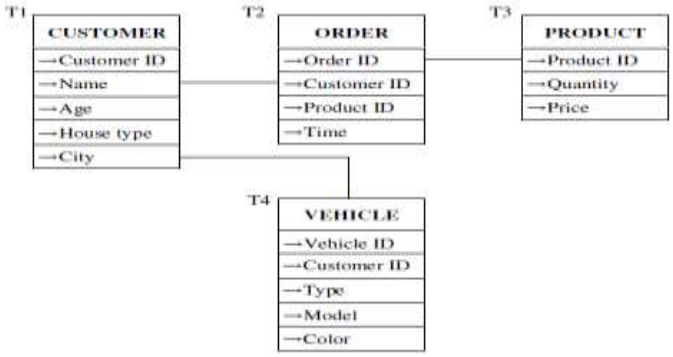
The objective of this work is to achieve the combination of good performance and low maintenance cost. This research work is motivated to measure the relationship among several attributes in the form of a quantitative metric using a robust mathematical model, which is implemented here using line fitting algorithm.

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.

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, *DW* and a user's table (*UT*) 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, *DW* 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 (T1), order (T2), product (T3) and vehicle (T4). Here, 'order' (T2) 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: T2 →T1; T2 →T3 and T4→T1, where Ti →Tj means that the foreign key of table Ti is the primary key of Tj.



A. Benefit of Materialized View

The purpose of materializing the view is to increase the efficiency of query, also representing the decrease of cost. Given the query collection Q, the cost is total time of querying Q. And for a given view V, the difference of the cost between pre-materialization and after materialization is called the efficiency of view V.

In order to response a query Q, we should first find the view with equal grade of query Q. If the view has been materialized, we can read it directly, otherwise we can get it from the minimal materialized view of the query Q. In conclusion, a view should be identified to carry a query, we could suppose this view as V, Harinarayan [5] found out through an experience, that time is proportional to size |v| when we use view V to response query Q, and the |v| can be seen as the cost of the query. However, it is not simple to estimate the size of a data joint, documentation [4] discussed many methods to solve this problem. To identify classes of data most queried, an algorithm called CM (Cluster and Merge) [12] [13] [14] was proposed.

This algorithm receives as input a description of the distribution of user queries, and provides in output a set of classes, compact, representing data patterns present in those queries.

This algorithm has three steps:

- Classification of queries: it is to determine the categories of data which the user is interested.
- Classification of attribute groups: it is to determine the groups of attributes for each class.
- Merging classes: merge the data classes to make the classes that are most compact.

A. Classification of queries

In this step, the algorithm determines the set of subclasses of each query, and the subclasses of interest. Those are inserted in

the ontology if they are not already present. For example, a query of the form:

SELECT A FROM S WHERE P

Where A is the set of attributes queried in S, P = {P1, P2, Pn} predicates specifying constraints of the query, and SP subclass of S satisfying P. All subclasses of interest is expressed by {Sp1, Sp2, ..., Spn}, where Pi are forming individual predicates P, and Spi subclass of S satisfying Pi For example, consider the following query:
SELECT population, area FROM_COUNTRY
WHERE region = "Europe" AND government = "Republic"
In this query, the subclasses of interest are "European Country" and "Republic Country".

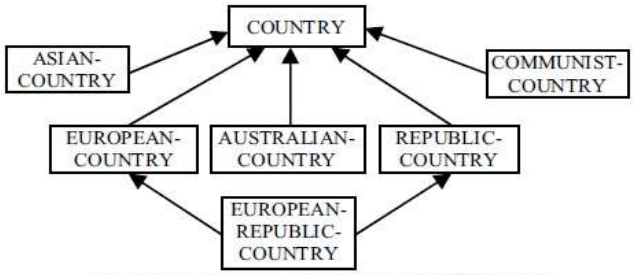


Fig.1. The ontology of subclasses of COUNTRY.

B. Classification of attribute groups

After the step of classification of queries, an ontology of classes is obtained, and for each class, the attribute groups queried and with what frequency. In this step, CM merges the attribute groups with similar frequencies to reduce the number of groups for each class. The merger is accomplished if the difference between their frequencies is less than a threshold known as CLUSTER-DIFFERENCE.

C. Merging classes

It is important that the number of data classes be reduced to improve queries processing. Thus, we should merge them when it is possible. Consider, for example, the classes of information:

- EUROPEAN-COUNTRY, {POPULATION, AREA}
- ASIAN-COUNTRY, {POPULATION, AREA}
- AFRICAN-COUNTRY, {POPULATION, AREA}
- N.AMERICAN-COUNTRY, {POPULATION, AREA}
- S.AMERICAN-COUNTRY, {POPULATION, AREA}
- AUSTRALIAN-COUNTRY, {POPULATION, AREA}

Finding the parameters of view selection cost

Then, we have built one user's table, *UT* to find the frequency of every query for computing the query frequency cost. The user's table is denoted as, *UT* 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. 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 *UT* for finding the frequent queries and their corresponding support value. 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.

4. OUR APPROACH

Our solution is an approach based on user behavior and their interactions with the system, particularly the distribution of their queries, to create the set of views to materialize.
It is divided into two phases:
• Creating candidate views for materialization: Based on the distribution of queries previously posed on the system, we extract all data most queried by users. These data are then classified as views.
• Selecting views to materialize: In this step, we select from among all the views created in the first phase, those that will be effectively materialized.

A. Creating candidate views for materialization

In our approach, we assumed that a data pattern is present in user queries, i.e. certain categories of data will be queried more frequently than others. Thus, it will be very useful to extract these patterns given the basis of which we will create the candidate views for materialization. This phase is divided into three steps:

• Extracting the attributes of interest
• Creating schemas of views.
• Extracting the most frequent constraints for each attribute and creating views.

We now describe the steps of our approach in more detail.

1) Extracting the attributes of interest.

Generally in a mediation system, a global schema representing the domain of use is provided. It is in terms of the latter are expressed the user queries. We analyze these queries to determine, among all the attributes of this schema, those in which users are interested, i.e. the most frequent attributes. For that, we are based on a set of queries posed previously for identify the information most queried. Let SQ = {Q_1, Q2,.....Q_{NQ}} all queries taken as input, and SA_0= {A_1, A2,.....A_{NAO}} the set of attributes present in the global schema.
The frequency of the attribute Ai is expressed by:

$$FA_i = \frac{NA_i}{NQ_i}$$

Where NA_i is the number of appearance of the attribute A_i. The average frequency of attributes is expressed by:

$$FM = \frac{1}{NA_0}\sum_{i=1}^{NA_0} FA_i$$

The attributes whose frequency is lower than the average frequency will be eliminated. We then obtain the set SA= {A_1, A2,....A_{NA}} of attributes which appear in the candidate views, where NA is the number of selected attributes. To do this, we defined the procedure EXTRACT_ATTRIBUTES.

```
SA={};
EXTRACT_ATTRIBUTES(SQ){
  SA0=get_All_Attributes(SQ);
     NA0=cardinality(SA0) ;
     FOR (i=1 TO NA0){
          NAi =0 ;
          FOR (ALL Q IN SQ){
               S=get_All_Attributes(Q) ;
               IF (Ai IN S){
                    NAi = NAi+1;
               }
          }
          FAi=NAi/NA0;
     }
     FM=Average(FAi)
          1≤i≤NA0
     FOR (i=1 TO NA0){
          IF(FAi>FM){
               SA = SA UNION {Ai}
          }
     }
}
```

The attributes obtained in this step will appear in the candidate views. It should then be collected in compact classes, or what we called the "views schemas". Thus is that we present in the next section.

2) Creation of views schemas

The problem of creating schemas is equivalent to a classification problem. Thus, we seek to create a compact set of attributes classes.

Different classification algorithms have been proposed. The most popular is k-Means. It partitions a dataset or points in k classes. Each class is represented by a center of gravity or centroid. From these centers, k-means calculates the distances to various points and they are attributed to the nearest centroid. Consider for example a dataset x_1, x_2,..., x_N to classified into k disjoint classes Ci where i∈[1,k], each one contains Ni points where Ni ∈[0,N]. Thus, k-means is in three steps:

(i) Initialize randomly k center c1, c2, ck by data points. For each point xt, and all k classes, repeating

Steps (ii) and (iii) until the sum of intra-classes distances cannot decrease.

(ii) Calculate the distance from xt to different cluster centers and assign it to that who's centroid is the nearest.

(iii) Recalculate the centroids of the different classes. In our case, it is impossible to define the centroids, and so we will not have the ability to calculate the distances.

3) Extraction of constraints

Until now, we have defined the attributes most queried. We have gathered these attributes in compact classes. We should then define the values (or constraints) of attributes of each class.

This phase is divided into three steps:

- Extraction of the most frequent values for each attribute.
- Definition of the most frequent instances of each class.
- Merging of instances of each class in a single.

B. Selection of views to materialize

The views created in the first phase of our approach cannot be all materialized. Indeed, the space for materialization, the frequency of update and the cost of access to sources is critical. A set of selection criteria have been defined in [Hadi 2012] and [Bichutskiy 2006], namely:

• The frequency of change: the views that rarely change are good candidates for materialization.

• The size of views: the views of small sizes are favoured for materialization than large ones.

• The availability of sources: The views, whose data resides in sources that are rarely available, should be materialized.

• The cost of access: the materialization of views whose data resides in sources with a high cost of access will improve the system performance.

Thus, a view will be materialized, if it satisfies at least two criteria.

III. Dynamic Modulating Strategy of Materialized Views

After the given space is filled with materialized views, and finishing the selection of materialized view though PBUS algorithm. It is also needed to adjust the dynamic modulating strategy of materialized views with the specific situation. The reasons are as follows: (1) The new views should be materialized to meet new queries (2) Take two views v and u in the view sets, among which v is materialized before u. When it is v that is materialized, for the whole query Q of data warehouse, the relatively benefit value is greater than the value of u. However, after more and more views are materialized, the benefit value of view v could be smaller. That is because many virtual views of response query choosing v would look for materialized views that have smaller cost and partial order to response, which lead to larger benefit of view u than view v. Because of the reasons above, DSAMV algorithm was put forward, which further optimize the selection on the foundation of PBUS algorithm.

DSAMV algorithm: (Dynamic Selection Algorithm of Materialized View)

{

Input: The initial selected materialized view set S through PBUS algorithm, multidimensional grid A
Output: Optimized set of materialized view S
Space=0:
do
{ For view of S, rank ordering from small to large according to the relative effectiveness $B_Q(v, S)/|v|$, then get {vi};
Choose the largest view w which belongs to A but $B_Q(v, S)/|v|$ of S;
Accumulate the view sizes that have minimum benefit

until $Space + \sum_{i=1}^{n} |v_i| > |w|$;

if $\sum_{i=1}^{n} B_Q(v_i, S) > B_Q(w, S)$ break;

else
{vi={v1} ∪ {v2} ∪ ... ∪ {vn}
S={S-{ vi}} ∪ {w}

Space= $\sum_{i=1}^{j} |v_i| - |w|$;

}
}

While($\sum_{i=1}^{n} B_Q(v_i, S) < B_Q(w, S)$)

Return S
}

Suppose the efficiency time *B* (*v, S*) *Q* of each materialized view v as m, and there are n views in materialized view S, then we can count the calculation efficiency and order time, which is O(mn + nlogn). And time complexity of the DSAMV algorithm does not exceed O (mn2 + n2logn). There are two view adjustment methods; the batch adjustment which varies from time period and the timely adjustment when there is an occurrence of a query. It is a more natural way to choose one of the methods though.

5. 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 recomputing 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.

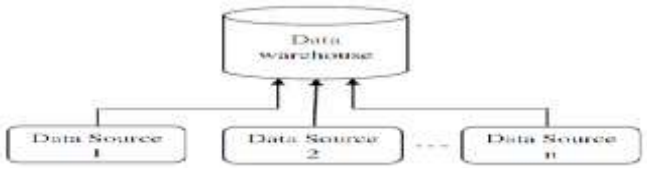


Figure 3: Data warehouse updating from the multiple sources,

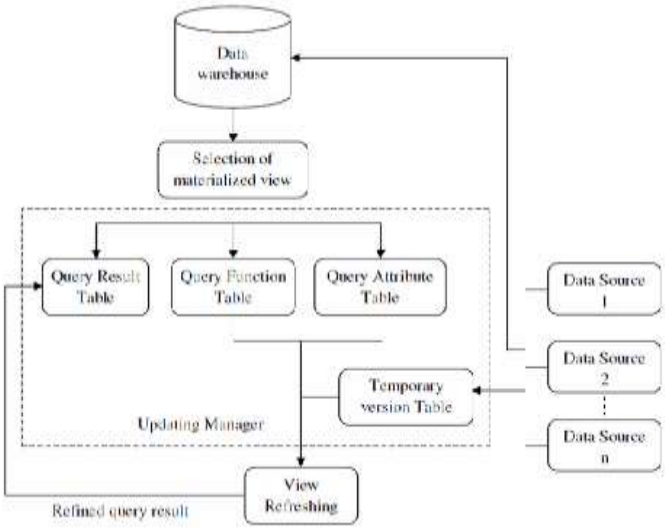


Figure 4: View Maintenance process

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

5.2 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 AT: 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 FT*:* 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 TT: 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 RT*:* 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.

6. CONCLUSION

The key to improve the efficiency of data warehouse query is to choice view materialization accurately. While the method of PBUS and PBS algorithm can only make an initial option of the best benefits of the view, without the ability to adjust over time. On the basis of the PBUS algorithm, we proposed DSAMV algorithm, which chooses the best view for materialization dynamically, and further improve the efficiency of OLAP queries. 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. The research on materialized view creation is getting more importance over the time as the numbers of users as well as the transactions are increasing for any real life system. This paper proposes a novel method of creating materialized views by analyzing the association among different attributes in the given relation using statistical method. This presents a quantitative measure of degree of relationship among the attributes. The knowledge of this quantitative measure helps to build the materialized view. Attribute is one of the most granular level of data representation. As this analysis is entirely based on the lowest level of granularity, the accuracy of the constructed materialized views are high. Moreover the proposed methodology is independent of the application areas. Hence it is applicable to any data-centric system. In our approach, the views are created based on a number of queries posed previously on the system. This is done one time. So that, the views are unchangeable during the system use. However, the distribution of user queries can change over time. Thus, we propose as a perspective, adding a dynamic aspect to our approach for taking into account the evolution of the distribution of user queries. The distribution of user queries is the only factor on which we were based in the personalization of the system. it is very interesting to exploit the user profile built, implicitly or explicitly, for each user in the process of personalization.

7. REFERENCES

[1] Dr.T.Nalini, Dr.A.Kumaravel, Dr.K.Rangarajan,"A Novel Algorithm with IM-LSI Index For Incremental Maintenance of Materialized View" JCS&T Vol. 12 No. 1 April 2012

[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] 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.

[4] Yang, J., Karlapalem. K. and Li. Q. (1997). A framework for designing materialized views in a data warehousing environment. Proceedings of the Seventieth IEEE International Conference on Distributed Computing systems, USA, pp: 458.

[5] 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.

[6] A. Shukla, P. Deshpande, and J. F. Naughton, "Materialized view selection for the multidimensional datasets," in Proc. 24th Int. Conf. Very Large Data Bases, 1998, pp. 488–499.

[7] Wang, X., Gruenwalda. L., and Zhu.G. (2004). A performance analysis of view maintenance techniques for data warehouses. Data warehouse knowledge, pp: 1-41.

[8] Mr. P. P. Karde, Dr. V. M. Thakare. "Selection & Maintenance of Materialized View and It's Application for Fast Query Processing: A Survey". Proceedings of International Journal of Computer Science & Engineering Survey (IJCSES) Vol.1, No.2, November 2010

[9] Abdulaziz S. Almazyad, Mohammad Khubeb Siddiqui. "Incremental View Maintenance: An Algorithmic Approach". Proceedings of International Journal of Electrical & Computer Sciences IJECS-IJENS Vol: 10 No: 03