Materialized View Selection for a Data Warehouse Using Frequent Itemset Mining

Mohammad Karim Sohrabi*, Vahid Ghods
Department of Electrical and Computer Engineering, Semnan Branch, Islamic Azad University, Semnan, Iran.

* Corresponding author. Tel.: +98(23)33654040; email: Amir_sohraby@yahoo.com
Manuscript submitted April 1, 2015; accepted June 10, 2015.
doi: 10.17706/jcp.11.2.140-148

Abstract: Data warehouses are subject oriented, consolidated, integrated, and time variant repository of possibly heterogeneous data. A data warehouse is used to response to on-line analytical queries over the millions records of data in an acceptable time. Since a data warehouse often has millions of records of data, it is an important challenge how we can reduce the time of on-line analytical processing. One of the most important issues which address this problem is the view materialization. Each sub-query results an intermediate table, called virtual view, which is used to find final result of the analytical query. These virtual views often are commonly used to response to several analytical queries. We can materialize such views to prevent multiple redundant computations and thus lead to reduction in response time of queries. The constraint of storage memory on one hand, and the maintenance cost of materialized views when the source data are updated on the other hand, cause that it is impossible to materialize all or even large part of views. Therefore, selection of a proper set of views to materialization plays a major role in performance. There are many methods of view selection to materialization which uses different techniques and frameworks to select optimal set of views to materialization. In this paper, we present a new efficient method to conduct selecting proper set of views to materialization using a frequent itemset mining approach. In our algorithm, the set of given queries is transformed to a transaction database where a transaction corresponds to a query and items of a transaction are the original query's predicates. Our performance study showed that this algorithm outperformed substantially the best former algorithms.

Key words: Data warehouse, on-line analytical processing (OLAP), view selection, view materialization, frequent itemset mining.

1. Introduction

A data warehouse is a collection of heterogeneous and distributed data which are collected from different data source into an integrated framework. Data warehouse, which define as a subject oriented, integrated, non-volatile and time variant collection of multiple data, provides an appropriate platform for On-Line Analytical Processing (OLAP) [1]. View materialization is a widely used strategy data warehouses to improve the analytical query processing performance. Since access to a materialized view is faster than computing the view on demand, using materialized view can speed up the analytical query processing in a data warehouse. So it will be desirable if we can materialize all the views of a data warehouse. However there are two major reasons which make it impossible: first limitation of storage space that causes all materialized views cannot be stored. Materialized views consume a large amount of disk space and the available practical disk space does not allow materializing all views. Second, maintenance of materialized views in case of

updating original base relations will be very expensive because when a base table changes, we should refresh all affected materialized views. Therefore we should select part of views to materialize. In order to acquire a quick response to analytical queries, selection of the proper set of views to materialize in the data warehouse is essential.

There are several cost metrics associated with the materialized views selection problem that should be addressed by proposed view selection methods. A proper view selection algorithm usually should consider the query execution frequencies, base relation update frequencies, query access costs, view maintenance costs and the system's storage space constraints as the most important issues related to selected set of materialized view.

Proposed materialized view selection methods often have used a special framework which had considered some or all these metrics. There are two main steps in a materialized view selection frameworks: in the first step, the method should find the candidate views for materialization. View selection methods usually use a Directed Acyclic Graph (DAG) of queries to obtain the candidate views in this step. This DAG represents the dependence relation of queries and is used to detect common sub-expressions between different queries. The most commonly used DAGs for materialized view selection are: AND/OR view graph, Multiple View Processing Plan (MVPP), and data cube lattice. Other identification techniques such as syntactical analysis of the workload and query rewriting also can be employed instead of multi queries DAGs which have been used by fewer view selection methods. In the second step, the view selection method constructs the set of views to materialize under the resource constraints, including storage space constraint and view maintenance cost constraint, using a heuristic algorithm. The most well-known algorithms have been proposed in the literature to solve the view selection problem, can be classified into four categories: deterministic algorithms, randomized algorithms, hybrid algorithms, and constraint programming.

As the main contributions of this paper, a new view selection method is developed which introduce a novel framework based on frequent itemset mining techniques to select the optimal set of views to materialize, such that, the sum of materialization cost and maintenance has been minimized.

The remaining of the paper is organized as follows: In Section 2, we discuss the existing studies related to the view selection for materialization methods. We represent our new view selection method and its algorithm in Section 3 and conduct experimental study in Section 4. Finally we conclude the study in Section 5.

2. Related Works

The Multiple View Processing Plan (MVPP) is a directed acyclic graph which presents the processing plan of a set of analytical queries. The MVPP based algorithms are one of the important attempts to address the view selection problem in data warehouses. A heuristic algorithm has been developed based on MVPP in [2], which find a solution based on individual optimal query plans. A new approach for materialized view selection based on candidate selection and enumeration techniques is presented in [3] that selects materialized views and indexes by searching over the reduced space of candidate materialized views at a fraction of enumeration cost. Use of evolutionary algorithms is another approach to materialized view selection based on multiple global processing plans of queries. Zhang *et al*. presented an evolutionary algorithm and combined with heuristic algorithm to preserve gains of both methods and attain better performance than them [4]. View Relevance Driven Selection (VRDS) algorithm has been developed to minimize total processing cost including query processing and view maintenance cost by selecting best views for materialization [5]. This algorithm performed better than greedy algorithms, which are focused mainly on updates, and MVPP based heuristic algorithms, which are concentrated only on materialized view features.

A general AND-OR view graph was considered in [6] and some greedy polynomial-time heuristics for selecting materialized views to optimize total query response time under a disk space constraint were represented. Another view selection under disk space constraint has been presented in [7] which is used A* algorithm to find optimal solution when the disk space is low. ASVMRT (Algorithm for Selection of Views to Materialize using Reduced Table) used the clustering methods to reduce tables in the data warehouse based on attribute-values density and considered the combination of reduced tables as materialized views instead of a combination of the original tables [8]. Faster computation time, reduced storage space, and better performance than former algorithms were reported advantages of this algorithm, however maintenance of reduced table was not considered in the work. Also in [9], a framework for materialized view selection has been proposed that exploits clustering to determine clusters of similar queries. This strategy shares the storage space between indexes and views.

Reference [10] proposed a two phase optimization method which was a combination of simulated annealing and iterative improvement, with the use of MVPP and addressed the tradeoff between performance and view maintenance. Another using of simulated annealing for materialized view selection has been proposed in [11] in which a parallel simulated annealing algorithm selecting views from an input MVPP. Although the combination of parallel simulated annealing and MVPP has been resulted in increasing the quality of obtained set of materialized views and improving the query processing time and decreasing view maintenance cost, it trapped to local minimum in some cases.

Since the frequency of access to a view is not constant over the processing time, the performance of static view selection methods can be deteriorated. Dynamic view selection methods attempted to overcome this problem. EMVSDIA (Efficient Materialized View Selection Dynamic Improvement Algorithm) is a two step algorithm that has been discussed in [12]. This algorithm dynamically adjusted the set of selected views in case of the reduction of efficiency, by substituting views which own large query probability. Another dynamic approach to view selection has been proposed in [13] which have determined the existing materialized views that are affected by adding new queries rather than all existing resources and so have re-optimized MVPP and have improve the total query processing cost of it. Also a two-phase optimization algorithm have been proposed in [13] which have reduced the sum of query processing costs and materialized view maintenance costs.

Reference [14] has used merging of incoming query as the global common sub-expressions of the previous merging to avoid a huge search space which some combination would not be considered. Since the global optimization is not acquired in this method, they rewrote some queries by using common sub-expression among queries to gain more optimal query processing cost.

Recently, a novel approach has been represented to deal with materialized view selection problem. Frequent pattern mining techniques have been used to address this problem [15]-[17]. One of the important problems in data mining is discovering frequent patterns from transaction databases, where each transaction has a set of items. Extraction of frequent patterns plays an essential role in numerous applications. Since several efficient methods have been proposed recently for frequent pattern mining [18]-[20], using this approach could be a proper solution to overcome the materialized view selection problem.

3. View Selection Method

In this section, we represent our materialized view selection method which is constructed based on frequent itemset mining techniques. First we describe how we can model the materialized view selection problem as the input for frequent itemset mining environment and extracting the proper set of selecting views corresponding to extracted frequent itemsets. Then we represent our MVFI algorithm.

3.1. Mining Technique for Materialized View Selection

Frequent itemset mining is a type of pattern mining which operates on transactional databases' log and extracts the itemsets which their frequency aren't less than a user defined minimum support threshold, called minsup. We can use frequent itemset mining to determine the most common sub-expression in queries of a data warehouse and so address one of the most important factors of selecting materialized views i.e. query execution frequencies. In this subsection, we first represent a model to map the query processing space of a data warehouse to a transaction database. Then we explain how we can mine the frequent itemsets of this transaction database, and how these extracted frequent itemset help us to determine more frequently used views of the corresponding data warehouse which should be select to materialize. Note that, there are other important issues related to selected set of materialized view such as original base relation update frequencies and view maintenance costs which effect on the view selection process which can be considered as future works.

Let Q={$Q1$, $Q2$, …, Qn} be the set of queries of the data warehouse which are operating on a set of relation, called R={$R1$, $R2$, …, Rm}. We want to select a proper set M of views to materialized from the set of all possible views, called V={$V1$, $V2$, …, Vk}, where the amount of required storage space of view Vi is shown by Si, for each $1 \le i \le k$. Given the storage space S, the sum of the required space of selecting views should not be larger than S. Suppose A={$A1$, …, Ab} be the set of the all dimensions which are used in the relations of data warehouse such that each relation Ri($1 \le i \le m$) include a number of dimensions. It is obvious that each query Qj($1 \le j \le n$) are working on some relations and are selecting some dimension of those tables. On the other hand, we can optimize each of our query using equation $\sigma_{P_1 \wedge P_2}(R_1 \bowtie R_2) = (\sigma_{P_1}(R_1)) \bowtie (\sigma_{P_2}(R_2))$, in which, $R1$ and $R2$ are two base relations, and $P1$ and $P2$ are the predicates on relations' individual attributes respectively. Therefore, we can transform all of our queries to a normal form in which, all the "select" operators directly operates on based relations, and the "join" operators work on the select's results. The set of all predicates on the relation Ri ($1 \le i \le m$) can be displayed as Pi={$P(i, 1)$, $P(i, 2)$, …,$p(i, mi)$}, in which, $P(i, j)$ means the jth predicate of relation Ri, and mi is the number of predicates on the relation Ri in all queries of Q. For illustration, consider an example in which we have a data warehouse with 3 relations $R1$($A1$, $A2$, $A3$, $A4$, $M1$) and $R2$($A4$, $A5$, $A6$, $M2$, $M3$) and $R3$($A6$, $A7$, $A8$, $A9$, $M4$), where "Ai"s are relations' dimensions and "Mi"s are relations' measurements.

If a query includes several relations which are joined together and several predicates which are operates on the join result, then the normalized form of query includes several sub-queries, each of which include on relation and some predicates on that relation such that joining of these sub-queries constructs the main normalized query. Fig. 1 shows an example query on our sample relations and optimized of the query.

Q_k	Normal (Optimized) form of Q_k
Select A_1, A_2, **Sum**(M_4) **From** R_1, R_2 , R_3 **Where** $R_1.A_4=R_2.A_4$ **and** $R_2.A_6=R_3.A_6$ **and** $A_5=2$ **and** $A_3>7$ **and** $A_8>6$	**Select** A_1, A_2, **Sum**(M_4) **From** **(Select** * **From** R_1 **Where** $A_3>7$) **Natural join** **(Select** * **From** R_2 **Where** $A_5=2$) **Natural join** **(Select** * **From** R_3 **Where** $A_8>6$)

Fig. 1. An example of optimizing a query.

Since several queries may be answered by a data warehouse, we can extract all sub-queries on each individual relation and construct set Pi corresponding to relation Ri. Now, or framework is ready to operating

frequent itemset mining techniques.

Consider an example in which there are five predicates on relation *Ri*, namely, *P*(*i*, 1), *P*(*i*, 2), *P*(*i*, 3), *P*(*i*, 4), and *P*(*i*, 5). It is obvious that different combinations of these predicate may be used in different sub-queries which can contain one, two, three, four, or all of these five predicates. The search space of the predicates of relation *Ri* has been demonstrated in Fig. 2. This a horizontal search tree for frequent itemset mining.

Using the apriori principle, we can prune this search tree based on an appropriate minsup. Our minsup can be determined based on the accessible storage space and the average required space of sub-queries (views). Based on apriori principle, a super pattern of an infrequent pattern is not frequent. Hence, if a node of tree is not frequent, then that node and all children will be pruned. For example, infrequency of node {*P*(*i*, 1), *P*(*i*, 3)} of the search tree of Fig. 2 means that the number of queries which use sub-query [Select * From *Ri* Where *P*(*i*,1) and *P*(*i*,3)], is not enough (is not equal or more than minsup) to convince us to materialize corresponding view of this sub-query. Therefore, we do not need to continue investigating nodes *P*(*i*, 1), *P*(*i*, 3), *P*(*i*,4)}, {*P*(*i*, 1), *P*(*i*, 3), *P*(*i*, 5)}, and {*P*(*i*, 1), *P*(*i*, 3), *P*(*i*, 4), *P*(*i*, 5)}.

Similarly, an individual search tree can be constructed corresponding to each of relations of the set *R*. Our frequent itemset mining method extracts the more frequently accessed views using a two phase mining approach. In the first phase, the access frequency of relations is determined and normalized as a number between 0 and 1. For example, suppose that there are four relations in a data warehouse, where relation *R*1 is accessed 27 times by queries (e. g. 3 different queries each of them executed 9 times) and relations *R*2, *R*3, and *R*4 are accessed 31, 19, and 28 times, respectively. The number of all access to relations is 27+31+19+28=105 and the normalized access frequencies to relations are 27/105, 31/105, 19/105, and 28/105 respectively. In the second phase, we construct the search tree of the predicates of each relation and mine it with its individual minsup, which is calculated by multiplying the general minsup and the relation's normalized access frequency.

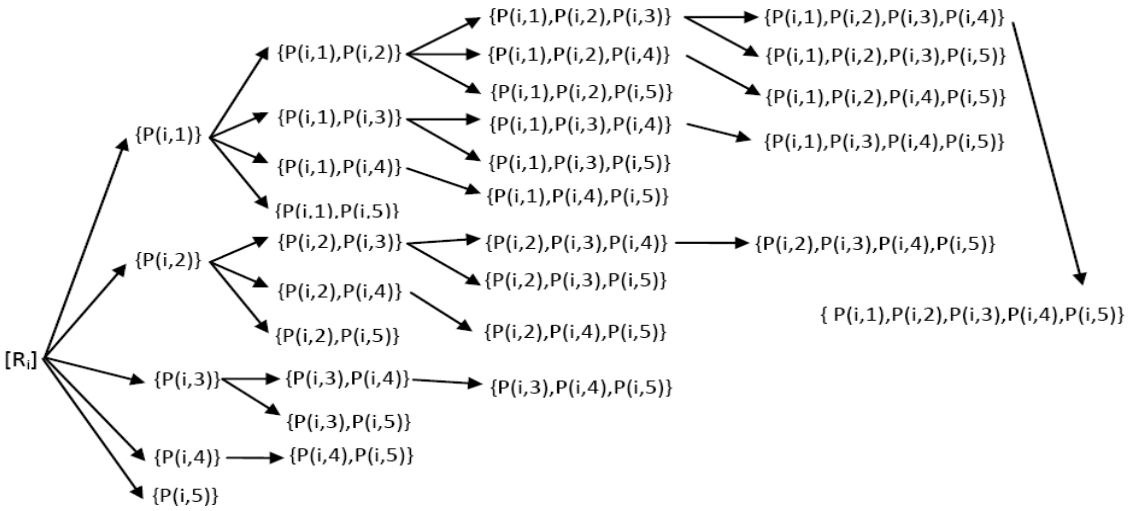

Fig. 2. The horizontal tree for frequent itemset mining.

3.2. Algorithm

In this section, we represent the algorithm of our Materialized Views selection based on Frequent Itemset mining (MVFI) and describe its operation. We first have pruned the set of candidate views (set of predicates) and exclude the predicates which the result of operating them on their corresponding relation is larger than the available storage space (*S*). Therefore the input set *P* for the first call of MVFI includes only predicates which their corresponding views will be fit in the system's storage space.

Fig. 3 shows the Materialized Views selection based on Frequent Itemset mining (MVFI) algorithm.

Input parameters of this algorithm include *R*, the set of all relations of data warehouse, m, the number of members of *R* which is the number of data warehouse's relations, AF, an array containing normalized access frequencies of relations of *R*, *P*, a set with m members each of which is a set of predicates related to corresponding member of *R*, and minsup, the minimum support threshold to extract frequent patterns.

For each relation of *R*, the MVFI algorithm calculates the corresponding minsup and call the procedure *MaterializedViewMining* to extract the most frequently used views which are related to that relation and its predicates considering the systems storage space and added the extracted view set *V*, which is the initial set of candidate materialized views. Since for each relation of *R*, MVFI extracts its frequently used views assuming use of all storage space, a final selection procedure is required to select best candidate views of all relations. *ViewSelection* is the procedure which does this operation and eliminates not qualified views from *V*. The Refined *V* is the final solution. Fig. 4 shows the *MaterializedViewMining* algorithm.

MaterializedViewMining is a recursive algorithm which is mine the candidate views search tree. This algorithm is constructed from two main parts. In the first part (in the main if block of algorithm), the algorithm check if selected views of *V* exceeding limitation of space. In the second part, the view is inserted to output set. Then, the node is expanded and its children are created and each child is called recursively.

```
Procedure MVFI
Input
   R: set of base relations of the data warehouse
   m: integer // number of relations
   AF: array of normalized access frequencies of relations
   P: set of m sets // include all atomic predicates on the relations of R
   S: integer      //amount of Storage Space
   Minsup: integer //minimum support threshold
Var
    i: integer;
   V: set of selected views to materialization
    min: integer;
Begin
   For i= 1 to m do
   Begin
                min := AFi ✕ minsup
                V=V ⋃ MaterializedViewMining (Pi, S, min, 1);
   End;
   ViewSelection (V);
End;
```

Fig. 3. MVFI algorithm.

```
Procedure MaterializedViewMining
Input
   P: set of predicates related to the input relation
   S: integer       //amount of Storage Space
   min: integer    //minimum support threshold corresponding to the input relation
   k: integer      //the number of predicate which its corresponding node in the search tree should be mined
Output
    V: set of candidate views to materialize
Begin
    if (sizeof(V) >= S) then return(V);
   Insert Pk to V;
   if (support(Pk)>=min) then    return(V ⋃ MaterializedViewMining(P, S, min, k+1));
End;
```

Fig. 4. Materialized view mining algorithm.

4. Experimental Results

In this section, we study the performance of the proposed algorithm. All our experiments were performed

on a computer with a 2 GHz core i3 CPU, 4 GB RAM and 500 GB hard disk. All the reported runtime include both computation time and IO time. ARMMVVM [16] is the most recent materialized view selection algorithm and has already shown its better performance than other materialized view selection algorithms. So we only compared our algorithms with ARMMVVM algorithm. In [16], a new parameter GAIN MEASURE (GM) has been introduced to measure the gain from work load to the resultant queries.

The gain measure of a view is computed as the sum of the saving in query cost for each view over answering from the base view. If a set D of views is chosen for materialization, the gain of D is the sum of the gain of all views in D. If D is a set of views that have already been selected for materialization, the gain of views is concerned with how materializing a view s improves the cost of computing other views, including it. Let $C(s)$ be the cost of computing another view from s. $C(s)$ is the number of records in s. The gain of s with respect to the D, $G(s, D)$, is defined as follows:

1) For each view $s \leq t$, $G(s)$ defined as $G(s)= C(r)-C(t)$ when r is the least cost view in D such that $s \leq r$ and $C(t)<C(r)$. When $C(t) \geq C(r)$, then $G(s)$ will be 0.
2) $G(s, D)= \sum_{s<r} G(s)$

Precisely, for each view s that is a descendant of t, if computing s from s is cheaper that computing s from any other view in the set D, then pre computing t gains s.

Using Gain Measure two approaches MVFI and ARMMVVM are compared and results are presented in Fig. 5. We also use the data warehouse of [16] which has five dimension tables and one fact table, for this compare. Fig. 5 shows the result of running two algorithms MVFI and ARMMVVM on this data warehouse.

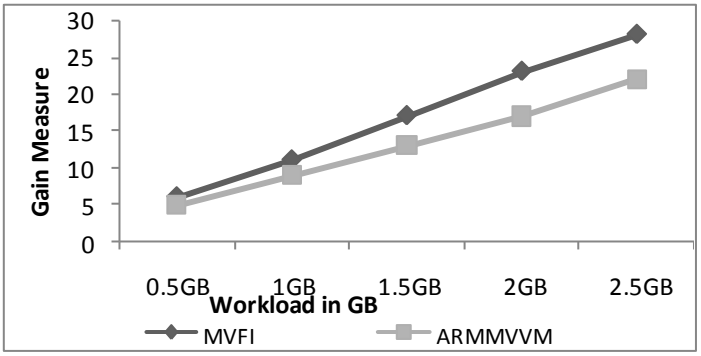

Fig. 5. The result of running algorithms on the data warehouse.

Fig. 5 shows the comparison between the MVFI algorithm and ARMMVVM using the gain measure. It is found by experiments that the gain using the MVFI is more than ARMMVVM algorithm.

5. Conclusion

Data warehouses need to answer many managerial level, complex, and analytical queries which require advanced computing techniques and time consuming processes. Since the response time of such queries should be acceptable, intermediate or final result of some of these queries should be stored as materialized views. Storage space constraint and materialized views maintenance costs make it impossible to materialize all views of data warehouse. Therefore, we should select the optimal set of views which their materialization causes the most speed up in query response and the less cost to maintenance. Hence, the selection of views to materialize is one of the most important issues in designing and developing a data warehouse. One of the major problems in materialized view selection is to determine query execution frequencies. The most common sub-expressions of queries makes the most frequently used views which should be materialized. A useful and appropriate way to address this problem is using frequent itemset mining approaches. In this paper, we presented a novel frequent itemset mining-based method to conduct materialized view selection

in data warehouses. Our experimental result showed that our algorithm achieved very appropriate efficiencies on various input datasets.

References

[1] Inmon, W. H. (1996). *Building the Data Warehouse* (2nd ed.). John Wiley and Sons, Canada.

[2] Yang, J., Karlapalem, K., & Ling, Q. (1997). Algorithms for materialized view design in data warehousing environment. *Proceedings of the 23rd VLDB Conference* (pp. 136-145).

[3] Agrawal, S., Chaudhari, S., & Narasayya, V. (2000). Automated selection of materialized views and indexes for SQL databases. *Proceedings of 26th International Conference on Very Large Databases* (pp. 496-505).

[4] Zhang, C., Yao, X., & Yang, J. (2001). An evolutionary approach to materialized views selection in a data warehouse environment. *IEEE Transactions on Systems, Man, and Cybernetics-Part C: Applications and Reviews*, *31(3)*, 282-294.

[5] Valluri, S. R., Vadapalli, S., & Karlapalem, K. (2002). View relevance driven materialized view selection in data warehousing environment. *Australian Computer Science Communications*, *24(2)*, 187-196.

[6] Gupta, A., & Mumick, I. (2005). Selection of views to materialize in a data warehouse. *IEEE Transactions on Knowledge and Data Engineering*, *17(1)*, 24-43.

[7] Gou, C., Yu, J. X., & Lu, H. (2006). A* Search: An efficient and flexible approach to materialized view selection. *IEEE Transactions on Applications and Reviews, 36(3)*, 411-425.

[8] Yang, J., & Chung, I. (2006). ASVMRT: Materialized view selection algorithm in data warehouse. *International Journal of Information Processing Systems*, *2(2)*, 67-75.

[9] Aouiche, K., Jouve, P., & Darmont, J. (2006). Clustering-based materialized view selection in data warehouses. *Proceedings of 10th East-European Conference on Advances in Databases and Information Systems (ADBIS06)* (pp. 81-95).

[10] Phuboonob, J., & Auepanwiriyakul, R. (2007). Two-phase optimization for selecting materialized views in a data warehouse. *World Academy of Science, Engineering and Technology*, *19*, 277-281.

[11] Derakhshan, R., Stantic, B., Korn, O., & Dehne, F. (2008). Parallel simulated annealing for materialized view selection in data warehousing environments. *Proceedings of 8th International Conference on Algorithms and Architectures for Parallel Processing* (pp. 121-132).

[12] Zhou, L. J., Ge, X. B., Wang, L. S., & Shi, Q. (2009). Efficient materialized view selection dynamic improvement algorithm. *Proceedings of Sixth IEEE International Conference on Fuzzy Systems and Knowledge Discovery* (pp. 294-297).

[13] Suchyukorn, B., & Auepanwiriyakul, R. (2013). Dynamic materialized view selection using 2PO based on re-optimized multiple view processing plan*. International Journal of Advancements in Computing Technology*, *5(14)*, 150-167.

[14] Suchyukorn, B., & Auepanwiriyakul, R. (2013). Re-optimization MVPP using common sub expression for materialized view selection. *International Scholarly and Scientific Research & Innovation, 7(7)*, 1114-1121.

[15] Nalini, T., Kumaravel, A., & Rangarajan, K. (2012). A novel algorithm with Im-Lsi index for incremental maintenance of materialized views. *Journal of Computer Science & Technology*, *12(1)*, 32-38.

[16] Rajyalakshmi, P. R., & Reddy, S. (2015). An association rule mining for materialized view selection and view maintanance*. International Journal of Computer Applications*, *109(5)*, 15-20.

[17] Nalini, T., Kumaravel, A., & Rangarajan, K. (2011). An efficient IMINE algorithm for materialized views in a data warehouse environment. *International Journal of Computer Science Issues*, *8(1),* 1694-0814.

[18] Sohrabi, M. K., & Ghods, V. (2014). Top-down vertical itemset mining. *Proceedings of the SPIE 9443*

sixth International Conference on Graphic and Image Processing.

[19] Sohrabi, M. K., & Barforoush, A. A. (September 2012). Efficient colossal pattern mining in high dimensional datasets. *Knowledge Based Systems, 33*, 41-52.

[20] Sohrabi, M. K., & Barforoush, A. A. (January 2013). Parallel frequent itemset mining using systolic arrays. *Knowledge Based Systems, 37*, 462–471.

Mohammad Karim Sohrabi was born in Iran in 1980. He received the B.S. degree in software engineering from the Computer Engineering Faculty, Ferdowsi University of Mashhad, Iran, in 2002. In 2005, he received the M.S. degree in software engineering from Computer Faculty, Amirkabir University of Technology (Polytechnic of Tehran), Tehran, Iran. He received Ph.D. degree in software engineering from the Computer Faculty, Amirkabir University of Technology (Polytechnic of Tehran), Tehran, Iran, in 2012.

He is the assistant professor of Engineering Faculty, Semnan Branch, Islamic Azad University, Semnan, Iran. He has more than 20 of journal and conference publications describing his research area. His research fields are data mining, data warehousing, software engineering, parallel and distributed systems, artificial intelligence and specially Frequent pattern mining.

Vahid Ghods was born in Iran in 1981. He received the B.S. degree in electronic engineering from Electrical Engineering Faculty, K N Toosi University of technology (KNTU), Tehran, Iran, in 2002. In 2005, he received the M.S. degree in digital electronic from Electrical Engineering Faculty, Semnan University, Semnan, Iran. He received Ph.D. degree in electronic from the Electrical Engineering Faculty, Science and Research Branch, Islamic Azad University, Tehran, Iran, in 2012.

He is the assistant professor of Engineering Faculty, Semnan Branch, Islamic Azad University, Semnan, Iran. He has more than 30 of journal and conference publications describing his research area. His research fields are image and video processing, machine vision, speech processing and recognition, artificial intelligence and specially OCR and handwriting recognition.