# Selection & Maintenance of Materialized View and It's Application for Fast Query Processing: A Survey

Mr. P. P. Karde[1]
Department of Information Technology,
HVPM's COET, Amravati.(M.S.)
P_karde@rediffmail.com

Dr. V. M. Thakare[2]
Post Graduate Department of Computer Science,
SGB, Amravati University, Amravati.
vilthakare@yahoo.co.in

## *Abstract*

*Quick response time and accuracy are important factors in the success of any database. In large databases particularly in distributed database, query response time plays an important role as timely access to information and it is the basic requirement of successful business application. A data warehouse uses multiple materialized views to efficiently process a given set of queries. The materialization of all views is not possible because of the space constraint and maintenance cost constraint. Materialized views selection is one of the crucial decisions in designing a data warehouse for optimal efficiency. Selecting a suitable set of views that minimizes the total cost associated with the materialized views is the key component in data warehousing. Materialized views are found useful for fast query processing. This paper gives an overview of various techniques that are implemented in past recent for selection of materialized view. The issues related to maintaining the materialized view are also discussed in this paper. Here some future aspects are also stated that might be useful for recent researchers.*

## *Keywords:*

*Data Warehousing, Query processing cost, Storage space, View Materialization, View Selection, View-Maintenance,*

## 1. Introduction

**E**very day, a large amount of information exchange into database systems of various organizations. There should be some provision for organizations to utilize such tremendous volume of data. Online analytic processing (OLAP) system provides some ways to take effective and efficient decision from such data. OLAP systems helps managers, executives and analyst make decisions by firing group-by SQL queries. Traditional databases are used for online transaction processing (OLTP) applications. These traditional operational database systems cannot process complex OLAP queries. The reason behind this is OLAP queries are involved in summarizing historical data that have been collected from different sources. OLTP applications simply access a small number of data from a single local operational database.

Data warehouse have been developed to overcome the weakness of traditional databases. A data warehouse is a very large database system that collects, summarizes, and stores data from multiple remote and heterogeneous information sources [1]. A data warehouse is a collection of materialized views, which are pre-computed and summarized from multiple operational data sources. These pre-computed materialized views are used to answer OLAP aggregate queries. This technique improves the OLAP query processing efficiency.

DOI : 10.5121/ijcses.2010.1202 

Materialized views play central role in the data warehouse, therefore recently database research community paying attention to the materialized view selection and maintenance. The attention is towards selecting a set of materialized views to pre-compute under specific resource constraints, such as disk space and maintenance time, in order to minimize the total query processing cost. In this paper various techniques used for materialized view selections are discussed in section 2. Section 3 gives brief overview of various materialized view maintenance techniques. Section 4 gives the comparison between all the discussed systems based on the various parameters that are considered during materialized view selection/maintenance and Section 5 gives the future aspect that might be useful for the researchers while designing and implementing materialized view selection and maintenance strategies.

## 2. Related Work

A number of parameters, including users query frequencies, base relation update frequencies, query costs, should be considered in order to select an optimal set of views to be materialized. Heuristic Algorithm (HA) [2] will set materialized views such that the total cost for query processing and view maintenance is minimal by comparing the cost of every possible combination of nodes. HA algorithm determines multiple view processing plans regardless of their query cost. HA may include the best processing plan because HA only works with the optimal plans.

In case of 0-1 Programming Algorithm [3] it considers all possible plans for each query to generate a single optimal view processing plan by applying 0-1 integer programming techniques. This works with all the possible join plan trees, therefore it can definitely get the best view processing plan in terms of query access frequency. In A* Heuristic Algorithm [4,5] , an AND-OR view graph and disk space constraints S is given, to deliver a set of views M that has an optimal query response time such that the total maintenance cost of M is less than by satisfying the constraint S. A* algorithm searches for an optimal solution in search graph.

Harinarayan et al. [6] 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. [7] 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. “AND-OR” view graphs were introduced to represent all the possible ways to generate warehouse views such that the best query path can be utilized to optimize query.

Ziqiang Wang and Dexian Zhang [9] proposed a modified genetic algorithm for the selection of a set of views for materialization. The proposed algorithm is superior to heuristic algorithm and conventional genetic algorithm in finding optimal solutions. Kamel Aouiche et al. [10] proposed a framework for materialized view selection that exploits a data mining technique (clustering), in order to determine clusters of similar queries. They also proposed a view merging

algorithm that builds a set of candidate views, as well as a greedy process for selecting a set of views to materialize.

# 3. Materialized View Selection

This section describes various algorithms used for materialized view selection. There are mainly two classes [1] of materialized view selection. First is materialized view selection under a disk space constraints and the second is materialized view selection under a maintenance time constraints. The problem of utilizing the limited resources disk space or maintenance time to minimize the total query processing cost comes under the materialized view selection with resource constraints.

## 3.1 Competitive A* Search Algorithm

Materialized view selection by considering the problem of disk-space constraints is implemented in terms of competitive A* algorithm is given in [1, 13]. Materialized view consumes large amounts of disk-space due to their large sizes that usually reach hundreds of terabytes. The available practical disk-space does not allow selecting all possible views to materialize. This motivates the problem of materialized view selection under a disk-space constraint. The benefit per unit space (BPUS) algorithm is developed to find out the greedy solution of disk-space constraint problem. In each iteration, the BPUS algorithm picks up one view with largest benefit per unit space among the remaining views to materialized, until the disk-space constraint is violated. Some lower bound has been considered for selecting materialized views. This lower bound can be very small and even be negative which makes the quality of solution not guaranteed well. Therefore a new competitive A* algorithm is developed to improve the solution. This algorithm basically works as follows:

### The Basic A* - Algorithm

**Input:** Lattice G and disk-space constraint S.
**Output:** A set of materialized views, M.

**1: begin**
**2:** Create an initial A* tree $T_{G;}$
**3:** Determine order of inserting views: ($v_1$, $v_2$,……, $v_n$);
**4:** Create a priority queue L;
**5: repeat**
**6:** Dequeue node x = ($N_x$, $M_x$) from L, where x has smallest (benefit having been acquired + estimated lower bound of expected benefit);
**7:** i = $N_x$;
**8:** if i = n then
**9:** return $M_x$;
**10:** end if
**11:** Enqueue left child into L;
**12:** if U($M_x$ U {$v_{i+1}$}) <= S then
**13:** Enqueue right child into L;
**14:** end if
**15:** until (L is empty)

**16:** return null;
**17:** end

The experiment results are carried out using the sales transactions of department stores and mail-order companies. The comparison of BPUS and A* Search algorithm on the basis of query cost and disk space constraint is given in Table 1:

Table 1: Comparison of BPUS and A* Search Algorithm.

| Disk Space Constraint | Query Cost | |
|---|---|---|
| | BPUS | A* Search |
| $0.5 \times 10^6$ | $4.8 \times 10^9$ | $4.7 \times 10^9$ |
| $1 \times 10^6$ | $4.6 \times 10^9$ | $4.6 \times 10^9$ |
| $1.5 \times 10^6$ | $4.5 \times 10^9$ | $4.4 \times 10^9$ |
| $2 \times 10^6$ | $4.4 \times 10^9$ | $4.4 \times 10^9$ |
| $2.5 \times 10^6$ | $4.4 \times 10^9$ | $4.2 \times 10^9$ |
| $3 \times 10^6$ | $4.4 \times 10^9$ | $4 \times 10^9$ |

## 3.2 Dynamic Improvement Algorithm

The high complexity and frequent vibration of static materialize view selection algorithm is overcome by using Efficient Materialized View Selection Dynamic Improvement Algorithm (EMVSDIA) in [11]. EMVSDIA is a two-step algorithm. In the first part, the input is Candidate View, Query Sample Space and Set of Materialized Views. If expectation, variance and constraint measure is not satisfied then view set must be adjusted in time. The candidate view from the view set is selected which has maximum benefit. During the second part, the views whose benefit sharply reduces should be substituted by those views which own large query probability.

## First Step Algorithm

**Input:** CV (Candidate View Set);
Sp (Useful Space);
$Q_{ser}(n)$ (Valid Sample Space);
$P_{Qser}(n)$ (Query Probability Set);
$E_n(V)$ and $D_n(V)$ (Expectation and Variance of View from P(n));
$E_{n-1}(V)$ and $D_{n-1}(V)$ (Expectation and Variance of View from P(n-1));
L(V) (Constraint Measure);
**Output:** $V_F$ (Candidate View Set);
**1: begin**
**2:** Initial $V_F$ = null;

**3:** if($E_n(V)$ == $E_{n-1}(V)$) && ($D_n(V)$ == $D_{n-1}(V)$) && (L(V)<lim) then
**4:** continue;
**5:** end if
**6:** else
**7:** $V = V_{Fact}$;
**8:** $Sp = Sp - |V_{Fact}|$;
**9:** $CV = CV - \{V_{Fact}\}$;
**10:** repeat
**11:** v = {v | max(Benefit (v,V))};
**12:** if(Sp>=|v|) then
**13:** $V_F = V_F \cup v$;
**14:** Sp = Sp - |v|;
**15:** end if
**16:** until (CV != null)
**17:** end else
**18:** return $V_F$;

## Second Step Algorithm

In second step the input which is the output from first step that is $V_F$ and Sp is selected. The final set of Materialized View will be generated as per the maximum benefit. The experiments are carried out based on a fact table. Dimension varies from 1 to 15. The comparison of EMVSDIA and BPUS is given in Table 2:

Table 2: Running time comparison

| Dimension | Running Time (s) | |
|---|---|---|
| | BPUS | EMVSDIA |
| 6 | 8 | 3 |
| 7 | 11 | 4 |
| 8 | 12 | 5 |
| 9 | 16 | 6 |
| 10 | 18 | 7 |
| 11 | 20 | 7 |

## 3.3 Shuffled Frog Leaping Algorithm

A novel shuffled frog leaping (SFL) algorithm technique for speeding up query answer in data warehouse environment is discussed in [12]. Materialized view selection problem comes under the NP-hard. The materialized view should be selected from large set of views such that it

minimizes view maintenance and query processing costs. The problem can be stated as: Given a set of queries Q and a storage space S, the view selection approach should select a set of views M to materialize, that minimize query response time and maintenance cost under the constraint that total space occupied by M is less than S.

Many materialized view selection algorithms have been proposed to deal with this problem. Such as, greedy heuristic algorithm [13], 0-1 integer programming [14] and genetic algorithm [15]. All these algorithms suffer from some problems. The greedy heuristic algorithm is highly problem dependent. The 0-1 integer algorithm might not be optimal for the best set of materialized views. In genetic algorithm it is hard to acquire good initial solutions. The conventional materialized view selection algorithms do not concerns about the storage constraint, ignore maintenance cost and computationally expensive. For efficiently dealing with this problem SFL is proposed in [12].

The SFL materialized view selection algorithm initially generates random population of P solutions (frogs). A set of frogs (view set M) is crated under the constraint that the total space occupied by M is less than S. Each frog composed of constant number of binary string, each AND-OR view graph is encode as binary string, where the constant number is the number of candidate views in the graph, the bit 0 denotes the corresponding node (view/query) is not materialized in the warehouse, the bit 1 denotes the corresponding candidate node is materialized.

In the second stage the SFL computes the fitness function. The materialized view selection algorithm aims to minimize the total query response time and the maintenance cost, it define the following fitness function F (G, M).

$$F(G,M) = \left( \frac{1}{1+T(G,M)} \right)$$

Where,

$$\arg\min\ T(G,M) = argmin \sum_{i,j=1}^{k,m} \left( f_{Q_i} Q(Q_i, M) + g_{v_j} U(V_i, M) \right)$$

With Constraints $\sum (S_v \leq S)$

Where,

$f_u$ - Frequency of the queries on u

$g_u$ - Frequency of the updates on u

$S_u$ – Space occupied by u

Q (u, M) – The cost of answering a query u by using the set of M materialized views.

U (u, M) – Maintenance cost for the view u in presence of the set of materialized views M.

In the next step the population P is sorted in descending order of their fitness. Divide P into m memeplexes, each containing n frogs (i.e. P = m X n). For each memeplex determine the best and worst frogs in terms of their fitness values. Then the frog with the global best fitness is identified. The SFL algorithm can find the global optimal solution within a reasonable amount of running time.

The experimental results are carried out using TPC-D benchmark datasets. The following Table 3 shows the comparison between heuristic algorithm, genetic algorithm and SFL algorithm on the basis of running time comparison.

Table 3: Query running time comparison.

| Query | HA | GA | SFL |
|---|---|---|---|
| 10 | 1.19 Hour | 16.72 Min | 1.34 Min |
| 20 | 5.26 Hour | 30.46 Min | 5.28 Min |
| 40 | 10.57 Hour | 51.85 Min | 13.61 Min |
| 60 | 20.38 Hour | 1.43 Hour | 21.52 Min |

# 4. Materialized View Maintenance

Materialized views are stored in data warehouse to enable users to quickly get search results for OLAP analysis. When the remote basic data source changes, the materialized views in data warehouse are also updated in order to maintain the consistency, this causes the need for handling the problem of materialized view maintenance.

## 4.1 Minimum Incremental Maintenance

There are two methods of materialized view maintenance. One is to re-compute the views, which leads to extra large storage and maintenance cost and sometimes it is unachievable due to storage space constraints. Second is incremental maintenance technique which is more preferable than re-computing the views. This technique is adopted in [16]. In [16] the incremental view maintenance techniques variation is given in terms of minimum incremental view maintenance. The principle of incremental view maintenance is that data source reports its changes to integrator who then calculates corresponding changes and inform the database with corresponding changes. Figure 1 shows the incremental maintenance over distributed data sources.

The incremental maintenance algorithm firstly calculates the incremental data of the materialized view through the incremental maintenance expression, then implements the modify operation. The incremental maintenance has many ways and strategies, and adopting different methods will lead to different workload, which affect the resources and efficiency of the system. A materialized view V is defined based on the basic relations between R1 and R2, and R1 and R2 changes have the impacts on the number of V, recorded as V<R1, R2>.

The calculation of V changes results from R1 changes and then calculates the changes of V using changes in R1 and R2 to implement modify operation. For the convenience of description, we give a definition: changes increment form R1 modified as: R1 Changes = R1 U Changes in R1; And then the calculation of V change is from R2 changes, through calculating the change of V using changes in R2 and R1 changes; Finally, modify operations can be implemented.

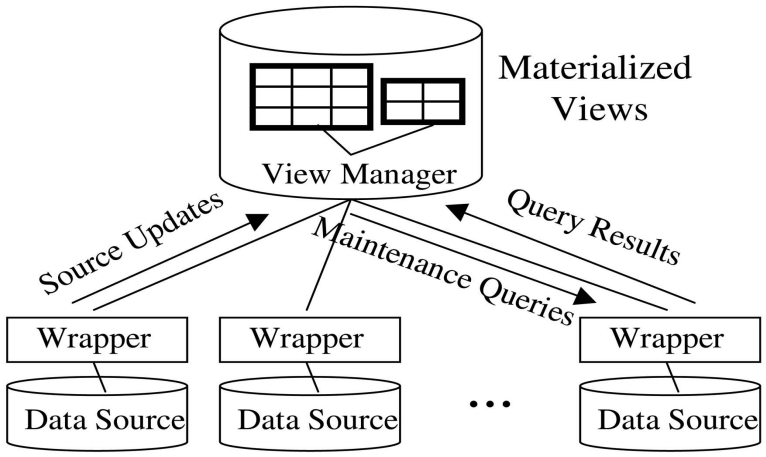


Fig 1: Incremental View Maintenance over Distributed Environment

The basic idea of minimum incremental maintenance is: for each view V based on relation {R1, R2, … Rn}, we obtain modification information from different sources, rank changing value by ascending, insert them into the message queue, remove modification information from the message queue, carry on incremental changes and modification operations, and then deal with the next. In this way, the minimum incremental calculation could be obtained by the ascending of relation changing value. The experimental results are carried out using the number of basic relation among the database. Table 3 represents the comparison between minimum incremental view maintenance and general incremental view maintenance.

Table 3: Incremental maintenance of the different views

| **Number of Basic Relations** | **Number of Incremental Maintenance** | |
|---|---|---|
| | **Minimum Incremental Maintenance** | **General Incremental Maintenance** |
| 3 | 110 | 170 |
| 5 | 190 | 500 |
| 7 | 450 | 850 |

## 4.2 A Compensation-Based Approach in Distributed Environments

The integrated data is usually stored as materialized views to allow better access, performance, and high availability. The source updates can be concurrent and cause erroneous results during view maintenance. State-of-the-art maintenance strategies apply compensating queries to correct such errors, making the restricting assumption that all source schemata remain

static over time. However, in such dynamic environments, the data sources may change not only their data but also their schema. Consequently, either the maintenance queries or the compensating queries may fail.

A novel framework called DyDa that overcomes these limitations and handles both source data updates and schema changes is proposed in [17, 18]. Three types of maintenance anomalies are identified, caused by source data updates, data-preserving schema changes, or non-data-preserving schema changes. A compensation algorithm to solve the first two types of anomalies is proposed. It has been shown that the third type of anomaly is caused by the violation of dependencies between maintenance processes. Put together, DyDa extends prior maintenance solutions to solve all types of view maintenance anomalies. The experimental results show that DyDa imposes a minimal overhead on data update processing while allowing for the extended functionality to handle concurrent schema changes. The intra compensation solution for first two anomalies is given in Intra Compensation algorithm.

First step is to determine which source updates are concurrent to the current maintenance query. Assumption is made that the network communication between any individual data source and the view manager is FIFO. This assumption guarantees that when we receive one maintenance query result from a particular source, all concurrent updates at that source must have already arrived at the view manager.

When the concurrent updates contain data updates, then the first anomaly occurs. When the concurrent updates contain RenameSCs (or data-preserving schema changes), then the second anomaly has occurred. Finally, when the concurrent updates contain DropSCs (or non-data-preserving schema changes), then the third anomaly has occurred.

Following algorithm depicts the pseudo code of this IntraCompensation algorithm. The QueryProcessing function accepts the original maintenance query Q and possibly its rewritten query Q' (due to data-preserving schema changes, see line 6) as input parameters. It returns the correct query result QR after compensation. It first sends the maintenance query Q' (equal to Q initially) to the source. If this query fails due to data-preserving schema changes, then we rewrite this query and process the new query Q' again (lines 6 and 7). If this query fails due to non-data-preserving schema changes (line 10), then we have to reschedule the maintenance processes (Section 5). Note that if there is no non-data preserving schema change, then the rewritten query will eventually succeed after taking all data-preserving schema changes into account. Then, we apply IntraComp algorithm in Algorithm to solve the first and second anomalies.

## Algorithm Query Processing and IntraComensation

```
1: Boolean QueryProcessing(Query Q, Query Q',
       QueryResults QR)
2: Boolean success = Issue(Q', QR); //Execute query Q', return
       TRUE if success, results in QR
3: ConcurrentUpdateSet
       CD=IdentifyConcurrentUpdates(QR); // By

4: if success=FALSE then
5:    if CD does not contain DropSC (non-data-preserving
          schema changes) then
6:       Q' = Rewrite(Q, CD); // Rewrite Q using all
             data-preserving schema changes in CD
7:       success = QueryProcessing(Q,Q',QR); // Recursive
         call for processing rewritten query Q'
8:       return success;
9:    else
10:      InterScheduler();// Section 5 handles
           non-data-preserving schema changes
11:      return FALSE; // We have to abort concurrent
           maintenance process
12:   end if
13: else
14:      success = IntraComp(Q,QR,CD); // Query succeeds.
           We compensate concurrent data updates
15:      return success;
16: end if
17: End Function
```

```
1: Boolean IntraComp(Query Q, QueryResult QR,
     ConcurrentUpdateSet CD)
2: ConcurrentDUSet CData;
3: ConcurrentSCSet CRen;
4: Boolean success = TRUE;
5: if CD does not contain RenameSC (or data-preserving
     schema change) then
6:    success = Compensation_Algorithm(Q, QR, CD);
     //No schema changes, apply compensation to all data, i.e.,
     CD
7: else
8:    while (CD not empty AND success) do
9:       SameSchema(CD, CData, CRen); // Extract data
           updates of same schema to CData and subsequent
           schema changes to CRen as in Example 2
10:      success = Compensation_Algorithm(Q, QR,
           CData); // Generate compensation query for each
           individual delta
11:      Q=Rewrite(Q, CRen); // Rewrite maintenance query
           Q based on schema changes
12:   end while
13: end if
14: return success;
15: End Function
```

IntraComp algorithm first detects if there is any concurrent data-preserving schema change. If not, it applies any existing compensating algorithm for data updates to solve first anomaly. If any concurrent data-preserving schema change is identified, then the algorithm generates a sequence of compensation queries for the data updates of the same schema.

The experimental results are carried out using Oracle 8i databases. In experimental setting, there are six tables evenly distributed over three different source servers with two tables each. Each table has four attributes and contains 100,000 tuples. The materialized join views are defined on these six tables. They contain all 24 attributes and reside on a fourth server. Fig. 2 depicts the total view maintenance cost measured in seconds (y-axis) under different numbers of source data updates (x-axis).

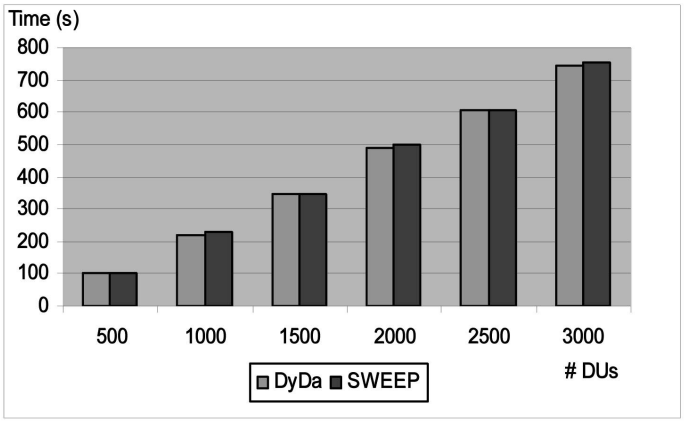


Fig 2: Sweep versus DyDa

## 5. Conclusion and Future Aspects

The selection of views to materialize is one of the most important issues in designing a data warehouse. So as to achieve the best combination of good query response where query processing cost should be minimized in a given storage space constraints. This paper gives the overall idea of the various materialized view selection and maintenance techniques. The space constraint is the most important factor while selecting the views to be materialized. In centralized as well as distributed environments incremental view maintenance techniques are found to be useful. There are still some issues that should be taken into account while handling the materialized views. These issues will be the future aspects in the query processing using materialized views. There should be some node selection strategy in the distributed environment for selecting and updating the materialized views.

The materialized view selection in distributed environment for query processing is also an open issue because it may happen that there may be some replicas of materialized view present over the network. Materialized views can be subdivided into number of parts so that any one of the part can be selected as per the query is also an open area for research. Therfore the node selection, materialized view selection and maintenance of materialized views in distributed environment and its implication for fast query processing and query optimization comes in the future aspects of this paper.

## References

[1] Gang Gou, Jeffery Xu Yu and Hongjun Lu, "A* Search: An Efficient and Flexible Approach to Materialized View Selection", *IEEE Trans. on Systems, Man and Cybernetics – Part C: Appl. And Reviews, Vol. 36, No. 3, May 2006.*

[2] Songting Chen, Xin Zhang and Elke A. Rundensteiner, "A Compensation-Based Approach for view Maintenance in Distributed Environments," *IEEE Trans on Knowledge and Data Engg., vol. 18, no. 8, Aug 2006.*

[3] Sayed Hamid Talebian and S. A. Kareem, "Using Genetic Algorithm to Select Materialized Views Subject to Dual Constraints," *IEEE Intl Conf. on Signal Processing Systems 2009.*

[4] Xian Sun and Ziqiang Wang, "An Efficient Materialized Views Selection Algorithm Based on PSO," *IEEE Intl. Conf. on Data Mining and Datawarehouse 2009.*

[5] Qingzhou Zhang, Xian Sun and Ziqiang Wang, "An Efficient MA-Based Materialized Views Selection Algorithm," *IEEE Intl. Conf on Control, Automation and Systems Engineering 2009.*

[6] 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.

[7] 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.

[8] H. Gupta. "Selection of Views to Materialize in a Data Warehouse". Proceedings of International Conference on Database Theory, Athens, Greece 1997.

[9] Ziqiang Wang and Dexian Zhang, Optimal Genetic View Selection Algorithm Under Space Constraint, International Journal of Information Technology, vol. 11, no. 5, pp. 44 - 51,2005.

[10] K. Aouiche, P. Jouve, and J. Darmont. Clustering-based materialized view selection in data warehouses. In ADBIS'06, volume 4152 of LNCS, pages 81–95, 2006.

[11] Zhou Lijuan, Ge Xuebin, Wang Linshuang, Shi Qian, "Efficient Materialized View Selection Dynamic Improvement Algorithm", *Sixth IEEE International Conference on Fuzzy Systems and Knowledge Discovery, 2009.*

[12] Xin Li, Xu Qian, Junlin Jiang and Ziqiang Wang, "Shuffeled Frog Leaping Algorithm for Materialized Views Selection", *Second IEEE International Workshop on Education Technology and Computer Science 2010.*

[13] C. H. Choi, J. X. Yu and G. Gou, "What difference heuristic make: maintenance cost view selection revisited," *Proceedings of the third Intl. Conf. on Advances in Web-Age Information Management, Springer-Verlag.pp.313-350, Jan 2002.*

[14] J. Yang, K. Karlapalem and Q. Li, "Algorithms for materialized view design in data warehousing environment," *Proceedings of Twenty Third Intl. Conf. on Very Large Data Bases, pp.136-145, Aug 1997.*

[15] W. Y. Lin and I. C. Kuo, "A Genetic Selection algorithm for OLAP data cubes," *Knowledge and Information Systems, vol.6, pp.83-102, Feb 2004*

[16] Lijuan Zhou, Qian Shi and Haijun Geng, "The Minimum Incremental Maintenance of Materialized Views in Data Warehouse," *Proceedings of Second IEEE International Asia Conference on Informatics in Control, Automation and Robotics 2010.*

[17] Bin Liu and Elke A. Rundensteiner, "Optimizing cyclic Join View Maintenance over Distributed Data Sources," *IEEE Trans on Knowledge and Data Engg, vol. 18, no.3, Mar 2006.*

[18] A. N. M. Bazlur Rashid and M. S. Islam, "An Incremental View Materialization Approach in ORDBMS," *IEEE Intl. Conf. on Recent Trends in Information, Telecommunication and Computing 2009.*

[19] Zhou Lijuan, Ge Xuebin, Wang Linshuang and Shi Qian, "Research on Materialized View Selection Algorithm in Data Warehouse," *IEEE Intl. Conf. on Computer Science-Technology and Applications 2009.*

## Author Information

### Short Biography

**1. Mr. Pravin P.Karde was born in** Amravati, Maharstra in 1975. He received the Post Graduate Degree (M.E.) in Computer Sceince & Engineering from S.G.B. Amravati University, Amravati in the year 2006 & pursuing the Ph.D degree in Computer Science & Engineering. Currently he is working as an Assistant Professor & Head in Information Technology Department at H.V.P.M's College of Engineering & Technology, Amravati,India. His interest is in Selection & Maintenance of Materialized View.

**2. Dr. V.M.Thakare** was born in Wani, Maharashtra in 1962. He was worked as Assistant Professor for 10 Years at Professor Ram Meghe Institute of Technology & Research, Badnera and P.G.Department of Computer Science, S.G.B. Amravati University, Amravati. Currently he is working as Professor & Head in Computer Science from last 9 years, Faculty of Engineering & Technology, Post Graduate Department of Computer Science, SGB Amravati University, Amravati.. He has published 86 papers in various National & International Conferences & 20 papers in various International journals. He is working on various bodies of Universities as a chairman & members. He has guided around 300 more students at M.E / MTech, MCA M.S & Mphil level. He is a research guide for Ph.D. at S.G.B. Amravati University, Amravati. His interests of research is in Computer Architecture, Artificial Intelligence, Robotics, Database and Data warehousing & mining.