A Comprehensive Study of View Maintenance Approaches in Data Warehousing Evolution

Hemant Jain
Guru Gobind Singh Indraprastha University
Delhi, India
919873517612
hemantjain.ipu@gmail.com

Anjana Gosain
Guru Gobind Singh Indraprastha University
Delhi, India
919811055716
anajna_gosain@yahoo.com

ABSTRACT
A data warehouse mainly stores integrated information over data from many different remote data sources for query and analysis. The integrated information at the data warehouse is stored in the form of materialized views. Using these materialized views, user queries may be answered quickly and efficiently as the information may be directly available. These materialized views must be maintained in answer to actual relation updates in the different remote sources. One of the issues related to materialized views is that whether they should be recomputed or they should be adapted incrementally after every change in the base relations. View maintenance is the process of updating a materialized view in response to changes to the underlying data called view maintenance. There are several algorithms developed by different authors to ease the problem of view maintenance for data warehouse systems. In this paper, we have provided a comprehensive study on research works of different authors related to DW view maintenance considering various parameters and presented the same in tabular way.

Categories and Subject Descriptors
H.2.7 [Database Management]: Database Administration- data warehouse and maintenance, updating, Source, query processing.

General Terms
Management, Theory, Documentation.

Keywords
Data warehouse, view maintenance, materialized views, data warehouse evolution.

1. INTRODUCTION
The A Data warehouse is a data repository used to provide reporting and analysis of large amounts of historical data. In a data warehouse, the query expressions that define views and real relations may be stored at different database sources residing at different sites. Once data sources modify, the data warehouse, mainly the materialized views in the data warehouse, should be updated also. This is the problem of view maintenance in data warehouses [15]. The sources may inform the data warehouse when an update occurs at source; it sends the notification to warehouse after that warehouse sends the query to source for the corresponding update as source receives the query it sends the reply to warehouse to that corresponding query [24]. This basic algorithm is neither convergent nor weakly consistent in a warehousing environment [28].

In literature, different approaches have been proposed to deal with Data Warehouse view maintenance namely basic view maintenance [8, 11, 12, 14, 20, 21, 22], incremental view maintenance [3, 6, 7, 9, 13, 15, 16, 17, 18, 23, 24, 25] and self maintainable maintenance [1, 2, 4, 5, 10, 19].

2. STATE OF THE ART
In Basic view maintenance approach, source and the data warehouse communicate with each other, when update occurs at source; it sends the notification to warehouse after that warehouse sends the query to source for the corresponding update as source receives the query it sends the reply to warehouse to that corresponding query [24]. This basic algorithm is neither convergent nor weakly consistent in a warehousing environment [28].

2.1 Basic View Maintenance
In Basic view maintenance approach, source and the data warehouse communicate with each other, when update occurs at source; it sends the notification to warehouse after that warehouse sends the query to source for the corresponding update as source receives the query it sends the reply to warehouse to that corresponding query [24]. This basic algorithm is neither convergent nor weakly consistent in a warehousing environment [28].

In [11] authors have presented a comparison of three materialized join view maintenance methods in a parallel RDBMS, which they refer to performance at the cost of using more space. The results of this study show that the method of choice depends on the environment, in particular, the update activity on base relations and the amount of available storage space as the naive, auxiliary relation, and global index methods.

In [12] authors extend the PVM-MED WRAP algorithm to achieve the Complete level of consistency, and also presented a scalable architecture for the proposed algorithm. In this Simulation shows that extending the PVM-MED WRAP algorithm to achieve the complete level of consistency limits the maximal parallelism feature.

In the context of data warehouse materialized view maintenance and associated issues along with the devised techniques to deal with them.

In [14] authors have proposed a method to minimize the unnecessary updates for materialized views without increasing of response time. This method, which is named VMOST (View Maintenance based On State Transferring), introduced four states for materialized web views. In this when receiving query/update requests, web views transfer between the four states in accordance with their accessing and changing history. According to the experimental data, VMOST is adaptive to the fast changing web environment and has a good overall performance.

A model of P2P-based collaborative view maintenance is proposed in this paper [20], which can not only exploit the potential merits of P2P and CSCW, but also overcome the problems that are overload and even
crash at the end of data warehouse. In this authors also proposed a collaborative maintenance.

In [21] authors have discussed capabilities of PIVOT and UNPIVOT operators; materialized view maintenance, view maintenance work with PIVOT and UNPIVOT operators and finally they have focused on the research trends in view maintenance.

In [22] authors have presented an efficient technique aimed at updating data and performing view maintenance for real-time data warehouses while still enforcing these two timing requirements for the OLAP transactions. Authors proposed approach aims at addressing the issues of applying updates and performing view maintenance while still effectively serving user OLAP queries in real-time data warehouse environment.

2.2 Incremental View Maintenance

In Incremental view maintenance approach, only changes in the materialized views of the data warehouse are computed rather than recomputing every view from scratch. ECA is an incremental view maintenance algorithm based on the centralized view maintenance algorithm. It is also the fastest algorithm that will let the data warehouse remain in a consistent state [28].

In [3] authors have described the architecture of the Whips prototype system, which collects, transforms, and integrates data for the warehouse. They have showed how the required functionality can be divided among cooperating distributed CORBA objects, providing both scalability and the flexibility needed for supporting different application needs and heterogeneous sources.

In [6] authors have proposed a new incremental approach to maintaining materialized views both in the data warehouse and in the data marts. This approach uses auxiliary relations derived from the materialized view and the base relations by normalizing the view according to the functional dependencies valid in the view. The motivation for using normalization in this approach is to increase the likelihood of sharing the auxiliary relations across several data marts, as has been shown by the use of normalization in relational database design.

In [7] authors have proposed a new compensation algorithm that is used in removing the anomalies, caused by interfering updates at the base relations, of incremental computation for updating the view. This algorithm does not assume that messages from a data source will reach the view maintenance machinery in the same order as they are generated, and they are also able to detect update notification messages that are lost in their transit to the view, which would otherwise cause the view to be updated incorrectly. Proposed algorithm also does not require that the system be quiescent before the view can be refreshed.

In [9] authors have proposed a maintenance algorithm that does not need the compensation step and applies to general view expressions of the bag algebra, without limit on the number of base relations per data source.

In [13] authors have proposed an incremental maintenance method for temporal views that allows improvements over the re-computation from scratch. They introduce formalism for temporal data warehouse specification that summarizes information needed for its incremental maintenance.

In [15] authors have presented an incremental view maintenance approach based on schema transformation pathways. This approach is not limited to one specific data model or query language, and would be useful in any data transformation or integration framework based on sequences of primitive schema transformations.

In [17] authors have tackled the problem of finding the most efficient batch incremental maintenance strategy under a refresh response time constraint; that is, at any point in time, the system, upon request, must be able to bring the view up to date within a specified amount of time. In this authors have also presented a series of analytical results leading to the development of practical algorithms.

In [18] authors have developed the change-table technique for incrementally maintaining general view expressions involving relational and aggregate operators. They show that the change-table technique outperforms the previously proposed techniques by orders of magnitude. The developed framework easily extends to efficiently maintaining view expressions containing outer join operators. They have proved that the developed change-table technique is an optimal incremental maintenance scheme for a given view expression tree under some reasonable assumptions.

Incremental maintenance technique is accepted in this paper [23]. In this idea and strategy of minimum incremental maintenance is presented. The materialized view definitions and maintenance expressions, as well as algorithms are given. The experiment shows that the maintenance cost of materialized views is decreased and data warehouse processing efficiency is improved.

In [24] authors have proposed an algorithm for incremental view maintenance with the inclusion of some existing approaches. They utilized the concept of version store for older versions of tables that have been updated at the source and they are also able to detect the update notification messages that are lost during updating the view.

In [25] authors have proposed an incremental view maintenance approach based on data source compensation called DSCM, which effectively overcomes the shortcomings of previous compensation solutions through minimizing the extent of compensation to the base relations of data sources and using the precise source states to evaluate the maintenance queries.

2.3 Self maintainable Maintenance

View self-maintenance is the process of incrementally refreshing a materialized view using the view instance and the update to some base relation, but without examining any of the base relations, or using only a specified subset of the base relations. When a view together with a set of auxiliary views can be maintained without accessing base data, we say the views are self-maintainable [2] [5].

In [1] author has reported on some interesting new results for conjunctive-query views under insertion updates: 1) the CTSM's are extremely simple queries that look for certain tuples in the view to be maintained; 2) these CTSM's can be generated at view definition time using a very simple algorithm based on the concept of Minimal Z-Partition; 3) view self-maintenance can also be expressed as simple update queries over the view itself.

In [2] authors have showed that by using key and referential integrity constraints, they often can maintain a select-project-join view without going to the data sources or replicating the base relations in their entirety in the warehouse. They derive a set of auxiliary views such that the warehouse view and the auxiliary views together are self-maintainable-they can be maintained without going to the data sources or replicating all base data.

In [4] authors have proposed an incremental technique for efficiently maintaining materialized views in these high performance applications by materializing additional relations which are derived from the intermediate results of the view computation. They presented an algorithm which uses the operator tree for the view to determine which additional relations need to be materialized in order to maintain the view. They also give an incremental algorithm for maintaining both the view and the additional relations which has several desirable features.

In [5] author has focused on the problem of determining view in the presence of functional dependencies. Here author, shows (i) SM of a conjunctive-query (CQ) view can be reduced to a problem of query containment, whose solution can be expressed as a (Boolean) query on
the view in safe, non recursive Data log. (ii) Special class of CQ views with no repeated predicates; two useful concepts can be defined: the well-founded derivation DAG and sub goal partitioning. (iii) Derive three simple conditions that each guarantee view SM under general functional dependencies.

In [10] authors gave a preliminary result on self-maintainability of deletions of views over XML data. We give a necessary and sufficient condition of self-maintainability of deletions, and an algorithm to implement self-maintenance of deletions of views for XML data.

This paper [19] provided an online view self-maintenance method based on source view’s increment to keep the materialized view consistent with the data source. In this, authors have used the cooperation between the integrator at warehouse and the monitor at data source to maintain the view, and the method does not require to query back to data source, it can accelerate view maintenance and lessen the burden of communication between data warehouse and data sources.

3. COMPARATIVE STUDY

We have analyzed the various research works on several parameters and presented their comparison below in the table 1. In the table the symbol ‘×’ indicates that the corresponding parameter does not exist and symbol ‘√’ indicates its existence in the related paper.

<table>
<thead>
<tr>
<th>Authors</th>
<th>Technique/Category</th>
<th>Issues addressed</th>
<th>Changes handled</th>
<th>Proposed Work</th>
<th>Type of Queries/Operations</th>
<th>Advantages</th>
<th>Disadvantages</th>
<th>Tool support / Implementation</th>
<th>Meta data supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nam Huyn (1996)</td>
<td>Self Maintainable Maintenance</td>
<td>Maintaining materialized view without accessing base relations</td>
<td>Separating SM test generation from SM test evaluation</td>
<td>Determine SM of views that are expressed as conjunctive queries</td>
<td>Not Mentioned</td>
<td>Speed up the update time testing &amp; Maintenance works + Improve ment in time &amp; space</td>
<td>Increases complexity + For large view this approach is not practical</td>
<td>Not Addressed</td>
<td>×</td>
</tr>
<tr>
<td>Quass, Gupta, Munic (1996)</td>
<td>Self Maintainable Maintenance</td>
<td>Auxiliary Views + Materialized View Maintenance</td>
<td>Insertion of multiple relations at once + Protected updates</td>
<td>Present an algorithm for determining auxiliary views</td>
<td>Select-Project- Join view + Semi Join operators + SQL Expressions</td>
<td>Auxiliary views can be maintained without going to the data sources + Reduce the sizes of auxiliary view</td>
<td>N.A.</td>
<td>Not Addressed</td>
<td>×</td>
</tr>
<tr>
<td>Wiener, Gupta, Zhuge (1996)</td>
<td>Incremental View Maintenance</td>
<td>Maintain one or more materialized view over the source data</td>
<td>Not Mentioned</td>
<td>Whips architecture for warehouse creation and maintenance</td>
<td>Select-Project- Join view + Aggregate operators</td>
<td>Allow concurrent queries + Consistently updating</td>
<td>Needed multiple query processor</td>
<td>C, C++ language</td>
<td>√</td>
</tr>
<tr>
<td>M.W. Vincent &amp; M. Mohania (1997)</td>
<td>Self Maintainable Maintenance</td>
<td>Materialized View + Multiple updates</td>
<td>Update the view using expression tree + Update are propagated in a bottom up fashion</td>
<td>Present an algorithm which use the operator tree for the view in order to maintain the view</td>
<td>Selection (σ) + Projection (Π) + Union (U) +</td>
<td>Allows multiple updates + View and relation are self maintainable</td>
<td>N.A.</td>
<td>Not Addressed</td>
<td>×</td>
</tr>
</tbody>
</table>

Table 1. Comparison of work by different authors in tabular manner
<table>
<thead>
<tr>
<th>Name</th>
<th>View Maintenance</th>
<th>Lambda</th>
<th>Method</th>
<th>SQL queries</th>
<th>Further optimization</th>
<th>Not Addressed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nam Huyn (1997) [5]</td>
<td>Self Maintainable Maintenance</td>
<td>Not Mentioned</td>
<td>Method based on testing query containment + Direct method based on the concept of well-founded deviation DAG</td>
<td>SQL queries</td>
<td>Further optimization is still a problem</td>
<td>Not Addressed</td>
</tr>
<tr>
<td>T.W. Ling &amp; E.K. Sze (1999) [7]</td>
<td>Incremental View Maintenance</td>
<td>Use of version numbers</td>
<td>Developed a new compensatio algorithm</td>
<td>Not Mentioned</td>
<td>Message from the data source is out of order as they are generated</td>
<td>Not Mentioned</td>
</tr>
<tr>
<td>Gupta &amp; Mumicking (1999) [8]</td>
<td>Basic View Maintenance</td>
<td>Materialized views + Application + Problems</td>
<td>Classificatio n of View Maintenanc e problem along dimensions</td>
<td>VM Techniques + New applications</td>
<td>Select Project Join View</td>
<td>Information Identificatio n is difficult</td>
</tr>
<tr>
<td>G. Moro &amp; C. Sarstoi (2001) [9]</td>
<td>Incremental View Maintenance</td>
<td>Compensatio n approaches</td>
<td>Changes propagation rules for DWH and Centralized database</td>
<td>Proposed an maintenance algorithm that does not need the compensatio n step</td>
<td>Select-Project-Join relational Views</td>
<td>Avoid the problems of compensatio n based algorithms + Strong consistency</td>
</tr>
<tr>
<td>Author(s)</td>
<td>Method</td>
<td>Algorithm</td>
<td>Benefits</td>
<td>Considers</td>
<td>Used</td>
<td>Notes</td>
</tr>
<tr>
<td>-----------</td>
<td>--------</td>
<td>-----------</td>
<td>----------</td>
<td>-----------</td>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td>Hua, Gao, Chen (2003) [10]</td>
<td>Self Maintainable Maintenance</td>
<td>Maintain Materialized Views + XML Data</td>
<td>Used XML query language in the place of relational query Language</td>
<td>Algorithm to implement self maintaining deletions of views over XML Data</td>
<td>XML Views</td>
<td>Consider only simple updates of database + XML query language are much more complex</td>
</tr>
<tr>
<td>Y.Zhang &amp; D. Yang (2004) [14]</td>
<td>Basic View Maintenance</td>
<td>Materialized Views updating</td>
<td>Introduce mew states like Semi-Active, Semi-Sleeping</td>
<td>Introduce VMOST method to minimize the unnecessary updates view</td>
<td>Not Mentioned</td>
<td>Large overhead + Communication cost is more</td>
</tr>
<tr>
<td>Hao Fan (2005) [15]</td>
<td>Incremental View Maintenance</td>
<td>Approach based on schema transformatio n Pathway</td>
<td>Maintaining Data Warehouse Data using AutoMed</td>
<td>Perform IVM along such evolvable pathways</td>
<td>Select Project Join Expression</td>
<td>Not Addressed</td>
</tr>
</tbody>
</table>

DOI: 10.1145/2347696.2347705  
http://doi.acm.org/10.1145/2347696.2347705
<table>
<thead>
<tr>
<th>Authors</th>
<th>Type of View Maintenance</th>
<th>Concurrent Updates</th>
<th>Introduce Commit + Temporal Detection + TimeStamp</th>
<th>Parallel Strategy + Temporal Subtraction Operator + Intersection Operator</th>
<th>Detection of relevant updates + Detection of concurrent updates + Better performance in response time</th>
<th>Due to additional Function complexity of the architecture is increase</th>
<th>Not Addressed</th>
<th>√</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y. Zuo, Y. Tang &amp; Z. Shu (2005) [16]</td>
<td>Incremental + Self Maintainable Maintenance</td>
<td>Concurrent Updates + Multiple dimensional view</td>
<td>Introduce Commit agent + Temporal Detector + TimeStamp Assign</td>
<td>Parallel Strategy for multiple dimensional view evaluation approach PMDVRES</td>
<td>Detection of relevant updates + Detection of concurrent updates + Better performance in response time</td>
<td>Due to additional Function complexity of the architecture is increase</td>
<td>Not Addressed</td>
<td>√</td>
</tr>
<tr>
<td>Hao He, L. Xie, Yang (2005) [17]</td>
<td>Incremental View Maintenance</td>
<td>Not Mentioned</td>
<td>Asymmetric approach for more effective VM</td>
<td>SQL Statements</td>
<td>Minimize the cost of view maintenance</td>
<td>N.A.</td>
<td>Commerci al DBMS + Linux Server</td>
<td></td>
</tr>
<tr>
<td>Gupta &amp; Mumick (2006) [18]</td>
<td>Incremental View Maintenance</td>
<td>Aggregate &amp; outer join operators</td>
<td>Update into view by refresh expression</td>
<td>Projection + Group by operators</td>
<td>Self Maintainable + Minimize the No. of queries</td>
<td>Cost Model for maintenance is not suitable</td>
<td>Not Addressed</td>
<td>×</td>
</tr>
<tr>
<td>Z. Shu, Y. Zuo &amp; Y. Tang (2008) [20]</td>
<td>Basic View Maintenance</td>
<td>Overload unbalance between DWH + Parallel View Maintenance</td>
<td>Introducing P2P and CSCW idea in WHIPS prototype</td>
<td>No. of messages in the PCVM is less compare to WHIPS + PCVM can parallelize the concurrent updates independently</td>
<td>Complex architecture compared to WHIPS architecture</td>
<td>Not Addressed</td>
<td>√</td>
<td></td>
</tr>
</tbody>
</table>
## 4. CONCLUSION AND FUTURE WORK

Data warehouses are prepared by collecting information from a number of sources and integrating it into one repository modified according to user requirement. Hence, solutions for data warehouse maintenance are important that can handle such sources updates. Recently, view maintenance approaches proposed by different authors have began to deal with problems like concurrent updates of different autonomous sources, materialized views consistency, functional dependencies, multiple materialized views etc. In this paper we have presented an analysis of different approaches being proposed by various authors to deal with the view maintenance in data warehouse. We have examined these approaches on various parameters and provided a comparative study in a tabular manner. In the future work, we propose a framework of data warehouse materialized view maintenance to overcome the above problems observed by various authors.

<table>
<thead>
<tr>
<th>Authors</th>
<th>Type of View Maintenance</th>
<th>Features</th>
<th>Challenges</th>
<th>Advantages</th>
<th>Technologies Used</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hoang Vu &amp; Gopalkrishnan (2009) [22]</td>
<td>Basic View Maintenance</td>
<td>Updates + VM + OLAP Transaction associated with deadlines</td>
<td>Technique (ORAD) aimed at updating data + Performing VM</td>
<td>Reducing memory overhead + Reducing no. of on-demand requests</td>
<td>N.A.</td>
<td>Used Simulator</td>
</tr>
<tr>
<td>Zhou, Shi &amp; Geng (2010) [23]</td>
<td>Incremental View Maintenance</td>
<td>Materialized Views</td>
<td>Changes in basic relations</td>
<td>Minimum Incremental Technique</td>
<td>Reduce maintenance cost + Improve DWH performance</td>
<td>Not Emphasized on computation complexity</td>
</tr>
<tr>
<td>Almazyad &amp; Siddiqui (2010) [24]</td>
<td>Incremental View Maintenance</td>
<td>Incremental computation + Existing Approaches</td>
<td>Version store + TXN No. + Statement No.</td>
<td>Algorithm with concept of Version store</td>
<td>No need to update from scratch + Free from locks</td>
<td>Message transfer in worst case is not desirable</td>
</tr>
<tr>
<td>Zhang, Yang &amp; Wang (2010) [25]</td>
<td>Incremental View Maintenance</td>
<td>Concurrent Updates + Round trip + Overhead</td>
<td>VM Approach based on data source compensations</td>
<td>Not Mentioned</td>
<td>Reduce the roundtrip of maintenance query + Reduce the overhead of VM</td>
<td>Handle only single VM + Compensations on cost is more</td>
</tr>
</tbody>
</table>

DOI: 10.1145/2347696.2347705
REFERENCES


