A Comprehensive Analysis of Materialized Views in a Data Warehouse Environment

Garima Thakur  
M.Tech (IT), Department of IT  
USIT, Guru Gobind Singh Indraprastha University  
Delhi, India  
thakur_garima_27@yahoo.co.in

Anjana Gosain  
Associate Professor, Department of IT  
USIT, Guru Gobind Singh Indraprastha University  
Delhi, India  
anjana_gosain@hotmail.com

Abstract—Data in a warehouse can be perceived as a collection of materialized views that are generated as per the user requirements specified in the queries being generated against the information contained in the warehouse. User requirements and constraints frequently change over time, which may evolve data and view definitions stored in a data warehouse dynamically. The current requirements are modified and some novel and innovative requirements are added in order to deal with the latest business scenarios. In fact, data preserved in a warehouse along with these materialized views must also be updated and maintained so that they can deal with the changes in data sources as well as the requirements stated by the users. Selection and maintenance of these views is one of the vital tasks in a data warehousing environment in order to provide optimal efficiency by reducing the query response time, query processing and maintenance costs as well. Another major issue related to materialized views is that whether these views should be recomputed for every change in the definition or base relations, or they should be adapted incrementally from existing views. In this paper, we have examined several ways of performing changes in materialized views their selection and maintenance in data warehousing environments. We have also provided a comprehensive study on research works of different authors on various parameters and presented the same in a tabular manner.

Keywords—Materialized views; view maintenance; view selection; view adaptation; view synchronization.

I. INTRODUCTION

Data warehouse is referred as a subject-oriented, non-volatile & time variant centralized repository that preserves quality data [1]. A data warehouse extracts and integrates information from diverse operational systems prevailing in an organization under a unified schema and structure in order to facilitate reporting and trend analysis. Information sources which are integrated in the data warehouse are dynamic in nature i.e. they may transform or evolve in terms of their instances and schemas. Moreover, requirements specified by the various stakeholders and developers frequently change owing to numerous reasons as mentioned below [17][18][19]:

1. Ambiguous or insufficient requirements during the developmental phase [17].
2. Change in the requirements during the operational phase of the Data Warehouse which results in the structural evolution of the data warehouse [18].
3. Reorganization of the data warehouse schema during the operational phase of the data warehouse as a result of different design solutions that are decided upon [18].
4. New user or business requirements arise or new versions need to be created [18][19].
5. Periodical revisions are made in order to eliminate the errors & redundancies [17][18].
6. The data warehouse must be adapted to any changes which occur in the underlying data sources [18][19].

Hence, data warehouse and views present in warehouse must evolve whenever there is any modification or update in the requirements or base relations, in order to fulfill the needs and constraints allocated by the various users who need the assistance of data warehouse system. In fact, data warehouse evolution process never ceases. Appropriate techniques should be devised to handle the above mentioned changes in the data sources as well as view definitions to keep the warehouse in its most consistent state.

Whenever any user poses a query, the query is processed directly at this repository thereby, eliminating the need to access the actual source of information. The resulting datasets that are generated in the response to the queries raised by the users are called as views, which represent functions derived from the base relations to support viewing of snapshots of stored data by the users according to their requirements. These derived functions are recomputed every time the view is called upon. Re-computing and selection of views becomes impossible for each and every query especially; when the data warehouse is very large or the view is quite complex or query execution rate is high. Thus, we accumulate some pre-calculated results (or views) in our central repository (i.e. data warehouse) in order to provide faster access to data and enhance the query performance. This technique is referred as materialization of views.

Materialized views act as a data cache that gather information from distributed databases and support faster and reliable availability of already computed intermediate result sets (i.e. responses to queries). Data sources in current scenario are becoming quite vast and dynamic in nature i.e. they change rapidly. Consequently, frequency of deletion, addition and update operations on the base relations rises unexpectedly. Whenever the underlying base relation is modified the
corresponding materialized view also evolves in reaction to those changes so that it can present quality data at the view level. Hence, we need certain techniques to deal with the problem of keeping a materialized view up-to date in order to propagate the changes from remote data source to the destined materialized view in the warehouse. These techniques can be broadly classified as view selection, view maintenance, view synchronization and lastly, view adaptation. Each one of them is explained in more detail in the next section.

The layout of the paper is as follows. In section 2, we address the above mentioned techniques and also give a brief on the literatures being reviewed for the same. Section 3, presents a comparative study of the various research works explored in the previous section. Lastly, we conclude in section 4.

II. STATE OF THE ART

In this section, we describe the various techniques designed to handle the evolution of a materialized view in response to the modifications in data sources it originated from. In addition, we also discuss the literatures being reviewed in context of each and every technique.

The tasks involved in evolution of materialized views in a data warehouse can be categorized as follows:

![Figure 1. Tasks in materialized view evolution](image)

A. View Adaptation & Synchronization

One of the factors that contribute to the changes in a materialized view is rewriting of views that leads to changes in the original view definition itself. This problem is addressed as view adaptation. Re-writing of view definitions generates the need to adapt the view schema to match it up with the most current view definition being referenced [2, 3]. View adaptation can be done either in incremental fashion or by performing full re-computation of the views [3]. If re-computation results in equivalent views then, there is no need to implement adaptation techniques because data is preserved. Non-equivalent definitions create new schema for the same view resulting in evolution of the original view. Some of the examples are listed below:

<table>
<thead>
<tr>
<th>Schema changes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rename</td>
<td>Data preserving, no adaptation required.</td>
</tr>
<tr>
<td>Drop/Delete</td>
<td>Data deleted, hence non-equivalent views might be generated</td>
</tr>
</tbody>
</table>

In [2] the authors have provided a comprehensive study on various adaptation techniques. They have also provided re-definitions of all SQL clauses and views when local changes are made to view definitions. But they have only handled single materialized view changes.

In [13] author has discussed various view adaptation techniques where only the changes in view definitions cause adaptation in the views. Relation algebra binary operators can be added to SQL clauses to adapt the views. Expression trees are used to evaluate view definitions.

Another technique employed to handle materialized views is view synchronization. This technique changes the view definition when the structure of its base relations changes. It addresses both equivalent & non-equivalent view re-definitions [3]. Some of the changes that result in creation of new schema definitions are as follow [3]:

<table>
<thead>
<tr>
<th>Schema changes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rename</td>
<td>Renames the attributes and tables in the original view</td>
</tr>
<tr>
<td>Drop/Delete</td>
<td>Deleted attributes or tuples or tables in original views</td>
</tr>
</tbody>
</table>

EVE (Evolvable View Environment), a general framework has been developed in [4] to handle view synchronization in large distributed dynamic environments like WWW. A view definition language, E-SQL, has also been designed along with some replacement strategies to propagate the changes in affected view components.

B. View Selection

The most important issue while designing a data warehouse is to identify and store the most appropriate set of materialized views in the warehouse so that they optimise two costs included in materialization of views: the query processing cost and materialized view maintenance cost.

Materialization of all possible views is not recommended due to memory space and time constraints [6]. The prime aim of view selection problem is to minimize either one of the constraints or a cost function as shown below:

![Figure 2. View Selection Process](image)
Hence, view selection problem is formally defined as a process of identifying and selecting a group of materialized views that are most closely-associated to user-defined requirements in the form of queries in order to minimize the query response time, maintenance cost and query processing time under certain resource constraints [6].

In [5] authors have developed a AND/OR graph based approach to handle view selection problem in data cubes present in the data warehouse by taking an example of TPC-D benchmark database. They have also proposed an optimization algorithm to select certain views, but, this algorithm does not perform well in some of the cases.

Another graph based approach has been discussed in [6] in order to select a set of views for special cases under disk-space and maintenance cost constraints. AND view graphs have been discussed to evaluate the global plan for queries and OR view graphs focus on data cubes. They have proposed greedy heuristics based algorithms to handle the same. But still the approach has certain limitations like very little insight into the approximation of view-selection problem in AND/OR view graphs. Problem in AND view graphs is still not known to be NP-hard.

In [7] the authors have proposed two algorithms, one for view selection and maintenance and the second one for node selection for fast view selection in distributed environments. They have considered various parameters: query cost, maintenance cost, net benefit & storage space.

In [8] presented a framework for automatically selecting materialized views and indexes for SQL databases that has been implemented as a part of performance tuning in SQL Server 2000.

In [9] authors have presented a framework for selection of views to improve query performance under storage space constraints. It considers all the cost metrics in order to provide the optimal set of views to be stored in the warehouse. They have also proposed certain algorithms for selecting views based on their assigned weightage in the storage space and query.

In [14] a clustering based algorithm ASVMRT, based on clustering. Reduced tables are computed using clustering techniques and then materialized views are computed based on these reduced tables rather than original relations.

C. View Maintenance

Re-computation of materialized views is quite a wasteful task in data warehousing environments. Instead, we can only update a part of the views which are affected by the changes in the base relations. Hence, View maintenance incrementally updates a view by evaluating the changes to be incorporated in the view so that it can evolve over time. If views are maintained efficiently then, the overhead incurred while performing expensive joins and aggregations is eliminated to a larger extent.

In [10] authors have proposed a framework for dynamic environments called DyDa, for view maintenance in order to handle both concurrent schema and data changes. They have identified three types of anomalies and also proposed some dependency detection and correction algorithms to resolve any violation of inter-dependencies occurring between the maintenance processes.

An algorithmic approach has been implemented in [11] for incremental materialized view maintenance. The authors have employed the concept of version store so that the older versions of relations can be preserved and retrieval of correct data in the desired state is available round the clock. They have further proposed architecture to support of DW augmented with a View Manager.

In [12] authors have designed algebra based algorithm for incremental maintenance of views by schema restructuring. They have proposed a SchemaSQL language to handle data updates and schema changes. Moreover, transformation operators have also been proposed to propagate data and schema changes easily.

View maintenance problem has been dealt in [3] by means of a compensation algorithm that eliminates interfering update anomalies encountered during incremental computations. Version numbers have been assigned to the updates occurring on the base relations to arrange them in a proper order. These numbers also help in detecting update notification messages that might be lost in the whole process of propagating the changes from source relation to views.

In [15] authors have presented PNUTS to handle asynchronous view maintenance in VLSD databases. The main approach is to defer expensive views by identifying RVTs & LVTs. PNUTS is also supported by a consistency model to hide details for replication of views. They have also listed the supported as well as unsupported views. Evaluation also reveals the performance of PNUTS on fault tolerance, throughput, complexity, query cost, maintenance, view staleness, latency, etc.

In [16] authors have discussed issues related to materialized views and their maintenance in Peer Data Management systems by using schema mappings (SPDMS). They have designed a hybrid peer architecture that consists of peers and super peers. Also, concepts of local, peer and global views have been developed to handle global view maintenance by handling peer views in local PDMS, where, relations are numbered. Mapping rules guide the changes to map one version number to a new version. A push-based algorithm for view maintenance has been developed to handle view maintenance in a distributed manner.
### III. COMPARATIVE STUDY

We have analyzed the various research works on several parameters and presented their comparison in the table below.

<table>
<thead>
<tr>
<th>Features</th>
<th>Technique</th>
<th>Author</th>
<th>Changes handled</th>
<th>Proposed work</th>
<th>Query Language Supported</th>
<th>Meta Data supported</th>
<th>Advantages</th>
<th>Disadvantages</th>
<th>Tool support / implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Gupta, Mumick, Rao &amp; Ross</td>
<td>Handling of local changes in view definition</td>
<td>Redefinition s of all SQL clauses + Guidelines for users &amp; DBA</td>
<td>SQL based view definition language</td>
<td>Additional information kept with materialization on</td>
<td>Query &amp; cost optimization</td>
<td>Only single materialized view changes addressed</td>
<td>Not addressed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(2001)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mohania</td>
<td>Changes in view definition</td>
<td>Expression trees + Relational binary operators + Join &amp; derive count values</td>
<td>SQL clauses: Select, From, Where</td>
<td>Additional results also materialized</td>
<td>Re-computation not needed + Cost of computing decreased</td>
<td>Overheads in maintain additional materialized results</td>
<td>JAVA + JDBC + MS-Access</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(1997)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Lee, Nica &amp; Elke</td>
<td>Schema changes of data sources</td>
<td>EVE framework + replacement strategies + Algorithms</td>
<td>E-SQL view definition language</td>
<td>Meta Knowledge base</td>
<td>Handling changes in large dynamic environment (WWW) + A general framework</td>
<td>Only addressed schema changes in sources + No cost and quality issues addressed</td>
<td>JAVA + JDBC + MS-Access</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(2002)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Dhote &amp; Ali</td>
<td>Selection of views to minimize query response time</td>
<td>AND/OR DAG to minimize the query response time + Optimization algorithm</td>
<td>SQL based</td>
<td>x</td>
<td>Heuristic based algorithm + Simple approach</td>
<td>Algorithm does not work well on certain cases + Cant be used on whole data cube (works only on lattice)</td>
<td>Not addressed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(2007)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Gupta &amp; Mumick</td>
<td>Global evaluation plan for queries + Data cubes</td>
<td>AND/OR view graphs + Greedy heuristics based algorithms</td>
<td>SQL based</td>
<td>x</td>
<td>Optimal solution for special cases (AND/OR views) + Polynomial time heuristics</td>
<td>Approximation in view-selection problem not addressed + Problem in AND view graphs not NP-hard + Solution fairly close to optimum</td>
<td>Not mentioned</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(2005)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Karde &amp; Thakare</td>
<td>Algorithm for creation and maintenance of views</td>
<td>Not mentioned</td>
<td>x</td>
<td>Query performance improved</td>
<td>Only distributed environment s highlighted</td>
<td>Not addressed</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(2010)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reference</td>
<td>Title</td>
<td>Methodology</td>
<td>Advantages</td>
<td>Disadvantages</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------</td>
<td>-------</td>
<td>-------------</td>
<td>------------</td>
<td>---------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ashadevi &amp; Balasubramanian [9]</td>
<td>View selection</td>
<td>Cost-effective view selection under storage space constraints</td>
<td>Framework for selecting views + Algorithm for the same + Cost metrics</td>
<td>All cost metrics considered</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Yang &amp; Chung (2006) [14]</td>
<td>View selection</td>
<td>Attribute-value density + Clustered tables + Selection of views based on clustered/reduced tables</td>
<td>ASVMRT algorithm for view selection</td>
<td>Faster computation time + Reduced storage space + 1.8 times performance better than conventional algorithms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chen, Zhang &amp; Elke (2006) [3]</td>
<td>View maintenance</td>
<td>Source Data updates + Preserving &amp; non-preserving schema changes + 3 types of anomalies</td>
<td>DyDa Framework + Dependency &amp; Correction algorithms</td>
<td>Maintenance of reduced tables not addressed + Updating Reduced tables needs attention</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Almazyad &amp; Siddiqui (2010) [10]</td>
<td>View maintenance</td>
<td>Incremental view maintenance + synchronization between DW and source + lost update notifications</td>
<td>Framework with version store</td>
<td>Extra cost on data updates + Cannot maintain mixed updates in single process</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ling &amp; Sze (2001) [12]</td>
<td>View maintenance</td>
<td>Update anomalies + Notification messages</td>
<td>Compensating algorithms + Version numbers</td>
<td>Algorithms does not require quiescent state before views can be refreshed</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Agrawal, Silberstein, Cooper, Srivastava &amp; Ramakrishnan (2009) [15]</td>
<td><strong>View maintenance</strong></td>
<td><strong>Asynchronous view (Remote view Tables RVT, Local View Tables LVT)</strong> &amp; <strong>Replication of views</strong></td>
<td><strong>Deferred Indexes &amp; Views in Very Large Scale Distributed databases, horizontally partitioned</strong></td>
<td><strong>PNUTS + consistency model + RVTs &amp; LVTs + Group-by views, select views, indexes &amp; equi-join views</strong></td>
<td><strong>SQL based</strong></td>
<td><strong>Metadata for view definitions &amp; partitions</strong></td>
<td><strong>Improved client latency + Improved scalability + Balanced view staleness, system complexity + Improved query cost</strong></td>
<td><strong>Views maintenance adds load to system + VLSD are complex + Decreased throughput + Complex failure recovery</strong></td>
<td><strong>C++ + FreeBSD 6.3 (Linux can also be used)</strong></td>
</tr>
<tr>
<td>Qin, Wang &amp; Du (2005) [16]</td>
<td><strong>View Maintenance</strong></td>
<td><strong>Global view maintenance by maintaining local views in PDMS</strong></td>
<td><strong>Schema changes + Schema mappings amongst peers</strong></td>
<td><strong>Hybrid peer architecture (P2P &amp; Peer-super peer) + Local, global &amp; peer views + Rules to use updategrams &amp; boosters + Push-based Algorithm</strong></td>
<td><strong>Not mentioned</strong></td>
<td><strong>One kind of Super peer maintains metadata for mapping schemas in inter-peers or inter-peer changes in local PDMS</strong></td>
<td><strong>Decentralized maintenance strategy + Higher efficiency + Parallelism + Efficient in 80-20 distribution + Central bottlenecks avoided</strong></td>
<td><strong>Information sharing is complex &amp; difficult in PDMS + Querying not addressed</strong></td>
<td><strong>Simulation system developed in JAVA</strong></td>
</tr>
</tbody>
</table>

### IV. CONCLUSION

In this paper we have presented an analysis of different approaches being proposed by various researchers to deal with the materialized views in data warehouse namely- view adaptation & synchronization, view selection and view maintenance. We have examined these techniques on various parameters and provided a comparative study in a tabular manner.

### V. FUTURE WORK

As future work, we will direct our research towards batch-oriented view maintenance and selection strategies. A thorough investigation of the methodologies to handle materialized views in highly distributed environments for query processing and analysis seems worth attention.

### REFERENCES


AUTHORS PROFILE

Garima Thakur is pursuing her M.Tech in Information Technology from Guru Gobind Singh(5,3),(994,992) Indraprastha University, Delhi, India. She has done her B.Tech in Computer Science branch from the same university in the year 2009. She is doing her research work in the field of Data warehouse & Data Mining and Knowledge Discovery.

Dr. (Mrs.) Anjana Gosain is working as reader in University school of information technology. She obtained her Ph.D. from GGS Indraprastha University & M.Tech in Information Systems from Netaji Subhas Institute of Technology (NSIT) Delhi. Prior to joining the school, she has worked with computer science department of Y.M.C.A institute of Engineering, Faridabad (1994-2002). She has also worked with REC kurukshetra. Her technical and research interests include data warehouse, requirements engineering, databases, software engineering, object orientation and conceptual modeling. She has published 18 research papers in International / National journals and conferences.