ISSN 2229-5518

QOP: Proposed Framework for Materialized View Maintenance in Data Warehouse Evolution

Hemant Jain, Anjana Gosain

Abstract— A data warehouse is generally applied to discover and integrate data from independent data source. In data warehouse large numbers of materialized views are stored in order to provide fast access to the integrated data. Maintenance of materialized views is one of the critical tasks in warehousing environment. They must be up to date to ensure accurate results and also to speed up the query processing significantly. Updating of materialized views is also important to ensure consistency because the source data usually change over time. In literature, few frameworks have been proposed for materialized view maintenance. Each of these frameworks has different characteristics, capabilities and complexities. But none of these frameworks focus on query optimization. In this paper, we present a theoretical framework called *QOP* to support data warehouse view maintenance. The proposed framework improves the functionality of previously proposed frameworks by primarily focusing on changes in the maintenance phase. This framework also provides the additional concept of query optimization in the maintenance phase.

Index Terms— Data warehouse, Data source, Incremental maintenance, Materialized view, Query optimization, Self maintenance, End Users.

1 Introduction

Data warehouse act as a central repository that collect data from different autonomous, distributed and heterogeneous data sources. Traditionally, data warehouses have been used to provide storage and analysis of large amounts of historical data [20]. Due to the large amount of data in the data warehouse, the issue of maintaining a materialized view draw much attention. Materialized views are the derived relations, which are stored as relations in the database [15]. Materialized views can be used for reducing query response time. Materialized views approach is quite promising in efficiently processing the queries because of the query intensive nature of data warehousing. To keep, a materialized view up-to date there is a need to propagate the changes from remote data source to the destined materialized view in the warehouse. Data warehouse contains many of materialized view to access data quickly and efficiently. In a data warehouse, the query expressions that define materialized views may be stored at different database sources residing at different sites. The sources may inform the data warehouse when an update occurs but they might not be able to determine what data is needed for updating the views at the data warehouse [18]. To avoid accessing the original data sources and increase the efficiency of the queries posed to a DW, some intermediate results in the query processing are stored in the DW. These intermediate results stored in a DW are called materialized views [17].

Many algorithms relating to the maintenance of materialized views have proposed in the literature. These algorithms may be divided into two categories i.e. incremental or self maintainable. In Incremental view maintenance approach, only changes in the materialised views of the data warehouse are computed rather than recomputing every view from scratch [19].

On the other hand when a view together with a set of auxiliary views can be maintained at the warehouse without accessing base data, we say the views are self-maintainable [2].

For incremental maintenance materialized views is defined as select-project-join (SPJ) with N base relation (R_1, R_2,....R_n). The materialized view in the data warehouse is defined as:

$$V=\Pi_{proj}(\sigma_{cond}(R_1 \bowtie R_2 .... \bowtie R_i ...... \bowtie R_n)$$

Where proj is a set of attribute name and cond is a Boolean expression, since R_1,....R_n are separate relations.

Once the changes in the base relations are obtained, the changes in the MV will be calculated. MV includes insert, delete, and modification in the tuples. For the insert tuples ΔV is incremental change. Then we increase tuple in MV; for the delete tuple, if ΔR_1 is removed from R_2, then $\Delta R_1 = \Delta R_1 \bowtie \Delta R_2$ and then equivalent tuple is removed from the MV.

The changes in the base relation (R_1, R_2,... R_n) will affect the change in MV. The change in the MV can be achieved in accordance with (R_1,R_2,....R_n).

For example, $V<R_1,R_2>$ show changes of MV which are considered by R_1 and R_2 changes.

$$V=\Pi_{proj}\sigma_{cond}(R_1 \bowtie R_2)$$

Hence incremental maintenance can be written as $V<R_1,R_2>$.

For self maintenance view V is defined over a set of base relations R, i.e (R= R1, R2.....Rn). After the changes in ΔR is obtained to base relations in reaction to which view requests to be maintained. If ΔV can be computed using only the MV in data warehouse and the set of changes in base relations ΔR, then the view is known as self maintainable otherwise we are involved a set of auxiliary view denoted by A. Defined on the same relation as view V. Therefore the set of views {V}UA is self maintainable.

- ***Hemant Jain*** *is currently pursuing M.Tech in computer science & engineering from Guru Gobind Singh Indraprastha University, Delhi, India. E-mail: hemantjain.ipu@gmail.com*
- ***Dr. (Mrs.) Anjana Gosian*** *is working as reader in University School Of Information technology. She obtained her Ph.D. from GGSIP University, Delhi, India. E-mail: anjana_gosain@hotmail.com.*

ISSN 2229-5518

The proposed framework expanded the WHIPS (Warehousing Information Prototype System) [3] by introducing the concept of query optimization and combines both incremental & self maintenance with proposed framework. It analyzes process between the sources and warehouse is made by using FIFO network.

The purpose of the query optimization is to optimize and reduce the complexity of the query while taking into account of present materialized view. In a relational database all information can be found in a series of tables. A query therefore consists of operations on tables. The most common queries are Select-Project-Join queries [23]. For a given query, there are many of plans that a DBMS can follow to process it and produce its answer. All plans are equivalent in terms of their final output but vary in their cost, i.e., the amount of time that they need to run. The area of query optimization is very large within the database field [24]. Materialized views mainly contained results of the queries.

The organization of the paper is as follows, literature review is discussed in section 2. In section 3, we discuss our proposed framework in detail. Section 4 presents the framework process discussed in previous section. And finally, we conclude in section 5.

2 Literature Review

The related work of various authors in context to incremental view maintenance [3, 6, 7, 8, 10, 11, 12, 13, 16] is presented below:

In [3] authors have described the architecture of the Whips prototype system, which collects, transforms, and integrates data for the warehouse. In [6] authors have proposed a new incremental approach to maintaining materialized views both in the data warehouse and in the data marts. In [7] authors have proposed a new compensation algorithm that is used in removing the anomalies, caused by interfering updates at the base relations. In [8] authors have proposed a maintenance algorithm that does not need the compensation step and applies to general view expressions of the bag algebra. In [10] authors have proposed an incremental maintenance method for temporal views that allows improvements over the recomputation from scratch. In [11] authors have presented an incremental view maintenance approach based on schema transformation pathways. In [12] 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. In [13] authors have developed the change-table technique for incrementally maintaining general view expressions involving relational and aggregate operators. Incremental maintenance technique is adopted in this paper [16]. In this idea and strategy of minimum incremental maintenance is presented.

The related work of various authors in context to self maintainable maintenance [1, 2. 4, 5, 9, 14] is presented below:

In [1] author has reported on some interesting new results for conjunctive-query views under insertion updates. 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. In [4] authors have proposed an incremental technique for efficiently maintaining materialised views in these high performance applications by materialising additional relations which are derived from the intermediate results of the view computation. In [5] author has focused on the problem of determining view in the presence of functional dependencies. In [9] authors gave a preliminary result on self-maintainability of deletions of views over XML data. This paper [14] provided an online view self-maintenance method based on source view's increment to keep the materialized view consistent with the data source.

The above mentioned literatures have highlighted efforts related to incremental and self maintenance which is exercised in our proposed framework. The next section presents the proposed framework which has taken accounts the modification done at the maintenance level.

3 The Architecture of Proposed Framework QOP

3.1 Framework Level & components

In this Framework, relational model is used to represent the warehouse data. In the relational model, views are defined and relation stored in the warehouse. The Framework is divided into four levels namely- Source level, Maintenance level, Warehouse level and User level. At the Source level, source data is converted in to the relational model by using the monitor and formatter and then sent it to the next higher level. In maintenance level, the maintenance process between the sources and warehouse is done by using FIFO network. In warehouse level, warehouse receives all view definitions and modifications to the specific syntax of the warehouse database; internal maintenance is also done in this level. In User level, users communicate to warehouse by using data query. Each level comprises of a number of components to manage particular tasks.

ISSN 2229-5518

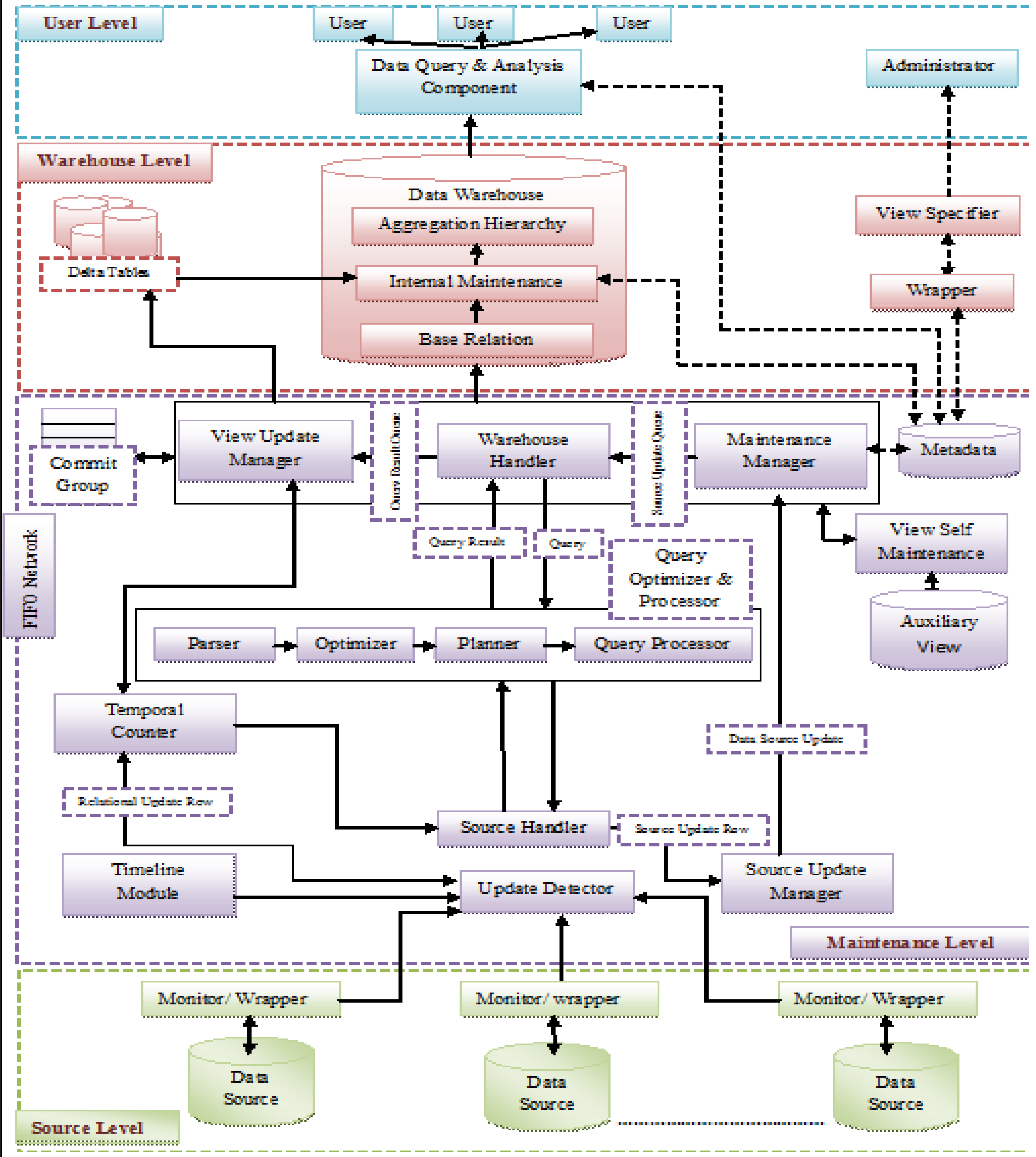

Fig 1. Architecture of QOP: Proposed Framework

ISSN 2229-5518

Now, let us discuss the working of the above mentioned levels and their respective components in more details:

Table1.
Source Level Components

1. Source Level	
Data Source	It is used to feed data into the data warehouse. It can be of any format like plain text file, relational database, other types of database etc.
Monitor	It detects the modifications that are performed on its source data. These modifications are then sent to the maintenance level.
Wrapper	It receives all view definitions and all adjustments to the view data in an internal format, and converts them to the specific syntax of the warehouse database.

Table2.
Maintenance Level Components

2. Maintenance Level	
Update Detector	It is responsible to achieves relational updates from data source by different ways and affixes them to the relational update row.
Timeline Module	It provides consistency of view maintenance during parallel processing.
Temporal Counter	It is responsible for correct detection of concurrent updates.
Source Handler	It calculates the source updates and handles the source updates and affixes them in to the source update row.
Source Update Manager	It is responsible for transferring the latest source updates to the view maintenance manager and managing each update and its relational updates.
Query Optimizer & Processor	It handles optimization and execution of declarative queries. After receive maintenance queries from the warehouse handler, it processes the query and returns the precise query result.
Maintenance Manager	It is responsible for receiving the latest Data source updates from source update manager, generating view maintenance tasks and affixes them to the source update row.
Warehouse Handler	It puts the final maintenance query result into the query result row and informs the view maintenance manager.
View Update Manager	It uses the maintenance query results to update the materialized views at the data warehouse.
Commit Group	Commit group is used by view update manager to save the maintenance result temporarily.
View Self Maintenance	It is responsible for maintaining the materialized views at the data warehouse without access to the base relation.

Table3.
Warehouse Level Components

3. Warehouse Level	
Data Warehouse	A data warehouse (DW) is a database used for reporting and analysis. The data stored in the warehouse are uploaded from the operational systems.
View Specifier	It parses the view into an internal structure also it adds significant information from the metadata.
Delta Tables	It is used to store the modifications to be applied to the materialized views in the data warehouse.

Table4.
User Level Components

4. User Level	
Data Query & Analysis Component	It is responsible for communication between users and data warehouse by using data queries, it also fulfilling the information needs to specific end users.
Administrator	Views are defined at the view specifier by an administrator.

4 Framework Operation

4.1 Source Level

This level represents the different data sources that feed data into the data warehouse. Each source may be completely independent of the warehouse or it can be of any format and their functions are isolated from others. DWH is generally applied to explore and integrate data from several data sources, which can be seen as a set of materialized views. In this source data is converted into relational data by using the source monitor and wrapper. Each monitor detects the modifications that are present into the data source. Then these modifications are sent to the maintenance level. Each wrapper is responsible for translating single source queries from the internal relational representation used in the view tree to queries in the native language of its source [3].

Below shows the communication processes between source level and the warehouse level:

1. Data source will send out notification to the warehouse after it finished a data updating at the source level.

ISSN 2229-5518

2. Data source will send out view update notification when it plans to do materialize view update. It will wait for the acknowledgement from the data warehouse before executing the view update.
3. Warehouse receives the notification and sends back the query to the source level about the update.

After finishing the maintenance process warehouse will send out acknowledgement to data source level via maintenance level to notify it of the accomplishment of the maintenance of materialized view in data warehouse.

4.2 Maintenance Level

Processing of maintenance level:

1. A timeline module provides consistency of view maintenance during parallel processing. It assigns a timestamp to each incoming message and query result message. Temporal counter is used for correct detection of concurrent updates and also handling parallel processing. In this, all the updates will be answered at the same time in parallel manner.
2. The source update detector achieves relational updates from data source by different ways and affixes them to the relational update row according to their committed sequence. Using this relational updates of a committed sequence and significant base relations, the source handler evaluates the equivalent source update based on the data source in the materialized view and affixes it in to the source update row.
3. The source update manager sends data source updates to the maintenance manager. The maintenance manager allocates a unique maintenance number for each received data source update and affixes them to the source update row. A task related to view maintenance is concerned by the maintenance manager for each data source update in the source update row and the equivalent query arrangement is executed to the warehouse handler with the help of query optimizer & processor.

   Query optimizer & processor includes four parts, functionality of following parts is given below [24]:

   - Parser – It checks the validity of the query and then translates it in to an internal form, usually a relational calculus expression or something equivalent.
   - Optimizer – It examines all algebraic expressions that are equivalent to the given query and chooses the one that is estimated to be the cheapest.
   - Interpreter – The Code Generator or the Interpreter transforms the access plan generated by the optimizer into calls to the query processor.
   - Query Processor – It actually executes the query.

Query Processing Algorithm
Procedure Query Processing (QP)
Input - Q(x) [i] as query
Output – QR(x) [i] as query result
begin
Step-1 Send Q(x) [i] to data source (DS [i]);
Step-2 Receive QR(x) [i] from DS [i];
Step-3 If (data update, DU(y) [i] exists in source update queue and x>y)
/* data updates happened concurrently*/
Step-4 QR(x) [i] = QO (Q(x) [i], DU(y) [i], QR(x) [i])
Step-5 End if
Step-6 Return QR(x) [i];
end

Fig2. Algorithm for Query Processing

Query Optimization Algorithm
Procedure Query Optimization (QO)
Input – SQL Query
Output – Execution Plan
begin
Step-1 SQL query contain many of relations, access each relation in the query by all possible manners.
Step-2 Generate a query tree for the sql query.
Step-3 Selection of plans to process each node in query trees and ordering the nodes for execution.
Step-4 Obtained the estimation cost of each plans & reserved the cheapest plan for further consideration.
Step-5The cheapest plan is final output of the optimizer to be used to process the query.
end

Fig3. Algorithm for Query Optimization

4. The query optimizer & processor queries related data source according to the query arrangement. After receiving the maintenance query from the warehouse handler, the source handler processes it through the compensation of source relations and returns the specific query answer. Then the warehouse handler attaches the final maintenance query answer into the query result row and informs the view maintenance manager.
5. After completion of maintenance query related to a data source update, the maintenance manager removes this data source update from the source update row and also source update manager removes the data source update and its relational updates from the related queues upon receiving its completion indication.
6. The view update manager will save the maintenance results in the commit group temporarily for efficient updating of materialized view in the DWH. Materialized view at the data warehouse is updated by view update manager using the final maintenance query result in the query result row and then removes the query results.

7. Self maintenance can be processed to maintain the affected view. The affected view can be maintained by use of the auxiliary view existed in the maintenance level.
8. Metadata is used to maintain the materialized views. It can be obtained by manually or through automated processes. Metadata module maintains catalogue information about the source and how to contact them, the relations stored at each source and the schema of each relation. The metadata module also keeps track of all view definition [3].

4.3 Warehouse Level

In this, materialized view is used to store aggregate, recomputed and summarized data. The Data Warehouse is a group of one or more materialized views of the data source. There are many of maintenance policies residing that enables the system to maintain only the necessary data in the data warehouse instead of maintaining the whole data warehouse. Base relations in the data warehouse are used to store a layer of self-maintainable relation. They are also materialized views and can be thought of as the cleaned and filtered source data required in the data warehouse. Internal maintenance in the data warehouse provides session consistency and also describes the process of maintaining the pre aggregations in the DWH. Internal Maintenance is started by the system accord [21].

In [21] authors describe the complete process of internal maintenance to maintain materialized view in the DWH. Modification in the materialized views is stored in the delta tables and it is also used to provide separation between maintenance level and warehouse level.

In this level, Metadata provides variety of information stored in metadata repositories such as information related to contents, structure and location of the warehouse, information of different issues like security, authentication etc.

4.4 User Level

This level of framework describes the user requirements & tasks they need to perform with the help of data warehouse. User requirements must be collected from people who will actually use and work with the data warehouse system [22]. After the maintenance process, data in the data warehouse is readily accessible to end user applications for querying and analysis purpose. Firstly, end users starts to make simple queries, after that they tend to comes with more complex forms of data analysis.

5 Conclusion

View maintenance is one of the major jobs in data warehousing environment. Due to improper view maintenance, required results are practically impossible to achieve from a data warehouse. In this paper we have proposed a novel framework for data warehouse view maintenance that is primarily driven by changes in the materialized view maintenance process of the data warehouse. Our proposed frame work has been divided into various levels namely: Source level, Maintenance level, Warehouse level, user level. The source level converted source data in to the relational model. The maintenance level performs maintenance process between the sources and warehouse is done by using FIFO network. The warehouse level receives all view definitions and modifications to the specific syntax of the warehouse database; internal maintenance is also done in this level. The user level provides communication between users and warehouse by using data query. The concept of query optimization is also providing in maintenance level to reduce the query cost and to speed up the query processing task.

References

[1] N. Huyn, "Efficient View Self-Maintenance", Proceedings of the ACM Workshop on Materialized Views: Techniques and Applications, Montreal, Canada, June 7, 1996.

[2] D. Quass, A. Gupta, I. S. Mumick, and J. Widom, "Making Views Self Maintainable for Data Warehousing", Proceedings of the Conference on Parallel and Distributed Information Systems, Miami Beach, FL, December 1996.

[3] J. L. Wiener, H. Gupta, W. J. Labio, Y. Zhuge, H. Garcia-Molina, and J. Widom, "A system prototype for view maintenance", Proceedings of the ACM Workshop on Materialized Views , June 7, 1996, pp. 26-33.

[4] Vincent, M. Mohania, Y. Kambayashi., " A Self Maintainable View Mainte nance Technique for Data Warehouses", ACM SIGMOD (1997) 7-22.

[5] N. Huyn, "Exploiting Dependencies to Enhance View Self Maintainability" http://wwwdb.stanford.edu/ pub/papers/fdvsm.ps. Technical Note, 1997.

[6] Mukesh Mohania, Kamalakar Karlapalem and Yahiko Kambayashi, "Main tenance of Data Warehouse Views Using Normalisation", Dexa'99, LNCS 1677, pp. 747-750, 1999.

[7] Tok Wang Ling, Eng Koon Sze. " Materialized View Maintenance Using Version Numbers", Proceedings of the Sixth International Conference on Database Systems for Advanced Applications, 1999.

[8] Gianulca Moro, Claudio Sartori, "Incremental Maintenance of Multi-Source Views", Proceedings of the 12th Australasian database conference 2001.

[9] Cheng Hua, Ji Gao, Yi Chen, Jian Su, "Self-maintainability of deletions of materialized views over XML data", International conference on machine learning and cybernetics, 2003.

[10] Sandra de Amo, Mirian Halfeld Ferrari Alves, "Incremental Maintenance of Data Warehouses Based on Past Temporal Logic Operators",J.UCS 10 (9): 1035-1064 (2004).

[11] Hao Fan, "Using Schema Transformation Pathways for Incremental View Maintenance", Proceedings of the 7th international conference on Data Warehousing and Knowledge Discovery (2005).

[12] Hao He, Junyi Xie, Jun Yang, Hai Yu, "Asymmetric Batch Incremental View Maintenance", 21st International Conference on Data Engineering, 2005.

[13] Himanshu Gupta, Inderpal Singh Mumick, Incremental maintenance . Information Systems, Vol. 31, Nr. 6 (2006), p. 435--464.

[14] Hai Liu, Yong Tang, Qimai Chen, The Online Cooperating View Mainte nance Based on Source View Increment. CSCWD 11th International con ference, pp. 753-756, April 2007

[15] A.N.M.B. Rashid and M.S. Islam, " Role of Materialized View Mainte nance with PIVOT and UNPIVOT Operators", IEEE International Ad vance Computing Conference (IACC'09), Patiala, India, pp. 951-955, March 6-7, 2009.

[16] Lijuan Zhou, Qian Shi, Haijun Geng, "The Minimum Incremental Mainte

ISSN 2229-5518

nance of Materialized Views in Data Warehouse", 2nd International Asia Conference (CAR), March 2010.

[17] C.Zhang, Xin Yao, "An Evolutionary Approach to Materialized Views Selection in a Data Warehouse Environment", IEEE Transactions on sys tems, Man, and Cybernetics, Vol.31, No.3, August 2001.

[18] X.Wang, L.Gruenwald, G.Zhu, "A Performance Analysis of View Mainte nance Techniques for Data Warehouses", Submitted to Journal of Knowl edge and Data Engineering, July 2004.

[19] Y. Zhuge, H. Garcia-Molina, J. Hammer, and J. Widom. "View mainte nance in a warehousing environment", In *SIGMOD*,pages 316–327, San Jose, California, May 1995.

[20] Gang Luo, Jeffrey F. Naughton, Curt J. Ellmann, Michael Watzke. "A Com parison of Three Methods for Join View Maintenance in Parallel RDBMS", ICDE 2003: 177-188.

[21] Michael Teschke, Achim Ulbrich, "Concurrent Warehouse Maintenance without Compromising Session Consistency", Proceedings of the 9th Inter national Conference on Database and Expert Systems Applications. pg 776-785, 1998.

[22] R.M. Bruckner, B.List, J.Schiefer, "Developing Requirements for Data warehouse Systems with Use cases", AMCIS Proceedings. Pg-66, 2001.

[23] R.Ghaemi, A.M.Frad, H.Tabatabaee, M.Sadeghizadeh, " Evolutionary Query Optimization for Heterogeneous Distributed Database Systems", World Academy of science. September 2008.

[24] Yannis E. Ioannidis, "Query Optimization", published in Journal ACM Computing Surveys, volume 28, march 1996.