ISSN XXXX XXXX © 2019 IJESC

Research Article Volume 9 Issue No.4

A Constraint Based Approach to View Maintenance

Anjana Gosain[1], Sangeeta Sabharwal[2], Rolly Gupta[3]
University School of Information Technology, GGS Indraprastha University, Delhi, India[1]
Netaji Subhas Institute of Technology, Delhi University, Delhi, India[2, 3]

Abstract:
Data Warehouse evolution is a critical problem in present scenario, therefore it needs to be updated periodically according to different type of evolution of information sources. Materialized views and view maintenance are playing an important aspect in practice, as they provide a systematic way to recomputation, maintenance and evolution of data. A number of view maintenance policies have been discussed to satisfy data concurrency and performance requirements. A view can be refreshed immediately after every update or may be refreshed periodically or on-demand. The view maintenance policies have implications on validity of query results and affect the performance parameters. In this paper, we introduce a Per-View constraint based update view maintenance policy in the data warehousing environment.

General Terms: Information Systems, Database Management, Database Administration.

Keywords: View Maintenance, Materialized views, Integrity maintenance.

1. INTRODUCTION

Materialized views are playing a vital role in providing solutions to application problems such as Decision Support Systems, Retailing, Data Warehousing, Data integration etc [18, 19]. Materialized views acts as a cache that can gather information from distributed databases and can provide support to 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. Therefore, frequency of deletion, addition and update operations on the base relations rises unexpectedly. However, a penalty (in terms of cost, speed) is imposed on every updation of transaction during view maintenance [14,15]. So, in order to reduce this penalty, several policies [6, 7,12] have been proposed, based on read/update transaction or on-demand queries. Three common policies include: (1) Immediate views, (2) Deferred views and (3) Snapshot views. In Immediate views [5, 6], a view is refreshed immediately after every update to the base table. This allows faster querying but slows the updation process. In Deferred views [7,8,9,10], a view is maintained on-demand, i.e. when the view is queried. This allows faster updates but slows querying. While in Snapshot view [11,12], a view is refreshed periodically and provides faster querying and updation process, but queries can read data that is not up to date with base tables. Therefore, to implement materialized views, maintenance policy plays a major role depending upon the requirements. This paper introduces a Per-View constraint based update view maintenance policy in the data warehouse environment. This constraint designing approach can be used for enhancing the view maintenance performance in DW by installing sources updates / DW views only when constraints are violated. This paper is structured as follows. Section 2 discusses literature survey. Section 3 outlines the Per-View constraint based update view maintenance approach. Section 4 discusses the relevant features to be taken into account during updating and constraint maintenance/ checking. Finally, section 5 summarizes the main conclusions and points out aspects for further research.

2. LITERATURE SURVEY

The concept of multiple maintenance policies was firstly supported by ADMS system [Roussopoulos86, 91]. The system concentrated on the advantages of materializing a view using view-caches, which are join indices rather than views. They also presented an analytical and experimental study, which compares the benefits of deferred incremental maintenance over recomputation. But, it lacked the performance comparison of different maintenance policies. [Zhuge95] and [Hull96] studied the consistency of views in distributed environments. [Blakeley90] presents an analytical performance comparison of join indices, fully materialized views and join view recomputation. [Hanson97] presents another analytical comparison of answering queries over single view by means of recomputation, immediate materialized view and deferred materialized view. They founded that either method can perform better depending upon selectivity; i.e. Recomputation is best for single SP view, select-project-join for SPJ, immediate maintenance for views and deferred maintenance for single aggregate view. [Srivastava88] studied optimal refresh policies based on queuing models, response time parameterization and cost constraints. Explicit or periodic refresh of a view group containing snapshot views was first proposed in [Adiba80]. The implementation techniques are described in [Lindsay86, Kahler87, Segev89]. [Lindsay86] presents changes to snapshot view based on update tags on base tables. [Kahler87] and [Segev89] focuses on techniques for maintenance of logs and computation of net update to views. [Blakeley86, Qian91, Griffin95, Colby96] proposed several incremental algorithms for view maintenance. [Adelberg95] presented strategies for view updating based on different priorities for transactions. [Kawaguchi97] discussed concurrency control and serializability in deferred views. In the recent decades, [Hartmut00] discussed the view maintenance techniques for hierarchical semi structured data. [Hamid02] presented the concept of knowledge warehouse. Maintenance of XML web warehouse was proposed in [Chen05]. The concept of multi-agent framework for immediate incremental view maintenance was presented in

[Yeung05]. [Nasir07] and [Erik08] discussed virtualization and fast track architecture in data warehousing. [Mouzoune12] highlighted the concept of e- maintenance framework. Many issues in maintenance of materialized view have yet to be explored more. There have been several isolated attempts to study certain aspects, but there is still a need for further study with actual implementation.

3. PER-VIEW CONSTRAINT BASED UPDATE VIEW MAINTENANCE

Most databases, like deductive or relational ones, allows definition of intentional information such as views or integrity constraints.Views and constraints are the most traditional types of intentional information. *Views* are means of deductive rules that allow defining new facts (view or derived facts) from stored (base) facts, while *constraints* state the conditions that need to be satisfied by each state of the database. Databases are updated through the application of *transactions* that consist of a set of updates of base facts. While applying a transaction, database consistency may be falsified, i.e. some integrity or dependency or freshness constraint may be violated. Therefore, databases should incorporate some mechanism to ensure constraints satisfaction; after the application of a transaction. There are several approaches of resolving this conflict; which are reasonable and the correct approach to be considered depending on the semantics of constraints and database. The best known approaches are constraint checking and constraint maintenance. *Constraint checking* rejects the transactions, if violates some constraint, while *constraint maintenance*, is concerned with trying to identify additional updates to be added to the original transaction to guarantee that the resulting transaction does not violate any constraint. Translating an update of a set of derived facts into appropriate updates of the underlying base facts is needed. So, we are providing effective policies that are able to obtain all valid solutions, without imposing strong restrictions on the materialized views and integrity constraints. Therefore, a Per-View constraint based update view maintenance policy will be helpful for an efficient data warehousing system.

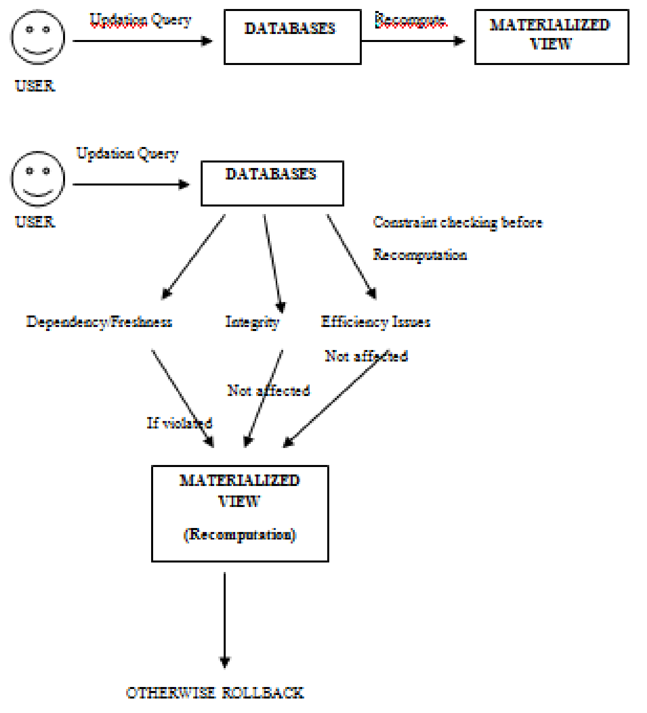

Figure.1. Per-View constraint based update view maintenance approach

4. THE DATA WAREHOUSE MODEL (Per-View Constraint-based Update Maintenance)

The Per-View constraint based update maintenance saves a large maintenance overhead by installing the source updates only when the constraints are violated and only for the DW views that have violated constraints. Defining different constraint for updating process can be categorized into following heads:

4.1 Dependency/ Freshness Constraint

The DW is composed of a set of materialized views, in which views are relations in a relational database. Each view can be defined in terms of other objects; which could be source relations as well as other data warehouse views. For each DW view, let V = <D, F> or V = <I, E> be the view's descriptor where:

***D*: *Dependency predicate*.** This describes the dependency Constraint of the view in terms of data sources or other views.
F*: *Freshness predicate Describes the freshness Constraints on the DW view.
I: Integrity predicate Refers to the linked *In*tegrity Constraint during updation.
E: Efficiency predicate Addresses the efficiency issues during *In*tegrity Maintenance.

The dependencies among the different objects can be modelled by a View Directed Cyclic/Acyclic Graph (VDAG). Here, we are assuming VDAG, Composed of one or more separable Components. A VDAG represents the source relations and DW views (materialized view) as nodes and their interdependencies as edges. But, the source relations appear in a VDAG as leaves. Let ***N*** be the set of all nodes and ***E*** be the set of all edges in graph. A DW view can be in one of three states:

***Fresh*:** Reflects up to-date state of data in the source databases
Unstable: Reflects non up to-date state.
Stale: The data in the view reflect the old data in the source databases.

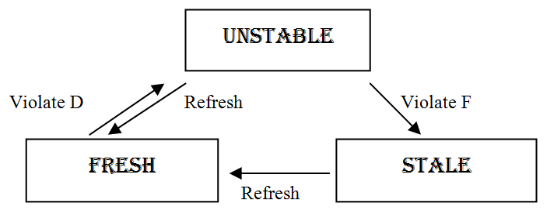

Figure.2. State diagram

After the initial loading, the view appears in the fresh state. If *D* is violated due to source updates but *F* is not violated, then the view moves into the unstable state. If both *D* and *F* are violated then the view moves into the stale state, and when refreshment occurs according to *D*, the view moves back into the fresh state (Fig 2). This concept deals with the freshness constraints of the proposed model. Furthermore, constraints can be bounded on views for updates, based on integrity and efficiency parameters.

4.2 Integrity Constraint

Source relations and DW views are a collection of information, is structured in form nodes of tree. They can be represented in form of nodes and interdependencies as edges. Most of the time, a node contains an information that is not explicitly stated in that particular node; but is stored in another resource and the former node just refers to that resource. This situation between the two sources raises the issues of referential integrity. However, this work is only applicable for schema-based constraints. We propose the methodologies for different updates. The aim is to preserve the integrity constraints in the updated source relations and DW views.

4.2.1 Methodology for Insertion

Firstly, the update operation is for attribute insertion (Fig.3). For attribute, we start checking whether it is a link attribute or not. If it is a link, then the method checks the link destination. If the link attribute contains a valid link, it is registered in link-bases. This linkbase keeps the links to any source inside database. The former condition indicates that insertion can be performed otherwise we need to cancel the insertion. If the target is an element, which contain link attribute then we will inserting all the element(s) without any link checking. Once it is done, we will insert the attribute links using above checking mechanism.

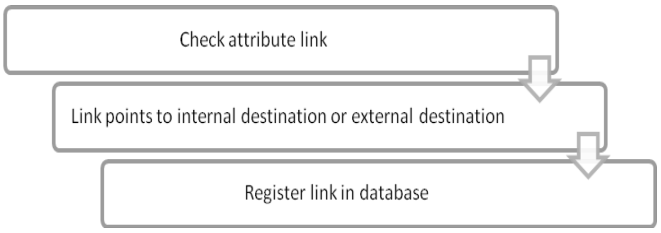
Check attribute link
Link points to internal destination or external destination
Register link in database

Figure. 3. Insertion check

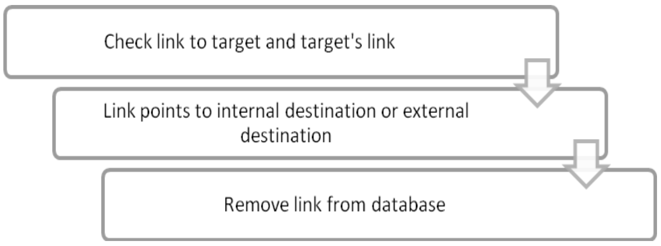
Check link to target and target's link
Link points to internal destination or external destination
Remove link from database

Figure. 4. Deletion check

4.2.2 Methodology for Deletion

The main concern for deletion is to avoid deleting a target, referred by other node(s). So, we need to check the linkbase before any deletion. If any source refers to the target node, cancel the deletion. If the deletion occurs, then remove all links to the targeted node from the link base. The delection check is shown in figure 4. However, If the target is an element(s), firstly remove the attribute links attached in the element(s) before starting deleting the element(s).

4.2.3 Methodology for Replacement

A replacement can be a deletion operation followed by an insertion. Therefore, the previous checking processes can be utilized. Before a replacement occurs, linkbase is checked. Replacement process is cancelled , if the old target is referred by another source. Otherwise, check whether the new target has a link. Depending on the link source, the old link direction is replaced with the new direction in the linkbase. In any kind of database process, a persistent reference is an important factor for integrity purposes. The current solution regularly checks the broken links and rectifies those founded. Our aim in this work is to avoid the broken links in the first place by checking before updating a source relations and DW views. So, we proposed a set of checking functions which are triggered before the actual update takes place; thus checks the destination of the link, register or remove the link in the database, and perform other operations before the updation decision.

4.3 Handling Efficiency Issues during Integrity Maintenance

We periodically address efficiency issues during the process of integrity maintenance. In this context, we can improve efficiency of existing methods by defining the ordering mechanism in maintenance of integrity constraints. For this reason, view updating is integrated into integrity maintenance. A technique for translating view updates efficiently, aims at reducing the number of alternatives considered during the process of view updating and the access required to perform this translation.

4.3.1 Making Explicit an Order for Integrity Maintenance

Ordering mechanism is provided by the Precedence Graph, by taking into account syntactical information associated to each integrity constraint. Violations of database consistency are produced because some event rule associated to an integrity constraint becomes true. Thus, the Precedence Graph will help us to state all relationships between repairs and potential violations of these conditions. Structuring the process of integrity maintenance determines the order of handling conditions to minimize the number of times that each condition is processed. The Precedence Graph provides the information to define an order of handling conditions. Intuitively, conditions without incoming edges should be considered first as they cannot be violated by the maintenance of any other condition. When a condition has been processed, it is removed from the Precedence Graph and the integrity maintenance process continues with the rest of the conditions.

4.3.2 Translating a Repair Corresponding to a View Update

When views appears in an integrity constraint, translation is done efficiently for a repair that consists on a view update. This can be performed more efficiently by performing an analysis of the view update request and of the database contents before translations. This analysis is done during the Analysis Step, as it considers only those alternatives that are relevant to the request and reduces the number of accesses. Translations are performed during the Translation Step, based on the results of the analysis step.

4.3.2.1 Analysis Step

Analysis Step determines the event rules in database relevant to translate the update request and to specialize them so that no additional access to the DB is needed to perform the translation. This specialization is performed by adapting a transformation to split a general rule into two rules: one specific rule for concrete instance and a general rule for rest of values.

4.3.2.2 Translation Step

During Analysis Step, the specialized event rules has been obtained to perform the translation without accessing the DB. Translation Step translates a given view update request into all possible translations, thus satisfying request. A translation defines a set of base facts to be inserted or deleted from the current DB. But, if a view is defined in terms of other views, the positive derived events is unfolded until a goal is reached where positive literals are base fact events.

5. CONCLUSION & FUTURE SCOPE

This paper has proposed a Per-View constraint based update view maintenance approach for data warehousing to implement view maintenance strategy. This Per-View constraint based update view maintenance based approach adopts an efficient views maintenance policy. The efficiency is achieved by applying restrictions before the updating process is completed. This approach takes into account the three relevant dimensions that participate into this process, i.e. dependency/ freshness constraint, integrity constraint, Efficiency issues related to integrity. We have proposed the techniques to draw solutions using the Per-View constraint based update view maintenance approach. We conclude that the research in this area is still appealing, for providing effective policies that are able to obtain all valid solutions, without imposing strong restrictions on the materialized views and view maintenance. We are planning to propose a constraint based framework for an efficient data warehousing system. This will help to seek the usefulness of policies in practical applications.

6. REFERENCES

[1].Ashish Gupta, Inderpal Singh Mumick 1995. Maintenance of Materialized Views: Problems, Techniques, and Applications. IEEE Data Eng. Bull. 18(2): 3-18.

[2].Inderpal Singh Mumick 1995. The Rejuvenation of Materialized Views. CISMOD : 258-264.

[3].Yue Zhuge, Hector Garcia-Molina, Joachim Hammer,. Jennifer Widom 1995. View Maintenance in a Warehousing. Environment. In Proc. of ACM SIGMOD.

[4]. R. Hull and G. Zhou 1996. A framework for supporting data integration using the materialized and virtual approaches, Sigmod.

[5].Himanshu Gupta, Inderpal Singh Mumick 2006. Incremental maintenance of aggregate and outerjoin expressions. Inf. Syst. 31(6): 435-464.

[6].Akira Kawaguchi, Daniel F. Lieuwen, Inderpal Singh Mumick, Kenneth A. Ross 1997. Implementing Incremental View Maintenance in Nested Data Models. DBPL 202-221.

[7].N. Roussopoulos and H. Kang 1986. Principles and techniques in the design of ADMS, IEEE Computer, pg. 19-25.

[8].B. Adelberg, H. Garcia-Molina and B. Kao 1995. Applying update streams in a soft real time database systems, In Sigmod.

[9].B. Adelberg, B. Kao and H. Garcia-Molina 1996. Database support for efficiently maintaining derived data, In EDBT.

[10].L. Colby, T. Griffin, L. Libkin, I. Mumick and Trickey 1996. Algorithms for deferred view maintenance, In Sigmod

[11].M. Adiba and B. Lindsay 1980. Database snapshots, In VLDB.

[12].B. Lindsay, L. Haas, C. Mohan, H. Pirahesh and P. Wilms 1986. A snapshot differential refresh algorithm, In Sigmod.

[13].Crespo Márquez, P. Moreu de León, J.F. Gómez Fernández, C. Parra Márquez & V. González 2009. The maintenance management framework: A practical view to maintenance management, Safety, Reliability and Risk Analysis: Theory, Methods and Applications – Martorell et al. (eds) , Taylor & Francis Group, London.

[14].Dragan Sahpaski, Goran Velinov, Boro Jakimovski, Margita Kon-Popovska 2009. Dynamic Evolution and Improvement of Data Warehouse Design, Fourth Balkan Conference in Informatics, IEEE.

[15].Lobo, J.; Trajcevski, G. 1997. Minimal and consistent evolution in knowledge bases, Journal of Applied Non-Classical Logics, 7(1-2), 1997, pp. 117-146.

[16].Maabout, S. 1998. Maintaining and Restoring Database Consistency with Update Rules, Workshop DYNAMICS'98 (post-conf. Work. JICSLP'98), Manchester.

[17].Mayol, E.; Teniente, E. 1997. Structuring the Process of Integrity Maintenance, 8th International Conference on Database and Expert Systems Applications (DEXA'97), LNCS-1308, Toulouse, France, pp. 262-275.

[18].Teniente, E.; Olivé, A. 1995. Updating Knowledge Bases while Maintaining their Consistency, The VLDB Journal, Vol. 4, Num. 2, pp. 193-241.

[19].Thilini Ariyachandra, Hugh Watson 2010. Key organizational factors in data warehouse architecture selection, Elsevier.