Global Journal of Computer Science and Technology: C
Software & Data Engineering
Volume 18 Issue 3 Version 1.0 Year 2018
Type: Double Blind Peer Reviewed International Research Journal
Publisher: Global Journals
Online ISSN: 0975-4172 & Print ISSN: 0975-4350

Incremental Maintenance of a Materialized View in Data Warehousing: An Effective Approach

By Dr. Sanjay S Solank

JSPMs Abacus Institute of Computer Application

Abstract- A view is a derived relation defined in terms of base relations. A view can be materialized by storing its extent in the database. An index can be made of these views and access to materialized view is much faster that recomputing the view from scratch. A Data Warehouse stores large amount of information collected from a different data sources. In order to speed up query processing, warehouse usually contains a large number of materialized views. When the data sources are updated, the views need to be updated. The process of keeping view up to date called as materialize view maintenance. Accessing base relations for view maintenance can be difficult, because the relations may be being used by users. Therefore materialize view maintenance in data warehousing is an important issue. For these reasons, the issue of self-maintainability of the view is an important issue in data warehousing. In this paper we have shown that a materialized view can be maintained without accessing the view itself by materializing additional relations at the data warehouse site.

Keywords: optimized view, ETL, incremental maintenance, view maintenance process, DMWS, view synchronization, expression tree.

GJCST-C Classification: *H.2.7*

Incremental Maintenance of a Materialized View in Data Warehousing: An Effective Approach

Dr. Sanjay S Solank

Abstract- A view is a derived relation defined in terms of base relations. A view can be materialized by storing its extent in the database. An index can be made of these views and access to materialized view is much faster that recomputing the view from scratch. A Data Warehouse stores large amount of information collected from a different data sources. In order to speed up query processing, warehouse usually contains a large number of materialized views. When the data sources are updated, the views need to be updated. The process of keeping view up to date called as materialize view maintenance. Accessing base relations for view maintenance can be difficult, because the relations may be being used by users. Therefore materialize view maintenance in data warehousing is an important issue. For these reasons, the issue of self-maintainability of the view is an important issue in data warehousing. In this paper we have shown that a materialized view can be maintained without accessing the view itself by materializing additional relations at the data warehouse site. We have developed a cost effective approach to reduce the burden of view maintenance and also proved that proposed approach is optimum as compared to other approaches. Here incremental evaluation algorithm to compute changes to materialized views in relational is presented.

Keywords: optimized view, ETL, incremental maintenance, view maintenance process, DMWS, view synchronization, expression tree.

I. Introduction

It has been observed that in most typical data analysis and data mining applications, timeliness and interactivity are more important considerations than accuracy; thus, data analysts are often willing to overlook small inaccuracies in the answer, provided that the answer can be obtained fast enough. This observation has been the primary driving force behind the recent development of approximate query processing techniques for aggregation queries in traditional databases and decision support systems [4], [5]. Numerous approximate query processing techniques have been developed: The most popular ones are based on random sampling, where a small random sample of the rows of the database is drawn, the query is executed on this small sample, and the results are extrapolated to the whole database. In addition to simplicity of implementation, random sampling has the compelling advantage that, in addition to an estimate of the aggregate, one can also provide confidence intervals of the error, with high probability. Broadly, two types of sampling-based approaches have been investigated: 1) pre-computed samples, where a random sample is pre-computed by scanning the database and the same sample is reused for several queries and 2) online samples, where the sample is drawn "on the fly" upon encountering a query. So the selection of these random samples in distributed environments for query processing is addressed in [6]. Data warehouses (DW) [6] are built by gathering information from data sources and integrating it into one virtual repository customized to users' needs. One important task of a Data Warehouse Management System (DWMS) is to maintain the materialized view upon changes of the data sources, since frequent updates are common for most data sources. In addition, the requirements of a data source are likely to change during its life-cycle, which may force schema changes for the data source. A schema change could occur for numerous other reasons, including design errors, the addition of new functionalities and even new developments in the modeled application domain. Even in fairly standard business applications, rapid schema changes have been observed. In [10], significant changes (about 59% of attributes on the average) were reported for seven different applications over relational databases. A similar report can also be found in [15]. These applications ranged from project tracking, sales management, to government administration.

In situations that real-time refreshment of the data ware-house content is not critical; changes to the sources are usually buffered and propagated periodically such as once a day to refresh the view extent. Two benefits are possible. One is to gain better maintenance performance. The other is that there are less conflicts with DW read sessions. In a data update only environment, most view maintenance (VM) algorithms proposed in the literature [17, 1, 14] group the updates from the same relation and maintain such a large delta change in a batch fashion. However, these algorithms would fail whenever source schema changes occur, which are also common as stated above. One obvious reason is that the data updates in this group may be schema inconsistent with each other if there are some schema changes in between. On the other hand, work has begun on incorporating source schema changes into the data warehouse, namely, view synchronization (VS) [8] aims at rewriting the DW view definition when the source schema has been changed.

Author: Professor, JSPM's Abacus Institute of Computer Application, Pune. e-mail: sanjay.solanki123@gmail.com

To handle the delete of any schema information of a data source, VS tries to locate an alternative source for replacement to keep the new view semantically as close to the original view as possible. Thereafter, view adaptation (VA) [12] incrementally adapts the view extent to keep the new view consistent. Such algorithms are also not sufficient to batch a group of mixed data updates and schema changes, since there could be a number of schema changes interleaved with some data updates. In this paper, we propose a solution strategy that is capable of batching a mixture of both source data updates

II. Definition of Terms

View evaluation can be represented by a tree, called an expression tree[5,9]. An expression tree is a tree, where the leaf nodes represent base relations and non-leaf nodes represent binary expressions in the relational algebra. The unary relational algebraic expressions are associated along the edges. A view or a query is optimized by the query optimizer before executing it. A query optimizer takes an expression tree as input and produces an output, called an optimized expression tree, which determines the internal sequence of operations for executing a query. Thus, an optimized expression tree defines a partial order in which operations must be performed in order to produce the result of the view.

Depth: The depth of leaf nodes, that is data base relations is 0. The depth d of a node is defined as max(depth of descendents)+1.

Height: The height of the optimized expression tree is defined as the maximum depth of any node in the tree.

Given a node i in the expression tree, its parent is denoted by ↑i, and op(i) and op(↑i) are the expressions associated with i and i, respectively. The children of node i are denoted by i' and i'' where i' is a sibling of i'' and vice versa. IR_i denotes the intermediate result of node i. The auxiliary relation associated with node I is denoted AR_i in the case where only one relation is needed, and by AR^1i and AR^2i when two are needed. The key of IR_i is denoted by K_i, and the keys of $IR_{i'}$ and $IR_{i''}$ are denoted by $K_{i'}$ and $K_{i''}$, respectively. Insertion and deletion of tuples are denoted by △ and ▽ respectively. The symbol δ either an inserted set or a deleted set of tuples. The instance of a relation, say Ri, before and after an update is denoted by Ri^{old} and Ri^{new} respectively, similiary for an auxiliary relation AR and a materialized view V.

III. Example & Simplification

Consider a data warehouse for a large research organization which has got many departments and each department has many research groups. Suppose this data warehouse is collecting data from four base relations whose schemas are as follows:

R1: emp_rschr(rschr_id,rname,deptno,major) This relation gives the researchers id, name, department and major.

*R2:*emp_paperpublish(rschrid,paper_id,paper_title,source_of_publiscation, year_of_publish)

This gives researchers id,paper id, paper title, source of publication and year of publish.

R3: emp_manager(rschr_id,deptno)

This relation contain one record for each manager and his department. Assume that each department has one manager. Since a manager is also a researcher, relation emp_rschr has a tuple for each manager.

R4: emp_groupleader(rschr_id,deptno)

This relation contains information about th research group name and who is leading this group. Since a group leader is also a researcher, relation emp_rshcr has a tuple for each group leader.

Suppose a user of the organization is interested in materializing and maintaining the following view:

'Researchers other than managers and group leaders along with their departments who have published more than 10 papers in the year 2010.'

In SQL, it is defined as a sequence of view definitions:

Create view mngr_or_groupleader (rschr_id, deptno) as select rschr_id, deptno from emp_rschr

UNION

(select rschr_id, deptno from emp_groupleader)

// This view is for finding manager and group leader

Create view rschr_ex_ manager_or_groupleader (rschr_id, deptno) as select reshr_id, deptno from emp_rschr where NOT EXISTS (select *from mngr_or_grouple ader where emp_rschr.id=mngr_or_groupleader.id)

//This view is for finding researcher, those are not manager or group leader.

Create view rschrpaperview2010 (rschr_id, paper_id, deptno) as select emp_paperpublish.rschr_id, paper_id, deptno from rschr_ex_manager_or_group leader, emp_paperpublish where rschr_ex_ manager_or_ group leader.rschr_id=emp_paperpublish.rschr_id and year= '2010'.

//This view gives the researcher those who have published paper in the year 2010.

Create view rschrpaperview(rschr_id,deptno) as Select rschr_id, deptno from rschrpaper view2010 group by rschr_id having count(*)>10;

// This view gives the researcher who published more than 10 research paper in the year 2010.

As base relations are updated, changes representing the researchers data come into the warehouse. Most warehouse do not apply the changes immediately. Instead, changes are deferred and applied to the auxiliary relations incrementally. Deferring the changes allows analysts that query the warehouse to see a consistent snapshot of the data throughout the day, and can make the maintenance more efficient. Figure 1 shows the optimized expression tree for the above view. Here, the nodes at leaf level are base relations and non-leaf nodes are expressions. Each non-leaf node in the tree corresponds to a relational algebraic expression given above.

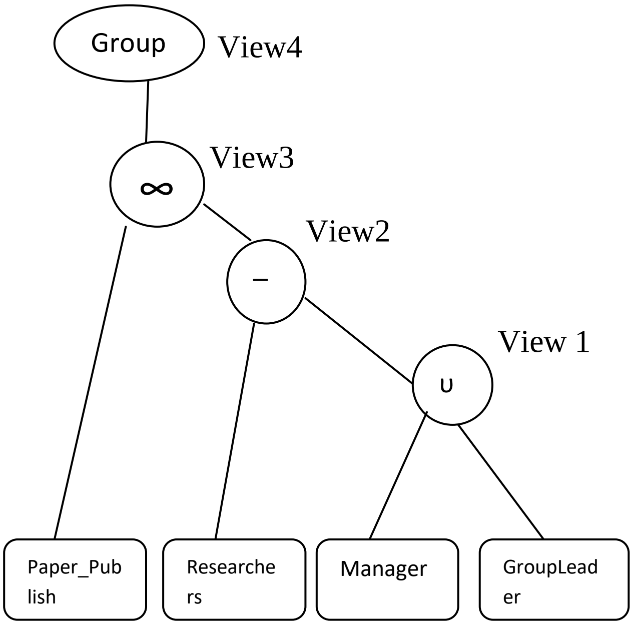

Figure 1: Expression tree

Suppose Researchers or Paper_Public relations are updated. In this case we materialize the two auxiliary relations View2 and View3. The contents of these views are derived while computing the view first time. By materializing these two auxiliary relations in the warehouse, the view is self-maintainable along with these auxiliary relations. Suppose new researchers joined the organization, therefore, one tuple for each new researcher in emp_rschr relation has to be inserted. These insertions will led to generate tuples that to be inserted in rschr_ex_manager_groupleader. Since these new researchers have not published any paper at the time of joining, these tuples cannot join with any tuples of emp_paper_publish, thus there will no change in the materialized views. Therefore, all auxiliary relations and materialized views are self_maintainable. Now consider another case where a set of tuples is inserted in emp_paper_publish relation, say R. Then, we first compute the research paper those are published in year 2010 and then it is join with rschr_ex_managergroupleader view. Lastly the intermediate result is grouped in the final auxiliary relation by performing count operation. In this case also, the view and auxiliary relations are self-maintainable.

IV. Procedure of Materialize Views Maintenance

The materialize view maintenance process can be divided into two functions: 1. Propagate and 2. Refresh. The work of computing the auxiliary relations happens within the propagate function, which can take place without locking materialize views so that the warehouse can continue to be made available for querying by analysts. Materialize views are not locked until the refresh function, during which time the materialize views are updated from the auxiliary relations.

The propagate function involves updating the auxiliary views incrementally from deferred set of changes. The final auxiliary view represents the net changes to the materialize views due to the changes in the underlying data sources.

The refresh function applies the net changes represented in the final auxiliary relation to the materialize views. This process carried out after a specific time interval or when the system has free cycles. So none of the data warehouse users or operations are affected by the view maintenance process. None of the query has to pay for view maintenance. The materialize view maintenance process totally hidden by users and running transactions. Whenever an interested change happens in the underlying data source, simply this desire change is stored in the auxiliary relations by comparing and joining it with others relations if required. This change is passed to the higher level auxiliary relations. Again the change is integrated and circulated to final auxiliary relation. Lastly the change is refreshed into the data warehouse when the refresh trigger is occur.

a) Analytical Cost Model

In this section we show the performance results of our materialize view maintenance method. The results are based on the following cost model.

i. *Cost Model*

The overall view maintenance cost of materialized views includes the cost of propagate the changes and the cost of refresh operations. Let $V_1,V_2,\ldots,V_m$ be the m materialized views. Let $B_1, B_2,\ldots,B_n$ be the n base relations and A_1,A2,…A_i be the i auxiliary relations. Let f_{u1}^{B1} ,…..,f_{un}^{Bn} be the update frequency to the base relations. Let $C^{ij}_{B\text{->}A}$ be the cost of propagating an update on base relation B_i to auxiliary relation A_j and $C^{jk}_{A\text{->}V}$ be the cost of refresh of auxiliary A_j to materialized view V_k. The overall cost of maintaining the views when keeping both the materialized views and the auxiliary relations is:

$$C_{MV}+AR=\sum_{i=1}^{i=n}(\mathrm{f}_{ui}^{Bi})*(\sum_{j=1}^{j=1}\mathrm{C}\text{–>}\,A\binom{n}{k}x^k a^{n-k}{}^{k}\ \ a^{n-k}$$

The total view maintenance cost with no auxiliary relations is:

$$C_{MV}=\sum_{i=1}^{i=n}(\mathrm{f}_{ui}^{Bi})*(\textstyle\sum_{k=1}^{i=n}\mathrm{C}$$

It is obvious that the cost of maintaining the materialized views directly from base relations is much more than the cost of maintaining materialized views through auxiliary relations.

V. Evaluation

To verify the feasibility and effectiveness of our view maintenance strategies and corresponding optimization framework, we have implemented the proposed techniques using Oracle 9i. All experiments were performed on a workstation with Pentium D 3.2 GHz processor, 1 GB of memory and 160 GB disks, running Windows XP.

Relation R1 contain 500000 records, R2 contains 25000 records, where as in R3 there are records of individual manager of a department and in R4 holds the records of group leaders.

We considered two types of changes:

Update-Generating changes: Insertions and deletions of an equal number of tuples over existing researchers and paper publishers. These changes mostly cause updates amongst the existing tuples in materialized view.

Insertion-Generating changes: Insertions over new researchers those who published certain number of research papers. These changes cause only insert into paper publish table.

The insertion-generating changes are very meaningful since in many data warehousing applications the only changes to the fact tables are insertions of tuples for new dates, which leads to insertions into materialized views.

Figure 2 shows four graphs illustrating the performance advantage of using incremental materialized view maintenance method which uses auxiliary views to store intermediate results. The view maintenance time is split into two functions propogate and refresh. While computing the intermediate result the data warehouse is remain free to the user.

Figure 2 (a) and (b) plot the variation in elapsed time as the size of the change set changes(delta relation), for a fixed size 500000 records in emp_rschr relation and 250000 records in emp_paperpublish relation.

We found that the incremental materialize view maintenance using auxiliary relations wins for both types of changes, but it wins with a greater margin for the update generating changes. The refresh time is going down by 20% in figure 2(b).

Figure 2(c) and (d) plot the variation in elapsed time as the size of the emp_paperpublish relation (source relation) changes, for a fixed size of 50000 records in change set(delta relation).

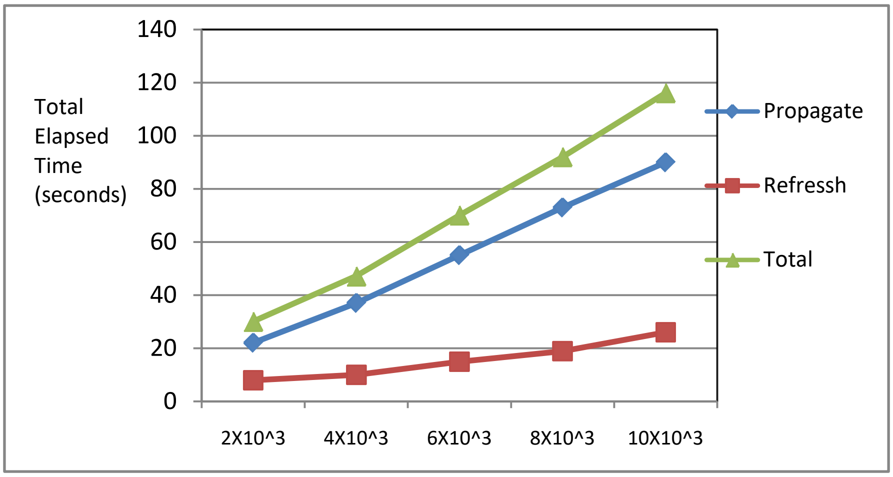

Figure 2 (a): Varying change set size for insert generating changes

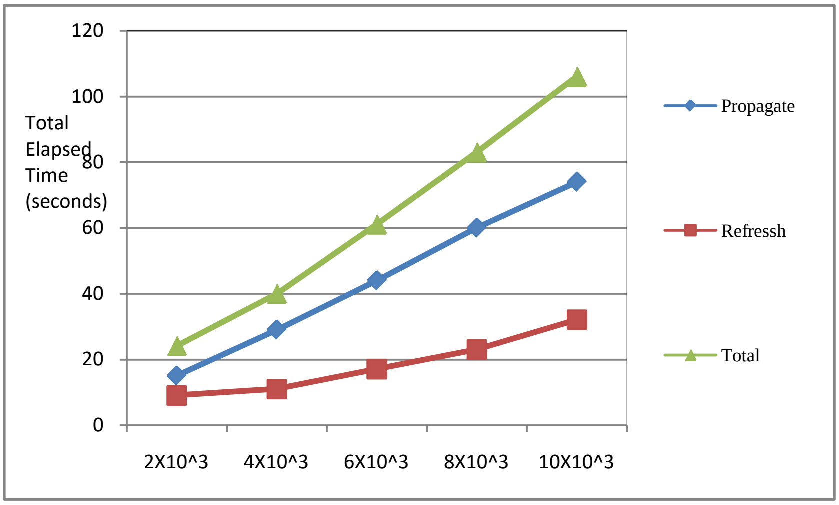

Figure 2(b): Varying change set size for update generating changes

VI. Conclusions

We have investigated one of the significant problems of a data warehouse, that is, materialized view maintenance and how to make warehouse materialized views self maintainable without accessing the data from underlying data sources. The study shows that it is possible to make warehouse views self maintainable by materializing additional auxiliary relations, which contain intermediate results, at a data warehouse site. Using efficient incremental materialize view maintenance technique it is possible to reduce the cost of view maintenance. Proposed materialize view maintenance technique using auxiliary relation and dividing the maintenance process into two steps: propagate and refresh require less maintenance time as compared to counting algorithm. Here the propagate function works implicitly and whenever the data warehouse is ideal the refresh function integrate the data into data warehouse views. The entire maintenance process is hidden from the data warehouse users.

References Références Referencias

1. A Segev and J. Park, "Maintaining Materialised Views in Distributed databases", In Proceedings of the IEEE International Conference on Data Engineering, 1989.
2. Segev and W. Fang," Currency based updates to distributed materialized Views", In proceedings of the IEEE International Conference on Data Engineering, 1990.
3. Abdulaziz S. Almazyad & Mohammad Khubeb Siddiqui," Incremental View Maintenance: An Algorithmic Approach", Internatioinal Journal of Electrical & Computer Sciences IJECS-IJENS Vol. 10, No. 03, 2009.
4. Bin Liu & Elke A. Rundensteiner, "Optimizing Cyclic Join View Maintenance over Distributed Data Sources", IEEE Transactions on Knowledge and Data Engineering, Vol. 18, No. 3, March 2006.
5. D. Agarwal, A. E. Abbadi, A. Singh and T. Yurek, "Efficient View Maintenance at Data Warehouses", Proc. ACM SIGMOD, pp. 417-427, 1997.
6. D. Lomet and J. Widom," Special Issue on Materialized Views and Data Warehousing", IEEE Data Engineering Bulletin 18(2), June 1995
7. E. N. Hanson, "A performance analysis of view materialization strategies", In SIGMOD pages 440-453, 1987.
8. GianLuca Moro and Claudio Sartori," Incremental View Maintenance on Multi-Source", In proceedings of IEEE, 2001.
9. Gray C. H. Yeung and William A. Gruver, "Multi agent Immediate Incremental View Maintenance for Data
10. Warehouses", IEEE Transaction on Systems, Man & Cybermetics- Part A: Systems & Human, Vol. 35, No. 2, March 2005.
11. Hao He, Junyi Xie., Jun Yang, Hai Yu," Asymmetric Batch Incremental View Maintenance", In the Proceedings of the 21st International Conference on Data Engineering, 1084-4627/05, 2005.
12. Heney E. Korth and Abraham Silberschatz, "Database System Concepts", McGraw Hill, 1986.
13. J. A. Blakeley, P.A. Larson and F. W. Tompa, "Efficient Updating Materialized Views", Proc. ACM SIGMOD, pp. 61-71, May 1986.
14. J. Chen, X. Zhang, S. Chen, K. Andreas and E. A. Rundensteiner, "DyDa: Data Warehouse Maintenance under Fully Concurrent Environments", Proc. ACM SIGMOD Demo Session, p.619, 2001.
15. J. Hammer, H. Garcia-Molina, J. Widom, W. Labio & Zhuge, "The Stanford Data Ware housing Project", IEEE Data Engineering Bulletin, June1995.
16. Jingren Zhou, PerAke Larson and Hicham G. Elmongui: Lazy Maintenance of Materialized Views",

in Proceddings of 33rd International conference on VLDB 2007, Vienna, Austria.

17. L. S. Colby, A. Kawaguchi, D. F. Lieuwen, I. S. Mumick and K. A. Ross, “Supporting Multiple View Maintenance Policies”, In Proceeding ACM SIGMOD International Conference on Management of Data, 1977.
18. Latha S. Colby, Timothy Griffin, Leonid Libkin, Inderpal Singh Mumick and Howard Tricky, “Algorithms for Defered View Maintenance”, In proceedings of ACM SIGMOD, 1996, Canada.
19. M. Adiba & B. Line\dsay, “Database Snapshots, ”In Proceedings of the sixth International Conference on Very Large Databases, pages 86-91, Montreal, Canada, October 1980.
20. M. Mohnia, “Avoiding re-computation: View Adaptation in Data Warehouses”, In Proc. Of 8th International Database Workshop, Hong Kong, pages 151-165, 1997.
21. N. Hyun, “Efficient View Self-Maintenance”, Proceeding of ACM workshop on Materialized views: Techniques & Applications”, Canada, June 7, 1996.
22. N. Roussopoulos, “An Incremental Access Method for Viewcache: Concept, Algorithms and Cost Analysis”, ACM Trans. On Database Systems, 16(3):535-563, 1991.
23. O. Wolfson, H. M. Dewan, S. J. Stolfo and Yemini, “Incremental Evaluation of Rules & Its Relationship to Parallesim”, In Proceedings ACM IGMOD, International Conference on Management of Data, pages 78-87, 1991.
24. R. Hull & G. Zhou, “A framework for supporting data integration using the materialized & virtual approaches”, In SIGMOD Int’l Conference, Canada, June 4 -6,1996.
25. R. Ramakrishan, K. A. Ross, D. Srivastava and S. Sudarshan, “Efficient Incremental Evaluation of Queries with Aggregation”, In International Logic Programming Symposium 1994.
26. S. Chaudhuri and U. Dayal, “An Overview of Data Warehousing and OLAP Technology”, In ACM SIGMOD Record, volume26, pages 65-74, 1974.
27. S. Chen, B. Liu and E.A. Rundensteiner, “Multiversion Based View Maintenance over Distributed Data Sources”, ACM Trans. Database Systems (TODS), vol.29, no. 4, pp. 675-709, 2004.
28. S. Solanki and Dr. Ajay Kumar, “A Comparative Study of Materialized View Maintenance Techniques in Data Warehousing”, IJRIME, Vol. 1, Issue 2, August 2011.
29. S. Solanki and Dr. Ajay Kumar, “A Comprehensive Study of Data Warehousing”, IJMR, Vol1, Issue 1, January 2012.