THE APPROACHES OF MATERIALIZE VIEW MAINTENANCE PROCESS IN DATA WAREHOUSING

Mr. Sanjay S. Solanki*

Dr. Ajay Kumar**

Abstract:

Materialized view maintenance is a significant issue due to the growing use of warehouse technology for information integration and data analysis. Materialized views are used to speed up query processing on large amounts of data. In large databases particularly in distributed database, query response time plays an important role as timely access to information and it is the basic requirement of successful business application. These views need to be maintained in response to updates in the source data.

Typically, a view is maintained immediately, as a part of the transaction that updates the base tables called as eager view maintenance. Immediate view maintenance imposes a significant overhead on update transaction that cannot be tolerated in many applications. A materialized view can be maintained lazily as maintenance is postponed until the system has free cycle or after a specific time period called a deferred view maintenance. Experiments using a prototype implementation in Oracle 9i show much faster response times for updates and also significant reduction in maintenance cost when combining updates.

Keywords: Data warehousing, View Maintenance, Auxiliary relation, Query cost, View manager.

* Asst. Prof., JSPM's Jayawant Institute of Management Studies, Tathawade, Pune-33.

** Director, JSPM's Jayawant Institute of Computer Applications,Tathawade, Pune-33.

1. Introduction:

Materialized views are important in data warehousing for fast retrieval of derived data regardless of the access paths and complexity of view definitions. When the underlying data sources are updated by insertion, deletion, or modification of tuples, a materialized view must also be updated to ensure the correctness of answers to queries against it. Updating the materialized view by full recomputation is often expensive and a more efficient technique can be to update the view incrementally. By this, we mean that the new view is computed from the existing view and the changes to the base relations.

Most database systems achieve this by eager view maintenance where all affected views are maintained as part of the update statement or the update transaction. Under eager maintenance, the view maintenance cost is born entirely by updates while the beneficiaries of the view get a free ride. View maintenance overhead can be quite high when multiple views require maintenance, resulting in poor response times for updates. Forcing updates to pay for maintenance seems rather unfair and may also be inefficient if there are many small updates. To address this situation, some database also support batch maintenance or deferred materialized view maintenance. In deferred view maintenance, maintenance is delayed and takes place only when explicitly triggered by a user. This is also called lazy view maintenance. This approach has the serious drawback that a query may see an out-of –date view and produce an incorrect result. Allowing the query optimizer to automatically use such views compromises corrections. The use of materialized views is no longer automatic and transparent to users. Query issuers have to know what views are used by a query, how they are maintained and whether they are or need to be, up to date at execution time.

We wanted a solution that both relieves the burden of view maintenance from updates and retains the property that queries always see up-to-date views. Under lazy view maintenance, updates do not maintain views but just store away enough information so that affected views can be maintained later. Actual view maintenance is done by low priority jobs running when the system has free cycles available. If the system has enough free cycles and a view is maintained before it is needed by queries, neither updates nor queries pay for view maintenance. If a view is not up-to-date when needed by a query, it is transparently brought up to date before the query is

allowed to access it. In this case the first beneficiary of the view pays for all part of the views maintenance by experiencing a delay. However, it only pays to maintenance of views that it uses and not for maintenance of other views affected by an update. Lazy maintenance allows updates to complete faster so locks are released sooner, which reduces the frequency of lock contention, lock conflicts and transaction aborts. This is particularly important for updates that affect highly-aggregated views because they tend to have higher rates of lock conflicts.

Maintenance task can also be reduced by merging maintenance tasks, which allows them to be processed more efficiently.

2. System Overview:

An overview of our design is explained in this section and also describes individual components in more detail. Figure 1 shows the overall system design for lazy materialization view maintenance. The wrapper monitor pair finds the interested changes across the data sources & sends it to integrator to integrate the changes into data warehouse.

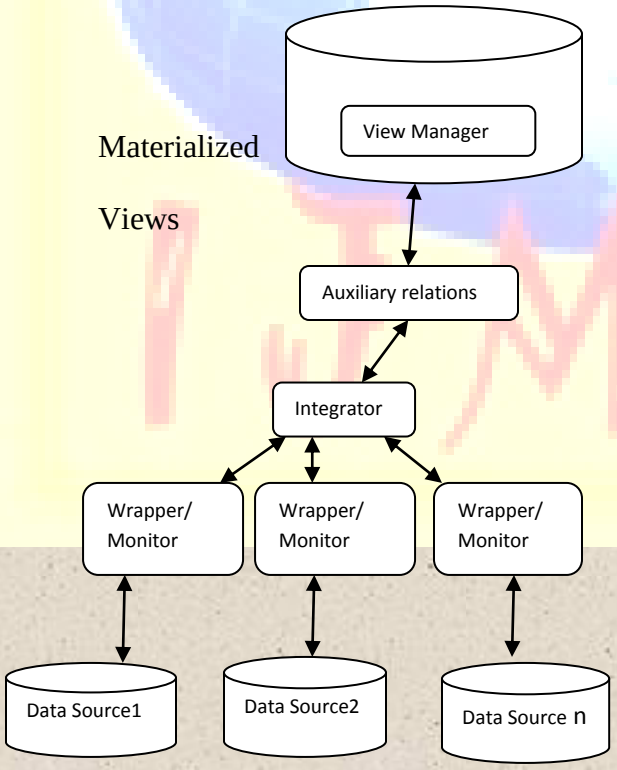


Figure 1 System Overview

Figure 1 shows system design for lazy view maintenance. We first describe the individual components and then explain the overall procedure.

Auxiliary relations: Execution of an insert, delete or update statement against a data source produces intermediate result. Obviously there are overheads incurred by maintenance of these auxiliary relations, but their use can often significantly reduce the cost of computing the updates to the materialized view. By maintaining these relations, a view can be self-maintained incrementally without recomputing intermediate results from scratch and the exact change to every intermediate step can be derived from them.

Maintenance Manager: This component keeps track of active view maintenance tasks and what delta changes are needed. It is also responsible for constructing view maintenance jobs and scheduling them.

To be able to quickly find all maintenance tasks for a given view, the manager maintains a hash table containing an entry for each materialized view with active maintenance tasks. Each entry has a linked list containing the maintenance tasks of the view. The list is sorted in an increasing order on count sequence number.

2.1 Update Transactions

Consider an update statement modifying a base relation S that is referenced by a number of materialized views. Eager maintenance updates all materialized views that reference S immediately after the update statement. In the case of lazy maintenance, view maintenance is skipped. Instead, enough information is saved so the affected views can be updated later. The intermediate result is stored in the auxiliary relations and after specific period the materialized views are refreshed.

An update transaction may contain multiple update statements. The transaction internally records which table is modified by which statement and which views are affected. Each update statement reports its own information at the end of its execution.

When the update transaction commits, maintenance tasks are constructed based on the information reported during execution. One maintenance task is generated per affected

materialized view. The tasks are then passed on to the maintenance manager and also written to the persistent task table. If the update transaction aborts, no information is saved and no maintenance tasks are constructed.

2.2 Lazy Maintenance

The view manager as shown in figure1 wakes up after specific time. If there are no pending maintenance tasks or the system is currently busy, it goes back to sleep. Otherwise, it decides what views to maintain and for each view, construct a low-priority back ground maintenance job and schedule it. Maintenance jobs for the same view are always executed in the commit order of the originating transaction.

The maintenance manager may combine multiple maintenance tasks for the same view into a larger job that can be executed more efficiently. During maintenance delta changes from auxiliary relations are used. When a maintenance job completes, it reports back to the maintenance manager. The manager then removes the completed tasks from its task list and releases any tuples from auxiliary relations.

2.3 Effect on Response Time

Lazy view maintenance is completely transparent to applications. Applications exploit materialized views in the same way as before and always see a state that is transactionally consistent with base tables. The only difference is in response time of updates and queries, which is the topic of this section. Suppose we have three updates followed by a query. All three updates affect a materialized view that is used by the query.

Under eager maintenance, each update has to wait until maintenance is done. If the affected views are expensive to maintain, update response times may be very slow. When the query arrives, the updates have completed and the view content is up to date so the query completes quickly.

Under lazy maintenance, the response time of the updates is much improved. Suppose the system gets a chance to maintain the affected views after the three updates. By combining the three updates, the total time spent on maintaining the views is reduced. If the query arrives after

lazy maintenance is done, its response time is the same as the eager maintenance. If the query arrives in the middle of lazy maintenance of a view that it needs, it is forced to wait until maintenance of that view is finished. Finally, if the query arrives immediately after updates and before the system has begun maintenance of the view, the query issues a on-demand maintenance request at the beginning and waits until it is finished. The total system response time for all the updates and the query is still improved over eager maintenance.

The total cost for materializing views can computed using the following strategy. The cost contains query processing cost (for selection, aggregation and joining), view maintenance (refresh view) cost. The cost is calculated in terms of block size B. The query processing cost in terms of block access is equal to size of materialized view Vi.

$$C_B(V_i)=S(V_i)$$

The query cost involving the joining of n dimensional tables with view Vi is given by:

$$C_j(V_{d1}, V_{d2},---, V_{dn}, V_i)= (S(V_{d1)} + S(V_{d1}) * (SV_i)) + (S(V_{d2)} + S(V_{d2})*S(V_i)) --- + (S(V_{dn}) +S(V_{dn})* S(V_i))$$

To process users query qi, which requires not only selection and aggregation of the view, but also the joining of view with other dimension tables, the query cost $C_q(qi)$ is given by:

$$C_q(V_i)= C_B(V_i)+ C_j(V_{d1}, V_{d2},---, V_{dn}, V_i)=$$

$$S(V_i)+ (S(V_{d1)} + S(V_{d1}) * (SV_i)) + (S(V_{d2)} + S(V_{d2})*S(V_i)) --- + (S(V_{dn}) +S(V_{dn})* S(V_i))$$

Thus the total Query cost Total(C_{qr}) for processing r user queries is given by

$$Total(C_{qr})= \sum_{i=1}^{r}(f_{qi}*C_q(q_i))$$

The re-computation of each view requires selection and aggregation from its ancestor view V_{ai} and their joining with n dimension tables. Therefore the maintenance cost is given by

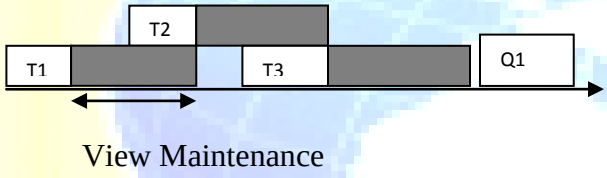
$C_m(Vi)= C_B(V_{ai}) + C_j((V_{d1}, V_{d2},---, V_{dn}, V_{ai}) = S(V_i)+ (S(V_{d1)} + S(V_{d1}) * (SV_{ai})) + (S(V_{d2)} + S(V_{d2})*S(V_{ai})) --- + (S(V_{dn}) +S(V_{dn})* S(V_{ai}))$

If there are j views which are materialized, the total maintenance cost Total(C_m) for these materialized views is given by:

$Total(C_m)= \sum_{i=1}^{j}(f_{ui}*C_m(V_i))$

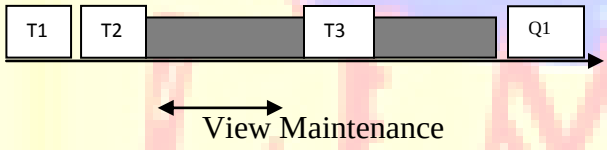
The total cost of query processing is the cost of query precessing & the cost of view maintenance

Total Cost(C_{total}) = Cost of Query Processing + Cost of Maintenance



View Maintenance Query

Figure 2 (a) Eager View Maintenance



View Maintenance Query

Figure 2 (b) Lazy View Maintenance

3. When to use Lazy Maintenance:

Each maintenance approach has its benefits and drawbacks and which approach is better for a particular view depends on the application. Generally, the choice of maintenance strategy for a materialized view depends on the following factors.

- The ratio of updates to queries and how soon queries follow after updates.

- The size of updates (number of rows affected by each update), relative to the view maintenance cost.

Eager maintenance is suitable for materialized views whose base tables are seldom updated and the updates are likely to be followed immediately by queries. It is also suitable for views where the input delta changes tend to be large but maintenance cost is relatively low. On the other hand, lazy maintenance is suitable for views with more frequent small updates and whose maintenance costs are relatively high.

4. Scheduling Maintenance Tasks:

Background scheduling:

Lazy maintenance can be triggered when the system has free cycles. In this case, the maintenance manager can freely choose which materialized view(s) to maintain. Scheduling of view maintenance has multiple, somewhat conflicting goals. First, it is desirable to hide view maintenance from queries as much as possible to improve query response time. Second, maintenance should be performed as efficiently as possible. Third, it is important to minimize the response consumed by pending maintenance tasks. Any scheduling policy represents a trade-off among these goals.

To hide view maintenance from queries, views could be assigned priorities based on how soon they are expected to be referenced by queries. The sooner the view is expected to be used, the sooner the view needs to be maintained. Future reference information can be estimated based on historical usage of the views.

If the view has multiple pending tasks, the manager must also decide whether and how many to combine into a single maintenance job. Combining tasks improves efficiency but could result in a long-running maintenance transaction. Other considerations may also be important when designing a scheduling policy. For example, we may know that a view is used only at a certain time of the day, for example, to produce reports. In that case, all that matters is that the view is brought up to date before that time.

Maintenance jobs run as low-priority background jobs but one could further reduce their impact on system resources. In case of a sudden burst in system workload, maintenance jobs can be paused or even aborted to avoid slowing down the system. We can also perform maintenance tasks in two phases. In first phase change computation will done and in second change will be applied to the materialized views.

On-demand Scheduling:

Lazy maintenance can also be triggered by a query. In this case, the views referenced by the query are maintained immediately. The maintenance manager must still decide whether and how to combine maintenance tasks. The maintenance jobs inherit the same priority as the query. A more interesting question is when it is possible to avoid maintaining a view even though it is referenced by a query. A view referenced by a query does not have to be brought up to date immediately if the pending updates do not affect the part of the view accessed by the query. It may be worthwhile to first check whether the pending maintenance tasks can cause a change in the view that is visible to the query. If not, the view does not have to be maintained immediately while still safely serving the query.

There are several ways to check. For example, we can project the query predicate onto each base table and scan the corresponding auxiliary tables with the projected predicate. If no scans return any tuples, we can safely deduce that the view content accessed by the query cannot be affected by the pending updates. This can be easily proven because it means that none of the m terms in the maintenance expression can produce a result affecting rows accessed by the query. However, this filtering operation can be expensive as maintaining the query.

In either scheduling mode, the maintenance manager schedules one job at a time for one view. This is achieved by monitoring the tasks status in the manager. There can be at most one task with the status of in progress for each materialized view. So that each transaction will preserve an ACID property.

5. Experimental Results:

To verify the feasibility and effectiveness of our view maintenance strategies and our corresponding materialized view optimization, we have implemented the proposed strategies in Oracle 9i. We deploy the view manager and the corresponding materialized view on a workstation with Pentium® 4 CPU 2.4 GHz processor, 1 GB of RAM and 160 GB disks, running Windows XP. In the experimental setting, there are four base relations(data sources) namely R1, R2, R3 and R4. The relation R1 contain 500000 records, R2 contains 250000 records, where in R3 there are 100 records and in R4 200 records. Total number of attributes are 24. A materialized join view is defined on all 24 attributes. Table 1 depicts

five different scenarios in which we update 100,200,300,400 & 500 records using a update statement.

Elapsed time required for Eager & Lazy maintenance approach		
No. of Updates	**Eager (seconds)**	**Lazy (seconds)**
1X10^2	105.3	17.7
2X10^2	156.5	21.5
3X10^2	171.2	23.2
4X10^2	179.3	25.8
5X10^2	188.2	28.9

Table 1 Time comparison for Eager & Lazy maintenance.

Figure 3 depicts the total view maintenance cost measured in seconds(y- axis) under different numbers of source data updates (x-axis).

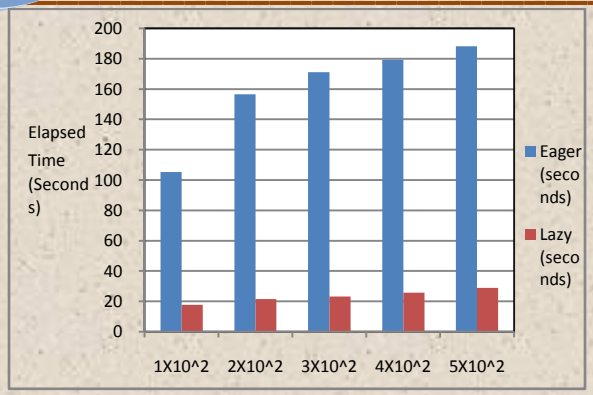


Figure 3 Elapsed time requirement of Eager & Lazy maintenance.

Figure 2 shows performance of lazy view maintenance over eager view maintenance. With lazy maintenance, update response time is reduced to virtually nothing. The system returns immediately after the records updated in the data warehouse. The eager method of view maintenance is expensive because each update requires holding lock on view to reflect the changes.

In eager maintenance, a query can exploit a view for free since it has already been maintained. However, update transaction are slowed down by view maintenance so they keep locks on the affected views longer, which may force queries and other updates to wait.

Under lazy maintenance, query response time depends on when the query arrives. Before execution begins, the query first checks with the maintenance manager if the requested view is up to date. If not, the query waits until all pending and in-progress maintenance of the view is completed.

6. Conclusions:

The materialized view is most beneficial for improving query performance as it stores precomputed data. We present the solution to materialized view maintenance in a mixed data update. Lazy maintenance can reduce update response time by orders of magnitude because updates no longer have to wait for views to be maintained. Eager maintenance is suitable for materialized views whose base tables are seldom updated and the updates are likely to be

followed immediately by queries. On the other hand, lazy maintenance is suitable for views with more frequent small updates and whose maintenance costs are relatively high. Under lazy maintenance the updates response time depends only on the cost of updating base relations and storing delta changes into auxiliary relation and not on the number and complexity of views affected. It allows updates to complete faster so locks are released sooner, which reduces the frequency of lock contention, lock conflicts and transaction aborts. A view is already maintained by each update transaction in eager maintenance so a query can exploit a view for free.

References:

- [BLT 86] J. A. Blakeley, P. Larson and F. Tompa,"Efficiently Updating Materialized Views", Proceeding of the ACM SIGMOD Conference, Washigton, 1986.
- [CJS 94] A. Courtney, W. Janseen, D. Severson, M. Spreitzer, and F. Wymore, "Inter-language unification, release 1.5", Technical report ISTL-CSA-94-01-01.
- [Cui 99] Y. Cui & J. Widom, "Storing Auxiliary Data for Efficient View Maintenance & Lineage Tracing", Technical Report, Stanford University,1999.
- [GL95] T. Griffin and L. Libkin, " Incremental Maintenance of a View with Duplictes", In SIGMOD, pages 328-339, May 1995.
- [GM95] A. Gupta and I. S. Mumick, "Maintenance of materialized views: Problems, technologies, and applications", IEEE Data Engineering Bulletin, Special Issue on Materialized Views and Data Warehousing, 18(2):3-18, June 1995.
- [Hammer 95] J. Hammer, H. Garcia-Molina, J. Widom, W. Labio & Zhuge, "The Stanford Data Warehousing Project", IEEE Data Engineering Bulletin, June1995.
- [Han87] E. N. Hanson, "A performance analysis of view materialization strategies", In SIGMOD pages 440-453, 1987.
- [Hull 96] 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.
- [Hyun 97a] N. Hyun, "Efficient View Self-Maintenance", Proceeding of ACM workshop on Materialized views:Techniques & Applications", Canada, June 7,1996.

- [JMS 95] H. V. Jagdish, I. S. Mumick & A. Silberschatz, "View Maintenance issues in the chronicle data model", In 14th Int'l conference PODS, 1995.
- [Jenn95] Jennifer Widom, "Research Problems in Data Warehousing", Proc. of 4th Int'l Conference on Information and Knowledge Management", Stanford, Nov 1995.
- [LGM96] W. Laboi and H. Garcia-Molina, "Efficient snapshot differential algorithms in data warehousing", In VLDB, pages 63-74, September 1996.
- [LW95] D. Lomet and J. Widom," Special Issue on Materialized Views and Data Warehousing", IEEE Data Engineering Bulletin 18(2), June 1995.
- [Quass 96] D. Quass, A. Gupta, I. S. Mumick, and J. Widom, "Making Views Self-Maintenable for Data Warehousing", Proceddings of the Conferecne on Parallel & Distributed Information Systems, Miami Beach, Fl, December 1996.
- [Silberschatz97] A. Silberschatz, H. F. Korth & S. Sudarshan,"Database System Concepts",3rd Edition, McGraw-Hill,1997.
- [TPC99] Transaction Processing Performance Council(TPC), "TPC Benchmark (Decision Support)", http:/www.tpc.org,1999.
- [Zhuge 95] Y. Zhuge, H. Garcia-Molina, J. Hammer & J. Widom, "Performance Analysis of WHIPS Incremental Maintenancet", Proceeding of the ACM SIGMOD Conference, San Jose, California, June 1995.
- [Zhuge 95] Y. Zhuge, H. Garcia-Molina, J. Hammer & J. Widom, "View Maintenance in a Data Warehousing Environment", Proceeding of the ACM SIGMOD Conference, San Jose, California, June 1995.
- [Zhuge 96] Y. Zhuge, H. Garcia-Molina, J. Hammer & J. Widom, "The Strobe algorithms for Multi-Source Warehousing Consistency", Proceeding of the Conference on Parallel & Distributed Information Systems, Milami Beach, Fl., December 1996.
- [Zhuge 97a] Y. Zhuge, H. Garcia-Molina,"Multiple View Consistency for Data Warehousing",Intl' Conference on Data Engineering, UK,April1997.