Chapter 3

Materialized View Maintenance Methods and Performance Evaluation

3.1 Introduction

3.2 Materialized View Maintenance System: An Overview

3.3 Materialized View Maintenance Methods

3.3.1 Eager (Immediate View Maintenance-IVM)

3.3.2 Lazy (Deferred View Maintenance-DVM)

3.4 Effect on Response Time by Eager (or IVM) and Lazy (or DVM) Approaches

3.5 Cost of Materialized View Maintenance

3.6 Scheduling Maintenance Tasks

3.6.1 Background Scheduling

3.6.2 On Demand Scheduling

3.7 Experimental Performance Evaluation of Eager (or IVM) and Lazy (or DVM) Methods

3.8 Differentiation of Eager and Lazy Methods

3.9 Conclusions

Chapter Summary
3.1 Introduction

The interest in materialized views has increased in recent years, primarily due to the expanding range of their applications [3][20][62][117]. Materialized view maintenance is a significant issue due to growing use of warehouse technology for information integration and data analysis. It is used to speed up query processing for large amounts of data. Materialized views and view maintenance are important for data warehouse, retailing and billing applications [87]. In large databases particularly in distributed database, the 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.

The process of materialized view maintenance may involve several steps [20][21], one of which is refresh which brings the view table up-to-date. Also, there are other steps involved in the process of maintaining a view. For example, it may be necessary to maintain table that store the history of updates to the base tables. View maintenance techniques depend on when the view is refreshed. A view can be refreshed within the transaction that updates the base tables, or the refresh can be delayed. The former case is referred to as immediate view maintenance, while the latter is called deferred view maintenance. Deferred maintenance may be done periodically or on-demand when certain conditions arise. In the past, the term deferred maintenance had sometimes been used for on-demand maintenance.

In some applications, the immediate view maintenance is simply not possible because, in a data warehousing system, if a component database does not know what views exist at the warehouse, it cannot modify transactions updating base tables so that they also refresh materialized views. Even in a centralized system, where all the views are known, it may be necessary to minimize the per-transaction overhead imposed by view maintenance. In such deferred maintenance is most appropriate. Other applications may have a certain tolerance for out-of-date data, or even require that the view be frozen for analysis and other functions. In this case, the view could be refreshed periodically or just before querying. The deferred maintenance also allows several updates to be batched together. Most of the work on
view maintenance has involved the immediate case [67],[119],[120]. Several incremental view maintenance algorithms for immediate maintenance have been proposed. These algorithms are based on the assumption that access to the pre-update base tables is available.

3.2 Materialized View Maintenance System: An Overview

An overview of the materialized view maintenance system consists of view manager, auxiliary relations, integrator, wrapper/monitor and data sources (Figure 3.1). Wrapper/monitor pair finds the interested changes across the data sources and sends these changes towards the integrator. After getting changes the integrator component integrates the changes into the data warehouse through auxiliary relations. At the data warehouse, there is a view manager which takes care of appropriate view maintenance.

![Materialized View Maintenance System](image-url)

**Figure 3.1** Materialized View Maintenance System
Materialized View Maintenance (MVM) process with the help of maintenance manager is as shown in Figure 3.1. The functions of individual component of a MVM process are explained below:

- **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.

- **View 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. This component also responsible for maintenance tasks. The component schedules the maintenance tasks and takes care of consistency. So that after refreshing the views the views in the consistent state as they were before refresh operation.

- **Data Staging Area:** After data is extracted from various operational systems and from external sources, the data should be prepared for storing in the data warehouse. The external data coming from several disparate sources needs to be changed, converted, and made ready in a format that is suitable to be stored for querying and analysis. Three major functions need to be performed for getting the data ready. The data is extracted, transformed and loaded into data warehouse.
- **Data Sources:** The bottom of the diagram (Figure 3.1) shows the data sources. These are the operational systems or external sources which function is to provide a data to the data warehouse. Their data model may be different.

The view manager as shown in Figure 3.1 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 the maintenance delta changes (intermediate result) 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.

### 3.3 Materialized View Maintenance Methods

Materialized views transparently pre-compute joins and aggregations and, when applicable, can reduce query execution time greatly. To ensure a correct result, a materialized view must be up to date whenever it is accessed by a query. Most database systems achieve this by **eager maintenance** where all affected views are maintained as part of the update statement or the update transaction. This may be called as **immediate maintenance approach** [6][87]. Some database systems also support deferred maintenance, where maintenance of a view is delayed and takes place only when explicitly triggered by a user. This approach is called as **deferred or lazy view maintenance**. The immediate approach has the limitation that each update transaction incurs the overhead of updating the view. The overhead increases with the number of views and their complexity.
3.3.1 Eager (Immediate View Maintenance- IVM)

In eager view maintenance approach, all views which require changes are maintained as a part of update transaction. Each update transaction, after updating the data source update the dependent view or views. The cost of maintaining a view 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. Data warehouse users may get fresh data by using this approach and do precise analysis.

Forcing updates to pay for view maintenance seems rather unfair and may also be inefficient if there are many small updates. Consider an update statement modifying a base table ‘R’ that is referenced by a number of materialized views. Eager maintenance updates all materialized views that reference ‘R’ immediately after the update statement. 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, it is not suitable for views with more frequent small updates and whose maintenance costs are relatively high. It cannot be triggered when the system has free cycles. A view is already maintained by each update transaction in eager maintenance so a query can exploit a view for free.

3.3.2 Lazy (Deferred View Maintenance- DVM)

In eager materialize view maintenance approach, the cost of maintaining a view is born entirely by updates while the beneficiaries of the view get a free ride. Facing updates to pay for view maintenance seems rather unfair and may also be inefficient if there are many small updates. To address this situation, some database systems also support deferred maintenance where maintenance of a view is delayed and takes place only when explicitly triggered by a user. This approach has a 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 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 the execution time.

There is a need that both relieves the burden of view maintenance from updates and retains the property that queries always see up-to-date views. Lazy view maintenance or deferred view maintenance achieves this [47][51][141].

Under lazy maintenance, updates do not maintain views but just store away enough information so that affected views can be maintained later. Actual 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 or the view pays for all part of the view’s 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 or deferred 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.

3.4 Effect on Response Time by Eager (or IVM) and Lazy (or DVM) Method

Consider an update statement which modifies a base relation ‘S’ and it 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 that 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 the execution. One maintenance task is generated per affected materialized view. The tasks are then passed on to the view manager and also written to the persistent task table. If the update transaction aborts, no information is saved and no maintenance tasks are constructed.

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 transnationally consistent with base tables. The only difference is in response time of updates transaction. 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 [142]. For instance 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 is. 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 an 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.

In short all the maintenance queries are batched and sent to the maintenance manager for view maintenance. Till batching the user queries can enjoy the materialized views. When the view manager wakes up, it finds the batched job and according to data, it integrate into the data warehouse views. So using lazy (IVM) the latest data may be little bit late available to data warehouse users. But it drastically reduces the maintenance task.
3.5 Cost of Materialized View Maintenance

The total cost for materializing views can be 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 \( B \) \[105][106][122]. The query processing cost in terms of block access is equal to size of materialized view \( V_i \).

\[
C_B(V_i) = S(V_i) \quad \text{--------- Equ. 3.1}
\]

The query cost involving the joining of \( n \) dimensional tables with view \( V_i \) is given by:

\[
C_j(V_{d1}, V_{d2}, \ldots, V_{dn}, V_i) = (S(V_{d1}) + S(V_{d1}) \times (SV_i)) + (S(V_{d2}) + S(V_{d2}) \times S(V_i)) - \ldots + (S(V_{dn}) + S(V_{dn}) \times S(V_i))
\]

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

\[
C_q(V_i) = C_B(V_i) + C_j(V_{d1}, V_{d2}, \ldots, V_{dn}, V_i) = S(V_i) + (S(V_{d1}) + S(V_{d1}) \times (SV_i)) + (S(V_{d2}) + S(V_{d2}) \times S(V_i)) - \ldots + (S(V_{dn}) + S(V_{dn}) \times S(V_i))
\]
Thus, the total Query cost \( \text{Total}(C_{qr}) \) for processing \( r \) user queries is given by

\[
\text{Total}(C_{qr}) = \sum_{i=1}^{r} (f_{qi} \times C_q(q_i)) \quad \text{--------Equ. 3.2}
\]

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

\[
C_m(V_i) = C_B(V_{ai}) + C_j((V_{d1}, V_{d2}, \ldots, V_{dn}, V_{ai}) = S(V_i) + (S(V_{d1}) + S(V_{d1}) \times (S(V_{ai})) + (S(V_{d2}) + S(V_{d2}) \times S(V_{ai})) \ldots + (S(V_{dn}) + S(V_{dn}) \times S(V_{ai}))
\]

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

\[
\text{Total}(C_m) = \sum_{i=1}^{j} (f_{ui} \times C_m(V_i)) \quad \text{--------Equ. 3.3}
\]

The total cost of query processing is the cost of query processing and the cost of view maintenance

\[\text{Total Cost}(C_{total}) = \text{Cost of Query Processing} + \text{Cost of Maintenance}\]

The net benefit and the storage effectiveness can be calculated to determine an optimal set of materialized views. The net benefit of materializing view calculated is as follows [165][168][169].

Net Benefit = Benefit – Maintenance cost – Storage cost

\[\text{(Bi)} = \sum_{i=1}^{j} (f_{qi}(V_{ni})) \times [C_t(V_{ni} < V_{ai}) - C_t(V_{ni} < V_i)] \quad \text{--------Equ. 3.4}\]

The storage effectiveness of views is given by \( n_i = \text{Net}(B_i) \times S(V_i) \).

Consider \( \text{Total}(C_{all}) \) is the total cost for processing users queries when no views are materialized in the data warehouse. When the materialized views are used then total cost is given by
3.6 Scheduling Maintenance Tasks

Data warehouse system requires regular maintenance to provide a steady performance, just like every car requires maintenance for maintaining the efficiency. The best way to ensure that everything gets done when it should be is to schedule maintenance tasks using background scheduling or on-demand scheduling\[87\][142][8][145]. The background scheduling can be hidden from the user and application. The application gets the free advantage of using view under this scheduling.

3.6.1 Background Scheduling

Lazy or DVM 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 the 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 improve 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. During the first phase, the exact change is computed and in the second phase the calculated change is applied to the materialized views.

3.6.2 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 and the queries can be served safely.

There are several ways to check whether a given view is referenced by a query. One way is to 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 view.

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 preserves an ACID property.
3.7 Experimental Performance Evaluation of Eager (IVM) and Lazy (DVM) Methods

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 Intel® Pentium® D CPU 2.66 GHz processor, 2GB 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 is 24. A materialized join view is defined on all 24 attributes. Table 3.1 depicts five different scenarios in which we update 1x10^5, 2x10^5, 3x10^5, 4x10^5 and 5x10^5 records using an update statement on a single view. A simulator is used to generate data for different data sources for evaluation purpose. Random data is generated and placed into the relations R1,R2,R3 and R4.

Table 3.1 Time comparison for Eager (IVM) and Lazy (DVM) maintenance

<table>
<thead>
<tr>
<th>No. of Updates</th>
<th>Eager (IVM) (seconds)</th>
<th>Lazy (DVM) (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 x 10^5</td>
<td>67</td>
<td>31</td>
</tr>
<tr>
<td>2 x 10^5</td>
<td>96</td>
<td>57</td>
</tr>
<tr>
<td>3 x 10^5</td>
<td>121</td>
<td>79</td>
</tr>
<tr>
<td>4 x 10^5</td>
<td>149</td>
<td>91</td>
</tr>
<tr>
<td>5 x 10^5</td>
<td>176</td>
<td>113</td>
</tr>
</tbody>
</table>

The total view maintenance cost measured in seconds (y-axis) under different numbers of source data updates (x-axis) depicted in Figure 3.3.
Elapsed time requirement of lazy view maintenance over eager view maintenance is better. With deferred view maintenance, maintenance time reduced nearly by 41% indicated in Figure 3.3. The immediate view maintenance is expensive because each update requires holding of lock on view to reflect the changed data. The eager method of view maintenance is expensive because each update requires holding lock on view to reflect the changes. In some applications, immediate or eager view maintenance is simply not possible. The reason is, in a data warehousing system, if a component database does not know what views exist at the warehouse, it cannot modify transactions updating base tables so that they also refresh materialized views. Even in a centralized system where all the views are known, it may be necessary to minimize the pre-transaction overhead imposed by view maintenance. In such cases, deferred or lazy view maintenance is most appropriate.

Other applications may have a certain tolerance for out-of-data, or even require that the view be frozen for analysis and other functions. In this case, the view could be refreshed periodically or just before querying. Deferred of lazy view maintenance approach.

![Figure 3.3 Elapsed time requirement of Eager (IVM) and Lazy (DVM) maintenance approach](image)
maintenance also allows several updates to be batched. If more than one view is affected by the updates, we defined two different views say V1 and V2 that pre-computed information from above four relations. We repeated the above experiment with the same number of updates on these two views.

Table 3.2  Time comparison for Eager (IVM) and Lazy (DVM) maintenance for two different views

<table>
<thead>
<tr>
<th>No. of Updates</th>
<th>Eager (DVM) (seconds)</th>
<th>Lazy (IVM) (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 x 10^5</td>
<td>129</td>
<td>73</td>
</tr>
<tr>
<td>2 x 10^5</td>
<td>171</td>
<td>91</td>
</tr>
<tr>
<td>3 x 10^5</td>
<td>207</td>
<td>123</td>
</tr>
<tr>
<td>4 x 10^5</td>
<td>236</td>
<td>162</td>
</tr>
<tr>
<td>5 x 10^5</td>
<td>278</td>
<td>198</td>
</tr>
</tbody>
</table>

Figure 3.4  Elapsed time for two views

Similar to the previous experiment, the update response time under lazy maintenance is almost nothing (Figure 3.4).

103
Chapter 3

The update response time for eager maintenance increased significantly because additional view maintenance has to be done as part of the update transaction. Under lazy maintenance the update response time is virtually unchanged by addition of a second view.

In eager maintenance, a query can utilize a view for free since it has already maintained before the query arrives. 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.

We issued a 500 update on the R1 relation followed by a query on view. In the eager maintenance case, we issued the query immediately after the update statement returned. In the lazy maintenance cases, we issued the query 20, 40, 60 seconds after the update statement. The table 3.3 shows these scenarios.

Table 3.3 Effect on query response time after delay

<table>
<thead>
<tr>
<th>Update time</th>
<th>Query delay time</th>
<th>Query response time</th>
</tr>
</thead>
<tbody>
<tr>
<td>70</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>70</td>
<td>20</td>
<td>74</td>
</tr>
<tr>
<td>70</td>
<td>40</td>
<td>38</td>
</tr>
<tr>
<td>70</td>
<td>60</td>
<td>5</td>
</tr>
</tbody>
</table>
Figure 3.5  Query response time

The query response time is compared and illustrated in Figure 3.5 and 3.6 for different situation. The update response time for lazy maintenance is too small to be visible in the figure. When the query is issued after lazy maintenance has completed, for example 60 seconds after the update in the figure, neither the update nor the query pay the cost for maintaining the view. The maintenance cost is hidden from the application.

Figure 3.6  System response time after delaying query
When queries are much more frequent than updates, only the first query after an update may be delayed; the other arriving later still use the view for free.

3.8 Differentiation of Eager and Lazy Methods

Each maintenance approach has its own benefits and drawbacks. Which view maintenance approach is better for a particular view is dependent upon the type of 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 and
- 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.

3.9 Conclusions

The materialized view is most beneficial for improving query performance as it stores pre-computed data. We present the solution to materialized view maintenance in a mixed data update. Lazy (or DVM) maintenance can reduce update response time by orders of magnitude because updates no longer have to wait for views to be maintained. Eager (or IVM) 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. Our experiments show the effective use of deferred (lazy) view maintenance technique. View maintenance overhead can be significantly reduced by combining multiple updates together. The view maintenance cost is hidden from the application.

Chapter Summary

Detail maintenance systems overview along with components involved in the maintenance is described in this chapter. Also the approaches of the materialized view maintenance are described in depth. It is shown that how different materialized view maintenance approaches effect on response time. The experimental result shows the effective implementation of eager (Immediate) and deferred (Lazy) view maintenance. The analytical model is derived to calculate the total materialize view maintenance cost. Different types of materialize view maintenance scheduling is explained to select the appropriate maintenance strategy. Lastly the experimental performance results are given to compare both the type of view maintenance in data warehousing.