Chapter 4

Analytical Models for Materialized View Maintenance Methods

4.1 Introduction

4.2 Materialized Views in Data Warehouse

4.3 Materialized View Maintenance in Data Warehouse

4.4 Classification and Implementation of Materialized View Maintenance Methods

4.4.1 Recomputational View Maintenance (RVM)

4.4.1.1 Self Maintainable Recomputation (SMR)

4.4.1.2 Not Self Maintainable Recomputation (NSMR)

4.4.2 Incremental View Maintenance (IVM)

4.4.2.1 Self Maintainable Incremental Maintenance (SMIM)

4.4.2.2 Not Self Maintainable Incremental Maintenance (NSIM)

4.5 Unrestricted Base Access

4.6 Runtime Warehouse Self-maintenance

4.7 Performance Evaluation of RVM and IVM

4.7.1 Evaluation Parameters

4.7.2 Evaluation

4.8 Advantages and Disadvantages of View Maintenance Methods

4.9 Conclusions

Chapter Summary
4.1 Introduction

Information is priceless, but making it useful and presentable is difficult. Many organizations currently face the challenge in trying to integrate mass amount of data to make it accessible. IT professionals face sizable challenges to support those applications as they continue to grow in both their number and sophistication [12],[32]. Typically, much of the data needed by a particular application reside in various database systems running on different platforms. Data collected is always segregated, usually according to departments, teams or even geographical area. Organizations need to migrate the data and access information from diverse enterprise business application within specified time frames. The information often required to the users is stored in the form of materialized views in the data warehouse.

A view is a derived relation defined in terms of base relations. A view thus defines a function from a set of base tables to a derived table; this function is typically recomputed every time the view is referenced.

4.2 Materialized Views in Data Warehouse

For the success of a data warehouse, the basic requirement is the ability to provide decision makers with both accurate and timely consolidated information as well as fast query response time [105][106]. For this purpose, a common method that is used in practice for providing higher information and best response time is the concept of materialized views, where a query is more quickly answered. A data warehouse stores materialized views over data from one or more sources in order to provide fast access to the integrated data sources, regardless of the availability of the data sources. Queries to large databases often involve joins between tables and aggregations such as SUM or both. These operations are expensive in terms of time and processing power. The type of materialized view we create determines how the materialized view is refreshed and used by query. We can use materialized views in a number of ways and can use almost identical syntax to perform a number of roles. A materialized view can replicate data, a process formerly achieved by using the
CREATE SNAPSHOT statement in Oracle [102][103]. Now CREATE MATERIALIZED VIEW is a synonym for CREATE SNAPSHOT.

Materialized views and view maintenance are important for data warehouse, retailing, banking and billing applications. In these environments, materialized views are often referred to as summaries, because they store summarized data. They can also be used to precompute joins with or without aggregations, replicating distributed data and it quickly accesses for complex joins. A materialized view eliminates the overhead associated with expensive joins and aggregations for a large or important class of queries [108][118][141].

A materialized view is thus like a cache- a copy of the data that can be accessed quickly. Like a cache, a materialized view provides fast access to data; the speed difference may be critical in applications where the query rate is high and the views are complex so that it is not possible to recompute the view for every query [20],[78]. Materialized views are useful in new applications such as data warehousing, replication servers, chronicle or data recording systems, data visualization, and mobile systems. Integrity constraint checking and query optimization can also benefit from materialized views. Materialized views are of three types: Materialized Views with Aggregates, Materialized Views containing only joins and Nested Materialized Views.

### 4.3 Materialized View Maintenance in Data Warehouse

Data warehouse contains large amounts of information, often collected from a variety of independent sources and integrated into a common repository for querying and analysis. As changes are made to the data sources, the warehouse views must be updated to reflect the changed state of the data sources. The process of updating a materialized view in response to changes to the underlying data is called view maintenance. The views either can be recomputed from scratch or incremental maintenance techniques can be used to calculate the changes to the views due to the source changes. In most cases, it is wasteful to maintain a view by recomputing it from scratch. Often it is cheaper to use the heuristic of inertia and thus compute only the changes in the view to update its materialization. A problem
known as the view maintenance problem is how to maintain the materialized views so that they can be kept up to date in response to updates of the actual relations in the remote data sources.

There are numerous algorithms developed to solve the view maintenance problem for traditional database systems [137]. In these database systems, query expressions defined views and actual relations are stored at the same database. The database systems understand view management and view definitions and know what data is needed for propagating updates to the views. In a data warehouse, the query expressions that define views and actual relations may be stored at different database sources residing at many sites. The sources may inform the data warehouse when an update occurs but they might not be able to determine what data is needed for updating the views at the data warehouse. Therefore, they may send only the actual data updates or the entire updated relations to the data warehouse.

Upon receiving this information, the data warehouse may find that it needs some additional source data in order to update the views. Then, it will issue some queries to some of the sources to request the additional source data. Some of the sources may have updated their data again before they evaluate the requesting queries from the data warehouse. Therefore, they will send incorrect additional data to the data warehouse, which subsequently will use the incorrect data to compute the views. This phenomenon is called distributed view maintenance anomaly. Solving the view maintenance problem in data warehouses is thus more complicated than that in traditional database systems.

4.4 Classification and Implementation of Materialized View Maintenance Methods

The existing data warehouse view maintenance techniques can be classified into two broad categories: Recomputational view maintenance (RVM) and Incremental view maintenance (IVM). Depending on whether the data warehouse has to query the remote data sources in order to calculate the new views, the techniques can be classified as self-maintainable or not self-maintainable [145].
4.4.1 Recomputational View Maintenance (RVM)

In RVM, view is computed from scratch by using the view definitions and other materialized views at the data warehouse. Entire data of view is recomputed at the data warehouse [145],[157]. Following are the two techniques belonging to RVM.

4.4.1.1 Self Maintainable Recomputation (SMR)

Here, the materialized views are computed from scratch by using the view definitions and other materialized views at the data warehouse. The current materialized views being maintained have no contribution to the calculation of the new views. Some techniques replicate all or part of the remote data at the data warehouse. We can view these replicated data as some kind of materialized views at the data warehouse.

A self-maintainable materialized view can be defined in two ways. In the first way, the view ‘V’ is defined as

\[ V = \pi_{proj}(\sigma_{cond}(v_1 \bowtie v_2 \bowtie \ldots \bowtie v_N)) \]  

--- Equ. 4.1
Where all $v_i$’s are self-maintainable materialized views stored in the data warehouse.

In the second way, view $V$ can be defined as

$$V = \pi_{proj}(\sigma_{cond}(r_1 \bowtie r_2 \bowtie \ldots \bowtie r_N))$$

--------Equ. 4.2

where all $r_i$’s are self-maintainable relations stored in the remote data sources.

The techniques in this category do not query the remote data source for additional data. In the average case, the amount of space needed is:

$$\text{Card}(V) ts(V) + \sum_{i=0}^{\text{Nav}} \text{Card}(A_{V_i}) ts(A_{V_i})$$

--------Equ. 4.3

Where $0 \leq \text{Nav} \leq N$.

Card( ) is the cardinality, ts( ) is the tuple size, Nav( ) is the number of auxiliary views per view.

If all auxiliary views are the same, the above formula can be reduced to

$$\text{Card}(V) ts(V) + \text{Nav} \text{Card}(A_{V}) ts(A_{V})$$

--------Equ. 4.4

Nav=0 when none of the auxiliary views is needed. It means that the materialized view is a replicated remote relation. This is the best case. The formula can be rewritten as:

$$\text{Card}(V) ts(V)$$

--------Equ. 4.5

In order to propagate an update to data warehouse replicated relation, the number of rows to be accessed at the data warehouse is the cardinality of the relation itself plus the cardinality of the update. That is,

$$\text{Card}(R) + \text{Card}(U)$$

--------Equ. 4.6

The materialized views in the data warehouse have to be maintained in a correct order from the bottom level to the top level in the view hierarchy. The bottom level views have to be maintained first. Then, those views in the second to the
bottom level must be maintained next and so on. Finally, the views at the top of the hierarchy are maintained. This is the order that updates from the remote data sources are propagated to the data warehouse.

An advantage of the techniques in this category is that the view maintenance anomaly problem is avoided as all necessary data are available at the data warehouse. The data warehouse knows the view definitions and what to do with the views to keep them up to date. It eliminates accesses to the remote relations and therefore, it does not compete with the remote data sources local resources. The data warehouse maintenance operations can then be totally separated from other OLTP operations. Whether a remote data source is available or not will not affect the data warehouse view maintenance process. However, in order to make the materialized views self-maintainable, additional materialized views that provide information necessary for view updates must be stored. Extra storage and time are thus needed to maintain these additional views.

4.4.1.2 Not Self Maintainable Recomputation (NSM)

The data warehouse can recompute the materialized views from scratch using the view definitions, possibly some other materialized views, and actual source relations periodically or whenever the source data is updated [2][62], [143]. That is, when an update occurs at the data source or periodically, the source will inform the data warehouse. According to the query expression that defines the view, the data warehouse may get part of data it wants from other materialized views at the data warehouse, and issue queries to the sources to get the other data it does not have. The sources send the query results back to the data warehouse. Based on the query results, the data warehouse calculates the views and stores the results as materialized views in the data warehouse. The current materialized views have no contribution to the calculation of the new views. The data warehouse may replicate part of the remote relations in the data warehouse. However, these data are not enough for maintaining the materialized views. Therefore, the data warehouse will have to query the remote data sources for additional data in order to maintain the views. An extreme case is where the data warehouse does not replicate any remote relations.
If the view maintenance process is not designed carefully, the distributed view maintenance anomaly problem will occur. Suppose that there is a data warehouse system where the remote data sources send updated relations to the data warehouse whenever an update occurs at the data sources. Upon receiving the information, the data warehouse is ready to compute the new views. But, now let us assume that the data warehouse finds that it needs some other relations at some remote data sources to compute the new views. It will issue queries to these data sources. Suppose the data sources that sent the updated relations to the data warehouse update the relations again before they receive the queries from the data warehouse. The data sources answer the query and send the results to the data warehouse. These results might contain extra information that is incorrect. The data warehouse will then use the incorrect data to compute the new views, which will result in incorrect new views. There are a lot of solutions to the distributed view maintenance anomaly problem.

The simplest solution is as follows. Instead of sending the updated relations to the data warehouse, the data sources just simply inform the warehouse that there is an update that occurs at the data sources. The data warehouse will issue queries to the data sources to request all relations required in the view definitions. After the data sources receive the queries, they will send all requested relations to the data warehouse. The anomaly problem is thus avoided. The not self-maintainable recomputation approach is simple. The anomaly problem can be avoided easily. However, the recomputation process is also time and resource consuming. The data warehouse sends queries back to the sources and waits for answers in order to compute the new views. Processing these queries consumes the sources’ local resources. If the sources are unavailable, the data warehouse will not get the answers it needs.

For storage space requirement at the data warehouse, we consider only the case where the data warehouse does not replicate any base relations. Therefore, the data warehouse always has to query the remote data sources. The data warehouse stores only materialized views. In this extreme situation, the amounts of space
needed in the best case, the average case and the worst case are the same and are equal to

\[
\text{Card}(V) \cdot ts(V) \quad \text{------Equ. 4.7}
\]

Only source data is required to be accessed. The reason is that the warehouse recalculates the full view using the source data each time. It does not use the data warehouse data. Suppose the system locks all base relations in order to evaluate the query expression that defines the view. If the nested-loop join method [56] is used to evaluate it, the total number of rows to be accessed is \(\text{Card}(r)^N\).

4.4.2 Incremental View Maintenance (IVM)

With incremental materialized view maintenance, new information is gathered and integrated into the view without throwing away all existing data [4], [152], [155]. Incremental maintenance offers at least two important advantages that may make it more attractive than refreshing a view. First, a complete refresh may be infeasible because of its duration. In particular, a warehouse may be unavailable to users during refresh, so refresh is often done at night or on a weekend. However, for large warehouses, the refresh time may exceed the available update window of a few hours. Furthermore, global corporations may need 24x7 operation and cannot have any downtime. Incremental maintenance performs a relatively small number of operations on each update, so it may be done concurrently with warehouse user transactions.

4.4.2.1 Self Maintainable Incremental Maintenance (SMIM)

In this category, the data warehouse views are maintained by using the view definitions, the materialized views and the view updates. The data warehouse will never query the remote data sources as the information at the data warehouse is enough for maintaining the views. In the extreme case, all remote relations can be replicated at the data warehouse. The self-maintainable warehouse approaches discussed in [7], [147] and [148] belong to this category.
Let us discuss how to maintain a view \( V \) that is defined as follows:

\[
V = \pi_{\text{proj}}(\sigma_{\text{cond}}(v_1 \bowtie v_2 \bowtie \ldots \bowtie v_N))
\]

---Equ. 4.8---

Where each \( v_i \) is a materialized view and is defined as either

\[
V_i = \pi_{\text{proj}}(\sigma_{\text{cond}}(v_{i1} \bowtie v_{i2} \bowtie \ldots \bowtie v_{iN}))
\]

where each \( v_{ij} \) is a view defined by other auxiliary materialized views,

or,

\[
V_i = \pi_{\text{proj}}(\sigma_{\text{cond}}(r_{i1} \bowtie r_{i2} \bowtie \ldots \bowtie r_{iN}))
\]

where each \( r_{ij} \) is a base relation.

The data warehouse never needs to query the remote data source to get additional data. The data warehouse maintenance operations can be totally separated from other OLTP operations. Whether the remote data source is available or not will not affect the data warehouse view maintenance process. However, in order to make the materialized views self-maintainable, the auxiliary views are stored in the data warehouse to provide the additional information. Extra storage and time overhead are therefore required to maintain the auxiliary views themselves. How to design materialized views at the data warehouse so that only necessary information are stored at the data warehouse is a major issue [62].

Similar to the self-maintainable recomputation techniques, the techniques in this category can replicate all or part of the remote data at the data warehouse. In the average case, the space needed is as follows:

\[
\text{Card}(V) \times \text{ts}(V) + \sum_{i=0}^{N_{\text{av}}} \text{Card}(A_{V_i}) \times \text{ts}(A_{V_i})
\]

---Equ. 4.9---

Where \( 0 \leq N_{\text{av}} \leq N \). If all auxiliary views are the same, the above formula can be reduced to

\[
\text{Card}(V) \times \text{ts}(V) + N_{\text{av}} \times \text{Card}(A_{V}) \times \text{ts}(A_{V})
\]

---Equ. 4.10---
In the best case, no auxiliary views are needed as the view is self-maintainable. The formula can be rewritten as follows:

\[ \text{Card}(V) \times t_s(V) \]  
\[ \text{Equ. 4.11} \]

In the worst case, all auxiliary views are needed to maintain the view. The formula can be rewritten as follows:

\[ \text{Card}(V) \times t_s(V) + \sum_{i=0}^{N} \text{Card}(AV_i) \times t_s(AV_i) \]  
\[ \text{Equ. 4.12} \]

No queries are sent to the data sources for additional information. Therefore, the number of rows accessed in the data source is equal to 0. For N base relations in a view, \( N_{av} \) should be less than or equal to N. In the worst case, \( N_{av} \) is equal to N.

### 4.4.2.2 Not Self Maintainable Incremental Maintenance (NSIM)

Instead of recomputing every view from scratch, only parts of the warehouse that change are computed. However, the data warehouse has to query the remote data sources whenever necessary because the information at the data warehouse is not enough to maintain the view. A number of existing approaches fall under this category. Among them are the unrestricted base access [149], [150], [151].

Here we consider the Eager Compensating Algorithm (ECA) for this category [68]. In ECA, a temporary table COLLECT is used to store intermediate query answers. For every update, the queries including compensated queries are sent to the data source. Note that the COLLECT table is empty only when there is no query to the data source, or the answers for all the queries are returned to the data warehouse before a new update occurs at the data source. This is the best case.

The space needed is as follows:

\[ \text{Card}(V) \times t_s(V). \]  
\[ \text{Equ. 4.13} \]
All tuples in the view table have to be accessed in order to find a tuple to integrate with the view update. However, the data warehouse may have to access data from remote sites except for the best case.

4.5 Unrestricted Base Access

In the unrestricted base access approach [149], [154], the data warehouse accesses the actual relations from the data sources whenever necessary in order to maintain the materialized views. There are many proposed algorithms that follow this approach. The Eager Compensating Algorithm (ECA) is the simplest among them. It is also the fastest algorithm that will let the data warehouse remain in a consistent state.

Suppose an update $U_i$ occurs at a data source. The data source sends $U_i$ to the data warehouse. Suppose the data warehouse wants to update the materialized view that is defined as

$$V = \pi_{\text{proj}}(\sigma_{\text{cond}}(r_1 \bowtie r_2 \bowtie \ldots \bowtie r_N))$$  \hspace{1cm} \text{-------Equ. 4.14}

The data warehouse determines the query for calculating the delta view as

$$Q_i = \pi_{\text{proj}}(\sigma_{\text{cond}}(r_1 \bowtie r_2 \bowtie \ldots \bowtie r_{i-1} \bowtie r_i \bowtie r_{i+1} \bowtie \ldots \bowtie r_N))$$ \hspace{1cm} \text{-------Equ. 4.15}

It then creates a temporary COLLECT table and Unanswered Query Set (UQS) set for processing this specific query and sets both the COLLECT table and UQS to empty.

4.6 Runtime Warehouse Self-Maintenance

Design-time self-maintainability is not flexible. It may be difficult or impossible for us to know the exact contents of the views and their updates at design time. To solve this problem, a run time warehouse maintenance approach has been introduced [62].
The basic idea of the runtime self-maintenance approach is that the data warehouse generates the self-maintainable test for the views to determine whether the views are self-maintainable for a particular update. At run time, the self-maintainable test determines the views for self-maintainability. If a view is self-maintainable, then the view can be maintained by the update information, the view itself and the query expression that defines the view. In this case, the run-time self-maintainable approach is similar to the self-maintainable warehouse approach discussed in the previous section. However, the data warehouse does not store and maintain any auxiliary views. If the view is not self-maintainable, then the data warehouse has to query the remote data sources for those relations it needs in order to update the view. In this case, this approach is similar to the unrestricted base access approach.

The runtime warehouse self-maintenance approach uses the view maintainability test first before doing any maintenance. This creates the overhead in the data warehouse. However, it maintains the views by using the views in the data warehouse or some base relations only, without requiring auxiliary views that are necessary in the self-maintainable incremental maintenance approach. This saves the storage space in the data warehouse.

### 4.7 Performance Evaluation of RVM and IVM

We consider materialized view maintenance, where views are stored in a data warehouse and integrated from multiple, often distributed and autonomous data sources. One of the major challenges is to correctly and efficiently update the views as the source data changes. In the performance evaluation, Select-Project-Join views are considered. We measure the performance of the techniques in terms of space and number of row accessed. The good technique is the one which takes less amount of space to store data including detail data taken from different data sources and the intermediate result. Also it must require less number of comparisons to integrate the data into the data warehouse. The result is it will take less amount of time to refresh the data warehouse. The overall impact is on the down time of the data warehouse. Obviously the downtime time is reduced.
4.7.1 Evaluation Parameters

The evaluation parameters which are used to check the performance of the analytical models are as shown in Table 4.1.

Table 4.1 Evaluation Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Space</td>
<td>Requirement of total amount of memory space to store data in the data warehouse, including space for auxiliary views.</td>
</tr>
<tr>
<td>Number of rows accessed</td>
<td>Requirement of total number of rows that must be accessed in order to integrate the update into the data warehouse.</td>
</tr>
</tbody>
</table>

4.7.2 Evaluation

To test the performance of the above specified techniques we calculate the ‘total amount of space requirement’ including space for auxiliary views using the equations derived previously and the results are shown as follows:

Table 4.2 Comparison of memory space requirement

<table>
<thead>
<tr>
<th>Cardinality of base relation or auxiliary view</th>
<th>SMR</th>
<th>NSMR</th>
<th>SMIM</th>
<th>NSIM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1X10^5</td>
<td>10^{6.1}</td>
<td>10^{5.6}</td>
<td>10^{6.1}</td>
<td>10^{5.8}</td>
</tr>
<tr>
<td>2X10^5</td>
<td>10^{7.4}</td>
<td>10^{6.8}</td>
<td>10^{7.4}</td>
<td>10^{7}</td>
</tr>
<tr>
<td>3X10^5</td>
<td>10^{7.6}</td>
<td>10^{7}</td>
<td>10^{7.6}</td>
<td>10^{7.1}</td>
</tr>
<tr>
<td>4X10^5</td>
<td>10^{7.7}</td>
<td>10^{7.1}</td>
<td>10^{7.7}</td>
<td>10^{7.2}</td>
</tr>
<tr>
<td>5X10^5</td>
<td>10^{7.8}</td>
<td>10^{7.3}</td>
<td>10^{7.8}</td>
<td>10^{7.5}</td>
</tr>
</tbody>
</table>
It is seen that, the space needed for SMR and SMIM approaches grow exponentially with the cardinality of updates (Table 4.2). No additional space is required at the data warehouse for the NSMR technique.

![Figure 4.2 Space used in the data warehouse](image)

**Figure 4.2  Space used in the data warehouse**

We can understand from Figure 4.2 that, the NSMR (Not Self Maintainable Recomputation) technique requires less amount of space as compared to SMR (Self Maintainable Recomputation), SMIM (Self Maintainable Incremental Maintenance) and NSIM (Not Self Maintainable Incremental Maintenance) because no data is stored at the data warehouse site for view maintenance. However, for the Self-Maintainable Recomputation and Self-Maintainable Incremental approaches, extra space is required at the data warehouse to store the replicated source data.

Another performance measure is of ‘**number of rows accessed**’ to integrate the change at the data warehouse site. The result indicates that the SMIM require less number of rows accessed, because it computes the changes incrementally and it is compared with data warehouse view contents. It is also observed that, when the cardinality is increased, the number of rows accessed for SMR and NSMR grows faster than those for SMIM and NSIM (Figure 4.3).
Table 4.3  Comparison of total number of rows accessed

<table>
<thead>
<tr>
<th>Cardinality of base relation</th>
<th>SMR</th>
<th>NSMR</th>
<th>SMIM</th>
<th>NSIM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1X10^5</td>
<td>10^8.5</td>
<td>10^8.8</td>
<td>10^8.1</td>
<td>10^8.3</td>
</tr>
<tr>
<td>2X10^5</td>
<td>10^9.2</td>
<td>10^9.6</td>
<td>10^9.5</td>
<td>10^9</td>
</tr>
<tr>
<td>3X10^5</td>
<td>10^10</td>
<td>10^10.2</td>
<td>10^9</td>
<td>10^9.5</td>
</tr>
<tr>
<td>4X10^5</td>
<td>10^10.3</td>
<td>10^10.4</td>
<td>10^9.4</td>
<td>10^9.9</td>
</tr>
<tr>
<td>5X10^5</td>
<td>10^11</td>
<td>10^11.1</td>
<td>10^9.6</td>
<td>10^10</td>
</tr>
</tbody>
</table>

The SMIM technique exhibits better performances because only exact change is compared with materialize views (Figure 4.3).

Figure 4.3  Number of rows accessed in the data warehouse.

We conclude from Table 4.3 and Figure 4.3 that, when the cardinality is increased, the number of rows accessed for self-maintainable recomputation (SMR) and not self-maintainable recomputation (NSMR) grow faster than those for self-
maintainable incremental maintenance (SMIM) and not self-maintainable incremental maintenance (NSIM).

### 4.8 Advantages and Disadvantages of View Maintenance Methods

Both data warehouse view maintenance methods: Recomputational view maintenance and Incremental view maintenance have some advantages and disadvantages. These are elaborated in the following Table 4.4.

<table>
<thead>
<tr>
<th>View Maintenance Type</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMR</td>
<td>Maintenance operations are totally separated from OLTP operations.</td>
<td>Data are replicated at data warehouse and needs extra storage.</td>
</tr>
<tr>
<td></td>
<td>Unavailable source will not block the maintenance process.</td>
<td>Needs to implement and maintain data transfer process.</td>
</tr>
<tr>
<td>NSMR</td>
<td>It is very simple to implement</td>
<td>Unavailable source will block the view maintenance.</td>
</tr>
<tr>
<td></td>
<td>No need to replicate the data and does not require extra space.</td>
<td>Evaluating query at the data sources consumes local resources.</td>
</tr>
<tr>
<td></td>
<td>No data transfer process is required</td>
<td>View maintenance operations are not separated from OLTP operations.</td>
</tr>
</tbody>
</table>
### 4.9 Conclusions

All data warehouse view maintenance methods can be classified into four major categories. They are SMR, NSMR, SMIM and NSIM. Among all the four categories, not self-maintainable recomputation (NSMR) requires less memory at the data warehouse as compared to other methods. But, requires lot of time to maintain a materialized view due to unavailability of data sources and data transfer from data sources. Self-maintainable incremental maintenance (SMIM) is the best in terms of number of rows accessed in order to propagate an update to the target materialized view in the data warehouse.

Both, SMR & SMIM totally separate the data warehouse view maintenance operations from the OLTP operations. Therefore, the view maintenance operations will not consume data sources local resources. These operations only consume the data warehouse's resources. Even if the remote data sources are not available, the data warehouse view maintenance process can continue running. However, a part or all source data are replicated at the data warehouse to make the data warehouse view maintenance process self-maintainable. These replicated data take space. Data transfer processes are implemented to transfer data from the remote data sources to the data warehouse. Design, implement and maintain these processes are time-

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMIM</td>
<td>Maintenance operations are totally separated from OLTP operations. Unavailable source will not block view maintenance process.</td>
<td>Data are replicated and it needs extra storage. Have to implement and maintain data transfer process.</td>
</tr>
<tr>
<td>NSIM</td>
<td>No need to replicate the data and no extra storage required. Data transfer process to transfer data from sources to data warehouse is not required.</td>
<td>Unavailable source will block view maintenance process. Evaluating query at the data sources consumes local resources. Maintenance operations are not separated from OLTP.</td>
</tr>
</tbody>
</table>
Chapter 4

consuming. A lot of unnecessary data may be duplicated at the data warehouse. However, these are the approaches that probably many large companies have to take if they want to separate their data warehouse view maintenance operations from their OLTP operations.

Both, the not self-maintainable recomputation and not self-maintainable incremental maintenance approaches suffer from some common disadvantages. As the remote data sources have to process queries from the data warehouse that consume their limited local resources, the OLTP system will be slow. Once, a data source is unavailable, the data source will not be able to answer queries sent from the data warehouse in time. It will block the data warehouse view maintenance process. The not self-maintainable incremental maintenance approach has some additional disadvantages. To avoid the anomaly problem, the view maintenance process must be designed carefully. If a lot of updates happen at the data sources, the data warehouse may issue many compensating queries. It is possible that the data warehouse may never get the final data.

Chapter Summary

This chapter covers the concept of materialized views in data warehousing and its categories. The fast access to warehouse data compels the need of materialized view. The analytical models for all four types of view maintenance types are derived and the performance evaluation of these techniques is done using analytical model. It is observed that incremental maintenance technique of materialized view maintenance requires less time to refresh the warehouse due to additional data storage at the data warehouse site. The comparisons of different view maintenance methods have also been explained.