Chapter 6
Conclusions, Limitations and Future Work

6.1 Introduction
6.2 Conclusions
6.3 Limitations of the Research Work
6.4 Future Research Work
Chapter 6

6.1 Introduction

A data warehouse is built for the purpose of information integration, decision support and analysis. It is the leading and most reliable technology used today by companies for resource planning, forecasting and data management. A data warehouse extracts, integrates and stores relevant information from a distributed set of data sources and process through OLAP and other tools for decision making. In this thesis, we have addressed the core issue that arise in maintenance of materialize view in the Data Warehouse.

Materialize view computation and maintenance are two very basic requirements that need to be addressed to achieve the benefits of applying materialized views such as efficient access, reliable performance and high availability of data. These two requirements face problems due to large number of data sources, increasing size in each data source and high amount of data at the data source.

The research work we carried out is, finding the optimized methods of materialized view maintenance in data warehousing. During the work, we found that materialized view maintenance in homogeneous data sources is quite simpler as compared to heterogeneous data sources. The number of updates in a given data set over different data sources is tested in different situations. We have applied the proposed techniques upon a set of data sources and we proved the success of these techniques.

A literature survey on the materialized view maintenance techniques covers almost all the aspects of the view maintenance in data warehousing. Separate chapter is devoted on the analysis of the existing techniques of the materialized view maintenance. By using simulator we have studied the existing view maintenance techniques. During research it has been found that the materialized view is most beneficial for improving query performance as it stores pre-computed data. We have presented the solution to materialize view maintenance in a mixed data update. During the simulation of materialized view maintenance we have understood that, if additional data is stored at the data warehouse, it reduces the view maintenance cost. The materialize view maintenance cost contain, the time required to compute the
changes which is to be reflected in the data warehouse and the time required to refresh the data warehouse. Using the proposed algorithm, it is proved that the view maintenance cost can be reduced by putting additional data at the data warehouse for maintenance.

6.2 Conclusions

We started the research work with the objective of studying the data warehouse system and to understand various research challenges in data warehouse system. Our prime objective was to optimize the materialize view maintenance process in the data warehouse by automating view maintenance in the presence of concurrent updates. This minimizes the overall view maintenance cost. The materialize view maintenance cost includes the cost and effort required to compute the changes required to refresh the data warehouse. The overall view maintenance cost of materialized views includes the cost of propagate the changes and the cost of refresh operation.

It is observed that the data volume doubles in every 18 months. In the survey it is found that, the data volume increased by 162 percent in the year 2012 as of 2011. It is also found that percentage of data stored in digitized form increased by 34 percent in the year 2000 and 162 percent in the year 2010. Therefore, due to tremendous growth of data, it became difficult to store, comprehend the data and convert it into more meaningful knowledge. In order to solve elevate these problems the data warehouse has been developed. Data warehouse provides powerful decision making tools in the hands of the end users in order to facilitate prompt decision making.

While studying the basic architecture of the data warehouse system, we have understood the number of research problems that arise from the warehousing system. The components of data warehousing system include wrapper, monitor, integrator, data warehouse data repository and data sources.
Wrapper

The responsibility of the wrapper component is to convert the data extracted from data sources into the compatible format of the data warehouses data model. If the data warehouse model is relational and the data sources consists of a set of flat files then the wrapper component must support the interface to convert the flat file into the relational data model. This data translation problem is inherent in all the data integration techniques. Generally, a component which is responsible to convert the incoming data into the compatible data models data called as translator.

Monitor

The responsibility of this component is to detect the necessary changes happen in the data sources. The data sources which supply data to the data warehouse may be cooperative sources, logged sources, queryable sources or snapshot sources. The cooperative data sources provide triggers like mechanism to notify a change happen at the underlying data source. The logged sources maintain log that also permit to check or query the log to find the interested changes. The queryable data sources allow the monitor to query for changed data, so that data polling can be used to detect the interested changes. The snapshot sources maintain the snapshot of the changes and provide these snapshots to the data sources.

A significant issue is to develop techniques and tools that automate or semi-automate the process of implementing wrapper/monitors, through a toolkit or specification based approach.

Integrator

The job of the integrator component in data warehousing architecture is to receive change notifications from the wrapper/monitor pair for the information sources and integrate these changes in the data warehouse. Viewing the problem in this way, the job of the integrator is to perform view maintenance in data warehouse. However, there are a number of reasons that conventional view maintenance techniques cannot be used and each of these reasons highlights a research problem associated with data warehousing:
The views stored at the warehouse site are more complicated than conventional views.

Data warehouses also store aggregated and summarized data.

The information sources update data independently and are unable or unwilling to participate in view maintenance.

In conventional view maintenance method, every updates sent to the views, whereas in data warehousing only large batch updates are combined.

We have studied the basic view maintenance methods: Immediate view maintenance and Deferred view maintenance. In the immediate view maintenance, view is maintained as a part of the transaction and in the deferred view maintenance, view maintenance is delayed. Actual maintenance is done explicitly or when the system has free cycles. In the deferred view maintenance if a view is not up to date, it is transparently brought up to date when needed. We have developed the simulator to test the performance of these methods. The results shown that, the response time of the system is reduced drastically using deferred view maintenance. The immediate view maintenance is expensive because each update requires holding lock on view to reflect the changes. Using deferred view maintenance the maintenance time is reduced by 41%.

We have also classified the materialized view maintenance methods on the basis of view computation: Recomputational view maintenance (RVM) and Incremental view maintenance (IVM). In RVM, view is computed from scratch by using the view definitions and other materialized views at the data warehouse. Entire view contents are recomputed at the time of view maintenance. The RVM is further classified into Self Maintainable Recomputation (SMR) and Not Self Maintainable Recomputation (NSMR).

In the IVM, the views are maintained incrementally. The changes in the data sources are computed and sent towards the data warehouse views incrementally. The IVM is classified into two types: Self Maintainable Incremental Maintenance (SMIM) and Not Self Maintainable Incremental Maintenance (NSIM). In SMIM, the
views are maintained by using the view definitions, the materialized views and the view updates. The data warehouse never query the remote data sources. In NSIM, 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.

We have developed the analytical models for the above mentioned four types of view maintenance methods. The results shown that, the NSMR requires 37% less memory space to store data at the data warehouse as compared to other three methods. The SMIM method requires 45% less number of rows accessed in order to integrate the data into data warehouse.

In the proposed research work, we have developed an efficient materialized view maintenance technique which reduces the materialized view maintenance cost. The maintenance cost is nothing but the time required to refresh the materialize views when the changes happen at the data sources. The total view maintenance cost consists of calculating the exact changes and integrates these changes into the views of a data warehouse without affecting user interaction with the data warehouse. The proposed view maintenance process is hidden from the users of the data warehouse and requires 20% less time to maintain the views. The changes of interest are computed incrementally and implicitly. No further queries are required to the data sources about the changes. These changes are clubbed together and integrated into the data warehouse at once when the data warehouse is free or whenever the maintenance manager run the maintenance trigger. Therefore, it reduces the downtime of the data warehouse.

We found that, using the proposed method the data warehouse views refresh time is going down by 20% as compared to Counting Algorithm and using Delta table method. The error ratio is only 0.0139% and it is negligible. Also the proposed method is faster that the counting algorithm and delta table method. The maintenance time is reduced due to proposed automated and self maintainable materialized view maintenance process. The changes are computed incrementally; it is stored in a batch and integrated in the data warehouse materialized views at once. At the time of view maintenance, the computed changes are not compared with the
6.3 Limitations of the Research Work

The research work has been tested using the simulator. The new suggested technique of materialize view maintenance may be applied on actual large data bases then slight changes are expected. One can consider the limitation of the new techniques as, the scale of the system is limited. The large number of attributes and relations (data sources) are not considered for the proposed model. Most of the research work we carried out for testing our technique is simulation based. Although the area of view maintenance using additional relations at data warehouse site has many dimensions, there is a scope to work in the field of view maintenance performance improvement.

Materialize view maintenance techniques can be classified into four major categories. They are SMR, NSMR, SMIM and NSIM. Using our proposed analytical model, we have captured the behavior of these techniques and plot the results. There may be the slight change in behavior of the model if it is applied on large scale data.

6.4 Future Research Work

In this thesis we have presented a comprehensive approach of materialize view maintenance in data warehousing. However, we have modeled a materialized view as a relation and provided a maintenance technique by installing triggers on source relations (data sources). Our experience with our experiments shows that, while the incremental view maintenance improves the view refreshing performance, it does not decrease the number of auxiliary relation at the data warehouse site. Therefore, to decrease the number of additional relation/s may be the future enhancement.

Our experimental results show that different query types have significant impact on the system’s performance. Future research would focus on validating the model against some real-world data warehouses. Also one can extend the
simulation model discussed in our thesis to incorporate more features making the model more realistic.

Since, the data sources are heterogeneous, complex extract-transform-load (ETL) processes are often necessary to achieve the integration of such data. How to manage views with some common ETL operations is still an unexplored but important problem.

It is well known that data warehouses are focused on decision support rather than on transaction support and that they are prevalently characterized by an OLAP workload [132][137]. Though, a lot has been written about how a data warehouse should be designed, there is no consensus on a design method yet. A comprehensive design method can be designed, so that it will be used by almost all the developers of the data warehouse.