Improve Data Warehouse Performance by Preprocessing and Avoidance of Complex Resource Intensive Calculations

Muhammad Saqib[1], Muhammad Arshad[2], Mumtaz Ali[3], Nafees Ur Rehman[4], Zahid Ullah[5]

**[1,2] City University of Science & Information Technology,
Peshawar, KPK, Pakistan**

**[3, 4, 5] Institute of Management Sciences,
Peshawar, KPK, Pakistan**

Abstract

A Data Warehouse is a computer system designed for archiving and analyzing an organization's historical data, such as sales, customers, products, salaries, or other information from day-to-day operations OLTP. Normally, an organization summarizes and copies information from its operational systems to the data warehouse on a regular schedule, such as daily, weekly, monthly, quarterly or annually; after that, management can perform complex queries and analysis OLAP on the information without slowing down the operational systems. Materialized views can be one best option in this regard and can be used in a number of ways. It can be used in distributed databases for replication and can also be used for efficient provision of data to a query through query re-writing. The process of data provision to queries can further be expedited if dependent child views are created on an already existing materialized view. Furthermore, these child-views are automatically created upon the creation of the base materialized view with some restrictions. This results in less-user dependent activity of creation of materialized views based on some parameters. These parameters are the number of child-materialized views and the type of the data a view contain. In this paper, a balanced approach is suggested to create sub-materialized views to answer user queries without consulting the fact table or parent materialized view that results in avoidance of resource intensive calculations and joining of multiple tables.

Keywords: *Materialized View, Aggregation Plan, OLTP, OLAP.*

1. Introduction

Most of the modern enterprises and organizations rely on knowledge-based management systems. In such kind of systems, knowledge is gained from data analysis. Nowadays, knowledge-based management systems include data warehouses as their core components. The purpose of building a data warehouse is twofold. Firstly, to integrate multiple heterogeneous, autonomous, and distributed data sources within an enterprise. Secondly, to provide a platform for advanced, complex, and efficient data analysis. Data integrated in a data warehouse are analyzed by the so-called On-Line Analytical Processing (OLAP) applications designed among others for discovering trends, patterns of behavior, and anomalies as well as for finding dependencies between data. Massive amounts of integrated data and the complexity of integrated data that more and more often come from WEB-based, XML-based, spatio-temporal, object, and multimedia systems, make data integration and processing challenging.[1]

Information and knowledge is one of the most valuable assets of an organisation and when used properly can assist in intelligent decision making that can significantly improve the functioning of an organisation. Data Warehousing is a recent technology that allows information to be easily and efficiently accessed for decision-making activities by collecting data from many operational, legacy and possibly heterogeneous data

Copyright (c) 2012 International Journal of Computer Science Issues. All Rights Reserved.

sources. On-Line Analytical Processing (OLAP) tools are well-suited for complex data analysis, such as multi-dimensional data analysis, and to assist in decision support activities while data mining tools take the process one step further and actively search the data for patterns and hidden knowledge in the data held in the warehouse. In our common practice, many organisations are building, and or planning to develop, data warehouses for their operational and decision support needs.[2]

Materialized views have been found to be very effective in speeding up query, as well as update processing, and are increasingly being supported by commercial database systems. Materialized views are especially attractive in data warehousing environments because of the query intensive nature of data warehouses. [3] Typical Data warehouse queries are complex and ad-hoc in nature and normally these queries access huge volumes of warehouse data and perform many joins and aggregations. Query response time and throughput are therefore more important than transaction throughput. The data warehousing environment provides a computerised interface that enables business decision-makers to creatively approach, analyse and understand business problems. The aim of the data warehouse system is to turn data into strategic decision making information and to bring solutions to users. This process is done by tuning the data at many steps.[2]

Materialized view eliminates the overhead associated with expensive joins and aggregations for a large or important class of queries. Queries to large databases often involve joins between tables, aggregations such as average, sum, count or both aggregation & joins.

Materialized views can provide massive improvements in query processing time, especially for aggregation queries over large tables.[6] A materialized view takes a different approach in which the query result is cached as a concrete table that may be updated from the original base tables from time to time. This enables much more efficient access, at the cost of some data being potentially out-of-date. It is most useful in data warehousing scenarios, where frequent queries of the actual base tables can be extremely expensive.

In addition, because the materialized view is manifested as a real table, anything that can be done to a real table can be done to it, most importantly building indexes on any column, enabling drastic speedups in query time. In a normal view, it's typically only possible to exploit indexes on columns that come directly from (or have a mapping to) indexed columns in the base tables; often this functionality is not offered at all.

A multidimensional data warehouse (MDW) is a repository in which data is organized along a set of dimensions D = d1, d2…. dn. A possible way to design a MDW is the star-schema in which, for each dimension, there is a dimension table Di that has di as its primary key and also uses a fact table. [3]
Materialized views were implemented first by the Oracle database [9]. These storage structures have attracted much attention since then. The life cycle of MVs have three major stages:

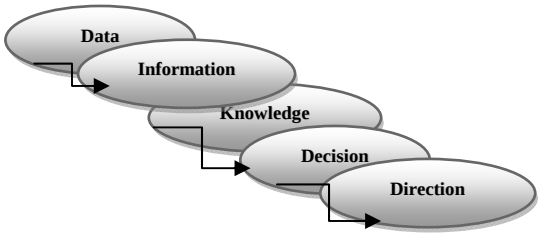
View design: determining what views to materialize, including how to store and index them.
View maintenance: efficiently updating materialized views when base tables are updated.
View exploitation: making efficient use of materialized views to speed up query processing. [8]

Creation and Maintenance
Data warehouses contain large amounts of information, often collected from a variety of independent sources. Decision support functions in a warehouse, such as on-line analytical processing (OLAP), involve hundreds of complex aggregate queries over large volumes of data. It is not feasible to compute these queries by scanning the data sets each time, Warehouse applications therefore build a large number of summary tables, or materialized aggregate views, to help them increase the system performance. [5]

It is a matter of high concern to decide what data is to be depicted in materialized views and in what numbers materialized views should be created. This decision is obviously influenced by the pattern users and applications access the data. It may happen, if not properly given attention, that a materialized view is created but the data depicted in that materialized view is never or rarely accessed. On the other hand, queries may not be redirected to use materialized views, instead, base tables are accessed in query responses.

The design, implementation and maintenance of materialized views is not single time activity, this process will be carried out through out the life of the data warehouse and database systems. The DBA periodically checks whether a materialized view is in use or not. If a materialized view is not in use, it is dropped to achieve space and time efficiency. New Materialized Views need to be created incase if more data extraction queries are accessing base tables. Various queries that require data from one domain can easily be answered by creating a relevant MV. There are various algorithms in this regards. Like MiniCon Algorithm, it is a scalable algorithm for answering queries using views. [7]

Copyright (c) 2012 International Journal of Computer Science Issues. All Rights Reserved.



(Figure1: Data Analysis Chart)

Types of Materialized Views:

The types of materialized views used are as follows:

a) Materialized views with aggregates:

Materialized views contain aggregates in data warehouses for fast refresh to be possible. The valid aggregates functions are Sum, Count, Average, Variance, Min, Max, Standard Deviation etc

b) Materialize Views with Joins:

Some materialized views contain only joins and no aggregates. The advantage of creating this type of view is that expensive joins will be pre-calculated. A materialized view containing only joins can be defined to be refreshed On Commit or On Demand.

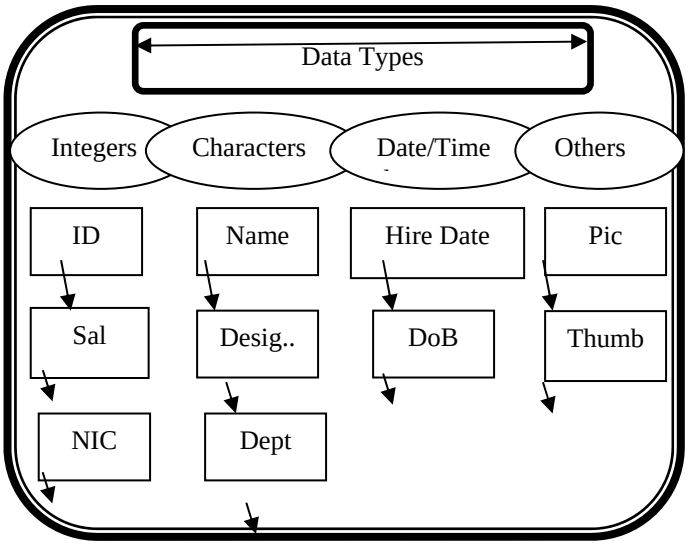
c) Nested Materialized Views:
A nested materialized view is that type of materialized view whose definition is based on another materialized view. A nested materialized view can reference other relations in the database in addition to referencing materialized views. [9]

2. Child-View Creation

Once a base-relation or a materialized view is created and populated, then the process of creating child MVs is started. In the first step the total number of columns is determined. Afterwards, the data types of all columns are determined. For this purpose data dictionary of the database is retrieved.

Data types of the columns are also important in deciding the kind of aggregation operation on the whole data set. Types of values in a column are first scanned whether these values can be used in aggregation or it can be made part of a subset in the form a child materialized view. It is a known fact that numeric data can easily be aggregated.

But there are certain values which can not be aggregated at all, for example, columns containing IDs can not be used in such context. We focus text databases in this work, because objects of various kinds that can be referenced from inside a database can not be aggregated. Images, audio or video files which are referenced from inside the database can not be aggregated or combined together as we aggregate numeric data using AVG or SUM operation. Character data again depends whether it can be used in aggregation or not. Like if *names* are stored in a column, so this column can not be summarized in any manner considering the values it contain. *Name* column supports no aggregation function excepts COUNT. But if we take *address*, it can be taken for aggregation based on House, Street, Sector, Town, City, Country and Region. *address* supports partial aggregation as it can be used to aggregate the data but standard functions of aggregation like SUM, AVG can not be performed on it. And if we consider, that means some string/character columns inherently have hierarchies while others do not. In figure 2 we have shown different data type's columns, by the combination of these we can create our child Materialized views.

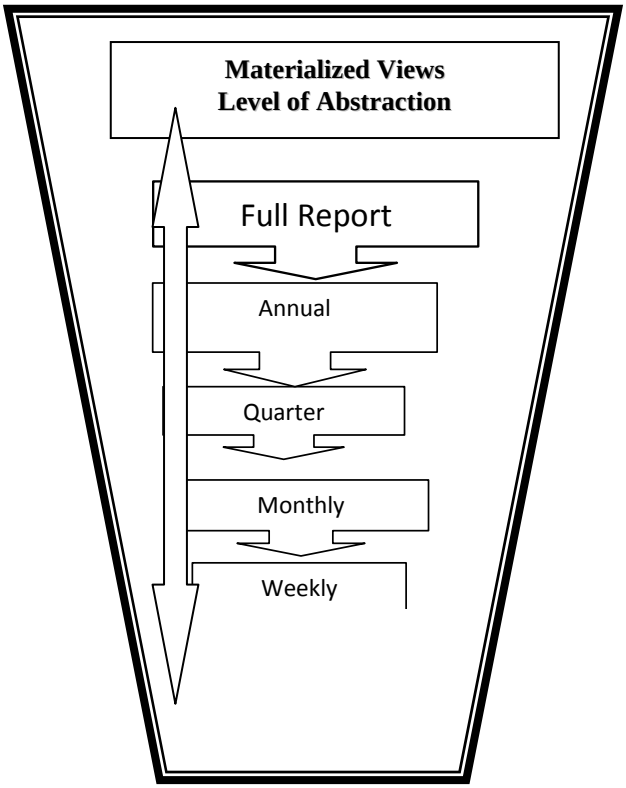


(Figure 2: Aggregation Levels)

We need to look for columns where hierarchies can be used as a tool to aggregate data. This can be done by observing the dataset manually as the data dictionary can not be used to have information regarding each column about its aggregation. In a typical DBMS data dictionary, no such information is found nor there do any space where the schema designer can add this aggregation information. Furthermore, in a dimensional context, the data warehouse schema is variable which may lead to a bit of performance overhead in maintaining such information. However, a subschema specific to a base MV is created in this work to

Copyright (c) 2012 International Journal of Computer Science Issues. All Rights Reserved.

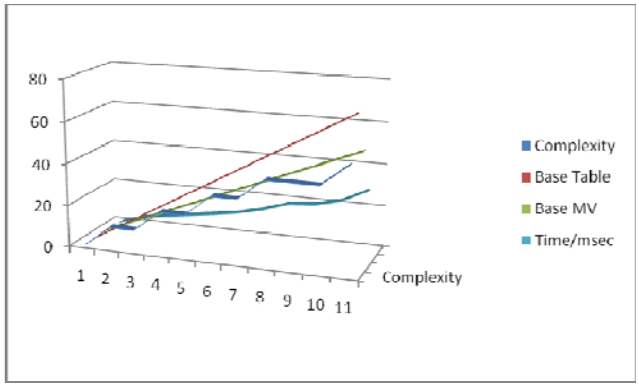
have information regarding each summarizable / aggregable column. Then the kind of aggregation function is decided for each column. If the column is numeric then standard group function may be used for aggregation and if it is a string (e.g. *address)* then contents of the value may be used for aggregation and if it is a DATE/TIME column then only compatible aggregation will be performed. Then the level of aggregation is also decided i.e. from the most granular to the most general level. This information will be used in context with other partner columns in a potential child MV. All such information result in an AGGREGATION PLAN of the MV.

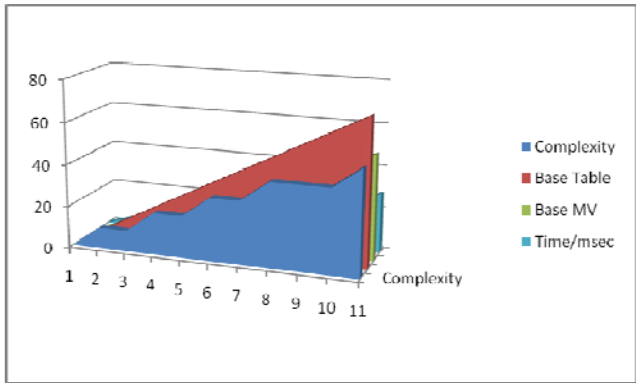


(Figure 3: MV Level of Abstraction)

Once an *aggregation plan* is finalized, then it can be implemented. We have shown Level of aggregations in figure 3. A table containing d columns can have 2^d various combinations of columns. In aggregation, the number of combinations of columns may be double of this because here content based aggregation is also performed. This will result in too many child MVs which will be surely unmanageable and unusable. For this either the aggregation plan is again observed and those potentially useful candidate MVs may be kept and rest of the MVs are deleted. In order to answer a query, a data integration system needs to translate a query formulated on the mediated schema into one that refers directly to the schemas in the data sources. [5]

The following chart shows a comparison of time vs queries complexity of base relation to MV tree.





S#	Complexity	Base Table	Base MV	Time/m sec
1	0	0	0	0
2	10	7	5	3.78
3	10	14	10	5.87
4	20	21	15	7.65
5	20	28	20	9.39
6	30	35	25	11.36
7	30	42	30	14.38
8	40	49	35	18.36
9	40	56	40	19.32
10	40	63	45	22.39
11	50	70	50	28.39

(Chart 1:
Complex Queries Vs Time Series for Base Relation & MV Tree)

3. Future Work

Copyright (c) 2012 International Journal of Computer Science Issues. All Rights Reserved.

The automatic creation of child materialized view is really a complex task. There are a lot of things needed to be considered. While working on this paper, we have found out areas where we think improvements can be brought in. The data dictionary of a database or data warehouse needs modification to include meta data for deciding which numeric and string attributes can be exploited for creating new MVs. Attributes containing multimedia data are not in the scope of this paper. However, our work can be extended by including multimedia data operations as well in MV child creation plan. The decision regarding the number of child materialized view also difficult to answer, so one can also find a balanced number of MVs with high access frequency and remove the rest of the MVs

Conclusion

Materialized Views do contribute in answering queries efficiently to improve the over all performance of Data Warehouse. Efficient query answering can further be speeded up by creating various child materialized views. Data extraction queries select the best MV which can fulfill its data requirements. For creating child MVs, initially the data types of fields/columns of the base MV are determined. All of the columns are grouped according to the data types. Incase of numeric data types, those columns are separated which can not be aggregated e.g, ID, SSN, from numeric attributes where various aggregation operations can be carried out. This whole process is carried out for string attributes as well. This whole process result into an aggregation plan, which later on is translated into a script and is then executed. The process of query answering is made efficient by having more MVs having the potential data required to fulfill maximum requirements of query.

References

[1] *"New Trends in Data Warehousing and Data Analysis"*Series: Annals of Information Systems, Vol. 3 Kozielski, Stanislaw; Wrembel, Robert (Eds.) 1st Edition. 2nd Printing. 2009, XVIII, 364 p. 152 illus.

[2] *"Advances and Research Directions in Data Warehousing Technology"*
Australasian Journal of Information Systems, Vol 7, No 1 (1999)
http://dl.acs.org.au/index.php/ajis/article/view/287

[3] *"A Case for Dynamic View Management"*
Yannis Kotidis AT&T Labs Research and Nick Roussopoulos University of Maryland
ACM Transactions on Database Systems, Vol. 26, No. 4, December 2001, Pages 388–423.

[4] *"Answering queries using views: A survey"*
A.Y. Halevy Department of Computer Science and Engineering, University of Washington, Seattle,WA, 98195

[5] *"Data Cubes and Summary Tables in a Warehouse*" Inderpal Singh Mumick, Dallan Quass, Barinderpal Singh Mumick, *"Maintenance of* June 1997 ACM SIGMOID RECORD.

[6] *"Materialized View Selection and Maintenance Using Multi Query Optimization"*
Hoshi Mistry, Prason Roy, S.Sudershan, K.Ramamritham
ACM SIGMOD 2001 May 21-24, Santa Barbara, California USA

[7] *"MiniCon: A scalable algorithm for answering queries using views"* Rachel Pottinger, Alon Halevy University ofWashington, Department of Computer Science and Engineering, Box 352350 Seattle,WA 98195, USA

[8] *"Optimizing Queries Using Materialized Views: A Practical, Scalable Solution"*
Jonathan Goldstein and Per-Åke Larson
Microsoft Research, One Microsoft Way, Redmond, WA 98052

[9] "Oracle 9i" Data Warehousing Guide, Release 2 (9.2), March 2002, A-96520-01

[10] *"Oracle Database 11g for Data Warehousing and Business Intelligence" Oracle Publishers.*

Copyright (c) 2012 International Journal of Computer Science Issues. All Rights Reserved.