Data Warehouse Design Considerations

1. Introduction

Data warehouse (DW) is a database containing data from multiple operational systems that has been consolidated, integrated, aggregated and structured to support the business analysis and a decision-making process [Inmon, 2005, Kimball, 2002]. Data warehousing is an art on the one hand and a science on the other. Designing a DW is like building a bridge that should be designed and constructed following both civil engineering and architecture principles. The absence of either science or art behind this construction will result in the collapse on the bridge. As being a bricklayer does not necessarily mean that one can also be an architect, a mere knowledge of database technology does not entail expertise on data warehousing. Indeed, DW necessitates simultaneous application of a number of technologies designed sophisticatedly through a body of principles called ‘architecture.’ This is why there is consensus among experts that managing the Data Warehouse System (DWs) is an extremely challenging task [Rainardi, 2007, Shin et al, 2006 ]. Due to the ever-increasing accessible data accumulated from the internet and other sources and owing to the exponential growth of DW enterprises, companies encounter a large body of data thereby gradually rendering it more and more vital for the companies to wisely control their DW. In such situations a sophisticated design is necessary to ensure high data integration, high performance and ease of maintenance. As such, issues related to designing of DWs with high performance are gaining importance. Thus, the contention of the present research in DW design is an art that should be based on decisions and skills of an expert. Accordingly, an advanced database design such as DW becomes much of a challenge as the applications of that database need become more sophisticated and refined. [March et al, 2007].

DWs have two significant architectures, namely, the data flow and the system architectures. Data flow in DW is a substantial architecture that can play an important role to enhance DW performance. It is a configuration of data stores along with the arrangement of how the data flow from the source systems through these data stores to the applications used by the end-users. This includes how the data are controlled, logged, and monitored as well as the mechanism to ensure the quality of the data in the data stores [Rainardi, 2007].

Data flow consists of many parts such as the data model, physical designing, ETL, data quality, metadata to name but a few. Indexing techniques as physical designing and Data Architecture as logical designing are two of the main components of data flow architecture. [Rainardi, 2007].

The data architecture is different from data flow architecture and the activity to produce data architecture is known as data modeling which encompasses how the data are arranged in each data store and how a data store is designed to reflect the business processes.

In addition to data architecture, another component that can significantly improve the query and loading performance of data warehousing is referred to as index techniques. Building indexes on a database has an important impact on the query performance, especially in huge databases such as DW where the queries are usually very complex and ad hoc. [Rainardi, 2007, Inmon, 2005, Kimball, 2002, P'Neil, 1997, O'Neil et al, 1995]. Inasmuch as the characteristics of a very large historical database are considerably
different from those of common transaction processing systems, the proper choice of index structures will enhance the query performance in DWs.

2. Problems and Objectives

In this section, data modeling and indexing technique as two efficient subcategories of data flow architecture of DW are discussed. The two components of DW designing can optimize its performance that in turn will prove extensively beneficial to DW analysts and designers equally.

2.1. Data Modeling

Normalization is a technique first mentioned by Codd [Codd, 1972] and has been deployed by Date [Date, 1997] for determining the optimal logical design to simplify the relational design of an integrated database, based on the groupings of entities to make better performance in storage operations. Most implementations for business applications are based on the relational model that has its own limitations one of which includes the complexity of access paths in physical storage structures of relational database. [Dadashzadeh, 1989, 2005] What is more, there are no references to any physical storage structure in the relational model, and it makes a logical view that is fully independent of the real based and physical organization of the data. Finally, although normal forms have many advantages which are considered the rule for relational database design, they suffer from low system performance [Date, 1997, 2003, Inmon, 1987; 1989].

Date [Date, 2003] supports the fact that denormalization speeds up data retrieval. Nevertheless, there is a disadvantage to denormalization in the case of systems that require potential frequent updates. Basically, updates in dw applications are not usual in that data warehouses entail fairly fewer data updates. It should not be neglected that data in a dw are retrieved during the process of most transactions [Kimball et al, 2002]. Consequently, applying denormalization strategies is best suited to a data warehouses system due to infrequent updating.

There are multiple ways to construct denormalization relationships for a database, such as Pre-Joined Tables, Report Tables, Mirror Tables, Split Tables, Combined Tables, Redundant Data, Repeating Groups, Derivable Data and Hierarchies [Mullins et al,2002, Claudia et al, 2003].

Hierarchical denormalization is a structure that is supported by most relational database management systems such as Oracle, DB2 and so on. Although designing, representing and traversing hierarchies are complex as compared to the normalized relationship, the main approach to reduce the query response time is by integrating and summarizing the data [Joy Mundy, 2006, Claudia et al, 2003]. Hierarchical denormalization is particularly useful in dealing with the growth of star schemas that can be found in many dw implementations [Claudia et al, 2003 and Shin, 2006].

According to the conventional wisdom, the issue open to discussion at this point is all relational database designs should be based on a normalized logical data model [Hanus, 1993]. The advantage of
normalization is to organize data with well-balanced structure to optimize data accessibility, which leads to some deficiencies that decrease system performance [Date, 1997, Westland, 1992, Menninger, 1995].

Furthermore, even though the idea of improving the system by reducing table joins is not new, the decision for denormalization is hardly practical because it calls for a load of administrative burden among which documenting the structure of the denormalization assessments, validating the data, and scheduling for data migration can be mentioned. On the contrary, there are some research studies which have indicated that denormalization may result in a better performance and a more flexible data structure for users. [Claudia et al, 2003, Shin, 2006]. Hence, the ventilation of the present argument between its pros and cons is of a vital importance and can minimize many ambiguities mystifying the truth.

The above-mentioned theoretical issues led the present researcher to pursue the following objectives:

(i) To explore the significance of enhancement in query processing performance through optimizing the dw design adhering to denormalized hierarchical technique as a logical data modeling
(ii) To investigate the possibility of positive effect(s) of Bitmap Index on columns involved by denormalized implementation

2-2 – Indexing Technique

There are various index techniques supported by database vendors such as Bitmap index [Inmon, 2005], B-tree [Comer, 1979, Kimball et al, 2002, Strohm, 2007], Projection index [O’Neil et al, 1997], Join bitmap index [O’Neil, 1995], Range base bitmap index [Wu et al, 1998] and so on. As we know, Bitmap index is advisable for a system that contains data that are not frequently updated by many concurrent processes [O’Neil, 2007, Cand Galemmo et al, 2003, Stockinger et al, 2007]. This is mainly due to the fact that Bitmap index stores large amount of row information in each block of the index structure. In addition, since Bitmap index locking is at the block level, any insert, update, or delete activity may result in locking an entire range of values [Lewis, 2006].

On the other hand, B-tree index is recommended for systems frequently updated. The reason is they do not need re-balancing as frequently as other self-balancing search trees. Further, all leaf blocks of the tree are at the same depth [Strohm, 2007].

Thus, choosing the proper type of index structures has a big impact on the DW environment. The main reason behind this problem is that there is no definite guideline for DW analysts to choose appropriate indexing methods. According to common practice, Bitmap index is best suited for columns having low cardinality and is recommended for low cardinality data [Chaudhuri, 1997, Kimball et al, 2002, Strohm. R 2007]. Strohm [Strohm. R 2007] concludes that the advantages of using Bitmap indexes are greatest for low cardinality columns, i.e., columns which have a small number of distinct values compared to the number of rows in the table. If the number of distinct values of a column is less than 1%, the column is a candidate for a Bitmap index. This assumption may be correct to some extent based on previous algorithms...
and based on old machine processing used by the database software and hardware respectively, but as the usage of data is exploding, this assumption may no longer be applicable.

The problems discussed above led to the following objectives:

(i) To compare the efficiency of Bitmap index as opposed to B-tree index on a column with high cardinality
(ii) To compare the query response time in multi-dimensional queries with the time that is needed to one-dimensional queries on both Bitmap index and B-tree index
(iii) To explore whether query utilizing Bitmap index executed within a range of predicates has any affinity by the cardinality conditions.

3- Summery of Chapters

The present research commences with the statement of the problem that triggered it and a brief review of its underlying theories in the first chapter, introduction that is followed by a review of the related literature and works in the second chapter. It covers the pertinent research on both hierarchical denormalization and index techniques discussing the advantage of incorporating hierarchical denormalization and Bitmap index in DW design. Further, it presents a generally applicable denormalization, hierarchical denormalization process models as logical designing and indexing techniques as physical designing. The chapter ends with a discussion of drawbacks of some related models and provides commonly accepted models and techniques. Chapter 3, Experimental Design, presents an experiment plan with a logical model of hierarchical denormalization introducing an optimized model. Likewise, it puts forth an experiment plan with a physical model by standard datasets introducing appropriate indexing technique. The fourth chapter, Results and Discussion, includes a detailed justification of query transactions used in experiments (in both data modeling and indexing) preceded by the results of all experiments along related discussions. The final chapter is the Contribution and Conclusion in which the contributions of the project are discussed together with its limitations and suggestions for future research.

4- Outcomes

4-1- Indexing techniques as physical designing

The performance measurement experiments can be presented in three main parts as follows:

(i) The index file size
(ii) Index construction time
(iii) Query retrieval time

4-1-1- Index file size and index construction time

The time taken to construct B-tree and Bitmap indexes is shown in Table 1. As it can be observed the Bitmap requires slightly more time to build high-cardinality columns (Product table) as compared with low-cardinality (Sales table) on the same columns. B-tree, on the other hand, requires considerably more time to build all indexes regardless of the column’s cardinalities.
Table 1: Index files size and index construction time

<table>
<thead>
<tr>
<th></th>
<th>Sales</th>
<th>Order</th>
<th>Product</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Size(MB)</td>
<td>Time(S)</td>
<td>Size(MB)</td>
</tr>
<tr>
<td>ID_Bit</td>
<td>326</td>
<td>1580</td>
<td>1222</td>
</tr>
<tr>
<td>Id_Bt</td>
<td>26211</td>
<td>21090</td>
<td>26532</td>
</tr>
<tr>
<td>Name_Bit</td>
<td>418</td>
<td>1673</td>
<td>1341</td>
</tr>
<tr>
<td>Name_Bt</td>
<td>26911</td>
<td>21638</td>
<td>26821</td>
</tr>
<tr>
<td>Active_Bit</td>
<td>288</td>
<td>1544</td>
<td>0.06</td>
</tr>
</tbody>
</table>

Table 1 summarizes the different index sizes over various kinds of data cardinality. In Figure 1, we consider only the size of the two columns on Bitmap and B-tree indexes. For high-cardinality cases, Bitmap generates a large number of small bitmap objects and it takes a considerable time to allocate memory of these bitmaps. Since the index file size of Bitmap index depends on the cardinality of the column, ultimately, the index size on the columns will be smaller than a B-tree even for full cardinality (100% distinct values) on the same column. Table 1 and Figure 1 show that to build index on a large column which is involved by B-tree is prohibitively expensive in terms of space and creation time.

Figure 1: Index file size of bitmap with various cardinality

4-1-2- Query processing time

In this section, we evaluated the time required to answer the queries. These timing measurements directly reflect the performance of indexing methods. A summary of all the timing measurements on several kinds of queries which will be shown in the presentation slides are depicted in Table 2.

<table>
<thead>
<tr>
<th>Tables</th>
<th>Sales</th>
<th>Order</th>
<th>Product</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Low Cardinality</td>
<td>Normal Cardinality</td>
<td>High Cardinality</td>
</tr>
<tr>
<td></td>
<td>Bitmap</td>
<td>B-tree</td>
<td>Bitmap</td>
</tr>
<tr>
<td>Query1A</td>
<td>0.018</td>
<td>0.051</td>
<td>0.019</td>
</tr>
<tr>
<td>Query1B</td>
<td>0.023</td>
<td>0.056</td>
<td>0.023</td>
</tr>
<tr>
<td>Query2A</td>
<td>0.017</td>
<td>0.078</td>
<td>0.017</td>
</tr>
<tr>
<td>Query2B</td>
<td>0.021</td>
<td>0.097</td>
<td>0.024</td>
</tr>
<tr>
<td>Query3A</td>
<td>21.21</td>
<td>113.52</td>
<td>22.12</td>
</tr>
<tr>
<td>Query3B</td>
<td>307.61</td>
<td>1230</td>
<td>308.56</td>
</tr>
<tr>
<td>Query4A</td>
<td>0.081</td>
<td>0.140</td>
<td>0.081</td>
</tr>
<tr>
<td>Query4B</td>
<td></td>
<td></td>
<td>0.044</td>
</tr>
<tr>
<td>Query5C</td>
<td>1.15</td>
<td>5.21</td>
<td>0.92</td>
</tr>
</tbody>
</table>
Table 2: Query response time per seconds

Figure 2 shows the query elapse time for the Product table (table with high cardinality). As it is indicated in this figure, Bitmap index is much faster than B-Tree index. Thus, it can be claimed that Bitmap index is suitable for all levels of column cardinality as shown in Figure 3 where the query elapse time is about constant for each query type.

Figure 2: Query elapse time for Bitmap and B-Tree index on high cardinality

Figure 3: Query elapse time for Bitmap index on various level of column cardinal

4-2- Hierarchical Denormalization (Data Modeling)

In order to compare efficiency of denormalization and normalization processes and analysis the performance of these data modeling, we build a series of queries on some columns for evaluation. In our dataset, there are 4 tables; Fact, D1, D2 and D1D2. Fact, D1 and D2 tables have approximately 1.7 billion of records and D1D2 table (a combination of D1 and D2 tables) has approximately 3.36 billion records. These records were randomly generated using PL/SQL Block by Oracle11G tools. These tables can be categorized into two database schemas, Schema 1 and Schema 2 which are portrayed in Figure 2 and Figure 3 respectively. In Schema 1, the tables were applied by normalization modeling where D1 table is
connected to the D2 table by one-to-many relationship and similarly, D2 table is also connected to the Fact table by one-to-many relationship. In Schema 2, D1D2 table is directly connected to the Fact table by one to many relationships. The D1D2 table is implemented by hierarchical technique. All attributes, except the keys(PK), of the dimensions are associated by Bitmap index; Schema 1 are contained by 5 indexed columns while Schema 2 are contained by 3 indexed attributes.

### Figure 4: Schema1 with normalized design

Schema1 includes a DW system with fact data which is chained with huge amount of data stored in D2 and D1 dimensions (shown in Figure 2) while Schema2 contains the fact table and one dimension table implemented by hierarchy technique (shown in Figure 5).

### Figure 5: Second schema with denormalized design

In order to evaluate the time required to respond to different query types including range, aggregation and join queries; we will briefly describe all of our Selected six SQL queries with 70 stored procedures during the presentation meeting. Basically, for each query, we use suffix ‘A’ to represent query on Schema1 and suffix ‘B’ to represent query on Schema 2.
We present the performance measurement experiments in three main parts as follows:

(i) Hierarchical denormalization effects on one-dimensional modeling 
(ii) Hierarchical denormalization effects on multi-dimensional modeling 
(iii) Bitmap indexing effects on the hierarchical denormalized modeling

4-2-1- One Dimensional

Figure 6 shows the query elapse time for one-dimensional queries which were applied on first and second schemas. This figure shows that although the query retrieval time on the first schema which has been designed by normalization method is faster than denormalized schema; query performance can be enormously enhanced by using index techniques especially Bitmap index technique.

![Figure 6: Query elapse times for one dimension queries](image)

4-2-2- Multi Dimensional

Figure 7 shows the query elapse time for Multidimensional hierarchical Queries which have been applied on first and second schemas. This diagram shows that using hierarchical denormalization method can improve system response time when the queries are unanticipated ad hoc queries.

![Figure 7: Query elapse times for multi dimension queries which are involved by join operations](image)

5- Achieved contributions
This research presented a practical view of indexing technique, normalization, denormalization and proposed hierarchical denormalization with fundamental guidelines to be used in DW design. It clearly portrayed the conventional academic idea of applying bitmap index for low cardinality datasets cannot be considered the best. All identified guidelines need to be given appropriate concentration at the time of initial design.

The outcomes of our experiments provided quite convincing perspectives to practitioners for some reasons: Firstly, the used database and datasets reflected the functionality and multiple aspects of the hierarchical denormalization prototype of a data warehouse. Secondly, several kinds of query instance and data populating the database were chosen to have wide both academic and industry relevance. Finally, in indexing techniques, the performance metrics also presented valuable information regarding the performance enhancement with DWs, which would be most interesting for those who work in a professional area.

Two significant categories of contributions can be concluded from this research:

1. Physical designing:
   1-1- Bitmap index is the conclusive choice for a DW designing no matter for columns with high or low cardinality.
   1-2- The widespread opinion on using Bitmap index and B-tree index in DW should be changed by giving the preference to Bitmap index in most DW designs.

2. Data Modeling:
   2-1- Hierarchical denormalization presents positive effects on DW performance.
   2-2- Using hierarchical denormalization reduces the number of the entities present in Snowflake Schema. Such a reduction will result in a lower relations and joins among the entities that can be a main way to enhance DWs performance.

References:


Inmon. W, Building the Data Warehouse, John Wiley Sons, fourth edition, 2005


Inmon. W. H, Building the Data Warehouse. John Wiley and Sons, 2005


Hanus. M, To normalize or denormalize, that is the question. In In Proceedings of Computer Measurement Group’s 1993 International Conference, pp. 413-423.

Date. C. J, .The normal is so...interesting. Database Programming and Design. 1997, pp.23-25

Strohm. R, Oracle Database Concepts 11g. Oracle, Redwood City,CA 94065. 2007


Menninger. D, Breaking all the rules: an insider’s guide to practical normalization, Data Based Advis. (Jan. 1995), pp. 116-121