Quality Oriented Logical Design for Data Warehouse Development

Munawar, Naomie Salim, Roliana Ibrahim

Abstract—To make an effective data warehouse logical design which is related to system performance, it is important to manage an appropriate methodology for data fragmentation, indexing and materialized view. Executing a query in a data warehouse may take hours or days to run without the proper optimization technique. This paper proposed a framework on integrating quality, deliverables in conceptual design as an input into the logical design, and the optimization technique to run query, to treat as continuous improvement from preceding stage.

Keywords—logical design, data warehouse, indexing, materialized view, fragmentation

I. Introduction

A data warehouse (DW) integrates data from external sources and from internal OLTP (On Line Transactional Processing) to support analytical query processing. This analytical query processing can be used by an enterprise to achieve a great competitive advantage. OLAP (On-Line Analytical Processing) tools represent data in a multidimensional version, enabling business users to formulate queries and perform analysis.

Quality issues raised by DW are crucial. For an effective DW system, the quality aspects should be incorporated properly at the various levels of DW development including in logical design phase.

Logical design is the most attracted phase with strongly impacts to the system performance. It is aimed at deriving out of the conceptual schemata the data structure that will actually implement the data mart or DW by considering some sets of constraints (e.g., concerning disk space or query answering time [22]). During logical design the designer defines which structures will be used to store information and how their performance can be optimized.

An acceptable (or good) DW performance is one of the important features that must be guaranteed for DW users. For this reason, providing means for increasing the performance of a DW for analytical queries is one of the important research and technological areas.

In this paper, we study logical design and practical issues related to the design of multidimensional modeling. The framework for investigation is I-LogiQ (Integrated Logical Design and Quality), a logical model for OLAP systems that extends our earlier proposal [12; 13; 14]. This model includes a number of concepts that optimize the technique to run query using indexing, materialized view, and fragmentation, commonly used in multidimensional database.

The paper is structured as follows. Section 2 describes related research. Section 3 presents the proposed framework. Finally, conclusion is described in section 4.

II. Related Works

Due to the increasing complexity of DWs, continuous attention must be paid for evaluation of their quality throughout their design and development [6]. It is also very important to consider quality issues at various levels of models including logical models. Quality of the DW logical models has been assured by proposing several metrics to evaluate the quality of star schemas at logical level. These proposed metrics were validated theoretically and empirically [17, 16]. [17] have proposed metrics for measuring multidimensional schemas analyzability and simplicity. Nevertheless, the metrics proposed in these approaches have not been empirically validated and consequently, have not proven their practical applications [4]. Recently, [16] proposed a set of metrics for assessing the understandability of DW schemas using structural metrics and also validated theoretically and empirically through a family of experiments.

Our proposed frameworks mainly focus on integrating quality and deliverables in preceding stage (conceptual stage) as an input into the logical design in order to treat as a continuous improvement in the next phase. In a DW where data is processed in stages, and where the quality of data at one stage is dependent on the DQ measurements in preceding stages, DQ can be assessed and monitored continuously in order to guarantee high quality levels. As a result, DQ is not only an integral part of DW project, but will remain a sustained and ongoing activity [12].

III. Proposed Framework

The logical design can be used for many purposes [2]: (i) as an intermediate representation between the conceptual design and the physical design, providing an operational view of the DW without necessarily dealing with performance nor physical representation of data, (ii) as a reference schema from which the physical design starts and to which the benefit of the selected materialized views is balanced, (iii) as a support to control the DW evolution both at its client and source levels.
The logical design of the DW serves to define the structure to ensure an efficient access to information. It can be presented as relational or multidimensional structure that takes as input the conceptual schema representation, the information requirements, the source database and non-functional requirements [26].

In relational implementations, the so-called star, constellation, and snowflake schemata are widely accepted to manage data cubes and are supported by various vendors. Concerning multidimensional implementations, several efficient multidimensional data structures such as condensed cubes, dwarfs, and QC-Trees have been proposed to manage data cubes. Comparison between relational implementation and multidimensional implementation can be seen in Table I, Table II and Table III.

TABLE I. COMPARISON OF RELATIONAL IMPLEMENTATION FOR MULTIDIMENSIONAL MODELING [11]

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Star Schema</th>
<th>Fact Constellation Schema</th>
<th>Snowflake Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>Efficiency</td>
<td>High</td>
<td>High</td>
<td>Moderate</td>
</tr>
<tr>
<td>Usability</td>
<td>High</td>
<td>Moderate</td>
<td>Moderate</td>
</tr>
<tr>
<td>Reusability</td>
<td>Low</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>Flexibility</td>
<td>High</td>
<td>High</td>
<td>Moderate</td>
</tr>
<tr>
<td>Redundancy</td>
<td>High</td>
<td>Low</td>
<td>Low</td>
</tr>
<tr>
<td>Complexity</td>
<td>Low</td>
<td>Moderate</td>
<td>Moderate</td>
</tr>
</tbody>
</table>

Traditional database systems are inadequate for multidimensional analysis since they are optimized for on-line transaction processing (OLTP), which corresponds to large numbers of concurrent transactions, often involving very few records. Conversely, multidimensional database systems should be designed for the so-called on-line analytical processing (OLAP), which involves few complex queries over very large numbers of records. Current technology provides both OLAP data servers and client analysis tools. OLAP servers can be either relational systems (ROLAP) or proprietary multidimensional systems (MOLAP).

Unlike OLTP systems where the logical data schema is hidden underneath an application layer, the logical multidimensional (MD) schema of an OLAP system is directly used by the end user to formulate queries. Thus, the MD schema is crucial as it determines the type of queries the user can formulate. One of the main problems in MD data models occurs when the modeled OLAP scenarios become very large since the dimensionality increases significantly, and therefore, this leads to extremely sparse dimensions and data cubes [22]. The system architecture for DW logical designs can be seen in Figure 1.

The way data are actually stored gives rise to different types of OLAP: relational OLAP (ROLAP), multidimensional OLAP (MOLAP), and hybrid OLAP (HOLAP). The choice of ROLAP or MOLAP should depend on the query complexity and performance. For more complex queries and quicker response times, MOLAP should be used because it stores the data in multidimensional databases (cubes) that provide extensive OLAP capabilities. In ROLAP, on the other hand, the data are stored as relational tables and the ROLAP engine generates MD views on the fly. But the ROLAP model works fine when query complexity is not that high and response time demands are not that great.
A crucial problem related to view materialization is that of accurately estimating the actual cardinality of each view [49; 48]. Since the number of possible views which can be derived by aggregating a cube is exponential in the number of attributes, most approaches assume that a constraint on the total disk space occupied by materialization is posed, and attempt to find the subset of views which contemporarily satisfies this constraint and minimizes the workload cost [43; 44; 45]. If the DW has already been loaded, view cardinalities can be quite accurately estimated by using statistical techniques based, say, on histograms [47] or sampling [46]. However, such techniques cannot be applied at all if the DW is still under development, and the estimation of view cardinalities is needed for design purposes.

Indexing a DW is very tricky [34]. If there are few indexes, the data loads up quickly but the query response time is slow. If there is too many indexes, the data loads slowly and need more storage space but the query response is good. So there is a trade-off between the number of indexes built and response time of queries. Indexing in any database, transactional or warehouse, most often reduces the retrieval time of query results [35]. Different indexing techniques have been developed which are being used in forward data retrieval in DW environments. Brief description of a few indexing techniques is given below.

### TABLE IV. COMPARISON OF INDEXING TECHNIQUES [34]

<table>
<thead>
<tr>
<th>Indexing Techniques</th>
<th>Advantages</th>
<th>Drawbacks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bitmap indexing</td>
<td>Widely used in DW environment [37]</td>
<td>It works pretty slow on high cardinality column data.</td>
</tr>
<tr>
<td></td>
<td>Reduced response time for large classes of ad-hoc query</td>
<td>A modification to a bitmap index requires more work on behalf of the system.</td>
</tr>
<tr>
<td></td>
<td>Reduced storage requirements as compared to other indexing techniques [37]</td>
<td>The concurrency for modifications on bitmap indexes is outrageous.</td>
</tr>
<tr>
<td></td>
<td>Dramatic performance gains on hardware with a relatively small number of CPUs or a small amount of memory [37]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Efficient maintenance</td>
<td></td>
</tr>
<tr>
<td>Cluster indexing</td>
<td>It can be formed dense and sparse to get optimized performance.</td>
<td>If data is not sorted then cost of sorting is also added up.</td>
</tr>
<tr>
<td></td>
<td>Good for range based queries but requires sorted data.</td>
<td>Also insertions often requires reordering of data, so it is costly operation in terms of time and resources in Data warehousing [38, 39].</td>
</tr>
</tbody>
</table>

A materialized view is much richer in structure than an index since a materialized view may be defined over multiple tables, and can have selections and GROUP BY over multiple columns. In fact, an index can logically be considered as a special case of a single-table, projection only materialized view [1]. This richness of structure of materialized views makes the problem of selecting materialized views significantly more complex than that of index selection.
Although indexing can help in providing good access support at the physical level, the number of irrelevant data retrieved during the query processing can still be very high. The horizontal fragmentation aims to reduce irrelevant data accesses [42; 36]. Moreover, fragments can be allocated to data marts if the data in data marts are derived from the warehouse data (i.e. top-down approach). Another advantage of allowing partitioning of a warehouse data is that an OLAP query can be executed in a parallel fashion [36].

Executing an OLAP query in a DW can be very expensive, particularly on large warehouse data, if the data is not modeled properly. Moreover, if OLAP queries need only a portion of data, it is advisable to fragment data so that a set of queries can be executed on each fragment as far as possible, thus query response time can be minimized [36].

In the relational DW context, there are three fragmentation types [42]: vertical fragmentation, horizontal fragmentation and hybrid fragmentation (horizontal fragmentation followed by vertical, or vice versa). Vertical fragmentation splits a relation R into sub-relations that are projections of R with respect to a subset of attributes. It consists in grouping together attributes that are frequently accessed by queries. Vertical fragments are built by projection. The original relation is reconstructed by joining the fragments. Horizontal fragmentation divides a relation into subsets of rows using query predicates. It reduces query processing costs by minimizing the number of local accessed instances. Horizontal fragments are built by selection. The original relation is reconstructed by fragment union.

There are two versions of horizontal fragmentation [42]: primary and derived. Primary horizontal fragmentation of a relation is performed using attributes defined on that relation. This fragmentation may reduce query processing cost of selections. Derived horizontal fragmentation, on the other hand, is the fragmentation of a relation using attribute(s) defined on another relation(s). In other word, the derived horizontal fragmentation of a table is based on the fragmentation schema of another table(s).

<table>
<thead>
<tr>
<th>Indexing Techniques</th>
<th>Advantages</th>
<th>Drawbacks</th>
</tr>
</thead>
</table>
| Hash-based indexing | • It reduces a large amount of data down to a reasonable number by transforming it through a hash function and indexing that number [40].  
• It reduces the average lookup cost by a careful choice of the hash function, bucket table size, and internal data structures [41].  
• It does not need the key to be sorted ordered.  
• Hash-based indexing technique is best for equality selections. | • Bad hash function leads to collision.  
• Hash-based indexing technique cannot support range queries.  
• Static hashing can leads to long chains.  
• Reverse of hash is still impossible. |

### References


### Conclusion

In this paper, we have proposed a framework based on logicial model for integrating data quality dimensions and deliverables in preceding stage (conceptual stage: dimensional fact model and class diagram) as an input in order to treat as a continuous improvement in logical stage. In a DW where data is processed in stages, and where the quality of data at one stage is dependent on the DQ measurements in preceding stages, DQ can be assessed and monitored continuously in order to guarantee high quality levels. A key step in logical design is optimization technique to run query among data fragmentation, indexing and materialized view. This paper shows how to choose the proper technique to run query effectively.