Optimization OLAP Design: Modern Principles and Challenge

Kothari Badal K.[1], Harish Nagar[2], Dr.Ashok R. Patel[3]

[1] *Research Scholar, Mewar University, Chittorgarh, Rajasthan, India*

ABSTRACT

In This paper our work is that demonstrates how one would construct a reliable and efficient storage engine for a contemporary OLAP server. We have discussed in detail how this integration would be performed in a practical environment. In addition, we have described how dimension tables and fact tables are encoded into a more compact integer format. Furthermore, we have explained how the storage of non-hierarchical attributes as a set of compressed Fast Bit bitmap indexes can be used to enhance OLAP analysis. We also described how the data of hierarchical attributes are stored and accessed in order to allow the efficient creation of the Sidera map Graph at run time.

Keywords- *OLAP, DBMS, AI,Knowledge.*

---

1. INTRODUCTION

While commercial OLAP systems may provide many functions, there is a minimal set that can and should be defined by any OLAP application.

- Pivot. This OLAP operation allows users to re-organize the axes of the cube. Pivot deals with presentation. provides a simple example of how the pivot operation works in practice.
- Slice. This is an operation whereby we select a subset of a multi-dimensional array (or cube) corresponding to a single value for one dimension member. This operation allows the user to focus in on values of interest. The process for a single value of the "colour dimension".
- Dice. The dice operation is a slice on more than two dimensions of a data cube. The user can draw attention to meaningful blocks of aggregated data; we show a multi-dimensional sub-cube of a larger cube space.
- Roll-up. This is a specific analytical technique whereby the user navigates among levels of data ranging from the most detailed (down) to the most summarized (up) along a concept hierarchy. Illustrates how the "colour dimension",.
- Drill down. This is a specific analytical technique whereby the user navigates among levels of data ranging from the most summarized (up) to the most detailed (down) along a concept hierarchy. Figure 1.4 shows how the "item dimension" is broken down to its item number

2. RELATED WORK

For more than 30 years, Structured Query Language (SQL) has been standard for data access within the relational DBMS world. Because of its relative age, however, numerous attempts have been made to modernize database access mechanisms [2]. Two themes in particular are noteworthy in the current context. In the first case, Object Relational Mapping (ORM) frameworks have been used to define type-safe mappings between the DBMS and the native objects of the client applications. With respect to the Java language, industry standards such as JDO (Java Data Objects) . as well as the open source Hibernate framework [3] have emerged. In all cases, however, it is important to note that while the ORM frameworks do provide transparent persistence for individual objects, additional string-based query languages such as JDOQL (JDO), or HQL (Hibernate) are required in order to execute joins, complex selections, sub-queries, etc. The result is a development environment that often seems as complex as the model it was meant to replace.

More recently, Safe Query Objects (SQO) [8] has been introduced. Rather than explicit mappings, safe queries are defined by a class containing, in its simplest form, a filter and execute method. Within the filter method, the developer encodes query logic (e.g., selection criteria) using the syntax of the native language. The compiler checks the validity of query types, relative to the objects defined in the filter. The execute method is then rewritten as a JDO call to the remote database. The approach is quite elegant, though it can be difficult to accurately model completely arbitrary SQL statements.

In terms of OLAP and BI specific design themes, most contemporary research builds in some way upon the OLAP data cube operator [7]. In addition to various algorithms for cube construction, including those with direct support for dimension hierarchies [10], researchers have identified a number of new OLAP operators [11], each designed to minimize in some way the relative difficulty of implementing core operations in "raw SQL". There has also been considerable interest in the design of supporting algebras [6]. The primary focus of this work has been to define an API that would ultimately lead to transparent, intuitive support for the underlying data cube. In a more general sense, these algebras have identified the core elements of the OLAP conceptual data model.

3. OBJECT ORIENTED OLAP QUERY

Given the relational model of the underlying DBMS, BI querying typically relies upon non-procedural SQL or one of its proprietary derivatives. Unlike transactional databases, however, which are often cleanly modeled by a set-based representation, the nature of BI/OLAP environments argues against the use of such languages. In particular, concepts such as cubes, dimensions, aggregation hierarchies, granularity levels, and drill down relationships map poorly at best to the standard logical model of relational systems. Moreover, the difficulty of integrating non-procedural queries languages into application level source code can be significant. Larger development projects typically encounter any of several associated limitations.

Given the above, Sidera provides a clean integration between the server's Object-Oriented OLAP (OOP) conceptual model (discussed in the previous subsection) and the OLAP client query language. In other words, the OLAP client queries can be specified in any Object Oriented Language such as Java or C#, with the programmer assuming that all OLAP data (cubes, dimensions, etc.) is stored in the local memory as a series of one or more cube objects.
The new Sidera server provides a source code re-writing mechanism that interprets the client's OOP OLAP query specification and decomposes it into the core operations of our comprehensive OLAP algebra. These operations are given concrete form within our OLAP query grammar and are then transparently delivered at run-time to the backend analytics server for processing. Note that the pre-processing work conducted on the client-side of the Sidera server — For example, defining Java libraries, parsing source code and generating XML versions of users' queries — is being performed by another grad student in our research group. The processing described in this thesis begins once the user's query is received on the backend Sidera server.

4. OLAP QUERY PROCESSOR

In This paper , we describe an OLAP query processor that efficient parses and executes the OLAP queries discussed in the previous chapter. The parsing and execution of these OLAP query is of course contingent upon the data storage engine — com- posed of Sidera, the Berkeley DB and FastBit components. Recall that the new Sidera server provides a "native language" query facility that enables one to support native, client-side OOP querying without the need to embed an intermediate, non-OOP language such as SQL or MDX. Sidera provides persistent transparency via a source code re-writing mechanism that interprets the developer's OOP query specification and decomposes it into the core operations of our OLAP algebra. These operations are given a concrete form within the OLAP grammar and then transparently delivered at run-time to the backend analytics server for processing. In this chapter, we focus on an OLAP query processor that includes a set of components for the efficient resolution of user-specified OLAP queries [7, 8, 9].

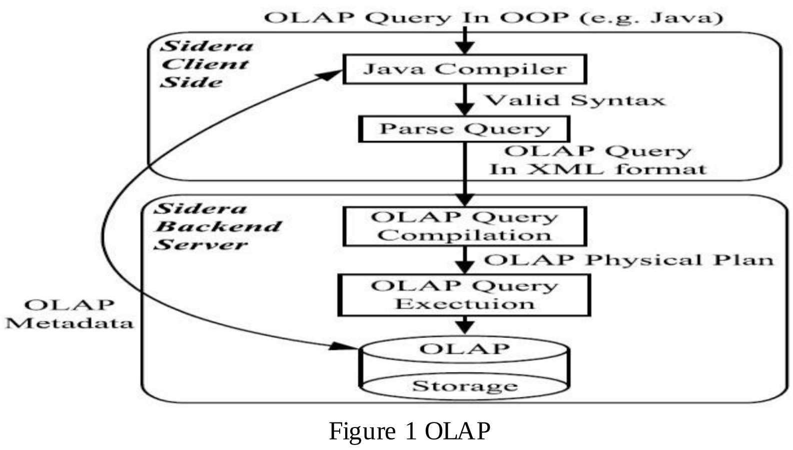

Figure 1 OLAP

The basic process functions as follows. The user defines their query in a completely object-oriented manner. From here, the query is then compiled on the client's side, while the native compiler verifies its syntax. Then, the query is parsed and converted into a second query that corresponds to our robust OLAP query grammar. After that, the query is written in an XML format and sent to the backend server. There, we must be able to interpret and execute the query. We note at the outset that the work conducted on the client-side of the Sidera server — for example, the Java Library API and source code parsing is being performed by another student. Our focus in this chapter, therefore, is related to the components of the OLAP query processor (i.e. the query compiler and execution) which are both found in the backend server. Moreover, in this chapter, we discuss how the aforementioned XML OLAP query is interpreted and executed on the backend server in an manner. Describes our OLAP query processor in terms of the basic steps that must be taken inside the backend server in order to parse, optimize and execute a query. The components of the query processor are:

1. OLAP Query Parser: builds a tree structure from the received XML OLAP query.
2. OLAP Query Translator: turns the parse tree into an OLAP expression tree composed of our OLAP algebraic operators.
3. OLAP Query Optimizer: transforms the OLAP expression tree of step 2 into the best physical query plan to be executed against the actual data.
4. OLAP Query-Execution Engine: takes a query-evaluation plan, executes the given plan and finally returns the answers to the user.

5. THE PRE-PROCESSOR: SEMANTIC CHECKING

The tasks of the parser in the previous section are to take an XML OLAP query, to convert it to a parse tree and to check if it is syntactically valid. Even if the query is syntactically valid, however, it may violate one or more semantic rules on the use of names, expressions, etc. The pre-processor is responsible for semantic checking. In short, it must check the OLAP query against the OLAP schema dentition as follows:

1. Cube name. Every cube mentioned under the <cube name> element must be a cube in the OLAP schema against which the query is executed. For example, there would be a semantic error within the parse tree o if the cube name Order does not exist in the OLAP schema.
2. Dimension uses. Every dimension name mentioned under any OLAP operations such as selection, projection, etc. must be a dimension in the schema of the cube mentioned under the cube name. For instance, the Order cube must have Product, Customer, and Time dimensions; otherwise a semantic error would be produced.
3. Attribute uses. Every attribute that is mentioned under the dimension must be an attribute of that dimension. Likewise every attribute under the <measure name> must be defined as a measure attribute in the cube specified under <cube name>. For example, , Type must be an attribute of dimension Product in cube Order and Quantity Ordered must be defined as a measure attribute in cube Order.
4. Compatibility. UNION, INTERSECT, and DIFFERENCE operations are applied between two OLAP queries. The results of two OLAP queries generated by any of the three above mentioned operations must be compatible. They must have the same number of attributes (features and measures), while each corresponding pair of attributes should have the same domain.
5. Hierarchy uses. Every dimension name that is used under the change level operation must have at least one hierarchy. One must also be certain that the target level is a valid level of the hierarchy.
6. Types. All attributes that are mentioned in the condition of the selection operation must be of a type compatible to their use. For instance, is used in the > (Greater Than) comparison. Since the attribute Age is of type integer, we must ensure that the other operand is also a numeric type.
7. Operation uses. Each operation can appear at most once under the <data query> element. The projection should exist under any <data query> because it determines the schema of the output result. The CHANGE LEVEL and the CHANGE BASE cannot be used within the <data query> due to the fact that they are applied to a result set.

6. CONCLUSIONS

The research presented in this paper describes a number of the core elements one would require if implementing a robust, high performance OLAP-specific data management system. We have discussed the motivation, design, implementation, and evaluation of our research and have emphasized their application to practical query environments. For query compilation purposes, we use a process similar to that of traditional relational database systems in that we use two main approaches to optimize queries (rule-based and cost-based). In our OLAP environment, an internal OLAP parse tree representing the user's OLAP query is created.

7. REFERENCES

[1] S. Agarwal, R. Agrawal, P. Deshpande, A. Gupta, J. Naughton, R. Ramakrishnan,and S. Sarawagi. On the computation of multidimensional aggregates.Proceedings of the 22nd International VLDB Conference, pages 506–521, 1996.

[2] Alfred V. Aho and Jeffrey D. Ullman. Data Structures and Algorithms. Addison- Wesley, 1983.M. O. Akinde and M. H. Bohlen. Efficient computation of subqueries in complex olap. In International conference on Data Engineering (ICDE), pages 163–174, 2003.

[3]C. Bauer and G. King. Java persistence with hibernate. Manning Publications Co., Green-wich, CT, USA, 2006.

[4] R. Bayer. Binary b-trees for virtual memory. Proceeddings of 1971 ACMSIGFIDET Workshop on Data Description, Access and Control, 1971.

[5] N. Beckmann, H. Kriegel, R. Schneider, , and B. Seeger. The r-tree: an efficient and robust method for points and rectangles. ACM SIGMOD, 1990

[6] K. Beyer and R. Ramakrishnan. Bottom-up computation of sparse and iceberg cubes. Proceedings of the 1999 ACM SIGMOD Conference, pages 359–370, 1999.

[7] L. Cabibbo and R. Torlone. Querying multidimensional databases. Proceedings of the 6th DBLP Workshop, pages 253–269, 1997.

[8] S. Chaudhuri and U. Dayal. An overview of data warehousing and OLAP technology. ACM SIGMOD Record, 26:65–74, 1997.

[9] S. Cohen, W. Nutt, and A. Serebrenik. Rewriting aggregate queries using views. in PODS, pages 155–166, 1999.

[10] F. Dehne, T. Eavis, S. Hambrusch, and A. Rau-Chaplin. Parallelizing the datacube. International Conference on Database Theory, 2001.

[11] E.Zimanyi E. Malinowski. Hierarchies in a conceptual model: From conceptual modeling to logical representation. Data & KNowledge Engineering, 2005.

[12] T. Eavis and R. Sayeed. High performance analytics with the r3-cache. Data Warehousing and Knowledge Discovery (DaWak), 2009.