Abstract

Although data warehouses are viewed as organized, summarized repositories of time-oriented data conceptually, the physical implementation determines the speed, efficiency, scalability, and extensibility of this view. Two major physical implementations exist today: data warehouses built upon relational database management systems (ROLAP) and warehouses built upon proprietary multi-dimensional databases (MOLAP). Both ROLAP and MOLAP have their own advantages and disadvantages due to their physical implementation. This paper presents another physical implementation using an object-oriented database or persistent objects—Object Oriented On-line Analytic Processing (O3LAP)—as a possible alternative, compares the O3LAP model with the current models, suggests possible extensions to the current OLAP models, defines the elements involved in the mapping of a logical model to the physical one, illustrates queries based on the O3LAP model, and discusses areas for future research.

1 Introduction

A data warehouse (DW) is a centralized repository of summarized data with the main purpose being to explore the relationship between independent, static variables, dimensions, and dependent, dynamic, variables facts or measures.

There has been a trend within the data warehousing community towards the separation of the requirements for preparation and storage necessary to analyze the accumulated data and the requirements for the exploration of the data with the necessary tools and functionality needed [e.g., Thompson, 1997].

In terms of the storage requirements, a convergent trend has been towards a multi-dimensional hypercube model (e.g., see Argawal, 1997)). In terms of analysis and the tools required for On-Line Analytic Processing (OLAP), there is a convergent trend towards standardizing this as well; e.g., the OLAP Council’s Multi-Dimensional Application Programmers Interface (MD-API) [OLAP Council].

Although the trends have been to separate the storage from the analysis, the actual physical implementation of a DW/OLAP system reconnects them. This is evident from the parade of acronyms used today, e.g., ROLAP, MOLAP, DOLAP, HOLAP, etc., where each physical implementation determines the advantages and disadvantages of storage access and analysis capabilities and also determines any possible extensions to the model.

Of the models cited above, the two most common in practice are the Relational On-line Analytic Processing (ROLAP) model and the Multidimensional On-line Analytic Processing (MOLAP) model.

The major advantage of ROLAP, which depends on relational database (RDB) technology, is that the database technology is well standardized (e.g., SQL2) and is readily available off-the-shelf. This allows for the implementation of a physical system that is based on open standards and has readily available technology. As this technology is well studied, there are mechanisms which allow for authorization schemes and for transactions, thus allowing for multi-user systems with the ability to update the data as necessary. The disadvantage of this technology is that the query language as it exists (SQL) is not sufficiently powerful or flexible enough to support true OLAP capabilities [Thompson, 1997]. Furthermore, there is an impedance problem in that the results returned, tables, always need to be converted to another form before further programming capabilities can be performed.

The major advantages of MOLAP, which depends on usually proprietary multi-dimensional (MDD) database technology, are based on the disadvantages of ROLAP and is the reason for its creation. MOLAP queries are very powerful and flexible in terms of OLAP processing. The physical model more closely matches the multidimensional model, and the impedance problem is remedied within a vendor’s domain. Nonetheless, there are disadvantages to the MOLAP physical model: 1) There is no real standard for MOLAP; 2) there are no off-the-shelf MDD databases per se; 3) there are scalability problems; and 4) there are problems with authorizations and transactions.

As the physical implementation ultimately determines the capabilities of the system, it would be advised to find a technology that combines and maximizes the
advantages of both ROLAP and MOLAP while at the same time minimizes the disadvantages. In this paper we seek to do just that. The physical implementation chosen and discussed is that of an object-oriented database (OODB) or persistent objects.

This paper will show that through an Object-Oriented On-Line Analytic Processing (O3LAP) framework, the advantages of MOLAP and ROLAP are combined and the disadvantages are minimized. Furthermore, through the use of O3LAP and through the use of object-oriented (OO) concepts applied to data warehousing, the capabilities of OLAP can be further extended.

What this paper contributes is a framework for object-oriented storage, retrieval, and manipulation based on open object technologies and thus provides a well-defined, readily available, and extendable technology.

2 Framework for Object-Oriented On-Line Analytic Processing

This section will discuss the advantages of the OODB model in relation to the RDB and MDD physical implementation, how OO concepts extend the OLAP model, the elements of mapping a logical schema to a physical one, and the classification of different O3LAP classes.

2.1 Advantages of using an OODB

The use of an object-oriented database management system (OODBMS) or even the use of persistent objects as the physical implementation of the Object-Oriented On-Line Analytic Processing system allows the retention of the advantages of ROLAP and MOLAP while presenting few of their disadvantages.

Like ROLAP, object-oriented databases are well standardized via the work of the Object-Oriented Database Management Group (ODMG) under the auspices of the Object Management Group (OMG) [Cattell, 1997]. There are numerous vendors of such databases, and object persistence can be easily implemented with utilities from companies such as ObjectStore [pseprof.objectdesign.com]. Also, as there has been much research into the area of OODBMSs, the issues of user authorization and database updates via transactions are well studied. Like MOLAP, the queries are flexible and powerful, and the problem of impedance mismatch is dispensed with by the use of query extensions to an object-oriented programming language.

Other advantages of using an OODB physical model applied to a data warehouse are also possible. Versioning, which is easily implemented in an OODB, can support dimensions and facts that change over time and also allow for the incremental development of the warehouse. Also, as there is a well-defined distributed object model, the Common Object Request Broker Architecture (CORBA), this makes distributed data warehouses more easily implemented and integrated.

2.2 Extending OLAP

Although there are benefits from physically implementing a Data Warehousing/On-line Analytic Processing system as shown above, additional benefits can also be gained by allowing OO concepts to be applied to the traditional OLAP model.

Through the use of OO concepts applied to a DW, the traditional analysis of numerical data can be extended to other data types. This is due to the fact that objects support the encapsulation of data with their associated display and manipulation methods. One example of such an extension could be a data warehouse for a clinical trial complete with x-ray (two-dimensional graphics) and dose delivery data (three-dimensional location and dosimetric data), as well as traditional patient data (age, gender, etc.). Other possibilities are the implementation of a genome data warehouse, a bibliographic data warehouse, or even a pictorial or sound bite data warehouse.

Finally, the use of OO concepts applied to the traditional DW/OLAP elements of dimensions, facts, and queries, allow for a richer implementation and will be illustrated and intertwined within the discussion of the mapping of the logical model shown in the next section.

2.3 Mapping the logical to the physical model

In terms of logical modeling, a convenient modeling tool for a multidimensional model has been the star schema and its variants [Kimball, 1996]. It can model simple dimensions with elements, e.g., store name, store type, etc., facts, e.g., amount of sales, and hierarchical dimensions with levels, where a dimension has hierarchical dimensional elements, e.g., cities within states within countries. Facts are linked with dimensions by the grain of the model (e.g., sales is associated with the store by the week).

The example used will be that of a sales and marketing system to analyze product sales to customers through distribution channels, similar to the benchmark provided by the OLAP Council [OLAP Council]. This example will focus on two dimensions, Product and Customer, and one fact table, Sales. Customer is a hierarchical dimension and has two levels, Retailer and Store. The fact table is associated with (the grain of) the group attribute of the Product dimension and the Store level of the Customer dimension. The associated star schema is illustrated in below:
In describing the mapping the focus will be on five major elements: dimensions, facts, extents, queries, and object identifiers.

2.3.1 Dimensions

The first element to be mapped is that of the dimension. In its simplest translation, each dimension is mapped to a class, with each dimensional element mapped to an attribute. However, we also make the distinction between simple dimensions, those containing no additional information concerning the dimensions, such as Product in the example with four elements, and more complicated dimensions which contain additional information or an explicit hierarchical definition, such as Customer. As one can see, within Customer there are two hierarchical dimensions, also known as levels, which also have additional information. With this in mind, we define the following:

**Dimension non-associative classes**: dimension tables that do not have additional information about a dimensional element, as in a simple star schema.

**Dimension associative classes**: dimension tables that do have hierarchical information or additional information about a dimensional element, as in a snowflake schema [Kimball, 1996].

Given this classification, the first category, non-associative classes, are mapped as described above: dimensional elements become simple attributes. The second category, associative classes, are mapped such that each level element within a dimension becomes a separate class and the hierarchy between the levels is represented as additional attributes (e.g., parent, child). The dimension itself is mapped to a class where it serves as the root of a hierarchical tree with links to the first level below and to a special parent, TOP, which represents null.

An example of mapping a dimensions to a classes using a Java code fragment is given below:

```java
//construction of non-associative class
class Product {
    //dimensional elements
    private String division;
    private String line;
    //etc.
    //simple constructor
    Product(String d, etc.) {
        division=d;
        //etc.
    }
    //simple accessors
    public String getDivision() {
        return division;
    }
    //simple mutators
    //etc.
}

//construction of associative class
class Customer {
    private Top top;
    private Retailer retailer;
    //etc.
}
```

Since dimensions are represented as classes, there are advantages that can be gained: 1) a dimension can have associated methods, such as a Store class can have the method, changeRank(), as now changes/updates can be made by the analysis to the warehouse due to transactional ability with OODBs; 2) it allows for richer data types, such as the Logo graphical class illustrated above, to allow users to browse visually; 3) it allows for the specialization of dimensions so that general dimensional classes can be defined for an organization and subclasses of those dimensions can be developed as required for the different data marts within the organization; and 4) class methods and attributes can be associated with the different levels / dimensions which allow for statistics, such as the number of different retailers that exist within a retailer dimension.

2.3.2 Facts

Facts are also mapped to classes. By the nature of facts, however, every fact is an associative class, as each fact is associated with the grain of the data warehouse.

An example of mapping a fact to a class using a Java code fragment is given below:

```java
public class Store {
    //informational attributes
    private String storeName;
    //etc.
    //link to parent
    private Retailer retailer;
    //constructor, accessors, mutators, etc.
}

//construction of associative class
class Retailer {
    //informational attribute
    private String retailerName;
    //Logo could be a graphical class
    //defined elsewhere
    private Logo companyLogo;

    //links
    private Customer[] customer;
    private Store store;
    //constructor, accessors, mutators, etc.
}

//construction of root/dimension class
class Customer {
    private Top top;
    private Retailer retailer;
    //etc.
}
```

Public class Sales {
    //associated grain
    private Product product;
    private Store store;
    //etc.
}
As there were advantages associated with dimensions as object classes, so too are there with facts: 1) methods allow for computed attributes, such as totalSales(), as illustrated above; 2) subtyping could allow for additional measures which perhaps change more frequently, thus allowing for different update schedules; 3) there can be specialized class methods and attributes associated with the facts, manipulation classes, which allow for non-additive or non-arithmetic aggregation, as well as more sophisticated statistical routines.

2.3.3 Extents

Normally in object-oriented programming, objects instantiated are transient—they no longer exist when the program has completed. What is required for Object-Oriented On-Line Analytic Processing is persistent objects. One way this persistence can be provided is through the use of extents or root objects

Since there may be millions of objects collectively defined as facts, it is important to be able to refer to them collectively rather than individually. Extents also provide this collective naming.

Extents and root objects, then, are associated with databases and provide permanence to selected objects and also provide the ability to collectively refer to a large set of similar objects. Other objects associated with objects associated with the root are also made persistent, and this is known as persistence through reachability [Khoshafian, 1993].

We define the term set to mean a collection of unordered, unique objects of the same object class and define collection or container objects as a persistent or transient set of objects or a set of other collection objects.

Manipulation classes are classes with class methods that can be applied to collections. This allows for the grouping of operations such as statistical or data processing in one class and also affords the possibility of subtyping to allow for additional or specialized functionality as required.

In the examples that follow there will be an extent associated with the Product class, AllProduct, two extents for the two-level Customer dimension, AllRetailer and AllStore, and one for the fact dimension Sales, AllSale. HS is a manipulation class that contains some statistical functionality associated with the AllSales extent. Finally, container objects are created as the results of queries run on extents or on other collections.

Having defined persistent sets, collections, and manipulation classes, the next step is to define queries on these entities.

2.3.4 Queries

Queries operate on extents or collection classes. Queries are the elements that compose OLAP operations. Object-oriented queries are simply paths through the hierarchy defined by the associative dimensions and facts. (Non-associative classes do not use any path navigation and are discussed below.) Path navigation replaces the normal multiple joins involved with relational model. This is an advantage, as multiple joins, as required in ROLAP systems, are slow and resource intensive [Patel, 1998]. Paths are similar to the traversal of doubly-linked lists.

Queries are run against sets and the results are also sets which can become permanent (named) or transitory. Queries can be directed towards the dimensions, against the facts, against the facts with constraints on the dimensions or vice versa. The Object Query Language (OQL) is rich and well-defined and provide set operations such as UNION, INTERSECTION, IN, etc. [Catell, 1997] which are vital for OLAP. For purposes of illustration, we will focus on a Java-like OQL language which assumes that 1) queries can be run against collections or extents; 2) set operations, such as UNION (AND) which allow duplicates to be eliminated, exist; and 3) equality with collections or extents implies an IN set operator.

2.3.4.1 Simple Queries

Simple queries are simply single queries against the dimensions or facts without Boolean operators.

A query returns dimension objects if the query is directed towards the dimension root object. For instance, a query which wishes to find all stores with space greater than 5,000 square feet would be

(1) AllStore.getSqFoot() > 5000

A query returns fact objects if the query includes the fact extent. For instance, given the non-associative Product dimension, to find the facts associated with the sales of Group X, one would simply find the set of facts that have “X” as the value of the associated group, and retrieve those stores:

(2) AllSale.getGroup() == “X”

A query may also may be run across the fact tables using either the normal or computed attributes. For
instance, to find the sales facts that have a total sales > 50,000:

(3) AllSale.getTotalSale() > 50000

Applying the manipulation class HS associated with AllSale facts allows for statistical calculations. Given the associative dimensional hierarchy Customer, to find the median sales for all stores associated with Retailer Y, the query is:

(4) HS.median(
    (AllSale.getStore().getRetailer() == "Y").getSales())

2.3.4.2 Compound Queries

Additional constraints on the dimensions involve the use of Boolean / set operators.

(5) XandY =
    AllSale.getStore().getRetailer() == "Y"
    AND
    AllSale.getGroup() == "X"

Queries can be run against collection classes and thus return further reduced collection classes, allowing for additional queries or application of manipulation class functionality. For instance, given the named collection XandY above, if the top three sales were required, the query is:

(6) HS.topN(XandY, 3)

2.3.4.3 Queries as classes

If we consider the elements of a simple query based on a dimensional class: 1) {extent, dimension, path}.attributes; 2) an operator {<,>, etc.}; 3) a {constant} or Item 1; and, 4) a Boolean constructor {AND, OR, etc.}, then a simple query can be represented as a query class with those attributes. A complex query is a class which is simply the aggregation of all the simple queries involved. Queries that are frequently issued could be represented as pre-computed persistent objects.

2.3.4.4 Uniqueness of facts returned

If we assume that the “=” operator is actually an IN set operator, then the question of unique retrieval comes into play; i.e., are we multiple counting in some instances. By definition, two facts cannot have different values at the grain link, so multiple ORs on this field will yield different fact objects. If the OR is between different levels within the same dimension, then it would be possible to have the same object returned, e.g., State = “PA” OR city = “Philadelphia.” In this case, it is important for the query to be “common denominatorized” by using the lowest level within each dimension which would then yield different fact objects. The AND operator makes no sense within a dimension as something cannot be more than one thing at one time. The AND operator across dimensions will yield unique fact objects also by definition of the grain. The OR operator across dimensions could possible yield non-unique objects and the need for efficient uniqueness checks needs to be explored.

2.3.4.5 Classification of OLAP classes

Yourdon’s methodology of object-oriented systems analysis separates object classes into three categories: data, control, and interface objects [Yourdon, 1995]. This separation can also be applied to our scenario and makes perfect sense to do so. The data objects are the dimensions and the facts. The control objects are queries, OLAP operations, and manipulation classes. The interface objects make human-readable the results of the control classes against the data classes.

2.3.4.6 Definition of OLAP operations in terms of queries

As mentioned previously, queries become the elements of which OLAP operations are composed. The authors have observed that most OLAP operations are simply restrictions on the dimensions. As a consequence, we define the familiar terms slice, dice, and pivot as follows: slice is a restriction on one dimension, dice is a restriction on two or more, and pivot changes the spatial relationship between dimensions. Based on these definitions, a slice is an example of Query (1) and dice is an example of Query (5).

The selection of the parameters of the queries should be supplied by the user via direct manipulation of interface objects. Familiar objects such as text can be used, as in a package such as Brio [www.brio.com], or as was suggested previously, with graphical objects such as companyLogo and maps instead of state / city names. Since pivoting implies a visual reference, the definition of pivoting is then simply a dynamic manipulation of the interface objects.

Drill-down and roll-up also imply a restriction on the dimensions, and this is echoed by Kimball, who states: "Drilling down in a data warehouse is nothing more than adding row headers from the dimension tables" [Kimball, 1996]. However, this may or may not be exactly true within our framework and is dependent upon whether the operation is directed towards an associative or non-associative dimension class.

This leads to the following observations: (1) non-associative classes have no explicit hierarchy, offer no true (path) navigation, and operate by restrictions on attributes within the non-associative object; whereas (2) dimension associative classes have an explicit hierarchy, offer true (path) navigation, and traversal up/down is achieved by a hierarchy path based on linking attributes or methods, but (3) it is possible to define a hierarchy based on non-associative classes.

If the non-associative classes are based on a hierarchy, then an object exists for each unique leaf node within the hierarchical tree with the individual path as part of its attribute list, whereas, the associative classes use paths for traversing the individual paths. As such, it can be conjectured that the speed and efficiency of the
simple non-associative hierarchical classes would be much greater than that of the associative classes. This is due to the fact that when querying non-associative classes, the whole set of objects is searched and the returned objects contain all the information concerning the parent/child relations, as opposed to finding the objects and traversing the paths. This is true in our example: the Product dimension consisted of elements that defined an implied hierarchy. This does suggest, however, that a combination of the two class types, a non-associative class consisting of associative classes could be defined and used to define multiple named hierarchies. For instance, a hierarchy which is composed of region, state, city, store can be composed to link the state directly to the store, skipping the city, if the proper objects were instantiated.

2.3.5 Object Identifiers

One of the major flaws of object-oriented databases pointed to by many is the problems with the unique object identifier (OID). As variables in memory are differentiated with a memory address, tuples in RDBMs are differentiated with primary keys, so too must objects be differentiated on something other than a key or memory address [Khoshafian, 1993]. The problem is exacerbated as the OID must be unique for each object and, since it cannot be reused when an object is deleted, must be sufficiently large to handle all the possible objects. Since we are interested in a persistent distributed database scheme, this is amplified.

However, the OID generation need not be simply a unique, system-generated 5-byte number without any significance. There are many other OID generating schemes, for instance, see [Khoshafian, 1993]. One scheme that is of particular interest is that of the OID which not only uniquely identifies the object but also has information within the identifier to indicate where it is located. A rare example would be the naming convention proposed by Sun to identify all Java classes with the hostname, node name, etc. This would come at a price in terms of bits required, but does allow for the possibility for a truly distributed warehouse. Furthermore, as also discussed in [Khoshafian, 1993], there can be a surrogate OID when the object is in memory which would allow certain objects to be located easily in memory; for instance, the dimension objects could be allowed to reside in memory for fast processing.

3 Conclusion and Future Research

Although the trend is the separation of the storage component from the analysis component in data warehouses, regardless of the theory, the physical implementation ultimately decides the reality. Moreover, the physical implementation circumscribes that which is possible in terms of extensions to the existing OLAP model.

It was suggested that the O3LAP model so defined is that of a hybrid of both ROLAP and MOLAP, with many of the advantages and with few of the disadvantages.

In terms of ROLAP, the similarity can be seen of the tuple and foreign key with the object and the linked attribute. In terms of MOLAP, it can be seen that the tree structure suggested by [Colliat, 1996] is similar to path traversal, especially when the dimensions remain in memory and the subcubes pointed to by the leaf nodes in MOLAP are similar to the facts pointed to by the grain (leaf) objects if they are clustered [Khoshafian, 1993] in a intelligent way so that similar facts remain close on a physical device, such as clustering by the time grain.

With objects instead of cubes or tables, it allows for different types of data, i.e., non-numeric, to be warehouse and searched. With the standards in place, it is easy to build such a warehouse off the shelf. Naturally, there need to be further work on defining generalized classes as opposed to the simple examples given in this paper (e.g., class customer implements AssociativeClass) and it is the plans of the authors to do those extensions.

There is some question of the scalability of the design and whether it would be sufficient for the data warehouses sizes that are in use today. Indeed, as was mentioned in a previous section, the number of “Booleaned” dimension categories and the determination of the unique objects returned may make the design unfeasible for larger implementations. We will explore these scalability issues in the future to see if the conjectures are true. Currently, however, for smaller scaled data warehouses, especially for those based on non-traditional, non-numeric fact models, it is hoped that O3LAP could have a definite place.

Bibliography


Khoshafian, Setrag, Object-Oriented Databases, John Wiley and Sons, 1993.

Kimball, Ralph, The Data Warehouse Toolkit, John Wiley and Sons, 1996.


