Multidimensional models - Constructing DATA CUBE

Antoaneta Ivanova, Boris Rachev

Abstract: *The goal of this article is to depict algorithms and new approach of creating data cube efficiently and to set tasks for future work. In short, the paper has described it as a data abstraction that allows one to view aggregated data from a number of perspectives: 1) Conceptually, the cube consists of a base cuboid, surrounded by a collection of sub-cubes that represent the aggregation of the base cuboid along one or more dimensions; 2) The basic methods for computing a group-by: sort-based and hash-based. 3) Dynamic Data Cube, which provide efficient performance and allows for the data cube expansion in any direction. 4) OO Conceptual Model Data Cube – class definition of data cube.*

Key words: *data warehouse, data cube, OLAP, MOLAP, multidimensional data model*

INTRODUCTION

Aggregation is predominant operation in decision support database systems. On-Line Analytical Processing (OLAP) databases often need to summarize data at various levels of detail and on various combinations of attributes.

The data cube, also known in the OLAP community as the multi-dimensional database.

A *data cube* [2],[6],[7] is constructed from a subset of attributes in the database. Certain attributes are chosen to be *measure attributes*, i.e., the attributes whose values are of interest. Other attributes are selected as *dimensions* or *functional attributes.* The measure attributes are aggregated according to the dimensions.

The Figure 1 below depicts a small, practical data cube example, consider a hypothetical database of sales information maintained by a company. This particular data cube has three feature attributes - *store, product*, and *time* - and a single measure attribute – product *sales* for a large chain of stores (sales is computed with the sum function).

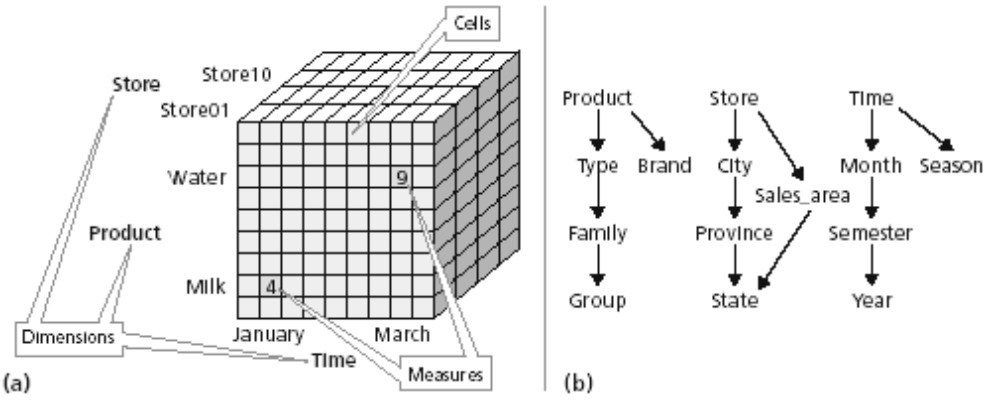

Figure 1. A multidimensional model data cube: (a) The cube itself is composed of cells that define fact attributes, while (b) the classification hierarchies display the dimensions that define the cube – *product*, *store* and *time*

By selecting cells, planes, or subcubes from the base cuboid, we can analyze sales figures at varying granularities. Such queries form the basis of OLAP functions like roll-up and drill-down.

In total, a d-dimensional base cube is associated with 2^d cuboids. Each cuboid represents a unique view of the data at a given level of granularity. Not all these cuboids need actually be present, however, since any cuboid can be computed by aggregating across one or more dimensions in the base cuboid. Nevertheless, for anything but the smallest data warehouses, some or all of these cuboids may be computed so that users may have rapid query responses at run time.

DATA CUBE ALGORITHMS

CUBE operator

One final note is in order at this point: described the data cube as a conceptual model. This is certainly true. However, in the case of a MOLAP server, it is also the physical model, as MOLAP stores the cube structure directly as a multi-dimensional array. Conversely, ROLAP servers must map this representation to a relational design.

OLAP *is* multi-dimensional data. That being said, one might legitimately ask "How does one prepare data for multi-dimensional analysis, particularly if some or all of the 2^d cuboids are required?" Strictly speaking, no special operators or SQL extensions are required to take a raw data set, composed of detailed transaction-level records, and turn it into a data structure, or group of structures, capable of supporting subject-oriented analysis. Rather, the SQL *group-by* and *union* operators can be used in conjunction with *d* sorts of the raw data set to produce all cuboids. However, such an approach would be both tedious to program and immensely inefficient, given the obvious inter-relationships between the various views. Consequently Jim Gray [4] et al. proposed the data cube operator as a means of simplifying the process of data cube construction. This paper defines that operator, called the *data cube.* The cube operator generalizes the histogram, cross-tabulation, roll-up, drill-down, and sub-total constructs found in most report writers. The writers explain the cube and roll-up operators, show how they fit in SQL, explain how users can define new aggregate functions for cubes, and discuss efficient techniques to compute the cube.

Subsequent to the publication of the seminal data cube paper, a number of independent research projects began to focus on designing efficient algorithms for the computation of the complete cube. Most were based upon the exploitation of the data cube *lattice*, a directed graph that depicts the relationships between all 2^d cuboids in a given d-dimensional space. Starting with the base cuboid – containing the full complement of dimensions - the lattice branches out by connecting every parent node with the set of child nodes/views that can be derived from its dimension list. In general, a parent containing *d* dimensions can be connected to *d* views at the next level in the lattice.

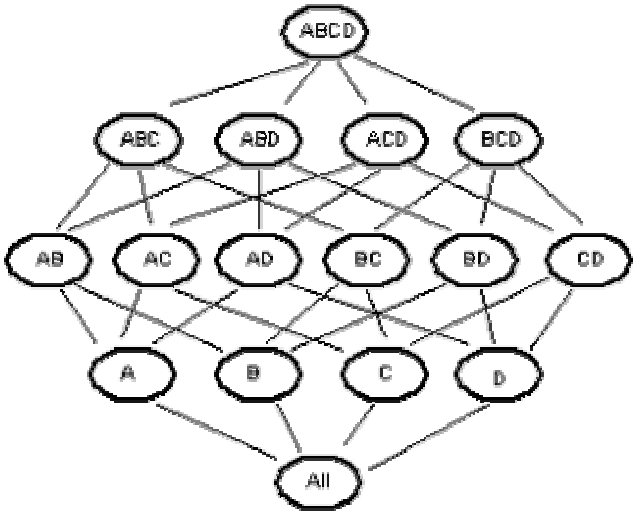

Figure 2. The Lattice structure for the CUBE operator

It should be clear from the lattice depiction that many views share common dimension values and that any efficient computational mechanism for producing group-bys, whether it be sort-based or hash-based, must exploit these relationships. For example, a three-dimensional cuboid can be viewed as the parent of three two-dimensional cuboids, each of which contains a distinct combination of two dimensions of the parent. Clearly, it should not be necessary to independently compute all four views since the parent and one or more of the children may be able to share some portion of the aggregation workload.

Though a number of algorithms have been proposed, the techniques of primary importance can be roughly divided into three categories.

Top Down. The top down methods [1] work directly from the lattice to compute smaller group-bys from larger parents. For example, the parent view ABCD might be used to generate ABC, AB and A. What sets the top down methods apart is the means by which they share the computation costs across views. Perhaps the best-known methods in this

class are the *PipeSort* and *PipeHash*. The basic premise of both algorithms is that a *minimum spanning tree* should be generated from the original lattice such that the cost of traversing edges - and thereby building cuboids - will be minimized.

There are five optimizations for combining multiple group-bys to efficiently compute:

- *Smallest-paren*t - this optimization, first proposed in aims at computing a group- by from the smallest previously computed group-by. In general, each group-by can be computed from a number of other group-bys. Figure 2 shows a four attribute cube (ABCD) and the options for computing a group-by from a group-by having one more attribute called its parent. For instance, AB can be computed from ABC, ABD or ABCD. ABC or ABD are clearly better choices for computing AB. In addition, even between ABC and ABD, there can often be big difference in size making it critical to consider size in selecting a parent for computing AB;

- *Cache-results* - This optimization aims at caching (in memory) the results of a group-by from which other group-bys are computed to reduce disk I/O. For instance, for the cube in Figure 2, having computed ABC, we compute AB from it while ABC is still in memory;

- *Amortize-scans* - This optimization aims at amortizing disk reads by computing as many group-bys as possible, together in memory. For instance, if the group-by ABCD is stored on disk, we could reduce disk read costs if all of ABC, ACD, ABD and BCD were computed in one scan of ABCD;

- *Share-sorts* - This optimization is specific to the sort-based algorithms and aims at sharing sorting cost across multiple group-bys;

- *Share-partitions* - This optimization is specific to the hash-based algorithms. When the hash-table is too large to fit in memory, data is partitioned and aggregation is done for each partition that fits in memory. We can save on partitioning cost by sharing this cost across multiple group-bys.

For OLAP databases, the size of the data to be aggregated is usually much larger than the available main memory. Under such constraints, the above optimizations are often contradictory. For computing B, for instance, the first optimization will favour BC over AB if BC is smaller but the second optimization will favour AB if AB is in memory and BC is on disk.

Bottom Up. As the dimensions increase, the high-dimension cuboids become increasingly *sparse*. Because child views are now almost as big as their parents, the top-down methods may become less efficient. As a result, a number of *bottom up* methods were proposed. Bottom up methods work by first aggregating (usually with a sort) on a single dimension, then recursively partitioning the current attribute in order to aggregate at successively finer degrees of granularity. Since the recursive sorting is performed on smaller and smaller partitions, most of the external memory sorting is avoided, restricted mainly to the first or second dimensions.

Array-based The algorithms [14], [5] presented above all correspond to the ROLAP model in that they operate on multi-dimensional tables. Given that many vendors (and customers) are attracted to the perceived performance benefits of the MOLAP model, it is also important to explore array-based approaches that directly support multi-dimensional data structures. In short, such methods structure the raw data set in a d-dimensional array that is typically stored on disk as a sequence of *chunks*. (A chunk is a means by which a large d-dimensional array can be partitioned into smaller d-dimensional sub-arrays). On dense data sets, there is little questions that array-based algorithms are very efficient. In sparse, high-dimensional environments, however, various performance-draining compromises must be employed in order to access the array.

Dynamic Data Cube

Range sum queries are useful analysis tools when applied to data cubes. A range sum query applies an aggregate operation (e.g., SUM, AVERAGE) to the measure

attribute within the range of the query. Efficient range-sum querying is becoming more important with the growing interest in database analysis.

Geffiner, Agrawal and A.El Abbadi [3] have presented the model of the range sum problem and discuss several previous approach. They present a performance analysis of the Basic Dynamic Data Cube, concluding that the method still has considerable update complexity as the dimensionality of the data cube increases. They present the Dynamic Data Cube (DDC), analyze the performance characteristics of the method, and demonstrate that the DDC is suited to dynamic growth of the cube, and that it handles clustered data more efficiently. This is important advantage of this method.

For many potential applications, however, it is more convenient to grow the size of the data cube dynamically to suit the data. For example, astronomers who are analyzing stars might form a data cube for their star database. They expect to discover more stars in the future. Clearly it would not be efficient to create a data cube that initially contains cells for all possible locations of star systems in the Universe, particularly since the vast majority of the resulting cells would always be empty. Rather, it is more practical to create the data cube initially only for locations of existing star systems; as additional systems are discovered, new cells can be added to the data cube. New star systems, however, can be found in any direction relative to existing systems, therefore the data cube must be able to grow in any direction relative to its existing cells. The direction of data cube growth should be determined by the data, and not a priori. The capability to grow the data cube dynamically in any direction is very important in many application environments.

This example also illustrates another problem. In many application domains data is essentially clustered, and there are large unpopulated regions in the data space. This additional information is not static. Range sum queries over a data cube formed from such data would be very useful, providing scientists with aggregate measurements for any arbitrary region of the cube.

Data Cube as a level in Object Oriented Data Warehousing design

Trujillo, Palomar and Gomez [10] have presented the approach uses a UML class diagram to specify the structure of a multidimensional model.

Figure 1 shows both a data cube and classification hierarchies. In Figure 1a shows a data cube typically used for multidimensional modeling. In this particular case, is defined a cube for analyzing measures along the *product, store*, and *time* dimensions, as shown in Figure 1b's classification hierarchies. In this example, *product sales* is related to only one *product* that is sold in one *store* to one *customer* at one *time*.

A measure is additive along a dimension if they can use the SUM operator to aggregate attribute values along all hierarchies defined on that dimension. The aggregation of some fact attributes—called roll-up in OLAP terminology— might not, however, be semantically meaningful for all measures along all dimensions.

In this example, *number of clients*—estimated by counting the number of purchase receipts for a given *product, customer, day*, and *store*—is not additive along the *product* dimension. Because the same ticket can include other *products*, adding up the *number of clients* for two or more *products* would lead to inconsistent results. However, other aggregation operators (SUM, AVG, MIN) —could be applied to other dimensions (*time).*

Defining the classification hierarchies of certain dimension attributes is crucial because these classification hierarchies provide the basis for the subsequent data analysis. Because a dimension attribute can also be aggregated to more than one other attribute, multiple classification hierarchies and alternative path hierarchies are also relevant. For this reason, directed acyclic graphs provide a common way of representing and analyzing dimensions with their classification hierarchies.

Figure 1b shows the different classification hierarchies defined for the *product, store*, and *time* dimensions. On the *product* dimension, they have defined a multiple classification hierarchy so that they can aggregate data values along two different hierarchy paths:

• *product–type–family–group;* • *product–brand.*

Other attributes, not used for aggregating purposes, can provide features for other dimension attributes, such as *product name.* For the *store* dimension, they have defined an alternative path classification hierarchy with two different paths that converge into the same hierarchy level:

• *store–city–province–state;* • *store–sales_area–state.*

Finally, they have also defined another alternative path classification hierarchy with the following paths for the *time* dimension:

• *time–month–semester–year;* • *time–season.*

In most cases, however, classification hierarchies are not so simple. The concepts of *strictness* and *completeness* are important for both conceptual purposes and for further multidimensional modeling design steps.

Once developers define the multidimensional model structure, users can define a set of initial requirements as a starting point for the subsequent data-analysis phase. From these initial requirements, users can apply a set of OLAP operations to the multidimensional view of data for further data analysis. These OLAP operations usually include the following:

• *roll-up,* which increases the level of aggregation along one or more classification hierarchies;

• *drill-down,* which decreases the level of aggregation along one or more classification hierarchies;

• *slice-dice,* which selects and projects the data;

• *pivoting,* which reorients the multidimensional data view to allow exchanging dimensions for facts symmetrically.

OO approach can elegantly represent multidimensional properties at both levels:

Structural level

This OO approach is not restricted to using flat UML class diagrams to model large, complex data warehouse systems. UML's package grouping mechanism groups classes into higher-level units, creating different levels of abstraction and simplifying the final model. In this way, a UML class diagram improves and simplifies the system specifications created with classic semantic data models such as the Entity-Relationship model. Their approach clearly separates the structure of a multidimensional model specified with a UML class diagram into facts and dimensions.

Facts and dimensions - Fact classes represent facts and the measures they are interested in, defined as attributes within these classes. Dimension classes represent dimensions.

Derived measures - They consider *derived measures* by placing the constraint /next to a measure in the fact class. For example - *number_of_clients, qty_sold* and *total_price.*

Classification hierarchies - For dimensions, a base class represents every classification hierarchy level. An association of classes specifies the relationships between two levels of a classification hierarchy.The base classes, including the dimension class, that belong to the classification hierarchy must contain an explicitly defined identifying attribute. The writers do this by placing the constraint next to one attribute in every class. This attribute is necessary to automatically generate the database schema from the UML class diagram into a target relational OLAP tool because these tools store the attribute in their meta-data to unambiguously identify every instance of a classification hierarchy level.

Relational commercial OLAP tools, however, use a default attribute within every classification hierarchy level that will be used in the subsequent data analysis phase. A *default* is a dimensional attribute that users want to analyze with a target commercial OLAP tool. This default attribute displays every time the user applies an OLAP operation, rather than having the tool prompt the user to specify which attribute to display before executing each operation.

Therefore, to plan for subsequent automatic generation into a target relational OLAP tool, we must qualify the default attribute for every hierarchy level in our UML class diagram. They call a default attribute a *descriptor* because they consider the term "default" too general. Thus, they define a descriptor in every class that represents a classification hierarchy level.

Dynamic level

They use *cube classes* to represent initial user requirements as the starting point for the subsequent data-analysis phase. A UML-compliant class notation properly and easily defines these classes.

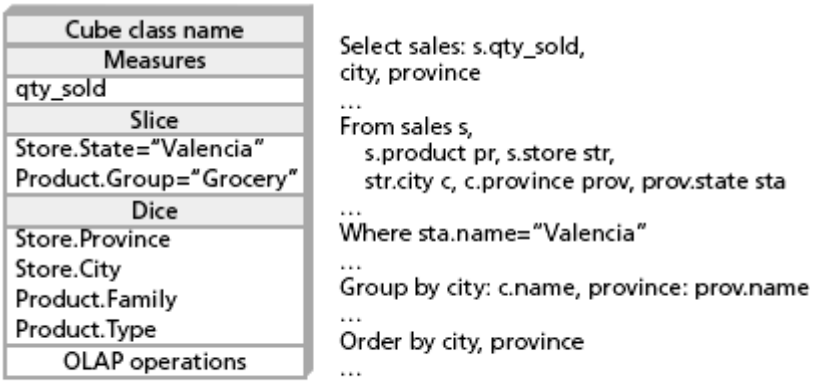

Figure 3.(a) Cube class example with parameters specified in the measures, slice, dice, and operations areas; (b) the class's corresponding Object-Query Language specification.

The basic components of the cube classes include the:

- head area, which contains the cube class's name;
- measures area, which contains the measures to be analyzed;
- slice area, which contains the constraints to be satisfied;
- dice area, which contains the dimensions and their grouping conditions to address the analysis; and
- cube operations, which cover the OLAP operations for a further data-analysis phase.

Figure 3 shows both the graphical notation of the cube class that corresponds to data requirement and its accompanying Object Query Language (OQL) specification. Figure 3a shows that the measure area specifies the measure to be analyzed, *qty_sold*. Constraints on dimension classification hierarchy levels—*group* and *state*—appear in the slice area, and the classification hierarchy levels for which we want to analyze measures—*family, type, province* and *city*—appear in the dice area. Finally, the cube operations section specifies the available OLAP operations.

For nonexpert UML or database users, the cube class's graphical notation facilitates the definition of initial user requirements. Every cube class has a more formal underlying OQL specification. Experts can use OQL to define cube classes by specifying the appropriate OQL sentences.

CONCLUSIONS AND FUTURE WORK

The *DDC* allows graceful growth of the data cube in any directions, making it more suitable for applications within involve change or growth.

For further work in direction "*dynamic data cube"*:

- to develop method of constraining the space requirements of the dynamic data cube of the full data cube size by deleting unnecessary data;

- to discuss the properties of the DDC which enable it to handle sparse and clustered data, as well as empty regions of the cube, efficiently.

OO Conceptual models of data cube are a very interesting direction for constructing and using efficiently information extracted from data cube.

OLAP tools implement a multidimensional model from two different levels:

- *Structural*—the structures that form the database schema and the underlying multidimensional model—also known as the metadata—that provides the model's key semantics (facts, measures, dimensions).

• *Dynamic*—refers to the definition of final user requirements and OLAP operations for further analyzing data.

For further work in this direction:

- to develop appropriate class definition for data cube;
- to add additional properties in class for further analyzing.

REFERENCES

[1] Agarwal S., R. Agrawal, P. Deshpande. On the Computation of Multidimensional Aggregates *In Proceedings of the .22nd international Conference on Very Large Databases,* 506-521**,** Mumbai (Bombay), 1996.

[2] Chaudhuri S., U. Dayal. An overview of datawarehousing and OLAP technology, *ACM SIGMOD Record* 1997*,* 26(1):65-74.

[3] Geffner S., D. Agrawal, A. El Abbadi, T. Smith. Relative Prefix Sums: An Efficient Approach for Querying Dynamic OLAP Data Cubes, *In Proc. of the 15th International Conference on Data Engineering*, Sydney, Australia, March 1999.

[4] Gray J.,A. Bosworth, A. Layman, H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-total, *In Proceedings of the 12Th. IEEE International Conference on Data Engineering,* 152–159, New Orleans,LA, February - March 1996

[5] Kotidis Y., N. Roussopoulos, An alternative storage organization for ROLAP aggregate views based on cubetrees, *ACM SIGMOD Record* 1998*,* 27(2): 248-258

[6] Lewis P., Database and Transaction Processing: An Application – Oriented Approach, Addison Wesley, New York, 2003

[7] Molina.H, J. Ullman, J. Windom, DataBase Systems: The Complete Book, Prentice Hall, 2002, 1119p.

[8] Mumick S., D. Quass, B. Mumick, Maintenance of Data Cubes and Summary Tables in a Warehouse, *In Proceedings of the ACM SIGMOD International Conference on Management of Data*, 100-111, Tucson, Arizona, May 1997.

[9] Palpanas T., Knowledge discovery in data warehouses, *ACM SIGMOD Record*, September 2000, 29(3)

[10] Trujillo J., M. Palomar, J. Gomez, Y. Song. Designing Data Warehouses with OO Conceptual Models, *Computer*, 66-75, December 2001, IEEE

[11] Ullman J., Efficient Implementation of Data Cube Via Materialized Views, http://db.stanfort.edu/~ullman

[12] Vassiliadis P., A Survey of Logical Models for OLAP Databases. *ACM SIGMOD Record*, December 1999, 28(4)

[13] Widom J., Research Problems in Data Warehousing, *Proc.4th Intl. CIKM Conference*., 1995.

[14] Zhao Y., P. Deshpande, J. Naughton. An Array-Based Algorithm for Simultaneous Multidimensional Aggregates, *In ACM SIGMOD International Conference*, 159-170, Tuscon, AZ, USA, June 1997.

ABOUT THE AUTHOR

Antoaneta Ivanova, PhD Student, Department of Computer Sciences and Technologies, TU-Varna, Phone: +359 899 885 497, E-mail:antoaneta_ii@yahoo.com

Assoc.Prof. Boris Rachev, Ph.D., Department of Computer Sciences and Technologies, TU-Varna, E-mail:Rachev@ieee.bg