

A Cartesian Data Model for Decision Support Systems

Mauro Guazzo *

Copyright by Codework 1998-2007

Abstract

This work motivates the need for a specific data model suitable for econometric and business data, specially in a decision support scenario.

The proposed model corresponds to a multidimensional view of the data and entails the nomenclature as well as the numeric data cube.

A set of sample primitives is presented.

The key idea is that the model must invade the realm of semantics, so as to allow the resulting system to support the user in most situations.

The actual implementation of the model in the **3-way TANGRAM** software system is briefly sketched.

1 Introduction

The great impact of the relational data model, both in theoretical and practical terms, may have made us overlook the fact that the adoption of a data model is simply instrumental to an approach and a class of applications.

This work motivates the interest to explore different paths—but always according to the classical paradigm:

1. Conceptual data model
2. Primitive operations
3. Development of a software system that approximates points 1. and 2.

The real-life motivation for this work was to shed some light on the concept of a “data workbench” as it emerges in decision support situations.

Examples of applications are within-company decision making, market analysis, data exploration and the analysis of national statistical data.

*Mauro Guazzo, Codework Italia srl, Corso Lanza 105, 10133 Torino Italy, E-mail mauro.guazzo@gmail.com —A brief biography of the author can be found at www.codework.it/tangram/cvitae.htm

The traditional name for the required software is Decision Support System (DSS); in recent years the synonym On-Line Analytical Processing (OLAP) has also gained wide acceptance.

The key feature of this software is that it must allow a problem analyst to directly manipulate pre-existing data in new and complex ways so as to respond to unanticipated questions in a short time.

The very fact that this theme deserves a specific software tool is far from obvious and requires a close examination of the real-life requirements.

This notes will only provide a partial justification of the fact that other classes of software (like Relational DBMS, data exploration packages and spreadsheets) do not provide a satisfactory solution.

It is an intriguing fact that all commercial DSS have in some way re-invented some multidimensional data structure, informally called a *DATA CUBE*.

This applies even to those DSS that adopt a Relational DBMS as their internal back-end for data access.

Unfortunately, the prevailing approach has been to let the market requests drive the design, so that too often points 1. and 2. have been hastily solved or skipped altogether.

This work proposes a formal definition of the data cube, which we call a Cartesian model, a complete set of primitive operations and some examples of higher-level typical operations.

The main point of this paper is that the model may usefully invade the realm of data semantics (that is, human meaning of the data) so that, ideally, *allowed* operations coincide with those that a human analyst considers *meaningful* operations.

A thorough comparison that contrasts the Relational and the Cartesian data model is beyond the scope of this work.

As a simple matter of curiosity, we may note in passing that either model seems to engulf the other, that is, considers the other as a special case of itself.

2 Cartesian data model

2.1 Formal definition

We define a *CUBE* of rank N the following data structure:

- an ordered set of N texts, that we call *DIMENSIONS*
- N ordered set of texts, of $n_1, n_2, n_3, \dots, n_N$ items respectively, which are called *ELEMENTS*
- a set of $n_1 \times n_2 \times n_3 \dots n_N$ numerical values, which we call *CELLS* and which are associated with the Cartesian product of the sets of elements.

This concludes the formal definition.

In this definition 'text' means a string of characters and 'numerical value' means a real number. Both these concepts will be narrowed and qualified by the semantic axioms that follow.

Examples of cubes in this paper will show only dimensions and elements. The associated numerical values need not be shown.

As a first example, the cube EXPORT, of rank 5, will be shown as follows:

VARIABLES	PRODUCTS	PARTNERS	PERIODS	VERSIONS
Net Price	Cast iron	Poland	Jan 94	Centr. Stat. Office
Quantity	Steel	Germany	Feb 94	Custom Excise
Income	Soft iron	France	Mar 94	Min. Foreign Trade
Weight	Copper	Greece	Apr 94	Chamber of Commerce
	Aluminum	Netherlands	May 94	
	Lead	Sweden	Jun 94	
	Zinc		Jul 94	
			Aug 94	
			Sep 94	
			Oct 94	
		Nov 94		
			Dec 94	

Such a scheme assumes the existence of $4 \times 7 \times 6 \times 12 \times 4 = 8064$ numeric cells.

Since all sets are ordered sets, we will be able to reference every element with an *INDEX* and every cell with *N* indices.

In some cases the elements (of some dimension) will be regarded as the catenation of several *FACETS*.

For example "Jan" and "94" are two facets (or segments) of the text string "Jan 94".

2.2 Semantic axioms

1. The contents of every cell in a cube may be one of the following:

- *MISSING*, (with the meaning of "non applicable" or "non known" or "not declared") or
- **genuinely numeric, meaning that arithmetic operations on (non-missing) cells are meaningful.**

Axiom 1. is violated if the cells contain telephone numbers, part numbers, numeric codes or passwords.

2. Dimensions and elements are natural-language texts.

3. Every element set contains *semantically unique* elements.

Axiom 3. is violated by the following set of texts:

“Bundesrepublik Deutschland”

“BRepublik Deutsch.”

“Federal Republic of Germany”

4. **The values in the cells are independent. This means that world knowledge (and expertise in the subject matter) does not provide a deterministic way to obtain a cell from the others.**

The EXPORT cube violates Axiom 4. if we know that the Income can be obtained as Net Price times Quantity.

2.3 Dimension classes

The user can classify each dimension in one of the three classes:

- **alpha** - nothing is asserted about the elements of such a dimension (their nature, unit of measure, preferred ordering, relationship between them, ...);
- **beta** - the elements are homogeneous.
This is understood in the very broad sense that every process (statement, reasoning, computation, printout, graph,...) that is meaningful for an element must be meaningful for every other element of the dimension.
- **gamma** - the elements are of class beta and there is a preferred or natural way of ordering them.

This classification shows that the chosen ordering of the sets (as required by the cube definition) has a semantic value only for gamma dimensions.

In all other cases it has a technical value only, meaning that the elements may be ordered as we please but a particular order must be expressed.

If you examine the EXPORT cube, you are likely to classify its five dimensions as follows:

VARIABLES	alpha
PRODUCTS	beta
PARTNERS	beta
PERIODS	gamma
VERSIONS	beta

The main motivation for declaring dimension classes to a software tool is, of course, that such tool will be able to offer class-specific primitives (for example, time-specific commands).

It is also clear that the proposed classification is very coarse: from this starting point the reader may easily define finer ones (and perhaps a software that handles them).

Typical cubes that are encountered in practice exhibit:

- a single alpha dimension (typically called Measures or Variables);
- several beta dimensions;
- a single gamma dimension (typically the time Periods).

Many other situations are encountered, including cubes with several gamma dimensions (or, unexpectedly, two time scales):

VARIABLES	TARGET YEARS	FORECASTING DATES
Population	2000	1989
Energy production	2005	1991
Food production	2010 2015	1993

A simple consequence of these definitions is that the unit of measure of a cell (See the Formal definition) depends only on the alpha dimensions associated with the cell.

3 Primitive operations

3.1 General approach

Before proceeding to propose primitive operations on cubes, a statement of intent is appropriate:

1. The primitives must tend towards an ideal situation in which *meaningful* operations coincide with *permitted* operations.
2. The correct association between cells and textual elements must never be violated or left to the care of the user. In other words all primitives must act on elements and cells in a co-ordinated and consistent way.
3. It is highly desirable that the primitives represent global operations, that act on whole cubes and not on their individual cells. It is also desirable that the same primitive be applicable to cubes of different size or rank.
4. We intend to define a complete set of primitives that, applied in cascade, may implement any conceivable operation on cubes.

Later we want to design those high-level primitives that are convenient for a specific application field.

For the sake of concision, each primitive will be defined informally, showing its effect on sample cubes.

3.2 A complete set of primitives

- SELECT

Reorders the elements of a given dimension.

If, by way of example, we apply the permutation 5 1 4 6 3 2 7 to the second dimension of the cube EXPORT we obtain the following result:

VARIABLES	PRODUCTS	PARTNERS	PERIODS	VERSIONS
Net Price	Aluminum	Poland	Jan 94	Centr. Stat. Office
Quantity	Cast iron	Germany	Feb 94	Custom Excise
Income	Copper	France	Mar 94	Min. Foreign Trade
Weight	Lead	Greece	Apr 94	Chamber of Commerce
	Soft iron	Netherlands	May 94	
	Steel	Sweden	Jun 94	
	Zinc		Jul 94	
			Aug 94	
			Sep 94	
			Oct 94	
			Nov 94	
			Dec 94	

The proposed definition of SELECT allows for repetitions and omissions in the index list. Such a list is termed a *ORDERED MULTISSET* of the original list.

The power of this primitive depends on the fact that the multiset of indices may depend on the elements and on the associated cells in a variety of ways.

For example the permutation we just used (5 1 4 6 3 2 7) has the effect of sorting the elements of the second dimension alphabetically.

Different permutations will correspond to different criteria, which may also depend on the contents of the numeric cells.

For example:

Reorder the PRODUCTS on the variable 'Income', on version 'Central Stat. Office', on the total of PARTNERS, on the total of PERIODS.

Note that the totals mentioned in this request have the sole purpose of determining the desired ordering of the products. The resulting cube would still be a $4 \times 7 \times 6 \times 12 \times 4$ cube and would not contain totals.

- COMPUTE

Appends a new element to a dimension, computing it as a function of existing elements of the *same* dimension.

The key point in this definition is that it applies to one dimension at a time.

The computational notation may be considered unimportant for our purposes.

Suppose, for example, that the formula is expressed with the four operations + - * / and that indices appear in square brackets.

The formula:

$$100 * ([4] - [1]) / [4]$$

would then produce a new element, say 'Percent difference', on the fifth dimension of the cube EXPORT.

The creation of this new element will bring about $4 \times 7 \times 6 \times 12$ new cells.

If some cells of elements 4 and 1 of VERSIONS are missing, then the corresponding cells of the new element will also be missing.

In other words we require a propagation of missing values in the computation.

- TRANSPOSE

Applies a permutation to the dimensions and restructures the cube accordingly.

For instance the permutation 1 5 4 3 2 transforms the cube EXPORT into a cube with dimensions:

VARIABLES	VERSIONS	PERIODS	PRODUCTS	PARTNERS
-----------	----------	---------	----------	----------

- CROSSOVER

Moves an element facet across dimensions and restructures the cube accordingly.

To see an example, we start off with a cube called ENERGY:

ACCOUNTS BY COUNTRY	YEARS
Energy Prod. - Austria	1994
Energy Consump.- Hungary	1995
Energy Consump.- Italy	
Energy Consump.- Austria	

As a first step, we indicate where to split the elements of 'ACCOUNTS BY COUNTRY' into two facets.

In this example we have used dashes '-' to show how each text is split into two facets.

As a second step, the set 'ACCOUNTS BY COUNTRY' is expanded to complete the Cartesian product of the two sets of facets:

ACCOUNTS BY COUNTRY	YEARS
Energy Consump. - Austria	1994
Energy Prod. - Austria	1995
Energy Consump. - Hungary	
Energy Prod. - Hungary	
Energy Consump. - Italy	
Energy Prod. - Italy	

The new elements that are thus created (such as 'Energy prod. - Italy') are associated with missing values.

As a final step of the CROSSOVER example, the facets associated with COUNTRIES migrate to another dimension ('YEARS') and the following result is obtained:

ACCOUNTS	YEARS BY COUNTRIES
Energy Consump.	1994 - Austria
Energy Prod.	1994 - Hungary
	1994 - Italy
	1995 - Austria
	1995 - Hungary
	1995 - Italy

Note that a particular case of CROSSOVER (where the facets that migrate coincide with the whole text) may cause a whole dimension to migrate to another.

- PROMOTE
Appends a one-element dimension to the existing dimensions. The rank is increased.
- DEMOTE
Drops the last dimension of a cube, which must consist of a single element. The rank is decreased.
- INPUT
Creates a new cube with texts and numeric values that are supplied by the user (or imported from some external format).
- OUTPUT
Produces some printed or graphical representation of a cube (or else exports it in some external format).

The primitives that we listed so far constitute a *COMPLETE* set, as far as operations on single cubes are concerned.

For a set of primitives to be complete, we require that it pass two simple conditions:

1. If two people have constituted two cubes containing the same data (with the same elements), there must exist a sequence of primitives in the set that transforms one cube into the other.
2. If a computation is possible with unrestricted cell-oriented operations, it must be obtainable also with a sequence of primitives in the set.

It is a simple matter to verify that condition 1. is verified by the proposed primitives.

As to condition 2., we note that the only primitive of the set that is capable of computation is COMPUTE.

We must then prove that the adoption of the cube-oriented, one-dimensional computation prescribed by COMPUTE is as powerful as unrestricted cell-oriented computation.

The proof is also rather elementary:

The successive application of CROSSOVER, TRANSPOSE and DEMOTE can transform a cube of any rank into a rank-1 cube and the generality of COMPUTE allows for any computation on such a cube.

Should this point be unclear, let us borrow an example from linear algebra. Imagine a rank-2 cube of shape 5×5 , that we regard as a square matrix. We wish to compute the determinant of this matrix.

The straight application of COMPUTE does not work, because it offers only computations on whole lines and on whole columns.

If, however, we use CROSSOVER to reduce the cube to a rank-1 cube of 25 elements, then the formula for the determinant can be directly entered and a 26th element can be created.

The question of when this ‘reduction to flat file’ is convenient is quite open to discussion. In many cases it is best to develop an ad-hoc primitive to do the job.

3.3 Examples of high-level primitives

A reasonable goal for a DSS software is to provide primitives that correspond to all typical requests in a given application domain.

In a typical situation a user will expect a single command to carry out what he conceives as a single high-level operation.

To provide more examples of this, we continue with other primitives that hint at those operations that are congenial with the Cartesian model.

- **TODATE**

Requires a cube with a dimension representing time periods. Produces a cube of the same rank and size in which the periods represent the ‘total-to-date’ (alias the discrete integral, or the running sum).

For instance the application of TODATE to the EXPORT cube will yield:

VARIABLES	PRODUCTS	PARTNERS	PERIODS	VERSIONS
Net Price	Cast iron	Poland	Jan 94 todate	Centr. Stat. Office
Quantity	Steel	Germany	Feb 94 todate	Custom Excise
Income	Soft iron	France	Mar 94 todate	Min. Foreign Trade
Weight	Copper	Greece	Apr 94 todate	Chamber of Commerce
	Aluminum	Netherlands	May 94 todate	
	Lead	Sweden	Jun 94 todate	
	Zinc		Jul 94 todate	
			Aug 94 todate	
			Sep 94 todate	
			Oct 94 todate	
			Nov 94 todate	
			Dec 94 todate	

The cells labeled “Mar 94 todate” in this new cube will be the sum of those labeled “Jan 94”, “Feb 94”, “Mar 94” in the original cube.

- GROUP

Requires a cube in which the elements of a dimension are constituted by two facets. Returns a cube in which one facet has disappeared and the cells have been aggregated on the second facet.

If, for example, we GROUP the cube ENERGY on ACCOUNTS, we get a result of this nature:

ACCOUNTS	YEARS
Energy Prod.	1994
Energy Consump.	1995

The aggregation or grouping of different cells into one can be done via a sum or some other symmetric function, like average, product, maximum, minimum.

- BLOWUP

Requires two cubes of the same rank. On every dimension the elements of the first cube must be in a many-to-one correspondence with those of the second cube. Produces a cube of the size of the first cube, in which the values of the second cube have been split in proportion with those of the first.

An example is badly needed at this point.

First cube, called INCOME, that will be used as a template (or a driver):

REGIONS	PERIODS
Italy North	Jan - Quarter1
Italy Center	Feb - Quarter1
Italy South	Mar - Quarter1
France East	Apr - Quarter2
France Central	May - Quarter2
France Atlantic	Jun - Quarter2
France Mediterr.	Jul - Quarter3
Germany South	Aug - Quarter3
Germany East	Sep - Quarter3
Germany North	Oct - Quarter4
	Nov - Quarter4
	Dec - Quarter4

Let the second cube 'BANK SAVINGS' represent the data to be "blown up":

COUNTRIES	PERIODS
Italy	Quarter1
France	Quarter2
Germany	Quarter3
	Quarter4

The result of BLOWUP will have the same size and the same elements as the INCOME cube.

Its numeric contents will represent an estimate of BANK SAVINGS by region and month, assuming that savings are proportional to income.

- APPEND

Appends two cubes on a given dimension. Requires the two cubes to be conformable (of the same size) on the other dimensions.

If we start with a cube TEXTILE PRODUCTION—WEST of this shape:

COUNTRIES	PERIODS
France	Quarter1
Switzerland	Quarter2
Austria	Quarter3
Italy	Quarter4

and with a cube TEXTILE PRODUCTION—EAST:

COUNTRIES	PERIODS
Poland	QRT I
Hungary	QRT II
Slovakia	QRT III
	QRT IV

then, by applying APPEND on the first dimension, we (predictably) get a result of this shape:

COUNTRIES	PERIODS
France	Quarter1
Switzerland	Quarter2
Austria	Quarter3
Italy	Quarter4
Poland	
Hungary	
Slovakia	

- MERGE

Requires two cubes of the same rank. Produces a cube that has, on each dimension, the set union of the elements of the input cubes.

If two cells are associated to the same elements in the input cubes and they contain different (non-missing) values, an error is signaled.

Let us see an example.

First input cube:

PRODUCTS	COUNTRIES	PERIODS
Peripherals	USA	1994
Cables	Canada	
Micro chips	Japan	

Second input cube:

PRODUCTS	COUNTRIES	PERIODS
Disk drives	South Korea	1995
Micro chips	USA	
Cables	Canada	

The result of the MERGE operation would be:

PRODUCTS	COUNTRIES	PERIODS
Peripherals	USA	1994
Cables	Canada	1995
Micro chips	Japan	
Disk drives	South Korea	

3.4 Classification of primitives

Having now sketched a sufficient variety of primitives, we can stop to attempt different classifications.

1. **Valence**, that is, the number of input cubes that a primitive requires.

In this proposal we only accept primitives that have up to 2 input cubes and produce up to 1 output.

This means a total of 6 types, as many as the combinations of 0, 1, 2 inputs and 0, 1 outputs.

The primitive OUTPUT, for example, has one input cube and zero output cubes (its output is a non-cube).

As a general rule, this classification of primitives will reckon only the number of proper cubes and ignore the auxiliary information that is supplied to the primitive.

The input to COMPUTE, for example, is a cube plus some ancillary information, like the choice of a dimension, the text associated to the new element and a formula to compute its cells.

2. **Rank preservation.**

Most primitives conserve the rank of the input cubes, others, like PROMOTE and DEMOTE, produce output cubes of different rank.

3. **Semantic tolerance.**

There are cases in which the correct operation of a primitive requires the assumption that two texts are synonyms at the semantic (human) level.

To see an example of this distinction, we must go back to the primitive APPEND.

Attentive readers may have noticed that the result of APPEND cannot accommodate all the elements of the input cubes.

The primitive must assume that certain texts are semantic synonyms (and may ask the user to confirm this fact).

In our examples, it was assumed that these texts were synonyms and that either version could be used in the result:

QRT I	—	Quarter1
QRT II	—	Quarter2
QRT III	—	Quarter3
QRT IV	—	Quarter4

On these grounds we classify APPEND as a semantic tolerant primitive, meaning that its correct operation requires human-level equivalence of two corresponding elements.

Contrast with MERGE, which works only at the formal level and assumes that two texts are equivalent only if they are equal on a character-by-character basis.

4. **Class intelligence**, that is, whether the primitive must be aware of alpha, beta, gamma dimensions and treat them differently.

Nearly all mentioned primitives produce meaningful results on dimensions of any class.

The only exception that we have seen so far is TODATE, that is meaningful only on a dimension of class gamma.

This (most common) case of class intelligence is often called *time intelligence*.

4 Implementation project

The proposed theoretical model has been implemented by a software system, initially code-named HELM and now called **3-way TANGRAM**, which has been used in more than one hundred projects, for a variety of companies and organizations.

As one may expect, 3-way TANGRAM is mainly applied where the data structure is complex and unanticipated requests are commonplace: for example corporate consolidation, controllership, market analysis, support to negotiation, etc.

The practical software development has been very faithful to the conceptual model.

The only noteworthy deviation from the model has been due to the fact that many users find it difficult to come to grips with cubes of high rank or varying rank.

The remedy has been to emphasize the use of three dimensional cubes, (where dimensions are associated by default to classes alpha, beta, gamma), so as to simplify the dialogue in the most common cases in which they do represent variables, items and time periods.

The inexperienced user can thus always refer to the elementary concepts of lines, columns and pages.

Logical dimensions above three can be coded as facets.

The primitives CROSSOVER, PROMOTE and DEMOTE guarantee that facets may become dimensions and dimensions above three can later “appear” and “disappear” at will. The current system maximum is 26 dimensions.

The software system is implemented for the Windows 95/98/00/NT platform, but the porting to other operating systems is envisaged.

The guiding principle in the development has been never to invade the territory of existing software classes (like graphic presentation systems or desk top publishing) but rather to interface with them.

At the time of writing, there are 79 primitives available in the 3-way TANGRAM system.

They can be classified in still another way, that is, by function:

Administration	23
Data Analysis and outputs	45
Namesystem (nomenclature) management	11
Total	79

The endless cycle of designing, implementing, testing and refining the software has been going on for almost ten years.

Ideas that have been brewing for such a long time have yielded a few general conclusions:

1. The Cartesian model appears suitable and efficient for most high-level operations that are encountered in business problems.
2. Complete user self-sufficiency is attainable.
The roles of passive users, power users and application developers are often very blurred. As the application gets more sophisticated, users must be able to smoothly move from ready-made services to a development environment.
Toolkits, open ended systems, high level programming, documented interface points are some ways to achieve this goal.
3. It is fundamental that as much “data semantics” as possible be transferred to the software system.

5 Bibliography and links

The 3-way TANGRAM material (including user manual and software) is maintained at the URL <http://www.codework.it/tangram>

The material includes simple guidelines for users who wish to contribute their own additions to the existing framework (*the TANGRAM co-operation project*).

A good bibliography on Decision Support is maintained by Prof. Daniel Power at the URL <http://dssresources.com>

The following references may provide a historical viewpoint of the DSS foundations:

- [1] E. Turban, “*Decision Support Systems*”, Mac Millan, New York, 1990
- [2] R.H. Sprague and H.J. Watson, “*Decision Support System*”, Pentice Hall, 1989
- [3] “Decision Support System”, *The international Journal* ISSN 0167 9236, Elsevier (North Holland)
- [4] M. Ginzberg and W. Reitman, “Decision Support Systems”, *Proc. New York Univ. Symposium*, E.A. Stohr (ed.), Amsterdam, 1982

- [5] S.L. Alter, “*Decision Support System—Current Practice and Continuing Challenges*”, Addison Wesley, 1980
- [6] Ariav, G. and M. Ginzberg, “DSS designing: A Systematic View of Decision Support”, *Comm. of the ACM*, Vol. 28, October 1985
- [7] R.H. Sprague and H.J. Watson, “*Decision Support System—Putting Theory into Practice*”, Prentice Hall, Englewood Cliffs, New York, 1986
- [8] R.H. Sprague and E.D. Carlson “*Building Effective Decision Support System*”, Prentice Hall, 1982
- [9] S.L. Alter, “A Taxonomy of Decision Support System”, *Sloan Management Review*, Vol.19 N.1 39-56