

# **SLICING < DICING < SIZING < RISING**

## **A THEORY OF DATACUBE AND PIVOTING**

**Tony Hürlimann**  
**Tony.huerlimann@unifr.ch**

**Working Paper**

**September 2004**

---

*DEPARTMENT D'INFORMATIQUE, UNIVERSITE DE FRIBOURG*  
*DEPARTEMENT FÜR INFORMATIK, UNIVERSITÄT FREIBURG*  
*Department of Informatics, University of Fribourg*  
*Site Regina Mundi, rue de Faucigny 2, CH-1700 Fribourg / Switzerland*

# **Slicing < Dicing < Sizing < Rising**

## **A Theory of Datacube and Pivoting**

Tony Hürlimann  
Tony.huerlimann@unifr.ch

### ***Summary***

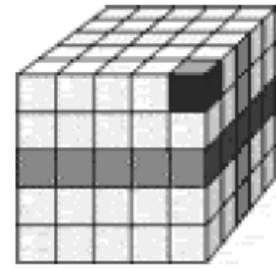
This paper gives a unified theory on datacube and pivoting in order to understand the OLAP functionalities. It is argued that all data operations in OLAP can be reduced to slicing, dicing, sizing, and rising operations in a multidimensional datacube.

Datacubes are the fundamental building block of every OLAP tool, hence our efforts are concentrated on them. The datacube operations are explained as functions which transform  $n$ -dimensional datacubes into datacubes of dimensions less, equal or higher than  $n$ . All operations are viewed in this perspective. This reduces the various kinds of OLAP-operations considerably and looks at them in a new unified way, which also might be interesting in implementing OLAP-tools.

Then this paper exposes the connections of  $n$ -dimensional datacubes with their many 2-dimensional representations, the *pivot-tables*. The many different pivot-table representations of an  $n$ -dimensional datacube are also understood in a unified way, such that all representations can be generated from each other by a very limited number of operations on the datacube which could be subsumed under the general operator *pivoting*. It is shown how a datacube of any dimension is transformed into *any* pivot-table representation using just pivoting and the mentioned cube operations *slicing*, *dicing*, *sizing*, and *rising*.

# 1 Introduction

Information is an increasing value resource, required from managers to schedule and monitor effectively the enterprise activities. Hence an IS (Information System) must collect and classify the information, by means of integrated and suitable procedures, in order to produce in time and at the right levels the synthesis to be used to support the decisional process, as well as to administrate and globally control the enterprise activity. While **databases** are the place where data are *collected*, **data warehouses** are systems that *classify* the data. According to William Inmon,



widely considered the father of the modern data warehouse, a Data Warehouse is a "*Subject-Oriented, Integrated, Time-Variant, Non-volatile collection of data in support of decision making*". Data Warehouses tend to have these distinguishing features: (1) Use a subject oriented dimensional data model; (2) Contain publishable data from potentially multiple sources and; (3) Contain integrated reporting tools.

**OLAP** is a key component of data warehousing, and OLAP Services provides essential functionality for a wide array of applications ranging from reporting to advanced decision support. According to [www.olapcouncil.org] OLAP (On-Line Analytical Processing) "*... is a category of software technology that enables analysts, managers and executives to gain insight into data through fast, consistent, interactive access to a wide variety of possible views of information that has been transformed from raw data to reflect the real dimensionality of the enterprise as understood by the user. OLAP functionality is characterized by dynamic multidimensional analysis of consolidated enterprise data supporting end user analytical and navigational activities including calculations and modeling applied across dimensions, through hierarchies and/or across members, trend analysis over sequential time periods, slicing subsets for on-screen viewing, drilldown to deeper levels of consolidation, rotation to new dimensional comparisons in the viewing area etc. ...*". The focus of OLAP tools is to provide multidimensional analysis to the underlying information. To achieve this goal, these tools employ multidimensional models for the storage and presentation of data. Data are organized in *cubes* (or *hypercubes*), which are defined over a multidimensional space, consisting of several dimensions. Each dimension comprises of a set of aggregation levels. Typical OLAP operations include the aggregation or deaggregation of information (*roll-up* and *drill-down*) along a dimension, the *selection* of specific parts of a cube (*dicing*) and the reorientation of the multidimensional view of the data on the screen (*pivoting*).

OLAP functionality is characterized by dynamic multi-dimensional analysis of consolidated enterprise data supporting end user analytical and navigational activities including:

- calculations and modelling applied across dimensions, through hierarchies and/or across members
- trend analysis over sequential time periods (data mining)
- slicing subsets for on-screen viewing
- drill-down to deeper levels of consolidation
- reach-through to underlying detail data
- rotation to new dimensional comparisons in the viewing area

## 1.1 Various OLAP approaches

The debate on the underlying model, supporting OLAP, is centered on two major lines. Whereas some vendors of traditional relational database systems (RDBMS), propose the **ROLAP architecture** (Relational On-Line Analytical Processing) [MikroStrategy 95/97, Bed Bricks 1997, Informix 1997], others support the **MOLAP architecture** (Multidimensional On-Line Analytical Processing) [Arbor 1996]. The advantage of the MOLAP architecture is, that it provides a direct multidimensional view of the data and is normally easy to use, whereas the ROLAP architecture is just a multidimensional interface to relational data and requires normally an advanced knowledge on the SQL queries. On the other hand, the ROLAP architecture has two advantages: (a) it can be easily integrated into other existing relational information systems, and (b) relational data can be stored more efficiently than multidimensional data.

Hence ROLAP is based directly on the architecture of relational databases and different vendors just extend the SQL standard language in various ways in order to implement the OLAP functionality. For example in ORACLE the *cube* and *rollup* operator expands a relational table, by computing the aggregations over all the possible subspaces created from the combinations of the attributes of such a relation. Practically, the introduced *CUBE* operator calculates all the marginal aggregations of the detailed data set. The value 'ALL' is used for any attribute which does not participate in the aggregation, meaning that the result is expressed with respect to all the values of this attribute.

```

SELECT group_function(column1), column2, group_function(column3)...
FROM table_list
[WHERE conditions]
GROUP BY CUBE (group_by_list)

SELECT group_function(column1), column2, group_function(column3)...
FROM table_list
[WHERE conditions]
GROUP BY ROLLUP (group_by_list)

```

In a ROLAP architecture, data are organized in a *star* (Figure 1) or *snowflake* schema. A star schema consists of one central *fact* table and several denormalized *dimension* tables. The *measures* of interest for OLAP are stored in the fact table (e.g. Dollar Amount, Units in the table SALES). For each dimension of the multidimensional model there exists a dimension

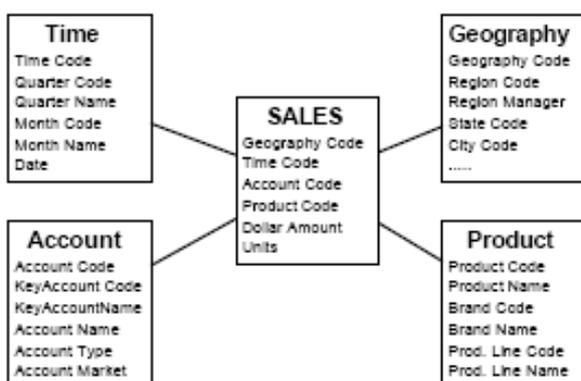


table (e.g. Geography, Product, Time, Account) with all the levels of aggregation and the extra properties of these levels. The normalized version of a star schema is a snowflake schema, where each level of aggregation has its own dimension table.

Multidimensional database systems (MDBMS) store data in n-dimensional arrays. Each dimension of the array represents the respective dimension of the cube. The contents of the array are the measure(s) of the cube.

**Figure 1: A Star Schema on different dimensions**

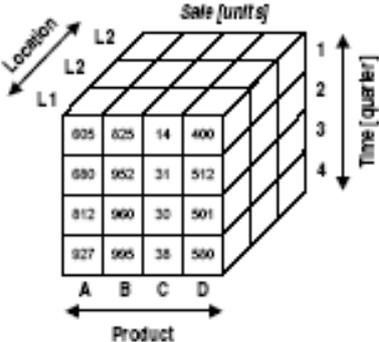
MDBMS require the precomputation of all possible aggregations: thus they are often more performant than traditional RDBMS [Coll96], but more difficult to update and administer. There have also been efforts to model directly and more naturally multidimensional databases; we call these efforts *cube-oriented* (or MOLAP architecture). This does not mean that they are far from the relational paradigm – in fact all of them have mappings to it – but rather that their main entities are *cubes* and *dimensions*. In practice, the cube’s data are extracted from databases using standard SQL queries and stored in “cubes”, which beside the data extracted contain pre-calculated data in order to speed up the viewing. One of the big advantage of the cube-oriented approach is its intuitive use of generating a multitude of various “views” of the data; another advantage is speed: Having the data in a cube, it “easier” to reorganize the data than in ROLAP.

In this paper, we analyze the cube-oriented architecture. In section 2 a definition of the fundamental concepts, such as datacube, is given. In section 3, the operators on datacubes are defined and analyzed. An interpretation of these operations in the light of OLAP is given in section 4. Section 5 introduces the basic concept of pivot-table, which is identified as a 2-dimensional representation of a datacube. Section 6 briefly summarizes the connection to LPL.

**2 Definition of Datacube**

A set of  $(n+1)$ -tuples  $(d_1, d_2, K, d_n, m)$  with  $d_1 \in D_1, d_2 \in D_2, K, d_n \in D_n, m \in M$  is called an ***n*-dimensional datacube**.  $D_1, D_2, K, D_n$  are finite sets of *members* and are the **dimensions** of the cube,  $M$  is also a finite set of *values* (normally numerical) called the **measurement**.

A dimension is a *structural* attribute of a cube that is a list of members, all of which are of a similar type in the user's perception of the data. For example, all months, quarters, years, etc., make up a time dimension; likewise all cities, regions, countries, etc., make up a geography dimension. A dimension acts as an index for identifying values within a multi-dimensional array. If one member of the dimension is selected, then the remaining dimensions in which a range of members (or all members) are selected define a sub-cube. If all but two dimensions have a single member selected, the remaining two dimensions define a spreadsheet (or a "slice" or a "page"). If all dimensions have a single member selected, then a single cell is defined. Dimensions offer a very concise, intuitive way of organizing and selecting data for retrieval, exploration and analysis.



A *member* is a discrete name or identifier used to identify a data item's position and description within a dimension. For example, January 1989 or 1Qtr93 are typical examples of members of a Time dimension. Wholesale, Retail, etc., are typical examples of members of a Distribution Channel dimension.

## Figure 2: A 3-dimensional Cube

A member combination is an exact description of a unique *cell* in the datacube which contains a single *value* (the measurement). A datacube can also be seen as a multi-dimensional array and can be visualized as “cubes”. A 3-dimensional cube with the dimensions “Location”, “Product” and “Time” is given in Figure 2. The measurement is “Sale”.

A cell can be seen as a single datapoint that occurs at the intersection defined by selecting one member from each dimension in a multi-dimensional array. The maximal number of cells is given by the cardinalities of the dimensions as:  $|d_1| \cdot |d_2| \cdot K \cdot |d_n|$ . The tuple-set  $D_1 \times D_2 \times K \times D_n$  is also called the Cartesian Product of the dimensions.

A datacube can be *dense* or *sparse*. It is called dense if a relatively high percentage of the possible combinations of its dimension members contain data values, otherwise it is called sparse. It is important to see, that very sparse datacubes are very common.

From a database point of view, an  $n$ -dimensional datacube is typically stored as a **database table** containing  $n+1$  fields, the  $n$ -first fields representing the dimensions and the  $(n+1)$ -th field represents the measurement (the data value). The first  $n$  fields are typically (but not necessary) foreign keys pointing to a table filled with basis “identifier” lists. It is important to note that a table is a more general concept than a datacube: (1) The same tuple of members in a datacube – defining a cell – can occur only once in a datacube, while it can occur several times in a table, except when a primary key is defined on the “dimensional” fields. In praxis, however, this is not a limitation, because we mostly analyse data which can be classified according some dimensions and hence *have* distinct tuples. (2) A table – besides the “dimensional” fields can contain several “measurement” fields. If this is the case, then we can build several datacubes with the same dimensions, or a datacube with an additional dimension, which contains the “measurement” field name as members.

We call the list of tuples defining a datacube – normally printed in a vertical way – on a piece of paper, the **standard view** or the **db-view** of the datacube.

In a mathematical notation, a cube can be represented as follows:

$$m_{i_1, i_2, K, i_n} \text{ with } i_1 \in D_1, i_2 \in D_2, K, i_n \in D_n, m \in M$$

Where  $i_1, i_2, K, i_n$  are called indexes. This notation is called *indexed notation*.

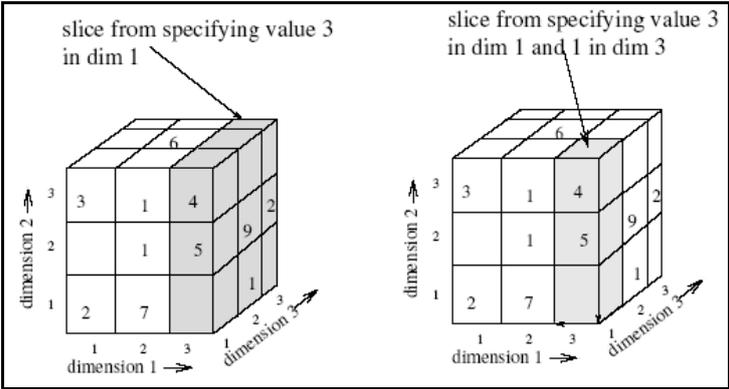
## 3 Operations on Datacubes

Several operations can be applied to datacubes. The result of these operations is again a datacube. The operators are: *slicing*, *dicing*, *sizing*, and *rising*. The operation *slicing* on an  $n$ -dimensional datacube will generate a datacube of dimension  $\leq n$ , while *dicing* and *sizing* will generate a datacube of dimension  $n$ , and *rising* will generate a datacube of dimension  $\geq n$ .

### 3.1 Slicing

Slicing means to “fix” a particular member of one or several of the dimensions (we call them “fixed” dimensions) and discarding all other tuples containing any of the other members of the “fixed” dimensions. Then we remove the particular members from all tuples.

The result is a datacube with fewer dimensions (Figure 3).



**Figure 3: Slicing a 3-dimensional datacube**

In fact, the operation “slicing on several dimensions” can be reduced to a sequence of “slicing on a single dimension”. The operation is commutative and associative on the dimensions, meaning that the order which it is applied to the “fixed” dimensions does not matter. “Slicing on a single dimension” reduced the dimension by one. Hence, applying this sliding operation to an  $n$ -dimensional cube, generates a  $(n-1)$ -dimensional cube. One can apply this operation at most  $n$  times successively to get a 0-dimensional cube, say a single cell of the original datacube.

In the mathematical notation, slicing is to generate a subcube as follows. As an example, let’s start with a 3-dimensional cube  $a_{i,j,k}$  with  $i \in I, j \in J, k \in K$  and  $I = \{i_1, i_2, \dots, i_m\}$ ,  $J = \{j_1, j_2, \dots, j_n\}$ ,  $K = \{k_1, k_2, \dots, k_o\}$ . Now, choosing the member ‘ $k_2$ ’ from  $K$  generates a new datacube  $b$  as follows

$$b_{i,j} \leftarrow a_{i,j,k_2}$$

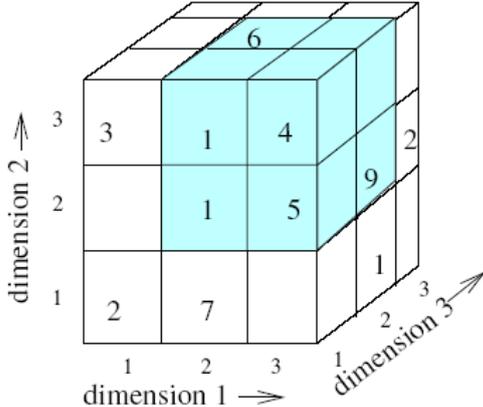
The operation  $\leftarrow$  means that for each  $(i, j)$ -combination in the cube  $a$ , the value of the cell is copied to the corresponding cell in  $b$ . It is in fact *matrix-assignment*.

Fixing two members in different dimensions generates a 1-dimensional cube  $c$ :

$$c_j \leftarrow a_{i_3,j,k_2}$$

**3.2 Dicing**

*Dicing* (we may call this also *down-sizing*) a  $n$ -dimensional datacube means to select a (possibly empty) subset of members from each dimension and discard all tuples which contains at least one member not in the selection (see Figure 4).



Note that the resulting datacube has the same dimension as the datacube before applying the operation. Even if the selection on a particular dimension contains a single member, it is not the same as slicing. The dimension is *not* reduced and a set containing a single member after all is also a set!

**Figure 4: Dicing a 3-dimensional datacube**

If no member is selected of any dimension, we get the *empty cube*. (Note that the empty cube is not the same than a 0-dimensional cube, which contains a single cell.)

If all members are selected from all dimensions, the resulting cube is the same as the original cube.

Mathematically, the operation is simply stated as follows. As above, we start again with the 3-dimensional cube  $a_{i,j,k}$ . Let  $P_{i,j,k}$  be a predicate (relation) which is true some  $(i, j, k)$ -combination and false on the others. Then dicing  $a_{i,j,k}$  on  $P_{i,j,k}$  means to remove all tuples from  $a_{i,j,k}$  for which  $P_{i,j,k}$  is false. This gives a new cube called  $b_{i,j,k}$  and we write :

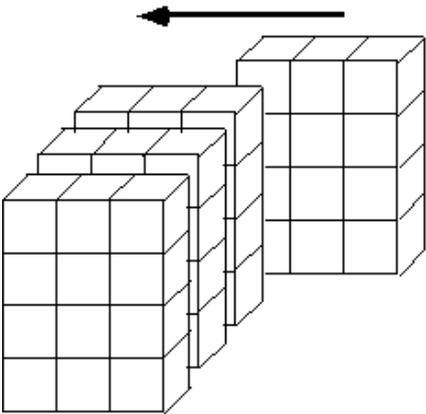
$$b_{i,j,k} \leftarrow a_{i,j,k} \text{ for all } (i, j, k) \in P_{i,j,k}$$

Another – somewhat shorter – notation is :

$$b_{i,j,k|P_{i,j,k}} \leftarrow a_{i,j,k}$$

**3.3 Sizing**

*Sizing* (we may call this also *up-sizing*) a  $n$ -dimensional datacube means to add tuples which contains members in a particular dimension that were not part of that dimension. For example, if a particular dimension called “Product” contained the members “rice”, “maize”, and “beans” and nothing else, then we add a new product “oat” to the “Products” list.



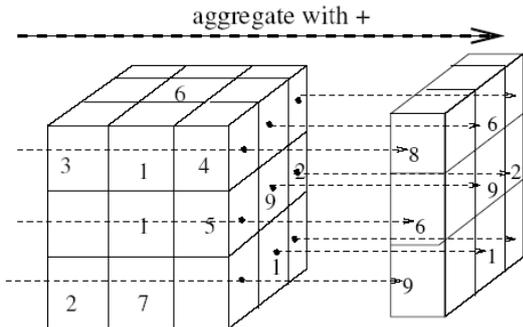
The consequence is an extension of the datacube in one or several particular dimensions. This operation can also be seen as “merging” a  $n$ -dimensional datacube with a  $(n-1)$ -datacube in which the  $n-1$  dimensions occur in both cubes. It is “adding a slice” to the cube. This operation does not change the dimension of the cube – just its size in one or several dimension (Figure 5).

**Figure 5: Sizing : Adding one member to a particular dimension.**

In mathematical notation, we can write given the same example as above:

$$b_{i,j',k} \leftarrow a_{i,j,k} \text{ with } j \in J, j' \in J \cup \{j_{m+1}\}$$

The element  $j_{m+1}$  is added to the domain  $J$ . However the additional cells are not filled.



**A important special case** occurs when the added “slice” is an *aggregated cube*. A aggregated cube is constructed from the  $n$ -dimensional cube as follows: Choose one dimension out of the  $n$  dimensions in this cube, then choose an operation that can be applied to the cells along the chosen dimension (p.e. summation for numerical cells) (see Figure 6).

**Figure 6: Sizing : SUM aggregates along one dimension.**

Applied the operation along all cells in the chosen dimension and report the result into a datacube of  $(n-1)$ -dimensions defined out of the remaining  $n-1$  dimensions (see Figure 6).

Mathematically, for our example above, we choose dimension  $J$  and the sum operator, then we get the aggregated cube  $b_{i,k}$  as following :

$$b_{i,k} \leftarrow \sum_{j \in J} a_{i,j,k}$$

Of course, one can also aggregate along the other dimensions  $i$  and  $k$  and we get the 2-dimensional cubes :

$$b'_{j,k} \leftarrow \sum_{i \in I} a_{i,j,k} \qquad b''_{i,j} \leftarrow \sum_{k \in K} a_{i,j,k}$$

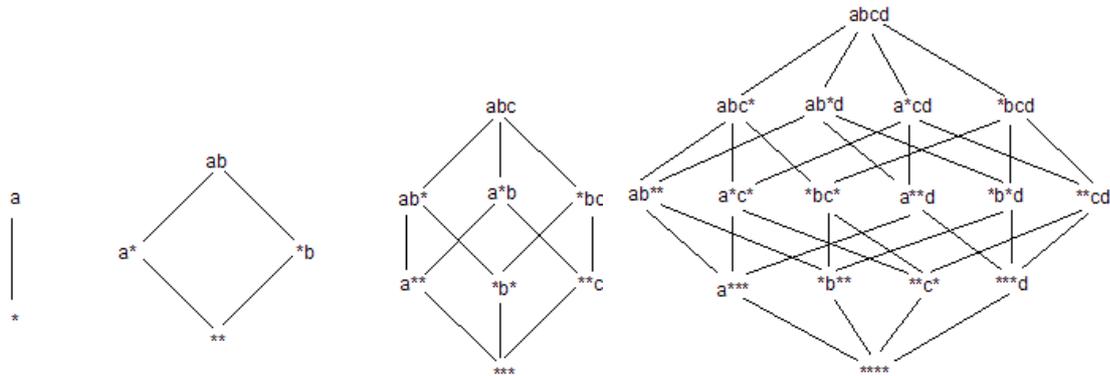
The aggregation can be continued to generate 1-dimension cubes:

$$c_i \leftarrow \sum_{j \in J, k \in K} a_{i,j,k} \qquad c'_j \leftarrow \sum_{i \in I, k \in K} a_{i,j,k} \qquad c''_k \leftarrow \sum_{i \in I, j \in J} a_{i,j,k}$$

Finally, we get the 0-dimensional cube by aggregating all cells :

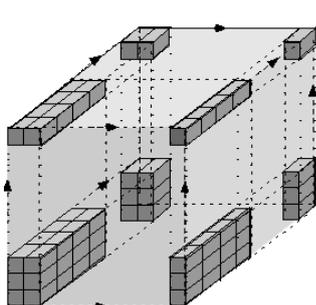
$$d \leftarrow \sum_{i \in I, j \in J, k \in K} a_{i,j,k}$$

Figure 7 shows that the generation of all aggregated cubes forms a lattice. From a 3-dimensional cube, for example, with the dimensions  $abc$ , one can generate 3 cubes of dimension 2 ( $ab^*$ ,  $a^*c$ ,  $*bc$ ), 3 of dimension 1 ( $a^{**}$ ,  $*b^*$ ,  $**c$ ), and 1 of dimension 0 ( $***$ ). The aggregated cube  $ab^*$ , for example, is built by aggregate the cube  $abc$  along the third dimension  $c$ .



**Figure 7: Cube-lattices up to 4 dimensions**

In general, from an  $n$ -dimensional cube with dimensions  $D_1, D_2, K, D_n$ , one can generate



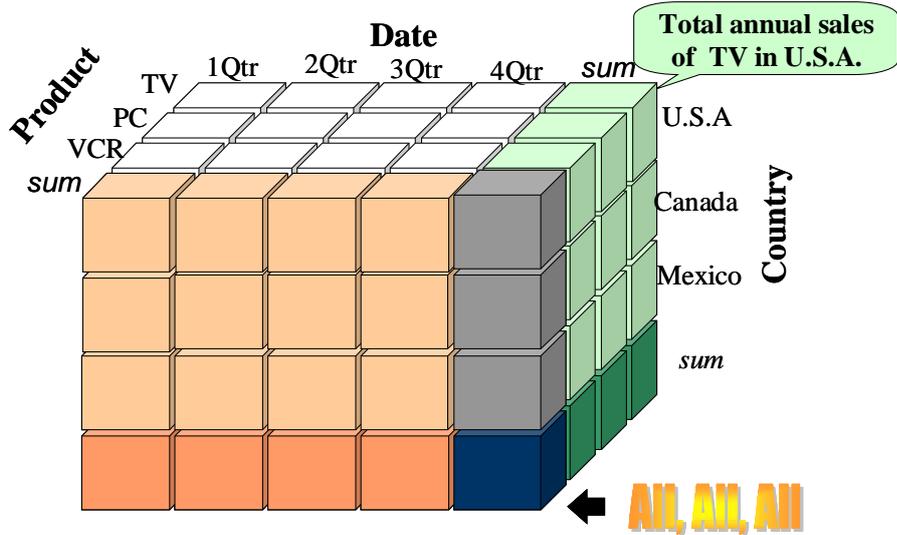
$\binom{n}{i}$  aggregated cubes of dimensions  $i$ . All aggregated cubes together with the original cube can be merged together to form a new cube – the **augmented cube**. It is also an  $n$ -dimensional cube, for which all dimensions are extended by just one member. If the original cube has dimension cardinalities of:  $c_1, c_2, K, c_n$ , then the augmented cube has cardinalities of  $c_1 + 1, c_2 + 1, K, c_n + 1$ .

**Figure 8: All aggregated cubes out of a 3-dimensional cube**

Figure 8 shows the original  $2 \times 3 \times 4$  cube of 3-dimensions and all its aggregated cubes and how they fit together in  $3 \times 4 \times 5$  **augmented cube**.

There are several aggregate operations (for numerical data) that can be considered: *Sum, Average, Max, Min, Count, Variance, Standard deviation*. More than one aggregate operation may be applied at the same time.

A concrete example is given in Figure 9. The cube represents the

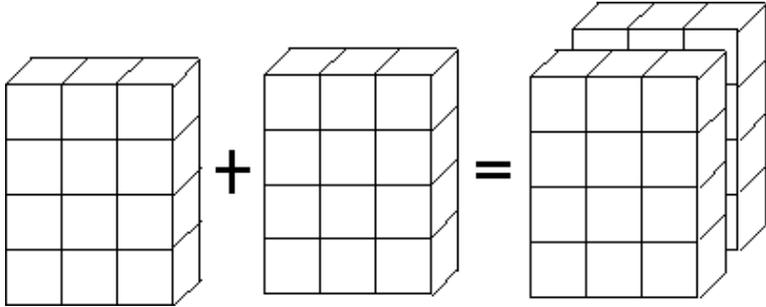


annual sales in three different countries of three products in four time periods. Summing from left to right gives the annual sale, summing from top to bottom gives the sales in all countries, summing from back to front gives the total sales of all products... the bottom, right corner in the front gives the total overall sales quantity.

**Figure 9: A concrete example of a 3-dimensional cube together with the aggregates**

3.4 Rising

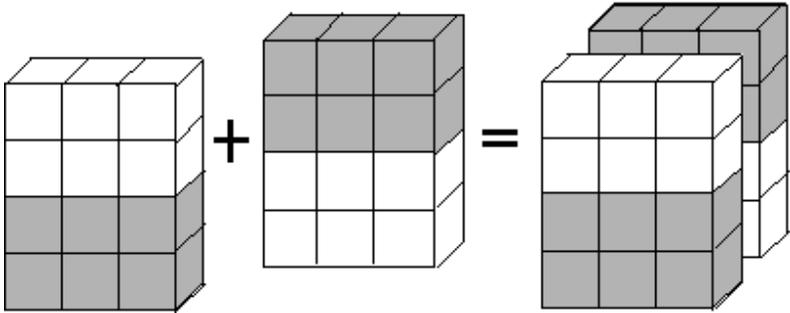
*Rising* an  $n$ -dimensional cube is an operation that “lifts” the datacube to a higher dimension. This can occur in two important applications. (1) Two datacubes having the same number of dimensions and having the same dimensions can be merged together. Figure 10 displays an example.



**Figure 10: A concrete example of a 3-dimensional cube together with the aggregates**

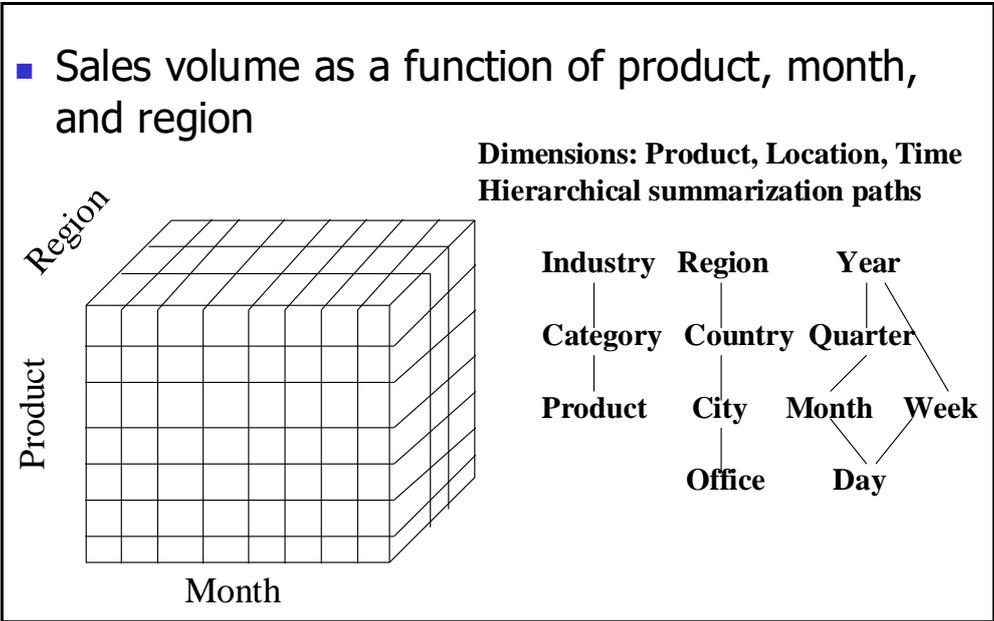
This case is especially interesting for comparative studies, comparing scenarios and outcomes of the same datacube (Multiple snapshot analysis in LPL). Rising implies to introduce an additional dimension into the resulting datacube. Hence the resulting datacube has then  $n+1$  dimensions.

(2) The second important application arises when the actual cube has to be partitioned into several groups, for example, along a time dimension, the months, one has to group the dimension along quarters; or along a product dimension, one has to group them into various product categories, etc (see Figure 12).



**Figure 11: Rising in order to partition a cube**

We partition the original cube into the desired parts and build from each part a complete datacube of the original size – by getting eventually a very sparse cube. Then we *rise* these cubes by “merging” them along a new dimension, which implements the partition (Figure 11).



**Figure 12: Examples of “grouping”**

The idea behind the partition of a cube is that certain dimensions can be structured into hierarchies (Year→Month→Week→Day, Continent→Country→Region→State...). The partition can be arbitrary however, it can be even on several dimensions, that is, certain tuples of the cube may belong to one part and other tuples to another part. The two most

important applications of this operation are (1) grouping and hierarchy building in OLAP tool and grouping and subgrouping in reports.

## 4 Interpretation of the Operations in the Light of OLAP

On-Line Analytical Processing (OLAP) is a category of software technology that enables analysts, managers and executives to gain insight into data through fast, consistent, interactive access to a wide variety of possible views of information that has been transformed from raw data to reflect the real dimensionality of the enterprise as understood by the user.

OLAP is implemented in a multi-user client/server mode and offers consistently rapid response to queries, regardless of database size and complexity. OLAP helps the user synthesize enterprise information through comparative, personalized viewing, as well as through analysis of historical and projected data in various "what-if" data model scenarios. This is achieved through use of an OLAP Server.

We use the OLAP glossary to go through the different "operations" needed to be able to characterize a system an OLAP, see [[www.OLAPCouncil.org](http://www.OLAPCouncil.org)].

### AGGREGATE (CONSOLIDATE, ROLL-UP)

"Multi-dimensional databases generally have hierarchies or formula-based relationships of data within each dimension. Consolidation involves computing all of these data relationships for one or more dimensions, for example, adding up all Departments to get Total Division data. While such relationships are normally summations, any type of computational relationship or formula might be defined."

The Roll-Up operations is typically to partition a cube and then to *rise* it a long the partition. This operation can be repeated several times in order to generate hierarchies of dimensions. Aggregation has been extensively discussed

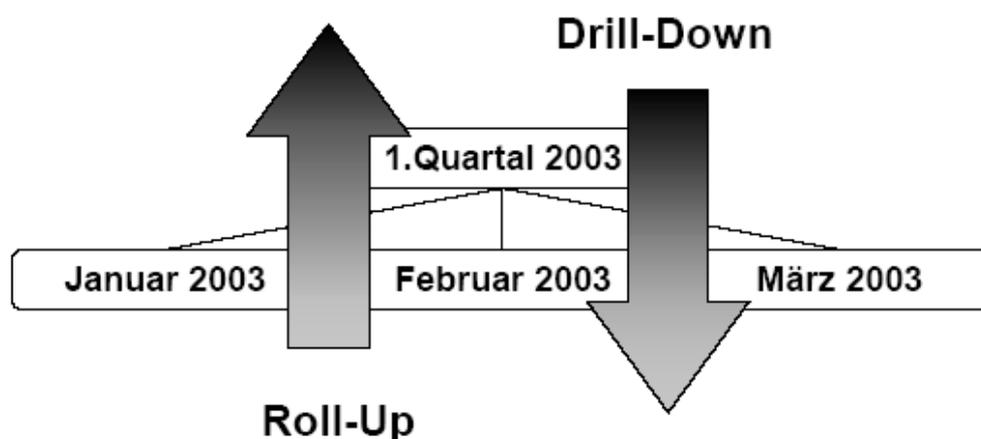


Figure 13: Drill-Down and Roll-Up operations

### DRILL DOWN/UP

"Drilling down or up is a specific analytical technique whereby the user navigates among levels of data ranging from the most summarized (up) to the most detailed (down). The

drilling paths may be defined by the hierarchies within dimensions or other relationships that may be dynamic within or between dimensions. For example, when viewing sales data for North America, a drill-down operation in the Region dimension would then display Canada, the eastern United States and the Western United States. A further drill-down on Canada might display Toronto, Vancouver, Montreal, etc.”

The Drill-Down means *slicing* a cube along the hierarchies defined before by a rolling-up process.

## ANALYSIS, MULTI-DIMENSIONAL

“The objective of multi-dimensional analysis is for end users to gain insight into the meaning contained in databases. The multi-dimensional approach to analysis aligns the data content with the analyst's mental model, hence reducing confusion and lowering the incidence of erroneous interpretations. It also eases navigating the database, screening for a particular subset of data, asking for the data in a particular orientation and defining analytical calculations. Furthermore, because the data is physically stored in a multi-dimensional structure, the speed of these operations is many times faster and more consistent than is possible in other database structures. This combination of simplicity and speed is one of the key benefits of multi-dimensional analysis.”

Multi-dimensional analysis is nothing else than cube manipulation!

## CALCULATED MEMBER

“A calculated member is a member of a dimension whose value is determined from other members' values (e.g., by application of a mathematical or logical operation). Calculated members may be part of the OLAP server database or may have been specified by the user during an interactive session. A calculated member is any member that is not an input member.”

By rising a cube, one can add member to a dimensions the cells of which are calculated. The aggregates are typical such calculated cells.

## CHILDREN & HIERARCHICAL RELATIONSHIPS

“Members of a dimension that are included in a calculation to produce a consolidated total for a parent member. Children may themselves be consolidated levels, which requires that they have children. A member may be a child for more than one parent, and a child's multiple parents may not necessarily be at the same hierarchical level, thereby allowing complex, multiple hierarchical aggregations within any dimension.”

“Any dimension's members may be organized based on parent-child relationships, typically where a parent member represents the consolidation of the members which are its children. The result is a hierarchy, and the parent/child relationships are hierarchical relationships.”

“Members of a dimension with hierarchies are at the same level if, within their hierarchy, they have the same maximum number of descendants in any single path below. For example, in an Accounts dimension which consists of general ledger accounts, all of the detail accounts are Level 0 members. The accounts one level higher are Level 1, their parents are Level 2, etc. It can happen that a parent has two or more children which are different levels, in which case

the parent's level is defined as one higher than the level of the child with the highest level.”

Using *rising* a cube it was shown how a hierarchy can be implemented.

**NAVIGATION**

“Navigation is a term used to describe the processes employed by users to explore a cube interactively by drilling, rotating and screening, usually using a graphical OLAP client connected to an OLAP server.”

**NESTING (OF MULTI-DIMENSIONAL COLUMNS AND ROWS)**

“Nesting is a display technique used to show the results of a multi-dimensional query that returns a sub-cube, i.e., more than a two-dimensional slice or page. The column/row labels will display the extra dimensionality of the output by nesting the labels describing the members of each dimension. For example, the display's columns may be:

January				February				March			
Actual		Budget		Actual		Budget		Actual		Budget	
Prod A	Prod B	Prod A	Prod B	Prod A	Prod B	Prod A	Prod B	Prod A	Prod B	Prod A	Prod B

These columns contain three dimensions, nested in the user's preferred arrangement. Likewise, a report's rows may contain nested dimensions: “

Chocolate Bars	Unit Sales	xxxx	xxxx	xxxx
	Revenue	xxxx	xxxx	xxxx
	Margin	xxxx	xxxx	xxxx
Fruit Bars	Unit Sales	xxxx	xxxx	xxxx
	Revenue	xxxx	xxxx	xxxx
	Margin	xxxx	xxxx	xxxx

**PAGE DISPLAY (PIVOT, ROTATE, ROW DIMENSION, COLUMN DIMENSION, HORIZONTAL DIMENSION, VERTICAL DIMENSION)**

“The page display is the current orientation for viewing a multi-dimensional slice. The horizontal dimension(s) run across the display, defining the column dimension(s). The vertical dimension(s) run down the display, defining the contents of the row dimension(s). The page dimension-member selections define which page is currently displayed. A page is much like a spreadsheet, and may in fact have been delivered to a spreadsheet product where each cell can be further modified by the user.”

“To change the dimensional orientation of a report or page display we use pivoting. For example, rotating may consist of swapping the rows and columns, or moving one of the row dimensions into the column dimension, or swapping an off-spreadsheet dimension with one of the dimensions in the page display (either to become one of the new rows or columns), etc. A specific example of the first case would be taking a report that has Time across (the columns)

and Products down (the rows) and rotating it into a report that has Product across and Time down. An example of the second case would be to change a report which has Measures and Products down and Time across into a report with Measures down and Time over Products across. An example of the third case would be taking a report that has Time across and Product down and changing it into a report that has Time across and Geography down.”

#### PAGE DIMENSION

“A page dimension is generally used to describe a dimension which is not one of the two dimensions of the page being displayed, but for which a member has been selected to define the specific page requested for display. All page dimensions must have a specific member chosen in order to define the appropriate page for display.”

See: “take out/in” operator in section 5.

#### SELECTION

“A selection is a process whereby a criterion is evaluated against the data or members of a dimension in order to restrict the set of data retrieved. Examples of selections include the top ten salespersons by revenue, data from the east region only and all products with margins greater than 20 percent.”

See: dicing

#### SLICE AND DICE

“A slice is a subset of a multi-dimensional array corresponding to a single value for one or more members of the dimensions not in the subset. For example, if the member Actuals is selected from the Scenario dimension, then the sub-cube of all the remaining dimensions is the slice that is specified. The data omitted from this slice would be any data associated with the non-selected members of the Scenario dimension, for example Budget, Variance, Forecast, etc. From an end user perspective, the term slice most often refers to a two-dimensional page selected from the cube.”

“The user-initiated process of navigating by calling for page displays interactively, through the specification of slices via rotations and drill down/up.”

Slicing and dicing have been explained.

## 5 Two-dimensional Representation of Datacubes

Datacubes must be represented on sheet of papers for reports or on a screen in order to be viewed by human beings, that is, they must be projected onto a two-dimensional space. We call a two-dimensional representation of a cube “*pivot-table*”. The reason for this *definition* will be clear in a moment.

### 5.1 Pivoting

i	j	k	a
i1	j1	k1	1
i1	j1	k2	2
i1	j1	k3	3
i1	j1	k4	4
i1	j2	k1	2
i1	j2	k2	4
i1	j2	k3	6
i1	j2	k4	8
i1	j3	k1	3
i1	j3	k2	6
i1	j3	k3	9
i1	j3	k4	12
i2	j1	k1	2
i2	j1	k2	4
i2	j1	k3	6
i2	j1	k4	8
i2	j2	k1	4
i2	j2	k2	8
i2	j2	k3	12
i2	j2	k4	16
i2	j3	k1	6
i2	j3	k2	12
i2	j3	k3	18
i2	j3	k4	24

Datacubes can be represented in many ways on a two-dimensional space. One such representation was shown in section 2: *The standard view*, which is a vertical (normally top-down) listing of all tuples in the datacube in a given order. An example is the 3-dimensional cube  $a_{i,j,k}$  with  $i \in \{i1, i2\}$ ,  $j \in \{j1, j2, j3\}$ , and  $k \in \{k1, k2, k3, k4\}$ . Its standard view is shown in Figure 14 (on the left). However in the same way as a vertical (from top to bottom) representation, one may depict it horizontally (Figure 14 on the right). These two views are the simplest projection of a cube into the 2-dimensional space where the data can be browsed or even edited.

i	j	k	a																				
i1																							
j1	j1	j1	j1	j2	j2	j2	j2	j3	j3	j3	j3	j1	j1	j1	j1	j2	j2	j2	j2	j3	j3	j3	
k1	k2	k3	k4																				
1	2	3	4	2	4	6	8	3	6	9	12	2	4	6	8	4	8	12	16	6	12	18	24

Figure 14: Standard view of a 3-dimensional cube

In a sense they are “extreme” views because *all* dimensions are listed vertically or horizontally. We could also imagine to project some dimensions horizontally and the other vertically. In our case, we can project 1 or two dimensions horizontally (Figure 15);

i	j	k	a		
		k1	k2	k3	k4
i1	j1	1	2	3	4
i1	j2	2	4	6	8
i1	j3	3	6	9	12
i2	j1	2	4	6	8
i2	j2	4	8	12	16
i2	j3	6	12	18	24

i	j	k	a									
	j1	j1	j1	j2	j2	j2	j2	j3	j3	j3	j3	
	k1	k2	k3	k4	k1	k2	k3	k4	k1	k2	k3	k4
i1	1	2	3	4	2	4	6	8	3	6	9	12
i2	2	4	6	8	4	8	12	16	6	12	18	24

Figure 15: 1 and 2 dimensions projected horizontally

In general for an  $n$ -dimensional cube this gives us  $n+1$  pivot-table representations. Furthermore, if we consider any permutation order on the dimensions, we get  $n!$  possibilities of pivot-tables. These permutations are a rich source on projecting the datacube on to a 2-dimensional space. We call this going from one permutation to another *pivoting*. Hence the name of *pivot-table*. Three examples are shown in Figure 16. In the first, 1 dimension is projected horizontally and the permutation is  $(i, k, j)$ . The second projects 2 dimensions horizontally and the permutation is  $(k, j, i)$ . The third also projects 2 dimensions horizontally and the permutation is  $(k, i, j)$ .

i	k	j	a		
		j1	j2	j3	
i1	k1	1	2	3	
i1	k2	2	4	6	
i1	k3	3	6	9	
i1	k4	4	8	12	
i2	k1	2	4	6	
i2	k2	4	8	12	
i2	k3	6	12	18	
i2	k4	8	16	24	

k	j	i	a				
	j1	j1	j2	j2	j3	j3	
	j1	j2	j1	j2	j1	j2	
k1	1	2	2	4	3	6	
k2	2	4	4	8	6	12	
k3	3	6	6	12	9	18	
k4	4	8	8	16	12	24	

k	i	j	a				
	i1	i1	i2	i2	i2		
	j1	j2	j3	j1	j2	j3	
k1	1	2	3	2	4	6	
k2	2	4	6	4	8	12	
k3	3	6	9	6	12	18	
k4	4	8	12	8	16	24	

Figure 16: Three other pivot-tables of the same datacube

The aggregated cubes can be viewed by “taking out” one or several dimensions from the original cube. So Figure 17 represents the aggregated cube  $b_{j,k} \leftarrow \sum_{i \in I} a_{i,j,k}$  and the cube  $c_{i,k} \leftarrow \sum_{j \in J} a_{i,j,k}$ . This implies that an aggregate operator was given (in our case SUM).

i	k	b		
	k1	k2	k3	k4
i1	3	6	9	12
i2	6	12	18	24
i3	9	18	27	36

i	k	c		
	k1	k2	k3	k4
i1	6	12	18	24
i2	12	24	36	48

**Figure 17: Two aggregated cubes**

We summarize: Given any  $n$ -dimensional cube and a permutation on the dimension as well as two numbers  $h$  and  $k$  and an aggregate operator, one can generate any pivot-table of the cube or one of its aggregated cubes,  $h$  being the number of dimensions that are projected horizontally, and  $k$  being the number of dimensions of “taken out”. The permutation is then interpreted as follows: project the  $n-h-k$  first dimensions vertically, the  $h$  following horizontally and the last  $k$  once are “taken out”. For example, the standard view of a 5-dimensional cube would be given as:  $perm = (1,2,3,4,5), h = 0, k = 0$ . The two pivot-tables in Figure 17 are defined by:

$$perm = (2,3,1), h = 1, k = 1, op = SUM \quad \text{and} \quad perm = (1,3,2), h = 1, k = 1, op = SUM .$$

## 5.2 Ordering

Until now, we did not much care about **order**. Each cell in the datacube is determined by the tuple of its members. Hence, the order in which the tuples are given does not matter. However, we could exploit this freedom to impose a specified order on each dimension’s member list. The order then imposed the sequence in which the cells are listed in a particular pivot-table. Any permutation order on the members on each dimension gives a particular pivot-table. The ordering is easy to specify: Given an initial order of the members, one only need to attach a permutation vector to each dimension.

## 5.3 Selection

Another operation in showing a cube as a particular pivot-table is **selection**. One can dice a cube first and then display the diced cube as a pivot-table. Another way to view this is to attach a Boolean on each member, and set its value to TRUE if the particular member should be displayed in the pivot-table and FALSE else wise.

## 5.4 The aggregates within the pivot-table

Often it is useful to view the **aggregates** together with the cube’s data. In our terminology we first *size* the cube with its aggregates and then show the *sized* cube as a pivot-table. Another way again is to integrate this information into the displaying operations: given a cube, we add (1) a permutation, (2) define a  $h$  and  $k$  and an (3) aggregate-op and now (4) a Boolean which says whether the pivot-table should be extended by the aggregates or not.

Normally however, there is no need to compute all aggregated cubes in order to display them in the pivot-table. Let’s explain this in the example of a 2-dimensional and then of a 3-dimensional cube. Figure 18 shows in the left a pivot-table with  $h=1$ . All three aggregated cubes are visible and attached as last row and last column. Hence here we need all aggregated

cubes ( $ab$ ,  $a^*$ ,  $^*b$ , and  $^{**}$  see Figure 19). The middle pivot-table with  $h=0$  displays also all aggregates. But they are not “natural”. Especially, in the last four rows only the last is needed (the total of all totals). A more natural way to display the pivot-table would be the picture on the right.

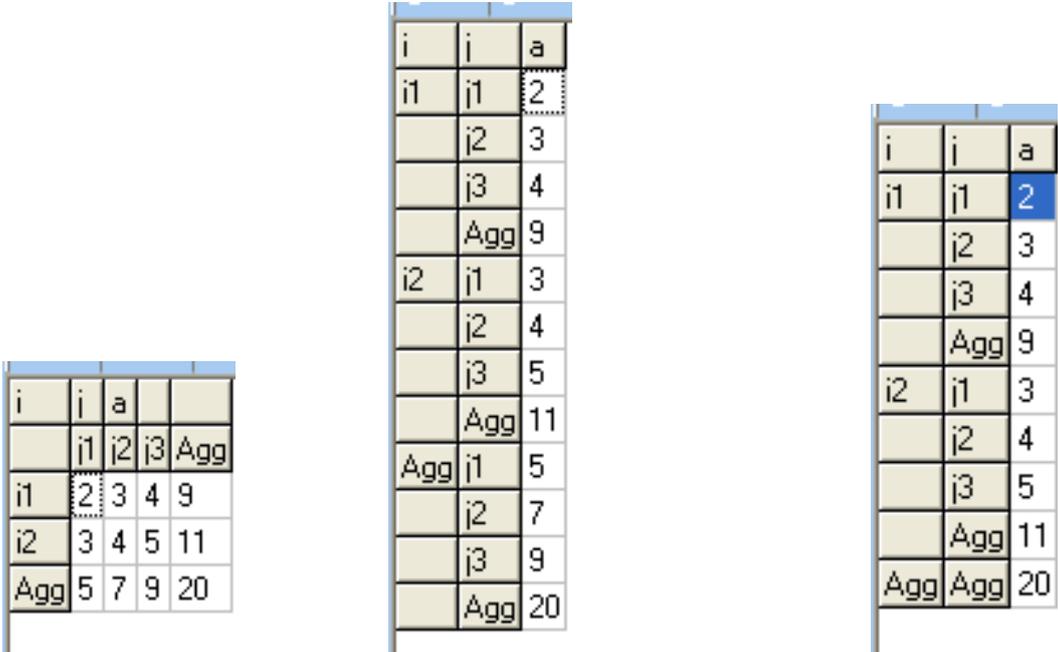


Figure 18: Pivot-tables of a 2-dimensional cube with aggregates.

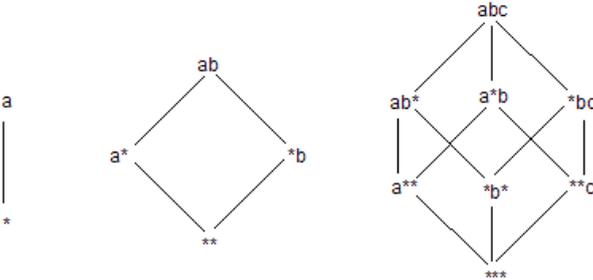


Figure 19: Lattices of the aggregates of cubes with dimension 1,2,3

For a 3-dimensional cube with  $h=0$  only the aggregates  $abc$ ,  $ab^*$ ,  $a^{**}$ ,  $^{***}$  (see Figure 19) are needed, with  $h=1$ , we need  $abc$ ,  $ab^*$ ,  $a^{**}$ ,  $^{***}$ , and  $a^*b$ ,  $^*b^*$ . The rule is the following: We need all aggregates along maximally two paths from  $abc$  to  $^{***}$  in the lattice. It is easy to find them. If we have a 5-dimensional cube, for example, with the dimensions  $abcde$  (the top node of the lattice) then we partition the dimensions into the vertically and horizontally displayed dimensions for a particular pivot-table (say  $h=2$ , then we have  $abc|de$ ). The two aggregates needed – following the paths in the lattice – are :  $ab^*|de$  and  $abc|d^*$ . Hence we follow the path in which the stars (\*) are filled from right to left beginning with the very last entry and beginning with the entry left to |. Following the path down to  $^{*****}$  this way, generates to paths and the corresponding aggregates to calculate and integrate into the pivot-table are given by collecting them in the two paths. Now we also see, why in the cases of  $h=0$  (all vertically) and  $h = n - k$  (all horizontally displayed) a single path is needed only.

It is easy to see, why this works in general. Suppose, you add an aggregate in which a dimension appears after a \* in the list before | (for example  $ab...^*p|...$ ), then rows would be added to the pivot-table with the format (let  $p_1$  be a member of  $p$  in the example):

.... | **Agg** | **p1** | ....

which is exactly what we won't have to be there. Likewise, if an aggregate is added in which a dimension appears after a \* in the list after the | (for example *ab...p|...\*x*), then columns would be added to the pivot-table with the format (let *x1* be an member of *x* in the example):

...  


---

**Agg**  


---

**x1**  


---

 ...

which is again exactly what we won't have to be in the pivot-table.

### 5.5 The aggregate Operators

Until now only mentioned SUM as an aggregate operator. The most useful are:

SUM	sums the cells values along a dimension
COUNT	counts the (non-empty) cells along a dimension
AVERAGE	calculates the average along a dimension
MAX	returns the maximal value along a dimension
MIN	returns the minimal value along a dimension
VARIANCE	calculates the statistical variance along a dimension
STD.DEVIATION	calculates the standard deviation along a dimension

It is possible in a pivot-table to display more than one aggregate. All we need is to calculate the same aggregates along the two paths in the lattice with the different aggregate operators.

### 5.6 Formatting

A pivot-table is – first of all – a 2-dimensional representation of a datacube. The parts of the tables can be thought to be “printed” in rows and columns, in vertically/horizontally arranged cells as shown in Figure 20 where the parts are just displayed in the grid without formatting.

Product						
Tiere	Periods	TimeLag	AMOUNT			
		7-9	9-11	11-14	14-18	SUM
Alpha	Per_1	0	3	86	20	27
	Per_2	67	31	16	37	42
	Per_3	8	47	7	84	5
	SUM	29	91	36	77	32
Betas	Per_2	69	84	71	30	16
	Per_3	32	46	24	82	27
	SUM	48	14	87	28	77
SUM	SUM	97	49	88	82	2

Figure 20: An unformatted layout of a pivot-table.

However, not all cells in the grid have the same meaning. While some “cells” in the pivot-table display the name of the dimensions, others display the data. Basically, a pivot-table can be partitioned into 4 sections: (1) a header, where the dimensions are displayed, (2) the member names of the dimensions to identify a row and a column (3) the aggregates which are “SUM” rows/columns, and (4) the data part. The formatting of these sections is independent from the layout. However, we are free to enforce visually by colors and other attributes the different sections. An example is shown in Figure 21. Different formatting could be chosen.

Product						
Tiere	Periods	TimeLag	AMOUNT			
		7-9	9-11	11-14	14-18	SUM
Alpha	Per_1	0	3	86	20	27
	Per_2	67	31	16	37	42
	Per_3	8	47	7	84	5
	SUM	29	91	36	77	32
Betas	Per_2	69	84	71	30	16
	Per_3	32	46	24	82	27
	SUM	48	14	87	28	77
SUM	SUM	97	49	88	82	2

Figure 21: A formatted layout of a pivot-table.

Another kind of formatting is *data formatting*: (1) the data can be formatted along a mask like ##.### (with three decimals, if they are numbers), (2) certain data can be shown in a different color (for example if they are negative), (3) The data can be shown as percent of a total, or as difference from a given value, etc.

### 5.7 Excel-Pivot Table

The spreadsheet software Excel offers the functionality of pivot-tables.

Sum of a		k				
i	j	k1	k2	k3	k4	Grand Total
i1	j1	1	2	3	4	10
	j2	2	4	6	8	20
	j3	3	6	9	12	30
i1 Total		6	12	18	24	60
i2	j1	2	4	6	8	20
	j2	4	8	12	16	40
	j3	6	12	18	24	60
i2 Total		12	24	36	48	120
Grand Total		18	36	54	72	180

The table here is a representation of a 3-dimensional datacube (i,j,k) (see Figure 15) as pivot-table with the specifications:

$$perm : (1,2,3), h = 1, k = 0, op = SUM$$

The dimensions *i* and *j* are displayed vertically and the dimension *k* is displayed horizontally. The aggregates attached are: *ijk, ij\*, i\*\*, \*\*\*, i\*k, \*\*k*.

The next pivot-table has the same datacube as source. The only parameter changed is *k*.

j	(All)
---	-------

The dimension  $j$  has been “taken out” and the specification is:

$perm : (1,3,2), h = 1, k = 1, op = SUM$

Sum of a	k				Grand Total
i	k1	k2	k3	k4	
i1	6	12	18	24	60
i2	12	24	36	48	120
Grand Total	18	36	54	72	180

The dimension  $i$  is displayed vertically and  $k$  horizontally.

In Excel pivot-table, the power comes from drag and drop the dimensions to (1) switch the order in one direction (horizontally/vertically), (2) switch between vertical and horizontal (3) “take out/in” a dimension. One can hide/show aggregates, select/unselect members, move members to change their order, format cells.

However there are important disadvantages in Excel pivot-table handling: (1) The table is a one-way construct, one cannot change any data, (2) certain formats get lost if the table is manipulated by *pivoting*, (3) it takes quite a time to understand, what you can do, many operations can be done by many different ways.

## 6 Representation of Datacube and their operations in LPL

LPL is designed to define and manipulate datacubes. The dimensions must be modeled as SETs and a datacube then is a multi-indexed entity in LPL. For example, to define a 3-dimensional cube one needs the declaration of four entities: 3 SETs, representing the dimensions and a PARAMETER (or a VARIABLE, or whatever is indexed).

```
SET i:=/i1 i2/; j:=/j1 j2 j3 /; k:=/k1 k2 k3 k4/;
PARAMETER a{i,j,k} := i*j*k;
```

The different operations on datacubes can be implemented in various ways depending often from the context.

Slicing:

```
PARAMETER b{j,k} := a['i1',j,k];
```

Dicing:

```
PARAMETER c{i,j,k|a>=10} := a[i,j,k];
```

Sizing:

```
SET I:=/i1 i2 i3/;
PARAMETER d{I,j,k|a>=10} := if(I in i, a[I,j,k], SUM{i} a[I,j,k]);
```

Rising:

```
set i:=/1:3/; j:=/a b c/; k:=/1 2 3 a b c/; l:=/1:2/;
parameter a{i}:=i; b{j}:=10*j;
c{k}:=if(k<=#i,a[k],b[k-#i]);
d{l,i}:=if(l=1,a[i],b[i]);
```

The generation of a pivot-table is a function of an extended WRITE statement (not yet implemented in the LPL4.50. The input information that an algorithm must have to generate a particular pivot-table is:

1. a datacube
2. a permutation of the dimensions ,  $h$  ,  $k$ , aggregate operator
3. a order for the members of each dimensions
4. a Boolean for each member indicating of selecting it or not
5. Formatting: (1) of the cell: mask, alignment, font, border, pattern, (2) of the value: as percent, as difference etc., (3) depend on an expression (p.e., negative number with another color).

How to specify these requirements within the LPL language is not yet fully clear.

## 7 Conclusion

*Data Viewing* is an important subject whenever mass of data is involved. For this we use *reporting*, *data browsing* and *data editing* tools. When uniformed data are stored along several dimensions, then we may pack them into datacubes. These data are best *viewed* and eventual *edited* through pivot-tables, the 2-dimensional representation of datacubes.

This paper tries to give a unified theory on datacube and pivoting. It was shown that all data operations in OLAP can be reduced to *slicing*, *dicing*, *sizing*, and *rising* operations in a multidimensional datacube.

The goal is to reduce all kind of proposed operations in OLAP to a few operations in manipulating datacubes. This gives also a new view in implementing OLAP tools. It is, furthermore, important to note that all aspects of data viewing of unified mass of data stored as datacubes can be accessed through *pivot-tables*. Pivot-tables are easy to understand (if implemented correctly) and easy to manipulate – at least from the point of view of the user. This paper shows that – given a datacube – a few operations and options determine a particular pivot-table. If the user understands these few operations, it is easy to use them. Unfortunately, Excel pivot-tables do not have these properties.

## 8 References

- Arbor Software Corporation, [1996], *Arbor Essbase*, see [<http://www.arborsoft.com>].
- Gray P., Watson H.J., [1998] *Decision Support in the Data Warehouse*, Prentice Hall, New Jersey.
- Informix Inc. [1997], *The INFORMIXMetaCube Product Suite*. see [<http://www.informix.com>].
- Kimball R., [1996], *The Data Warehouse Toolkit*, J. Wiley Sons, New York.
- Kurz A., [1999]; *Data Warehousing*, MITP-Verlag, Bonn.
- MicroStrategy Inc., [1995], *Relational OLAP: An Enterprise-Wide Data Delivery Architecture*. White Paper, see [<http://www.strategy.com>].
- MicroStrategy Inc., [1997], *MicroStrategy's 4.0 Product Line*, see [<http://www.strategy.com>].
- Red Brick Systems Inc., [1997]. *Red Brick Warehouse 5.0*. see [<http://www.redbrick.com>].

