

AN INTRODUCTION TO

MULTIDIMENSIONAL
DATABASE
TECHNOLOGY



Kenan Technologies

Table of Contents

INTRODUCTION.....	3
WHAT IS A MULTIDIMENSIONAL DATABASE?	4
RELATIONAL DATABASE STRUCTURE	4
CROSS TAB VIEWS OR ARRAYS	5
INCREASINGLY COMPLEX RELATIONAL TABLES	7
MULTIDIMENSIONAL SIMPLIFICATION.....	8
PERFORMANCE ADVANTAGES	10
ADDING DIMENSIONS	10
REAL WORLD BENEFITS.....	11
WHERE MULTIDIMENSIONAL TECHNOLOGY ISN'T APPROPRIATE.....	12
FEATURES OF MULTIDIMENSIONAL DATABASES.....	15
MULTIDIMENSIONAL DATA VIEWS: ROTATION	15
MULTIDIMENSIONAL DATA VIEWS: RANGING.....	17
MULTIDIMENSIONAL DATA VIEWS: HIERARCHIES, ROLL-UPS & DRILL DOWNS	18
MULTIDIMENSIONAL VIEWS: QUERIES.....	21
MULTIDIMENSIONAL DATA VIEWS: CLIENT/SERVER ARCHITECTURE AND MULTI-USER ACCESS.....	23
MULTIDIMENSIONAL COMPUTATIONS	23
THE TIME DIMENSION.....	24
SPARSE DATA HANDLING.....	25
REAL WORLD APPLICATIONS	26
MULTIDIMENSIONAL DATA INTEGRATION.....	26
RELATIONAL, MULTIDIMENSIONAL AND SPREADSHEET STRUCTURES.....	27
THE RIGHT DATA STRUCTURE FOR THE RIGHT JOB	27
INFORMATION RETAILING	29

INTRODUCTION

The information industry has witnessed a steady evolution in database power and flexibility. From flat file and hierarchical to relational and distributed relational technologies, data structures have evolved to match more closely the way people visualize and work with data.

Accompanying this proliferation of database technologies is a growing realization that there is no single "best" data structure for all applications within an enterprise. Many companies are, therefore, abandoning their search for the holy grail of the globally applicable database and are instead selecting the most appropriate data structure on a case-by-case basis from a palette of "standard" database structures.

The latest step in the evolution of databases is the multidimensional database. Springing from econometric research conducted at MIT in the 1960s, the multidimensional database has matured into the database engine of choice for data analysis applications. This application category is widely recognized today as OLAP (On Line Analytical Processing). The central reason for the multidimensional database's rise to corporate necessity is simple: it facilitates flexible, high performance access and analysis of large volumes of complex and interrelated data—even when that data spans several applications in different parts of the organization. Today's corporate computing environments require powerful database applications optimized for analysis.

Aside from its inherent ability to integrate and analyze large volumes of enterprise data, the multidimensional database offers a good conceptual fit with the way end-users visualize business data. Most business people already think about their businesses in multidimensional terms. Managers tend to ask questions about product sales in different markets over specific time periods, for example; and financial officers think about revenues and expenses in different business units for different accounting periods.

Working within a multidimensional data structure is also familiar. A monthly Profit and Loss (P&L) statement with its row and column format is an example of a simple two-dimensional data structure. A three-dimensional database might be a stack of these P&L worksheets: one for each month of the year. With the added third dimension, end-users can more easily examine P&L items across time for trends. Insights into business operations can be gleaned and powerful analysis tools such as forecasting and statistics can be applied to examine relationships and project future opportunities.

This paper introduces and examines multidimensional database technology and clarifies the need to add this important option to the corporate database "tool kit" as the foundation for decision support applications. It describes the differences between multidimensional database technology and traditional relational database technology, and concludes with a discussion of the concept of Information Retailing, a new paradigm for software applications that deliver corporate data tailored to end-user information needs.

WHAT IS A MULTIDIMENSIONAL DATABASE?

A multidimensional database is a computer software system designed to allow for the efficient and convenient storage and retrieval of large volumes of data that is (1) intimately related and (2) stored, viewed and analyzed from different perspectives. These perspectives are called dimensions.

Getting answers to typical business questions from raw data often requires viewing that data from various perspectives. For example, an automobile marketer wanting to improve business activity might want to examine sales data collected throughout the organization. The evaluation would entail viewing historical sales volume figures from multiple perspectives such as:

- Sales volumes by model
- Sales volumes by color
- Sales volumes by dealership
- Sales volumes over time

Analyzing the Sales Volumes data from any one or more of the above perspectives can yield answers to important questions such as:

What is the trend in sales volumes over a period of time for a specific model and color across a specific group of dealerships?

Having the ability to respond to these types of inquiries in a timely fashion allows managers to formulate effective strategies, identify trends and improve their overall ability to make important business decisions. Certainly, relational databases could answer the question above, but query results must also come to the manager in a meaningful and timely way. End users needing interactive access to large volumes of data stored in a relational environment are often frustrated by poor response times and lack of flexibility offered by relational database technology and their SQL query building tools. What follows now is an explanation of the reason for their frustration and a set of examples that assist in comparing the multidimensional database to its most common alternative—the relational database.

RELATIONAL DATABASE STRUCTURE

The table on the facing page illustrates a typical relational database representation of a Sales Volumes dataset for the Gleason automobile dealership. The data in our relational table is stored in "records." Each record corresponds to a row of the table and each records is divided into "fields." The fields in a table are arranged vertically in columns. In our example, the fields are SALES VOLUMES, COLOR and MODEL. The SALES VOLUMES field contains the data we wish to analyze. The COLOR and MODEL fields contain the perspectives we will analyze the data from. Thus, the first record in the table below tells us that six sales were made for blue mini vans.

SALES VOLUMES FOR GLEASON DEALERSHIP

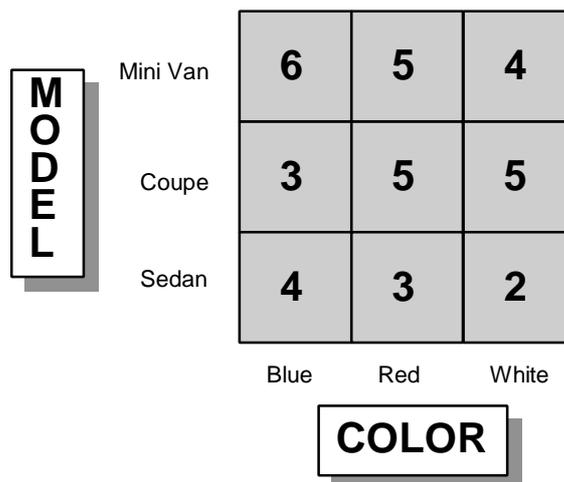
MODEL	COLOR	SALES VOLUME
MINI VAN	BLUE	6
MINI VAN	RED	5
MINI VAN	WHITE	4
SPORTS COUPE	BLUE	3
SPORTS COUPE	RED	5
SPORTS COUPE	WHITE	5
SEDAN	BLUE	4
SEDAN	RED	3
SEDAN	WHITE	2

If we examine the dataset by columns (fields), we discover that the first field, MODEL, ranges across only three possible values: MINI VAN, SPORTS COUPE and SEDAN. The second field, COLOR, also ranges across three possible values: BLUE, RED and WHITE.

CROSS TAB VIEWS OR ARRAYS

There is an alternative way of representing this data. The figure below displays what is commonly known as a "cross tab" view, or data matrix. In this representation the SALES VOLUMES figures are located at the x-axis/y-axis intersections of a 3x3 matrix.

Sales Volumes



This matrix is an example of a two-dimensional "array." An array is the fundamental component of a multidimensional database.

In an array, each axis is called a dimension (one of our data perspectives), and each element within a dimension is called a position. In this example, the first dimension is MODEL; it has three positions: MINI VAN, SEDAN and COUPE. The second dimension is COLOR; it also has three positions: BLUE, WHITE and RED. The Sales Volume figures are located at the intersections of the dimension positions. These intersections are called cells and are populated with our data or measures.

There are several reasons why the array is a more efficient and effective means of presenting our Sales Volume dataset than the relational table we previously examined. First, a great deal of information is gleaned immediately upon direct inspection of an array. In our example, an end user recognizes straight away that there are exactly two dimensions of three positions each. It is much more difficult to make this same observation from examination of the relational table. Second, the array conveniently groups "like" information in columns and rows. For example, all Sales Volume figures for SEDANs are lined up and, if desired, can be added very quickly for a total or quickly compared to get an immediate idea of the relative popularity of each SEDAN COLOR. Likewise, all Sales Volume figures for each COLOR are lined up and can be quickly added to determine the most popular selling models.

In short, our multidimensional array structure represents a higher level of organization than the relational table. The structure itself contains much valuable "intelligence" regarding the relationships between the data elements because our "perspectives" are imbedded directly in the structure as dimensions as opposed to being placed into fields. For example, the structure of our relational table can only tell us that there are three fields: COLOR, MODEL and DEALERSHIP. The relational structure tells us nothing about the possible contents of those fields. The structure of the array, on the other hand, tells us not only that there are two dimensions, COLOR and MODEL, but it also presents all possible values of each dimension as positions along the dimension. Because of this structured presentation, all possible combinations of perspectives containing a specific attribute (the color BLUE, for example) line up along the dimension position for that attribute. This makes it very easy not only to access or retrieve data according to the specific data type we are interested in, but it makes data browsing and manipulation highly intuitive to the end-user. As a result, this "intelligent" array structure lends itself very well to data analysis.

This same "intelligent" array structure that presents information to the end user in a highly organized fashion, also offers quantifiable performance advantages. To begin to understand this, we must recognize that a query or any other kind of data manipulation task is simply a request to select and organize data in a particular fashion (Sales Volume for all BLUE SEDANs, for example). Unlike the relational structure which must scroll through each individual record to identify all possible matching records, the multidimensional array structure "meets the query half way" by presenting data in a pre-organized fashion. The performance advantages of the multidimensional array structure will be elaborated on in a moment.

INCREASINGLY COMPLEX RELATIONAL TABLES

Thus far we have examined two dimensional datasets as viewed in relational tables and in arrays. This can easily be extended to a third dimension. We will now add a new field called DEALERSHIP (with three possible values) to our relational table:

SALES VOLUMES FOR ALL DEALERSHIPS

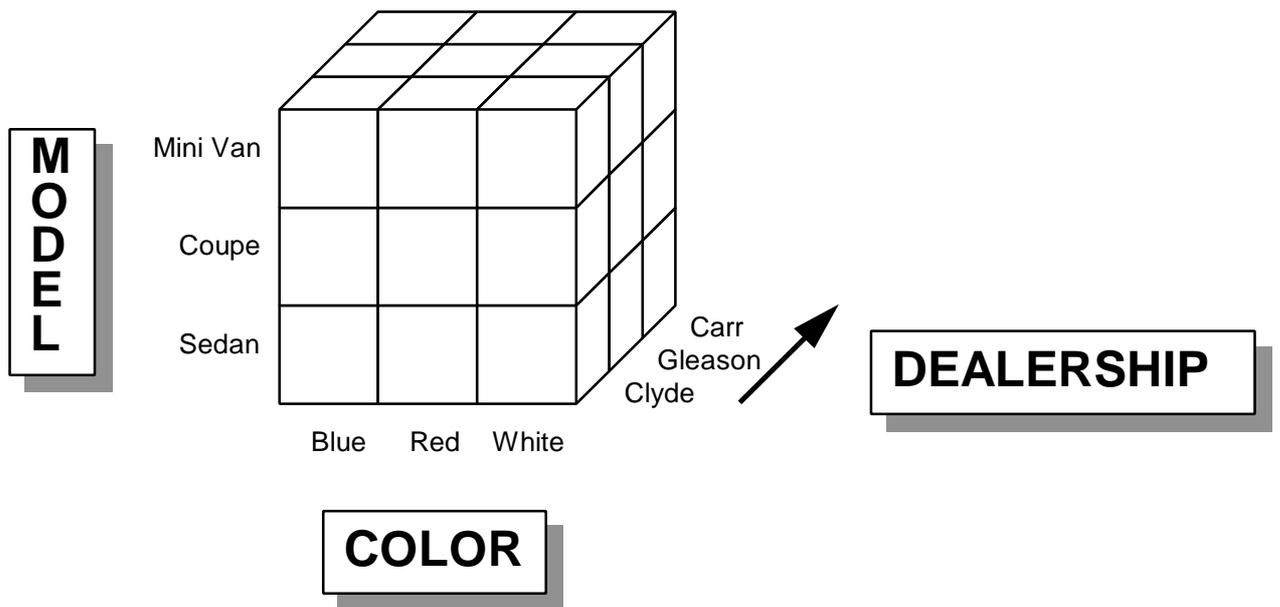
MODEL	COLOR	DEALERSHIP	VOLUME
MINI VAN	BLUE	CLYDE	6
MINI VAN	BLUE	GLEASON	6
MINI VAN	BLUE	CARR	2
MINI VAN	RED	CLYDE	3
MINI VAN	RED	GLEASON	5
MINI VAN	RED	CARR	5
MINI VAN	WHITE	CLYDE	2
MINI VAN	WHITE	GLEASON	4
MINI VAN	WHITE	CARR	3
SPORTS COUPE	BLUE	CLYDE	2
SPORTS COUPE	BLUE	GLEASON	3
SPORTS COUPE	BLUE	CARR	2
SPORTS COUPE	RED	CLYDE	7
SPORTS COUPE	RED	GLEASON	5
SPORTS COUPE	RED	CARR	2
SPORTS COUPE	WHITE	CLYDE	4
SPORTS COUPE	WHITE	GLEASON	5
SPORTS COUPE	WHITE	CARR	1
SEDAN	BLUE	CLYDE	6
SEDAN	BLUE	GLEASON	4
SEDAN	BLUE	CARR	2
SEDAN	RED	CLYDE	1
SEDAN	RED	GLEASON	3
SEDAN	RED	CARR	4
SEDAN	WHITE	CLYDE	2
SEDAN	WHITE	GLEASON	2
SEDAN	WHITE	CARR	3

It is obvious that the addition of the third field (DEALERSHIP) with three possible values (CLYDE, GLEASON, CARR) has made this relational table an even more awkward vehicle for presenting data to the end user.

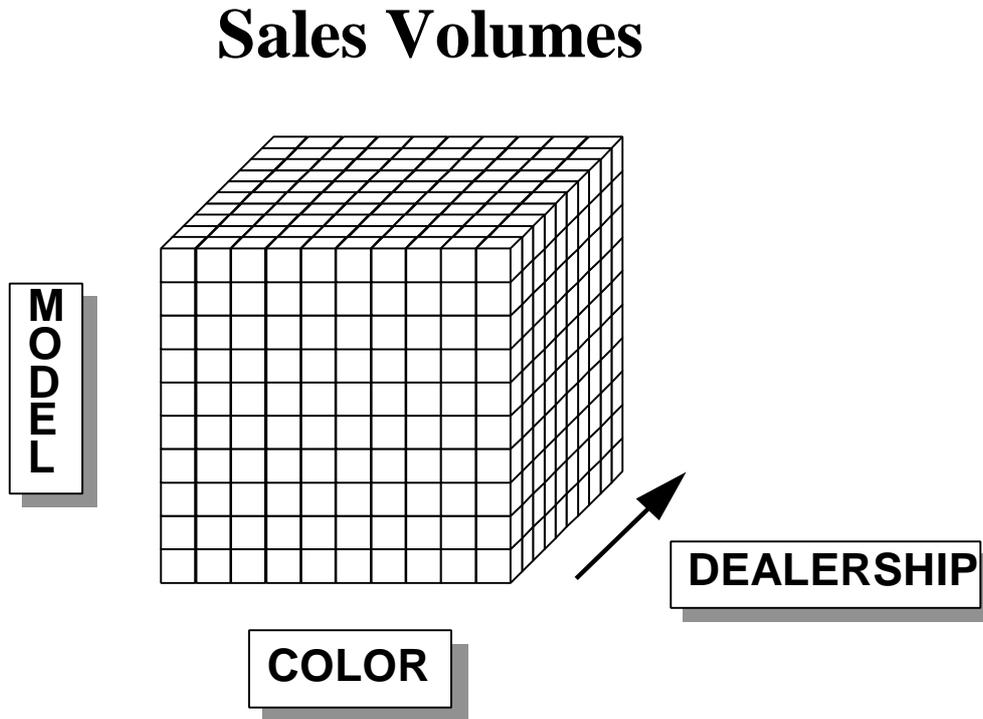
MULTIDIMENSIONAL SIMPLIFICATION

In a multidimensional structure, however, the new DEALERSHIP field translates directly into a third dimension with three positions. We now have a 3x3x3 array containing 27 cells. Once again, the end user is able to view data along presorted dimensions with the data arranged in an inherently more organized, and accessible fashion than the one offered by the relational table.

Sales Volumes



Let us now consider a larger three-dimensional array. Each of our three dimensions now has 10 positions along each dimension. Conceptually, this array would look like the 10x10x10 cube below:



In relational table format, this array translates into a table of 1000 records!

SALES VOLUMES FOR ALL DEALERSHIPS

MODEL	COLOR	DEALERSHIP	VOLUME
MINI VAN	BLUE	CLYDE	2
MINI VAN	BLUE	GLEASON	2
MINI VAN	BLUE	CARR	2
MINI VAN	RED	CLYDE	1
MINI VAN	WHITE	GLEASON	3
•			
•			
•			
RECORD NUMBER....	998		
RECORD NUMBER....	999		
RECORD NUMBER....	1000		

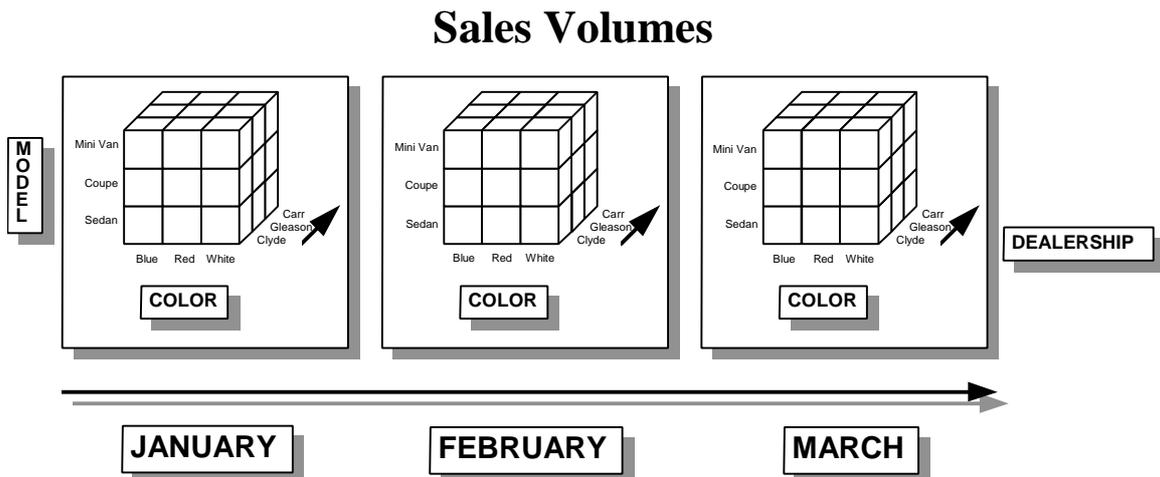
PERFORMANCE ADVANTAGES

Using the 10x10x10 array as an example, let us consider the performance advantages of using a multidimensional structure for data retrieval. A user wants to know the Sales Volume figure when CAR TYPE=SEDAN, COLOR=BLUE, and DEALERSHIP=GLEASON. A relational system might have to search through all 1000 records just to find the right record. In contrast the multidimensional structure has more "knowledge" about where a particular piece of data lies. The multidimensional system only has to search along three dimensions of 10 positions each to find the matching "record" or cell. This is a maximum of 30 position searches for the array versus a maximum of 1000 record searches for the table. If we assume that the average search equals half the maximum search, we are looking at 15 versus 500 searches—a 3300% improvement in performance!

Let us examine another comparison. Suppose that we want to find the total sales of MODEL=SEDAN across all color types and dealerships. In the relational world, it is almost certain that nearly all 1000 records will have to be searched to extract those records where MODEL=SEDAN. In our three-dimensional array, however, we only have to sum the contents of one 10x10 "slice" of the array.

ADDING DIMENSIONS

Our three dimensional example can be extended to four dimensions by adding a time dimension to indicate the month of the year in which a sale was made. Visualizing a fourth dimension is more difficult than visualizing three. Imagine twelve boxes into which our three dimension array can be placed. When in the JANUARY box, the cells of the array contain Sales Volume figures for JANUARY. When in the FEBRUARY box, the cells of the array contain Sales Volume figures for the month of FEBRUARY, and so on.



This paradigm can be extended to five or more dimensions.

REAL WORLD BENEFITS

So far, the evidence presented for multidimensional performance has been conceptual. In actual implementations, these "conceptual" arguments for performance are borne out by benchmarks indicating that multidimensional databases often perform at least an order of magnitude better than relational databases for queries against multidimensional data. Data manipulation that requires minutes to run in a relational environment in many cases requires only seconds to run in a multidimensional environment. The performance benefits are particularly impressive for queries designed to generate cross tab views of data. The performance advantages offered by multidimensional technology facilitates the development of interactive decision support applications that can be impractical in a relational environment.

It must be noted that any data manipulation action possible with a multidimensional database is also possible using relational technology. Multidimensional databases offer several advantages, however:

- **Ease of Data Presentation and Navigation:** Intuitive spreadsheet-like data views are the natural output of multidimensional databases. Obtaining the same views in a relational world requires the end user to either write complex SQL queries or use an SQL generator against the relational database to convert the table outputs into a more intuitive format. Even for end users well skilled in SQL, some forms of output, such as ranking reports (i.e. top ten, bottom 20%), simply cannot be performed with SQL at all!
- **Ease of Maintenance:** Multidimensional databases are characterized by extreme ease of maintenance. Because data is stored in the same way as it is viewed (i.e. according to its fundamental attributes), no additional overhead is required to translate user queries into requests for data. To provide this same level of intuitiveness, relational databases use indexing and sophisticated joins which require significant maintenance and storage.
- **Performance:** Multidimensional databases achieve performance levels that are difficult to match in a relational environment. These high performance levels enable and encourage OLAP applications. Comparable performance levels can be approached in a relational environment through database tuning (indexing and keys), but the database cannot be tuned for all possible ad hoc queries. Performance is query specific, resulting in lost flexibility. Tuning also requires the resources of an expensive Database Specialist.

WHERE MULTIDIMENSIONAL TECHNOLOGY ISN'T APPROPRIATE

You may be asking yourself : Why can't I take any dataset in relational format, call it multidimensional and layer it into a multidimensional array? There is no reason why this cannot be done. There are, however, performance related reasons for not placing certain types of datasets in multidimensional arrays. Take, for example, the relational table below.

It contains personnel data:

PERSONNEL

LAST NAME	EMPLOYEE#	EMPLOYEE AGE
SMITH	01	21
REGAN	12	19
FOX	31	63
WELD	14	31
KELLY	54	27
LINK	03	56
KRANZ	41	45
LUCUS	33	41
WEISS	23	19

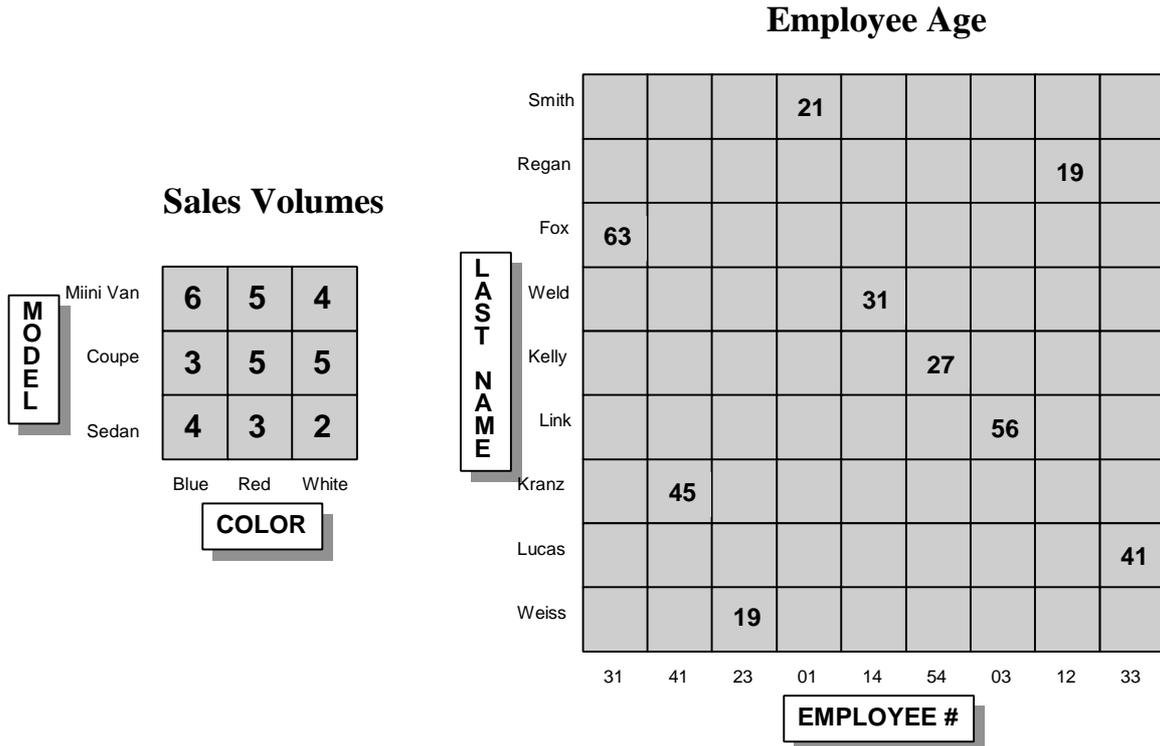
We can set up a multidimensional array to hold these nine records of information by selecting LAST NAME and EMPLOYEE# as our two dimensions. AGE would fall along the intersections of the positions of the two dimensions. Observation shows (not surprisingly) that there are no common last names and no common employee numbers between records. Thus, each of the two dimensions has nine positions yielding a 9x9 array containing 81 cells—yet only 9 of the 81 cells actually contain an AGE value (or occurrence)! Let us compare this data structure with our first example:

SALES VOLUMES FOR GLEASON DEALERSHIP

MODEL	COLOR	VOLUME
MINI VAN	BLUE	6
MINI VAN	RED	5
MINI VAN	WHITE	4
SPORTS COUPE	BLUE	3
SPORTS COUPE	RED	5
SPORTS COUPE	WHITE	5
SEDAN	BLUE	4
SEDAN	RED	3
SEDAN	WHITE	2

Our first model, also containing nine records, has only three possible values for each field—yielding only three positions along each dimension for an array of 3x3. Thus, this multidimensional array has nine cells each containing a value.

When cast in cross-tab data view, the two contrasting arrays look like the following:



The data in the personnel database example is not multidimensional because there is no inherent interrelationship between the elements of the different records. In other words, no last name is matched to more than one social security number and no social security number is matched to more than one last name. As we can see, this results in a sparsely populated 9x9 multidimensional array. In contrast, the data in our original example is very multidimensional in that there is a record (or sales volume) figure corresponding to every possible combination of the values in the MODEL field and COLOR field. This results in a completely populated 3x3 array.

For performance reasons, we do not want to store our personnel dataset in a multidimensional array. In relational form, we require a maximum of only nine searches to locate a record. In array form, two dimensions of nine positions each must be searched for a total of 18 searches. Thus, performance suffers. Compare this to our auto sales dataset which requires nine searches in relational form but only three searches along each

of two dimensions for a total of six searches in multidimensional form. The relative performance advantages of storing multidimensional data in a multidimensional array increase as the size of the dataset increases. Similarly, the relative performance disadvantages of storing non-multidimensional data in a multidimensional array increase as the size of the dataset increases.

Aside from performance considerations, there is no inherent value to storing non-multidimensional data in a multidimensional database. Remember that multidimensional databases are designed for ease and performance in manipulating and analyzing complex data structures—data structures where there are large numbers of interrelationships. Generally speaking, a user accessing the personnel database would not need this kind of analytic power. For example, as was said earlier, it is highly unlikely that this user would want to sum the ages of the employees or observe the trend in employee age as EMPLOYEE# increases! Because there are a very limited number of meaningful relationships between the data elements of the personnel database, and the information content of this database primarily resides in the information content of its individual data elements, it is not a good candidate for multidimensional database technology.

In contrast, our sales volume dataset has a great number of meaningful interrelationships. A user of the sales volume dataset will frequently want to compare sales of one model to sales of another, total sales for each color, or look at the sales volume trend over time for a specific model and color. The interrelationships of the sales volume dataset are more important to the end user than are the individual data elements themselves. These are the characteristics of a dataset that should be stored in a multidimensional database.

This brings us to a fundamental observation on the value of data to an enterprise: *The greater the number of inherent interrelationships between the elements of a dataset, the more likely it is that a study of those interrelationships will yield business information of value to the company.* Most companies have limited time and resources to devote to analyzing data. It is, therefore, crucial that these highly interrelated dataset types be placed in a multidimensional data structure for greatest ease of access and analysis.

There are numerous examples of applications that are ideal for multidimensional technology. The following list provides a sample:

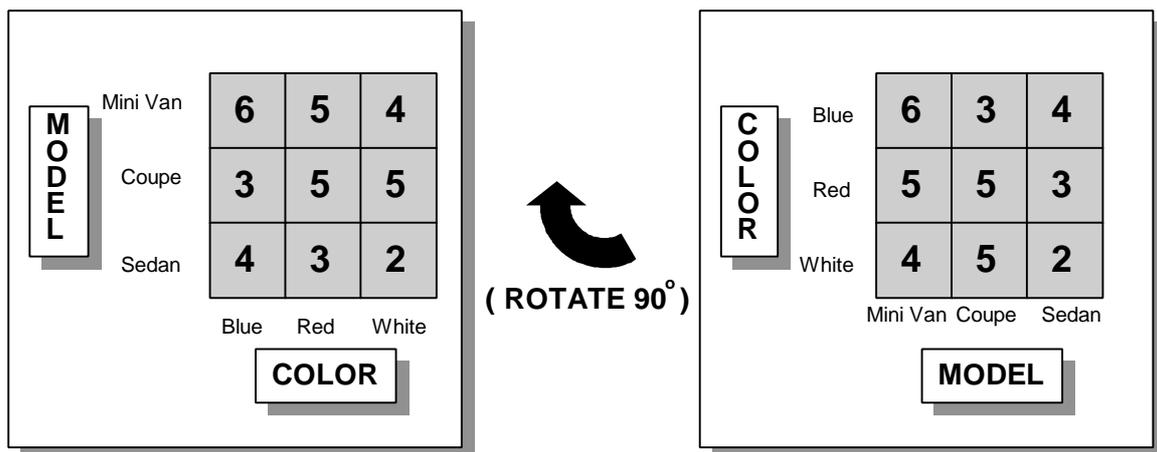
- Financial Analysis and Reporting
- Budgeting
- Promotion Tracking
- Quality Assurance and Quality Control
- Product Profitability
- Survey Analysis

FEATURES OF MULTIDIMENSIONAL DATABASES

MULTIDIMENSIONAL DATA VIEWS: ROTATION

As illustrated previously, the arrangement of data in a multidimensional array has inherent usefulness. Unlike the view of a relational table, the view of an array explicitly shows dimensions, positions along dimensions, intersections and cell values. In fact, viewing a multidimensional array is similar to viewing a spreadsheet. To get a better understanding of the power of viewing an array versus a relational table, consider how frequently most end users create spreadsheet views of data, then consider how infrequently the typical end user directly browses the contents of a relational database! The immediate usefulness of data in array format is capitalized upon in multidimensional databases. Consider the previously introduced two-dimensional array. The view is Sales Volume by MODEL by COLOR. The data is immediately available for viewing. Multidimensional databases allow direct browsing of arrays, just as an end user would browse a spreadsheet. In a relational environment, this kind of structured view must be imitated through a complex query against the database. After browsing the first view, the end user might decide that a more useful view would be SALES VOLUME by COLOR by PRODUCT. In a relational environment, this new view requires a complex query, or sort. In the multidimensional environment, this new view is termed a "rotation" of the old one. The view is rotated 90 degrees clockwise without any rearrangement of the data! The ease and speed with which a rotation can be performed is another example of the inherent advantages of manipulating data in a multidimensional array. The illustration below demonstrates how rotation works in a multidimensional environment.

Sales Volumes



View #1

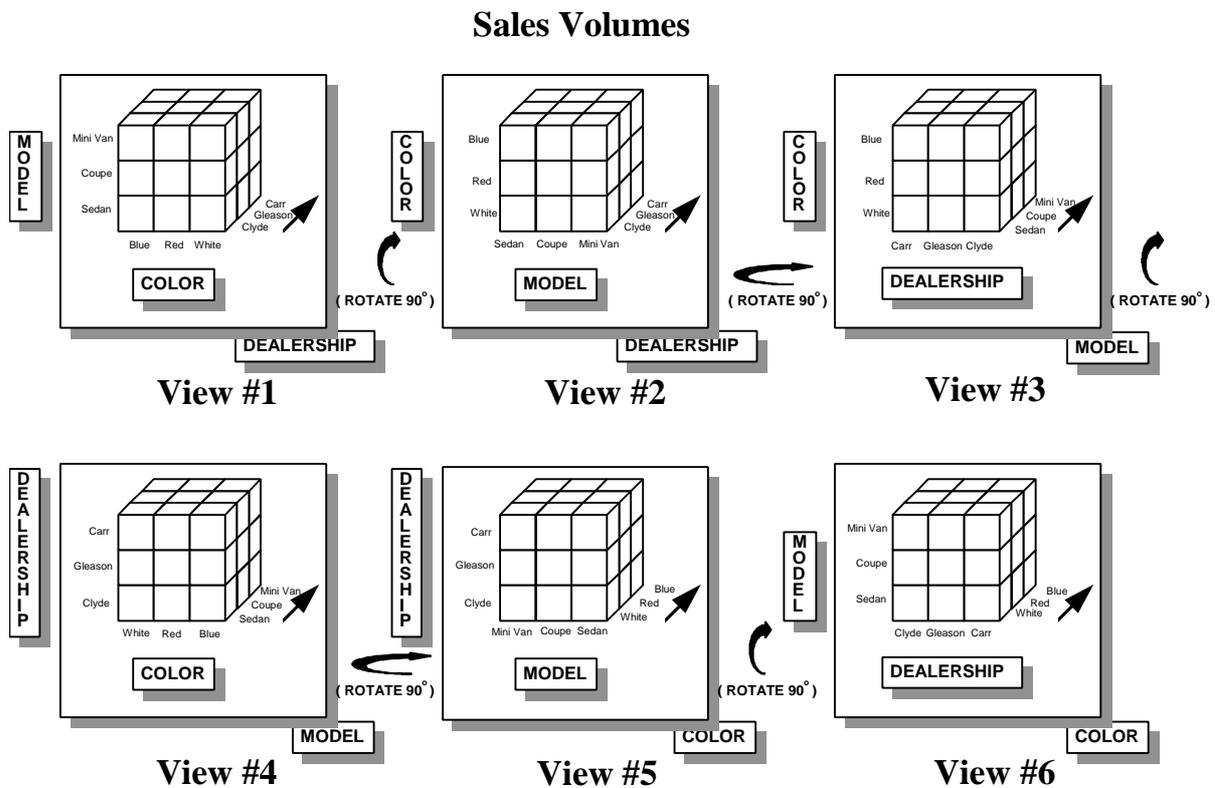
View #2

The two dimensional array above has two possible views: MODEL x COLOR or COLOR x MODEL. In both cases, the intersection of the two dimensions yields the Sales Volumes. Our previous example had three-dimensions: MODEL, COLOR, and

DEALERSHIP. This three dimensional cube has six possible views that may be of interest to a user:

- #1) MODEL by COLOR (with DEALERSHIP in the background)
- #2) COLOR by MODEL (with DEALERSHIP in the background)
- #3) COLOR by DEALERSHIP (with MODEL in the background)
- #4) DEALERSHIP by COLOR (with MODEL in the background)
- #5) DEALERSHIP by MODEL (with COLOR in the background)
- #6) MODEL by DEALERSHIP (with COLOR in the background)

The graphic below illustrates how these six views can be accessed through multidimensional rotation.



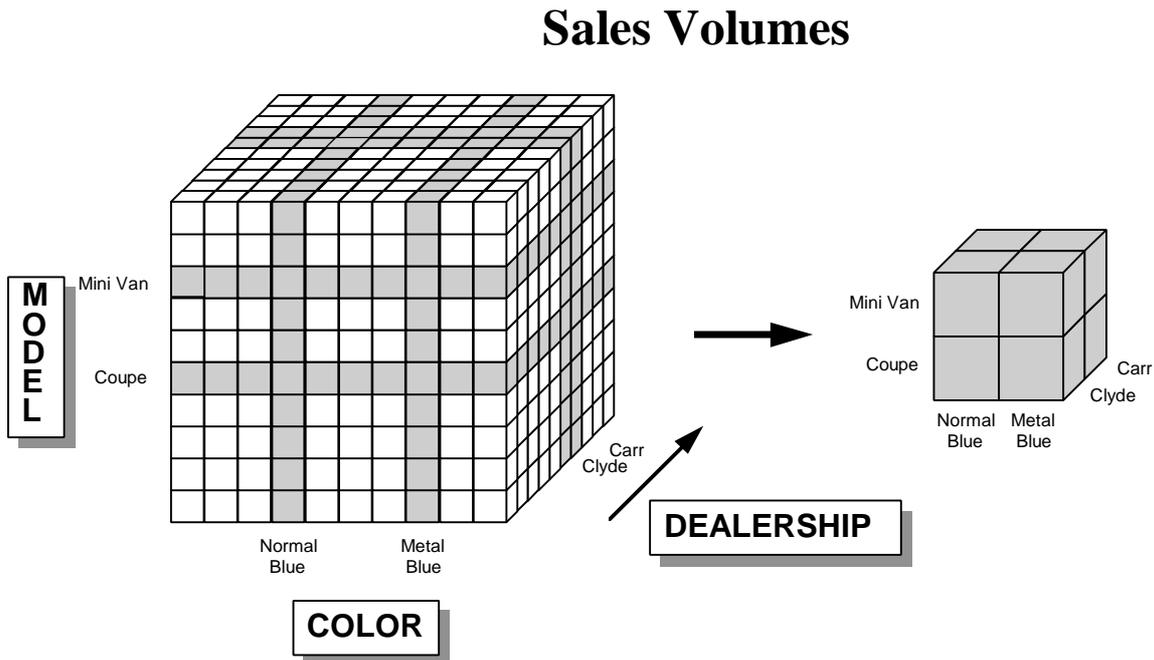
The ability to easily access different views of multidimensional data provides added flexibility to various business users, each of whom brings different questions and requirements to the data. Any of the six views displayed above can be accessed through simple rotation. Because the multidimensional database stores data so that rotations are simple, no rearrangement of the data is required. Rotation is also referred to as "data slicing," because each rotation yields a different "slice" or two dimensional table of data. We have seen that the number of possible views (slices) increases exponentially with the number of dimensions. So, a two-dimensional array has two views. A three dimensional array has six views. A four-dimensional array has 24 views. A five dimensional array has 120 views. Multidimensional database technology makes all views equally accessible to the end user with none of the inefficiencies of sorting against a relational database.

MULTIDIMENSIONAL DATA VIEWS: RANGING

A well implemented multidimensional database allows the end user to quickly range in on the exact view of the data required. Take our previous three-dimensional model with ten positions along each dimension. The end user may want to determine how Sales Volumes for models painted with a new metallic paint called METAL BLUE compare to Sales Volumes for the same models painted with NORMAL BLUE. Our end user knows that only the SPORTS COUPE and the MINI VAN have received the new paint treatments. The end user also knows that only two of the dealerships have an unconstrained supply of these cars: CARR and CLYDE. Through an operation called ranging, the end user selects the desired positions along each dimension:

- for the model dimension SPORTS COUPE and MINI VAN
- for the dealership dimension CARR and CLYDE
- for the color dimension METAL BLUE and NORMAL BLUE

We can illustrate this feature through the following graphic of the array:



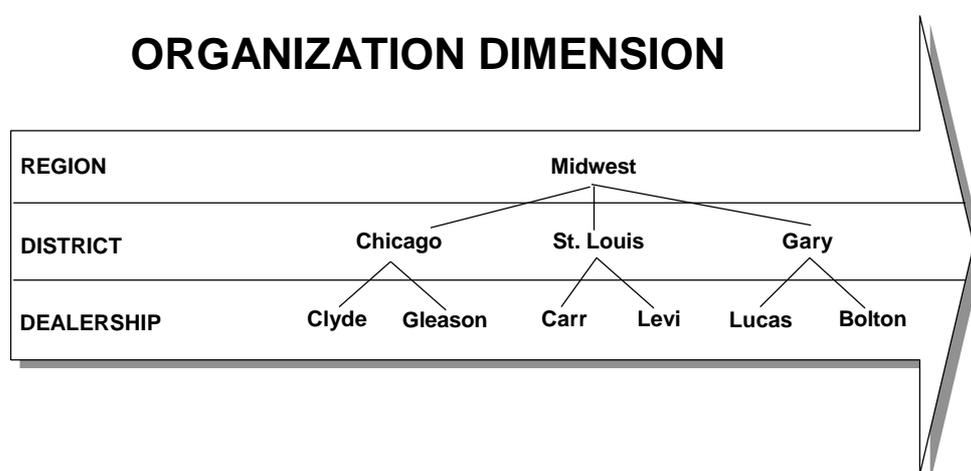
The reduced array can now be rotated and used in computations just as its larger parent was. Ranging is analogous to a complex query in a relational environment. Ranging is also referred to as "data dicing," because the data is scoped down to a subset grouping. The performance of ranging operations in a multidimensional array is once again significantly better than the performance of the analogous query in a relational environment. Again, the reason is that relatively few resource consuming searches are required in the multidimensional environment to obtain the desired view of the data.

MULTIDIMENSIONAL DATA VIEWS: HIERARCHIES, ROLL-UPS & DRILL DOWNS

As previously described, end users often need several different views of the same data (i.e., Sales Volume by MODEL versus Sales Volume by DEALERSHIP). Often, however, views are very similar (i.e., Sales Volume by DEALERSHIP versus Sales Volume by DISTRICT). In this last example, the two different views of Sales Volumes are both related to levels in the sales organization. A user may first want to view Sales Volumes at the DISTRICT level. If Sales Volumes at that level do not meet expectations, the user may then move down to the DEALERSHIP level to determine which dealerships are not meeting quota. There is a natural relationship between Sales Volumes at the DEALERSHIP level and Sales Volumes at the DISTRICT level. The Sales Volumes for all the dealerships in a district sum to the Sales Volumes for that district. There are many real-world instances of such different but closely related views of data.

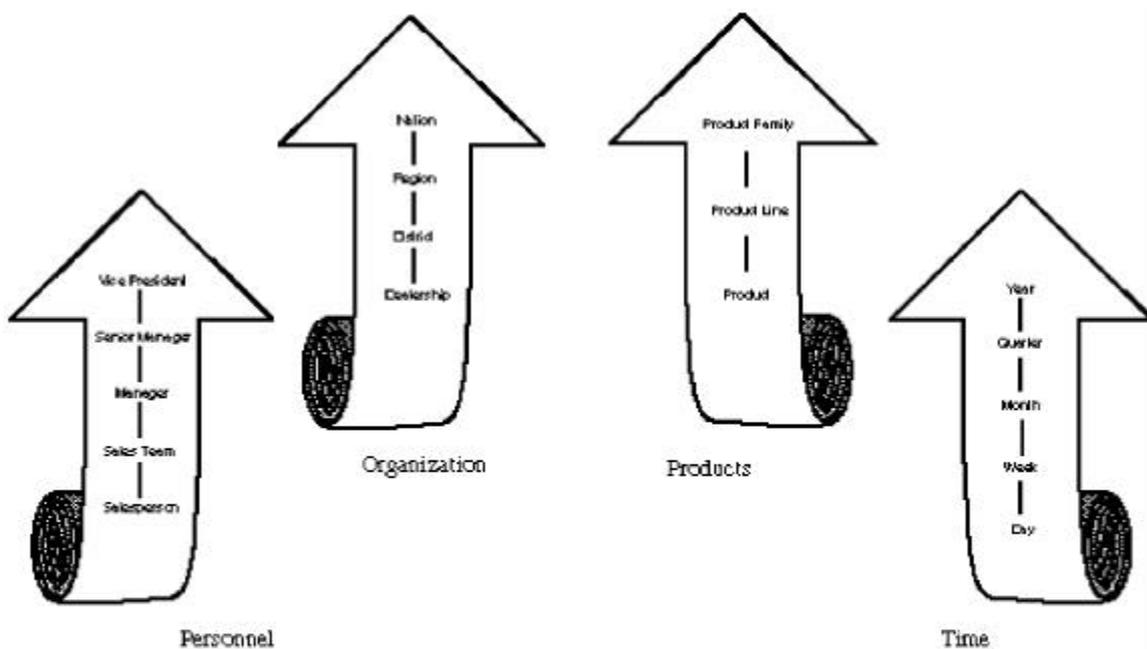
Multidimensional database technology is specially designed to facilitate the handling of these natural relationships. Although it is possible to create separate, independent dimensions for both DEALERSHIP and DISTRICT in the above example, a more elegant and powerful solution is to define two related aggregations on the same dimension. One aggregation is DEALERSHIP and the other is DISTRICT. These two aggregations are then perceived as being part of the same dimension. In our example, all dealerships fall under a sales district. Thus, the DISTRICT is a higher level aggregation within an overall dimension we can call ORGANIZATION. When we relate each position along the DEALERSHIP aggregation to a position along the DISTRICT aggregation, we have defined a *hierarchy* within the overall ORGANIZATION dimension.

A well implemented multidimensional database should enable the creation of such hierarchies. In fact, the best implementations allow definition of multiple levels within a hierarchy. For example, the DEALERSHIP level is related to DISTRICT level, which can be related to a REGION level. The graphic below illustrates this three level hierarchy.



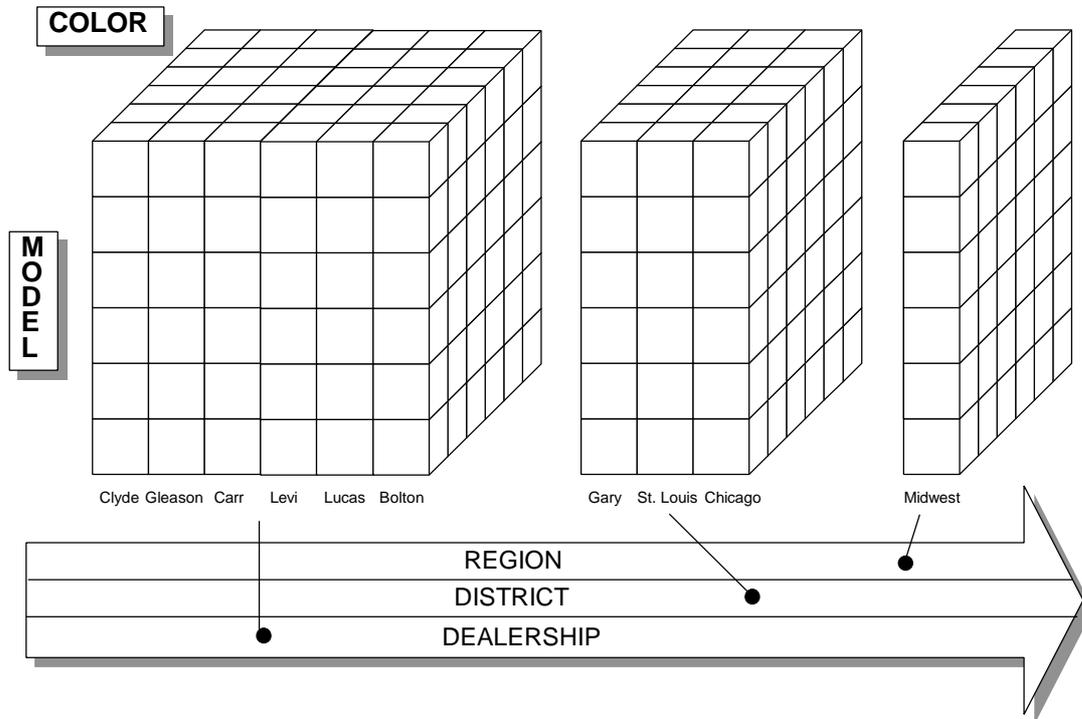
The alternative in a relational database environment is to create a field for each level on the hierarchy or to create separate tables defining the relationships. This approach is less efficient and less elegant than the multidimensional approach which innately understands hierarchical relationships between different aggregations within dimensions.

For end-users, the ability to define hierarchies allows for very quick data manipulations and detailed analysis along different levels within all the dimensions of a multidimensional array. Moving up and moving down levels in a hierarchy is referred to as "roll-up" and "drill-down."



Rolling-up and Drilling-down Through Multiple Dimensions. In the previous graphic, we saw that users can roll-up and drill-down through a single dimension, ORGANIZATION. Well designed multidimensional databases also allow users to roll-up and drill down through multiple dimensions concurrently. Thus, in this example, an end-user could hold the positions MANAGER, DISTRICT and PRODUCT constant, while drilling-down or rolling-up through sales figures over the TIME dimension.

Sales Volumes



Each view, as defined by the chosen levels along each dimension, can then be rotated or ranged as previously described. The analog of rolling-up or drilling-down in a relational world requires re-running a query for each new view. The advantages are once again performance related.

A well designed multidimensional database also allows for the implementation of an important concept called *multiple simultaneous hierarchies*. For example, the end user may wish to roll-up along the ORGANIZATION dimension in two different fashions to get answers to two different business questions. The first question is: what are the Sales Volumes by sales region? The second question is: what are the Sales Volumes by distribution point? For business reasons, the DEALERSHIPS falling under a REGION may not fall under the same DISTRIBUTION POINT. Two different (but simultaneous) hierarchies are required to roll-up from DEALERSHIP to SALES REGION, and from DEALERSHIP to DISTRIBUTION POINT.

SALES HIERARCHIES

SALES ORG.	DISTRIBUTION ORG.	MODEL	COLOR TYPE
REGION	IMPORT POINT	SIZE	PAINT BASE
DISTRICT	DISTRIBUTION POINT	TYPE	COLOR
DEALERSHIP	DEALERSHIP	MODEL	

MULTIDIMENSIONAL VIEWS: QUERIES

The high degree of structure in a multidimensional array translates into a very simple and efficient query language. Not only is the language more intuitive, the query output is more immediately useful to the end user. Let us use our previous Sales Volume dataset of three dimensions with three positions each. If we wish to display Sales Volume by MODEL for each DEALERSHIP (summing SALES VOLUME by COLOR for each MODEL within each DEALERSHIP), the following simple command results in the desired output:

```
PRINT TOTAL.(SALES_VOLUME KEEP MODEL DEALERSHIP)
```

This command can also be issued by pointing and clicking on a graphical representation of the multidimensional model. In either case the output looks like the following two dimensional table:

DEALERSHIP

MODEL	CLYDE	GLEASON	CARR
MINI VAN	7	5	6
SPORTS COUPE	4	6	8
SEDAN	3	8	12

The format of the query output corresponds to the way data is stored in the array. The output is of immediate value. Trends emerge and comparisons are easily made.

Let us compare our multidimensional query with an SQL query against the same data in a relational database:

```
SELECT MODEL, DEALERSHIP, SUM(SALES_VOLUME)  
FROM SALES_VOLUME  
GROUP BY MODEL, DEALERSHIP  
ORDER BY MODEL, DEALERSHIP
```

This is the same data and the same request, but the relational environment command structure is more complex and less intuitive. The output of this query is:

MODEL	DEALERSHIP	SUM(SALES_VOLUME)
MINI VAN	CLYDE	7
MINI VAN	GLEASON	5
MINI VAN	CARR	6
SPORTS COUPE	CLYDE	4
SPORTS COUPE	GLEASON	6
SPORTS COUPE	CARR	8
SEDAN	CLYDE	3
SEDAN	GLEASON	8
SEDAN	CARR	12

This is also the same data, but in a much less intuitive and useful format. No trends emerge.

By using a report writer in addition to SQL, it is possible to rearrange the data slightly to get the following report:

```
MINI VAN
    CLYDE      7
    GLEASON    5
    CARR       6
SPORTS COUPE
    CLYDE      4
    GLEASON    5
    CARR       8
SEDAN
    CLYDE      3
    GLEASON    8
    CARR      12
```

This format is somewhat better than the raw SQL output, but it is still not nearly as helpful as the cross tab output from our multidimensional query. It does not, for example, lend itself well to trend analysis. Meaningful patterns are not immediately visible.

Unfortunately, even this small improvement comes at the expense of extra work in addition to the SQL statement.

To actually get two-dimensional output from our SQL query similar to multidimensional output would require writing a custom routine in "C" code with embedded SQL commands. This is not a task many end users (or information technologists for that matter!) wish to take on. In summary, the ease of data access and the usefulness of query outputs are both greatly enhanced by the inherent power of the multidimensional query language.

MULTIDIMENSIONAL DATA VIEWS: CLIENT/SERVER ARCHITECTURE AND MULTI-USER ACCESS

One of the key advantages to a well implemented multidimensional database is its ability to allow many users across an enterprise to simultaneously access and view the same data. A well implemented multidimensional database enables concurrent users to define their own unique and independent views of the same data structure. Each end user is able to rotate, range, drill-down and roll-up through the database without impacting the view of any other user. The multidimensional database should also support multiple users without experiencing a degradation in performance. Multidimensional databases must be open as well, supporting multiple front-end tools for application development as well as data browsing and analysis. Finally, multidimensional databases need to work in a true client-server way, supporting the distribution of data and processing between client and server machines.

MULTIDIMENSIONAL COMPUTATIONS

As previously mentioned, data stored in a multidimensional array offers a high degree inherent organization that greatly facilitates analysis. For example, in our two-dimensional example, all of the Sales Volume figures for COLOR=BLUE are in a neat row. Similarly, all of the Sales Volume figures for CAR=SEDAN are conveniently lined up. This built-in organization makes arithmetic operations that are typically of interest to end users inherently easier to generate (simpler query structure) and more efficient to perform.

For example, calculating the total number of BLUE cars sold is a simple matter of summing one column in our array. The same operation in a relational model requires searching for the appropriate records one-by-one, retrieving the Sales Volume figures and then summing them.

Of course, mathematical operations are often more sophisticated than this. Multidimensional databases are well equipped to handle demanding mathematical functions. Therefore, applications based on multidimensional database technology typically have one dimension defined as a "business measurements" dimension. Along this dimension, we might find a position for Sales Volume figures and also positions for other business measurements such as Budget Volume (Planned), Profit, Revenue, etc. Let us place this business dimension in the context of a three-dimensional model where the first two dimensions are MODEL and COLOR. In this example, each position of the BUSINESS MEASUREMENT dimension is a plane (or two-dimensional slice) of business measurement data. In analyzing this data, it is unlikely that an end user would be interested in simply summing along a column or row. It is more probable that the user would want to perform an arithmetic operation across entire slices of business measurements. For example, an end-user might want to calculate the VARIANCE between BUDGET and ACTUAL figures. In our multidimensional array this operation is elegantly accomplished by dividing the ACTUAL array by the BUDGET array. In a multidimensional database this operation is

performed very efficiently because arrays, in this case ACTUAL and BUDGET, can be treated like individual cells, making mathematical operations on very even large arrays extremely fast. This can be carried out even across arrays of varying size and shape. For example, some cars may be sold at some dealerships and not at others. In this case, the BUDGET and ACTUAL arrays will not look exactly the same for given MODELS. Powerful multidimensional databases can conform one array to the other by intelligently ignoring the cells where there is no data. In the relational world, operations such as dividing a table of ACTUAL by another table called BUDGET requires a tremendous amount of resources because the operation requires joining two large tables or performing a self join over each individual cell or record. In a large dataset, this query can take hours if not days to run!

Sales Volumes

	Actual	Budget	Variance
Mini Van	11	10	0.1
Coupe	16	12	0.33
Sedan	8	10	-0.2
	16	16	0.0

BUSINESS MEASUREMENTS

We conclude that a well implemented multidimensional database, in addition to providing more efficient data access, integrates computational tools very tightly with the database structure. This feature adds an extra measure of speed and efficiency to computations against multidimensional data. In the relational environment, tight integration of computational tools with the data structure is rare. This is yet another reason to consider implementing a multidimensional database for data access and analysis intensive applications.

THE TIME DIMENSION

Getting answers to many commonly asked business questions requires analysis of data over periods of time. The relational world has no special provision for handling time effectively. Well designed multidimensional databases, however, have special provisions for handling the time dimension. In many multidimensional databases TIME is a predefined dimension with predefined hierarchies for rolling-up and drilling-down across

days, weeks, months, years and other special time periods such as lunar months and fiscal years. This exhaustive treatment of time as a predefined dimension has two significant advantages. First, it eliminates the effort required to build sophisticated hierarchies every time a database is set up. Second, because the time dimension is hard coded into the foundation of the multidimensional data engine, extra performance dividends can be netted when performing frequent timeseries analyses such as lags, leads, and forecasts.

SPARSE DATA HANDLING

Our previous example of the personnel data placed in a multidimensional array provides an extreme example of what is called a "sparse array." Sparse arrays are datasets that have few populated cells. Other multidimensional arrays we have presented show datasets with values at every intersection of the positions along the dimensions. This is the other extreme, a fully populated array. In practice, most multidimensional datasets fall somewhere between the extremes of being completely populated and sparsely populated. For example, the Gleason automobile dealership might sell Blue and Red Mini vans in every District, but may only sell White Mini Vans in two Districts. In a relational table, records for White Mini vans would be present in the Districts where they are sold. In a multidimensional data structure, however, there will still be a cell position for this MODEL, COLOR and DISTRICT even though there is no corresponding sales volume figure. The cell is empty and a special NA value is placed in the cell to indicate this. A cell containing an NA requires as much storage and processing resource as a cell containing an actual value.

In a large array with many dimensions, we may have many NA values for what is still, essentially, a multidimensional dataset. For example, a certain dealership might be closed for renovations for a period of time, or a particular car type might be removed from production in the middle of the year to be replaced with another. These two examples would result in large numbers of NA values in the multidimensional array.

For some multidimensional datasets, the number of NA values can exceed 90%. This is referred to as a sparse array. For reasons of performance optimization and minimization of storage space requirements, it is important to treat this sparse array in a special fashion. Well designed multidimensional databases have special provisions for handling sparse data. The database engine detects blocks of NA values, removes them, and compresses the multidimensional array. This special treatment of sparse arrays dramatically improves performance while reducing storage requirements.

REAL WORLD APPLICATIONS

MULTIDIMENSIONAL DATA INTEGRATION

The notion of separating operations systems or On Line Transaction Processing (OLTP) applications from the more analysis intensive On Line Analytical Processing (OLAP) applications appears to be gathering steam. The reasons are simple. OLAP workload profiles typically have sporadic, yet occasionally intense periods of activity when users launch complex queries and then let the system idle while the data is examined or further manipulated at the workstation. This workload pattern is represented by Figure A. In contrast, transaction oriented OLTP applications typically display a relatively smooth workload profile as shown in Figure B. If the two workloads are mixed, as in Figure C, the OLTP applications tend to become overloaded by the transient workload peaks generated by the OLAP applications. The productivity and patience of end users of the operational applications suffer as a result.

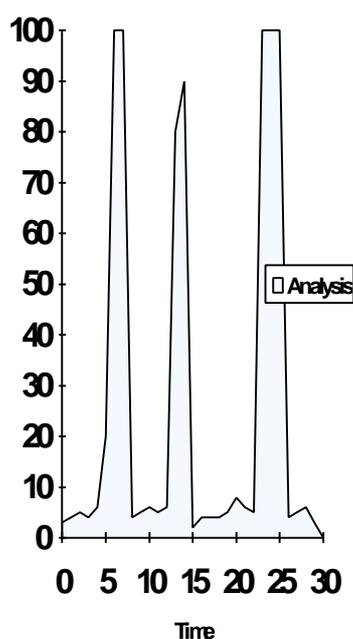


Figure A

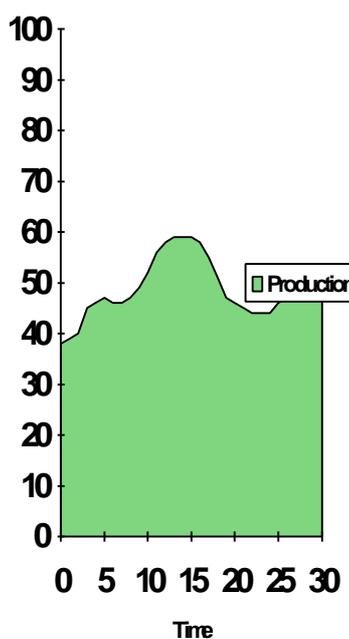


Figure B

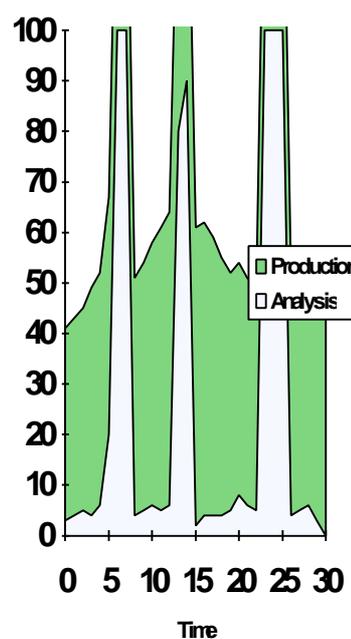


Figure C

Most data analysis environments recognize the desirability of separating analysis from operational applications. The well designed multidimensional database supports this separation by offering a variety of tools for automating the extraction and integration of data from relational and other data structures across the enterprise and then providing a separate environment for data analysis and reporting.

RELATIONAL, MULTIDIMENSIONAL AND SPREADSHEET STRUCTURES

THE RIGHT DATA STRUCTURE FOR THE RIGHT JOB

As previously mentioned, multidimensional database technology is optimized for high-performance data access, manipulation and analysis. Multidimensional technology is not designed to take the place of relational and other database technologies employed for transaction intensive applications. In fact, by providing a staging point for data integration, manipulation and analysis, multidimensional database technology can be seen as a complement to relational database technology. Throughout this paper we have seen examples of why different database structures are appropriate for different applications needs. We have also presented some of the generic capabilities of multidimensional databases. In this section of the paper we will summarize the key advantages of relational, multidimensional and spreadsheet technologies.

Historically, information technologists have selected and rightly prefer relational databases for applications that record and store high volumes of transactions. However, relational databases were never designed to handle high volumes of data analysis. For these applications, a multidimensional database is more ideally suited. Multidimensional data structures are intended to complement existing transaction system including relational technology by providing:

- a staging point for the integration of data extracted from a company's existing data storage systems
- a data structure that maps to end-users' understanding of business issues
- an environment optimized for analysis intensive applications

Spreadsheets are another important database structure used in today's business computing environment. Spreadsheets provide powerful desktop analysis capabilities but are limited to modeling data on a small scale. Moreover, they lack the ability to view data from multiple perspectives.

A relative newcomer to the desktop market are multidimensional spreadsheets, examples of which are Improv, CA-Compete, and Excel 5.0. These products have begun to address end-user access to multidimensional data models. They are, however, severely limited in their ability to:

- Extract and integrate enterprise-wide data from disparate sources
- Analyze large volumes of data (anything over 10MB)
- Provide dynamic data rotation, drill-down and roll-up capabilities
- Support multiple concurrent users and views of the same data models

The following table summarizes some of the Pros and Cons of each technology.

Data Storage	Advantages	Disadvantages
Relational	<ul style="list-style-type: none"> • Established standard • Efficient for transaction processing 	<ul style="list-style-type: none"> • Does not support end users' conceptual view • SQL is difficult and inefficient for creating cross tab views
Multidimensional	<ul style="list-style-type: none"> • Supports end user business queries • Powerful and efficient for comparative analysis • Handles large volumes of data • Easily maintainable 	<ul style="list-style-type: none"> • Not well suited to transaction processing • No established standard • Some data is stored redundantly
Multidimensional Spreadsheet	<ul style="list-style-type: none"> • Extends spreadsheet capabilities to handle rotations • Familiar to end users • Standard desktop application • Superb formatting 	<ul style="list-style-type: none"> • Not scaleable to large data sets • Can be complex for non-spreadsheet users • No drill down or roll-up capabilities • Limited development environment • Can't integrate data from multiple sources

In conclusion, today an alternative approach is needed to meet new demands of information professionals and end-users alike. The need is for transparent desktop access and analysis of enterprise wide data. The technology ideally suited to this need is the multidimensional database. Multidimensional databases offer sophisticated data manipulation and presentation capabilities, a strong conceptual fit with end users understanding of their business data and reporting needs, and performance and programming advantages for analysis and decision support applications.

INFORMATION RETAILING

In this paper we have introduced some of the key concepts and applications of multidimensional database technology. We have also compared and contrasted multidimensional, relational and spreadsheet technologies to illustrate how each can be used most effectively in today's corporate computing environments. This paper postulates that multidimensional databases are uniquely structured to provide a complement to relational and other data storage technologies for analysis intensive applications. These types of applications can be broadly defined by the concept of Information Retailing.

Information Retailing is a paradigm used to describe the benefits offered by a new set of software tools. These tools are designed to extract, integrate and repackage data from disparate corporate data sources, delivering valuable business insights to a broad range of end-users.

The concept of Information Retailing can be perhaps best explained by drawing an analogy to the world of consumer goods. In that world, retail stores provide a valuable service to shoppers by neatly organizing and displaying products in a manageable shopping space. Goods are organized into like categories and generally, the retail outlet is organized to make it easy for consumers to find what they want and even to make the "shopping" experience as pleasurable as possible. Information Retailing software is designed to provide the same type of intuitive and satisfying experience for today's "data shoppers." These shoppers are executives, managers, and analysts who need to have their hands on the pulse of their business' key performance variables. Data consumers expect that the data they need will be accessible in a unified environment and that it will be organized into meaningful categories and names. They also expect to have the opportunity to perform standard and ad-hoc reporting, and that the results will be immediately available. And, of course, these users demand that technical issues be transparent to them. Multidimensional databases respond to these needs by:

- Providing an integrated environment for storing and accessing enterprise data
- Delivering the performance improvements required to perform interactive analysis of this data
- Offering a comprehensive set of tools that create an intuitive environment attractive to end-users.

This paper provided an overview of multidimensional database technology. A companion General Information Guide entitled [Acumate® Product Overview](#) introduces Kenan® Technologies' Acumate ES product and examines the other components of Information Retailing technology that make possible successful delivery of information, not just data, to today's data consumers¹.

¹ Acumate and Kenan are registered trademarks of Kenan Systems Corporation.