

Introduction to OLAP (A beginner's guide to OLAP & the concepts behind it)

Seagate Info Technical Roadmap Series

Overview

OLAP is a term used more and more in industry, yet it is an acronym that is misused as often as it is misunderstood. This document sets out to explain what OLAP really means, how & why it is useful, and how to take advantage of its capabilities. It does not attempt to explain in any detail how to use Crystal Decisions products to create OLAP applications.

The intended audience is anyone who wants to understand more about OLAP & intends to use Crystal Decisions OLAP products in the future. No special technical knowledge is assumed, although a basic understanding of relational databases would be useful.

Contents

OLAP EXPLAINED	2
<i>What is OLAP?</i>	2
<i>Why do we need OLAP?</i>	2
Increasing data storage	2
Data versus Information	3
Data layout	3
OLAP FUNDAMENTALS	5
<i>What is a cube?</i>	5
<i>Multidimensionality</i>	7
Four dimensions and beyond	9
"Slicing & dicing"	10
Nested dimensions	10
<i>Hierarchies & groupings</i>	11
"Drill-down", "Drill-up" & "Drill-across"	12
<i>Consolidated Data</i>	12
Pre-consolidated versus On-Demand	13
Sparse Data	14
<i>Storing the data - ROLAP, MOLAP and HOLAP?</i>	15
ROLAP	15
MOLAP	15
HOLAP	15

OLAP explained

This chapter explains OLAP, why it exists and why it is used in preference to traditional methods of data storage & analysis.

What is OLAP?

OLAP means many different things to different people, but the definitions usually involve the terms "cubes", "multidimensional", "slicing & dicing" and "speedy-response". OLAP is all of these things and more, but it is also a misused & misunderstood term, in part because it covers such a broad range of subjects. We will discuss the above terms in later sections; to begin with, we explain the definition & origin of OLAP.

OLAP is an acronym, standing for "On-Line Analytical Processing". This, in itself, does not provide a very accurate description of OLAP, but it does distinguish it from OLTP or "On-Line Transactional Processing".

The term OLTP covers, as its name suggests, applications that work with transactional or "atomic" data, the individual records contained within a database. OLTP applications usually just retrieve groups of records and present them to the end-user, for example, the list of computer software sold at a particular store during one day. These applications typically use relational databases, with a fact or data table containing individual transactions linked to meta tables that store data about customers & product details.

OLAP applications present the end user with information rather than just data. They make it easy for users to identify patterns or trends in the data very quickly, without the need for them to search through mountains of "raw" data. Typically this analysis is driven by the need to answer business questions such as "How are our sales doing this month in North America?". From these foundations, OLAP applications move into areas such as forecasting and data mining, allowing users to answer questions such as "What are our predicted costs for next year?" and "Show me our most successful salesman".

OLAP applications differ from OLTP applications in the way that they store data, the way that they analyze data and the way that they present data to the end-user. It is these fundamental differences (described in the following sections) that allow OLAP applications to answer more sophisticated business questions.

Why do we need OLAP?

When first investigating OLAP, it is easy to question the need for it. If an end user requires high-level information about their company, then that information can always be derived from the underlying transactional data, hence we can achieve every requirement with an OLTP application. Were this true, OLAP would not have become the important topic that it is today. OLAP exists & continues to expand in usage because there are limitations with the OLTP approach. The limits of OLTP applications are seen in three areas.

Increasing data storage

The trend towards companies storing more & more data about their business shows no sign of stopping. Retrieving many thousands of records for immediate

analysis is a time and resource consuming process, particularly when many users are using an application at the same time. Database engines that can quickly retrieve a few thousand records for half-a-dozen users struggle when forced to return the results of large queries to a thousand concurrent users.

Caching frequently requested data in temporary tables & data stores can relieve some of the symptoms, but only goes part of the way to solving the problem, particularly if each user requires a slightly different set of data.

In a modern data warehouse where the required data might be spread across multiple tables, the complexity of the query may also cause time delays & require more system resources which means more money must be spent on database servers in order to keep up with user demands.

Data versus Information

Business users need both data and information. Users who make business decisions based on events that are happening need the information contained within their company's data. A stock controller in a superstore might want the full list of all goods sold in order to check up on stock levels, but the manager might only want to know the amount of fruit & frozen goods being sold. Even more useful would be the trend of frozen good sales over the last three months.

In order to answer the question "How many frozen goods did we sell today?", an OLTP application must retrieve all of the frozen good sales for the day and then count them, presenting only the summarized information to the end-user. To make a comparison over three months, this procedure must be repeated for multiple days. Multiply the problem by several hundred stores, so that the managing director can see how the whole company is performing and it is easy to see that the problem requires considerable amounts of processing power to provide answers within the few seconds that a business user would be prepared to wait.

Database engines were not primarily designed to retrieve groups of records and then sum them together mathematically and they tend not to perform well when asked to do so. An OLTP application would always be able to provide the answers, but not in the typical few-seconds response times demanded by users.

Caching results doesn't help here either, because in order to be effective, every possible aggregation must be cached, or the benefit won't always be realized. Caching on this scale would require enormous sets of temporary tables and enormous amounts of disk space to store them.

Data layout

The relational database model was designed for transactional processing and is not always the best way to store data when attempting to answer business questions such as "Sales of computers by region" or "Volume of credit-card transactions by month". These types of queries require vast amounts of data to be retrieved & aggregated on-demand, something that will require time & system resources to achieve.

More significantly, related queries such as "Product sales broken down by region" and "Regions broken down by product sales" require separate queries to be performed on the same data set.

The answer to the limitations of OLTP is not to spend more & more money on bigger & faster databases, but to use a different approach altogether to the problem and that approach is OLAP.

OLAP applications store data in a different way from the traditional relational model, allowing them to work with data sets designed to serve greater numbers of users in parallel. Unlike databases, OLAP data stores are designed to work with aggregated data, allowing them to quickly answer high-level questions about a company's data whilst still allowing users to access the original transactional data when required.

OLAP fundamentals

As discussed, OLAP applications are used to solve everyday business questions such as "How many cars did we sell in Europe last month?" or "Are our North American stores throwing away more damaged goods year-on-year?" To answer these questions, large amounts of transactional or base data must be retrieved and then summed together. More subtly, they require a different approach to storing & retrieving the data.

Although different OLAP tools use different underlying technologies, they all attempt to present data using the same high-level concept of the multidimensional cube. Cubes are easy to understand, but there are fundamental differences between cubes and databases that can make them appear more complicated than they really are. This section sets out what cubes are, how they differ from databases and why they provide OLAP applications with more power than the relational database.

Storing data in cubes introduces other new terms & concepts and these are all explained in this section.

What is a cube?

The cube is the conceptual design for the data store at the center of all OLAP applications. Although the underlying data might be stored using a number of different methods, the cube is the logical design by which the data is referenced.

The easiest way to explain a cube is to compare storing data in a cube with storing it in a database table.

Products	Store	Volume
Bulbs	Uptown	40
Bulbs	Midtown	52
Bulbs	Downtown	36
Batteries	Uptown	104
Batteries	Midtown	22
Batteries	Downtown	78
Fuses	Uptown	56
Fuses	Midtown	31
Fuses	Downtown	58

Fig 3.1.1. A relational table containing sales records.

Figure 3.1.1. shows a set of sales records from three electrical stores displayed in a transactional database table. There are two field columns "Products" and "Store" that contain textual information about each data record and a third value column "Volume". This type of table layout is often called a "fact table".

The columns in a table define the data stored in the table. The rows of textual information and numeric values are simply instances of data; each row is a single data point. A larger data set would appear as a table with a greater number of rows.

	Bulbs	Batteries	Fuses
Uptown	40	104	56
Midtown	52	22	31
Downtown	36	78	58

Fig. 3.1.2. The data from figure 3.1.1. as a two-dimensional cube.

Figure 3.1.2. shows the same data now arranged in a "cube". The term "cube" is used somewhat loosely, as this is in fact a two-dimensional layout, often referred to as "a spreadsheet view" as it resembles a typical spreadsheet.

The axes of the cube contain the identifiers from the field columns in the database table. Each axis in a cube is referred to as a "dimension". In this cube, the horizontal dimension contains the product names and is referred to as the "Products dimension". The vertical dimension contains the store names and is referred to as the "Store dimension".

In the database table, a single row represents a single data point. In the cube, it is the intersection between fields that defines a data point. In this cube, the cell at the intersection of Fuses and Midtown contains the number of fuses sold at the midtown store (in this case, 31 boxes). There is no need to mention "Volume" as the whole cube contains volume data.

This co-ordinate driven concept of finding data is the reason why we can't just ignore one of the dimensions in a cube. For example, the question "How many bulbs did we sell?" has no direct meaning with this cube unless it is qualified by asking for data from a particular store.

The term "field" is used to refer to individual members of a dimension, so for example, Uptown is a field in the Store dimension. Notice that the two dimensions contain apparently unrelated fields. Dimensions are usually comprised of the same class of objects, in this example all of the products are in one dimension and all of the stores are in another. Attempting to mix fields between the two dimensions would not work because it would not make sense, it would not be possible to create a unique cell for each data point and any attempt to display the data would also not be possible.

Note that we have avoided using the terms row & column dimension. Although a cube appears to have rows & columns just like a table, they are very different from the rows & columns in a database. In a database, row & column refer to specific components of the data store, in a cube; they simply describe the way the cube is presenting the data. For example, the cube in figure 3.1.2 can also be displayed as in figure 3.1.3., with the dimensions reversed.

Products	Uptown	Midtown	Downtown
Bulbs	40	52	36
Batteries	104	22	78
Fuses	56	31	58

Fig. 3.1.3. The two-dimensional cube reoriented.

Both figures 3.1.2. or 3.1.3. are valid layouts, the important point is that the first diagram shows "Products by Store" and the second shows "Stores by Product". This is one of the advantages of the cube as a data storage object; data can be quickly rearranged to answer multiple business questions without the need to perform any new calculations. A second advantage is that the data could be sorted either vertically or horizontally, allowing the data to be sorted by store or product regardless of the cube's orientation.

From this simple two-dimensional cube, we can now explain some further concepts.

Multidimensionality

In the previous section, we looked at a simple two-dimensional cube. Although useful, this cube is only slightly more sophisticated than a standard database table. The capabilities of a cube become more apparent when we extend the design into more dimensions. Multidimensionality is perhaps the most "feared" element of cube design as it is sometimes difficult to envisage. It is best explained by beginning with a three-dimensional example.

Staying with the data set used in the previous section, we now bring in more data, in the form of revenue & cost figures. Figures 3.2.1. & 3.2.2. show the different ways that the new data could be stored in a table.

Products	Store	Revenue	Cost	Volume
Bulbs	Uptown	\$250	\$115	40
Bulbs	Midtown	\$325	\$145	52
Bulbs	Downtown	\$225	\$155	36
Batteries	Uptown	\$416	\$255	104
Batteries	Midtown	\$88	\$55	22
Batteries	Downtown	\$312	\$180	78
Fuses	Uptown	\$160	\$95	60

Fig 3.2.1. A "degenerate" table layout

All values in a
single column

Products	Store	Measures	Value
Bulbs	Uptown	Revenue	\$250
Bulbs	Uptown	Cost	\$115
Bulbs	Uptown	Volume	40
Bulbs	Midtown	Revenue	\$325
Bulbs	Midtown	Cost	\$145
Bulbs	Midtown	Volume	52
Bulbs	Downtown	Revenue	\$225

Fig 3.2.2. The "canonical" table layout

As can be seen, the degenerate layout results in a wider table with fewer rows while the canonical model results in a narrower table with more rows. Neither layout is particularly easy to read when viewed directly.

The simplest OLAP layout is to create three separate two-dimensional cubes for each part of the data, one for the revenue figures, one for costs and one for volumes. While useful, this layout misses out on the advantages gained by combining the data into a three-dimensional cube. The three-dimensional cube is built very simply by laying the three separate two-dimensional "sheets" (the Volume, Cost & Revenue figures) on top of each other.

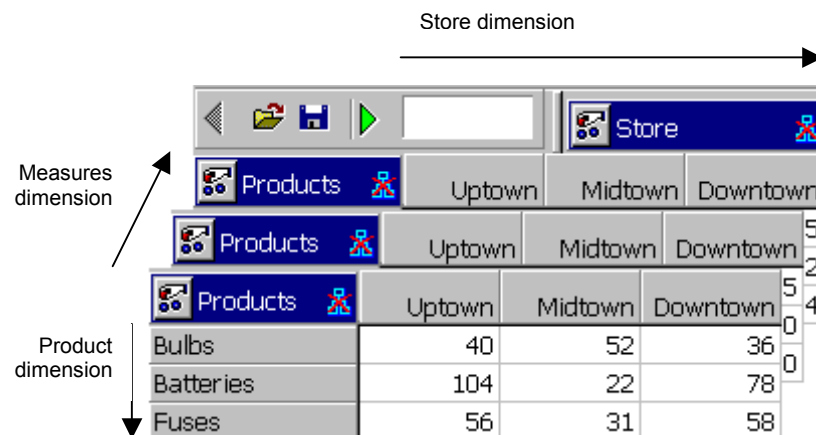


Fig. 3.2.3. The three-dimensional cube.

As can be seen from figure 3.2.3., the three-dimensional layout becomes apparent as soon as the three layers are placed on top of each other. The third dimension, "Measures" is visible as the third axis of the cube, with each sheet corresponding to the relevant field (Volume, Cost or Revenue).

The actual data points are located by using a co-ordinate method as before. In this example, each cell is a value for the revenue, cost or volume of a particular product sold in a particular store.

As before, the data can be reoriented & rearranged, but this time, more sophisticated data rearrangements can be made. For example, the view from the right-hand face of the cube in figure 3.2.3. shows the revenue, cost & volume figures for all products sold in the Downtown store. The view from the topmost face shows the revenue, cost & volume figures for bulbs across all three stores.

This ability to view different faces of a cube allows business questions such as "Best performing product in all stores" to be answered quickly by altering the layout of the data rather than performing any new calculations, thus resulting in a considerable performance improvement over the traditional relational database table method.

Four dimensions and beyond

Although the word "cube" refers to a three-dimensional object, there is no reason why an OLAP cube should be restricted to three dimensions. Many OLAP applications use cube designs containing up to ten dimensions, but attempting to visualize a multidimensional cube can be very difficult. The first step is to understand why creating a cube with more than three dimensions is possible and what advantage it brings.

As we saw in the previous section, creating a three-dimensional cube was fairly straightforward, particularly as we had a data set that lent itself to a three-dimensional layout. Now imagine that we have several three-dimensional cubes, each one containing the same product, region & measures dimensions as before, but with each one holding data for a different day's trading. How do we combine them? We could just add all of the matching numbers together to get a single three-dimensional cube, but then we could no longer refer to data for a particular month. We could extend one of the dimensions, for example the measures dimension could have the fields "Monday's costs" and "Tuesday's costs", but this would not be an easy design to work with and would miss out on the advantages of a multidimensional layout.

The answer is simple, we create a fourth dimension, in this case the dimension "Days" and add it to the cube. Although we can't easily draw such a cube, it is easy to prove the integrity of the design. As stated before, each data point is stored in a single cell that can be referred to uniquely. In our four-dimensional design, we can still point to a specific value, for example the value for revenue from bulbs sold Uptown on Monday. This is a four dimensional reference as it requires a field from four dimensions to locate it:

1. The Revenue field from the Measures dimension.
2. The Bulbs field from the Product dimension.
3. The Uptown field from the Store dimension.
4. The Monday field from the Days dimension.

Without actually having to draw or visualize the whole cube, it is quite easy to retrieve and work with a four-dimensional data set simply by thinking about the specific data cells being requested.

The issue of visualizing the data set leads onto the second step in picturing the cube. Although the cube might have four (or more) dimensions, most applications only present a two-dimensional view of their data. In order to view only two dimensions, the other dimensions must be "reduced". This is a process similar to the concept of filtering when creating an SQL query.

Having designed a four-dimensional cube, a user might only want to see the original two-dimensional layout from figure 3.1.2., Products by Store. In order to display this view, we have to do something to the remaining dimensions Measures & Days. It makes no sense just to discard them as they are used to locate the data. Instead, we pick a single measure & day field, allowing us to present a single two-dimensional view of the cube.

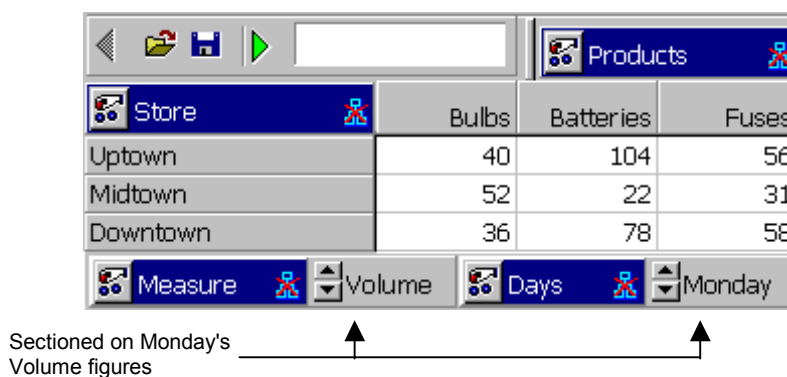


Fig. 3.2.1. Two-dimensional view of a four-dimensional structure.

We have to pick a field from the remaining dimensions because we need to know from which Measures field and which day to retrieve the Product & Store information. The dimensions that don't appear in the view are often referred to as "section" dimensions, as the required view is "sectioned" on a specific field from these dimensions.

Although it is difficult to visualize at this point, it is the dimensions and the fields in those dimensions that define cubes, not the data stored in the cube. A table is often described by the number of columns & rows that it has, while the number of dimensions and the number of fields in each dimension define a cube.

"Slicing & dicing"

This is a phrase used to describe one part of the process used to retrieve & view data stored in an OLAP cube. Because of the size of most real-world OLAP cubes and their complexity, the process of locating the right set of data is often referred to under the heading "Navigation".

As data can only effectively be displayed in a two-dimensional format, the multi-dimensional cube must be restricted into flat "slices" of data. When picking a particular orientation of data in a cube, the user is literally "slicing & dicing" the data in order to view a simple flat layout.

Nested dimensions

Although data can only be viewed in a two-dimensional or flat layout, it doesn't mean that only two dimensions can be fully displayed at one time. It is perfectly possible to display more than one dimension on each axis.

For example, the user might want to see revenue & cost figures for all products sold in each store on Monday. Rather than displaying revenue & cost separately, the Measures dimension can be "nested" inside the Store dimension, displaying both revenue & cost data simultaneously and allowing direct comparison to be made between them. This layout can be seen in figure 3.2.3..

Store	Measure	Bulbs	Batteries	Fuses
Uptown	Revenue	250	416	160
Uptown	Cost	115	255	95
Midtown	Revenue	325	88	140
Midtown	Cost	145	55	80
Downtown	Revenue	225	312	90
Downtown	Cost	155	180	75

Days: Monday

Fig. 3.2.3. Two-dimensional view with nested dimensions.

Hierarchies & groupings

So far, we have only looked at simple dimensions, each containing only a few fields. Real world data sets can create dimensions with many thousands of fields and this introduces two problems. Firstly, dimensions with large numbers of fields can become difficult to manage, so an application requires a way of dividing the fields into groups. Secondly, many users don't want the individual values for each field; they want the total value for all of the fields, or for a particular group. Fortunately, grouping fields to create "hierarchical" dimensions can solve both of these problems.

A hierarchical dimension derives its name from the concept of fields being at different levels, with some fields being at a higher level in a dimension than others. For example, a store might sell three types of bulbs, 60 Watt, 100 Watt and 150 Watt. Rather than always having to list data for each bulb type, a user might want to group the bulbs together as a method of simplifying the data view and so that they can answer questions such as "How many bulbs did we sell today?".

The method used by all OLAP applications is to create a "parent field" that groups together other fields. For example, figure 3.3.1. shows the parent field "Bulbs" and the child fields that comprise it. This parent field is at a higher level in the dimension and is often described as being at a higher hierarchical level.

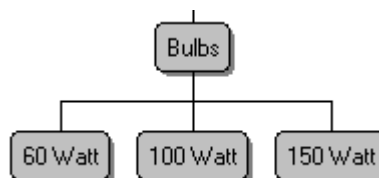


Fig. 3.3.1. The Bulbs hierarchy within the Products dimension.

Once the parent field has been defined, this automatically makes any fields below it into child fields. If the children of the parent field don't have any child fields of their own, then they are referred to as the base fields in the dimension as there are no further levels below.

This grouping together allows both problems to be solved. An application can now sum together data for the three bulb types and store it against the new field "Bulbs". Without the new field, a cube has nowhere to store the values that are created by summing together the values for each individual bulb type. It also

allows an application to simplify its layout by only presenting the parent fields to the user, typically a much smaller list than the full list of base fields.

Note that hierarchies are not restricted to single levels. For example a global computer retailer might begin by grouping stores together by cities, these cities would then sum into countries and the countries sum into continents. A top-level "World" field would realize the final global picture.

"Drill-down", "Drill-up" & "Drill-across"

These are common phrases used when dealing with OLAP applications and are very simple to explain.

If an application only displayed the parent fields in a cube, such as data for the whole of the US and the user wanted to view data for a particular State, then breaking down the US parent field into its children is called "Drilling-down". Conversely, "Drilling-up" refers to the process of selecting a child field and displaying its parent field.

"Drill-across" describes any changes to the sectioned fields. For example, a user might want to change the display of January's budget figures for each department figure to February's. If the original view was sectioned on January, then the change to February is called "Drilling-across".

Consolidated Data

Another of the key OLAP descriptions is the concept of "speedy-response", literally the time it takes for a user to receive the data that they require. As we have already discussed, although database engines can sum data, such as the total number of sales in a region, they struggle to do so. This is because they were not primarily designed to answer such questions and hence, don't store data in a way that makes it efficient to answer them.

OLAP applications, by their very nature are required to provide aggregated or "consolidated" data. This is data derived by grouping together sets of related fields to provide higher-level information. As we have seen, OLAP applications create new fields to group together existing base fields into sensible, manageable groups (hierarchies). The high-level data points referenced by these hierarchical fields rarely exist in the original data sets, so the values must be evaluated by the application.

The technical process of evaluating high-level values from base data is beyond the scope of this document, but the principle is as follows.

Store	Bulbs	Batteries	Fuses	All Products
Uptown	40	104	56	200
Midtown	52	22	31	105
Downtown	36	78	58	172
All Stores	128	204	145	477

Grand Total

Fig. 3.4. A two-dimensional cube with consolidated data.

Consider the two-dimensional cube in figure 3.4.1.. The values in the grey boxes are the original base data values and the values in the white cells are the extra high-level values that are created by the OLAP application. These parent cells usually represent the sum of their child fields' values, so can be evaluated by adding together the data points from the level below.

It is worth mentioning that the top-most cell in the structure (marked in bold) can be evaluated by summing the child fields from either the column or the row dimension if both dimension hierarchies are simple summations. Obviously it is quicker to use the calculation that requires the fewest values to be added. This is another advantage of OLAP data storage; the cube design allows considerable performance gains to be made when generating consolidated data.

Hierarchies that are more complicated than simple additions can also be represented in a dimension, but these are beyond the scope of this document.

Pre-consolidated versus On-Demand

Multidimensional data stores allow OLAP applications to evaluate large amounts of high-level data very quickly, typically several orders of magnitude faster than a database. However, because OLAP data stores can often be so large, evaluating the full set of values can still take minutes or hours to complete. Usually, it is the requirements of the users (in terms of report response times) that drives the decision between the two methods of consolidation, working out the values in advance and storing them or working them out as required.

When application data sets were smaller and computers were less powerful, almost all OLAP applications took the pre-consolidated route. This meant that the OLAP cube would be built periodically (usually daily or weekly) to include recent changes in its figures and then stored somewhere for fast data retrieval. The advantages of rapid retrieval of data & hence speedy reporting response times always outweighed the disadvantages of building a large multidimensional data cube periodically and storing it on disk.

More recently, with faster processors and the emergence of ever larger data cubes, the pressures of disk storage have seen many applications return to the on-demand method, where high-level data is only evaluated as it is required by the users. This is a more complicated method of working, as it typically requires all relevant calculations to complete within seconds so that the users are not aware that the values that they requested did not exist a few seconds earlier.

Assuming that the performance issues can be kept under control, then the on-demand method offers the simple advantage of massively reduced storage requirements that translates directly into cost-savings on disk purchases. More sophisticated OLAP tools allow the application to prestore some of the consolidated data and leave other sections to be evaluated on-demand. This allows the application to get the best from both techniques.

Ultimately, the choice rests on several factors, how much data is being stored, the power of the server on which the application runs and the requirements of the users. Once these factors have been taken into account, a sensible decision can be made on the method used to evaluate the high-level data.

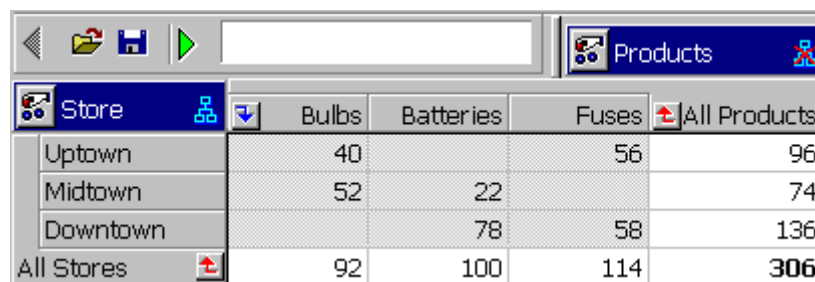
Sparse Data

While really beyond the scope of this document, it is worth briefly discussing the concepts behind sparse data. Sparse data is literally data with "holes" in it. Sparse cubes have gaps in their data where there are no values and are the most common type of cube. Most real world cubes are very sparse, while the remainder are very dense (very few holes).

These "holes" occur because not all combinations of fields in the dimensions apply. Sparse data typically appears in applications that display information about products and stores. In a large retail organization, not all of the company's stores will sell every product in their range, so where a store name coincides with a product that it does not sell, the value simply remains empty (or null).

Relational Databases are also sparse, but do not appear to be when viewed directly. This is because a database table only shows the records that exist, any data that doesn't exist usually does not appear in the table, so the empty values are not readily apparent.

Figure 3.4.2. shows our simple two-dimensional cube, but this time with some of the numbers removed. It is the presence of these empty cells that makes a sparse cube.



Store	Bulbs	Batteries	Fuses	All Products
Uptown	40		56	96
Midtown	52	22		74
Downtown		78	58	136
All Stores	92	100	114	306

Fig. 3.4.2. A sparse two-dimensional cube.

It is important to note that these empty cells are not zero values, they are effectively holes in the data. For example, in figure 3.4.2., the empty value at the intersection of Bulbs and Downtown shows that the Downtown store does not sell bulbs.

The ability to handle sparse data is an important part of the performance of an OLAP application. If the data cells don't exist, then they should not occupy any space in the cube, although their location in the cube would still be valid.

The concept that empty cells do not occupy any physical space is very important to modern OLAP applications because a large OLAP cube has the potential to be enormous! As an example, imagine a five-dimensional cube, where each dimension has ten thousand fields. The maximum number of data points that could be stored in such a cube is derived by multiplying together the number of fields in each dimension, in this case resulting in a cube with 100,000,000,000,000,000 cells! Although an enormous number, this value is only the potential cell count, literally the maximum number of cells that could be stored in the cube. In reality, the actual cell count might be quite small, for example a few million, resulting in a manageable cube size because the unused data points require no physical storage space.

The concept of a maximum size for a cube exists because in a cube, the fields are all fixed, so they can only reference a finite number of cells. In a database, more fields can be added in the columns as required, resulting in a gradual growth in size of the table.

As an analogy, a database table is like a balloon. It can expand and contract as data is added and deleted. A cube is more like a box, its shape is rigidly fixed and data can be added up to the maximum amount that will fit in the box.

Storing the data - ROLAP, MOLAP and HOLAP?

These three terms refer to different ways of physically storing data that is held within an OLAP cube. Each method still attempts to present data as a cube, but uses different underlying technology to achieve the results.

ROLAP

Stands for "Relational OLAP". This term describes OLAP applications that store all of the cube data, both base and high-level in relational tables. The application hides the presence of the tables by presenting the data in a cube layout. Vendors who only have a relational database engine available to them have to take this method of data storage.

The multidimensional views are generated by combining base & aggregate data tables together with complicated (often multi-pass) SQL statements, often resulting in poor reporting performance combined with the difficulty of maintaining tables of aggregate data in order to improve reporting response times.

MOLAP

Stands for "Multidimensional OLAP". This term describes OLAP applications that store all of the cube data, both base and high-level in proprietary multidimensional data files. The application copies the base data from the underlying table into a multidimensional data format (usually a binary data file) and then evaluates the consolidated values.

The multidimensional data views are automatically present in this method and performance is often very quick, particularly if the cubes are small enough to fit into RAM. More typically, the data is stored in large disk files.

The biggest drawback with this method is the duplication of base data that occurs when it is copied into the cube, requiring extra disk space and processing time.

HOLAP

Stands for "Hybrid OLAP". This term describes OLAP applications that store high-level data in proprietary multidimensional data files, but leave the underlying base data in the original data tables.

This method has the big advantage of not requiring duplication of the base data, resulting in time & disk space savings.

The cube drives the multidimensional views, so the application requires a robust link between the multidimensional data file and the relational table that stores the base data beneath it.

The information contained in this document represents the best current view of Crystal Decisions on the issues discussed as of the date of publication, but should not be interpreted to be a commitment on the part of Crystal Decisions or a guarantee as to the accuracy of any information presented.

This document is for informational purposes only. CRYSTAL DECISIONS MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, IN THIS DOCUMENT. CRYSTAL DECISIONS SHALL HAVE NO LIABILITY OR OBLIGATION ARISING OUT OF THIS DOCUMENT.

© Copyright 2001 Crystal Decisions, Inc. All rights reserved. Crystal Reports, Crystal Enterprise, and Crystal Decisions are the trademarks or registered trademarks of Crystal Decisions, Inc. All other trademarks referenced are the property of their respective owners.

Specifications and product offerings subject to change without notice.