

## Tom's Ten Data Tips – May 2008

### OLAP

OLAP, short for On-Line Analytical Processing, performs a unique function in between SQL and spreadsheet functionality. There are *four* core requirements for which neither SQL nor spreadsheets are fully adequate. These are support for:

- multiple dimensions (alas, doable with SQL)
- hierarchies (technically possible with SQL but very cumbersome)
- dimensional calculations, and
- separation of structure and representation

For standard reporting both the expected analytic questions as well as the context/domain are known beforehand. OLAP analysis requires that the context is given, this becomes manifest in the choice of dimensions that can be queried in the OLAP cube. For analytic questions where the context can not (yet) be specified beforehand, a more flexible solution is needed. The 'price' you pay for this flexibility is that composing a query requires far more expertise. As early as 1993 E.F. Codd published a whitepaper introducing OLAP. Codd realized the inherent weakness and performance issues of SQL and suggested an arrangement of data in arrays to allow fast analysis – the essence of MOLAP cubes (see also tip# 8). As Ralph Kimball has said: "A freshman in business needs a PhD in SQL."

### 1. Response Time For OLAP Should Approach 1 Second

Through many usability studies, three interface response thresholds have been discovered. When a response is returned within less than 0,1 second, the user has an impression of working in "real-time". For 0,1-1 second, the user experiences "interactivity", and keeps his train of thought with the task at hand. For a response from 1-10 seconds, the train of thought is broken, but the user stays with the task. If a response takes longer than 10 seconds, the user will switch to another task (like when you go fetch coffee after starting up your computer).

It is a pure coincidence that there happens to be a factor of 10 in between these thresholds (or at least we don't know of any psycho-physiological law to explain these exact values). However, given that OLAP lends its strength from supporting the user's analytic thought processes, it is extremely important that the interface matches his

mode of working. This absolutely requires interactivity. At present, many solutions are being delivered that don't come near one second response times. Nonetheless, these psychological laws point to the actual interface requirements.

## 2. Choose Parsimonious Display Forms

There is no universally best way to display multi-dimensional results on screen. The fact of the matter is that for most business questions there are more than two dimensions to consider, and a computer screen limits our display to a flat plane. The problem is often compounded by the fact that many users have rather small displays. What this implies is that some efficient higher dimensional view needs to be imposed on the flat plane. Visualization is one option, and "nesting" tables is your most likely alternative.

Choose your display functions in such a way that as little screen surface as possible is used up to display dimension members. Any referencing/locating information will stand in the way of the actual content. And the less space there is for the content, the more scrolling you will need to do, which makes it exponentially harder to understand what you are looking at. Nesting dimensions across rows and columns uses too much space; preferably put dimensions into screen pages (aka tiles) and also invest in 'screen real estate' (large monitors).

## 3. OLAP Designers And Users Must Interact Frequently

There is a fundamental assumption underlying the design of an OLAP system: that requirements can be specified, and the expected interaction between frequently occurring queries and the (structure of) the data can principally be known. Of course, in practice, requirements are rarely perfectly specified. And the eventual load on the OLAP server is quite hard to predict. In some instances, it can make sense to 'physically' (in technical terms this is likely to imply a change to the *logical* data model) alter the representation of the underlying data to better meet the needs of the end users.

For these reasons, it is extremely important that end users interact on a regular basis with the architects and DBA's of the system. This way they can periodically review how the current system continues to stay fit for purpose.

#### 4. Data Sparsity Leads To Exponential Storage Problems

This seemingly paradox is actually a 'unique feature' of OLAP. The reasons are *purely mathematical*, and unrelated to compression techniques, it has nothing to do with database storage (so ROLAP suffers just the same as MOLAP – see tip# 8), and also has nothing to do with poor storage of sparse data. So what *is* the problem here?

Input data is often quite sparse. For example a retail store has data on store sales by product by time. Because not every product is sold in every store, and not every product available is sold every day, this 3-way structure contains a lot of empty cells. As dimensions get added, like customer segment, organizational hierarchy, scenario (actual sales/plan/variance), etc. every new dimension and (ragged!) hierarchy leads to exponentially more empty cells. Pre-calculating all of these becomes unwieldy after about 5-7 dimensions in the cube. The consequence is that in many cases, in particular with a lot of (unclustered) sparsity, and in particular with many dimensions, this effect calls for a sensible pre-loading/pre-calculation strategy (see also tip# 10). Careful administration is required to avoid performance degradation (see also tip# 1). This problem is often ill understood, and requires thorough familiarity with the underlying data.

#### 5. Building OLAP Directly On Top Of Operational Systems Is (Usually) Not A Good Idea

In order to avoid having to warehouse data, it is sometimes tempting to build an OLAP solution directly on top of operational systems. This has become more feasible now that many OLAP solutions offer remarkably powerful ETL functionality. The reasoning is that since the end user "only" needs OLAP to interface, (considerable) cost saving are possible by this architecture. There is no such thing as a free lunch, however...

The reasons why this architecture works so poorly are:

- The operational systems were never designed for integrated processing, yet this is assumed "a given" for data prior to entering the OLAP environment
- The operational environment (usually) contains limited history which is dearly needed for proper OLAP
- Each OLAP environment (customized per department) needs its own purpose ETL interface. What looked cheap for the first OLAP mart, becomes prohibitively expensive when adding more OLAP marts later.

- Multiple OLAP environments will *all* put an ETL drag on the performance of the operational system. Again, looked cheap initially, proves expensive when developing multiple OLAP marts.

## 6. Performing OLAP Directly On The DWH Doesn't Work, Either

The question may then arise why the OLAP environment "needs" it's own storage and ETL – why not run directly on the Data Warehouse?

The answer is threefold:

1. the OLAP environment should be optimized for a given end user group. That means they need not "see" all the data present in the corporate warehouse. Making an intelligent (sub) selection is needed to facilitate searching, and to provide data in a format that is as intuitive as possible for the end user (*not* an easy task).
2. There is relatively infrequent direct access to detailed data, and by comparison a lot of direct access to the aggregated data via the OLAP interface. This calls for disparate maintenance requirements (and different indexing needs – see bullet #3)
3. The DWH contains a very large volume of detailed data, and therefore the number of indexes will be small (say, 2-5). The OLAP environment contains relatively little data, and for speed of access therefore allows dozens of indexes, which would be impractical to store and maintain in the DWH.

## 7. Ragged Versus Leveled Hierarchies Have Their Own Terminology

Hierarchies are the cornerstone of OLAP dimensions. OLAP tools need to support both leveled and ragged forms of hierarchy. An example of a leveled hierarchy is time: days, months and years. Space/distance might be another one. A typical example of a ragged hierarchy is product: different product types have less or more elaborate subdivisions. Other examples might be organizational structure, or territory. Both leveled and ragged hierarchies can be used with nominal, ordered, or metric values although leveled hierarchies are more commonly associated with metric values, and ragged hierarchies with nominal values.

We often use different terminology to describe each. For a leveled hierarchy we talk about resolution, level, or granularity. A ragged hierarchy is typically described with parent/child, ancestor/descendent. It is important to get this distinction conceptually 'right', because in

the real world we often encounter mixed hierarchies: they appear leveled near the top, and only become ragged closer to the bottom.

## 8. MOLAP Versus ROLAP (Should) Matters Only For The Architect

Initially, OLAP really equated to MOLAP (Multi dimensional OLAP), the 'other flavors' originated later. And for the end user the difference really is irrelevant, or... at least it *should* be. However, the speed of data manipulation and/or retrieval can differ widely! How do MOLAP and ROLAP differ?

MOLAP achieves fast performance by pre-calculating all possible combinations of values in every dimension. So when the cube is queried, a pre-populated file is accessed and performance becomes an I/O matter. ROLAP stands for Relational OLAP, where querying is performed on a relational database. As a result, performance depends on the database and therefore SQL (MOLAP tools use MDX as query language). Typically, the underlying database is equipped with more powerful hardware. By populating the database with pre-calculated values (effectively aggregates) as much as possible, decent (or acceptable) performance can be realized.

Advantages of MOLAP are typically faster performance. ROLAP allows all data to remain in one environment which is advantageous.

## 9. What HOLAP Means Depends On What OLAP Means

After the distinction between ROLAP and MOLAP had settled in the marketplace, a 'new' form appeared: HOLAP for Hybrid OLAP. What this "Hybrid" entails really depends on the provider, whether it is a ROLAP or a MOLAP solution.

For MOLAP providers, HOLAP consists of (still) storing aggregates in the cube, and occasionally accessing detail data in the underlying relational database. So all but the finest granularity may be stored in the cube, and if you want to go one level deeper, the MOLAP cube "reaches" out to the (SQL!) database.

For ROLAP providers, information from the SQL database is aggregated and positioned "in front of" the database in some OLAP application server. What this functionally comes down to is storing query results in a cache.

## 10. To Pre-Calculate Or Not To Pre-Calculate

Despite the fact that MOLAP is typically the fastest way to give access to data, there are still some problems associated with “full pre-calculation”. When there are many dimensions, ragged hierarchies, and in particular due to sparsity (see tip# 4), building up the cube becomes an enormous processing job. But even if the calculation window is sufficient, storage can become an issue as the cube will be (many) orders of magnitude larger than the underlying data.

The size of a fully calculated cube can cause performance to degrade. What happens is that at some point, I/O time of an exponentially larger cube makes access slower than the actual calculation at run-time would require. In such cases, it is better *not* to fully calculate the entire OLAP cube. When you decide which cells to pre-calculate and which not, consider:

- Are fields slow to calculate?
- Are cells frequently viewed?
- Are cells frequently the basis (input) for other calculations?

In such cases it makes much more sense to pre-calculate.