

## The OLAP Gap

*By Boris Gendelev  
Principal, Wheaton Group*

***Original version of an article that appeared in the April 1998 issue of  
“Database Programming and Design”***

*[Note: Despite dramatic increases in raw computing power and a proliferation of end-user software tools since the publication of this article, virtually all of the content remains highly relevant. After the study of the (then) state-of-the-art that resulted in this article, the author went on to develop a breakthrough interface based on natural language feedback.]*

So much is in the press about creating a data warehouse decision-support environment, you might get the impression that all the technological issues have been solved, the tools and techniques are in place, and now this goal is just a matter of implementation. But a closer look reveals some serious usability gaps.

Consider the example of a marketing analyst trying to design and improve relationship marketing programs. This analyst is lucky enough to have a data warehouse containing half a decade of customer transaction history – not only what each customer bought and when, but details such as their merchandise returns and address changes, as well as a complete record of all the offers each customer received.

On the technology side, the marketing analyst is seemingly in great shape. She has access to a SQL server running on a solid box with parallel-processing capabilities and a low price/performance ratio, a SQL builder interface, a report writer, a ROLAP or MOLAP tool, and the latest data mining tools fresh out of the package.

Some would expect that a few mouse clicks would reveal in the data a slew of profitable insights. They'd be wrong. To understand how and why the reality is different, let's take a closer look at some of the analyst's concrete tasks.

Let's say the marketing analyst has an intuition that a specialty catalog (or retail store chain) is the ticket to business growth. Through intuition or research, she has some idea what specialties would be viable. Naturally, she wants to identify customer characteristics that are predictive of future buying in that category of merchandise. In marketing parlance, she wants to "segment the customer base."

She launches her ROLAP or MOLAP tool, selects some interesting customer dimensions, picks appropriate metrics, produces a report, perhaps does some drilling down, and eventually zeros in on a set of relevant dimensions and a combination(s) of dimension values that define the target audience.

Too many characteristics to select from? Too much detail to see a pattern? No problem. She can start up the data mining tool, throw every customer attribute into the pool, and let it crunch. The only thing I haven't mentioned is the source of customer dimensions and metrics (attributes). Where did they come from? Sounds like a silly question. Weren't they built when the real or virtual multidimensional hypercube was set up as the marketing data mart?

Perhaps the question isn't so silly. For example, a typical desired attribute might be: "How long ago did a customer buy X, Y, or Z?" But the analyst is very lucky if she had a premonition of what derived data will be of interest months after the data mart is built. Even if she did think of that particular question, what are the chances that she thought of making that attribute not just "life-to-date," but also "by start of each marketing campaign," as required for predictive analysis? What are the chances that all, or even most, of the relevant attributes will be ready for her? What about a variation such as "How long ago did a customer buy X, Y, or Z without returning the merchandise?" The chances are close to nil that relevant derived attributes are there and ready to use.

In the real world, business users prefer not to beg the IS team for help. Technical folks often attach little meaning to data content, have a limited grasp of marketing issues, and speak a "different language" (not to mention their backlog). Instead, our analyst turns to a SQL-based query tool that should allow her to create her own derived tables/views and then go on happily slicing, dicing, and data mining. Case closed, right? Wrong.

### **Promises and Disappointments**

If our analyst wants the simpler life-to-date version of the query, she might figure that she can do a relational projection on a join of Items and Orders. In SQL, it might look like this:

```
SELECT Order. CustID, Max(Order. Date)
FROM Item, Order
WHERE Item. OrderID = Order. OrderID AND Item. Sku IN (X,Y,Z)
GROUP BY Order. CustID
```

We cannot expect the user to be steeped in the relational theory or SQL semantics. Yet to write the query she has to combine joins, projections and selections in just the right way. She can probably manage that in a simple query. But the complexity of additional conditions "by start of each campaign" and "without returning the merchandise" is likely be too much to handle.

In the past, the user had to learn not to ask for a Customer attribute directly. (Nowhere does the previous query mention a Customer entity.) Fortunately, SQL is becoming more powerful, more expressive, and more usable. One can now write:

```
SELECT CustID,
(SELECT Max(Date)
FROM Order
Where EXISTS
```

```
(SELECT * FROM Item
WHERE Item. OrderID = Order. OrderID
AND Item. Sku IN (X,Y,Z) )
And Customer. CustID = Order. CustID)
FROM Customer
```

This query is not only probably more logical, but also more powerful and manageable because in one query many unrelated customer attributes can be defined:

```
SELECT CustID,
(SELECT ... ),
(SELECT ... ),
...
FROM Customer
```

However, this kind of query is still far less than ideal for several reasons. First, such queries are usually more verbose because the join condition is repeated in each subquery, which makes the query more error prone. It would be nice if the user could write FROM Customer's Order, or better yet, if anything that is derived from Orders could be put into a single subquery such as SELECT . . . , . . . , . . . FROM Customer's Orders. Second, the user might think that Customer's Orders is implied without the join condition, because it seems natural that a computer would "know" such an obvious thing. And third, relative to the less intuitive "older" version, the performance of such a query might be horrendous. The engine has to be smart enough to essentially turn the newer version into the older one or pay the penalty of executing each subquery separately, redoing the same join over and over. Optimization is possible, but such an optimization problem would not even exist if SQL were more expressive.

With computing costs decreasing dramatically, why should a user worry about query performance at all? Because at the same time, the amount of data users want to access, the frequency with which it is accessed, and the desired complexity of data manipulation are all increasing at the same or a greater rate. Sacrificing some efficiency to gain some usability is acceptable only to a point. When the users either have to pay too much or wait too long for queries, then the ad hoc, iterative flavor goes right out the window. Physical data organization has a lot to do with performance as well, and relational database products designed for OLTP do not shine in the data-warehousing world.

The truth is, even with improvements, SQL would not be an appropriate tool for a business user. It's not SQL's fault – after all, it simply implements relational calculus, Boolean logic, and aggregation functions. The problem is that these are abstract, formal problem-solving techniques involving concepts invented specifically for the database domain. What made anyone think that a typical user could put the full power of relational calculus to productive use?

## The GUI Mess

With most GUI tools, instead of typing you can pick tables (Customer, Order, Item) and column names (Age, Gender, Order Date, Payment Type, Product Type, Price). Some products also let you select relationships (Bought, Contains) and either parts of SQL syntax (WHERE, GROUP BY, UNIQUE, JOIN, UNION, COUNT, SUM, AVG, >, <, AND, OR) or relational, numerical, and logical operations (selection, projection, join, union, set difference, count, sum, average, greater than, less than, logical and, and logical or). The selection may occur from a menu or by clicking on icons or by placement in the right spot on the QBE grid. That saves typing, but does not make it any easier if the user doesn't know what syntax or concepts to apply to certain components of the database. Writing SQL is a form of programming. In that respect, most GUI SQL builders are to SQL what visual development environments are to C/C++ or Basic: They help you enter syntactically correct code faster, but do little to help you make it semantically correct.

Modern programming environments have debuggers that show you what happens as you execute code step by step. But the same cannot be said of most GUI SQL builders. A possible exception is Geppetto's Workshop's Ant Colony, in which users perform step-by-step transformations of live data views.

Most GUI SQL builders default to natural joins among tables. This approach prevents the mistake of assuming a join condition without specifying it in the SQL statement. Some, however, do not allow the user to do anything but natural joins or joins predefined by a database administrator, which greatly restricts the range of possible queries.

Several GUI query tools allow for anticipated common calculation, joins, projections, and selections – "views," in SQL terminology – to be predefined (typically by a database administrator). For example, Order Date and Payment Type might already appear to the user as not only Order but also Item attributes, avoiding the need to create a view that joins Item and Order. But as previously discussed, only a small fraction of what will be needed can be anticipated and predefined.

Overall, the approach to making query formulation accessible to non-technical users is limited to pre-computation and restriction of functionality. Many query tools win awards for ease of use but do not support complex subqueries or multi-pass processing – leaving more complex, and often more interesting, queries to be written directly in SQL. The products that do not restrict functionality do not necessarily make the user write SQL, but they do require the creation of a sequence of relational operations and inputs that lead to the desired output. These products are not for business users.

Another defect of the end-user (as opposed to the administrator) component of mainstream query tools is that they are SQL builders as opposed to logical schema extenders. Yet reusing existing queries (or parts of them) is absolutely essential for iterative data manipulation. For example, if the user has already defined a number of Customer attributes, it is extremely helpful when these attributes are available under the Customer entity list. But that usually doesn't happen because end-user queries and the views implied in them are created at the row and column level, not at a semantic level of entities with attributes.

Why is it so hard to preserve full functionality, support iteration, and yet hide conceptual complexity from a user? What happened to the promise of visual metaphors, the "graphical" part of GUI?

A graphical metaphor for a technical concept can be powerful indeed. For example, a folder with a stack of order folders attached to it represents a logical data model for Customer through an easy-to-grasp visual metaphor. Perhaps data manipulation can be represented by graphical metaphors as well. With an icon suggestive of "total" on the desktop, the user would quickly learn that dragging the icon to the relationship line between Promotion and Order totals order data for each promotion.

However, when applied to arbitrary database queries, the visual metaphor approach breaks down for several fundamental reasons. Many data manipulation concepts are just not naturally graphical. For example, what visual metaphor will portray "Amount of the First Order?" Or relationship between promotion and purchase based on a condition that the first line item of the order is from the catalog sent in that promotion? Or on the percentage of line items related to that promotion? Even if an appropriate visual metaphor were found for each data manipulation concept, these metaphors – being from many different domains – would likely create an unintelligible mix.

Even straightforward metaphors such as an ER diagram become unintelligible if too many elements are in the picture. Their usefulness deteriorates even more rapidly when – as it would be the case with data manipulation queries – query elements are nested. The cube metaphor and the star or snowflake schema that support the cube were devised to make Entity-Relationship models more accessible, but only for one specific class of queries – slicing and dicing along predefined dimensions – and not for atomic data. So the use of such metaphors only begs the question of how an ad hoc user would turn a tangled ER diagram into a nice, neat star diagram.

## **The Natural Alternative**

As an alternative to metaphor or formal query language-based interfaces, natural language has a lot of appeal. After all, when users think up a query and communicate it to technical people, they do so in "natural" language. It offers the flexibility to manage complexity much more gracefully than SQL. If you are a doubter, write down SQL statements required to express the following queries and observe how convoluted the last two become:

- For each Customer, what was the biggest order amount?
- What was customer's payment type from the order with the biggest amount?
- What was customer's payment type with the biggest total amount in a single month?

And natural-language queries would be self-documenting to boot!

Unfortunately, today's natural-language understanding technology is far from being able to offer a comprehensive, reliable solution. In my product tests, many queries were not understood at all – the programs "gave up" and were unable to resolve the impasses (see sidebar, "The Proof is in the Query"). Worse, many queries were misunderstood, which isn't too surprising considering that even intelligent database programmers often misunderstand user requests. The difference is, a good

programmer will engage the user in a dialog of clarification questions, restatements, and examples, but natural-language query vendors make only rudimentary attempts to incorporate such behavior in their software.

The bottom line is that the technology for allowing a user to enter a natural-language query of arbitrary complexity does not yet exist. Nor is it evident that the problems associated with pure natural-language processing will be overcome anytime soon. In fact, the number of research papers on natural-language processing cataloged on the Web peaked at the end of the last decade and has been shrinking ever since – and probably not because the problems were solved but because as many AI problems proved too hard to solve.

### **Wish List**

We technologists are failing a large class of data warehouse users, perhaps even giving them false promises. A lot more work needs to be done by database, access tool, and interface makers before these issues are addressed satisfactorily.

The SQL has to become more expressive so that neither SQL servers nor generators have to optimize queries that, if expressed more naturally, would be inherently optimized. Physical storage and retrieval have to rely more on a combination of inverted lists and nested tables in order to match, and thus optimize, performance of the two major processing patterns: selective slicing and dicing and complex multilevel mass aggregations. Metadata repositories have to become more dynamic and incorporate derived attributes and entities (real or virtual) – not just those set up by an administrator but also those implied in end-user queries.

1. Ultimately, an interface targeted to non-techies is the most crucial piece of the puzzle. In my view, natural language is the only promising direction. But the only way that natural language can be made practical in the near future is to let it be not quite as "natural" as the language you and I speak, yet understandable without training and to make query entry not quite as freestyle as it would be in a word processor. At the same time, with expert-system techniques, it should be possible for natural-language tools to imitate dialog between a user and a database programmer if a statement becomes ambiguous, asking "do you mean X, Y or Z?"
2. Restate the query in a different way to confirm understanding.
3. When the meaning becomes unclear, ask "Given that I think you are trying to do X, what does Y mean?" and show examples of how Y can be defined. Allow for the possibility that X is not the goal and offer examples of alternative interpretations.
4. Allow alternative interpretations, restatements, examples, and prior queries to be gradually modified, applying the same behavior to building a component of a query as to the entire query; that is, points 1 through 3.

5. Demonstrate each subquery and calculation implied in a query on a sample of real data – saying in effect that "this is how my interpretation of your query will be computed."

These goals are eminently reachable and I'm confident we'll see some serious progress soon. I also believe that real breakthroughs are more likely to come from power users rather than traditional software vendors. The original spreadsheet, statistical analysis, and multidimensional packages were all brainchildren of power users frustrated with the lack of easy-to-use yet flexible software.

When the unrestricted database query is made accessible to end users, the next challenge will be to move from the tool-centric world to the application-centric one. Ultimately, the user should not be switching between – or even be aware of – database query versus slicing and dicing, versus reporting, versus statistical analysis, versus data mining, but she should simply be able to run applications such as "Customer Segmentation" or "Market Basket" that invoke the tools behind the scenes.

*Boris Gendelev is a Principal at Wheaton Group, and can be reached at 847-205-0916 or [boris.gendelev@wheatongroup.com](mailto:boris.gendelev@wheatongroup.com). The firm specializes in direct marketing consulting and data mining, data quality assessment and assurance, and the delivery of cost-effective data warehouses and marts.*