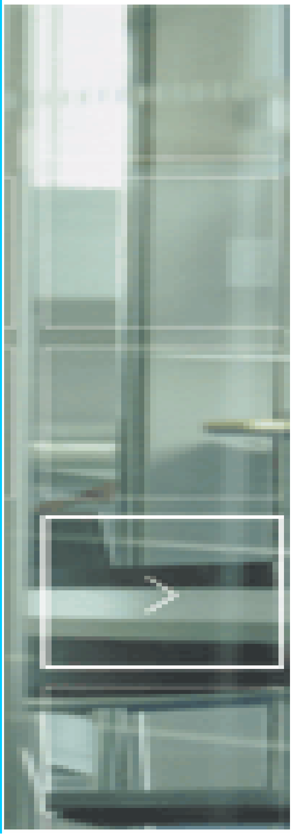
Criteria for Query Tools

Author: BALWANT RAI
Organization: Evaltech, Inc.
Evaltech Research Group,
Data Warehousing Practice.
Date: 02/18/04
Email: erg@evaltech.com



Abstract:
In this article we have gone through the basic features of the query tools. This list of capabilities is by no means complete. The new high-end query tool should have these capabilities .Cross-browsing is needed to visit some or all of the dimension tables in dimension modeling in order to set the constraints. Aggregate navigation is the ability to automatically choose pre-stored summaries, or aggregates, in the course of processing a user's SQL requests. Partitioning of a single complex sql statement into several small sql statements is called multipass SQL. Semi-additive are the numeric measures in common fact table that are not completely additive.

Query tool features:

Cross-Browsing of Dimension Attributes.

Query tool used against a data warehouse involves:

- Visit some or all of the dimension tables in dimension modeling in order to set the constraints
- We launch the join between multiple tables involving several of the dimension tables together with the large central fact table

Cross-browsing is needed to visit some or all of the dimension tables in dimension modeling in order to set the constraints. A real dimension table, such as customers, takes the form of a large dimension table with lots of attributes. It is absolutely mandatory for a query tool to present, in real time, a list of the valid values in a dimension attribute and let the user choose one or more of the values to set a constraint. All sophisticated query tools has the basic browsing capability .On the other hand, Cross-browsing refers to the capability of a query tool to present the valid values, subject to a constraint elsewhere on that dimension table. This ability to cross-browse distinguishes showroom demos from query tools used for serious querying in large data warehouse environments.

Open Aggregate Navigation.

One of the most exciting new developments in data warehousing is the emergence of aggregate navigation, a capability that changes the architecture of all end-user applications. Aggregate navigation is a technique that enables DBAs to optimize performance by storing aggregate values in the database, without requiring end users to know about the existence of those aggregates. Aggregate navigation is the ability to automatically choose pre-stored summaries, or aggregates, in the course of processing a user's SQL requests. Aggregate navigation must be performed silently and anonymously, without the end user or the application developer being aware that the aggregations even exist. Open aggregate navigation occurs when the aggregate navigation facility is a separate module that is available for all query tool clients simultaneously. Nothing is worse, or more shortsighted, than an aggregate navigation facility embedded in a proprietary query tool and unavailable to other end-user clients. Unless the current proprietary aggregate navigators embedded in query tools are made into openly accessible modules, big DBMS vendors may take this business away from query tool providers.

Our data warehouse foundation layers consist of hundreds of millions of records in large fact tables, surrounded by a small set of six to eight dimension tables that serve as entry points into the fact tables. If database engines were infinitely fast, this design would be satisfactory, and we could concentrate on other issues such as application programming and front-end tools. However, we all know that we need to calculate some values in advance, such as totals, and store them in the database in order to improve performance. If we use this baseline measure in our business often, we certainly do not want to process an entire year's worth of transactions every time we ask for this denominator. Every good DBA will build a set of aggregates to accelerate performance.

The building of aggregates is a huge double-edged sword in the big data warehouse environment. On the positive side, aggregates have a stunning effect on performance. The highest-level aggregates, such as yearly national sales totals, frequently offer a 1000-fold improvement in runtime, compared to processing the daily sales or the daily transactions. Other than ensuring that the database optimizer is working correctly, the addition of aggregates to a data warehouse is the most effective tool that the DBA can bring to bear on performance.

Aggregates have two big negative impacts.

- They obviously take up lot of space. If we build aggregates in three primary dimensions, such as product (product brand level, category level, and department level), market (country level, district level and region level), and time (week level, month level, and year level), these aggregates can multiply geometrically to overwhelm the database.
- The second problem, and the one directly addressed by the aggregate navigators, is that an end user's query tool must specifically call for an aggregate in the SQL or it won't be used. If the end users' tools have to be hard coded with knowledge of the aggregates, then the DBA doesn't have the flexibility to change the aggregate profile in the data warehouse. Aggregates can't be added or subtracted because all the end-user applications would have to be recoded.

The aggregate navigator addresses the problem of hard coded end-user applications by sitting between the end-user application and the DBMS, and intercepting the end user's SQL. With an aggregate navigator, the end-user application now speaks "base-level" SQL and never attempts to call for an aggregate directly. Using metadata describing the data warehouse's portfolio of aggregates, the aggregate navigator transforms the base-level SQL into "aggregate-aware" SQL. The end user and the application designer can now proceed to build and use applications, blissfully unaware of which aggregates are available. To tune system performance, the DBA can adjust the warehouse's portfolio of aggregates on a daily or weekly basis. This unhooks the dependence of end-user applications on the back-room physical aggregates.

There are two main ways to store aggregates: in the original fact and dimension tables as extra records, or in separate fact and dimension tables as extra records. In both cases, you store exactly the same number of records. The separate fact and dimension table approach is emerging as the recommended technique, even though it proliferates quite a few tables. The problem with storing the aggregates in the original fact and dimension tables is that, in order to distinguish the aggregate records from the base-level records, you must introduce a special level field in each of the affected dimensions. This makes for rather complex table administration because now the dimension tables have several null entries for individual attributes that no longer make sense at the aggregated levels. The recommended technique is to take each kind of aggregate and place it in a separate table.

All of the aggregate navigators mentioned above work only on dimensional data warehouses, which consist of large central fact tables filled with additive numeric facts, surrounded by smaller dimension tables filled with text-like values used for constraints and report breaks. The vendors also offer interesting statistics-collection capabilities for monitoring the user community's SQL and advising the DBA on which new aggregates to build. At present, none of these vendors offer integrated tools for actually building the aggregates. They are content to let the extract tool providers such as Prism Solutions Inc. (Sunnyvale, Calif.), Evolutionary Technologies Inc. (Austin, Texas), Carleton Corp. (Burlington, Mass.), and Vality Technology Inc. (Boston) manage the building of the aggregates in a separate phase.

In thinking about the approach these four vendors have taken, I believe that the future lies in network server-based aggregate navigators such as Intelligent Warehouse and MetaCube. The DBAs of the world will appreciate a solution that provides the aggregate navigation benefit transparently to all end-user tools. I suspect that the proprietary tool providers will find a way to unhitch their aggregate navigator modules from their tools and provide the capability as a networked, ODBC-compliant resource. Clearly, these vendors understand aggregate navigation.

It will also be interesting to see how soon the big DBMS players notice this new development and add aggregate navigation to their bundles.

Multipass SQL.

Partitioning of a single complex sql statement into several small sql statements is called multipass SQL The query tool then automatically combines the results of the separate queries in an intelligent way. Multipass sql are used when we need to calculate comparisons or to correctly calculate non-additive measures in report break rows. The query tool must break the report down into a number of simple queries that are processed separately by the DBMS.. Multipass SQL also allows drilling across several conformed data marts in different databases, in which the processing of a single galactic SQL statement would otherwise be impossible. Multipass SQL gives the aggregate navigator a chance to speed up the report. Each atomic SQL request is simple and easily analyzed by the aggregate navigator.

Semi-Additive Summations.

Semi-additive are the numeric measures in common fact table that are not completely additive. Anything that is a measure of intensity is generally not additive, especially across the time dimension. For example, inventory levels and account balances are not additive across time. These facts are called semi-additive. Creating a useful summary at the end of the month by averaging the bank balance across time is the best example of semi-additive fact.. Unfortunately, we cannot use the basic SQL AVG function to calculate this kind of average across time. Averages can be calculated across all of the dimensions except time dimension. If you fetch four accounts and two time periods from the DBMS, avg will divide the total account balance by 8 rather than doing what you want, which is to divide by two. It isn't difficult to divide by two, but it is a distraction for the end user or the application developer, who must stop and store the number two in the application explicitly. What is needed is a generalization of the sum operator to become avg time sum. This function automatically performs a sum, but also automatically divides by the cardinality of the time constraint in the surrounding query. This feature makes all applications involving inventory levels, account balances, and other measures of intensity significantly simpler.

Show Me What Is Important. The growth in the power and capacities of data warehouses has its advantage and its drawbacks.

Advantages: We are able to store mind-boggling amounts of low-level data in your databases: Fact tables with a billion records are fairly commonplace.

Drawbacks: We are in much more danger of getting back too much data to comprehend usefully.

Query tools must help you automatically sift through the data to show you only what is important. At the low end, you simply need to show data rows in your reports that meet certain threshold criteria. This process involves more than just adding a HAVING clause to the SQL. In an aggregate, multipass SQL environment, the criteria for including or excluding a record from the user's view may not be known until after the DBMS has long since passed back all the results. Thus this filtering function is rightfully the responsibility of the query tool. The high end of showing what is important is the exciting new area of data mining, which merits a whole separate article. Increasingly, query tools need to embed data mining capabilities into their user interfaces and underlying architectures.