"One Size Fits All" Database Architectures Do Not Work For DSS

By Clark D. French

The current state of internal database technology today is a "**One Size Fits All**" approach. Whether the database is being used to solve an OLTP problem or a DSS problem, all of the leading database manufacturers believe they have the right solution. This would be correct if the needs of an OLTP application resembled those of a DSS application. But in fact, their needs are very different. The physical database internal architecture designed to optimize for OLTP is very different from one designed to optimize for DSS. Not only would these two architectures be different, they would also be opposing. An architecture that optimized for DSS will degrade OLTP, and vice versa. We can say that OLTP and DSS have "**Opposing Laws of Database Physics**"

There is also a relationship between the volume of data and the ability of a given optimized approach to work well. When the amount of data a database has is small, the difference in performance between an OLTP-based database architecture and a DS- based architecture is not very important. As the volume of data increases, the need for and the ability for one approach to stand out above another becomes more apparent

The current database manufactures have designed database architectures optimized for their main business, OLTP. They have attempted to use their existing OLTP architectures to address the DSS side of the business. The performance of even simple user questions, such as "**How many of my female customers in Mass. bought product A?**", can take hours to run. This has created a problem: "**Too Much Data, Not Enough Information**".

I believe that over the next several years both the traditional database manufactures and others will develop internal database architectures optimized for DSS. New and immature examples exist today with products like Interactive Query Accelerator from Sybase, Redbrick, and OMNIDEX from DISC. The future may bring separate products or substantial add-ons to the traditional database providers' existing product suites.

SIGMOD ' 95,San Jose , CA USA

The following table illustrates some of the differences between OLTP and DSS from an application usage perspective.

OLTP vs. DSS.

Pre-determined Queries	AdHoc Queries
Simple Queries	Complex Queries
Small Found Sets	Large Found Sets
Short Transactions	Long Transactions
Update/Select	Select (Read Only)
Real Time Update	Batch Update
Detail Row Retrieval	Aggregation and Group By
High Selectivity Queries	Low Selectivity Queries

The above comparison, although obvious, helps highlight the fact that most of the major commercial databases today have designed their products for OLTP.

AdHoc Queries

One of the most important problem areas a database must address to be successful at DSS is adhoc query performance. I submit that a difficult problem is made worse by the traditional database providers who attempt to solve it using OLTP based query optimizers , indexes and physical architectures Adhoc queries become easier when a new approach is taken. There is a better way than just using parallel table scans.

One approach is instead of trying to predict the queries and apply poorly suited indexing and query processing techniques, to attempt to understand the data and its usage to see if query performance can be improved. I have found that there is a relationship between certain attributes of the data and how the data is typically used within the SQL language for DSS These data attributes are typically static over the life of the data and hence predictable, where the queries are not Some of these data attributes are cardinality, distribution, and value range.

Data Attributes and SQL

There is a relationship between a column's cardinality and its usage with the SQL language. A column that has a low cardinality, say under 1000 unique values, tends to be used in SQL WHERE clauses and GROUP BY clauses most frequently in DSS. When used in the SQL WHERE clause, a low cardinality column is usually used in EQ and NE predicates. It is rare to see a low cardinality column used in range queries and aggregates in DSS.

A column that has an extremely high cardinality, say over 100,000 unique values, tends to be used in SQL WHERE clauses and in aggregates in the SQL PROJECTION most frequently in DSS. When used in the SQL WHERE clause, a high Cardinality column is usually a range predicate The following table highlights this relationship:

Cardinality (K)	SQL Usage
K < 1000	EQ, NE Predicates and GROUP BY
1000 < K < 100000	Both
K > 100000	Range and Aggregation.

As stated above, data cardinality, distribution and value range tend to be relatively static over time. A column like STATE may have a cardinality of 50; it may grow to 100 but will probably not grow to 100000. With a column like STATE, CA has many more records in a typical database than RI and that tends to stay relatively static over time.

Now that we have described the relationship between data attributes and SQL common usage; *is it possible to design an index that takes advantage of known attributes of the data to increase query time performance?*

Bitmap Indexes

One example is the bitmap indexing used by products such as OMNIDX, Model 204, Foxpro, and IQ Accelerator from Sybase. Developed in the 1960's, this indexing approach gains performance by only handling low cardinality data. It represents each distinct value as arrays of bits where a 1 or 0 in each relative position in the array represents True or False for that value for the corresponding relative record within the database relational table. This approach is sometimes referred to as inverted-list.

The use of bitmaps also helps in another problem area for DSS. The traditional OLTP database optimizer approach of building up record lists, although great for OLTP small found sets, is cumbersome at best for large multimillion record found sets typically found in DSS. Bitmaps are ideal for representing large found sets. When multiple predicates are used in a query, the arrays of bits for each value can be easily combined using boolean operations. Thus, for a 1 million record table called *customer*, asking the question; "Please tell me the rows where STATE=MA and PRODUCT=A", would result in reading two 128k byte arrays from disk and then ANDing them together. This is a far more efficient approach than scanning 80 percent of two large traditional OLTP B-trees or doing a table scan. It should also be noted that this indexing approach is very inefficient at OLTP type queries If the found sets were traditionally small, reading in 128k would be a slower approach to solving the query. This indicates that OLTP and DSS are often opposing problems. Optimizing for one often slows down the other propartionately

This bitmap, value-based indexing scheme also addresses the typical DSS usage. It is very good at determining EQ, NE and GROUP BY queries. It is compact when used correctly, quick to load, and can be incrementally loaded without creating working set problems

Other Techniques

There are other techniques for gaining adhoc query performance in DSS by understanding attributes of the data and its usage within SQL Some of these are currently under patient pending and being used by Sybase's IQ Accelerator product. I believe that, we concentrate on DSS-only database architectures, new techniques will develop and be proven in the industry

The fundamental question is whether one database approach can be everything to everyone The answer will come, as it should, from the market, which will decide if it will accept database products that are focused exclusively on DSS but offer better performance.