SOLVING THE LOAD/QUERY CONUNDRUM IN DIMENSIONAL DESIGN

HIRED BRAINS, INC.

By Neil Raden
March 2005

ABOUT THE AUTHOR

Neil Raden is the founder of Hired Brains, Inc., http://www.hiredbrains.com. Hired Brains provides consulting, systems integration and implementation services in Business Intelligence, Data Warehousing, Performance Management and Information Integration for clients worldwide. Hired Brains Research provides consulting, market research and advisory services to the Business Intelligence, Data Warehousing and Information Integration industry. Based in Santa Barbara, CA, Raden is an active consultant and widely published author and speaker. He welcomes your comments at nraden@hiredbrains.com.

PREMISE

When it comes to the physical design of relational databases for data warehousing, it's always been a case of "pay me now or pay me later." Query performance enhancement tends to lengthen the time it takes to update the database (latency) and optimizations for super-fast loading tend to yield sluggish performance and an unsatisfying user experience. In today's environment of ever-increasing needs for timely data, not just loaded but put into play, we need a "pay me now or pay now approach."

THANK YOU DR. MOORE

Thanks to innovations resulting from the declining cost of hardware, solutions are actually available. Moore's Law postulated a halving in price and doubling in density of transistors on integrated circuits every two years almost forty years ago. Not only has that prediction proven to be true, similar price/performance gains have occurred throughout the technology industry. The amount of memory, storage, bandwidth and CPU power available today, would have been unfathomable to us just a few years ago.

Unfortunately, much of the existing knowledgebase about system design is still based on the concept of managing from scarcity. What we accept today as good design technique is, if examined closely, often the result of tradeoffs made long ago due the extreme scarcity and very high cost of computing resources. The design of relational databases for data warehouses is a good example. The two prevalent schema designs are normalized and dimensional. Normalized designs are well suited to high-speed, high concurrency transaction processing, but are very poor performers at satisfying queries. Dimensional schema are excellent performers at satisfying very complex queries, but are typically sluggish to update. Each was developed in an era when the tradeoff was necessary. Is there a way, given the state of the computing environment, to have both excellent query *and* load performance in a single design?

TWO APPROACHES

There are two basic approaches to resolving the load/query conundrum:

1. Optimize the design for the fastest possible ingest of data, as close to real-time as possible, and solve the sluggish query problem with massive amounts of parallel processing and RAM.
2. Optimize the design for fastest possible query response and solve the ingest speed problem with some clever new ways of maintaining indexes in real-time, taking advantage of the plentiful resources at our disposal

Option 1 is typically implemented with a normalized schema design, an approach that is designed to allow the fastest possible insert of records because there is no redundancy of data and there are no indexes, or very few. It has actually been around for quite a while, but it suffers from a few drawbacks. First, it is very expensive, relatively speaking, because it requires a lot of horsepower to resolve analytical queries. Rather than have an optimizer figure out the most clever approach to getting the answer, the system resources are spun up to 100% and tables are scanned from top to bottom. Second, it is too easy to defeat the design by presenting queries that cause the system to have to shift data around. This occurs when the keys used to partition the data play only a minor role in a query, forcing too much of the burden on one or a few processors, a condition called skew. Also, by employing a schema with very few or even no indexes, simple browsing operations, like pick lists, can be inordinately slow.

Before getting to Option 2, we need to discuss database design for data warehouses in general. This article is concerned strictly with relational databases that are used for satisfying analytical queries. Many data warehouses are composed of a series of databases, most of which serve an intermediate purpose or, in this writer's opinion, no purpose at all. A staging area, for example, has unique performance requirements, but it is never (or at least should never) be queried by end-users or applications. The so-called Enterprise Data Warehouse is often a massive structure, designed in Third Normal Form, but is never queried directly. Instead, downstream structures are created that host the actually query processing. Routine, programmed and ad hoc queries are presented directly to relational databases that may be called data warehouses or data marts, but the name is irrelevant. Whatever they are called, they have the same requirements for fast update and fast query response.

DIMENSIONAL SCHEMA

With that in mind, Option 2 is a far superior choice if, in fact, it can be implemented. The most common approach for designing the schema is some variant of what is known as the Star Schema. The point of a Star Schema is to minimize the number of joins needed to satisfy a query by reducing the number of tables in the design, designing the keys to line up with the "dimensions" of the business models, those axes on which the numeric measures rotate. They typically have small tables for the dimensions (though not always) and very large tables for the "facts," the numbers. A great deal has been written about the Star Schema and we'll assume the reader is already familiar with the basics.

Star Schemas are slow to update because they are heavily indexed. For example, each dimension table carries a primary index, a separate index for any of the attributes that are foreign keys, and may even index certain non-key fields that are frequently used to constrain or group a query. The fact tables are usually index-heavy because they are composed of a multi-part key (three or four at a minimum, sometimes a dozen or more) and just one or more "facts." The preferred approach for a Star Schema is to populate it at the lowest possible level of detail, not to summarize it, so that any kind of question can be answered. Because this tends to make the fact tables large (huge, even), selectively summarizing (aggregating) some of this data into smaller, dependent fact tables is typical.

SOLVING THE PROBLEM

The problem with Option 2 is that it may be possible to jam millions of records a minute into the tables, the time it takes to adjust the indexes and perform some aggregation can cripple the strategy. This has always been the greatest drawback to the approach. Some specialized databases have come and gone that addressed this problem, to a certain degree, but there is no general solution to users of the "merchant" databases like Oracle or DB2. With that in mind, it seems logical to look for a solution that combines some good algorithms with ramped-up computing platforms.

INDEXING

Parallel processing used to be an exotic solution that required lots of money and skill. With the emergence of data warehouse

appliances, like DATAllegro, applying the power of parallel processing to the data warehouse load/query problem has become much simpler and far less costly. In fact, an appliance approach, especially one that works in concert with an existing database like Oracle, is a perfect Moore's Law play. Data can be "ingested" to the dimensional schema in a massively parallel mode, and clever partitioning of the indexes in RAM, made possible by the low cost of memory, allows updates to occur continuously and in near-real time, without sacrificing the indexes needed to keep lots of queries humming. In fact, it is possible to not only load data in near real-time, but also "publish" it, that is, make it available for queries, and launch the queries to execute in near real-time. In other words, data can flow from source system, to the data warehouse and out to the recipient of a query in a matter of seconds.

AGGREGATION

The single greatest factor in improving query performance in a dimensional schema is aggregation. Data is (or should be) loaded at a very fine grain, such as Call Detail Records, individual items from Point-of-Sale system or clickstream data from a commerce site. Individual transactions at this level possess attributes that are lost as data is aggregated. Even though queries may resolve at aggregated levels, the selection and grouping of the data may depend on these lower-level attributes. In those cases, having only aggregated data will not suffice, and a data warehouse solution needs to be able to sift through mountains of records to resolve the query. On the other hand, many queries are concerned with only the aggregated data and sifting through millions or even billions of records to get an answer is not very efficient. Having some data pre-aggregated is very advantageous. But which data?

It turns out that these partial aggregation approaches, often called sparse aggregation, are a little tricky. They are possessed of an ineffable shiftiness that makes it very difficult to come up with one, permanent recipe. Some aggregates are used at the beginning of a month, others at the end. Some vary by time of day. It is very difficult to find a pattern. One thing that is certain is that the right scheme at any given point is to create only a small number of aggregates, but selecting the right ones is an art. However one comes up with an approach, the load/query problem will reappear unless the database is able to maintain and update aggregates at the same speed it can maintain indexes. Having a scalable, cost-effective appliance that is

geared to perform that function is an infinitely better solution than enduring long load windows and/or sluggish query performance.

CONCLUSION

In a period of abundant computing resources, the solutions to the load/query problem with data warehouses are feasible. The best solution is one that applies the power of massively parallel computing where it is needed, clever use of abundant RAM and a clear sense of mission, evidenced by the use of dimensional schema to provide crisp query performance. Purpose-built appliances, designed to provide exceptional performance with extremely attractive TCO, like DATAllegro, were bound to appear, and now one has.

The limitations of data warehouses, long update cycles, poor query performance, restricted access and concurrency governors are artifacts. Thanks to Moore's Law or, more appropriately, the innovations of the high technology industry it forecasted, resources are available at price points that make sense. By combining these tools with innovative and effective designs and algorithms, a purpose-built data warehouse appliance can eliminate all of these drawbacks.