

# BI Is In the Details:

How Columnar Technology Leverages the Latent Value in SQL Databases



## Hired Brains, Inc.

By Neil Raden  
September, 2009

## ABOUT THE AUTHOR



Neil Raden is the founder of Hired Brains, Inc./Hired Brains Research. Hired Brains <http://www.hiredbrains.com> provides consulting, systems integration and implementation services in Business Intelligence, Data Warehousing, Performance Management, Advanced Analytics and Information Integration for clients worldwide. Hired Brains Research provides consulting, market research and advisory services to the technology industry. Based in Santa Barbara, CA, Neil Raden is an active consultant and widely published author and speaker and is the co-author of “[Smart \(Enough\) Systems](#)” (Prentice Hall, 2007) He welcomes your comments at [nraden@hiredbrains.com](mailto:nraden@hiredbrains.com).

## Executive Summary

If you are responsible for BI (Business Intelligence) in your organization, there are three questions you should ask yourself:

- Are there applications in my organization for combining operational processes with analytical insight that we can't deploy because of performance and capacity constraints with our existing BI environment?
- Can we only allow analysts to operate on subsetted or aggregated data due to performance limitations?
- Are we unable to do the optimization work manually to allow deep, detailed analytical queries to execute? Shouldn't the computer be able to do the hard work, such as performance optimization, that the IT staff currently performs?

Yesterday's requirements for data warehouses were to provide a standalone utility for publishing scrubbed, integrated data in relatively small chunks. Today, organizations need to understand every aspect of their business, down to the finest detail. Aggregating information to suit an old data architecture deprives analysts of the rich record that is lost when data is summarized. To really get the latent value out of ever larger data warehouses, BI tools need to speak their language, and that language is SQL. Products that are capable of generating SQL robust enough (and fast enough) to interactively perform complex analytics are called ROLAP, or Relational OLAP. Products like MicroStrategy and Tableau are premier examples of the ROLAP class<sup>1</sup>.

As the role of data warehouses continue to evolve, and volumes continue to expand, performance is emerging as the key issue. Originally conceived as repositories of accumulated historical data, updated periodically and queried only after all updates, indexing, aggregation and subsetting were performed, the mission of data warehouses must support an increasingly mixed usage profile. They are becoming indispensable in on-line commerce, operational processing and even automated decision-making. Update windows are shrinking, in some cases to no time at all as they are online 24/7 and updated in real time.

The load process of a data warehouse is a complicated one, as data is extracted and fed from sources, scrubbed and integrated and finally loaded. Query processing ranges from simple selections to complex analytical processing to data mining and predictive modeling. Queries can be initiated by people or

---

<sup>1</sup> MicroStrategy and Tableau are not identical in function. Though both reach into the database to access large amounts of data for analysis, Tableau is oriented to provide interactive visualizations of the data, while MicroStrategy uses the database to perform complex calculations before displaying the results for further navigation and investigation. This is, of course, a vast simplification of the features of these products.

unattended processes. Existing relational database optimizers and load balancers are overwhelmed by the depth, breadth and volume.

Improving performance is dependent on many factors, and not just the performance of the technology stack. In most cases, expert developers and tuners, always in short supply, are pressed into service to keep up with demand. There is a constant tradeoff between indexing and aggregation for speeding up query processing (which slow the load process), and fewer indexes and aggregates to speed the load process (but degrade query performance). These efforts are never permanent, either. The time it takes developers to constantly tune and reconfigure systems as requirements cycle and change can be unbearable. Important initiatives can be postponed or even cancelled. Better solutions are called for.

When organizations reach the point where their data warehouse applications impede business initiatives, they turn to alternatives. One approach to load latency, for example, is to create very simple structures, with few or no indexes, which allow records to be streamed into the warehouse at processor speed. The downside is that query performance suffers miserably, only to be addressed with massive parallelization, a solution that has worked well for some large data warehouses. However, the cost for this kind of solution is prohibitive for many organizations and usually involves a complete redesign and implementation (with concomitant learning curve) of the entire data warehouse environment.

The logical solution to data warehouse performance today should be a non-intrusive one that complements existing environments, and represents breakthrough achievement and delivers at least one order of magnitude boost in performance. This paper will discuss how Vertica enhances the capabilities of analytical work through the implementation of different architecture for relational database-based analytical work. By simply rotating the orientation of the database from rows to columns, dramatic improvements in load and query speed, scalability and cost effectiveness are achieved. For organizations employing analytical tools that depend on database performance such as Microstrategy, this is particularly pertinent.

Note to reader: If you are comfortable with the issues of performance techniques in relational data warehouses, you may want to skip the section on “Data Warehouse Performance Limitations” and proceed directly to “Some History: How Did We Get Here?”

## Data Warehouse Performance Limitations

The domain of data warehousing is analysis and is characterized by queries that are far more complex than operational or Online Transaction Processing (OLTP) queries. For example, a business may wish to ask what appears to be a simple question, such as:

*Show the number of adverse reactions by drug type, by prescriber type, this year versus last*

But a question like this could dim the lights in the computer room because it might cause the database to thrash through literally billions of records, and sort and merge them a few times before arriving at an answer. Because there are so many criteria involved in this seemingly innocuous question, a relational database would have no hope of finding the answer without some physical optimizations beforehand, especially indexes on the most relevant attributes and aggregations to prevent scanning and sifting all of the un-aggregated data.

Relational databases fulfill a number of roles simultaneously. Capturing accounting entries, registering point-of-sale ticket items or making airline reservations are all examples of *transaction processing*. Until a few years ago, the major relational databases were mostly focused on the speed and reliability of these operations, also called OLTP for on-line transaction processing. Good throughput was assured by a combination of careful design in the database engine itself, such as optimizer technology, as well as good application design by developers. The most common approach was to use a normalized schema design that minimized or eliminated duplication of information, allowing for fast insert of a small number of records or fast delete without searching for related records. Indexes were also kept to a minimum as they added overhead to the insert or delete operations.

The downside of these approaches was the difficulty in processing analytical queries against these designs without constructing a completely separate physical implementation – a data warehouse. “Getting the data out” was the primary motivation for building data warehouses. Data warehouses were originally conceived as being batch updated, read-only repositories of historical data to provide the reporting and, somewhat later, the analytical needs not met by the OLTP systems. But data warehouses have an entirely different performance problem to solve. In almost every way, data warehousing stretches the capabilities of relational database management systems. In the loading of a data warehouse, extracts from operational systems are applied, often many millions of records at a time. Systems designed for transaction processing are

overwhelmed by the volume, and special tools are needed that bypass the transaction logging, rollback/commit and incremental referential integrity checking. The indexes used in data warehousing are larger and more complex, to facilitate the types of queries that are common in decision support. Rebuilding these indexes can be too time-consuming, resulting in update cycles that are too long. In fact, as data warehousing has matured, required update cycles are approaching real-time.

In the same way that OLTP designs are deficient for reporting and ad hoc queries, the classical design of data warehouses as read-only repositories updated on a periodic basis is running into some limitations today. As update frequency increases from monthly to weekly to daily and finally to intra-day, squeezing in a data warehouse refresh is becoming increasingly difficult. More and more applications are emerging for data warehouses that require real-time information, finally compressing the update cycle time to zero. While data warehousing used to be able to count on the leisurely operation of a quiet database and instead focused its optimization efforts on query processing, today both aspects require strict attention.

And speed is not the only requirement. Reducing the variability of queries is sometimes more important than actual speed. It is far easier for people (and applications) to balance their own workload when they have a reasonable expectation of the time to process a query.

With evolving standards-based architectures and applications, the latent value of data warehouses is being unlocked by composite applications that combine operational and analytical processing in a single application. The “clients” of a data warehouse in the next few years are just as likely to be unattended agents as they are people. A DBA’s toolbox of tuning solutions for data warehouses today includes schema design, aggregation and indexing, and they are being stretched by these new developments. Data warehouses today are characterized by four performance-related deficiencies.

- Data preparation latency
- Query latency
- Enhancement latency
- Query governors

Three performance enhancing techniques have been employed to solve relational database performance problems: physical schema, indexing and aggregation. They are discussed in the Appendix. In most cases today, though, these techniques are not enough.

## Some History: How Did We Get Here?

To understand the value of today's high performance analytical databases requires some historical perspective. Two decades ago, the information technology (IT) resources for Business Intelligence (BI) were not adequate to gather and store any but the most high-level summary data. To a lesser extent, this was also true for the operational systems which resorted to sometimes extreme measures to minimize the amount of (on-line) storage, such as the infamous Y2K effect. Over time, however, as storage costs dropped dramatically, a newer approach for BI emerged: to gather information from operational systems and store it elsewhere for various purposes such as reporting or extraction/downloading to other applications. Gradually, this process became formalized as the data warehouse concept we know today.

BI tools as we know them today pre-date the data warehouse concept and operated from small regions on mainframes (Comshare, EPS), relatively compact servers (Express, IFPS) or even desktop computers (Javelin, Cognos Impromptu). As a result, the tools were configured for fairly small amounts of data. Finance departments prepared reports for monthly closings, formal management reports and even budget preparation and variance reporting. Marketing and sales departments tracked monthly sales, coupon redemption and activity reporting. The common thread was high level data and small amounts of it. Despite almost unimaginable expansion of computing and storage resources over time, this model persists to this day. The preponderance of BI applications in production still operate from data marts, "cubes" and other extracted data sets, not the primary analytical data stores (data warehouses, e.g.) as the direct source for analytical queries.

There are two reasons this has endured. First, most data warehouses were never designed or built to support large scale analytical query activity. This may seem like a paradox, but in practice, data warehouses were designed to support batch reporting and the publishing of data to downstream analytical applications, not ad hoc analysis or complex analytical queries. General purpose relational databases required a great deal of physical tuning to be able to support this type of activity as their internal optimizers were designed for a different type of usage. To a certain extent, this is no longer the case, and the nature of that transformation is a major theme of this paper. Second, expectations for query performance from data warehouses were so low that the BI tools vendors kept with their original program of operating from aggregated subsets of the data in their own engines.

This arrangement wasn't universal. Some database vendors attacked the problem and developed relational databases designed for BI. Teradata, for example, delivered a product that was the anchor of some of the largest and most complicated data warehouses and remains a leader in the field by bundling the engine with its own parallel processing hardware. Red Brick delivered a

database based firmly on the idea of dimensional schema, but never gained enough traction to persist and was eventually acquired. Other offerings languished and the major production databases such as Oracle and DB2 slowly improved their functionality and started to narrow the gap with the specialty tools<sup>2</sup>. The problem was, the BI tools were not positioned to take advantage of this change in orientation. To truly leverage the latent value of data in a relational database for analytical purposes requires either an army of highly skilled SQL programmers, or a BI tool that can create efficient SQL dynamically. By the mid-90's however, a new crop of BI tools emerged to do exactly that. They not only fired SQL against databases, they provided the kind of desktop BI that analysts needed.

## Enter OLAP

In 1993 a white paper entitled “The Twelve Rules of On Line Analytical Processing” (OLAP) was conceived by E.F. Codd, the father of the relational database. There was some controversy about these rules at the time as, apparently, Codd was paid by Arbor Software (merged with Hyperion, acquired by Oracle) which had a vested interest in the “rules” corresponding with their software product, Essbase. As a result, OLAP remains more of a marketing term than a rigorous definition. Codd laid out the concepts of dimensionality of the data, navigation, flexible reporting, etc.

Codd/Arbor did identify some unique characteristics for OLAP. While mute on actual implementation or architecture, it was clear that the meta-model distinguishes OLAP from relational databases – the dimensional orientation of the data rather than record-based, and the ability to interactively, in more or less real time, navigate through the data and calculations.

Because the term OLAP was a little fuzzy, and many vendors had products that were at least directionally headed to OLAP functionality, the term became nuanced with qualifications such as OLAP-lite, MOLAP, HOLAP, DOLAP. In short order, though, the real differentiation emerged as MOLAP versus ROLAP.

MOLAP is multidimensional OLAP (which is a *non sequitur* as all OLAP is multidimensional) meaning, specialized structures, or “cubes” store the data that service the queries and their request languages are proprietary or, much later, MDX, a pseudo standard. ROLAP relies on the relational database to execute the queries and uses the universal standard SQL.

---

<sup>2</sup> In practice, some of the OLTP database vendors simply acquired other database technology to expand their offerings such as Sybase's acquisition of what is now Sybase IQ or IBM acquiring Informix and Red Brick. IBM and Oracle have chosen to position their databases as capable of a full-range of processing types, but in practice, have not been able to keep up with the offerings of vendors dedicated to analytics.

## ***ROLAP: Leveraging the Relational Database***

Information Advantage, Holos, MicroStrategy and Prodea were early pioneers in ROLAP. All of them except MicroStrategy, the sole pioneer still active and thriving, were acquired and disappeared (though the founder of Prodea, Larry Barbetta, went on to create nQuire, which was acquired by Siebel and subsequently Oracle and is now very active in the space as Oracle BI). Part of the reason for the attrition in ROLAP was that the market perceived them as too slow and too complicated to administer. In most cases, organizations didn't have data warehouses yet, so the limited range of MOLAP tools, coupled with their excellent query performance, propelled them ahead.

Microstrategy saw early it needed to do two things – optimize its SQL-generating engine for each database (and even each version) and push the processing as far into the database as possible, keeping their own engine light. The logic was that a relational database existed in an environment that had access to greater storage and processing as well as people and procedures to provide security, reliability, access (via SQL) and sustainability. A ROLAP tool could offload most of the time-consuming processing to the database and concentrate on the formation of queries and manipulation of results.

There were drawbacks to ROLAP. Multidimensional or “cube” based OLAP tools held a significant query performance advantage for a number of reasons:

- The amount of data propagated to the cubes was small
- The cubes themselves were physically organized for OLAP queries
- The query languages were proprietary and thus needed no optimization
- Relational databases were not optimized for complicated, multi-pass SQL queries
- The volume of data in data warehouses was significantly greater than in MOLAP tools
- There were many more tuning parameters to a relational database than to a MOLAP cube
- Initially, best practices for designing schema and tuning them in relational databases for analytical processing were not available
- Even when good practice for ROLAP was clearly understood, relational DBA's had difficulty deviating from their practices for operational systems

## ***Why ROLAP is Important***

A piece of paper or a computer screen as pieces of real estate is only capable of presenting a small amount of data (excepting visualizations). It is natural to assume that BI deals only with these relatively limited domains. As a result, BI has been applied for the most part to report and analyze the effects of business events retrospectively—what has happened. But today, partly because of the capability to do so much more (the push of technology) and partly as the result to do so much (the pull of business requirements), understanding cause and effect

and predicting the future is now the goal in BI. The presumption in ROLAP is that there is a mountain of information buried in the details that's been obscured by aggregation. To truly understand the meaning behind the data, BI has to be able to operate at every level of detail, seamlessly and accurately. The resource in short supply is time, so there isn't time for refining and the moving the data from one structure to another, we have to rely on the primary analytic database to load the data continuously and quickly while providing extreme query performance.

The solution is to flow data into one structure that is capable of both very low latency data load and excellent query performance at the same time. And by leveraging the universal SQL language, maximum flexibility is assured. Even when the result of a BI query appears to be a report of summarized data, the calculations behind it can be driven by attribute instances of thousands, millions or even billions of low-level detail data.

In addition to the typical uses of BI in an organization, there a rapidly expanding need for operational robots whose SQL is dynamically generated based on instantaneous conditions to:

- Provide detail to a rules engine or predictive model for fraud detection
- Generate a real-time recommendation
- Issue purchase orders for a continuous replenishment system
- Sift through streaming Web clickstream information to dynamically reconfigure web sites

## **What Columnar Databases Do**

For a complete description of the features and functions of Vertica, please refer to their website at [www.vertica.com](http://www.vertica.com). What follows is a more general description of the concept.

Classic relational databases store information in records and collections of records in tables (actually, Codd referred to them as tuples and relations, hence, relational databases). This scheme is excellent for dealing with the insertion, deletion or retrieval of individual records as the elements of each record, or the attributes, are logically grouped. These records can get very wide, such as network elements in a telecom company or register tickets from a Point of Sale application. This physical arrangement closely mirrors the events or transactions of a business process. When accessing a record, however, the database accesses the entire record, even if only one attribute is needed in the query. In fact, since records are physically stored in "pages," the database has to retrieve more than one record, often dozens, to access one piece of data. Furthermore, wide database tables are often very "sparse," meaning there are many attributes with null values that still have to be processed when used in a query. These are major hurdles in using relational databases for analytics.

Columnar database technology inverts the database structure and stores each attribute separately. This completely eliminates the wasteful retrieval (and burden of carrying the needless data) as the query is satisfied. As an added benefit, when dealing with just one attribute at a time, the odds are vastly increased that instances are not unique which leads to often dramatic compression of the data (depending on the “distinct”-ness of each collection of attributes). As a result, much more data can fit in memory, and moving data into memory is much faster.

The dramatic uptick in load and query speed from columnar orientation, column-wise indexing and its attendant data compression for greater in-memory processing are only part of the story. Vertica is also a “massively parallel processing” database (MPP) which provides for linear scaling and fault-tolerant processing. The concept behind MPP is that processors are tied to certain pieces of the data, the data is evenly distributed across the nodes and the number of processors can expand almost without limit, using relatively inexpensive, non-proprietary components.

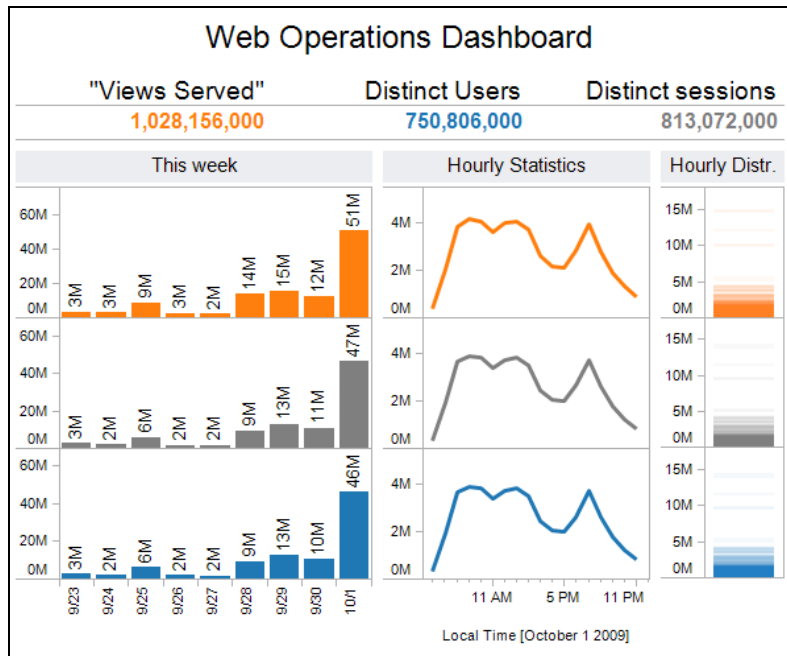
In summary, the Vertica Analytical Database innovations include:

- shared-nothing (MPP), column architecture residing in commodity hardware
- aggressive data compression leading to smaller, faster databases
- auto-administration: design, optimization, failover, recovery
- concurrent loading and querying

This combination of features makes Vertica a perfect companion to a high-end analytical tools like Tableau and Microstrategy. Some examples are:

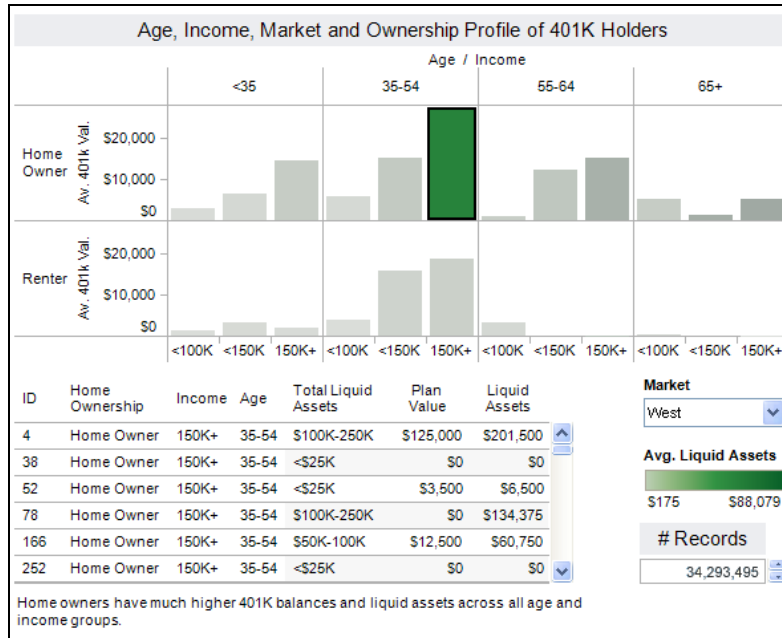
Market basket analysis: The application looks at each item purchased and determines a list of complementary items to recommend to purchasers. The market basket application uncovered insightful new data about the items that customers typically purchase together, enabling one organization to present these items as complementary offerings to online customers. As a result, the average order size for orders with the market basket pairings is more than twice the average order size for orders without pairings. In addition, the information has helped this company better serve its customers, with a deeper understanding of purchasing preferences.

Visualization of intraday/near real-time data: For example a web operations dashboard for a successful website looks at millions of records in near real time to identify anomalies, problems and/or opportunities. One of the objectives is to prevent outages which can cost tens of thousands of dollars a minute. ([http://news.cnet.com/8301-10784\\_3-9962010-7.html](http://news.cnet.com/8301-10784_3-9962010-7.html)). See Figure 1 below. Immediately, we can see that traffic spiked on October 1 and that hourly statistics show a multi-modal distribution.



**Figure 1 -One view analyzing hundreds of millions of real-time records about website views, sessions and users for a large-scale media website.**

Visualization of detailed demographic data to spot trends and correlations: Understanding and analyzing demographic data such as age, income, and home ownership status of millions of individual users cannot always be done at an aggregate level. Specifying groups of individuals on the fly, based on the detailed records allows for greater insight. In Figure 2, it's easy to see at a glance that homeowners in the 35-54 range have the highest balances and that those with incomes above \$150K lead the pack. Some of the individual details of the customers in that group are also included in the display.



**Figure 2 - Dashboard profiling the age, income, market and home ownership status of a financial institution's 30+ million 401K holders.**

Dynamic Business Process Analysis: creating a business process without a formal process definition by assembling events and subevents from logs and piecing them together as a time series. This allows you to evaluate not only the business processes you've defined formally, but all the ones that are not yet identified. This can include events and process steps that occur externally, such as your partners and/or customers

## Conclusion

Until recently, data warehouse performance was regarded as a manageable problem. Business people seeking information were not, strictly speaking, on the critical path as far as systems performance was concerned. Daily loads that did not complete overnight needed to be addressed, and overloaded systems could be cleared by governing the number of users or limiting access to certain data. All of this could be addressed with existing IT best practices.

But the concurrent requirements for fast update of data warehouses and near-OLTP response times to queries cannot be solved with existing relational database tools today. Vertica offers a purpose-built solution for taking over complex analytical tasks. All of these features work in concert with the existing BI tools and environments and supplement their features smoothly and transparently. Data warehouses have a particularly severe problem to solve due to continued scaling up in volume and workload and vastly increased expectations in speed. Vertica offers a transparent solution with excellent TCO potential.

## Appendix:

### ***Dimensional Schema***

A great deal has been written<sup>3, 4</sup> about various “dimensional” designs for relational databases, but in essence, these approaches are designed to avoid excessive table joins and scans by arranging reference information around the actual “facts” that are analyzed, and heavily indexing them all of the tables. This sort of approach would be unacceptable for an OLTP, but it works fairly well for data warehouses. In general, the “facts” are fairly low level bits of data, and the “dimensions” describe hierarchies for rolling the data up. The problem is that the rolling up, or aggregation, of the data can be time consuming.

### ***Effect of Aggregation on Data Warehouse Performance***

In most cases, data warehouses today contain very fine-grained data even though most queries request aggregated information. To facilitate this, a typical tuning technique is to design a number of aggregate tables (database tables that contain aggregated or “rolled up” data) that lie at the axes of common queries. In fact, designing and maintaining a sparse aggregation scheme<sup>5</sup> is the single most effective performance booster for data warehouse queries<sup>6</sup>. In other cases, incremental aggregation is performed, where the dependencies of the data, described by the hierarchies in the dimension tables, can prescribe a set of incremental updates to the existing aggregate. This latter approach, though difficult, is absolutely essential for real-time updating of a data warehouse. While it may be possible to trickle-feed new un-aggregated data into the data warehouse in real-time, the latency in updating aggregated data can derail the whole process.

In a data warehouse, data is extracted from various sources and typically rearranged in a dimensional schema. The data is separated into Facts and Dimensions. In a typical arrangement, a dimension might be Product or Store and the fact table would contain instances of transactions, such as sales, by product by store by time. Because dimensions contain both hierarchies (product → type → line of business or store → district → region), and non-hierarchical attributes (store size, store type, product packaging) even relatively simple models can rapidly expand to different role-ups.

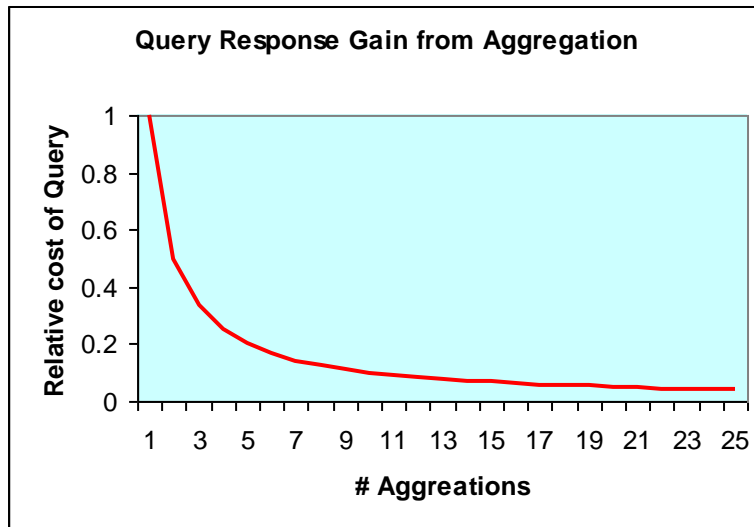
---

<sup>3</sup> Kimball et al, “The Data Warehouse Toolkit”

<sup>4</sup> [http://www.intelligententerprise.com/030630/611warehouse1\\_1.jhtml](http://www.intelligententerprise.com/030630/611warehouse1_1.jhtml)

<sup>5</sup> Sparse aggregation simply means that only a small number of the possible aggregate tables are created, anticipating which are most likely to be used.

<sup>6</sup> This feature is known as “aggregate awareness,” although it is only effective if either the person or process initiating the query is aware of the aggregates available, or if the database software is able to “rewrite” the query at runtime, a feature that most databases lack.



As can be seen from the diagram above, a relatively small number of aggregate tables can have a dramatic effect on query performance. While it may make sense to roll up one or more levels for very large tables, like cash register receipts to daily sales, for example, compressing the number of rows by 70-, 80- or even 90-99%, this is usually a starting point, but not a complete answer. For example, if the cash register receipts were dimensioned by Time, Location, Product, what aggregation level would be chosen for Product? Even a modest-sized data warehouse with many dimensions, levels of hierarchy and non-hierarchical attributes will exhibit the same need for acceleration, but the combinatorics of grouping scenarios can rapidly exceed even good intuition. A data warehouse with only six or seven dimensions could easily have *thousands* of distinct aggregation possibilities, but most of the possible gain can be achieved with only a few tables. The question is, which ones?

Two factors tend to dampen this effect. First, this sort of dramatic improvement is only possible when the developers know which of the nearly infinite variety of aggregates will give the greatest lift, which is unlikely, and even if they did, the constant evaluation, development and tuning is too costly. Second, the answer is not constant as usage patterns change and cycle constantly. Today's aggregates may not deliver next week.

## ***Indexing a Database***

Traditional relational database indexing is fairly uniform across products and generally comes in two flavors: B-trees and hashes, each of which has its strengths and weaknesses. For the most part, neither one is satisfactory for an analytical environment, for a number of reasons. First, these indexes are essentially maps of column values to rows in a single table, and provide no

information about the intersection of these values across tables. Second, the content of the actual index is either the key value itself, a unique, internal identifier of each row, called a rowid, or RID or even the unique key of the row that contains it. When tables reach 10's or 100's of millions or even billions of rows, which is common today, these indexes become unwieldy, consuming huge amounts of memory and/or causing swapping and thrashing in memory. The key to index performance is to have it stay resident in memory; otherwise it just adds more fetching overhead. One alternative that can keep indexes small is the bit-mapped index.

Bit-mapped index is generic term and there is wide variation in the performance of these indexes. The advantage of a bit-map is that it is very small if the number of distinct values (the cardinality) of the attribute being indexed is very small, but for higher cardinalities, the bit-mapped indexes of the major database vendors are not very effective. Another advantage of a bit-map is that in some cases, it can substitute for the values themselves and applying Boolean operations to them in memory can speed up operations to a great extent. This is all conditioned on the index being in memory for 100% of the operation being performed, a situation called a *covering index*.

It is extremely difficult to get a covering index in a standard database because the amount of available memory on the server is shared by, first, the operating system and any other applications running, and second the database server. But the database server has to juggle a lot of processes at once beside just query processing, and the available memory is continually getting recruited for caching temporary results and pages from the database, the sometimes huge requirements of a join or sort/merge, and there is no guarantee that an index will be able to request the memory it needs to "cover" the operation.

### ***And Storage***

One last thing to consider is that while the performance, size and cost of computing devices such as CPU's, RAM and disk drives have and continue to improve at dramatic rates, the mechanical ability to fetch data from a rotating platter has not kept up. Any technique that is "disk bound" simply cannot keep up with one that is in-memory.