How NoSQL Solves Business Problems

Generally, NoSQL solutions are designed to process data in a distributed environment. Distributing data across multiple nodes in a data center results in a linear performance improvement. However, understanding how to store data in ways that allow scalability requires a rethinking of your overall approach to systems design.

The scientist-philosopher Thomas Kuhn coined the term *paradigm shift* to identify a recurring process he observed in science, where innovative ideas come in bursts and impact the world in nonlinear ways.

Today, organizations are experiencing a paradigm shift in situations where there is a need to manage large amounts of data driven by web-based systems. As the amount of available data continues to explode, traditional data management methods struggle to keep pace. New ideas about how data is stored, retrieved, and used continue to emerge and challenge conventional wisdom.

We use Kuhn's concept of paradigm shift as a way to think about and explain the NoSQL movement and the changes in thought patterns, architectures, and methods emerging today. As we move through this paper, we'll examine the drivers and use cases that opened the NoSQL door. At the end of this paper, you will know about the laws driving the NoSQL movement and, when presented with similar scenarios, you'll be able to identify what NoSQL options are available to solve your business problem.

Why have organizations moved away from traditional SQL and relational technologies in favor of nontraditional NoSQL systems? We'll begin by looking at Moore's Law and the PowerWall, the two rules at the heart of the NoSQL movement. We'll examine how these two events sparked a migration away from single-CPU serial information processing toward parallel processing more closely aligned to NoSQL systems. Finally, we'll see how the NoSQL movement is gaining steam and influencing how organizations think about solving business problems and implementing technology in today's rapidly changing environment.

NoSQL leverages low-cost multicore processors

As the growth of the web exploded in the late 1990s, it became clear to strategic planners that, to remain competitive, the ability to rapidly process large amounts of data was critical. While many hoped that single-chip performance would scale at the same rate as the data explosion, Moore's Law and the PowerWall proved that this was not possible.

Moore's Law is a rule of thumb in the history of computing where the density of integrated circuits placed on a silicon chip doubles every two years. This observation, attributed to Gordon Moore, has remained remarkably accurate over the past fifty years and is expected to continue to hold true until at least 2015.

In figure 1, we can see how the density of chips exponentially increased starting in the early 1970s and is projected to continue for the next several years.

However, a different rule applies to how the rate at which circuit speed increases. In 2005, a problem prevented clock speeds from increasing at the same rate that circuit densities increased, causing chip designers to find a new way to meet the continual demand for faster processing.

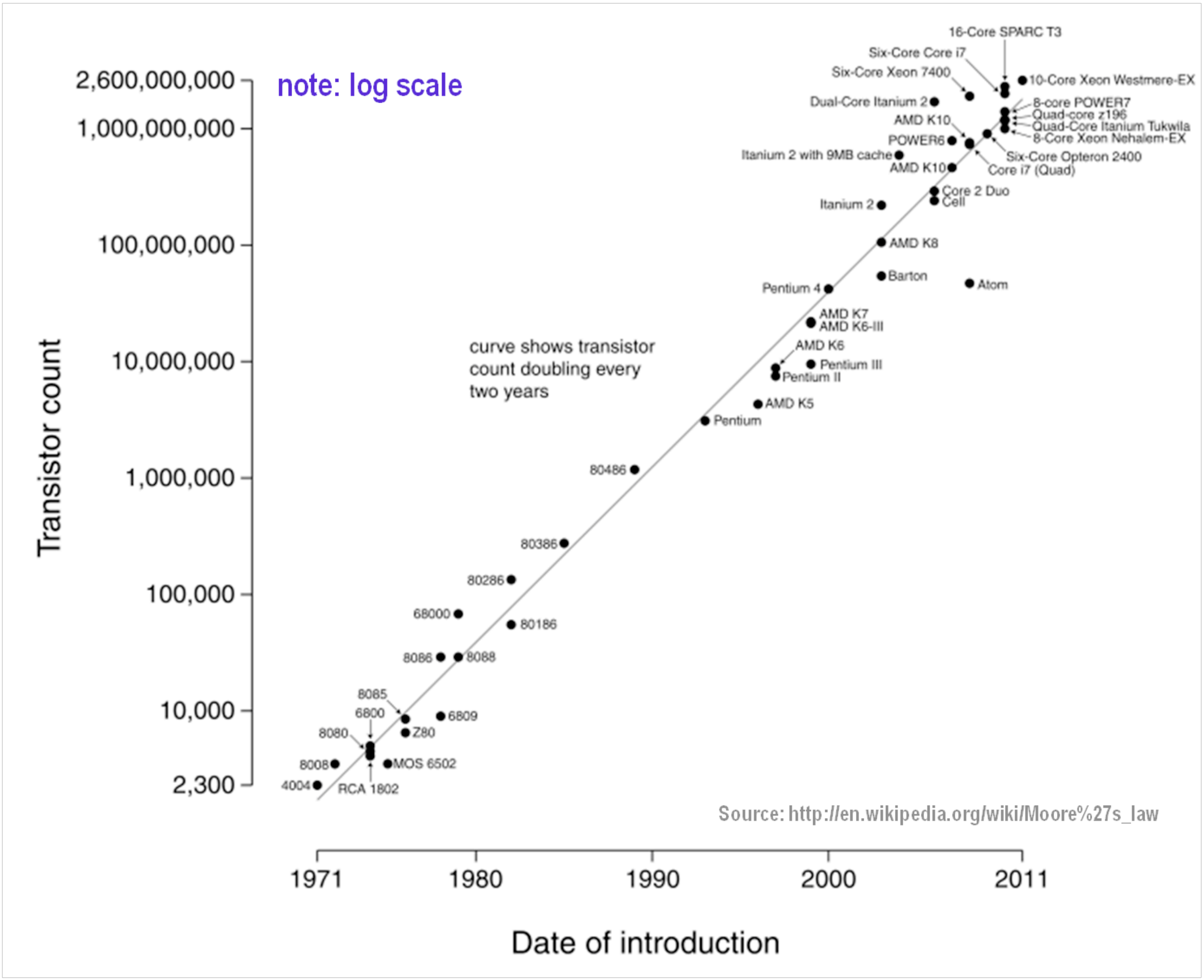

Figure 1 Moore's Law states that the number of circuits on a chip doubles approximately every two years. This chart shows that, when we graph the log of number of transistors vs. time, we get a straight line. This is characteristic of exponential growth.

One factor that determines how powerful the CPU is is to see how rapidly their memory states change; faster change equals increased computing power. While there are many factors, the speed of light generally determines how quickly signals propagate in an integrated circuit. Designers and manufacturers believed that increasing density would increase computing power. However, around the year 2005, a barrier was reached—the PowerWall. Even though transistors were getting smaller, they needed to dissipate the heat from switching on and off. Since the rate that the heat moves away from the switches is determined by the physics of silicon crystal, the density soon prevented heat escape that was sufficient to keep the chips from failing. While engineers attempted many designs; they could not alter the laws of thermodynamics and, as a result, clock speeds stopped increasing. See figure 2 for a graphical representation of the PowerWall.

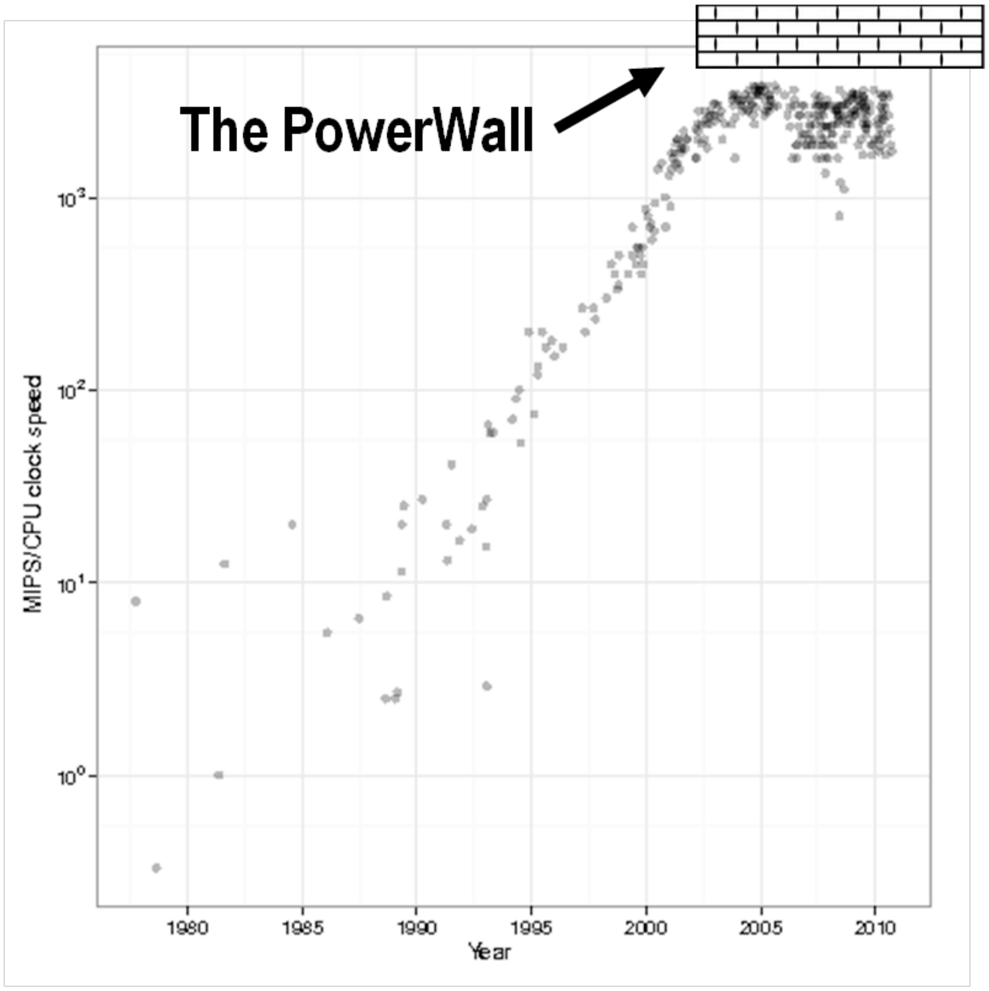

Figure 2 The PowerWall shows that the rate of exponential clock speed growth ended in 2005. This does NOT imply that Moore's Law is slowing but rather that CPUs have speed limitations. These speed limitations led chip designers to increase the number of core processors per chip and highlight how parallel programming and distributed NoSQL architectures are superior.

The PowerWall came as a surprise to many engineers and manufactures, as the continued exponential growth in processing speed came to a halt. Chips could no longer meet increased performance by increasing chip-processing speed. In spite of the PowerWall, market demand for faster processing did not decrease, in fact it continued to increase and with it, hardware engineers solved the problem by moving from serial to parallel processing when they added more core processors to a single chip.

Moore's Law and the PowerWall show us that processing large data sets cannot be done with a single CPU system. By moving away from the traditional serial processing toward NoSQL parallel processing solutions (which have parallel processing as part of their core architecture), the ability to rapidly process large data sets will improve.

NoSQL helps us shift from serial to parallel processing. The PowerWall tells us that with two-dimensional silicon chips, if you need to increase performance, serial processing has limits with respect to the amount of data that can be processed in a unit of time. In a single CPU system, each task is performed in succession with the output from one task becoming the input to the next task allowing you to manage system state. This one-at-a-time method is frequently referred to as *serial processing*. In contrast, systems with multiple CPUs can process multiple tasks at the same time and is referred to as Parallel Processing. Figure 3 shows the difference in how serial and parallel processing systems handle tasks.

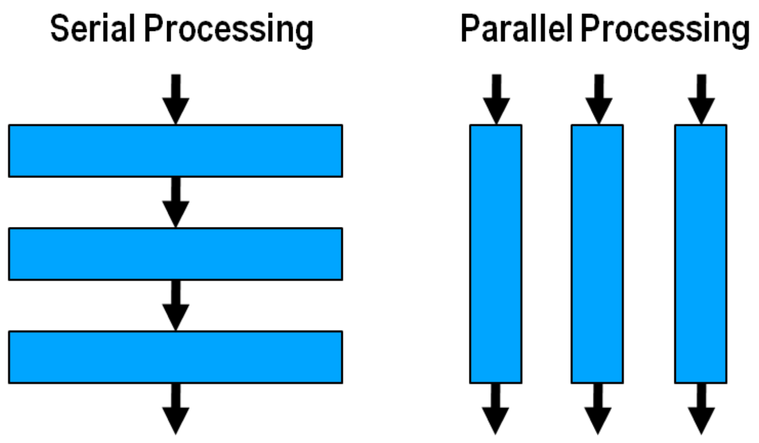

Figure 3 The PowerWall and the demands of large data sets are forcing software away from the traditional serial processing techniques and moving toward parallel processing. In serial processing, each iteration of a loop can access all variables calculated by a prior iteration. In parallel processing, the focus is on independent transformations of data where there is no interaction between transformation pipelines.

The shift from serial to parallel processing began when designers placed multiple copies of the original Central Processing Unit (CPU) on the same integrated circuit. In this design, each CPU core had its own program counter, incoming instructions, and data to fetch and store. While the design solved the original PowerWall issue, it highlighted another concern: how would the work be coordinated? What happens if you write into memory and the other CPU overwrites the result before the first one gets the data back? Even though coordination was a known challenge, it was not relevant in serial processing systems. As it turned out, however, adding multiple processors to a single-computer system made it significantly more difficult to program using traditional methods and languages.

When we look at ways to increase processing power, we can scale up (the process of making a single processor faster by increasing clock speed, adding more RAM, or adding faster hard drives) or scale out (techniques used to take existing business problems and partition them onto separate CPUs). NoSQL systems differentiate themselves from earlier technologies because they use scale-out technologies to increase the processing power. Figure 4 shows the difference between scale up and scale out.

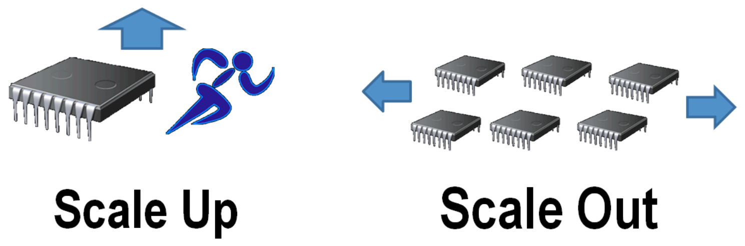

Figure 4 Scale up vs. scale out. On the left we have a collection of scale-up technologies to make a single processor run faster: faster clock rates, more RAM and faster disk drives. On the right we show how scale out technologies used by NoSQL systems, focus on dividing problems up onto separate processors and keeping the results reliable even if some processors fail.

NoSQL moves parallel processing from science to business

One of challenges of working with large data sets is the need to process large amounts of data in a short amount of time. For example, if it took two weeks to generate a weather prediction for tomorrow, the result wouldn't be very useful and, in time, you would most likely pay no attention to the information. Many scientific computing systems, often called grid computing systems, have moved away from serial computing and today use inexpensive commodity processors working together to solve complex scientific calculations. While these techniques were well understood in scientific circles, they had not yet been applied to solving business problems. With the growth of the web for business this all began to change.

In the early 90s, a single computer could search all the known web pages in a single night and create an index of the words that were stored in each web page. These web crawlers worked well when the number of web pages and changes to indexes could be done by a single CPU in a reasonable timeframe. However, by the end of the decade, the number of available web pages had exploded and a single CPU could no longer perform the task due to the size of the web and the rate of change. Companies that provided search engines, like Yahoo and Google, soon realized there was a market to tap where rapid and reliable web-searching returned profits, so they put the best of their best to come up with a solution.

The solution was a form of parallel computing. Software engineers partitioned the web into multiple zones; each zone had a computer that would index the web sites and web pages in that zone. After all zones were indexed the indexes were merged into a single index. A single search would then be able to find all of the pages that had the requested result. Parallel computing had come to web searches and the stakes were enormous.

It should be noted that most of the search companies did not store search results in relational databases. Instead, they wrote custom software to harvest web page information, extract keywords and store the list of pages that contained each keyword in there own structures. While they didn’t formally call this technology NoSQL, they were one of the first who contemplated a modern data center where commodity hardware and parallel processing solved business problems.

NoSQL drives the use of thousands of commodity processors

Today, there are many organizations that use large scale parallel processing where 10,000 to 50,000 separate computer systems, with multiple cores, all work together to solve business problems. While there are organizations that continue to depend on a single system for many tasks, they tend to process Mainframe lower volumes of data and depend on expensive and complex hardware and software to keep up with the workload.

Although parallel processing is not new, it is unique in its ability to consistently maintain processing flow even when there are component failures. Hard drives crash, power supplies fail, motherboards stop working, and yet these systems manage to keep jobs moving through the data center by using commodity processors that can be swapped out when a failure occurs. In contrast, a mainframe failure would immediately cripple an organization and could take days or weeks for a replacement system to be ordered, delivered, setup and configured before processing could resume; costing an organization hundreds of thousands of dollars in lost revenue.

The inability of relational systems to rapidly utilize additional hardware when demand increases is not the only factor motivating database architects to look at alternative solutions. The need to build complex software that will respond to changes in business requirements quickly and efficiently is also a factor.

An attribute of NoSQL systems is its ability to harness the collective power of thousands of CPUs and achieve scalable processing so that as each additional processor is added to the pool there is a linear relationship to the amount of work the system can handle. NoSQL systems are frequently written from the ground up with scalability issues in mind. To understand how these systems work we must understand the role of a large number of processors and how to handle independent component failures without impacting overall results. Today most NoSQL systems are designed with the modern 10,000 CPU data center as the core deployment platform.

Although performance concerns have been the primary driver that has driven people away from SQL systems they are not the only concern. Let’s look at a few other challenges that relational database management systems (RDBMSs) have.

NoSQL avoids the object-relational quagmire

The earliest computer systems focused on tabular data that could easily be represented in a flat file. For example, financial systems stored general ledger data in a series of columns and rows that represented debits and credits. These systems were easy to model, the data was consistent and the variability between how customers stored financial information was minimal.

When other departments in the organization began to see how analyzing data could assist them in making better business decisions, the amount and types of data captured needed to change. No longer did the simple financial information meet the needs of the entire organization, but the new data requested did not conform to a tabular structure. Engineers did what they do best: they attempted to work within their existing structure to accommodate the new requirements. After all they had a sizeable investment in people and systems that needed to be leveraged to support this new request.

One solution was to create a four-step translation process that first extracted HTML data from web pages and store the data in a series of objects that provided carefully guarded access methods to each data element. Next, they built a set of tools to shred the data (a process to extract each of the data elements from the objects and convert those objects into a series of statements that would store and retrieve the data). After the data was "shredded" into tables, a third process was created to extract all the data from individual tables and join them back together reconstituting the objects. Finally, the data inside the objects was translated back into web pages.

To simplify, the four translation steps are:

- Translation 1: Web HTML forms converted to objects
- Translation 2: Objects converted to tables
- Translation 3: Tables converted to objects
- Translation 4: Objects converted to HTML web pages

The four-translation model is shown in figure 5.

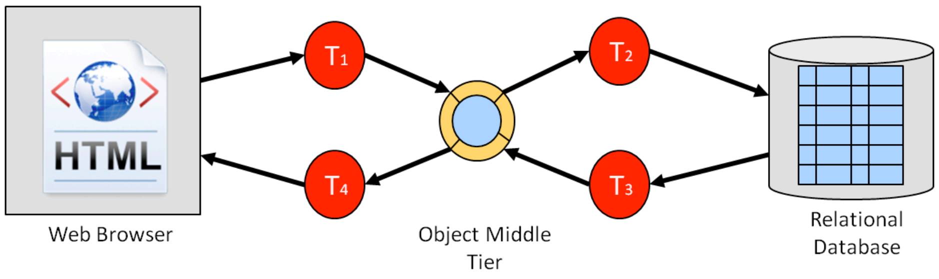

Figure 5 The four-translation model is the model that is used when objects are used as a middle layer between a web page and a relational database. The first translation is the conversion from HTML web pages to middle-tier objects. The second translation is a conversion from middle tier objects to relational database statements such as SQL. RDBMS systems return only tables so the third translation is the transformation of tables back into objects. The fourth translation is converting objects into HTML for display in web pages.

Imagine using this process on a daily basis as you come home each evening to put away your clothes. You would start by removing all of the thread, unsewing your clothes bit by bit and then putting them away in uniform bolts of cloth. When you get up in the morning to go to work, you would then retrieve your needle and thread and sew all your clothes back together again. If your thinking this seems like a lot of work, it is. Imagine storing complex web form data into tables and you will soon look for a better method.

There have been multiple efforts to mitigate the four-translation pain. Tools, like Java Hibernate and Ruby on Rails were the only options until developers realized that using a NoSQL solution to store the document structure directly in the database without converting it to another format or shredding it into tables and rows is a better solution.

This lack of translation makes NoSQL systems simpler to use and allows subject matter experts (non-programming staff) to participate in the application development process. By encouraging subject matter experts to work directly with technical staff, course corrections can be made early in the software development process saving time and money associated with rework.

NoSQL technologies show us how moving from storing our data in tables to documents opens up possibilities for new ways of using and presenting data. As we move our systems out of the back room to the World Wide Web, we'll see how NoSQL solutions can make implementation less painful.

NoSQL moves us from tables to documents

Did you know that, when you look at a web page, you are looking at a tree structure? Web pages are based on the concept of markup structures that contain hierarchical display elements. They are created by adding links to other web pages using an open international standard for sharing pages and links. Instead of storing pages in tables, the World Wide Web (WWW) was created around tree-like document structures called Hyper-Text Markup Languages (HTML). Sections of any web page could contain any arbitrary collection of other sections. Tables could contain tables, which could contain paragraphs, which, in turn, could link to images.

As you can imagine, these structures did not fit neatly into relational tables. It became necessary to use a more flexible tree structure to store documents because trees could contain trees could contain trees—and it worked.

The markup structure of HTML files was derived from other markup languages that were designed to store interchangeable documents in a standardized way. As it turned out, there was a demand for a standard that enabled users to exchange general-purpose structures that contained not only web pages but also traditional business data. Businesses wanted to exchange purchase orders and invoices using the same technologies that they used to shared web pages.

A new standards group called the World Wide Web Consortium (W3C) was created to standardize how both documents and data could be exchanged using the same markup standards. This standard was called the Extensible Markup Language (XML). The XML standard grew from a single standard that encoded tree-like data to an entire family of standards for transforming and validating XML data.

The birth of the World Wide Web has been the primary source of implementing data structures different than the traditional table store. As software developers use HTML, XML, and other web formats, we see greater innovation in system development with NoSQL structures. As NoSQL tools continue to mature, the number of new technologies and solutions available to businesses will explode.

At first glance, it may be difficult to see the impact web technologies have had on NoSQL systems. Like many NoSQL databases, the web is designed to handle distributed document storage and retrieval issues and to scale to billions of web sites and web browsers. Like web browsers, NoSQL systems are designed to work in distributed environments and many NoSQL systems use web technology (such as caching), web terminology (put, post, and get) and web formats in their design (JSON and XML). NoSQL systems strive to leverage the existing knowledge of web developers and to make NoSQL databases as familiar as possible to the web community.

In our next section, NoSQL returns more than tables; we take what might seem like a departure from discussing NoSQL systems. We'll find, however, similarities in the need to rapidly analyze and report on large amounts of data in the areas of data warehouse and business intelligence systems and how NoSQL solutions can support these same needs.

NoSQL returns more than tables

In this section, we will look at how the MDX query language (a variation of SQL), was created to solve analysis and reporting challenges. It is the story behind the MDX query system and it tells us why restricting your database to returning only tabular data structures limits your ability to analyze large data sets.

Sales organizations with large amounts of data know that by examining their sales data they can see purchasing patterns and behaviors of their customer base that when satisfied, increase sales. Initially, sales data was stored in an SQL table where each row contained the number of a particular item that was sold at a particular store each day. The resulting sales fact table, with its millions of rows, was cumbersome and time consuming when attempting to gather meaningful information.

Leveraging business intelligence information soon became the hallmark of successful organizations. The ability to implement pre-calculated totals (aggregates) resulted in quicker response and faster decision-making. Calculations could be performed on a small set of summary data and those who were willing to sacrifice disk space were rewarded with faster results.

New systems, designed around a central fact table (measurements, metrics, or facts of a business process) with a star-shaped set of dimensions (multidimensional systems with one or more fact tables), extended beyond the table, column, and row design. These multidimensional systems that use cubes (an array of data that is understood with 0 or more dimensions), dimensions, measures (the summary of an element in the fact table), and categories (something summarized for example, a particular product a company sells) soon found that the normal SQL structure was not capable of supporting their needs.

A new query language, multidimensional expressions (MDX), was created. Instead of returning flat table structures, it returned data structures that could be used with pivot table-type interfaces (a data summarization tool used found in spreadsheets that can perform summary, count, total, and average operations on the data structure). These structures allow users to slice and dice data with a simple mouse click. As MDX's popularity and functionality increased, many vendor systems migrated or created languages and systems to support this new structure.

While not a radical change from SQL, MDX was the first departure from the SQL standard that set the stage for a new way of thinking where RDBMS did not dominate.

For many, the MDX language isn't associated with the NoSQL movement most likely because the data is stored in tabular structures. MDX is not so much a radical departure from SQL as it is an evolutionary step away from SQL to meet the needs of data analysts. MDX, when combined with concepts such as OLAP cubes, dimensions, categories, and measures created its own mini-paradigm shift that showed how the mighty SQL could not meet all the needs of an organization.

It's clear that every day tens of thousands of users go to work and analyze data using these concepts. It's important for the NoSQL community to recognize how important these systems are in organizations today. NoSQL systems need to not only work with data warehouse and business intelligence (DW/BI) solutions, they need to enhance their functions so that they can be extended to cost-effectively use large numbers of processors to create summary data.

The DW/BI ecosystem is a good example of how specialized our data processing systems have become. NoSQL will continue to widen the gap between traditional notions about data and computing to today where new ideas, approaches, tools and most importantly new thinking are used to solve business problems.

Summary

In this paper, we saw how NoSQL solves challenging business problems. Opting to use NoSQL technologies can help organizations gain a competitive edge in their market, making them more agile and better equipped to adapt to changing business conditions. The NoSQL approaches that leverage large numbers of commodity processors save companies time and money and increase service reliability.

Last updated: November 12, 2012