# NoSQL, Big Data and Graphs

Technology Choices for Today's Mission-Critical Applications

# NoSQL, Big Data and Graphs

## Technology Choices for Today's Mission-Critical Applications

It used to be that databases were just tasked with digitizing forms and automating business processes. The data was often tabular – take an accounting ledger, for example –and the processes being modeled were reasonably static. Today, the types of data that we are interested in are much more diverse and dynamic. We are interested in capturing information about all sorts of things that are happening around us, which requires us to deal with dynamic systems that often generate large quantities of data that are semi-structured and volatile, where the connections between the discrete data points are as important as the sum of its distinct parts.

Broadly speaking, the trends at play are as follows:

**Volume and Velocity.** The volume of data created and handled is growing exponentially. Eric Schmidt famously said in 2010 that every day we create as much data as was created in total from beginning of written history through 2003. And it continues to accelerate as the millions of systems running our world capture more and more data from more and more sources.

**Variety.** The shape of data is becoming more complex, less structured and less predictable as well as more heterogeneous. This is reflective of an increasing variety of data sources, together with the need to model real-world systems such as social networks, biological systems and the World Wide Web.

**Connectedness.** Many of the real-world systems that businesses are seeking to model are highly interconnected. These rich, connected data sets bring an implicit need to understand and navigate relationships, which exert an effect upon both the individual data points and the overall system. James Fowler, author of *Connected,* is among the researchers who are discovering that one can often understand a person better by learning about those around him than by learning about him as an individual; and that it is not just the first-order relationships that impact behavior, but the second and third as well.

Together, these trends have led to the rise of a new breed of databases, generally referred to as NoSQL (Not Only SQL). Let's explore these characteristics and the technologies that have evolved to support them.

## A TALE OF SCHEMALESSNESS

Schemalessness does not literally mean that the data has no structure, but rather that data is bucketed into containers that are much more fluid than the relational model permits. This is useful when data lacks uniformity as well as when its structure and content can change unpredictably and frequently.

The first organizations to develop solutions for these problems were Internet giants faced with extreme growth in a highly dynamic and rapidly evolving world. Amazon developed DynamoDB and SimpleDB to deal with the problem of product catalogs and shopping cart and order data. This made it possible for them to quickly evolve the business, as it grew to sell increasing variety of items: from books to music to videos to software and household goods. Facebook developed Cassandra to back its growing messaging platform; and Google developed Big Table to handle the variety of data that is the World Wide Web. There is no doubt that these pioneers were (and still are) pushing the limits. However, as organizations embrace new challenges that entail volume, velocity, variety and connectedness, more companies are coming against the same limitations when they try to apply relational technology to these problems. The way was paved for the emergence of new database technologies, leading to the rich choice technologies available in today's market.

Because the new database technologies each addressed different issues, they ended up differing significantly in their feature set, data model, query language and architecture. From here, four major patterns emerged for real-time data processing: the key-value store, the column family database, the document database and the graph database. A fifth technology, Hadoop, oriented at large-scale batch analytics, also emerged. One thing that brought these technologies together, despite their differences, was that none were relational. Hence the term NoSQL stuck, distinguishing them from traditional databases. Some took this as a negative connotation, implying an imperative to move away from SQL, which was never the intention. The term NoSQL today is used inclusively and is generally understood to mean *not only SQL*, heralding a polyglot database landscape that extends beyond but also includes relational databases.

By understanding the characteristics of one's data and how one would like to use it, and combining it with a basic knowledge of the strengths and weaknesses of each respective database platform, one is able to make educated, informed decisions as to what technologies are most suitable for any given situation.

## BIG DATA: A TALE OF HORIZONTAL SCALING AND RELAXED CONSISTENCY

The sheer amount of data generated by human beings and our multitude of devices is growing explosively. Each action we take leads to new data being generated: whether in

retail systems, telephony networks or the World Wide Web. The need to handle this ever-growing mountain of data quickly and efficiently increases day by day.

Most traditional technologies for data scaling involve scaling up. While this model leads to simpler programming paradigms and operational characteristics, it tends to get expensive at the high end, as the cost per unit of processing increases markedly as machines scale up. Scaling horizontally is nothing new, but has always been more a more complex proposition, and not for the timid. With this model, the need to maintain consistency across clusters of machines can be a hindrance with respect to latency and efficiency.

The big breakthrough that many (but not all) of the NoSQL systems offered was to recognize that not all systems required the rigorous ACID consistency adhered to by relational databases, which was responsible for much of the overhead of horizontal scaling. While it's true that an ATM withdrawal needs to guarantee to debit the account and dispense money in a single transaction and update the balance immediately in the system of record, the same isn't true if one is considering a Facebook "Like" operation or a search engine that is being updated with new web page information. Being off by fractions of a second – or even seconds or minutes – often has little to no impact on the overall system, for the right type of system.

The NoSQL systems that emerged to handle large volumes of data did so by compromising ACIDity in order to achieve cheap horizontal scalability. The most common way of spreading data in this way is through sharding, which allows different types of data to be stored on different (one or more) machines inside of a cluster. Whatever the sharding method used by the database, sharding works best when one has a single key associated with a piece of data. Here, the responsibility for making sense of both the key and the attached data lies with the application.

The data sets that lend themselves the most readily to being sharded are those that are simple and atomic. In fact, many NoSQL database, including MongoDB and Cassandra (with the notable and significant exception of graph databases, discussed below), advise against attempting to join or relate data. Users needing to relate data are left building their own join logic in the application. This includes not just relating data at the time it is read, but managing relationships when data items are deleted or modified to avoid broken or dangling relationships.

In his book *NoSQL Distilled,* Martin Fowler makes a useful generalization: grouping the first three members of the NoSQL family (namely key-value stores, column family databases and document databases) together as aggregate-oriented databases, which are distinct from graph databases. In the first three, the unit of operation is atomic and restricted to a single "item" (which can be itself an aggregate). This leads to excellent performance with CRUD (create-read-update-delete) operations for individual data items, but not for data that are inter-connected.

## CONNECTED DATA: ENTER THE GRAPH

One NoSQL technology has emerged to manage connected data and that is the graph database. Graph databases are more and more commonly found in applications where the data model is connected, including social, telecommunications, logistics, master data management, bioinformatics and fraud detection. Graph databases, like the other NoSQL databases, follow a pattern where the data is in effect the schema. In a graph database, individual data items are represented as nodes. Nodes are connected to other nodes through relationships as shown in Figure 1.

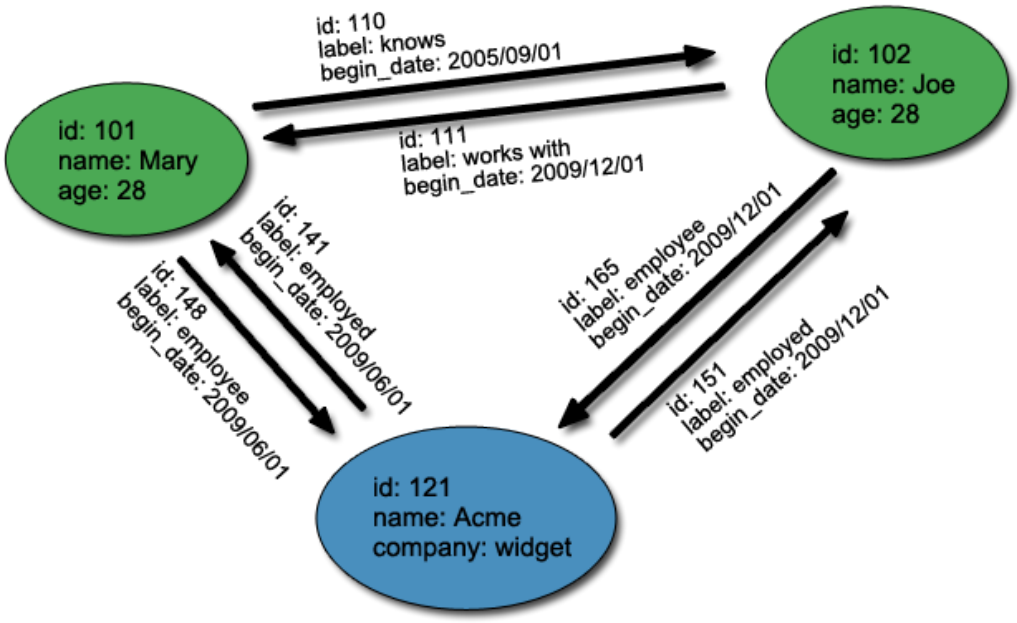


**Figure 1: Semantics of Relationships**

As you can see from Figure 1, relationships have a type and direction, defining the semantics of the relationships. Further, the *property graph* model, which is the one used by the leading commercially available graph database, allows attributes to be associated with nodes as well as with relationships. This model allows for high-fidelity expressiveness for any data set having value in connections. It provides the benefits of structure without the schema rigidity of the relational database, allowing graph databases to share the same agility benefits as are offered by other NoSQL offerings.

Graph databases, unlike their other NoSQL brethren, are often fully ACID, so as to support transactional system-of-record applications. One of the biggest differences between graph databases and both relational and other NoSQL databases is that the connections between nodes (i.e., the relationships) directly link node in such a way that relating data becomes a

simple matter of following relationships. In essence, one tackles the join problem by specifying connections at insert time, so that they can be walked –rather than calculated – at query time. This property, which can be found in those graph databases that support native graph processing, is known as index-free adjacency. It allows queries to traverse literally millions of nodes per second, leading to response times that are several orders of magnitude faster than with relational databases for connected queries such as friend-of-friend and shortest path.

Gartner recently named Strategic Big Data one of the top 10 strategic trends for 2013 and pointed to five specific types of data – all of them graphs – as the "five richest big data sources on the Web:" the social graph, intent graph, consumption graph, interest graph and mobile graph. Add this to the strategic data sources inside one's organization that are inherently graphs, and one becomes surprised at how pervasive graphs are. The good news is that the graph database space has been evolving very rapidly, making 2013 a good year to start looking at graph databases as a technology optimized for leveraging – or better leveraging – these strategic data sets inside of your organization.

## MAKING THE RIGHT CHOICE

There are transactional and analytical processing needs in every business, which leads to technologies optimized for OLTP and OLAP. There are also needs for a multitude of types of "intelligence" when running a business. One involves discrete information about individuals: so-called "atomic" intelligence. The other involves knowledge about the network and relationships: "connected" intelligence. As your business evolves to tackle more volume, velocity, variety and connectedness, chances are that NoSQL will find a place where aggregate data stores (namely key-value, column-family and document stores) can help solve problems related to atomic intelligence and graph databases can markedly improve one's ability to leverage connected intelligence. All this in healthy companionship with one's existing relational database infrastructure.

**About the Authors:** Michael Hunger is the author of numerous articles and books on programming and on graph databases. Philip Rathle has nearly 20 years of experience working as a product manager, consultant and information architect. Both are active in the NoSQL space, at Neo Technology, creator of Neo4j, the world's leading graph database.