Yes or No on NoSQL?

By: Rich Seidner

fac adc

The rise of non-relational database technologies has engaged a growing IT audience. In order to understand what's gained when employing non-SQL approaches, we spoke with renowned database expert, David McGoveran, who has consulted on bleeding-edge distributed applications and database solutions since 1976 when he founded Alternative Technologies.

Q: How does NoSQL fit in the universe of database approaches?

David McGoveran: "NoSQL" means different things to different people. Early on, it was portrayed as "not SQL." More recently, that's been tempered somewhat and is considered "not only SQL." The big picture that holds this community together is a rebellion against the RDBMS [relational database management system] solution providers. RDBMS solutions are often expensive to purchase and maintain. They have lots of features for which most developers will never see a use, a kind of commercial software code bloat. And there's a difference between a pure programmer's IT perspective (i.e., rapid solution delivery) versus the data modeler's (i.e., long-term asset creation).

The NoSQL community can be seen as an outgrowth of the anti-relational movement that came about with object-oriented programming. They share lots of the same issues and concerns.

Q: Please explain one of those issues.

D.M.: One issue has to do with the level of data organization. An RDBMS contains facilities for managing data, whereas a database is just an organized collection of data or, more generally speaking, an arbitrary container for data called a data store. With a data store, you're dealing with low-level physical access and implementation (e.g., a file system). A fair number of the NoSQL solutions are at the data-store level. Hadoop, for instance, is focused on the functionality necessary to distribute data of lots of types and offers high availability. File organization is not as important in Hadoop as is data distribution and access over a large number of nodes. The NoSQL community is concerned with being able to cheaply (e.g., with little up-front design effort) scale up to thousands of servers and nodes, perhaps geographically distributed, or thousands of control processors, often maintaining some partitioning or "sharding" of the data. Support for MapReduce algorithms is often considered essential, but its proper use can be tricky.

Q: Are all NoSQL solutions data stores?

D.M.: No. There are a variety of DBMS solutions in the NoSQL camp, but the one thing they won't embrace is SQL, about which NoSQL is pretty rigid. NoSQL insists that "one size does *not* fit all," meaning there's no one approach to data management. Some DBMS solutions that would be classed as NoSQL allow programmers to access the data stored in a way that will best meet their application needs. Their physical organizations are application-specific. Among them are graph (linked) databases, text databases, columnar databases, stream-oriented databases, and key-value (aka associative) databases.

Q: From an application perspective, what kind of application should use NoSQL?

D.M.: Consider a NoSQL solution if your application involves collecting masses of data, and you can't develop a data model for it, or if you can't physically organize the data and develop a logical schema and metadata before you start to use it. Often the data is organized in non-uniform ways or for human consumption rather than computer processing. Eventual data copy consistency is often more important than insuring the data satisfies business rules and relationships. Social media is a good example. In such applications, you want to find data or mine relationships -- consider Amazon, LinkedIn, Google, eBay, Netflix, etc. -- in ways that are not traditional data-management problems.

Q: From an architectural perspective, what kind of application should use NoSQL?

D.M.: The best architecture for high availability and scalability depends heavily on product capabilities and intended use. If application objectives change too much over time, your implementation might break. It's inherently brittle and requires upfront investment. For relatively stable IT environments, RDBMS solutions work great, because those products were designed to meet the needs of availability, scalability and reliability in such applications. But when you don't have a lot of predictability about how the database is going to change and grow, the kinds of data, how many nodes you're going to have, what types of nodes, nor how many processors you'll need … all of a sudden, a relatively low-level simple solution that's open-source and low-cost appears very attractive to a developer who is being hit over the head every morning for not having delivered a solution.

Q: How should an enterprise think about using NoSQL?

D.M.: I suggest considering a NoSQL solution if any of the following are true:

- First, when discovery of relationships is more important than consistent processing and specific data results.
- Second, if the data processing is meant to be inductive (e.g., suggestive) rather than deductive (i.e., precise).
- Third, when the application is changing very fast, data complexity is great (variety or amount).
- Fourth, if physical issues, like big data or a high degree of parallelism, are more crucial than data integrity. You must be willing to throw away data consistency in favor of performance and scalability.
- Fifth, if you have a mission-critical one-off application for which a fixed data organization is ideal, in which case the costs and risks may be lower than licensing a vendor's RDBMS or trying to force an open-source RDBMS to fit the need.

fac adc

Rich Seidner *is a technology veteran, an independent researcher, a writer and an editor serving technology clients. Based in Woodside, Calif., his blood type is "technology."*

« PREVIOUS FEATURE NEXT FEATURE »

Go to Homepage

IT SOFTWARE STRATEGY | S/W DEV TOOLS & TIPS | TECH TRENDS | EXPERT INSIGHT

Terms of Use - Privacy - Contact Us - About Us - Disclaimer - Leave Feedback