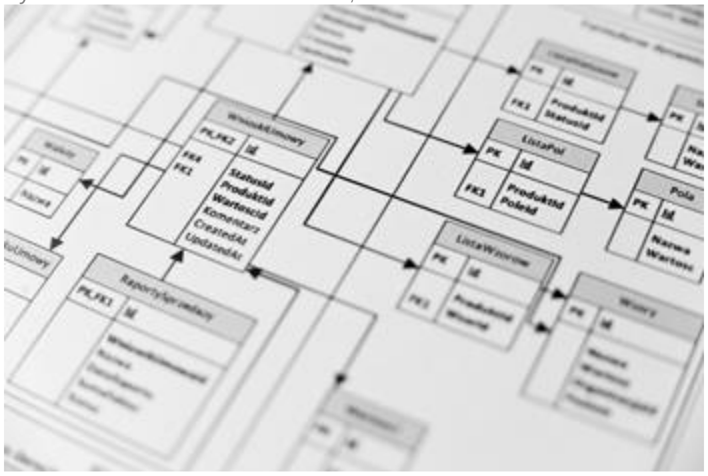
A Review of Different Database Types: Relational versus Non-Relational

By Keith D. Foote / December 21, 2016 / 0 Comments



Relational databases are also called Relational Database Management Systems (RDBMS) or SQL databases. Historically, the most popular of these have been Microsoft SQL Server, Oracle Database, MySQL, and IBM DB2. The RDBMS's are used mostly in large enterprise scenarios, with the exception of MySQL, which is also used to store data for Web applications.

All relational databases can be used to manage transaction-oriented applications (OLTP), and most non-relational databases, in the categories of Document Stores and Column Stores, can also be used for OLTP, adding to the confusion between them. OLTP databases can be thought of as "operational" databases, characterized by frequent, short transactions that include updates, touch a small amount of data, and provide concurrency to thousands (if not more) of transactions (some examples include banking applications and online reservations).

James Serra, a Big Data Evangelist at Microsoft, discussed the many differences, advantages and disadvantages, and various use cases of relational and non-relational databases during his Enterprise Data World 2016 Conference presentation.

He began by discussing the fact that the integrity of data is very important, so RDBMSs support ACID transactions (Atomicity, Consistency, Isolation, and Durability). RDBMSs have provided for data integrity needs for decades, but the exponential growth of data over the past 10 years or so, along with many new data types have changed the data equation entirely, and so non-relational databases have grown from such a need.

Non-relational databases are also called NoSQL databases. NoSQL has become an industry standard term, but the name is beginning to lose popularity since it doesn't fully cover the complexity and range of non-relational data stores that are available. Some of the most known NoSQL or non-relational DBs that Serra discussed are MongoDB, DocumentDB, Cassandra, Coachbase, HBase, Redis, and Neo4j. There are literally hundreds, if not thousands, more.

Hadoop is also part of this entire discussion, said Serra. But, “keep in mind Hadoop is a file system with components made up of Hadoop Distributed File System (HDFS), Yarn, and MapReduce.” So while it is a significant part of the relational and non-relational discussion, it includes many other components as well. For an outline of Hadoop, see the DATAVERSITY® article titled Hadoop Overview: A Big Data Toolkit.

The Origins

If an organization is using SQL Server, said Serra,

“And I need to index a few thousand documents and search them. No problem. I can use Full-Text Search. But what happens if I need to store and analyze a few million web pages?”

Enter Hadoop and non-relational databases. Using SQL Server, if an internal company application needs to handle a few thousand transactions per second it’s no problem. SQL Server can handle that with a nice size server. But in a situation where users can enter millions of transactions per second, this becomes a serious problem. Enter NoSQL as a solution, said Serra. But most enterprise data still only needs an RDBMS.

Main Differences Between Relational and Non-Relational Databases

In his presentation, Serra listed multiple slides (see the presentation video at the end of this article) that detail the many variances in databases, including pros and cons. A short list of the most fundamental elements discussed by Serra includes:

Relational Databases

Pros

- Relational databases work with structured data.
- They support ACID transactional consistency and support “joins.”
- They come with built-in data integrity and a large eco-system.
- Relationships in this system have constraints.
- There is limitless indexing. Strong SQL.

Cons

- Relational Databases do not scale out horizontally very well (concurrency and data size), only vertically, (unless you use sharding).
- Data is normalized, meaning lots of joins, which affects speed.
- They have problems working with semi-structured data.

Non-relational/NoSQL

Pros

- They scale out horizontally and work with unstructured and semi-structured data. Some support ACID transactional consistency.
- Schema-free or Schema-on-read options.
- High availability.

- While many NoSQL databases are open source and so “free”, there are often considerable training, setup, and developments costs. There are now also numerous commercial products available.

Cons

- Weaker or eventual consistency (BASE) instead of ACID.
- Limited support for joins.
- Data is denormalized, requiring mass updates (i.e. product name change).
- Does not have built-in data integrity (must do in code).
- Limited indexing.

Non-Relational Stores in Short

There are many different kinds of non-relational stores; Serra gave an overview of the main types. In today’s market the numerous commercial offerings have created a number of platforms that actually combine different data models into one system. According to Serra, Key-Value Stores offer very high speed via the least complicated data model. Anything can be stored as a value, as long as each value is associated with a key or name.

“Wide-Column Stores are fast and can be nearly as simple as Key-Value Stores,” he remarked. They include a primary key, an optional secondary key, and anything stored as a value.

Document Stores contain data objects that are inherently hierarchical, tree-like structures (most notably JSON or XML). Word documents are not Document Stores, he joked. This is a naming confusion that non-data people sometime make. “It is way of storing all the data in one structure. All information can be stored in one document,” said Serra.

He also touched on Graph Stores, remarking that “Graph Stores are totally different from what we’ve talked about so far. They are not typically scalable, but do have some great use cases and they are really good for storing relationships.”

Use Cases for NoSQL Categories

Serra discussed a number of different non-relational use cases as well during his presentation, a few of these mentioned were:

- **Key-Value Stores:** [Redis] Used for cache, queues fit in memory, rapidly changing data, and store blob data. Examples: sensor data, shopping cart, leaderboards, graph data bases, stock prices. Fastest “performance.”
- **Document Stores:** [MongoDB] Have a flexible schemas, dynamic queries, defined indexes, good performance on big DB. Examples: order data, customer data, log data, product catalog, user generated content (chat sessions, tweets, blog posts, ratings, comments). Fastest development.

- **Wide-Column Stores:** [Cassandra] Come with real-time querying of random (non-sequential) data, huge number of writes, sensors. Examples: Web analytics, time series analytics, real-time data analysis, banking industry.

"You may not have the data volume for NoSQL," said Serra. "But there are other reasons to use NoSQL. Such examples include storing semi-structured data, schema-less data models, and a need for high availability data."

NewSQL

Serra then discussed what he calls NewSQL, or a mixing of the various data models into what amounts to a Relational + NoSQL Store. They are designed for Web-scale applications, but still require up-front schemas, joins, and table management that can be labor intensive. They are effectively an effort to make the data scalable and still provide many of the traditional SQL operations.

Uses for Different Database Technologies

Serra also talked about many of the reasons why an organization would use SQL or NoSQL. He said that for traditional OLTP business systems (i.e. ERP, CRM, In-house app) relational databases (RDBMS) are still the primary and most efficient choice. Other choices he discussed were:

- Data warehouses (OLAP) are good for relational database (SMP or MPP).
- Web and mobile global OLTP applications work well with non-relational database (NoSQL).
- Data lakes are good for Hadoop.
- Relational and scalable OLTP would work well with NewSQL.

Summary When to Choose NoSQL

Serra ended his presentation with an outline of when and where an enterprise would want to choose a non-relational or NoSQL system over a more traditional relational platform. Some of those reasons include:

- When bringing in new data with a lot of volume and/or variety.
- Data is non-relational/semi-structured.
- Your team will be trained in these new technologies (NoSQL).
- You have enough information to correctly select the type and product of NoSQL for your situation.
- You can relax transactional consistency when scalability or performance is more important.
- You can service a large number of user requests vs rigorously enforcing business rules.

He closed by saying that "RDBMS is for enterprise OLTP and ACID compliance, or databases under 1 terabyte. NoSQL is for scaled OLTP and JSON documents. Hadoop is for Big Data Analytics." The choices on the market today are numerous, but so are the needs of different enterprises. How (and when) to choose the right database system is something that every enterprise must now contend with to maintain marketplace advantages.