Introduction

NoSQL – purpose-built data management

NoSQL systems are purpose-built solutions, designed to address specific technical requirements. NoSQL systems originated to provide high throughput, fault-tolerant horizontally scalable simple data storage and retrieval with a bare minimum of additional functionality. Specifically, NoSQL systems were created in order to provide:

- Horizontally distributed data management of simple structured and unstructured data across a large cluster of commodity storage systems,
- Highly fault-tolerant data management and ability to continue operating even after multiple hardware and system failures,
- Very high throughput for simple read/write operations, with limited or no transaction semantics,
- Schema-less or flexible schema definitions allowing highly variable data and record structures,
- Application and application developer-centric special purpose data models and APIs.

On the other hand, RDBMS systems, like Oracle Database, are designed to provide general purpose data management capabilities and standard APIs for a very wide variety of requirements. Hence, they incorporate a lot of features and functionality.

Not all applications require the full set of RDBMS functionality. If the application doesn't require all of the functionality in a typical RDBMS, why would the customer pay for the hundreds of features that they don't need? For example, web-centric customer service, loyalty card programs and customer profile management applications primarily require fast, scalable key-based access to data. In such scenarios, a cost-effective, purpose-built NoSQL database is an attractive alternative to a relational system. Enterprises, ISVs and SIs are *actively* identifying applications and data management processes which can be implemented and managed more effectively using special-purpose NoSQL systems.

NoSQL Application Example – simple data, simple queries

Customer Profile Management

Let's look at a specific example of an application that matches the capabilities of a NoSQL system – customer profile management. In the past, customer profiles were typically financial transaction-oriented data structures/repositories. Today, a customer profile includes a much richer data set that includes information from a wide variety of customer interaction points that include both structured and unstructured data. Capturing and managing this new class of data is crucial to the enterprise, in order to more effectively obtain a 360 degree view of a customer and to optimize customer interactions. This information enables a broad category of Line-of-Business applications from Marketing, Advertising, Customer Service, Risk Analysis, Fraud Detection, Personalized content, Promotional Campaigns, Loyalty Programs, Inventory Management, etc. These Line-of-Business applications leverage the richer user-profile data in order to provide

a. More personalized customer experiences via targeted product offerings, special promotions, loyalty rewards, more informed/context sensitive interactions, etc.,

b. Better operational insight into how their customers interact with them, how they perceive the company and its products and services, with a longer and more complete historical perspective,
c. Better competitive insight into how customers perceive their competition,
d. Better operational decision making, based on a more detailed understanding of their customers.

Modern customer profile management applications benefit directly from the capabilities of NoSQL systems because they provide

a. A horizontally scalable, distributed data management system to manage the large volume of data that is part of a rich customer profile and that can grow with little to no maintenance as the customer base grows,
b. A highly fault tolerant system, ensuring that the customer profile data is always available to the applications that need to access and update it,
c. A flexible or no-schema data store, which facilitates a wide variety of data record formats to be stored in a given customer profile and for those record formats to change over time,
d. Specialized, high throughput application-specific read and write access to the portions of the customer profile that are important for that particular application[1].

A great example of this is our recent work with well known Entertainment company on their Next Generation Experience (NGE) application. This company wants to create a centralized repository for each of its customer's experience across all of its entertainment properties (theme parks, restaurants, cruises, merchandizing outlets, etc.) over a span of decades. The value of this solution is that a customer's visits as a child, then as a parent, then as a grandparent are all available in the system, regardless of which (other) system the interaction originated from. They want the customer profile repository to contain both structured and unstructured data, with a schema-flexible format so that it can evolve over time. Each customer's profile will include URL-style pointers to records of activities at the various properties, as well as photos, privileges, loyalty program links, associated groups, other related customers, customer feedback, etc. For this company the concept of "a customer profile" includes many different sources and types of data, over an extended period of time, resulting in a very large and varied set of data – centralized and served from one data management infrastructure. This customer profile will be used to drive existing and future applications that provide the customer with a personalized, enhanced experience, utilizing both real-time and Business Intelligence Advanced Analytics access to the customer profile. Originally this centralized repository was designed to be managed in a relational database. However, last year they decided to re-design the NGE repository to use a NoSQL database. They made this decision because

a. the NGE customer profile repository primarily required simple queries, simple data, flexible schemas and horizontally distributed storage, which was more efficiently addressed using a special purpose NoSQL database,
b. of specific application developer technical synergies with NoSQL technology. In particular, their applications didn't manage data using SQL nor did it represent the data as SQL tables. From the application point of view, records were just JSON objects consisting of simple,

[1] It is important to note that these applications often require support for transactions and concurrency. Not all NoSQL databases provide this functionality. Oracle NoSQL Database does. For more information see section on Oracle NoSQL Database on page 9.

> but variable data elements and pointers to other records. Their natural application-developer-centric technical inclination was to eschew using SQL and focus on a simpler, more special purpose database and API that was closer to their application's view of the data.

They are now also looking at a NoSQL Database. This is great example of a common use case for a NoSQL Database.

RDBMS vs NoSQL – The right tool for the right job

Sophisticated RDBMS systems, for example the Oracle Database, encompass a very rich set of features and functionality, primarily focused on general purpose OLTP and Data Warehousing use by many different types of applications. NoSQL systems, on the other hand, encompass a very limited set of features and functionality while providing horizontal scalability, availability and data modeling flexibility for specific applications that manage simple data and simple queries.

NoSQL systems typically include a set of software packages running across dozens, hundreds or even thousands of smaller systems. Efficient horizontal scalability, high availability and concurrent data processing are provided out of the box, but the integration effort to weave the software components into an integrated solution is left up to the customer. Many of the advanced features that are common with mature RDBMS systems are not present in NoSQL systems. This *requires* NoSQL technology users to integrate their NoSQL data with the RDBMS systems in order to make use of the advanced features that they need. NoSQL systems can do lots of specific-purpose, simple operations extremely quickly, but are not designed with the features to perform complex, general purpose operations in a integrated way.

Let's look at an example of how this plays out in terms of understanding what types of application characteristics are best suited for NoSQL and which ones are not.

Simple Data, Simple Queries

Let's take the example of customer profile management discussed earlier. The functionality that is required is very simple – the application needs to read and write a few records based on the primary key -- customer ID. These records combine structured and unstructured data, stored in a way (often de-normalized) that allows the application to only operate on the subset of data that it needs for that specific type of transaction. Customer profile management applications typically have to perform an extremely high number of these simple operations of lookup based on a customer ID with minimal latency. The latency component is critical because these applications often implement the "last mile" interface with the customer. Customers and other downstream systems interact directly with the content managed in the NoSQL Database, and real-world studies have shown that any increase in latency can be linked directly to reduced revenue and loss of business.[2]

[2] Amazon.com conducted an extensive study that demonstrated a direct relationship between increased latency and loss of revenue. For every 100ms (that's 1/10th of a second) of increased latency on their web site, they observed a decrease of 1% in revenue. In other words, an almost imperceptible increase in latency translated directly into lost business because customers stopped browsing or using the site.

These "last mile" applications, based on rich customer profile management, are a perfect fit for NoSQL databases. An RDBMS can provide the same level of functionality and throughput, but they may not fit the technical requirements as closely or as efficiently as a special purpose NoSQL database . This example can be extended to sensor data (machine profiles, rather than customer profiles), loyalty card programs, financial data, product data, etc. The bottom line is that for these kind of operations (simple queries over simple data), NoSQL databases can be more efficient than RDBMS systems.

Simple Joins

Let's imagine a slightly more complex requirement, where the customer profile application needs to perform multiple lookups in a small set of related tables (joins). For example, a customer profile is likely to include a list of products (stored as product IDs) that the customer has "liked" or "disliked" in the past.

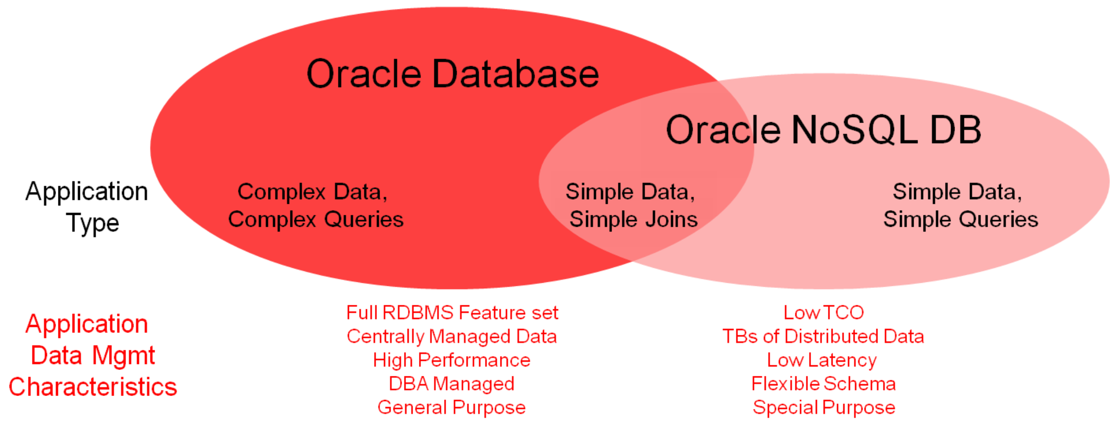
In this case, it's relatively easy to express the SQL JOIN queries in an RDBMS (joining the Customer Profile table with Product table) and have the query optimizer do the right thing to optimize and execute the joins. In a NoSQL database, it's up to the application (and the application developer) to implement the appropriate primary and foreign key lookups in the most efficient order as separate operations in order to satisfy the query. In a NoSQL database, the application logic would have to contain code to perform a lookup of the Product table for each product ID in the customer profile. Both the RDBMS and the NoSQL database *could* be used support this kind of application requirement – the RDBMS is easier to implement, while the NoSQL database may perform the simple queries more efficiently but some of the functionality would have to be implemented within the application logic (the application developer has to implement the joins using client-side code to perform the appropriate lookups). So, why wouldn't a developer choose the "easier" option of using an RDBMS for this kind of application? Primarily for three reasons:

a. Simple joins are basically just incremental key lookups, sometimes described as client-side joins. These simple joins are easily implemented in application code
b. When scalability and performance are the key requirement, simple special-purpose data stores like a NoSQL Database offer an option that is better performing at a lower cost of operation,
c. Over the past decade application development has changed. Even within established RDBMS shops and especially in new/innovative technology companies, the development of web-scale applications and new customer-oriented services has moved towards Java-based application development. Developers are much more accustomed to building data management and access to specialized, high performance, high scalability databases such as Key-Value based NoSQL databases in order to leverage the advantages that these products provide, even if it requires some additional application development effort. It's a "religious" discussion!

Complex Queries

Finally, let's imagine that the application requires more than simple queries or simple joins. The application may need to perform complex, multi-table joins[3], to perform complex analytics; it may require fixed schemas in order to enforce referential integrity or other business rules; it may require heightened data security, data lifecycle management or other advanced features that are commonly found in an RDBMS. In this case, the clear choice is to use an RDBMS-based solution. A NoSQL technology based solution would require significant functionality that would have to be implemented in the application. In fact, this is the primary argument in favor of staying with an RDBMS, rather than switching to a NoSQL database. If the application needs the rich functionality of the an RDBMS, don't try moving to a platform that doesn't have (nor will it ever have) that functionality.

The above characterization of use cases reflects what we're seeing in our customer base – complex application functionality continues to be implemented using the Oracle Database. However, in application scenarios where simple functionality, horizontal scalability and schema flexibility are the primary requirements, these applications are being implemented in NoSQL databases. The fact is that a full customer solution typically includes ***both*** types of operations, and therefore ***both*** types of databases. Customers are actively engaged in identifying the right database management tool for the right job.



They are undertaking the necessary steps to adopt and integrate NoSQL databases as part of an overall solution, where that tool makes sense. When talking with customers it's critical to recognize the technical challenges that are being addressed, that one size does not fit all and how/when to use

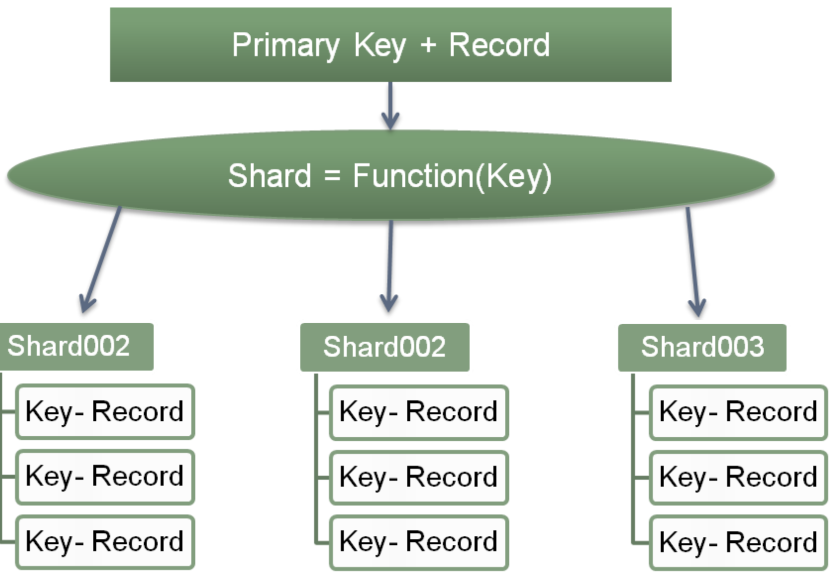
[3] An example of a complex multi-table join: In an effort to reduce inventory overhead, a retail manager wants to identify low margin items that are not selling well in the northwest region. They might ask "Give me the Supplier name, Item Number and Item Description from the Supplier and Inventory tables where the Price/Cost is less than 10% and the QuantityOnHand is greater than 20 and the total sales for the last 6 months in the NW region is less than $1000 from the Sales and Inventory tables." This is relatively easy to express in SQL and let the SQL optimizer and query planner figure out how to get the data. But in a NoSQL application it would require writing multiple queries and application code to collect and process the results of those queries in order to produce the same result.

What is NoSQL?

NoSQL (commonly interpreted as "not only SQL") is a broad class of database management systems developed over the last decade that is primarily identified as not adhering to the widely used relational database management system model. Although it is inherently difficult to define a technology based on what it *isn't*, and although NoSQL systems vary significantly in implementation, features and behavior, there are common design principles and technology requirements that NoSQL databases share in common.

NoSQL systems

NoSQL systems provide horizontal scaling across a very large numbers of servers (tens, hundreds or even thousands of servers) by using a technique that has been deployed for many years in conventional RDBMS databases, called sharding. Sharding requires that a separate database run on each server and that the data be physically partitioned so that each database has its own subset of the data stored on its own local disks. NoSQL systems maximize throughput by limiting how the sharded data is managed and accessed. NoSQL systems typically do not provide support for operations that require accessing multiple shards of data – this includes joins, distributed transactions and coordinated/synchronized schema changes – because of the I/O overhead and coordination required. NoSQL systems typically implement limited transaction capability, either by relaxing transaction consistency (by using "eventual consistency"), providing shard-local ACID consistency only or by disallowing transactions altogether.



The primary use case for NoSQL systems is horizontally distributed, sharded data sets with a flexible schema, and simple read/write operations. More complex, general purpose functionality is not considered to be the primary goal of NoSQL databases.

Oracle NoSQL Database – Value Proposition

The NoSQL Database is a highly scalable, high performance, high availability distributed Key-Value database for near-real-time access to data. Like other NoSQL databases, it is focused on providing horizontally scalable, simple data management functionality. It differentiates itself from other NoSQL databases because:

- Oracle NoSQL Database is based on mature, field-proven, production quality software. Oracle NoSQL Database utilizes Berkeley DB Java Edition as the basis for its storage and replication technology. Berkeley DB Java Edition has been powering mission-critical applications in the field for almost a decade, including companies like Amazon.com, LinkedIn, Yammer, Sand Vine and Macys.com. Many of the other NoSQL databases are either creating their own storage and replication layers from scratch (and finding out how difficult that is). Others are adopting existing open source storage and replication technology (which may not be well suited to their specific requirements).
- Oracle NoSQL Database is the *only* NoSQL database developed and supported by a major database vendor.
- Oracle NoSQL Database is integrated with other related products like Oracle Database, Oracle Business Intelligence, Event processing, Oracle Spatial, RDF & Graph, Oracle Coherence and Hadoop/MapReduce . Other NoSQL databases require that the customer figure out how to implement integration with their other IT assets. Most Big Data projects require multiple, complementary data management technologies and Oracle is the only vendor that has a comprehensive offering of integrated technologies.
- Oracle NoSQL Database has unique features like configurable ACID transactions, and Dynamic Storage Node rebalancing, that many of the other NoSQL systems lack. ACID transactions make application development much simpler and Dynamic Storage Node rebalancing ensures robust, consistent and scalable database deployment. Additionally, the Oracle NoSQL Database is not only proven to deliver high throughput, but also guarantees *predictable* throughput and latency through automatic, highly tuned database cache eviction policies and Java garbage collection parameters. Other NoSQL databases often have unexpected limitations when it comes to performance management and predictability, specifically related to file system cleanup, compaction and Java garbage collection. A report from Amazon based on actual studies showed that 100ms (that's 1/10 of a second!) of **added latency** in accessing a page from the browser caused a 1% **decrease** in revenue!
- Oracle NoSQL Database is much easier to deploy and simpler to manage[4].
- Oracle NoSQL Database has been extensively benchmarked using the industry standard YCSB benchmark[5], that has conclusively demonstrated a) scalability to hundreds of storage nodes, b) across 10s of TB of data, c) 1.25 *million* operations per second.[6]

Oracle NoSQL Database is Oracle's flagship NoSQL database product. Sales can confidently recommend the Oracle NoSQL Database, alongside the Oracle Database, in order to address the needs of cost effective extreme data scalability and flexibility.

---

[4] Oracle NoSQL DB Quickstart Guide; http://docs.oracle.com/cd/NOSQL/html/quickstart.html

[5] The YCSB benchmark is the de-facto standard benchmark used for NoSQL databases. More information can be found here: http://research.yahoo.com/Web_Information_Management/YCSB

[6] Oracle NoSQL Database benchmark links:

https://blogs.oracle.com/charlesLamb/entry/oracle_nosql_database_exceeds_1

https://blogs.oracle.com/charlesLamb/entry/oracle_nosql_database_performance_tests

Other NoSQL databases

There are over a hundred different NoSQL databases currently being offered on the market. Customers who become early adopters of open source NoSQL databases often invest in programming staffs that participate in the development and maintenance of these products. There are currently no standards in the NoSQL technology space, so each NoSQL product is different in terms of features, behavior and implementation. It is not within the scope of this paper to do a full competitive analysis of all of the alternative NoSQL databases that are available. For product specific competitive comparisons, see the competition section of the Oracle NoSQL Database OTN site.

The value proposition of Oracle NoSQL database is discussed in the previous section. Oracle NoSQL Database as **the best integrated, highest quality NoSQL database** from the **best of breed purveyor of enterprise-class supported, industrial grade database software**.

NoSQL Database and RBBMS Database

The NoSQL Databases and the RDBMS Database complement each other. Each solves a different type of requirement.

The NoSQL Database is designed to cost-effectively manage large volumes of simple, structured and unstructured data. However, it is often the case that important subsets of that data need to be loaded into the RDBMS Database in order to access more advanced capabilities like complex queries, data security, data lifecycle management and Advanced Analytics. Typically the same ETL-class tools that support loading Hadoop data into RBDMS systems are also used for loading NoSQL data into an RDBMS. The Oracle NoSQL Database supports the Oracle Big Data Connectors (ODI and OLH), as well as direct access to its data via Oracle Database External Tables, allowing customers to combine relational and NoSQL data in the same query.

Just as Oracle NoSQL Database data may be moved into Oracle Database for analytical functions, data stored in the Oracle Database may be moved into Oracle NoSQL Database in order to enable high volume, high velocity web-based applications. A typical example of this is moving certain aspects of customer profile data and inventory information into the NoSQL Database in order to drive a web-based consumer retail application (e.g. loyalty card programs).

As discussed earlier, customers may choose to replace their RDBMS Database with Oracle NoSQL Database, but for only the subset of data and functionality that are best suited for the NoSQL technology approach. Most customers will use a combination of an RDBMS and Oracle NoSQL Database to address their overall data management requirements.

Where do NoSQL Database projects get started?

Although a NoSQL database project may start in almost any technical group, our experience indicates that there are a few common ways that NoSQL projects get started.

NoSQL technology is typically evaluated in the context of *Big Data* data management infrastructure and can get started in almost any technical group within a customer's organization. It is often the case that these projects and opportunities are being driven by application developers. When discussing application areas for a NoSQL Database or when introducing the technology to customers, it is crucial to include Application Architects and Developers in the technology discussion. It is often within these groups that NoSQL technology evaluation and research is occurring.

Customers are constantly looking for opportunities to do more (increase business value, deliver new services, derive new insights) with less (using commodity hardware, open source software). The potential value and capabilities of Big Data hold a lot of promise and customers are actively exploring ways to leverage Big Data and/or NoSQL technologies. This technical exploration often leads to different kinds of Big Data or NoSQL initiatives within the company. These initiatives fall under some broad categories:

- **New initiatives (experiments) to leverage Big Data**: For example, some customers are leveraging social media data from Facebook and Twitter for example, to provide a more comprehensive view of their customers. They are combining these Big Data insights with their NoSQL customer profile applications.
- **New applications using existing data**: Projects to take existing data and reuse it in new ways. For example, an entertainment company might leverage customer visits and interactions (e.g. hotel bookings, restaurant and car reservations, retail activities, etc.) from existing systems and repurpose that data into a NoSQL repository to provide a personalized experience for visitors.
- **Cost effective data management choices**: Use NoSQL technologies to provide efficient special purpose data management for existing applications that do not require the full capabilities of an RDBMS. For example, a large electricity utility company is planning to manage several terabytes of metering data in a special purpose NoSQL repository because the application requirements, data structures and queries are very simple.
- **Private cloud services**: Several large customers have initiatives to build an in-house data services platform that includes RDBMS, NoSQL, MapReduce and other related technologies. By combining the three technologies together into a service platform, customers can harness the rich feature set of an RDBMS, plus the distributed, scalable, simple storage and processing of NoSQL (for interactive data usage) and Hadoop/MapReduce (for batch usage).

Conclusion

NoSQL database technology is here to stay. It addresses specific technical requirements that are not as efficiently or cost effectively addressed with other data management technologies, including RDBMS systems. The RDBMS and Oracle NoSQL Database are *complementary* in the sense that they work together to address the overall data management needs of our customers, each providing the technical capabilities required by today's complex and evolving applications. Customers can rely on the quality, integration and support of the Oracle Database and Oracle NoSQL Database, and deploy their applications with confidence, maximizing their technology investment and minimizing their risk; as opposed to opting for other NoSQL database products with unknown scalability, quality and integration challenges.