International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2013): 6.14 | Impact Factor (2015): 6.391

NoSQL in Action - A New Pathway to Database

Priya Badlani

Vivekanand Education Society's Institute of Technology,
Hashu Advani Memorial Complex, Collector Colony, Chembur East, Mumbai, Maharashtra 400074

Abstract: ***With the uninterrupted growth, databases are growing very fast and becoming more complex in the Volume, Variety and Velocity and hence management of database has been the biggest challenge. The data collection is currently managed and exploited mostly by using conventional data management tools such Relational Database Management Systems(RDMS), profoundly has its accurate name as SQL or conventional search engines. In the last two decades, the relational databases proved to be powerful and superior because of its highly semantic features and usage. Soon as Big Data stepped into the IT industry, handling large volume of data and variety in data type and structure has lead to a tedious job. To handle such Big Data, relational databases are not suitable because of its strict data constraints, structure, relations and so on. Hence, data aggregation becomes impossible. NoSQL databases provide an efficient and clear framework to aggregate large volumes of data, structures and relations. Now, to stipulate the problem, the modeling and migration of data is required. Currently, there is no tool for the migration of Relational databases to NoSQL databases. This migration requires conversion of relational (sql) database query to NoSQL database query. Relational databases (RDBMS) have struggled to keep up with the wave of modernization, leading to the rise of NoSQL as the most viable database. The aim of this paper is to highlight and evaluate the ways in which NoSQL can be used, Data Modeling concept in NoSQL, migration process in NoSQL database.***

Keywords: NoSQL, Roadmap to NoSQL, Data Modeling, Data Migration ETL Approach, Data Loading, Data Extraction.

1. Introduction

1.1 What is NoSQL

NoSQL, which stands for Not Only SQL, is a mechanism for storage and retrieval of data which is modeled in means other than the tabular relations used in relational databases. The easiest way to think of NoSQL, is that of a database which does not adhering to the traditional relational database management system (RDMS) structure. It does not employ SQL to manipulate data. It may not provide full ACID (atomicity, consistency, isolation, durability) guarantees, but still has a distributed and fault tolerant architecture. NoSQL technology was originally created and used by Internet leaders such as Facebook, Google, Amazon, and others who required database management systems that could write and read data anywhere in the world, while scaling and delivering performance across massive data sets and millions of users.

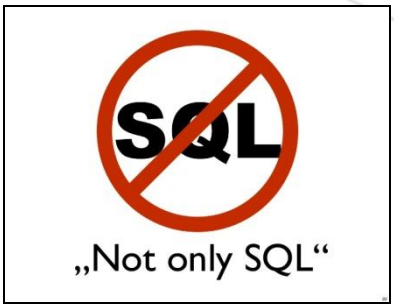

Figure 1: Symbolic Representation of NoSQL

1.2 Roadmap to NoSQL

NoSQL is a database technology designed to support the requirements of cloud applications and to overcome the scale, performance,data model, and data distribution limitations of relational databases(RDBMS's).Today,almost every company and organization has to deliver cloud applications that personalize their customer's experience with their business,with NoSQL being the database technology of choice for powering such systems.

1.2.1 NoSQL Data Management for Big Data needs

The first reason to use NoSQL is because you have big data projects to tackle. A big data project is normally recognized by:

- **Data velocity**-lots of data coming in very quickly, possibly from different locations.
- **Data variety**-storage of data that is structured, semi-structured, and unstructured.
- **Data volume**-data that involves many terabytes or petabytes in size.
- **Data complexity**-data that is stored and managed in different locales, data centers, or cloud geo-zones.

As with any emerging yet complex application development framework, successful implementation relies on an ecosystem of different components that can be combined to address the development of the appropriate solution. For big data, that ecosystem revolves around some key architectural artifacts, including scalable storage, parallel computing, and data management paradigms that are united through an application development platform. And while it is important to review the ways these different components are blended together, from the perspective of application development, the interdependence between analytic algorithms and the underlying data management framework warrants a more in-depth review of big data storage paradigms.

The reason is straightforward: when applications already depend on a traditional relational database (RBDMS) model and/or a data warehousing approach to data management, it may be sufficient to port the RDBMS tools as a way of scaling performance on a big data appliance. However, many algorithms that expect to take advantage of a high-performance, elastic, distributed data environment are not

suited to consume data in traditional RDBMS systems. That means developers must consider different methods for data management.

These analytic algorithms can employ one of an array of alternative means for data management that are typically bundled under the term "NoSQL databases." The term "NoSQL" conveys two different concepts. The first suggests a data management framework that is **not** a SQL-compliant one. The second more generally acknowledged (that is, more frequently presented) meaning is that the term stands for "Not only SQL," suggesting environments that combine traditional SQL (or SQL-like query languages) with alternative means of querying and access.

1.2.2 NoSQL for Continuous Availability

IBM divides database availability into three sections:

1) **High Availability:** database and application is available in scheduled period, when maintenance period system is temporarily down.
2) **Continuous Operation:** system available all the time with no scheduled outages.
3) **Continuous Availability:** combination of HA and CO, data is always available, and maintenance is done without shutdown the system.

This is the reason to consider a NoSQL solution is that your applications need to be continuously available. Note that this is different from just "high availability", where unplanned downtime, although not desired, is still expected. Continuous availability describes a feature of systems that can't go down. In today's marketplace, where the competition is just a click away, downtime can be deadly to a company's bottom line and reputation. Average downtime costs vary considerably across industries, from approximately $90,000 per hour in the media sector to about $6.48 million per hour for large online brokerages.

1.2.3 NoSQL for Data Location Independence

A third motivation to use NoSQL is that you need true location independence with a database. The term "location independence" practically means the ability to read and write to a database regardless of where that I/O operation physically occurs, and to have any write functionality propagated out from that location, so that it's available to users and machines at other sites.

Such functionality is easy to articulate, but difficult to architect for most traditional databases. Master/slave and normally shared architectures can sometimes meet the need for location independent read operations, but writing data everywhere is a different matter.

The reasons for needing location independence are many and include servicing customers in many different geographies and needing to keep data local at those sites for fast access.

1.2.4 NoSQL for More Flexible Data Model:

One of the major reasons IT profeesionals move to a NoSQL database from a legacy RDBMS is the more flexible data model that's found in most NoSQL offerings.

This is the reasons why you might use a NoSQL datastore: While the relational model works well for a number of use cases, a NoSQL data model can support many of those use cases and others that don't fit well into an RDBMS.

Moreover, a NoSQL datastore is able to accept all types of data-structured, semi-structured, and unstructured-much more easily than a relational database. For applications that have a mixture of datatypes, a NoSQL database is a good option.

2. Data Manipulation in NoSQL

NoSQL data systems provide a more relaxed approach to data modeling often referred to as **schema-less modeling,** in which the semantics of the data are embedded within a flexible connection topology and a corresponding storage model. This provides greater flexibility for managing large data sets while simultaneously reducing the dependence on the more formal database structure imposed by the relational database systems.

The flexible model enables automatic distribution of data and elasticity with respect to the use of computing, storage, and network bandwidth in ways that don't force specific binding of data to be persistently stored in particular physical locations. NoSQL databases also provide for integrated data caching that helps reduce data access latency and speed performance.

The loosening of the relational structure is intended to allow different models to be adapted to specific types of analyses. For example, some are implemented as key-value stores, which nicely align to big data programming models like MapReduce. Although the "relaxed" approach to modeling and management paves the way for performance improvements for analytical applications, it does not enforce adherence to strictly-defined structures, and the models themselves do not necessarily impose any validity rules. This potentially introduces risks associated with ungoverned data management activities such as inadvertent inconsistent data replication, reinterpretation of semantics, and currency and timeliness issues. This article discusses four different NoSQL approaches:

- Key-value stores
- Document stores
- Tabular stores
- Object stores

a) Key-value stores:

A **key-value store** is a schema-less NoSQL model in which data objects are associated with distinct character strings called **keys**, similar to the data structure known as a **hash table**. Many of the NoSQL architectures rely on variations on the key-value theme, in that unique keys are employed to both identify entities and to locate attribute information about those entities. This pervasive use of unique keys lends a degree of credibility to this basic approach to a schema-less model.

As an example, consider the data subset represented in Table for example, where the *key* is the name of the automobile

make, while the *value* is a list of names of models associated with that automobile make.

The key-value store does not impose any constraints about data typing or data structure. It is the responsibility of the consuming business applications to interpret the semantics of the data organization.

The core operations performed on a key-value store include:

- Get(key), which returns the value associated with the provided key.
- Put(key, value), which associates the value with the key.
- Multi-get(key1, key2, .., keyN), which returns the list of values associated with the list of keys.
- Delete(key), which removes the entry for the key from the data store.

When using a key-value store, ensuring that the values can be accessed means that the key must be unique. To associate multiple values with a single key (such as the list of car models in the example in Table 1), the developer must consider the representations of those values and how they are to be linked to the key.

Key-value stores are essentially long, "thin" tables, and can be indexed by key value to speed data queries (in that there are not many columns associated with each row). The table's rows can be sorted by the key value to simplify finding the key during a query. A query essentially comprises two steps: the first step is to calculate the unique key, and the second is to use that key as an index into the table. Because of the need to calculate the key to access any information about the entity, it is difficult to expect to execute general SQL-style queries such as "what are the most popular models of cars based on sales?" These kinds of questions would typically be answered using code, as opposed to a query engine.

While key-value pairs are very useful for both storing the results of analytical algorithms (such as the number of times specific phrases occur within massive numbers of documents) and for producing those results for reports, the model does pose some potential drawbacks. One weakness is that the model will not inherently provide any kind of traditional database capabilities (such as atomicity of transactions, or consistency when multiple transactions are executed simultaneously). Those capabilities must be provided by the application itself.

Another potential weakness: as the volume of data increases, maintaining unique values as keys may become more difficult; addressing this issue requires the introduction of some complexity in generating character strings that will remain unique among an extremely large set of keys. For example, a global company may attempt to manage data associated with millions of customers, many of whom sharing the same or similar names. Duplication in the set of names will mean that the name itself will be insufficient when used to differentiate different entities. The upshot is that additional data attributes will need to be added to the composed character string to be used to generate a unique key.

b) Document Stores

A document store is similar to a key-value store in that stored objects are associated (and therefore accessed via) character-string keys. The difference is that the values being stored, which are referred to as "documents," provide some structure and encoding of the managed data. There are different common, standard encodings, including XML (Extensible Markup Language), JSON (Java Script Object Notation), BSON (which is a binary encoding of JSON objects). Aside from these standard approaches to packaging data, other means of linearizing the data values associated with a data record or object for the purposes of storage may be employed.

Figure 1 shows examples of data values collected as a "document" representing the names of specific retail stores. Note that while the three examples all represent locations, the representative models are different. The document representation embeds the structure of the model, allowing the meanings of the document values to be inferred by the application.

One key distinction between a key-value store and a document store is that the latter embeds attribute metadata associated with stored content, which essentially provides a way to query the data based on the contents. For example, using the example in Figure 1, one could search for all documents in which "MallLocation" is "Wheaton Mall" that would deliver a result set containing all documents associated with any "Retail Store" that is in that particular shopping mall.

c) Tabular Stores

Tabular or table-based stores are largely descended from Google's original BigTable design to manage structured data. Hadoop's HBase model is an example of a NoSQL data management system that evolved from BigTable. (For background on BigTable design, see this paper via Google's research website.)

The tabular model allows sparse data to be stored in a three-dimensional table that is indexed by a row key (that is used in a fashion that is similar to the key-value and document stores), a column key that indicates the specific attribute for which a data value is stored, and a timestamp that may refer to the time at which the row's column value was stored.

As an example, various attributes of a Web page can be associated with the Web page's URL: the HTML content of the page, URLs of other web pages that link to this Web page, the author of the content. Columns in a BigTable model are grouped together as "families" and the timestamps enable management of multiple versions of an object. The timestamp can be used to maintain history—each time the content changes, new column affiliations can be created with the timestamp of the when the content was downloaded.

d) Object Data Stores

Object data stores are essentially a hybrid approach to data storage and management; in some ways, object data stores and object databases seem to bridge the worlds of schema-less data management and the traditional relational models.

On the one hand, approaches to object databases can be similar to document stores except that while the document stores explicitly serialize the object so the data values are stored as strings, object databases maintain the object structures. That is because they are bound to object-oriented programming languages such as C++, Objective-C, Java, and Smalltalk.

As opposed to some of the other NoSQL models , object database management systems are more likely to provide traditional ACID compliance (that is Atomicity, Consistency, Isolation, and Durability)—characteristics that are bound to database reliability. Yet this is one of the few similarities to a traditional relational database, and it is important to remember that object databases are not relational databases and are not queried using SQL.

3. Data Modeling in NoSQL

Techniques like logical to physical mapping and normalization / de-normalization have been widely practiced. However, with the recent emergence of NoSQL databases, data modeling is facing new challenges.

Generally speaking, NoSQL practitioners focus on physical data model design rather than the traditional conceptual / logical data model process.

Contrary to traditional, centralized scale-up systems (including the RDBMS tier), modern applications run in distributed, scale-out environments. To accomplish scale-out, application developers are driven to tackle scalability and performance first through focused physical data model design, thus abandoning the traditional conceptual, logical, and physical data model design process.

With its rigidly fixed schema and limited scale-out capability, the traditional RDBMS has long been criticized for its lack of support for big and unstructured data. By comparison, NoSQL databases were conceived from the beginning with the capability to store big and unstructured data using flexible schemas running in distributed scale-out environments.

3.1 Data model design process:

Relational data model designs are the key to successful medium- to large-scale software projects. As NoSQL developers take up business / data model design ownership, a new problem arises which is the data modeling tools. Traditional RDBMS logical and physical data models are made by professionals using tools, such as PowerDesigner or ER/Studio. With NoSQL, there isn't a professional-quality data modeling tools available. Hence, stakeholders review application source codes in order to uncover data model information. These stakeholders include non-technical users such as business owners or product managers. Other approaches, like sampling actual data from production databases, can be very tedious. Now here we will require investment in automation and tooling. NoSQL projects should use the business and data model design process shown in the following diagram.

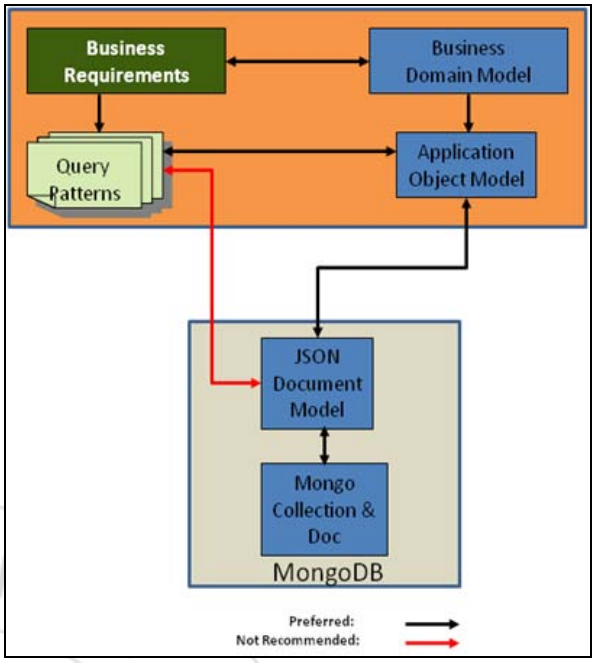

Figure: MongoDB's document-centric model

4. Data Migration approach (Using ETL) in NoSQL

Data accumulated by information systems is one of the important assets for most of the companies. Pushed by customer demands and pulled by changes in technologies, companies from time to time migrate from one information system to another. Hence, data from the legacy system has to be migrated to the new system. Despite the significance of this process, knowledge about migration process is limited. F. Matthes and C. Schulz in fixed a state of the art and provided a literature review of the data migration problem. F. Matthes and C. Schulz define the term data migration as follows: "Tool-supported one-time process which aims at migrating formatted data from a source structure to a target data structure whereas both structures differ on a conceptual and/or physical level."

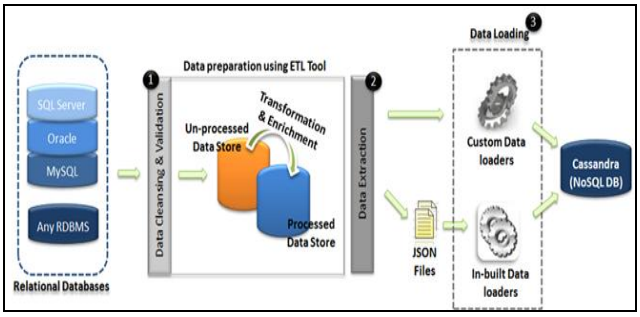

Figure : Data Migration Approach

The overall migration approach would be as follows:
A. Data preparation as per the JSON file format.
B. Data extractions into flat files as per the JSON file format or extraction of data from the processed data store using custom data loaders.

Data loading using in-built or custom loaders into NoSQL data structure (s).The various activities for all the different stages in migration are further discussed in detail in below sections.

4.1 Data Preparation and Extraction

- ETL is the standard process for data extraction, transformation and loading.
- At the end of the ETL process, reconciliation forms an important part. This comprises validation of data with the business processes.
- The ETL process also involves the validation and enrichment of the data before loading into staging tables.

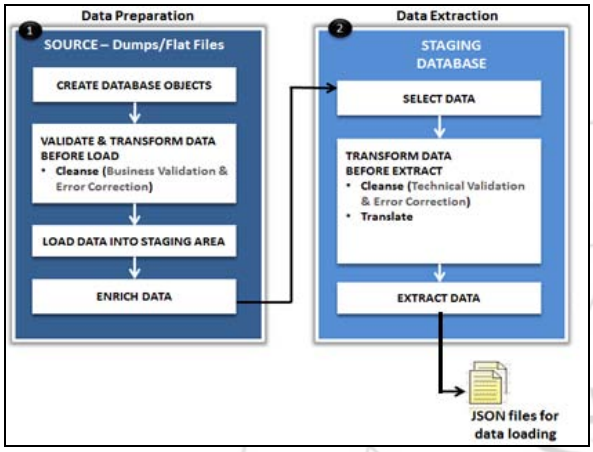

Figure: Data Prepration and Extration Approach

4.2 Data Preparation Activities

The following activities will be executed during data preparation:

1) Creation of database objects
2) Necessary staging tables are to be created as per the requirements based on which will resemble standard open interface / base table structure.
3) Validate & Transform data before Load from the given source (Dumps/Flat files).
4) Data Cleansing
5) Filter incorrect data as per the JSON file layout specifications.
6) Filter redundant data as per the JSON file layout specifications.
7) Eliminate obsolete data as per the JSON file layout specifications.
8) Load data into staging area
9) Data Enrichment
10) Default incomplete data
11) Derive missing data based on mapping or lookups
12) Differently structured data (1 record in as-is = multiple records in to-be).

4.3 Data Extraction Activities (into JSON files)

The following activities will be executed during data extraction into JSON file formats:

1) Data Selection as per the JSON file layout
2) Creation of SQL programs based on as the JSON file layout.
3) Scripts or PLSQL programs are created based on the data mapping requirements and the ETL processes. These programs shall serve various purposes including the loading of data into staging tables and standard open interface tables.
4) Data Transformation before extract as per the JSON files layout specification and mapping documents.
5) Flat files in form of JSON format for data loading

4.4 Data Loading

NoSQL data structures can be accessed using different programming languages like (.net, C, Java, Python, Ruby etc.). Data can be directly loaded from the relational databases (like Access, SQL Server, Oracle, MySQL, IBM DB2, etc.) using these programming languages. Custom loaders could be used to load data into NoSQL data structure(s) based on the enactment rules, customization level and the kind of data processing.

5. Conclusion

The aim of this paper was to focus on taxonomy of NoSQL by explaining its modeling and migration techniques and why we should use NoSQL for various reasons and benefits and to show effectively how they are processed to store large amount of data with high scalability. This work also provides knowledge about why to select NoSQL database and use it effectively. As a result it is expected that it would help adopt NoSQL databases where databases are designed to scale as they grow.

As the two databases (SQL and NoSQL) behave differently according to the type of queries used, the choice of which database to use lies on the type of application the system will be using. The decision to use a NoSQL data store instead of a relational model must be aligned with business users' expectations. The key question: How will the performance of a NoSQL data store compare to their experiences using relational models?

As should be apparent, many NoSQL data management environments are engineered for two key criteria:

- Fast accessibility, whether that means inserting data into the model or pulling it out via some query or access method, and
- Scalability for volume, so as to support the accumulation and management of massive amounts of data.

Both of these criteria are addressed through distribution and parallelization, and the NoSQL styles described above are amenable to extensibility, scalability, and distribution. Moreover, these characteristic features dovetail with programming models like MapReduce that effectively manage the creation and running of multiple parallel execution threads. The key is leveraging data distribution. Fortunately, distributing a tabular data store or a key-value store allows many queries and data accesses to be performed simultaneously, especially when the hashing of the keys maps

to different data storage nodes. NoSQL methods are designed for high performance computing for reporting and analysis, and smart data allocation strategies will enable linear performance scalability in relation to data volume.

There are many new companies who have embraced the different NoSQL models and are bringing their customized versions to market. If you are interested in NoSQL, there is little risk in trying out the different approaches, and it may make sense to develop a simple "pilot" project model that can be deployed in different ways to explore the similarities and differences in terms of ease-of-use, space performance, and execution speed.

Yet while the performance behaviors for NoSQL data management systems are appealing, they will not completely replace a relational database management system. Choosing to use NoSQL is not necessarily an easy decision. One must weigh the business requirements as well as the skills needed to transition from a traditional approach to a NoSQL approach before committing to the technology.

Choosing to use SQL or NoSQL technologies blindly or based on popular demand is harboring the illusion that you automatically made the right choice. Both (SQL and NoSQL) have pros and cons, and the right architecture depends on the requirements of the applications you build. That cranky old SQL database is still tremendously powerful and can reliably handle your transactional demands with integrity. Look for NoSQL options when you are nearing the fringe limitations of relational databases and the vastness of your data handling or scale of operations simply demands a more distributed system. With thoughtful choices, you can become the ONE-free your data and build the next-generation of amazing applications!

References

[1] Ilya Katsov, "NOSQL DATA MODELING TECHNIQUES" Available: https://highlyscalable.wordpress.com/2012/03/01/nosql-data-modeling-techniques/. [Accessed: March 1, 2012]. (General Internet site)
[2] Donovan Hsieh, "NoSQL Data Modeling" Available: http://www.ebaytechblog.com/2014/10/10/nosql-data-modeling/. [Accessed: Oct. 10, 2014]. (General Internet site)
[3] DataStax, "NoSQL Explained" Available: http://www.datastax.com/nosql-databases. (General Internet site)
[4] Nosql-database.org, "LIST OF NOSQL DATABASES" Available: http://nosql-database.org/. (General Internet site)
[5] Phani Krishna Kollapur Gandla, "Migration of Relational Data structure to Cassandra (No SQL) Data structure" Available: http://www.codeproject.com/Articles/279947/Migration-of-Relational-Data-structure-to-Cassandr. [Accessed: Nov. 11, 2011]. (General Internet site)
[6] MongoDB, "RDBMS to MongoDB Migration Guide" Available: https://s3.amazonaws.com/info-mongodb-com/RDBMStoMongoDBMigration.pdf. [Accessed: May, 2016]. (General Internet site)
[7] M.A. Mohamed, O.G Altrafi, M.O Ismail, "Relational vs NoSQL Databases: A Survey." in International Journal of Computer and Information Technology (ISSN:2279-0764) Volume 03-Issue, May 03 2014. (Research paper)
[8] Mital Potey, Megha Digrase,Gaurav Deshmukh, Minal Nerkar, "Database Migration from Structured Database to non-Structured Database." in International Conference on Recent Trends and Advancements in Engineering Technology ICRTAET 2015 (ISSN: 0975-8887), 2015. (Research paper)
[9] Luke P. Issac, "SQL vs NoSQL Databases Differences Explained" Available: http://www.thegeekstuff.com/2014/01/sql-vs-nosql-db/ [Accessed: Jan. 14, 2014]. (General Internet site)
[10] Girts Karnitis, Guntis Arnicans, "Migration of Relational Database to Document-Oriented Database: Structure - Denormalization and Data Transformation." in 7th International Conference on Computational Intelligence, Communication Systems and Networks (CICSyN), 2015. (Research paper)

Author Profile

Priya Badlani is currently pursuing her Master's degree in Computer Application from Vivekanand Education Society's Institute of Technology, Chembur, Mumbai. During the year 2013 she had completed her bachelor's degree in Computer Science from Smt. C.H.M College of Science, Commerce, Arts and Management,Ulhasnagar, Mumbai.