(Un)Structuring for the Next Generation: New Possibilities for Library Data with NoSQL

Matthew D. Harrington
Duke University Libraries, matthew.harrington@duke.edu

Dennis B. Christman
Duke University Libraries, dennis.christman@duke.edu

Author ORCID Identifier: https://orcid.org/0000-0003-0857-7923

Matthew D. Harrington and Dennis B. Christman, "(Un)Structuring for the Next Generation: New Possibilities for Library Data with NoSQL" (2018). *Proceedings of the Charleston Library Conference.*
http://dx.doi.org/https://doi.org/10.5703/1288284317069

(Un)Structuring for the Next Generation: New Possibilities for Library Data with NoSQL

Matthew D. Harrington, Duke University Libraries, matthew.harrington@duke.edu

Dennis B. Christman, Duke University Libraries, dennis.christman@duke.edu

Introduction

A recent ACRL TechConnect blog post pointed out some of the difficulties working with personal names as data. The assumptions that names do not change or that all full names have a first and last name are not always true. Nor is it true that a preferred name is preferred in all contexts (Phetteplace, 2018). However, most library systems are designed based on assumptions such as these, and when exceptions are found, the data are forced to comply with those assumptions rather than challenging them. Libraries may be driven by the lofty idea that clean, structured data is the ultimate achievement, but the structure of library data is rarely uniform and static. It is dynamic, diverse, and increasing at a rapid pace. Instead of trying to force nonconforming data into a rigid structure, libraries need to examine ways in which that structure can become more flexible in order to accommodate these changes.

These changes reflect a larger trend that has been ongoing for many years. In 2001, Doug Laney published an article that outlined three data trends that would become known as "Gartner's 3 Vs." The model is meant to show how the demands on our data system have changed and will continue to change, and in response to these changing demands, data systems will need to adapt. The first trend Laney describes is volume, which refers to the sheer amount of data a system needs to interact. The second, variety, refers to different types of data and a decrease in the consistency of that data. Laney goes as far as saying that without significant work to address this, it will be the single greatest barrier to data management. Lastly, velocity refers to the rates at which our systems are accessed and need to transact data (Laney, 2001).

In the context of the library, these trends have all made an impact. As libraries try to connect users with more resources, systems need to interact with more data, thus increasing the volume of resource metadata. The metadata is then pulled from an increasing number of sources, mixing MARC catalogs, Dublin Core repositories, article databases, and more. And though the velocity of library data does not compare with the system traffic of major e-commerce or social media sites, the library still must process an influx of transactional data from both patrons and vendors. Strained resources and budget cuts further intensify these trends for the library. The library is headed toward a critical tipping point in data management if a flexible alternative is not found.

Structured and Unstructured Data

A useful concept to use while examining the flexibility of library data structures is that of structured and unstructured data. Structured data follows a rigidly defined data model that allows a machine to easily make connections and process it, such as data stored in a relational database. Conversely, unstructured data is very loose. There is little uniformity in where different data points may appear. This makes it very difficult for a machine to process and relate to other data. For example, a group of Twitter posts contains primarily human-readable information.

Some data are highly structured, and some highly unstructured, but most lie somewhere in between. Even in Twitter posts, an example commonly provided for unstructured data, there are some pieces of data that appear very regularly. A machine can easily detect who posted the tweet, who liked it, who reposted it, who replied, and when each of those events happened. Only the body of the posts is truly difficult to process by machine. But the data remain divided between what can easily be processed by machine and what must be interpreted by a human.

Semistructured data, on the other hand, is a blend of both structured and unstructured data. It contains structure, but that structure is more fluid. Instead of applying a rigid structure to data, semistructured data has a flexible schema that adapts to changes easily, yet it remains machine readable.

Libraries can use this framework to look at how effective their current systems are at storing different types

https://doi.org/10.5703/1288284317069

of library data. On the structured end is circulation data. The exact fields collected will vary by system and institution, but it is easy to have uniform data regarding circulation transactions. Patron data, on the other hand, is less structured. Most fields collected will be uniform for most patrons, but data models may vary based on factors such as patron type. An institution might want to store data related to a student's academic department, but that field may not be relevant for most staff. Likewise, acquisitions data may vary according to vendor or type of purchase. The data about a book purchase from an institution's own university press is going to look vastly different from an international order for a subscription-based streaming video package. There are similarities between these orders, but there are also differences, which require a more flexible data model.

Arguably the least structured type of library data is bibliographic data. MARC data is often thought of as very ordered and structured, but this framework dismantles that notion. There are certainly well-defined rules on what data to put where in a MARC record, but taking that record and fitting it into a rigid model that a relational database can interpret is very challenging. MARC can be used to describe an enormous variety of resources. It can describe books, films, music, manuscripts, maps, datasets, and many more resource types. Each resource can then be physical or digital and published as a single monograph, a multipart monograph, a serial, or an integrating resource. Every combination requires a different set of data to accurately describe it. Even something as simple as the title of a book can be complicated. MARC has 14 types of title fields, many of which are repeatable. Modeling this in a relational database would require either a single table with a long list of attributes, most of which would be empty, or many small tables to represent each different possibility, which would require a lot of on-demand processing power to join the tables together when needed. Yet most integrated library systems continue to use a relational database to store MARC data. To find alternatives, one must first examine the relational database model to study how it operates and then determine whether the model is most appropriate for storing library data, MARC or otherwise.

The Relational Database Model

Even though there are many commercial and open-source options available for relational database systems (RDMS), they share some characteristics. On a basic level, they rely upon the storage of structured data in a flat, tabular format. This means each attribute in a table must conform to specific rules regarding data type and size, and each can only contain up to one value. This schema is defined in the design process before the database is populated with data, and these design decisions can be difficult to change once the database is built.

Relational databases also rely on the use of joins. One of the main functions of the relational database is to reduce redundancy through database normalization, thereby maintaining data integrity. Normalization involves logically dividing attributes into different relations and joining those relations using primary and foreign keys. This can have a dramatic effect on performance, though. As systems become more complex, the number of joins multiplies, and as a result, simple queries must access additional tables of data to return the necessary attributes. Not only does this slow performance, it also leads to difficulty distributing the database across multiple servers.

Lastly, relational databases generally use SQL to query the data. SQL is a powerful language for querying data, and it works very well when the schema is static. However, because the data are so reliant upon joined tables with rigidly defined attributes, slight changes to those attributes can severely disrupt the functionality of the query. Furthermore, SQL cannot easily query complex relationships nor complex documents stored within an attribute.

Despite these disadvantages, relational databases are popular and widely used, partially because they perform well in the right circumstance and partially because there have been few alternative solutions. In the library world, the relational database has been a fixture in both commercial and open-source integrated library systems for many years, but the increased volume, variety, and velocity of data is quickly rendering them ineffective. New material types continually present difficulties in cataloging, new packaging/pricing models lead to complex workarounds in acquisitions, and new forms of technology challenge even the communication formats stored in patron records. As a result, libraries engage in tedious data cleanup and normalization projects to ensure all data fit within the parameters of the database model. In this sense, data are often viewed as malleable pieces that must adapt to a rigid structure rather than vice versa. Instead, libraries need a system flexible enough to handle messy data rather than constantly trying to fit messy data into neatly arranged cells.

The NoSQL Database Model

Recently, there has been an increasingly popular alternative to relational databases. When data is dynamic and the schema must have flexibility to accommodate those changes, NoSQL databases offer a viable solution. NoSQL as a term describes many types of nonrelational databases, but they generally possess similar properties. They usually have flexible and extensible schemas, which do not need to be defined before data has been added. Data is typically stored as JSON or XML documents rather than organized in a table. And NoSQL databases use languages other than SQL to query data.

There are several types of NoSQL databases, and the simplest is the key-value database. Key-value databases are composed of data values paired with associated keys for retrieval. The range of those values may vary. Some are even able to store JSON documents as values, which makes them appear more like document databases. This simple structure allows them to be highly scalable when there is a need to continually write and retrieve data, such as storing circulation transactions in a library system.

Document databases typically store JSON or XML documents, and because they represent holistic views of data, retrieval is fast. Null values are unnecessary; the only data contained within the document are the data elements with values. Furthermore, those documents do not have to contain the same elements. A document describing a photograph would require different elements than one describing a print book, much like parts of a MARC record are tailored to a specific material type. The document model is more appropriate when data has complex descriptions, especially those containing hierarchies acting as one-to-many relationships, such as a patron record. The patron record may contain elements unique to the specified patron type, and it may also use nested data to store repeatable elements, such as addresses stored in an array.

A third NoSQL model is the graph database, which uses a triple to store relationships between data. A triple is simply a three-part construction consisting of a subject, a predicate, and an object. To represent this structure, graph databases use nodes and edges, so in the structure of a triple, nodes are subjects and objects, and the edge is the predicate defining the relationship between those two nodes. This type of database is useful for highlighting relationships, especially many-to-many relationships such as those found in social media sites. Library bibliographic data also contains several types of many-to-many relationships. For example, a book may have several authors and one author may write several books. The author and the work might be the two nodes in this triple, and the edge describes the author/work relationship.

Another type of NoSQL database that combines the characteristics of these three models is the multimodel database. Multimodel databases may function as key-value stores, document databases, and graph databases at the same time. Multimodel databases have broader capabilities for managing a variety of data, which means a single database may be able to adapt to the diverse data formats and relationships while also maintaining the flexibility to change as new technologies and workflows force that data to change.

A Sample NoSQL Database

To show what a multimodel NoSQL database can do, we created a sample database that included a list of actors and the films in which they acted. The goal was to be able to query graph traversals between two actors in order to determine how many degrees of separation exist between them. For our database, we chose ArangoDB. Like many NoSQL databases, there is a “community edition” available to freely download.

ArangoDB operates like a document database, except those documents are stored as a value with a key (see Figure 1), making it also function like a key-value database for retrieval. Except for the three metadata elements at the top of the document, other documents are not required to contain the same elements.

Figure 1. Sample JSON document for the film *Black Sheep*.

Documents are stored within collections, but a collection may be set up to act as either a node or an edge in order to graph links between documents. Edges look like nodes, but they also contain "from" and "to" elements to construct the link between two nodes. This allows the database to contain triples, which define the relationships between the documents stored in nodes.

To query data stored in collections, ArangoDB uses a proprietary language called AQL. Most queries function like "for" loops. This allows queries to run an indeterminate number of graph traversals across the edge. When querying for degrees of separation between actors, the number of degrees is unknown, which makes this a difficult query to construct in a relational database. However, ArangoDB can easily process the query, using a simple for loop to return the results in Figure 2. These show that Chris Farley acted with Tim Matheson in the film *Black Sheep*, and Tim Matheson starred with Kevin Bacon in *National Lampoon's Animal House*. Therefore, there are two degrees of separation between Chris Farley and Kevin Bacon.

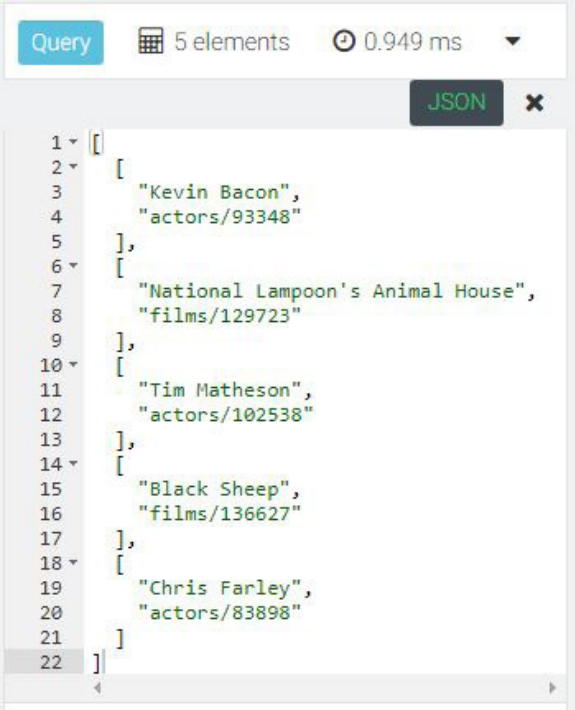

Figure 2. Degrees of separation between Kevin Bacon and Chris Farley.

The graph function in ArangoDB also allows those relationships between actors to be visualized. To see all actors within one degree of separation of Chris Farley, a graph can be constructed based on the edge collections (see Figure 3).

Here, Chris Farley is at the center of the graph, and the five films from this database starring Chris Farley link out from his name. Actors from those five films appear linked to those films, and the search depth of the graph may be expanded outward to include

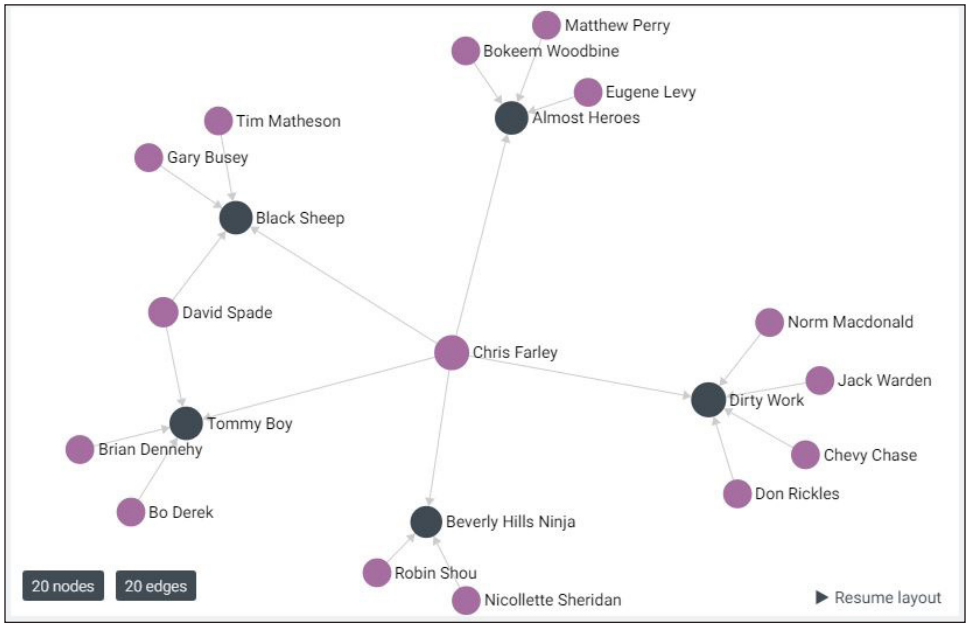

Figure 3. Graph a single degree of separation from Chris Farley.

additional films and actors. This allows a viewer to navigate through the links that connect an actor to other actors and films.

NoSQL and Library Data

NoSQL databases offer a more flexible structure than traditional relational database systems, and therefore they may be better suited for the increased variety, volume, and velocity of library data. NoSQL documents store data without the need for predefined schemas that have placed limits upon the rows of relational tables, and because they provide a holistic view of the document, there is no need to access data that may be spread across multiple tables through complex joins. Furthermore, using a more powerful scripting language as an alternative to SQL means the database may be able to handle more complex queries. In order to effectively deliver quality resources to patrons, libraries will need to develop new systems that utilize technologies like NoSQL to match the growing demands of our data.

References

Laney, D. (2001, February 6). 3D data management: Controlling data volume, velocity, and variety. *Application Delivery Strategies*. Retrieved from https://blogs.gartner.com/doug-laney/files/2012/01/ad949-3D-Data-Management-Controlling-Data-Volume-Velocity-and-Variety.pdf

Phetteplace, E. (2018, May 14). Names are hard. *ACRL TechConnect*. Retrieved from https://acrl.ala.org/techconnect/post/names-are-hard/