Data Modeling for NoSQL Document-Oriented Databases

Harley Vera, Wagner Boaventura, Maristela Holanda, Valeria Guimarães, Fernanda Hondo
Department of Computer Science
University of Brasília
Brasília, Brasil.
{harleyve,wagnerbf}@gmail.com, mholanda@cic.unb.br,
{valeriaguimaraes, fernandahondo}@hotmail.com

Abstract

In database technologies, some of the new issues increasingly debated are non-conventional applications, including NoSQL (Not only SQL) databases, which were initially created in response to the needs for better scalability, lower latency and higher flexibility in an era of bigdata and cloud computing. These non-functional aspects are the main reason for using NoSQL database. However, currently there are no systematic studies on data modeling for NoSQL databases, especially the document-oriented ones. Therefore, this article proposes a NoSQL data modeling standard in the form of ER diagrams, introducing modeling techniques that can be used on document-oriented databases. On the other hand the purpose of this article is not structure the data using the model proposed, but it does helping with the visualization of data. In addition, to validate the proposed model, a study case was implemented using genomic data.

1 Introduction

Huge amounts of data are produced daily. They are generated by smart phones, social networks, banks transactions, machines measured by sensors are part of Internet of Things provide information that is growing exponentially. The management of this data is currently performed in most cases by relational databases that provide centralized control of data, redundancy control and elimination of inconsistencies (Elmasri and Navathe, 2010); but, some of these factors restrict the use of alternative database models. Consequently, certain limiting factors have led to alternative models of databases in these scenarios. Primarily, motivated by the issue of system scalability, a new generation of databases, known as NoSQL, is gaining strength and space in information systems. The NoSQL databases emerged in the mid-90s, from a database solution that did not provide an SQL interface. Later, the term came to represent solution that promote an alternative to the Relational Model, becoming an abbreviation for Not Only SQL.

The purpose, therefore, of NoSQL solutions is not to replace the Relational Model as a whole, but only in cases in which there is a need for scalability and bigdata. In the recent years, a variety of NoSQL databases has been developed mainly by practitioners looking to fit their specific requirements regarding scalability performance, maintenance and feature-set. Subsequently, there have been various approaches to classify NoSQL databases, each with different categories and subcategories, such as key-value stores, column-oriented and graph databases, oriented-document. MongoDB (MongoDB, 2015), Neo4j (Partner et al., 2013), Cassandra (D. Borthakur et al., 2011) and HBase (F. Chang et al., 2008) are examples of NoSQL databases. This article only applies to NoSQL document-oriented databases, because of the heterogeneous characteristics of each NoSQL database classification.

Nonetheless, data modeling still has an important role to play in NoSQL environments. The data modeling process (Elmasri and Navathe, 2010) involves the creation of a diagram that represents the meaning of the data and the relationship between the data elements. Thus, understanding is a fundamental aspect of data modeling (R. F. Lans, 2008), and a pattern for this kind of representation has few contributions for NoSQL databases.

Addressing this issue, this article proposes a standard for NoSQL data modeling. This proposal uses NoSQL document-oriented databases, aiming to introduce modeling techniques that can be

used on databases with document features.

The remainder of the paper is organized as follows: Section II presents related works. Section III explores the concepts of modeling for NoSQL databases based on documents, introducing the different types of relationships and associations. Section IV shows the proposal model in the context of NoSQL databases based on documents. Section V presents the study case to validate the proposal model. Finally in Section VI, presents the conclusion of the research and future works.

2 Related Works

Katsov (H. Scalable, 2015) presents a study of techniques and patterns for data modeling using different categories of NoSQL databases. However, the approach is generic and does not define a specific modeling engine to each database.

Arora and Aggarwal (R. Arora and R. Aggarwal, 2013) propose a data modeling, but restricted to MongoDB document database, describing a UML Diagram Class and JSON format to represent the documents.

Similarly, Banker (K. Banker, 2011) provides some ideas of data modeling, but limited to MongoDB database and always referring to JSON (D. Crockford, 2006) format as a modeling solution.

Kaur and Rani (K. Kaur, K.Rani, 2013) present a work for modeling and querying data in NoSQL databases, especifically present a case study for document-oriented and graph based data model. In the case of document-oriented propose a data modeling restricted to MongoDB document database, describing the data model by UML diagram class to represent documents.

3 Data Modeling For Document-Oriented Database

An important step in database implementation is the data modeling, because it facilitates the understanding of the project through key features that can prevent programming and operation errors. For relational databases, the data modeling uses the Entity-Relationship Model (Elmasri and Navathe, 2010). For NoSQL, it depends on the database category. The focus of this article is NoSQL document-oriented databases, where the data format of these documents can be JSON, BSON, or XML (S. J. Pramod, 2012).

Basically, the documents are stored in collections. A parallel is made with relational databases, the equivalent for a collection is the record (tuple) and for a document it is the relation (table). Documents can store completely different sets of attributes, and can be mapped directly to a file format that can be easily manipulated by a programming language. However, it is difficult to abstract the modeling of documents for the entity relationship model (R. F. Lans, 2008).

3.1 Modeling Paradigm for document-oriented Database

The relational model designed for SQL has some important features such as integrity, consistency, type validation, transactional guarantees, schemes and referential integrity. However, some applications do not need all of these features. The elimination of these resources has an important influence on the performance and scalability of data storage, bringing new meaning to data modeling.

Document-oriented databases have some significant improvements, e.g., index management by the database itself, flexible layouts and advanced indexed search engines (H. Scalable, 2015). By associating these improvements (some being denormalization and aggregation) to the basic principles of data modeling in NoSQL, it is possible to identify some generic modeling standards associated to document-oriented databases. Analyzing the documentation of the main document-oriented databases, MongoDB (MongoDB, 2015) and CouchDB (CouchDB, 2015), similar representations of data mapping relationships can be found: **References** and **Embedded Documents**, a structure which allows associating a document to another, retaining the advantage of specific performance needs and data recovery standards.

3.2 References Relationship

This type of relationship stores the data by including links or references, from one document to another. Applications can solve these references to access the related data in the structure of the document itself (MongoDB, 2015). Figure 1 shows two documents one of them for **Fastq** files and the other to **Activities**.

3.3 Embedded Documents

This type of relationship stores in a single document structure, where the embedded documents are disposed in a field or an array. These denormalized data models allow data manipulation in a single database transaction (MongoDB, 2015).

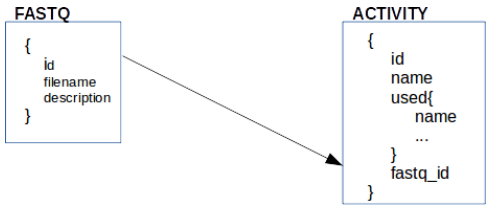

Figure 1: Example of documents referenced

Figure 2 shows a document of a genome **Project** with a **Activity** embedded document.

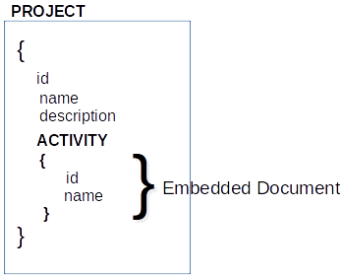

Figure 2: Example of Embedded documents

4 Proposal For Document-Oriented Databases Viewing

Unlike traditional relational databases that have a simple form in the disposition in rows and columns, a document-oriented database stores information in text format, which consists of collections of records organized in key-value concept, ie, for each value represented a name (or label) is assigned, which describes its meaning. This storage model is known as JSON object, and the objects are composed of multiple name/value pairs for arrays, and other objects.

In this scenario, the number of objects in a database increases the abstraction complexity of the logical relationship between the stored information, especially when objects have references to other objects. Currently, there is a lack of solutions to conceptually represent those associated with a NoSQL document-oriented database. As described in (R. Arora and R. Aggarwal, 2013), there is no standard to represent this kind of object modeling, several diferent manners of modeling may arise, depending on each data administrator's understanding, which makes learning difficult for those who need to read the database model.

Therefore, this section proposes a standard for document-oriented database viewing. Our proposal has some properties, considering the conceptual representation modeling type, such as:

- Ensuring a single way of modeling for the several NoSQL document-oriented databases.
- Simplifying and facilitating the understanding of a document-oriented database through its conceptual model, leveraging the abstraction and making the correct decisions about the data storage.
- Providing an accurate, unambiguous and concise pattern, so that database administrators have substantial gains in abstraction, understanding.
- Presenting different types of relationships between collections defined as References and Embedded documents.
- Assisting the recognition and arrangement of the objects, as well as its features and relationships with other objects.

The following subsections present the concepts and graphing to build a conceptual model for NoSQL document-oriented databases.

4.1 Assumptions

Before starting the discussion about the approach of each type of the conceptual modeling representation, it is important to highlight some basic concepts about objects and relationships in a document-oriented database:

- A document (or object) describes a set of attributes that have their properties organized in a key-value structure.
- Information contained in an document is described by the identifier (key) and the value associated with the key.
- Different types of relationships between documents are defined as **References** and **Embedded Documents**
- Because NoSQL is a non-relational data database, the concepts of normalization, do not apply.
- Some concepts of relationships between objects are similar to ER modeling, such as cardinality (one-to-one, one-to-many, many-to-many).

4.2 Basic Visual Elements

The proposed solution for a conceptual modeling to the NoSQL document-oriented databases has two basic concepts: **Document** and **Collections**.

As noted previously, a document is usually represented by the structure of a JSON object, and as many fields as needed may be added to the document. For this solution, a document and a collection of documents is represented by Figure 3.

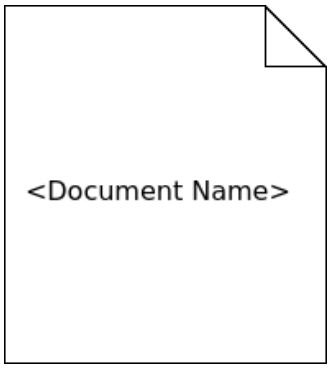

Figure 3: Graphical representation of a Document

The following section presents the definitions of relationship types and degrees for the objects features.

4.3 Embedded Documents 1..1

This section proposes a model that represents the one-to-one relationship for documents embedded in another document. In this case, the proposal is to use the representation of an individual Document within another element that represents a Document. In Figure 4, cardinality is also suggested to specify the one-to-one relationship type.

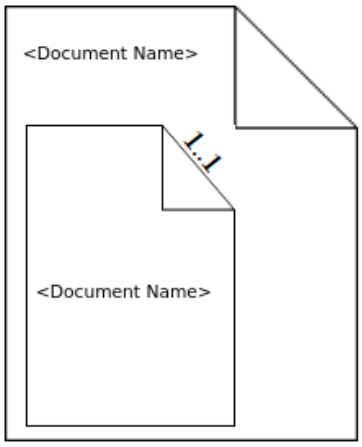

Figure 4: One-to-one relationship for embedded documents

4.4 Embedded Documents 1..N

A one-to-many relationship in embedded documents is represented by the Figure 5. This is the case when the notation to represent the cardinality is the same used in UML (F. Booch et al., 2005) and is placed in the upper right corner of the embedded documents. According to the cardinality one-to-many the larger document has embedded multiple documents within it.

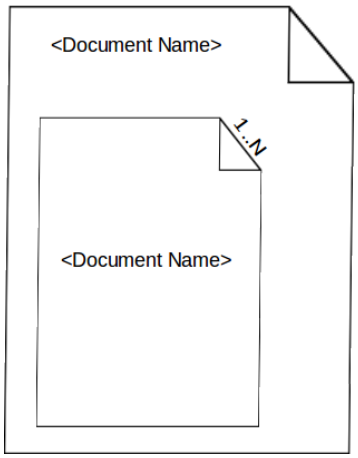

Figure 5: One-to-many relationship for embedded documents

4.5 Embedded Documents N..N

A many-to-many relationship in embedded documents is represented by the Figure 6. According to the cardinality many-to-many the larger document has a many to many relationship with the embedded document. The representation of the cardinality is the same used in UML (F. Booch et al., 2005).

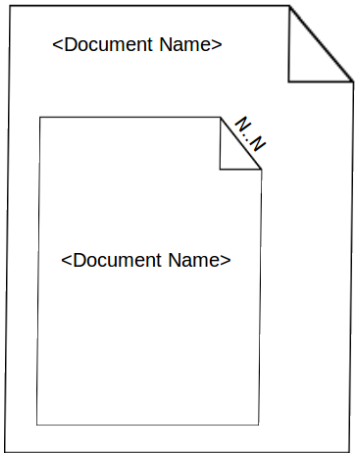

Figure 6: Many-to-many relationship for embedded documents

4.6 References 1..1

A document can reference another, and in this case, one must use an arrow directed to the referenced document, as shown in Figure 7. One can see that the directed arrow makes the left document references to the right document. Furthermore, the cardinality of the relationship should be specified above the arrow. The notation of cardinality is based on UML (F. Booch et al., 2005).

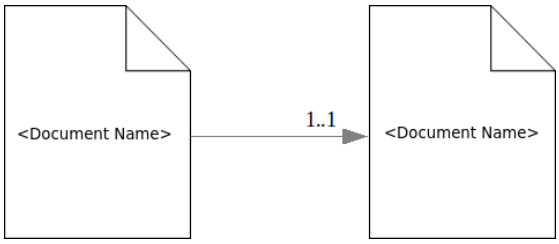

Figure 7: One-to-one relationship for documents referenced

4.7 References 1..N

In NoSQL, a document can reference multiple documents. To represent this relationship one should use an arrow directed to the referenced documents, as shown in Figure 8. The left document references multiple documents on the right side, by the directed arrow. Furthermore, the cardinality of the relationship is represented by the notation "1..N" as in UML (F. Booch et al., 2005).

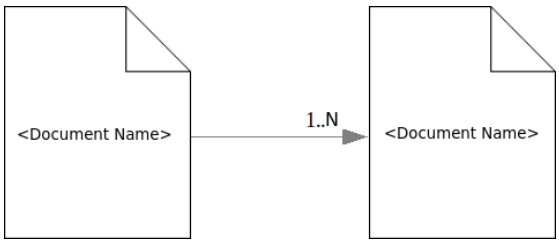

Figure 8: One-to-many relationship for documents referenced

4.8 References N..N

To represent this relationship a bidirectional arrow is used between reference documents, as shown in Figure 9. The left document references multiple documents on the right side and the right document references multiple documents on the left side. Furthermore, the cardinality of the relationship is represented by the notation "N..N" as in UML (F. Booch et al., 2005).

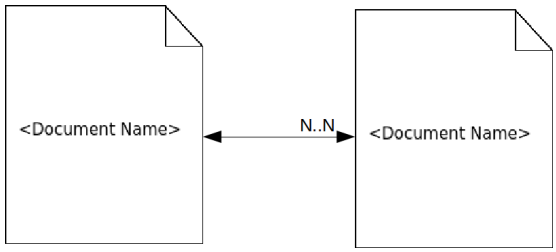

Figure 9: Many-to-many relationship for documents referenced

5 Case Study

In order to evaluate our proposal, part of the workflow described in (J. C. Marioni et al., 2014) was used. This workflow aimed at identify and comparing expression levels of human kidney and liver RNA samples sequenced by Illumina. The workflow was designed in three phases (Figure. 10):

- Filtering: all the sequenced transcripts were filtered, generating new files with good quality sequences.
- Alignment: transcripts were mapped to the human genome used as reference.
- Statistical Analysis: a sort process was first executed, followed by a statistical analysis with the mapped transcripts to discover which genes are mostly expressed both in kidney and liver samples.

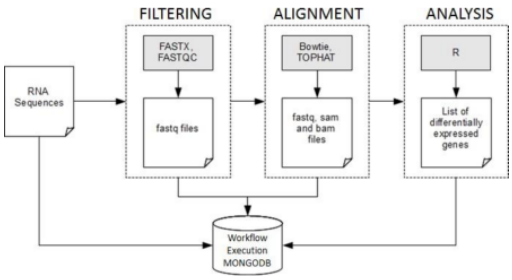

Figure 10: Workflow for analysis of differential expression among kidney and liver RNA samples

After analyzing the previously mentioned concepts, we have chosen to create a collection of documents for each PROV-DM type used to create a graph node. We also defined a collection for genomic documents (raw data). The reference relationship approach was chosen to connect all PROV-DM components, complementary information of PROV-DM and genomic documents. Based on (R. de Paula et al., 2013) we defined the documents and the attributes. A set of minimum information related to each one of these entities.

Figure. 11 shows our document based data representation, explained as follows:

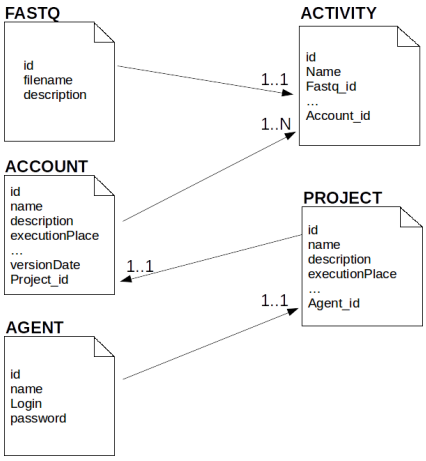

Figure 11: Data Modeling for Genomic Provenance for NoSQL Based on Documents.

- Project: stores different experiments of one agent. Attributes: Id, name, description, coordinator, start date, end date and observation.
- FASTAQ: files used or generated in the activity; Attributes: Id, filename and description.
- Activity: represents the execution of a program; Attributes: Id, name, program, version program, command line, function, start date, end date, account ID, used (name FASTQ, local, size), wasGenerateBy (name FASTQ, local, size), and wasAssociatedWith (Agent name).
- Account: represents the performance of an experiment; Attributes: Id, name, description, execution place, star date, end date, observation, version and version date and project Id.
- Agent: represents the person responsible for a program or a phase in the workflow. Attributes: Id, name, login and password.

5.1 Implementation

In this case study, we have considered the MongoDB NoSQL database to store provenance and data files. The primary motivation for this choice was MongoDB's ability to manipulate large volumes of data. MongoDB is an open-source Document-Oriented database designed to store large amounts of data from multiple servers. It uses JSON- style documents with dynamic schemas. The number of fields, content and size of the document can differ from one document to another. In practice, however, the documents in a collection share a similar structure (MongoDB, 2015) and can be mapped directly to a file format that can be easily manipulated by a programming language.

MongoDB documents have a maximum size of 16MB. This feature is important to ensure that a single document cannot use excessive amounts of RAM. In order to store files larger than the maximum size, MongoDB provides a GridFS API (MongoDB, 2015). It automatically divides large data into 256 KB pieces and maintains metadata for all pieces. GridFS allows for the retrieval of individual pieces as well as entire documents.

GridFS uses two collections to store the data: fs.files collections, containing metadata about files, and fs.chunks collections, which store the actual 256k data chunks. The collections FS.file contains the name of the FASTQ file. Thus, it was possible to implement the relationship between MongoDB Collection Activity using Reference Document. In other words, we implemented the connection between Level 1 and Level 2 through the File Name attribute that was present in fileprovenance.files and Activity Collection. Figure. 12 illustrates this particular implementation.

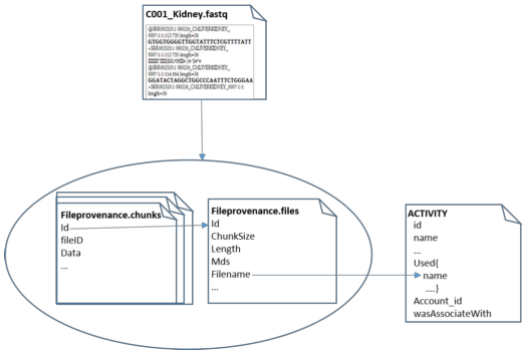

Figure 12: GridFS Implementation

6 Conclusions and Future Works

In contrast to relational database management systems, NoSQL databases are designed to be schemaless and flexible. Therefore, the challenge of this work was to introduce a data modeling standard for NoSQL document-oriented databases, in contrast to the original idea for NoSQL databases.

The objective was to build compact, clear and intuitive diagrams for conceptual data modeling for NoSQL databases. While the current studies propose generic techniques and do not define a specific modeling engine to NoSQL database, our idea was to present a graphical model for any NoSQL document-oriented database. Moreover, while other studies describe techniques based on UML Diagram Class and JSON format as a modeling solution, we have a new approach to solve the conceptual data modeling issue for NoSQL document-oriented databases.

Future work includes: verifying our model for other NoSQL database classifications, such as key-value and column.

References

R. Elmasri and S. Navathe. 2010. *Fundamentals of Database Systems*. Pearson Addison Wesley.

MongoDB. 2015. *Document database*. [Online] Available: `http://www.mongodb.org/` [Retrieved:April, 15].

J. Partner, A. Vukotic, and N. Watt. 2013. *Neo4j in Action*, O'Reilly Media.

D. Borthakur et al. 2011. *Apache hadoop goes realtime at facebook*, in Proceedings of the 2011 ACM SIGMOD International Conference on Management of data. ACM, 2011, pp. 1071–1080.

F. Chang et al. 2008. *"Bigtable: A distributed storage system for structured data*, ACM Transactions on Computer Systems (TOCS), vol. 26, no. 2, 2008, p. 4.

R. F. Lans. 2008. *Introduction to SQL: mastering the relational database language*, Addison-Wesley Professional.

H. Scalable. 2015. *Nosql data modeling techniques*. [Online] Available: `http://highlyscalable.wordpress.com/2012/03/01/nosql-data-modeling-techniques/` [Retrieved:April, 15].

R. Arora and R. Aggarwal, 2013. *Modeling and querying data in mongodb*, International Journal of Scientific and Engineering Research (IJSER 2013), vol. 4, no. 7, Jul. 2013, pp. 141–144.

K. Banker, 2011. *MongoDB in action*, Manning Publications Co.

D. Crockford, 2006. *RFC 4627 (Informational) The application json Media Type for JavaScript Object Notation (JSON)*, IETF (Internet Engineering Task Force)

K. Kaur, K.Rani, 2013. *Modeling and querying data in NoSQL databases*, In Big Data, IEEE International Conference on (pp. 1-7). IEEE.

S. J. Pramod, 2012. *Nosql distilled: A brief guide to the emerging world of polyglot persistence*,

MongoDB, 2015. *Data modeling introduction*, Online Available: `http://docs.mongodb.org/manual/core/data-modeling-introduction/` [Retrieved: April, 15].

CouchDB, 2015. *Modeling entity relationships in couchdb*, [Online]. Available: http://wiki.apache.org/couchdb/ [retrieved: April, 15]

G. Booch, J. Rumbaugh, and I. Jacobson, 2005. *The unified modeling language user guide.*, Pearson Education India.

J. C. Marioni, C. E. Mason, S. M. Mane, M. Stephens, and Y. Gilad, 2014. *RNA-SEQ: An assessment of technical reproducibility and comparison with gene expression arrays*, Genome Research, vol. 18, no. 9, pp. 1509–1517.

R. de Paula, M. Holanda, L. SA Gomes, S. Lifschitz and M. E. MT. Walter, 2013. *Provenance in bioinformatics workflows.*, BMC Bioinformatics 14 (Suppl 11):S6.