A New Algorithm for Storing and Migrating Data Modelled by Graphs

https://doi.org/10.3991/ijoe.v16i11.15545

Zakariyaa Ait El Mouden (✉), Abdeslam Jakimi
Moulay Ismail University, Meknes, Morocco
mouden.zakariyaa@outlook.com

Abstract—NoSQL databases have moved from theoretical solutions to exceed relational databases limits to a practical and indisputable application for storing and manipulation big data. In term of variety, NoSQL databases store heterogeneous data without being obliged to respect a predefined schema such as the case of relational and object-relational databases. Those solutions, also surpass the traditional databases in storage capacity; we consider MongoDB for example, which is a document-oriented database capable of storing unlimited number of documents with a maximal size of 32TB depending on the machine that runs the database and also the operating system. Also, in term of velocity, many researches compared the execution time of different transactions and proved that NoSQL databases are the perfect solution for real-time applications. This paper presents an algorithm to store data modeled by graphs as NoSQL documents, the purpose of this study is to exploit the high amount of data stored in SQL databases and to make such data usable by recent clustering algorithms and other data science tools. This study links relational data to document datastores by defining an effective algorithm for reading relational data, modelling those data as graphs and storing those data as NoSQL documents.

Keywords—Graph schema, NoSQL, Document database, Object-Relational database, MongoDB

1 Introduction

XML (eXtensible Markup Language) is a powerful tool used to encode documents in order to exploit their data with different database management systems, algorithms and applications [1, 2]. XML is a sample language based on the use of tags and can be easily readable by both human and machine.

Many frameworks and APIs (Application Programming Interfaces) use XML files for mapping between their different components, we cite for example Struts, Hibernate, EJB (Enterprise JavaBeans), JMX (Java Management Extensions) for Java web applications and Ajax for JavaScript web applications. GEXF (Graph Exchange XML Format) is an extension of XML which specifies the main parameters to describe graphs and networks with a set of nodes and their positions in addition to the set of edges and their type (directed or undirected) and many other parameters.

No one can deny that the world is currently dominated by SQL relational databases, as they still take place in training programs for different computer science branches, another reason of their popularity is the massive amount of disponible documentations for SQL databases as they're existing since almost half a century. SQL databases respects the ACID properties [3], where a transaction should be done completely or canceled at all (Atomicity), each transaction leads the system from a valid state to another valid state without violating any constraints (Consistency), each transaction is executed as if it is the only existing transaction in the system (Isolation) and finally each transaction must be saved once executed (Durability).

Recently NoSQL (Not Only SQL) is making its own way to the top list of the most popular databases, many reasons harried up the appearance of those solution as the growing volume of data existing nowadays and their heterogeneity, the majority of the collected data are non-structured and cannot be fit into a relational or object-relational database [4] as those models use a predefined schema for each created structure (table, view, ..) and each transaction in the language of data manipulation (DML) firstly compares the data given in parameters with the existing schema of the structure and generates errors in the case of a minimum mismatch. NoSQL rejects the ACID properties and have no schemas to be respected and verified before each transaction.

The performance of relational databases decreases as the amount of data increases, also high availability can't be guaranteed as the relational databases are based on consistency instead of availability [5]. Also, databases must be easily replicated as the majority of recent systems are distributed, and if we take for example a relational database in Oracle DB system, the replication needs an additional effort. Due to all those limits of SQL, each high organization has developed its own database management system that responds to its needs, those systems are classified as NoSQL databases [5].

The purpose of this study is to define a new syntax in order to migrate graphs from relational database to document database. This migration will make tasks easier for developers who will no longer have to copy those data to the new NoSQL created structures, it also will open the doors to exploit those data by recent algorithms. which is the main objective of our research project. In this paper, we are interested in data modelled by graphs, in order to apply graph clustering algorithms and especially spectral clustering.

This paper is organized as follows. Section 2 describes the background of the work and gives overview of the different NoSQL datastores and graph schemas. Section 3 details the syntax of the proposed algorithm to store graph schemas in document-oriented MongoDB database. Section 4 gives a comparison between the experimental results after storing graphs in both SQL and NoSQL databases. Section 5 closes the paper with conclusions and perspectives.

2 Background

2.1 Related work

The work presented in this paper is a part of a research project of knowledge extraction from data modeled by graphs [6, 7] where the main objective is the classify a set of individuals using graph approaches such as spectral clustering algorithms [8-11].

Many algorithms and approaches were developed to convert between schemas, models or databases. We cite for example the conversion between XML and object-relational model [12], the conversion between UML and XML [13, 14], the conversion between relational data and graph schemas [15] and recently the conversion of SQL databases to NoSQL [16] such as the conversion from MySQL to Cassandra which is a NoSQL column-oriented database [17].

In our context, no approaches were proposed to deal especially with data modeled by graphs, where the input and the output are graph schemas. In [16] the authors proposed a database migration from SQL to NoSQL database with MongoDB as a proposed use case to compare the performances insertion, selection and update queries.

Also, in [18] the authors investigated NoSQL document datastores in order to evaluate their suitability to replace relational databases in managing clinical data, their experimental results proved that NoSQL and XML are the perfect choices in term if speed, flexibility and scalability. Still in the field of health, another work [19] presented a NoSQL database to store health data instead of relational solutions whose proved their limits against volume and heterogeneity, the proposed model was based on a distributed document DBs and was implemented in the cloud environment to access the distributed properties, the experimental results proved that the NoSQL model surpassed the existing relational model in writing queries, flexibility and extensibility. We note that the authors in [19] compared their model to SQL Server and it would be better if they compared their model to an object-oriented model in Oracle databases for example, in order to validate their hypothesis.

Our contribution is a GEXF graph schema storage in MongoDB database, which can be seen as a conversion from XML schema to MongoDB database and benefit from the performances of MongoDB queries in terms of time, complexity and flexibility [20]. Thereafter, we are going to focus on the document-oriented databases, also called document stores. This category of databases is the nearest to the key value model, where the key is a unique ID that can be automatically generated or manually given, and the value is a document in JSON format in the case of MongoDB [5]. Document stores are a good solution for large data sets with changing states where no schema definition is forced which gives the possibility to store variable data and update the attributes of inserted documents dynamically.

2.2 NoSQL

Nowadays, data are provided from different sources, the amount of those data is still in progress, the need of real-time processing, also the majority of data are un-

structured. All those factors prove that data in the future could not be processed by traditional systems such as SQL databases. NoSQL has proved its performances against the big data challenges such as volume, velocity and variety.

Recently, the big companies in information and communications technology (ICT) started to develop their own systems that perform with their data, those systems are classified as NoSQL database management systems. In general, we can regroup those systems in four families; Key-value, document-oriented, column-oriented and graph-oriented databases.

Key-value databases. In this model, data are stored as pair of keys and values, where the key is a BLOB (Binary Large OBject). Key-value datastores use keys to handle data, which is similar to the use of the object-oriented structures such as maps, collections, or vectors. The basic structure of those databases gives them a high performance in term of time, especially for read and write queries [21]. Among the benefits of such datastores; the low time response and the ease of scalability. Redis and Amazon DynamoDB are two of the most familiar key-value databases.

Column-oriented database. This model is the closest to relational databases except that data are stored by columns rather than rows. Column datastores support high scalability which make them more performant in data mining field as the values of a single column are tend to have similar behaviors and can easily be classified in clusters [22]. In addition to Apache HBase, Cassandra are widely used in social networks' applications such as the case of Facebook. The strength of Cassandra is related to its implantation of both DynamoDB key-value store and BigTable [23] column-oriented store which adds a fastness data access and high storage capacity [3, 21].

Graph-oriented databases. Data are structured as graphs, where each data point is a vertex with properties, and the edges represent the similarities between each pair of data points having similar behaviors. This model is adapted to relational and object-relational models, as they all focus on the coupling between the stored data [3]. Graphs are widely used in different fields which gives high value to graph datastores, we cite for example the metabolic networks, protein-protein interaction networks, chemical structure graphs, gene clusters and genetic maps, in addition to websites and social networks in computer science. Neo4j is the most used graph database, it's used to store a high number of heterogeneous nodes liked with weighted edges, also it has its own querying language called Cypher which is less complicated that SQL. The experimental studies prove that Neo4j gives high performances in term of time for nodes with string typed properties, in the other hand it takes more time to convert strings to numeric values.

Document-oriented databases. Data are stored as semi structured documents such as XML (eXtensible Markup Language) or JSON (JavaScript Object Notation) files. Document datastores are flexible as they are schema-less and read/write queries does not verify the data before each transaction, all those factors produce less execution time compared to schema models such as relational or object-relational ones. MongoDB [24] is one of the most popular document datastores, used to manipulate an unlimited number of heterogeneous documents in a single database, with a high volume and low execution time. Thereafter, this work is going to focus in this category of NoSQL datastores in comparison with an SQL object-relational database.

2.3 Graph schema

Graphs as a structure are widely used in various fields; starting with computer science, we cite for example websites and social networks. Applications of graphs are also used outside the field of computer science, such as metabolic networks, chemical structure graphs, gene clusters and genetic maps [3].

With the growth of graphs applications, the storage of those structures has become a critical issue. Formerly, graphs were stored by their adjacency matrix or by their set of nodes and edges. Recently, the graphs are used to model heterogeneous data where nodes don't share the same schema or parameters, also the links between the nodes can carry information, the traditional storage solutions have become outdated and other solutions started to see the light in order to deal with graphs in the big data context. GEXF (Graph Exchange XML Format) is an open source document-oriented format based on XML and used to describe and store graphs and complex networks. Another recent solution for describing graphs and complex networks is Graph-oriented databases such as Neo4j [7].

A graph in GEXF is described by its set of nodes and edges with additional parameters as the color of the node, the 3D position and the weight of the edges.

The main issue with databases nowadays is that the majority of small and medium institutions still use SQL models, especially the Relational Databases and more precisely MySQL, which is a free Relational Database Management System (RDBMS) in addition to Oracle DB for both relational and object-relational models. In other hand, the recent algorithms for knowledge extraction and classification deal with heterogeneous data, in other words those approaches are designed for NoSQL databases [25]. In brief, the need is to develop a bridge between those two technologies to convert the existing models without redoing the conception of information systems and also migrate the existing data to NoSQL databases to get exploited by recent techniques and approaches (Fig. 1).

The graph schema construction is generated using the algorithm of conversion between relational data and graph schema published in [15].

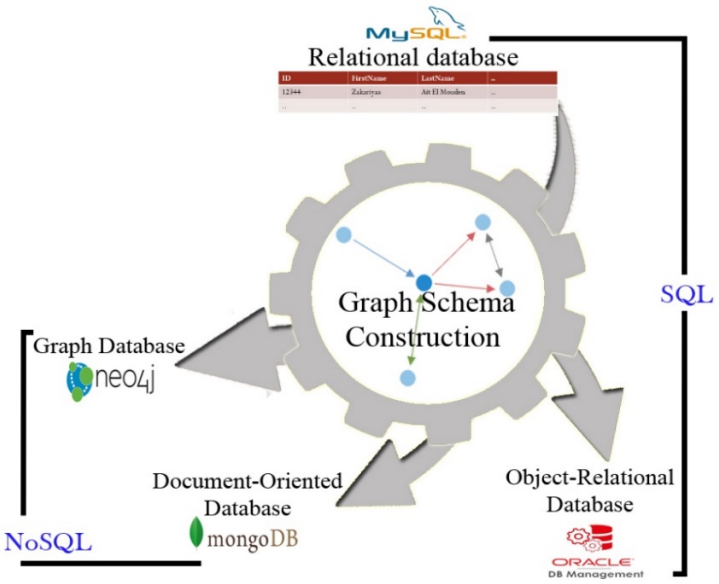

Fig. 1. Process overview.

3 Graph schema to MongoDB

3.1 Syntax

In graph schemas, a graph is defined by its description *D*, its set of vertices V and its set of edges *E*. Then for a graph *G* its schema $\hat{G}$ is given by the following syntax (1).

$$\hat{G} = D \,\&\, \{V; E\} \tag{1}$$

The symbol & is used to describe a concatenation of simple parameters and objects, where *D* is a simple parameter or a set of simple parameters that are extracted from inside the tag, while *V* and *E* are objects. The graph Description *D* is defined as follows (2).

$$D = T \,\&\, M \tag{2}$$

T is the default edge type, which can take two possible values (3).

$$T = \text{DIRECTED} \mid \text{UNDIRECTED} \tag{3}$$

In GEXG, a graph mode (*M*) can be static or dynamic where the nodes move in the 3D space according to time. The symbol | denotes an alternative. In our case, we consider the case where the graph is always static, so the Description in (2) is modified to the syntax (4).

$$D = \{T; \text{STATIC}\} \tag{4}$$

The set of nodes V in (1) is a list of the entire vertices of the graph G (5).

$$V = \{ v_1 , v_2 , \ldots . v_n \} \tag{5}$$

Where n is the order of the graph ($n = |V|$) and each vertex v_i ($i \in [1, n]$) is defined as follows (6).

$$v_i = [n] \ \& \ \{id \ [; l; s; c; p]\} \tag{6}$$

The value of the parameter *id* is a textual unique value that identifies each node. The parameter *l* describes the displayed label of the node, by default a node is labeled by its *id*, but in several cases other parameters are preferred to label the nodes. The parameter *s* is the visualization size, it's a numeric value that refers to the importance of the vertex in the graph. The parameter *c* is the color of the node in RGB encoding given as an object (7).

$$c = r \ \& \ g \ \& \ b. \text{ With } r, g, b \in [0, 255] \tag{7}$$

The last parameter *p* in (6) defines the position of the node in the 3D space, p is an object (8).

$$p = x \ \& \ y \ [\ \& \ z \ ] \tag{8}$$

x, *y* and *z* are real numeric values. *z* is an optional value, which is the case for all the parameters between [] in (6).

We move now to the set of edges E which contains the entire edges of the graph to link between the vertices of V.

$$E = \{ e_1 , e_2 , \ldots . e_k \} \tag{9}$$

Where k is the size of the graph ($k = |E|$) and each edge e_i ($i \in [1, k]$) is defined as follows (10).

$$e_i = [k] \ \& \ \{id \ ; src; trg \ [; w]\} \tag{10}$$

id is the unique parameter that identifies each edge. *src* is the id of the source node of the edge and *trg* is *id* of the target node. The parameter *w* is the weight of ne node in the case where *G* is a weighted graph.

3.2 Conversion to MongoDB syntax

We consider the function M_{db} that allows us to convert a graph schema to MongoDB syntax. The function M_{db} can take any type parameters. To convert the convert the entire graph schema we divide the conversion to multiple processes (11). We add that the function M_{db} is divisible (12 and 13).

$$\mathrm{M}_{db} (\hat{G}) = \mathrm{M}_{db} (D , \{V; E\}) \tag{11}$$

$$\mathrm{M}_{db} (\hat{G}) = \mathrm{M}_{db} (D) , \mathrm{M}_{db} (\{V; E\}) \tag{12}$$

$$M_{db}(\hat{G}) = M_{db}(D), M_{db}(V), M_{db}(E) \quad (13)$$

In MongoDB, the objects are separated with comas (,). We start with the conversion of the description D (14 and 15).

$$M_{db}(D) = M_{db}(T), M_{db}(M) \quad (14)$$

$$M_{db}(T) = \texttt{"defaultedgetype": "undirected" | "directed"} \quad (15)$$

As we already mentioned, we deal with static graphs, which means the value of $M_{db}(M)$ is a constant (16).

$$M_{db}(M) = \texttt{"mode": "static"} \quad (16)$$

Later on, we calculate the conversion of the set of vertices V (17). To do this, we calculate separately the conversion of each vertex (18).

$$M_{db}(V) = \{\alpha \,\&\, M_{db}(\{ v_1, v_2, .. v_n \}) \,\&\, \zeta\} \quad (17)$$

$$M_{db}(V) = \{\alpha \,\&\, (M_{db}(v_1), M_{db}(v_2), .., M_{db}(v_n)) \,\&\, \zeta\} \quad (18)$$

As we deal with a list of objects, it's important to respect the MongoDB syntax to create a list of nodes (vertices) where each element is an object of type node, the list opening is stored in the textual constant α (19) and the closing string is stored in the constant ζ (20).

$$\alpha = \texttt{"\"nodes\": {\"node\":["} \quad (19)$$

$$\zeta = \texttt{"]}\} \texttt{"} \quad (20)$$

Now for the conversion of the vertices, each vertex is converted separately as follows (21). The number of nodes n is ignored by the conversion function as it can be easily calculated (22 and 23).

$$M_{db}(v_i) = M_{db}([n] \,\&\, \{id\,[; l; v; c; p]\}) \quad (21)$$

$$M_{db}(v_i) = M_{db}(\{id\,[; l; s; c; p]\}) \quad (22)$$

$$M_{db}(v_i) = \{\&\, M_{db}(id), M_{db}(l), M_{db}(s), M_{db}(c), M_{db}(p) \,\&\}\} \quad (23)$$

The parameters *id* and *l* are simple data whose stand for id and label, the conversion of a simple data parameter is obtained with the following syntax (24).

$$\texttt{"parameter": "value"} \quad (24)$$

The parameter *s* is an object with one numeric value to describe the size of the node. The conversion can be described as follows (25).

$$M_{db}(s) = \texttt{"vize-size":} \\ \texttt{\{ "value": vize-size\_value \}} \tag{25}$$

For the parameter c which is an object with three integer values, the conversion can be obtained as follows (26).

$$M_{db}(c) = \texttt{"vize-color":} \\ \texttt{\{"r":r\_value,"g":g\_value,"b":b\_value \}} \tag{26}$$

The last parameter for a vertex is its position, which is an object with three values x, y and z as an optional value. In our case we deal only with 2D space (27).

$$M_{db}(p) = \texttt{"vize-position":} \\ \texttt{\{"x": x\_value, "y": y\_value \}} \tag{27}$$

Thereafter, for edges conversion and as an edge is an object with non-complex parameters, we can summarize the conversion as follows (28 and 29).

$$M_{db}(E) = \{\alpha' \,\&\, M_{db}(\{ e_1 , e_2 , .. e_k \}z) \,\&\, \zeta\} \tag{28}$$

$$M_{db}(E) = \{\alpha' \,\&\, M_{db}(e_1), M_{db}(e_2) , .. M_{db}(e_k) \;\&\, \zeta\ \} \tag{29}$$

With α' (30) and ζ (20) the opening and the closing constants successively.

$$\alpha' = \texttt{"\textbackslash"edges\textbackslash": \{\textbackslash"edge\textbackslash":[".} \tag{30}$$

Thereafter, for the conversion of the edges, each edged is converted separately as follows (31). The number of edges k is ignored by the conversion function as it can be easily calculated (32 and 33).

$$M_{db}(e_i) = M_{db}([k] \,\&\, \{id;\ s;\ t;\ [w]\}) \tag{31}$$

$$M_{db}(e_i) = M_{db}(\{id;\ s;\ t;\ [w]\}) \tag{32}$$

$$M_{db}(e_i) = \{\&\, M_{db}(id) , M_{db}(s) , M_{db}(t) , M_{db}(w) \;\&\} \tag{33}$$

The parameters *id, s* (source node), *t* (target node) and *w* (weight) are simple data whose can be expressed with the syntax (24).

3.3 Conversion Algorithm

The functions presented in the previous subsection can be summarized in the following algorithm (Fig. 2).

`graph_schema()` is the function that allows us to build a graph schema from a relational database. The graph schema is used generally to describe or visualize a set of tuples stored in an SQL database such as MySQL or Oracle db, more details about this function in [15]. The first part of the algorithm (from line 2 to line 7) takes a node as a parameter, then generates its MongoDB syntax from the calculated graph schema

in line 1, this part summarizes the instructions from (17) to (27) detailed in the previous subsection. The second part of the algorithm (from line 8 to line 13) takes an edge as a parameter and returns its MongoDB syntax using the instructions from (28) to (33) detailed in the previous subsection.

The combination of the two parts of the algorithm generates a MongoDB document to store the graph schema as document-oriented object which open doors to exploit those objects in several fields such as data science, data mining, data visualization and other big data branches.

Algorithm graph_storage

```
tmp :    String
G   :    A graph schema in GEXF format
mdb :    An empty MongoDB document
input    rdb : Table in a relational database
1)  G = graph_schema(rdb)
2)  tmp = "nodes: { node: ["      // init list of nodes
3)  foreach node n in G.nodes do
4)     tmp = tmp + { "id:" + id_value +
                     ", viz-size:{ value:" + vize-size_value +
                     "}, viz-color:{ r:" + r_value + ",g:" + g_value
                                         + ",b:" + b_value +
                     "}, viz-position:{ x:" + x_value + ",y:" + y_value +"}}"
5)  end foreach
6)  tmp = tmp + "]}"                  // close list of nodes
7)  add the content of tmp to mdb document
8)  tmp = "edges : { edge: ["   // init list of edges
9)  foreach edge e in G.edges do
10)    tmp = tmp + { "id:" + id_value +
                     ", source" : source_value +  ", target" : target_value +
                     ", weight" : weight_value +"}"
11) end foreach
12) tmp = tmp + "]}"              // close list of edges
13) add the content of tmp to mdb document
output   mdb : MongoDB document
```

Fig. 2. Conversion algorithm

4 Graph storage in SQL and NoSQL datastores

To highlight the difference between storing graph schemas in SQL and NoSQL, we propose the use of Oracle object relational-database 11g [26] for SQL model and MongoDB server 4.0 [20, 27] as NoSQL document-oriented database. The comparative study will focus on three major axes; First we compare the ease of programming

the query to store the same graph in both database models. Secondly, we compare the execution time needed to execute both queries built in the first axis. The third axis is the volume of the stored graph in the SQL and NoSQL databases.

4.1 Ease of programming and Variety

Oracle database uses SQL3 and later versions for object-relational programming. In SQL3 each data type is defined with data definition language before executing the insertion query which is a part of data manipulation language. The data definition language requires the verification of each data manipulation query to the predefined schema in data definition language which makes the query very sensible to any minor error in the syntax.

In the other hand, and as the majority of NoSQL databases, MongoDB does not require any data definition which makes NoSQL queries more flexible and easier to program, but the same SQL query (See Fig. 3) will be expressed with more characters in MongoDB as it is obliged to redefine parameters name before their values (See Fig. 4).

```
INSERT INTO graphs VALUES (
 graph(
   '2nd Degree graph', 'undirected',
   nodes(
       node('A112', 20.0, color(130,0,130),
                  position(145,-61,0)),
       node('B341', 22.0, color(230,10,10),
                  position(-212, 13,0))
   ),
   edges(
       edge('E0', 'A112', 'B341', 0.27))
));
```

Fig. 3. SQL Object-Relational query.

Fig. 4 shows the insertion query of the same graph schema of Fig. 4, we mention that an object in document-oriented model is called a document. As you can notice, MongoDB queries are usually longer than SQL queries, but MongoDB syntax is easier to explain as it is schema-less as we already mentioned.

```
db.graph.insert({
     "description": "2nd degree graph",
     "edgetype": "undirected",
     "nodes": {
             "node" : [
              {
                 "label": "A112",       "value": 20.0,
                 "position": { "x": 145, "y": -61, "z": 0 },
                 "color": { "r": 130, "g": 0, "b": 130 }
              },
               {
                 "id": "B341", "value": 22.0,
                  "color": [230, 10, 10]
                   "position": { "x": -212, "y": 13 },
              }
                    ]},
     "edges": {
             "edge": [
              {
                  "id": "E0", "weight": 0.27,
                  "source": "A112", "target": "B341"}
                    ]}
  });
```

Fig. 4. NoSQL MongoDB query.

For example, in the first node document (A112), the position value contains the three dimensions *x*, *y* and *z*, but in the second node document (B341) the parameter *z* is absent. The object color is composed of three integers for green, red and blue (RGB) which can be explained as a composed document in the node A112 or a table of integers which is the case in the node B341. In MongoDB, the parameter doesn't have to be present with the same name in the set of documents, we cite for example the parameter label and id in the nodes A112 and B341 respectively, but this difference must be taken in consideration in the manipulation queries. Also, an optional parameter doesn't have to be declared with NULL constraint and specify the value `NULL` in each transaction, we only ignore it.

4.2 Velocity and Volume

In this part, we evaluate the performance of the studied database management systems in term of velocity and volume; to do this we consider five built graphs. Then, foreach graph we measure the time needed to store the graph and also the volume of the stored graph. Table 1 presents the five used graphs sorted by their order (number of nodes) and their size (number of edges).

Table 1. Set of graphs used for measurments.

Graph	***G1***	***G2***	***G3***	***G4***	***G5***
Order |V|	10	20	40	80	80
Size |E|	10	40	100	200	400

The next two figures (Fig. 5 and Fig. 6) show the time needed to store the graph and its size in the database respectively.

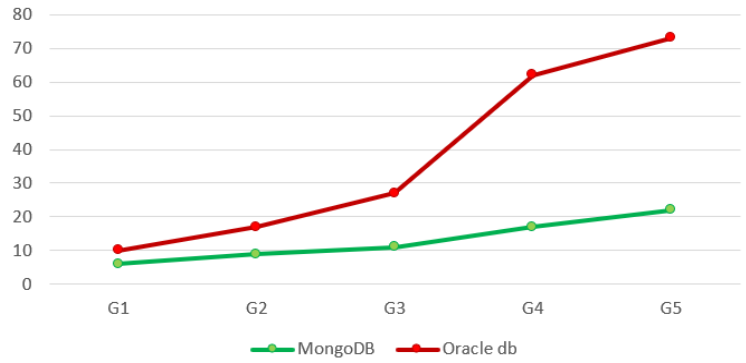

Fig. 5. Execution time to insert graphs in milliseconds (*ms*).

As MongoDB is schemaless, insertion queries don't have to be compared to any schema before insertion which makes the insertion faster than Oracle database or any other SQL DBs. Also, the experiments show that SQL databases are more sensible to data expansion in comparison to NoSQL databases that are created to store high volumes of heterogeneous data.

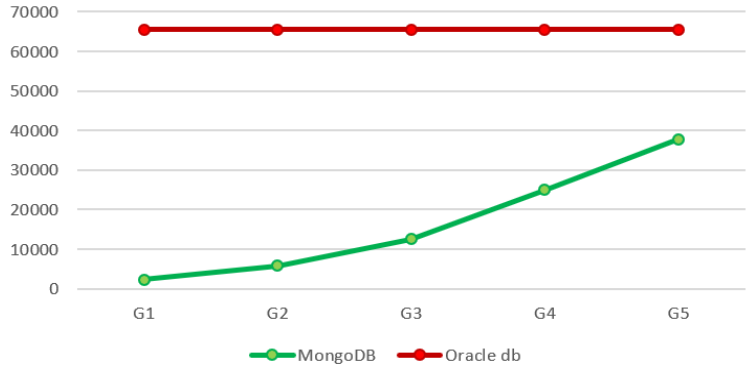

Fig. 6. Graph sizes in bytes.

Due to data compression, the five graphs occupied less space in document datastore in comparison to the object-relational database where the same storage size was allocated to store data with different sizes.

5 Conclusions and Perspectives

We have presented an algorithm to store graph schemas in document-oriented database. For the application, we have used MongoDB as a NoSQL database management system and Oracle db as SQL database, we have also used other tools to test our proposed process, such as XML for document representation of graph schemas, java to code the different functions of the process and Gephi for data visualization.

The purpose of this contribution is to move from SQL to NoSQL datastores without being obliged to redefine the structures and the constraints, neither being forced to manually migrate data from a system to another. This algorithm is meant to be applied after graph schema building from a relational database. As NoSQL models are different and each organization choses the model depending on their data, future works are needed to take us from an SQL model to any different NoSQL model and give each organization the choice to continue with their preferred NoSQL product with data preservation.

As perspectives, we plan to create the graph-oriented version of the algorithm and to develop a system that links all the developed approaches from reading data from SQL databases to graph schema creation and graph storage in different NoSQL models. The output of such algorithms is deployed by unsupervised clustering algorithms to extract knowledge from data modeled by graphs.

6 References

[1] El-Seoud, S. A. and El-Sofany, H. (2009). Schema Design and Normalization Algorithm for XML Databases Model. International Journal of Emerging Technologies in Learning (iJET), 4: 11-21 https://doi.org/10.3991/ijet.v4i2.768

[2] Grout, I. A. and da Silva, A. C. R. (2009). Remote laboratory description language based on xml. International Journal of Online Engineering (iJOE), 5: 25-34 https://doi.org/10.3991/ijoe.v5s1.1007

[3] Vicknair, C. et al. (2010). A comparison of a graph database and a relational database: a data provenance perspective. Proceedings of the 48th annual Southeast regional conference, ACM, Atricle No. 42 https://doi.org/10.1145/1900008.1900067

[4] Parker, Z., Poe, S. and Vrbsky, S. V. (2013). Comparing nosql mongodb to an sql db. Proceedings of the 51st ACM Southeast Conference, ACM Artciel No. 5 https://doi.org/10.1145/2498328.2500047

[5] Hecht, R. and Jablonski, S. (2011). NoSQL evaluation: A use case oriented survey. 2011 International Conference on Cloud and Service Computing, 336-341 https://doi.org/10.1109/csc.2011.6138544

[6] El Mouden, Z. A., Jakimi, A. and Hajar, M. (2019). An application of spectral clustering approach to detect communities in data modeled by graphs. Proceedings of the 2nd Inter-

national Conference on Networking, Information Systems & Security, ACM, Article No. 4 https://doi.org/10.1145/3320326.3320330

[7] Ait El Mouden, Z., Taj, R. M., Jakimi, A. and Hajar, M. (2018). Towards for Using Spectral Clustering in Graph Mining. International Conference on Big Data, Cloud and Applications. Communications in Computer and Information Science, 872: 144-159. https://doi.org/10.1007/978-3-319-96292-4_12

[8] Schaeffer, S. E. (2007). Graph clustering, Computer science review, 1: 7-64

[9] White, S. and Smyth, P. (2005). A spectral clustering approach to finding communities in graphs. in Proceedings of the 2005 SIAM international conference on data mining, 274-285 https://doi.org/10.1137/1.9781611972757.25

[10] Zelnik-Manor, L. and Perona, P. (2005). Self-tuning spectral clustering. Advances in neural information processing systems, 17:1601-1608

[11] Von Luxburg, U. (2007). A tutorial on spectral clustering," Statistics and computing, 17: 395-416 https://doi.org/10.1007/s11222-007-9033-z

[12] Machkour, M., Afdel, K. and Khamlichi, Y. I. (2016). A reversible conversion methodology: Between XML and object-relational models. in 2016 7th International Conference on Information and Communication Systems (ICICS), 270-275 https://doi.org/10.1109/iacs.2016.7476063

[13] Jensen, M. R., Møller, T. H. and Pedersen, T. B. (2003). Converting XML DTDs to UML diagrams for conceptual data integration, Data & Knowledge Engineering, 44: 323-346 https://doi.org/10.1016/s0169-023x(02)00142-8

[14] Gasevic, D., Djuric, V., Devedzic, V. and Damjanovi, V. (2004). Converting UML to OWL ontologies, in Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters, 488-489 https://doi.org/10.1145/1013367.1013539

[15] Ait El Mouden, Z. Jakimi, A. and Hajar, M. (2019). An Algorithm of Conversion Between Relational Data and Graph Schema. in International Conference Europe Middle East & North Africa Information Systems and Technologies to Support Learning, Smart Innovation, Systems and Technologies, 111: 594-602, https://doi.org/10.1007/978-3-030-03577-8_65

[16] Zhao, G., Lin, Q., Li, L. and Li, Z. (2014). Schema conversion model of SQL database to NoSQL, 2014 Ninth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, 355-362 https://doi.org/10.1109/3pgcic.2014.137

[17] Gopalan, M. G., Prasanna, C., Krishna, Y. S., Shanthini, B. and Arulkumar, A. (2017). MYSQL to cassandra conversion engine," in 2017 Third International Conference on Sensing, Signal Processing and Security (ICSSS), 503-508 https://doi.org/10.1109/ssps.2017.8071648

[18] Lee, K. K.-Y., Tang, W.-C. and Choi, K.-S. (2013). Alternatives to relational database: comparison of NoSQL and XML approaches for clinical data storage, Computer methods and programs in biomedicine, 110: 99-109 https://doi.org/10.1016/j.cmpb.2012.10.018

[19] Goli-Malekabadi, Z., Sargolzaei-Javan, M. and Akbari, M. K. (2016). An effective model for store and retrieve big health data in cloud computing. Computer methods and programs in biomedicine, 132: 75-82 https://doi.org/10.1016/j.cmpb.2016.04.016

[20] Botoeva, E., Calvanese, D., Cogrel, B. and Xiao, G. (2018). Expressivity and complexity of MongoDB queries. in 21st International Conference on Database Theory (ICDT 2018), Article N. 9, 1-23

[21] Kamal, S. H, Elazhary, H. H. and Hassanein, E. E. (2019). A Qualitative Comparison of NoSQL Data Stores. International Journal of Advanced Computer Science and Applications, 10(2): 330-338

[22] Esbai, R., Elotmani, F., and Belkadi, F. Z. (2019). Toward Automatic Generation of Column-Oriented NoSQL Databases in Big Data Context. International Journal of Online and Biomedical Engineering (iJOE), 15: 4-16 https://doi.org/10.3991/ijoe.v15i09.10433
[23] Chang, F. et al. (2008). Bigtable: A distributed storage system for structured data. ACM Transactions on Computer Systems (TOCS), 26, Article No. 4
[24] Chodorow, K. (2013). MongoDB: the definitive guide: powerful and scalable data storage. O'Reilly Media, Inc
[25] Moniruzzaman, A. and Hossain, S. A. (2013). Nosql database: New era of databases for big data analytics-classification, characteristics and comparison, arXiv preprint arXiv:1307.0191
[26] Loney, K. (2008). Oracle database 11g the complete reference. McGraw-Hill, Inc.
[27] Chodorow, K. (2011). Scaling MongoDB: Sharding, Cluster Setup, and Administration: " O'Reilly Media, Inc

7 Authors

Zakariyaa Ait El Mouden received his MSc in Computer Science and Distributed Systems from Ibn Zohr University, Agadir, Morocco. He is currently working toward his PhD in the Software Engineering & Information Systems Engineering research team, Faculty of Sciences and Techniques of Erchidia, Moulay Ismail University, Meknes, Morocco. His research interests include machine learning, graph analytics and NoSQL systems. (mouden.zakariyaa@outlook.com)

Abdeslam Jakimi is a Professor in the Faculty of Sciences and Techniques of Erchidia, Moulay Ismail University, Meknes, Morocco. He received his PhD in Computer Science and Telecommunications in 2009 from Mohamed V University, Rabat, Morocco. His current research interests include requirements engineering, user interface prototyping and design transformations, scenario engineering and big data. (ajakimi@yahoo.fr)

Article submitted 2020-05-12. Resubmitted 2020-06-26. Final acceptance 2020-06-27. Final version published as submitted by the authors.