Semi-Structured Schema for a Big Data (S-SSBD)

Shady Hamouda[1], Raed Sughayyar[1] and Omar Elejla[2]
[1]*Emirates College of Technology, Abu Dhabi, U.A.E.*
[2]*Faculty of Information Technology, IUG, Palestine*

Keywords: Semi-Structured, Big Data, Schema.

Abstract: Big data has become a crucial issue and has emerged as one of the most important technologies in the modern world. One of the concerns that need to be addressed regarding big data is the lack of a method that can handle a semi-structured data model with a flexible schema. None of the existing semi-structured models can handle a large volume of data with a flexible schema. Therefore, these requirements give significance to designing and develop a schema for semi-structured data. In addition, these issues and challenges have to be addressed by researchers when designing a method or algorithm to retrieve information from a large amount of data. This study aims to design a semi-structured data schema for big data applications. This study assessed the flexible schema feature and data based on the semi-structure features. The result showed that the proposed schema can cover and handle any change in the schema.

1 INTRODUCTION

Big data is a research area that is open for discussion (Stanescu, Brezovan, & Burdescu, 2016). Five main dimensions known as the 5V characteristics can describe big data (Pokorny, 2013; Rodríguez-Mazahua et al., 2016). These dimensions include volume, which concerns the data scalability as data grow every day; velocity, which represents the speed of data and indicates the critical point of big data by measuring the performance of transactions; variety, which represents the data format, such as structured, semi-structured and unstructured; veracity, which is concerned with data accuracy; and value, which describes the importance of the data. The elements of the taxonomy of big data are described in Figure 1.

Goli-Malekabadi, Sargolzaei-Javan, and Akbari (2016) considered variety as one of the most important dimensions for big data management because it describes the type and nature of data. Therefore, the present thesis focuses on converting structured data types to semi-structured data types.
In the relational data model, the data and the relationship among them are organized into tables (Zhao, Huang, Liang, & Tang, 2013). Each table has rows and columns, where rows represent records and columns represent the attributes. Moreover, the

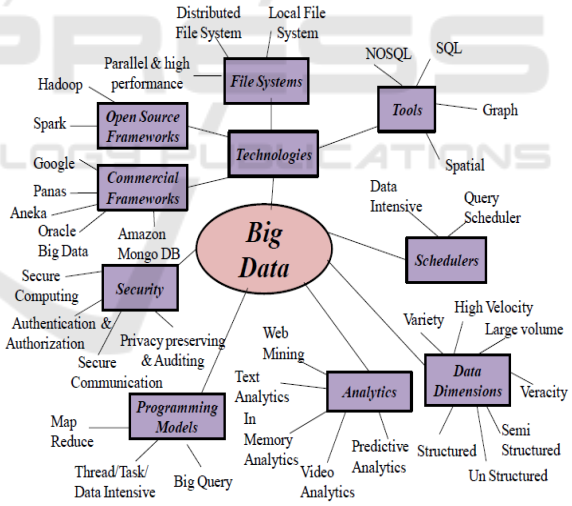

Figure 1: Big data taxonomy (Pokorny, 2013).

model uses SQL to access and process data (Neves & Bernardino, 2015). As Kune, Konugurthi, Agarwal, Chillarige, and Buyya (2016) mentioned, most organizations use a relational database as structured data to store and access their data. Thus, these organizations encounter problems with speed and the increasing size of data. For this reason, relational database application has become a bottleneck when the data increase and require more scalability (Moore, Qassem, & Xhafa, 2014).

Hamouda, S., Sughayyar, R. and Elejla, O.
Semi-Structured Schema for a Big Data (S-SSBD).
DOI: 10.5220/0010687600003064
In *Proceedings of the 13th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K 2021) - Volume 2: KEOD*, pages 202-209
ISBN: 978-989-758-533-3; ISSN: 2184-3228

On the other side, semi-structured data are emerging as one of the biggest data models for handling large amounts of data. Semi-structured captured and stored data have become huge and need to process flexible schemas and speed through a distributed system (Lombardo, Di Nitto, & Ardagna, 2012).

A semi-structured data format can store data in extensible markup language (XML), JavaScript Object Notation (JSON) and Binary JavaScript Object Notation (BSON). Moreover, a document-oriented database stores data in semi-structured data using the key-value concept. The value of a key can be any data type that gives database flexibility to store any kind of data (Zhao et al., 2013).

2 SEMI-STRUCTURED DATA

A semi-structured data is a form of structured data that can deal with any data type without a formal structure. Semi-structured data (Yaish & Goyal, 2013) are required to store and handle large amounts of data with flexibility schemas, which have emerged as one of the biggest data models for handling large amounts of data. Many models are proposed to handle semi-structured data as follows:

Numerous properties of a conceptual model for semi-structured data need to be addressed to handle semi-structured data. These properties can include no strict structure, no strict participation/instances, hierarchical structure, non-hierarchical structure, ordering, the irregular structure of data, disjunction, self-evolving, mixed content, abstraction, separation of structure and content explicitly, partial relationship/participation, heterogeneous, *n*-array relationship, inheritance, reuse potential, constraints, functional dependencies, symmetric relationship and recursive relationship (Ganguly & Sarkar, 2012). These properties are evaluated with the semi-structured data model presented in Table 1.

Ganguly and Sarkar (2012) found that none of these models can overcome all of the properties and requirements of the semi-structured data model. Only the GOOSSDM model can solve most properties of semi-structured requirements, but the integration of semantic web technologies still has a problem with all of these models. In addition, GOOSDM has drawbacks in representing schema-less data and data with a particular timestamp (Banerjee, Shaw, Sarkar, & Debnath, 2015).

Banerjee et al. (2015) likewise mentioned a conceptual model of big data transforming data model to a logical level, which is represented in XML, JSON, or BSON. JSON is a lightweight language of the data format used to store and exchange data and it follows the concept of JavaScript language ((Bansel & Chis, 2016). Moreover, JSON is a serialization format, which has schema-less data with all kinds of data types, such as string, number, list and array, and nested structure (Florescu & Fourny, 2013).

The features of JSON include easy preparation, string array analysis, suitability for semi-structured data, and cloud database (Mathew & Kumar, 2015). JSON can represent the logical model of big data better than XML because it represents data through a collection of key-value pairs that are more suitable for representing semi-structured data (Peng, Cao, & Xu, 2011). Moreover, JSON is useful and faster than XML in analyzing data (Florescu & Fourny, 2013). Binary-encoded serialization of JSON (BOSN) is used to support complex data types in different programming languages and supports flexible schema more than JSON (Bansel & Chis, 2016). Therefore, JSON plays an important role in representing semi-structured data in a NoSQL database, as it is lightweight and uses flexible data to deal with formatted semi-structured data, and can be compatible with most programming languages.

This challenge has led to the presentation of the "not only structured query language (NoSQL) database" as a new concept of database technology. One of the most powerful types of NoSQL databases is the document-oriented database that supports a flexible schema and stores data in a semi-structured format. The concept of NoSQL is to be fast and efficient in data processing in terms of scalability, reliability, agility, and performance (Grolinger, Higashino, Tiwari, & Capretz, 2013). Moreover, Hashem and Ranc (2016) mentioned that the NoSQL distribution supports flexible schema that will give flexible methods to handle and process semi-structured data. Therefore, NoSQL can be preferred to overcome scalability, high performance, and variability.

A document-oriented database is designed for storing, retrieving, and managing document-oriented or semi-structured data. The central concept of a document-oriented database is the notion of a document where the contents within the document are encapsulated or encoded in some standard format such as JSON or BSON, And XML. Storing data in JSON or BSON representation is easy by mapping the object structure of most of the programming languages directly into this representation. Consequently, a set of related documents are stored and represented by a collection that considers the table of the relational database. Each document can

Table 1: Evaluations of the semantic properties of the conceptual model with semi-structured models (Ganguly & Sarkar, 2012).

Properties / Model Name	ERX	ORA-SS	XER	EReX	XUML	XSEM	GOOSSDM	GN-DTD
No strict structure	F	F	F	X	F	F	F	F
No strict participation/instances	F	F	F	X	F	F	X	P
Hierarchical structure	F	F	F	F	F	F	F	F
Non-hierarchical structure	X	P	X	F	P	F	F	P
Ordering	F	F	F	X	P	F	F	F
The irregular structure of data	X	X	X	F	F	F	F	F
Disjunction	X	F	X	F	F	X	F	X
Self-evolving	X	F	X	X	X		F	X
Mixed content	X	F	F	F	F	F	F	X
Abstraction	X	F	F	X	X	X	F	F
Separation of structure and content explicitly	X	F	F	X	X	X	F	F
Partial relationship/participation	X	F	F	F	F	F	F	F
Heterogeneous	X	F	X	F	F	X		F
N-array relationship	X	F	X		F	X	F	F
Inheritance	X	X	X	F	F	X	F	F
Reuse potential	X	F	F	X	X	F	F	F
Constraints	F	F	X	F	F	F	F	P
Cardinality	X	F	X	X	X	X	X	F

F=fully supported; X = not fully supported; P=partially supported
ERX:Entity relational for XML;ORA-SS:object relationship attribute model for semi-structured data; XER:Extensible ER; EReX:Entity relational extended to XML;XUML:Executable of Unified Modeling Language;XSEM:Conceptual model for XML;GOOSSDM:Graph object-oriented semi-structured data model.

be identified by a unique key that is known to the user or can be automatically created by the database. This ID becomes an index for the document or can create other indexes depending on the application requirements to speed up the query process.

Relationships between the collections can be identified in two ways: embedding the document and referencing it. In the embedded document, the document can contain another document. This model can lead to the de-normalization of the database. Reference documents consider the relationships of the relational database, which shows the relationships between the collections and the foreign key. Therefore, a document-oriented database is used to achieve the objective of this study.

However, there are many challenges and complex problems for design schema for document-oriented databases, also there are no tools or methodology available to support a good schema (Mior, Salem, Aboulnaga, & Liu, 2017). Moreover, there is still a lack of strategies for conceptually representing the data model for a document-oriented database (Guimaraes et al., 2015). Therefore, the data model can be used for big data application still has some issues and challenges such as: how to design a schema with association relationship and use query effectively (Mason, 2015).

3 ALGORITHM TO MAPPING STRUCTURE DATABASE (USING ENTITY-RELATIONSHIP) SCHEMA TO SEMI STRUCTURE SCHEMA

In large data, conceptual models can be important for developers to consider and management perspectives because they can describe the structure and application semantics. Also, conceptual modeling presents challenges for large data applications (Storey & Song, 2017). The entity-relationship diagram is a concept of relational database design. It describes the design

Table 2: Symbols and type of algorithm.

Model	Type	Notions	Descriptions
ER Schema	Strong Entity	Ei	Ei….n (i=1 to n)
	Weak Entity	WE	
	Attribute	A	Ai….An (i=1 and n number of attributes)
	Multi-Value attribute	Mv	
	Relationship	R	Relationships can be one to one (1:1) or one to many (1:N) or many to many (M:M)
S-SSBD	Collection	C	
	Document	D	
	Key-value	K	
	Embedded document	E_m	The embedded model applies between two entities
	Reference document	R_f	Reference model applies between two entities
	Array	[ ]	Array data type

of the conceptual schema, which can be assigned to another data schema, and transform to semi-structured data formatted, thus an algorithm is proposed to map the ER schema to semi-structured schema.

The component of ER schema is entities, attributes, and relationships. We represent entity by E and the series of entities will be $E_i....E_{n\ (i=1\ to\ n)}$, the number of attributes represented by use $A_j.....A_{(j=1\ to\ n)}$, and also R represents the type of relationships (1:1) or (1:N) and (M: M).

Table 2 notions use to map ER schema to S-SSBD as the following algorithm:

3.1 Algorithm of Mapping ER Schema to S-SSBD

The study describes how to map the ER schema to the S-SSBD. This study highlights the mapping algorithm presented by (Hamouda & Zainol, 2019; Hamouda, Zainol, & Anbar, 2019).

Input: ER Schema
Output: S-SSBD
1: BEGIN
2: for each strong entity, $Ei_{(i=1…n)}$ do
3: create new collection $Ci_{(i=1….n)}$ where n= number of collection for each entity
4: for each weak entity, $WE_{i(i=1…n)}$ do
5: create $WE_{i(i=1…n)}$ as embedded documents into belong strong entity ($WE_i\ E_m \subseteq E_i$)
6: end for
7: for each multi-value attribute, "MVi" do
8: store multi-values as array data type into belong strong entity $\forall\ MV_{i(i=1…n)}\ [] \subseteq Ei$
9: end for
10: for each (1:1) relationship between two entities (E1 and E2) do
11: If E1 datasets size is less than 16 MB and no other relationship with other entity
12: E1 store as an embedded document into E2 ($E1\ E_m \subseteq E2$)
13: else
14: apply reference document between E1 and E2 ($E1\ R_f \subseteq E2$)
15: end if
16: end for
17: for each (1:M) relationship between two entities (E1 and E2) do
18: if M dataset size is less than 16 MB records then
19: M side store as an embedded document into 1 side ($E2_{(N)}\ E_m \subseteq E1_{(1)}$)
20: else
21: apply reference document between E1and E2 ($E1\ R_f \subseteq E2$)
22: end if
23: end for
24: for each (M: M) relationship between two entities (E1 and E2) do
25: for E1 side do
26: create array data type into an embedded document
27: store the primary key of E2 with other related attributes
28: Update E1 with the embedded document of E2 ($E2 : \{[E_m]\} \subseteq E1$)
29: end for
30: for E2 do
31: create array data type into an embedded document
32: store the primary key of E1 with other related attributes
33: Update E2 with the embedded document of E1 ($E1 : \{[E_m]\} \subseteq E2$)
34: end for

35: end for
36: END

The pseudo-code of the above algorithm is describing how to map the ER schema to S-SSBD. It takes the ER schema as input and mapping the S-SSBD as output. This algorithm was applied to the case study to generate an S-SSBD.

4 CASE STUDY 1: W3SCHOOL SCHEMA

This case study is the schema of W3Schools Web site (http://www.w3schools.com/). This schema has characteristics that are completely different from those typically used in most applications,
such as integration restrictions, relationships, and different data types.

The W3school schema can be similar to other businesses that have products with categories and these products have suppliers and then offer these products to customers for ordering and shipping. Therefore, this schema is chosen to implement mapping from a relational database to a document-oriented database through S-SSBD and then evaluate how S-SSBD can cover any new business requirements when changing the schema.

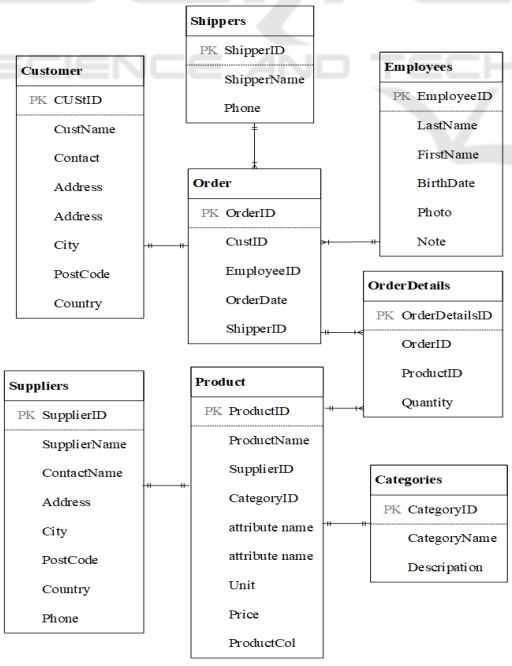

Figure 2: Entity relational schema for W3schools (Rocha, Vale, Cirilo, Barbosa, & Mourão, 2015).

As we can see from the schema, the product has categories and suppliers, and an order has order details and shipping to the customer through the employees. Therefore, S-SSBD applied to map the above schema, it creates product collection and store categories and suppliers as embedded documents, and also, creates order collection and stores the order details with shipping as embedded documents, moreover, these orders and products will manage through the collection of employees. The algorithm of mapping ER schema to S-SSBD were applied to the schema shown in Figure 2, and the output of this schema, presented in Figure 3, is as follows:

i. Created new collections for the main strong entity, which are PRODUCT, ORDER, and EMPLOYEES.

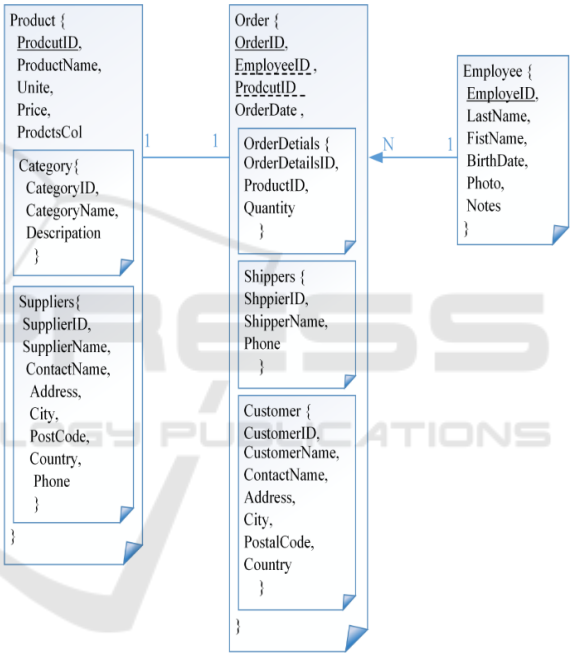

Figure 3: The S-SSBD for W3schools.

ii. Mapped the relationship between PRODUCT and CATEGORY by store CATEGORIES as embedded documents in the PRODUCT
iii. collection. Also, mapped the relationship between PRODUCT and SUPPLIERS by store SUPPLIERS as embedded documents in the PRODUCT collection
iv. Mapped the relationship between ORDER and ORDERDETAILS by creating an embedded document for ORDERDETAILS in the ORDER collection. Also, the relationship between ORDER and SHIPPERS was mapped to create embedded documents for SHIPPERS in the ORDER collection.

v. Mapped the relationship between ORDERS and CUSTOMERS by creating an embedded document for CUSTOMERS in the ORDERS collection.

The evaluation of this case study assessed the flexibility of the schema by checking whether the features of the S-SSBD were compatible with any business requirement changes.

5 EVALUATION: FLEXIBILITY OF THE SCHEMA

The flexibility of the schema refers to the evaluation of the capability to meet and respond to business change requirements during the development process (Rathor, Batra, & Xia, 2016). Accordingly, this evaluation tested the flexibility of the S-SSBD using the W3schools schema in Figure 3 to determine whether it could keep up with new business requirements. In a relational database, the new field should add to all table records to change its schema. By contrast, an S-SSBD allows the addition or alteration of the data for a specific document in any structure without changing the database schema. An S-SSBD thereby permits the application to use the required data while ignoring the unrequired data.

In the W3schools schema 2, the relationship between PRODUCTS and SUPPLIES is one to one. In this case, the W3schools schema needs to change the business requirements by allowing one PRODUCT to have many SUPPLIES or each SUPPLIER to provide many PRODUCTS; however, this new requirement creates difficulty in changing the relational database schema. To incorporate this requirement in the relational database, a new table should be added to allow the PRODUCT to have many SUPPLIERS, as shown in Figure 4. The relational database schema will not allow the same PRODUCT to have many SUPPLIERS because it is fixed and has change constraints.

The previous scenario indicates that the change required will affect the database schema, query level, and reporting level. Given that the relational database needs to change, all queries related to PRODUCT and SUPPLIER need to be redesigned and recoded. By contrast, an S-SSBD supports a flexible schema with semi-structured data that can add or change relationships between entities without adverse effects. In the previous case in Figure 3, the relationships between PRODUCTS and SUPPLIERS can be changed without affecting the schema because the PRODUCT collection stores the SUPPLIERS as embedded documents and lists SUPPLIERS for each product. In an S-SSBD, mapping one-to-one or

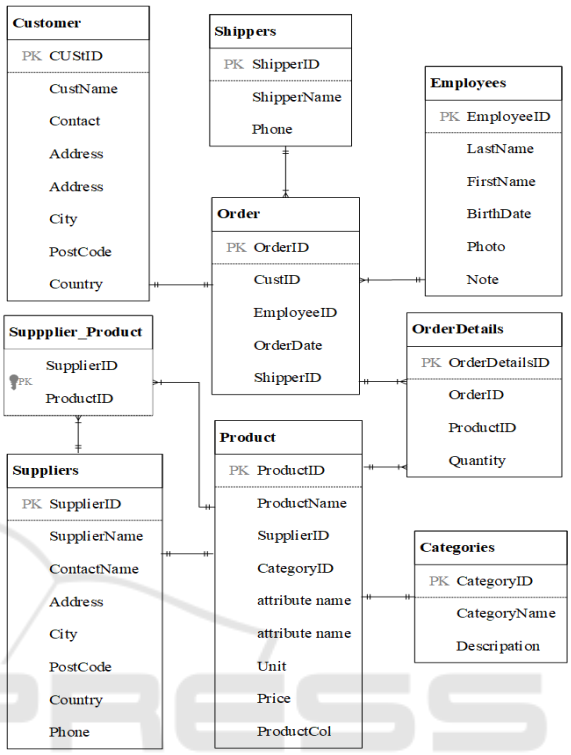

Figure 4: ER schema after changing the relationship between product and supplier.

one-to-many relationships can occur via the embedded documents. Therefore, this schema can store the many SUPPLIES as embedded documents into each PRODUCT, as shown in Figure 3. In addition, the second evaluation assessed the flexible schema feature and data based on the semi-structure features.

6 CONCLUSION

A semi-structured data can be formatted in a document in a way that is more useful than a table when a large amount of data is available. The proposed schema provides features for the conceptual representation of a document-oriented database. For example, it presents a flexible schema by allowing the application to change or update business requirements over time, and it represents relationships as embedded and reference documents. Additionally, semi-structured data with a flexible schema can handle two data dimensions of big data—

that is, a semi-structure and a large volume of data. In future work, this study can be extended to migrate the structure database to a document-oriented database using the proposed schema of semi-structured data.

REFERENCES

Banerjee, S., Shaw, R., Sarkar, A., & Debnath, N. C. (2015). *Towards logical level design of Big Data.* Paper presented at the 2015 IEEE 13th International Conference on Industrial Informatics (INDIN).

Bansel, A., & Chis, A. E. (2016). *Cloud-Based NoSQL Data Migration.* Paper presented at the 2016 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP).

Florescu, D., & Fourny, G. (2013). JSONiq: The history of a query language. *IEEE internet computing, 17*(5), 86-90.

Ganguly, R., & Sarkar, A. (2012). Evaluations of Conceptual Models for Semi-structured Database System. *International Journal of Computer Applications, 50*(18).

Goli-Malekabadi, Z., Sargolzaei-Javan, M., & 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.

Grolinger, K., Higashino, W. A., Tiwari, A., & Capretz, M. A. (2013). Data management in cloud environments: NoSQL and NewSQL data stores. *Journal of Cloud Computing: Advances, Systems and Applications, 2*(1), 1.

Guimaraes, V., Hondo, F., Almeida, R., Vera, H., Holanda, M., Araujo, A., Lifschitz, S. (2015). *A study of genomic data provenance in NoSQL document-oriented database systems.* Paper presented at the Bioinformatics and Biomedicine (BIBM), 2015 IEEE International Conference on Bioinformatics and Biomedicine.

Hamouda, S. and Zainol, Z. (2019). Semi-Structured Data Model for Big Data (SS-DMBD). In *Proceedings of the 8th International Conference on Data Science, Technology and Applications - DATA,* ISBN 978-989-758-377-3; ISSN 2184-285X, pages 348-356. DOI: 10.5220/0007957603480356.

Hamouda, S.; Zainol, Z. and Anbar, M. (2019). A Flexible Schema for Document Oriented Database (SDOD). In *Proceedings of the 11th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management - KEOD,* ISBN 978-989-758-382-7; ISSN 2184-3228, pages 413-419. DOI: 10.5220/0008353504130419

Hashem, H., & Ranc, D. (2016). *Evaluating NoSQL document oriented data model.* Paper presented at the Future Internet of Things and Cloud Workshops (FiCloudW), IEEE International Conference on Future Internet of Things and Cloud Workshops.

Kune, R., Konugurthi, P. K., Agarwal, A., Chillarige, R. R., & Buyya, R. (2016). The anatomy of big data computing. *Software: Practice and Experience, 46*(1), 79-105.

Lombardo, S., Di Nitto, E., & Ardagna, D. (2012). *Issues in handling complex data structures with noSQL databases.* Paper presented at the Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), 2012 14th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing.

Mason, R. T. (2015). *NoSQL databases and data modeling techniques for a document-oriented NoSQL database'.* Paper presented at the Proceedings of Informing Science & IT Education Conference (InSITE).

Mathew, A. B., & Kumar, S. M. (2015). *Analysis of data management and query handling in social networks using NoSQL databases.* Paper presented at the Advances in Computing, Communications and Informatics (ICACCI), 2015 International Conference on Advances in Computing, Communications and Informatics.

Mior, M., Salem, K., Aboulnaga, A., & Liu, R. (2017). NoSE: Schema design for NoSQL applications. *IEEE Transactions on Knowledge and Data Engineering, 29*(10), 2275-2289.

Moore, P., Qassem, T., & Xhafa, F. (2014). *'NoSQL'and Electronic Patient Record Systems: Opportunities and Challenges.* Paper presented at the P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2014 Ninth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing.

Neves, P., & Bernardino, J. (2015). *Big Data Issues.* Paper presented at the Proceedings of the 19th International Database Engineering & Applications Symposium.

Peng, D., Cao, L.-d., & Xu, W.-j. (2011). Using JSON for data exchanging in web service applications. *Journal of Computational Information Systems, 7*(16), 5883-5890.

Pokorny, J. (2013). NoSQL databases: a step to database scalability in web environment. *International Journal of Web Information Systems, 9*(1), 69-82.

Rathor, S., Batra, D., & Xia, W. (2016). What Constitutes Software Development Agility?

Rocha, L., Vale, F., Cirilo, E., Barbosa, D., & Mourão, F. (2015). A Framework for Migrating Relational Datasets to NoSQL1. *Procedia Computer Science, 51*, 2593-2602.

Rodríguez-Mazahua, L., Rodríguez-Enríquez, C.-A., Sánchez-Cervantes, J. L., Cervantes, J., García-Alcaraz, J. L., & Alor-Hernández, G. (2016). A general perspective of Big Data: applications, tools, challenges and trends. *The Journal of Supercomputing, 72*(8), 3073-3113.

Stanescu, L., Brezovan, M., & Burdescu, D. D. (2016). *Automatic mapping of MySQL databases to NoSQL MongoDB.* Paper presented at the Computer Science and Information Systems (FedCSIS), 2016 Federated Conference on Computer Science and Information Systems.

Storey, V. C., & Song, I.-Y. (2017). Big data technologies and Management: What conceptual modeling can do. *Data & Knowledge Engineering, 108*, 50-67.

Yaish, H., & Goyal, M. (2013). *A multi-tenant database architecture design for software applications.* Paper presented at the Computational Science and Engineering (CSE), 2013 IEEE 16th International

Conference on Computational Science and Engineering.

Zhao, G., Huang, W., Liang, S., & Tang, Y. (2013). *Modeling MongoDB with relational model.* Paper presented at the Emerging Intelligent Data and Web Technologies (EIDWT), 2013 Fourth International Conference on Emerging Intelligent Data and Web Technologies.