A Flexible Schema for Document Oriented Database (SDOD)

Shady Hamouda[1,2], Zurinahni Zainol[2] and Mohammed Anbar[3]
[1]*Emirates College of Technology, Abu Dhabi, U.A.E.*
[2]*School of Computer Sciences, Universiti Sains Malaysia, 11800 USM, Penang, Malaysia*
[3]*National Advanced IPv6 Centre (NAv6), Universiti Sains Malaysia, 11800 USM, Penang, Malaysia*

Keywords: Schema, Document-Oriented Database, NoSQL.

Abstract: Big data is emerging as one of the most important crucial issues in the modern world. Most studies mention that a relational database cannot handle big data. This challenge has led to the presentation of the not only structured query language (NoSQL) database as a new concept of database technology. NoSQL supports large volumes of data by providing a mechanism for data storage, retrieval and more flexible than a relational database. One of the most powerful types of NoSQL database is the document-oriented database. Recently, many software developers are willing to migrate from using relational databases to NoSQL database because of scalability, availability, and performance. The document-oriented database has challenged as to how to obtain an appropriate schema for the document-oriented database. The existing approach to migrate a relational database to a document-oriented database does not consider all the properties of the former, especially on how to handle various types of relationships. This research proposed a flexible schema for a document-oriented database (SDOD). This study evaluated the development of agility based on the schema of a document-oriented database and query execution time. The evaluation verifies the reliability of the proposed schema.

1 INTRODUCTION

The demand to migrate from a relational database to a document-oriented database has increased because of the complex structure of data and the different data types (El Alami & Bahaj, 2016; Hanine et al., 2016; Stanescu et al., 2017). Therefore, moving from table to document will be more efficient. The idea of migration from a relational database to a document-oriented database is conducted by migrating the data without loss or changing any structure of the original database.

However, there are many challenges and complex problem for design schema for NoSQL database such as there are no tools or methodology available to support a good schema (Mior et al., 2017). Also, there is still a lack of strategies for conceptually representing the data model for a document-oriented database (Guimaraes et al., 2015). Moreover, NoSQL databases offer many challenges such as de-normalization is required for high performance in NoSQL databases and repositories documents (Mior, 2014).

According to Zhao et al. (2014), normalization and de-normalization are some of the variables that must be considered when designing a schema, as it can measure the schema quality. Mehmood et al. (2017), found the evaluation schema quality can be determined through the performance of normalization and de-normalization of the documents.

The problem is that an RDBMS data model based on the relational model, and has a fixed schema with structured data, whereas the NoSQL can have any data format (Yoon et al., 2016). Moreover, there is no schema for a document-oriented database can recognise table type, document structure, and forming documents (Kanade et al., 2014; Moore et al., 2014) due to the various ways of storage, management and implementation in NoSQL. Besides, Tauro et al. (2012) and Liang et al. (2015) mentioned that the relational database and NoSQL are different models, which make difficulties to migrate data from the relational model to the NoSQL models.

In fact, the data models of NoSQL systems are very complex as there are no tools available to represent a scheme for NoSQL databases (Mohan,

Hamouda, S., Zainol, Z. and Anbar, M.
A Flexible Schema for Document Oriented Database (SDOD).
DOI: 10.5220/0008353504130419
In *Proceedings of the 11th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K 2019)*, pages 413-419
ISBN: 978-989-758-382-7

2013). According to Mohan (2013), there is no standard schema represent the data modeling of the document-oriented database. Therefore, designing a schema and implementing the document-oriented database for big data applications are needed (Guimaraes et al., 2015; Ogunyadeka et al., 2016; Storey & Song, 2017). Also, modeling and querying data in a document-oriented database is important, as, it describes the rules and function of the system (Atzeni et al., 2016). Therefore, it is necessary to define a schema for the underlying principles of a document-oriented database.

However, Mior et al. (2017) finding, although the schema of a document-oriented model is flexible, it is still necessary when designing the schema to decide whether data is normalizing or de-normalizing it before applying to a document-oriented database application. Moreover, González-Aparicio et al. (2017) observe that the normalization of the data model is one of the important research issues and there are no standard principles of normalization in the NoSQL database.

In addition, there is no any a variable method to normalize or de-normalize data to define an embedded and reference approach for handling the various types of relationships (Guimaraes et al., 2015, Khan and Mane, 2013, Hanine et al., 2016, Mehmood et al., 2017). According to Mehmood et al. (2017), normalization and de-normalization are the two variables that must be considered when designing a schema. These variables can affect the performance and storage effectively as the databases grow very quickly. Therefore, this paper overcomes the issues presented above by designing a schema for a document-oriented database.

2 THE PROPOSED SCHEMA FOR DOCUMENT ORIENTED DATABASE (SDOD)

SDOD improve the concepts and data model of the document-oriented database which present in the previous studies Arora and Aggarwal (2013); Atzeni et al., (2016); Bhogal and Choksi (2015); Hashem and Ranc (2016) by providing features and specification to design schema for a document-oriented database. This study aims to design a schema for the document-oriented database based on ER Model.

2.1 SDOD Specification

The SDOD normalize the data redundancy through two concepts; first is embedded document de-normalization the relationships by the store a subdocument into super-document collections, and second is reference document that can be used to normalization the relationship through linked the collections between each other's using a foreign key.

The component of ER schema is entities, attributes, and relationships. This research represents entity by Collection (C) and, attributes represent by use Key-value (K), and the type of relationships (1:1), (1: N), and (M: M) is represented through embedded and reference document. This section converts ER specification to the SDOD specification based on the following Table 1.

Table 1: Specification of SDOD.

ER Specification	SDOD Specification	Descriptions and conditions
Strong entity	Strong Entity {}	Each Strong entity is representing by a collection with two braces.
Weak entity	Strong Entity {Weak entity {}}	The weak entity is described by using embedded document into strong entity.
Hierarchical Entity	HigherCollection {k1.ki,lower collection[{k1..kj}]}	The hierarchical entity is represented by the higher hierarchical entity with all the related attributes and the lower collection store as embedded documents.
Ordinary attribute	{K1,…..,Ki}	The attributes of each entity are described by using K, which represents the name of the attribute. These attributes are listed inside two braces with a comma in between.
Composite attribute	CA{ K1, …,Ki }	CA→ the name of the composite attribute Ki → a set of attributes names listed into the document with a comma between them i → number of composite attributes
Multi-valued attribute	MV[K1,…,Ki]	The multi-valued attribute is described by the name of this attribute with an array data type and [K1..Ki] represents all of the multi-values.
Derived attribute	K#	The derived attribute is described by a hash after the attribute K.

Table 1: Specification of SDOD. (cont.)

ER Specification	SDOD Specification	Descriptions and conditions
Attribute of the relationship	(K1,….,Ki)	The attributes of relationships are added to relationship document.
Primary key	K	The primary key is represented by an underline of the attribute.
Foreign key	K	The foreign key is represented by a dashed line of the attribute.
Unary relationship	K@	The unary relationship is described by adding @ after the attribute K and the name of the relationship.
One-to-one relationship (1:1)	Embedded document	The 1:1 relationship is representing by using embedded document into the related collection.
	Reference document	if more relationships exist or the dataset of one side will exceed 16 MB (as the document size is 16 MB) , then this relationship should be represented by using the reference document.
One-to-many relationship (1:M)	Embedded document	The 1:M relationship is described through the embedded document or reference document depending on the size of the many relationship side . If the many relationship side is small if M dataset is not exceed 16 MB or less than a tens of thousands/hundreds of thousands/ millions records, and also, there is no any other relationship. Thus, 1:M is represent through embedded document, this embedded document store into the related document.
	Reference document	If the dataset of the many relationship side is exceed 16 MB or more than tens of thousands/hundreds of thousands/millions records. The 1:M relationship is described as a reference between two entities through foreign key.
Many-to-many relationship (M:M)	Collection1 name [{K1,…. Ki},….., Collection2 name{K1,…. Ki}]	The M:M relationship describes the references between two entities. This type of relationship is identified by creating two collections named: Collection1 and Collection 2, and then storing the primary key of the related collection of the document of Collection 1 inside the collection side of Collection 2, and vice versa. If store the document of Collection 1 inside Collection 2, then the name of the document root becomes Collection 1.

The principal concept of the NoSQL data model involves storing data as pairs of key–value. The key is considered as the field of a table in a relational database, while the value associated with the key can be any type of data structure. Unlike the case in the relational database, all the field data should have the same structure and may have a null value. Moreover, this pair of key-value can be collected into a document that can represent the field with the record in the relational database.

Consequently, a set of related documents are stored and represented by a collection, which considers the table of the relational database. Each document can be identifying by a unique key 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 identifying in two ways: embedded and reference document.

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. A document-oriented database store data using key-value concept into an organized document in which each key is used to store a value and this value can be identified by a key. This concept can be represented by a JavaScript Object Notation (JSON) object. A 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 type, such as string, number, list and array and nested structure (Florescu & Fourny, 2013).

According to Soransso and Cavalcanti (2018), the structures of the document are able to deal with collections of heterogeneous documents. The document in a collection is a self-describing structure allows finding the intended data/content through specific parts of these documents.

3 CASE STUDY: W3SCHOOL SCHEMA

The 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 implements of mapping from a relational database to a document-oriented database through SDOD, and then evaluates how SDOD can cover any new business requirements when changing the schema.

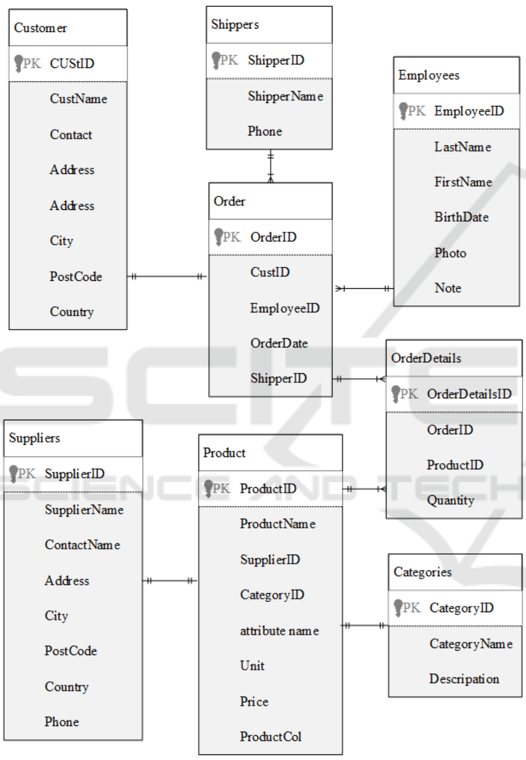

Figure 1: Entity relational schema for W3schools (Rocha et al., 2015).

As we can see in the above schema, the product has categories and suppliers, and the order has order details and shipping to the customer through the employees. Therefore, SDOD applied to map the above schema Figure 1. It creates product collection and store categories and suppliers as embedded documents. Also, creates order collection and stores the order details with shipping as embedded documents. Moreover, these order and products can be managing through the employee collection. The SDOD of mapping the above schema can be as shown following figure.

```
Order {
"OrderID" ,
"EmployeeID" ,
"OrderDate" ,
    "OrderDetials" : {
        "OrderDetailsID" ,
        "ProductID" ,
        "Quantity"
    },
    "Shippers" : {
        "ShppierID",
        "ShipperName",
        "Phone"
    },
    "Customer" : {
        "CustomerID",
        "CustomerName",
        "ContactName",
        "Address",
        "City",
        "PostalCode",
        "Country"
    }
}

Employee {
    "EmployeID",
    "LastName",
    "FistName",
    "BirthDate",
    "Photo",
    "Notes"
}

Product {
    "ProdcutID",
    "ProductName",
    "Unite",
    "Price",
    "ProdctsCol",
    "Category" : {
        "CategoryID",
        "CategoryName",
        "Description"
    },
    "Suppliers" : {
        "SupplierID",
        "SupplierName",
        "ContactName",
        "Address",
        "City",
        "PostCode",
        "Country",
        "Phone"
    }
}
```

Figure 2: SDOD for W3schools.

4 EVALUATION DEVELOPMENT OF AGILITY

Development agility refers to the capability to meet and respond to the business change requirements during the development process (Rathor et al., 2016). Agile methodologies can enable organizations to achieve flexibility in software development for managing unpredictable and changing conditions (Maruping et al., 2009) .

This section evaluates the development agility to identify business change requirements using the W3Schools schema in Figure 1. This schema has characteristics that are completely different from those typically used in most applications, such as integration restrictions, relationships, and different data types.

In a relational database, the new field should be added to all table records to change its schema. By contrast, an SDOD allows adding or changing data for a specific document in any structure without changing the database schema. An SDOD allows the application to use the required data and ignore the unrequited data.

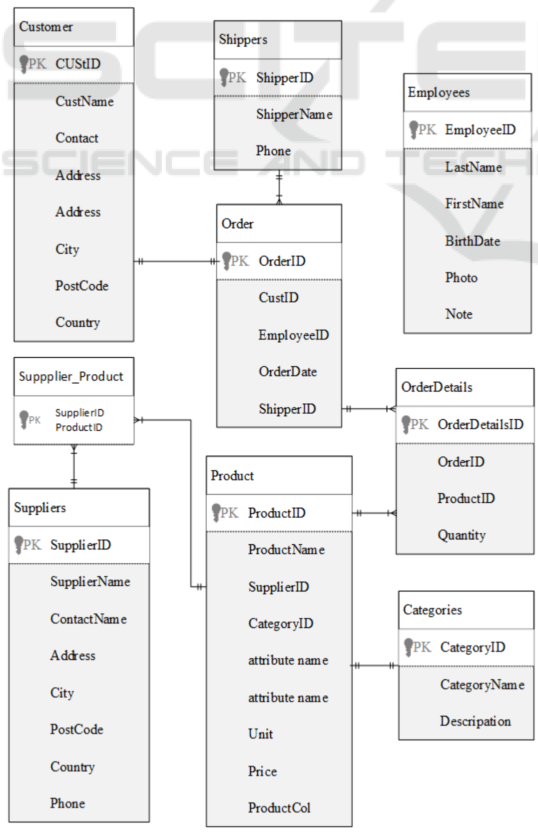

Figure 3: Schema after changing the relationship between product and supplier.

In the W3schools schema in Figure 1, the relationship between products and supplies is one to one. If the W3schools schema needs to change the business requirements by allowing one product to have many supplies or each supplier to provide many products, then this new requirement brings difficulty in changing the 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 the current data. 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 database schema needs to change, all queries related to product and resource need to be redesigned and recorded.

By contrast, an SDOD supports a flexible schema with semi-structured data that can add or change relationships between entities without effect. In the previous case, 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 SDOD, mapping one-to-one or one-to-many relationships can be through the embedded documents. Therefore, this schema will store the many supplies as embedded documents into each product, as shown in Figure 2.

As we can see that the SDOD is flexible schema and can support the new business requirements. The evaluation of this paper is used to evaluate the development agility of business requirements and the result shows that the SDOD provide a flexible schema. On the other hand, JSON is more compact than XML for semi-structured data because XML has many rules and complexity in representing a semi-structured data format. 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.

5 PERFORMANCE TEST OF RELATIONAL DATABASE AND DOCUMENT-ORIENTED DATABASE BASED ON SDOD

This evaluation clarifies whether the proposed method is statistically significant. The statistical significance between the relational database and document-oriented database is measure using T-test. The T-test is used to compare the means from two different groups of data and to determine any significant difference between the means of two groups that may be related in certain features. This test measures the query execution time for the relational database (Oracle) and document-oriented database (MongoDB). This evaluation evaluates the significance of Create, Retrieve, Update, Delete and Join operations based on data migrations.

In Table 2, the first show the database operation that used for this evaluation. The numbers that show in the third and fourth columns are the times in millisecond to execute each query for a relational database (Oracle) and a document-oriented database (MongoDB).

Table 2: Performance test of a relational database (Oracle) and a document-oriented database (MongoDB).

Operation	Relational Database (Oracle) (Time in Millisecond)	Document-Oriented Database (MonogDB) (Time in Millisecond)
Select	2.226	0.003
	1.826	0.002
	2.376	0.001
	0.182	0.002
	0.171	0.099
Insert	16,900	2,444
	18,210	5,326
Update	58,587	340
	815	2
	1,925	112
	94,080	582
	20,085	2,390
Delete	1,514	67
	2,651	140
	8,266	251
	3,196	799
	2,752	7
Join	1,969	210
	647	3
	1,028	102
	1,780	155

As the above table shows, the same query was run in MongoDB and Oracle, Therefore, this T-Test is paired test (oracle,mongo, conf.level = 0.95, vr.equal=FALSE, alternative="greater", paired=TRUE).

The type of T-test is paired and the input data for this test is Oracle and Mongo database. The T-value is 2.101 in favor of mongo with 20 degrees of freedom; also, the p-value which is the probability of the result is 0.02426. In addition, the alternative hypothesis is true, the difference in means is greater than zero, and the percent confidence interval is 95.

The T-test shows that the mean difference in the execution time between Oracle and MongoDB is statistically significant (i.e., p-value is smaller than 0.05 for the confidence level of 0.95, p=0.024).

The T-test shows that the mean difference in the execution time between Oracle (mean(Oracle)=11162.47, sd (Oracle)= 23233.4) and MongoDB (Mean(MongoDB)=615.72, sd(MongoDB)= 1291.533) is statistically significant (t(20)=2.101, p=0.02). In other words, based on the performance test shown in Table 2, there is strong evidence that MongoDB queries execute significantly shorter (measured in milliseconds) compared to Oracle.

6 CONCLUSION

This study successfully proposes a flexible schema that can cover all data types and properties of a document-oriented database. Besides, this study overcomes the issues in handling the relationships in a complex database by covering all types of relationships to be compatible with a document-oriented database. SDOD provide new feature such as flexible schema by allows adding any field in any documents without any constrains and allows each document to have different numbers of fields. Thus, it can be an efficient schema for a semi-structured data with flexible schema that will keep up with the new business requirements.

In addition, the document and embedded document in semi-structured data are less homogeneous than those in structured data, therefore, it allows each key to have different contents in each document, and these contents can be blend in the same document without a structure. The future work will proposed a schema for other types of NoSQL models.

REFERENCE

Arora, Rupali, & Aggarwal, Rinkle Rani. (2013). Modeling and querying data in mongodb. *International Journal of Scientific and Engineering Research, 4*(7).

Atzeni, Paolo, Bugiotti, Francesca, Cabibbo, Luca, & Torlone, Riccardo. (2016). Data modeling in the NoSQL world. *Computer Standards & Interfaces*.

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

Bhogal, Jagdev, & Choksi, Imran. (2015). Handling big data using NoSQL. Paper presented at the *Advanced Information Networking and Applications Workshops (WAINA), 2015 IEEE 29th International Conference on*.

El Alami, Alae, & Bahaj, Mohamed. (2016). *Migration of a relational databases to NoSQL: The way forward.* Paper presented at the Multimedia Computing and Systems (ICMCS), 2016 5th International Conference on.

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

Guimaraes, Valeria, Hondo, Fernanda, Almeida, Rodrigo, Vera, Harley, Holanda, Maristela, Araujo, Aleteia, . . . Lifschitz, Sergio. (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.

Hanine, Mohamed, Bendarag, Abdesadik, & Boutkhoum, Omar. (2016). Data Migration Methodology from Relational to NoSQL Databases. *World Academy of Science, Engineering and Technology, International Journal of Computer, Electrical, Automation, Control and Information Engineering, 9*(12), 2369-2373.

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

Kanade, Anuradha, Gopal, Arpita, & Kanade, Shantanu. (2014). A study of normalization and embedding in MongoDB. Paper presented at the *Advance Computing Conference (IACC), 2014 IEEE International.*

Maruping, Likoebe M, Venkatesh, Viswanath, & Agarwal, Ritu. (2009). A control theory perspective on agile methodology use and changing user requirements. *Information Systems Research, 20*(3), 377-399.

Mior, Michael J. (2014). Automated schema design for NoSQL databases. Paper presented at the *Proceedings of the 2014 SIGMOD PhD symposium*.

Mohan, C. (2013). History repeats itself: sensible and NonsenSQL aspects of the NoSQL hoopla. Paper presented at the *Proceedings of the 16th International Conference on Extending Database Technology*.

Moore, Philip, Qassem, Tarik, & Xhafa, Fatos. (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*.

Ogunyadeka, Adewole, Younas, Muhammad, Zhu, Hong, & Aldea, Arantza. (2016). A Multi-key Transactions Model for NoSQL Cloud Database Systems. Paper presented at the *2016 IEEE Second International Conference on Big Data Computing Service and Applications (BigDataService)*.

Rathor, Shekhar, Batra, Dinesh, & Xia, Weidong. (2016). What Constitutes Software Development Agility?

Rocha, Leonardo, Vale, Fernando, Cirilo, Elder, Barbosa, Dárlinton, & Mourão, Fernando. (2015). A Framework for Migrating Relational Datasets to NoSQL1. *Procedia Computer Science, 51*, 2593-2602.

Soransso, RASN, & Cavalcanti, Maria Cláudia. (2018). Data modeling for analytical queries on document-oriented DBMS. Paper presented at the *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*.

Stanescu, Liana, Brezovan, Marius, & Burdescu, Dumitru Dan. (2017). An Algorithm For Mapping The Relational Databases To MongoDB--A Case Study. *International Journal of Computer Science & Applications, 14*(1).

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

Tauro, Clarence JM, Aravindh, S, & Shreeharsha, AB. (2012). Comparative study of the new generation, agile, scalable, high performance NOSQL databases. *International Journal of Computer Applications, 48*(20), 1-4.

Yoon, Jongseong, Jeong, Doowon, Kang, Chul-hoon, & Lee, Sangjin. (2016). Forensic investigation framework for the document store NoSQL DBMS: MongoDB as a case study. *Digital Investigation, 17*, 53-65.