Document-Oriented Data Schema for Relational Database Migration to NoSQL

Shady Hamouda
Emirates College of Technology
Abu Dhabi, UAE &
Universiti Sains Malaysia
Penang, Malaysia
Shady.hamouda@ect.ac.ae

Zurinahni Zainol
Universiti Sains Malaysia
Penang, Malaysia
zuri@usm.my

Abstract— **"Big data" has become a crucial issue and one of the most important technologies in the modern world. The management of a semi-structured data format for big data is an important concern that must be addressed. Most relational database management systems fail to handle the scalability and flexibility of big data. NoSQL, which is a new concept in database technology, offers support a large volume of data, thereby providing a good data storage and retrieval mechanism. However, NoSQL has no standard data modeling method. The important properties in handling the database relationships need to be considered in NoSQL data modeling. This study proposes a document-oriented data model for big data and then applies this model to migrate relational database applications to NoSQL on the basis of the properties of the ER model.**

Keywords—data model; NoSQL; migration; document-oriented data

I. Introduction

According to [1], 35 zettabytes of data are expected by 2020, with healthcare data accounting for 25,000 petabytes of that total. "Big data" has become an important research topic in developing technologies that can manage and process a large volume of data [2]. Current relational database management systems (RDBMS) are inefficient at handling big data applications and software requirements, such as fast and low-cost analysis, processing, and management [3, 4]. Moreover, RDBMS does not support horizontal scaling for distributed environments and does not achieve effective data portability [5]. Nowadays, many organizations face problems regarding big data handling and storage as most applications still use "RDBMS" [6, 7]. A recent research [8] discussed the problems that occur in a relational database, which is a challenge in big data handling due to its need to process and integrate variety and velocity data.

NoSQL, which is a new concept in database technology, offers the possibility to support large volumes of data. This technology has various ways of storage, management, and implementation and has four database categories: key–value, column, document, and graph [6, 9]. Each category employs different strategies in data type handling and data processing and retrieval and has various query processes.

The demand to convert relational databases to NoSQL exists. Some organizations have considered migrating from the traditional RDBMS [10, 11] as the data volume for storage is large and the current applications cannot satisfy scalability and availability [12].

According to [13, 14], NoSQL has the full scalability required to support horizontal scaling. Therefore, transforming the current SQL databases into NoSQL is a necessary and emerging issue. However, migrating data from a relational database to a NoSQL database is difficult. According to [3], the migration process involves the challenges of obtaining the appropriate schema for the NoSQL database and integrating the migration process of the relational schema to the NoSQL schema. Designing a standard schema for a NoSQL database is difficult because tools or methods that support a good schema or model are unavailable [15]. According to [4], relational databases and NoSQL are different models. No standard method or algorithm recognizes table type, document structure definition, migration field, and document formation [11].

The current study aims to design a schema for document-oriented data to migrate from the current traditional database schema to NoSQL.

II. Literature Review

A. Data Models

A semi-structured data format can store data in XML, JSON, and BSON. Another key point document database can store semi-structured data as a key value without constraints; therefore, the value of the key can be any data type that gives flexibility to the database to store any type of data [16].

A semi-structured data type organizes data without structured data formatting, which associates data models with relational or other data forms; therefore, many models, such as ERX [34], ORA-SS [35], GN-DTD [33], XER [36], EReX [37], and GOOSSDM [17], have been proposed to handle semi-structured data. Major XML data models currently use directed edge-labeled graphs to represent XML documents and their schemas [33]. According to [17], none of these models can satisfy all of the properties and requirements of semi-structured data models, and only GOOSSDM can address most of the properties of semi-structured requirements. However, integrating semantic web technologies is still problematic with this model. In addition [18], GOOSDM has drawbacks in representing schema-less data and data with particular timestamps. Meanwhile, ORA-SS and GN-DTD employ different concepts to represent the data model because they are incompatible with NoSQL.

Therefore, this work proposes a document-oriented data schema (DODS) as a conceptual model for document-oriented data and transforms this model into a logical model to support logical design.

B. Migration from relational database to NoSQL database

Database normalization identifies the relationships between tables in relational databases through foreign keys [15]. Then, data are organized and accessed through structured query language (SQL). NoSQL has different concepts for its four data models, and each mode has a different concept for data representation and storage. Currently, many studies propose models to migrate relational databases to NoSQL. The following are the different ways and models proposed for the migration.

1) Convert through CSV file

Chickerur et al. [3] proposed a method to transfer existing relational database applications to MongoDB by converting the tables into CSV files and importing the converted files from the MongoDB command. This work compared MySQL and MongoDB through the execution time of “create, retrieve, update, delete” operation queries using a case study of an airline database. Performance timing tests across various operations showed that MongoDB performs well in terms of query execution time for big data. However, this method only transfers the tables without presenting the relationships among them. Moreover, MongoDB does not normalize data after their migration to the document database.

2) Use query-oriented data model

[22] proposed a query-oriented data modeling method to transform a relational schema to a NoSQL database schema based on data structure and data query. This model uses a three-phase framework: first, it describes the stored data structure for the problem domain and data query requirements; next, it uses query-oriented data modeling to analyze the data model for the NoSQL database; and finally, it uses query-oriented data modeling to generate the data schema for the NoSQL database. This study did not describe the normalization process and the data model for the document database on the basis of the relationships among the entities. Meanwhile, [16] proposed a model that investigates the migration of SQL databases to NoSQL with relational algebra. This study described the theories and methods of NoSQL solutions, compared the latter with RDBMS, and used relational algebra to design a MongoDB model. The researchers also provided mapping between MongoDB and RDBMS. This study focused on mathematical modeling and thus needed to study and analyze RDBMS and MongoDB through mathematical modeling using relational algebra.

In another case, [23] designed a framework called NoSQLayer, which has two modules: one is the data migration module, which identifies and analyzes the relational database and converts it to a NoSQL data model structure; the other is the data mapping module, which is designed to be an interface between the application and the DBMS. NoSQLayer designs a seamless database migration process that allows the developers to continue creating queries in the relational model, and the data will be fetched by a NoSQL database. This framework aims to keep the data in the NoSQL database and to create queries in the relational model. However, the migration process depends on the conceptual structure of the underlying source model and may be unsuitable for some other applications [24].

3) Metamodeling

Feng et al. [25] developed a model to transform UML class diagrams into Cassandra databases. This model depends on model-driven architecture and prepares a metamodel of the UML diagram and the Cassandra data model. This model is used to transform all the required data and relationships. However, the different data models between relational databases and NoSQL databases present two migration problems: obtaining a NoSQL schema and integrating the migration process into software development.

Meanwhile, Kartinis et al. [26] provided a technology to help companies migrate their relational database systems to document-oriented databases. This technology depends on the metamodel approach and implements two steps to migrate data: first, the physical data structure from the existing relational database is obtained, and a metamodel is created. Then, data from the relational database are migrated to a NoSQL database, and a metamodel data structure is created. On the physical level, the relational model stores data without presenting the relationships or full-view complexity of data in an N:N relationship type. However, the relational model cannot automatically determine the additional information stored in the relationship table. On the logical level, the model shows how data are presented in logical units or entities and how these data are physically stored in the database.

According to [27], the relationships in the document-oriented model can be expressed as embedded documents and links between documents; unfortunately, choosing one of them to design the relationships remains an issue as the selection depends on the application requirements and needs. Embedded documents can be used for a limited amount of data and in special cases. Meanwhile, the linking approach is more appropriate than embedding for large amounts of data because using the latter for sizable data will make the document and the collections considerably big and lead to poor processing and retrieval performance.

Hanine et al. [28] designed a model to migrate data from relational databases to NoSQL. This model has three phases: extracting data from the source database, transforming the data, and migrating the converted data to the target database. Unfortunately, this methodology only creates the MongoDB database automatically from a relational schema without testing its performance. Furthermore, this model does not cover migration and the handling of the relationships among the entities. This methodology still needs to be improved to cover all types of databases.

4) Normalization rule model

Gu et al. [30] used traditional theory normalization rules (2NF, 3NF, and BCNF) to design a MongoDB schema by converting the results of the normalization rules into a MongoDb schema as follows:

- The multi-valued attributes are converted into array data.
- Both data items are converted into a single-data item if any full functional dependency exists.
- The same relational database rules are applied for the partial and transitive dependency.

The results showed that this model can design a MongoDB schema from the normalization rules and that NoSQL can reduce the foreign keys and collection of related databases to improve efficiency. However, this study used a small schema for the normalization rules of the MongoDB schema. The method misses important features, such as weak entities, composite attributes, M:M relationships, primary keys, and foreign keys.

5) Uniform mid-model

Liang et al. [4] designed a uniform mid-model to improve model transition and data migration between relational databases and NoSQL. This model combines relational database properties, such as entities with properties and the relationship, data, and query features, to form an object. The model uses the relational database as input into the mid-model. Next, the mid-model extracts all the entities and relationships into an object and transfers these objects to a physical model of the NoSQL database. However, the mid-model method cannot be applied to all types of NoSQL databases.

6) NoSQLayer framework

The NoSQLayer framework has two modules: one is the data migration module, which can identify and analyze the relational database and convert it into a NoSQL data model structure; the other is the data mapping module, which is designed to be an interface between the application and the DBMS. This framework facilitates a seamless database migration that allows the developers to continue creating queries in the relational model. Meanwhile, the data are fetched by a NoSQL database. This framework automatically and transparently migrates data and models from RDBMS to NoSQL. The NoSQL framework keeps the semantics of the original database without changing the source code. This framework can complete data migration while maintaining the entire structure of the original database and storing it as a NoSQL model. This model aims to keep the data in the NoSQL database and creates queries in the relational model. However, the migration depends on the conceptual structure of the underlying source model and may be unsuitable for some other applications.

In summary, most of the above-mentioned methods aim to migrate data from relational databases to NoSQL databases without considering an integrated way to migrate all the properties of the relational database, especially the handling of the types of relationships of complex databases; this issue is important in the migration of complex databases. This study provides a compatible method for migrating relational databases to NoSQL databases using relational database properties, such as tables, relationships, functions, constraints, keys, normalization processes, and joint operations.

III. PROPOSED WORK

This study aims to convert the ER model into a document-oriented data model by addressing the migration issues of complex databases with many tables and records. The migration of the ER model to a document-oriented model consists of two main stages: (1) designing the DODS, and (2) migrating the ER model to the DODS, as shown in Figure. 1.

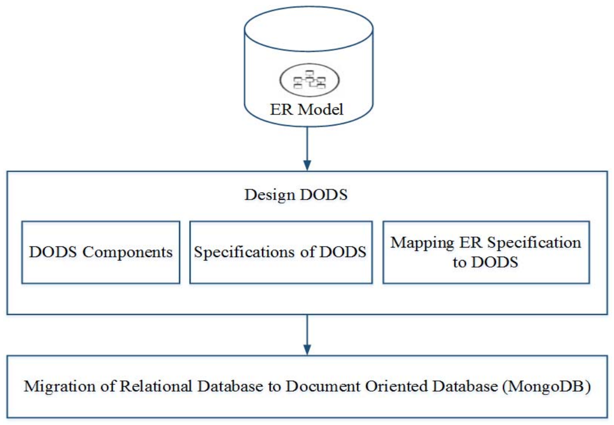

ER Model
Design DODS
DODS Components
Specifications of DODS
Mapping ER Specification to DODS
Migration of Relational Database to Document Oriented Database (MongoDB)

Fig. 1. Stages of migration

A. *Designing the DODS*

This stage aims to design a schema for the document-oriented model and support a good schema through a new method of recognizing data document structure, definition, migration field, and document format. This stage consists of the following sub-stages: (1) DODS components, (2) DODS features, and (3) mapping of ER model and DODS specifications.

This study uses the DODS as a schema for the document-oriented model, which can be used as a standard model for the document-oriented database. DODS focuses on transforming the conceptual model to a logical model to support the logical design of big data on the basis of the following features of the ER model:

- Entity and attribute identification
- Entity relationship identification
- Relationship resolution
- Keys (primary, foreign)
- Constraints
- Normalization processes

B. *DODS Components*

Many DODS components are considered in designing a database based on the ER data model. These components can contain the logical and physical data models. Therefore, this stage describes the following main components in designing the DODS:

- A collection that represents the database table
- A document that represents the table record
- Key–value pairs as keys that represent the attribute and value of the data type according to the type of value,
- The array data type, which can be used to represent multiple values or many documents
- An embedded document that represents the relationships among documents
- A reference approach (link the collection) that represents the type of relationships between the collections
- Different types of relationships (1:1, 1:M, M:M, and unary relationships)
- Different types of attributes
- An entity

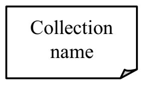
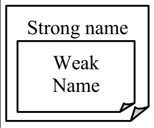
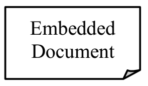
C. *Features of DODS*

The features of the ER model were used as a standard to describe a database schema model and the notions of the DODS shown in TABLE. 1.

D. *Mapping between ER and DODS specifications*

After identifying the DODS components, the ER model is converted to the DODS on the basis of the following conversion table.

TABLE. 1. Specifications of DODS

Specification	Document model	Descriptions and conditions
Strong entity		The folded corner describes the strong entity and the entity name written on it.
Weak entity		The weak entity is described using a double-folded corner with the name of this entity.
Ordinary attribute	**{K1,….,Kn}**	The attributes of each entity are described using K1, which represents the name of the attribute; n represents the number of attributes. These attributes are listed inside two braces with a comma in between.
Composite attribute	**CK: [{ K1, Ki,.. }]**	Composite attributes can be described as an array with the following embedded documents: CK: [{ K1, Ki … }] CK:→ composite attribute [] → array data type {}→ embedded document Ki → composite attribute names listed on the document with a comma between them i → number of attributes
Multi-valued attribute	**KM: [{vi }, …..{ vm}]**	The multi-valued attribute is described by the name of this attribute k with an array data type. Vi represents all of the values, and m represents the number of values.
Derived attribute	**K#**	The derived attribute is described by a hash tag after the attribute K.
Attribute relationship	**(K1,….Kj)**	The attributes of relationships are described by the names of the attributes between two brackets. j represents the number of related attributes.
Primary key	$\underline{\mathbf{K}}$	The primary key is represented by an underline under the attribute.
Foreign key	$\dashuline{\mathbf{K}}$	The foreign key is represented by a dashed line under the attribute.
One-to-one relationship		The 1:1 relationship is described by a folded upper-right corner of the embedded document written with the

	Embedded document	type of the relationship. This embedded document is stored into the related document.
	1 $\overline{\qquad\qquad}$ 1 Reference approach	If more relationships exist or both sides have big data, then this relationship should be represented using the reference model.
One-to-many relationship	$\boxed{\text{Embedded Document}}$	The 1:N relationship is described through the embedded document or reference approach depending on the size of the entity of the N-side relationships. N is small if it is less than a couple of hundreds of documents based on an assumption that documents are not large. Thus, 1:N is described by a folded upper-right corner of the embedded document written with the type of relationship. This embedded document is stored into the related document.
	1 $\xrightarrow{\qquad}$ N Reference notion for 1: N relationship	N is large if it is tens or hundreds of thousands or millions of documents. The 1:N relationship is described as a reference between two entities with an arrowhead that indicates the direction of the N relationship.
Many-to-many relationship	1 $\xleftrightarrow{\qquad}$ N Reference notion for M: M relationship	The M:M relationship describes the references between two entities with an arrowhead between these entities. This type of relationship is identified by creating two collections C1 and C2, and then storing the primary key of the related collection C1 inside the collection side of C2, and vice versa. If the document of C1 is stored inside C2, then the name of the document root becomes C1, and the collection name is [{Ki, Ki,..},{Ki, Ki,..}, ….]
Unary relationship	K@	The unary relationship is described by adding @ after the attribute K and the name of the relationship.

E. Migration from a relational schema to a document-oriented schema

The second stage is migrating the relational database to the document-oriented database by normalizing and denormalizing data with two models, namely, embedded document and reference model, which also support a joint operation between the database collections. The migration process from the relational database to the document-oriented database is shown in Figure 2.

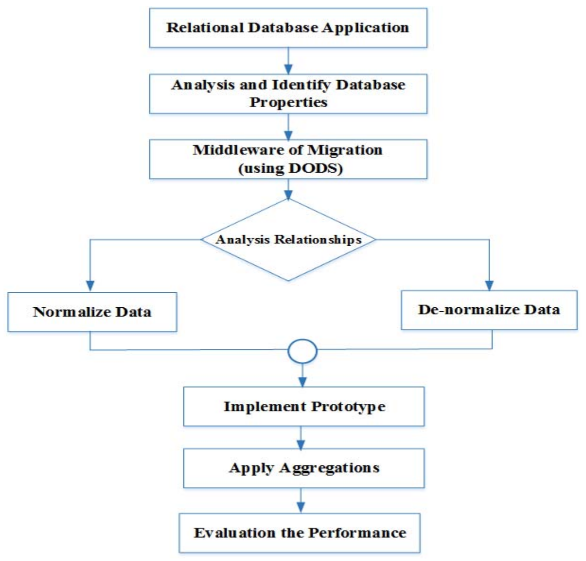

Relational Database Application
Analysis and Identify Database Properties
Middleware of Migration (using DODS)
Analysis Relationships
Normalize Data
De-normalize Data
Implement Prototype
Apply Aggregations
Evaluation the Performance

Fig. 2. Migration process from a relational database to a document-oriented database (MongoDB)

The process in Figure 2 is as follows:

1. The database to be migrated is selected.
2. The database properties are identified and analyzed.
   - The names of the entities are identified.
   - The attributes of each entity are identified with the primary and foreign keys.
   - The relationships between the entities are identified.
   - These relationships are analyzed.

- The constraints and keys are identified.

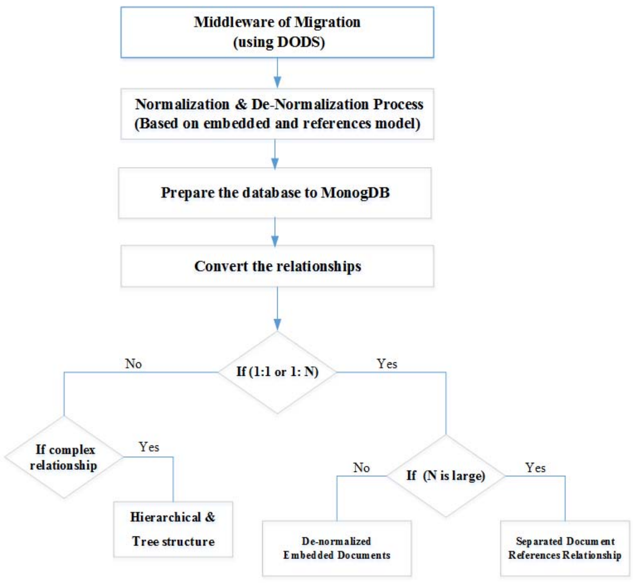

Middleware of Migration (using DODS)
Normalization & De-Normalization Process (Based on embedded and references model)
Prepare the database to MonogDB
Convert the relationships
If (1:1 or 1: N)
No
Yes
If complex relationship
Yes
Hierarchical & Tree structure
If (N is large)
No
Yes
De-normalized Embedded Documents
Separated Document References Relationship

Fig. 3. Migration middleware process

3. The relational database is migrated to the document-oriented database (MongoDB) by use of the model illustrated in Figure 3, which is implemented through a prototype.

Normalization and de-normalization techniques are applied to handle the relationships of the relational database depending on the relationship type:

- In 1:1 relationships, document-embedding is applied depending on which data can be associated with the other data, and the small document is embedded into large documents.
- In 1:N relationships, N is denormalized if it is small, and these entities are combined. These entities have a relationship with the same document and are designed in separate documents if N is large and involves numerous changes.
- Complex relationships (M:M) are handled using hierarchical and tree structures.
- Unary relationships can add the same primary key of the relationship even with a different name.

4. The prototype is implemented to handle and retrieve the data through the MongoDB database, and the aggregation technique is applied to support joint operation, analysis, and reporting of the data.
5. Data integrity, database structure, table relations, data types, and performance of the new database (MongoDB) are evaluated and compared with those of the relational database.

F. *Case Study*

The company database schema [32] in Figure 4 shows the ER schema model used as a case study for mapping the ER schema to the document-oriented schema by use of DODS.

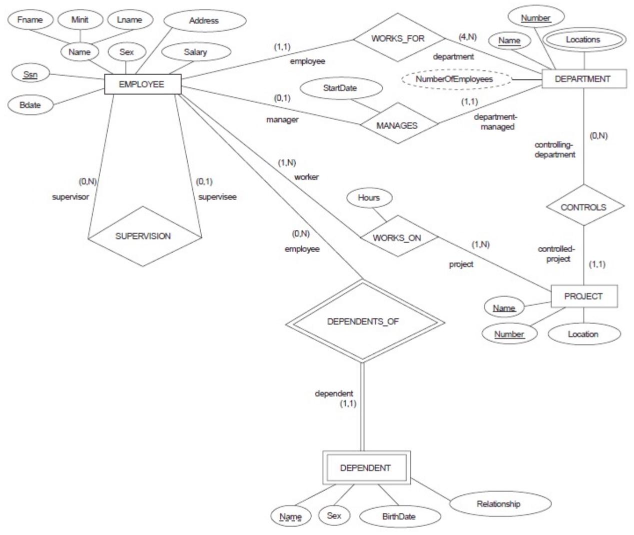


Fig. 4. ER schema for a company database [32]

By applying the DODS model, Figure 5 shows the diagram of the document-oriented data model after mapping the schema of Figure 4.

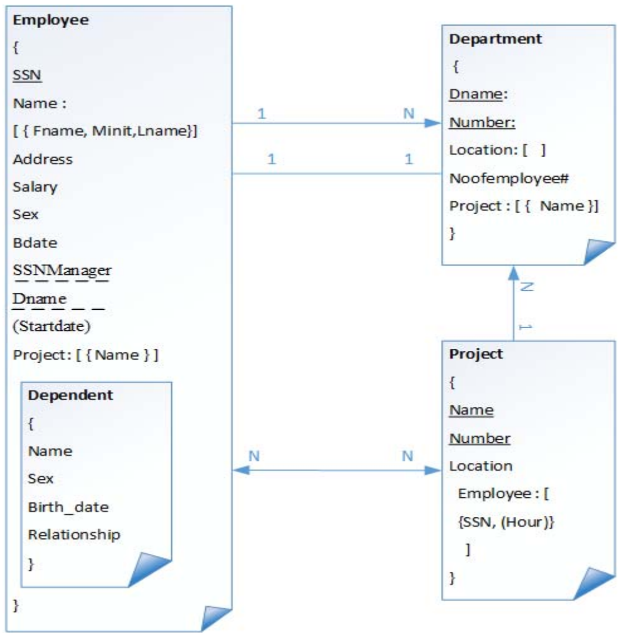


Fig. 5. Diagram of the document-oriented data model

The newly designed diagram of the document-oriented model needs to be converted into JSON. The MongoDB database is designed as follows.

Although the principal concept of the NoSQL data model involves storing data as key–value pairs [31], the key is considered the field of the table in a relational database, and the value associated with the key can be of any type of data structure. Unlike the case in the relational database, all the field data in the NoSQL model should have the same structure and may have null values. Moreover, this key–value pair can be collected into a document that can represent the field with record in the relational database.

Consequently, a set of related documents is stored and represented by a collection that considers the table of the relational database. Each document can be identified by a unique key that is known to the user or automatically created by the database. This ID becomes an index for the document or can create other indexes, depending on the application requirements, to accelerate the query process. Relationships between the collections can be identified by embedding the document or referencing it.
In the embedded document model, the document can contain another document. This model can denormalize the database. Meanwhile, referenced documents consider the relationships of the relational database, which shows the relationships between the collections and the foreign key.

A document-oriented NoSQL model is designed for storing, retrieving, and managing document-oriented or semi-structured data. The central concept of the document-oriented database is the notion of a document wherein the contents are encapsulated or encoded in a standard format, such as JSON. Data are easily stored in JSON or BSON by mapping the object structure of most of the programming languages directly into this representation. Therefore, a document-oriented database is used to achieve the objective of this study.

IV. CONCLUSION

This study proposes the DODS, which can cover all data types in databases. Its design requires embedding and normalizing semi-structured data. Moreover, DODS overcomes the issues of handling the relationships of a complex database that can cover all types of relationships to be compatible with the document-oriented model. Finally, this study provides a compatible migration method for all the tables, relationships, functions, and constraints to normalize and denormalize data. For future work, automatic prototyping will be implemented to map a relational database to a document database, especially DODS-based MongoDB using a healthcare database.

V. REFERENCES

[1] Katal, A., M. Wazid, and R. Goudar. *Big data: issues, challenges, tools and good practices*. in *Contemporary Computing (IC3), 2013 Sixth International Conference on*. 2013. IEEE.

[2] Chen, J., et al., *Big data challenge: a data management perspective.* Frontiers of Computer Science, 2013. **7**(2): p. 157-164.

[3] Chickerur, S., A. Goudar, and A. Kinnerkar. *Comparison of Relational Database with Document-Oriented Database (MongoDB) for Big Data Applications*. in *2015 8th International Conference on Advanced Software Engineering & Its Applications (ASEA)*. 2015. IEEE.

[4] Liang, D., Y. Lin, and G. Ding. *Mid-model Design Used in Model Transition and Data Migration between Relational Databases and NoSQL Databases*. in *2015 IEEE International Conference on Smart City/SocialCom/SustainCom (SmartCity)*. 2015. IEEE.

[5] Ogunyadeka, A., et al. *A Multi-key Transactions Model for NoSQL Cloud Database Systems*. in *2016 IEEE Second International Conference on Big Data Computing Service and Applications (BigDataService)*. 2016. IEEE.

[6] Goyal, A., et al. *Cross platform (RDBMS to NoSQL) database validation tool using bloom filter*. in *Recent Trends in Information Technology (ICRTIT), 2016 International Conference on*. 2016. IEEE.

[7] Gudivada, V.N., D. Rao, and V.V. Raghavan. *Nosql systems for big data management*. in *2014 IEEE World Congress on Services*. 2014. IEEE.

[8] Assunção, M.D., et al., *Big Data computing and clouds: Trends and future directions.* Journal of Parallel and Distributed Computing, 2015. **79**: p. 3-15.

[9] Tauro, C.J., S. Aravindh, and A. Shreeharsha, *Comparative study of the new generation, agile, scalable, high performance NOSQL databases.* International Journal of Computer Applications, 2012. **48**(20): p. 1-4.

[10] Han, X., et al. *A big data model supporting information recommendation in social networks*. in *Cloud and Green Computing (CGC), 2012 Second International Conference on*. 2012. IEEE.

[11] Kanade, A., A. Gopal, and S. Kanade. *A study of normalization and embedding in MongoDB*. in *Advance Computing Conference (IACC), 2014 IEEE International*. 2014. IEEE.

[12] Bansel, A. and A.E. Chis. *Cloud-Based NoSQL Data Migration*. in *2016 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)*. 2016. IEEE.

[13] Lee, C.-H. and Y.-L. Zheng. *Automatic SQL-to-NoSQL schema transformation over the MySQL and HBase databases*. in *Consumer Electronics-Taiwan (ICCE-TW), 2015 IEEE International Conference on*. 2015. IEEE.

[14] Győrödi, C., et al. *A comparative study: MongoDB vs. MySQL*. in *Engineering of Modern Electric Systems (EMES), 2015 13th International Conference on*. 2015. IEEE.

[15] Guimaraes, V., et al. *A study of genomic data provenance in NoSQL document-oriented database systems*. in *Bioinformatics and Biomedicine (BIBM), 2015 IEEE International Conference on*. 2015. IEEE.

[16] Zhao, G., et al. *Modeling MongoDB with relational model*. in *Emerging Intelligent Data and Web Technologies (EIDWT), 2013 Fourth International Conference on*. 2013. IEEE.

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

[18] Banerjee, S., et al. *Towards logical level design of Big Data*. in *2015 IEEE 13th International Conference on Industrial Informatics (INDIN)*. 2015. IEEE.

[19] Atzeni, P., et al., *The relational model is dead, SQL is dead, and I don't feel so good myself.* ACM SIGMOD Record, 2013. **42**(2): p. 64-68.

[20] Hashem, H. and D. Ranc. *Evaluating NoSQL document oriented data model*. in *Future Internet of Things and Cloud Workshops (FiCloudW), IEEE International Conference on*. 2016. IEEE.

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

[22] Li, X., Z. Ma, and H. Chen. *QODM: A query-oriented data modeling approach for NoSQL databases*. in *Advanced Research and Technology in Industry Applications (WARTIA), 2014 IEEE Workshop on*. 2014. IEEE.

[23] Rocha, L., et al., *A framework for migrating relational datasets to NoSQL.* Procedia Computer Science, 2015. **51**: p. 2593-2602.

[24] Aboutorabi, S.H., et al. *Performance evaluation of SQL and MongoDB databases for big e-commerce data*. in *Computer Science and Software Engineering (CSSE), 2015 International Symposium on*. 2015. IEEE.

[25] Feng, W., et al. *Transforming UML Class Diagram into Cassandra Data Model with Annotations*. in *2015 IEEE International Conference on Smart City/SocialCom/SustainCom (SmartCity)*. 2015. IEEE.

[26] Karnitis, G. and G. Arnicans. *Migration of Relational Database to Document-Oriented Database: Structure Denormalization and Data Transformation*. in *Computational Intelligence, Communication Systems and Networks (CICSyN), 2015 7th International Conference on*. 2015. IEEE.

*[27]*Mason, R.T., *NoSQL Databases and Data Modeling Techniques for a Document-oriented NoSQL Database.*

[28] Hanine, M., A. Bendarag, and O. Boutkhoum, *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, 2016. **9**(12): p. 2369-2373.

[29] Stanescu, L., M. Brezovan, and D.D. Burdescu. *Automatic mapping of MySQL databases to NoSQL MongoDB*. in *Computer Science and Information Systems (FedCSIS), 2016 Federated Conference on*. 2016. IEEE.

[30] Gu, Y., et al. *Application of NoSQL database MongoDB*. in *Consumer Electronics-Taiwan (ICCE-TW), 2015 IEEE International Conference on*. 2015. IEEE.

[31] Ahuja, S.P. and B. Moore, *State of big data analysis in the cloud.* Network and Communication Technologies, 2013. **2**(1): p. 62.

[32] Elmasri, R., *Fundamentals of database systems*. 2008: Pearson Education India.

[33] Zainol, Zurinahni, and Bing Wang. "*GN-DTD: Graphical notations for describing XML documents."* Advances in Databases Knowledge and Data Applications (DBKDA), 2010 Second International Conference on. IEEE, 2010.