Toward a Smart Approach of Migration from Relational System DataBase to NoSQL System: Transformation Rules of Structure

Abdelhak Erraji, Abderrahim Maizate, and Mohammed Ouzzif

Abstract In the last few years, databases have become very important and very large because they play a strategic and important role in most organizations and because they receive a huge flow of information from multiple sources every moment in building BigData. This situation identified several limitations and weaknesses in relational database management systems (RDBMS), such as availability, real-time response, horizontal scalability, decision support, advanced data analysis, and especially the management of Bigdata which can reach zeta bytes in storage. This requires the storage and organization of this data in a new management system database not fixed by a rigid structure and resolves all problems associated with the storage of database in a relational system. In this view, organizations need a new NoSQL (not only SQL) system that overcomes the limitations of the relational system. The change of the database management system requires the migration of the databases from the relational system to another NoSQL, taking into consideration all stored data and keeping the majority of the possibilities and functionalities of the old system, with all the advantages of the new system. In this paper, we will identify the elements of relational databases that belong to nature: data, structure, and semantics, which we must migrate to a NoSQL system, such as a document-oriented system. Also in this paper, we will present the rules for transforming the structure of the relational system to another document-oriented NoSQL, according to the principal's basics of a new approach to migration.

Keywords Migration DataBase · NoSQL · Database · Transformation rules · BigData

A. Erraji (✉)
RITM ESTC Laboratory, Hassan II University, ENSEM, Casablanca, Morocco

A. Maizate · M. Ouzzif
RITM ESTC Laboratory, Hassan II University, ESTC, Casablanca, Morocco

M. Ben Ahmed et al. (eds.), *Innovations in Smart Cities Applications Volume 5*,
Lecture Notes in Networks and Systems 393, https://doi.org/10.1007/978-3-030-94191-8_63

1 Introduction

To be able to control the data as well as the users, we need database management systems. A database management system is a set of computer software that is used to manipulate databases and performs ordinary operations such as viewing, modifying, building, transforming, copying, backing up, or restoring databases of data. The relational model is based on a solid mathematical model based on the logic of the first-order predicates. It is based on simple concepts that make it strong at the same time as it is his weakness. Distributed storage is a real constraint on systems' today's relationships, to which is added the complexity of the structures of the data manipulated by the systems. At the time of the Web 2.0 revolution and given the current use of the Internet, the amount of data available on the web is increasing year by year exponentially. Today, companies are facing a huge increase in the quantity of data they need to store and process. This is especially true for very fashionable web applications like Twitter, Facebook, and Google. These must cope with the huge amount of activity generated by their millions of users. So the first basic need that NoSQL answers are performance. Relational database management systems (RDBMS) are the predominant solution for the storage and processing of data, but the experience of these companies has shown that the RDBMS model is reaching its limits in the face of such large amounts of data [1]. The NoSQL databases are mainly designed to optimally store given datasets, but as a counterpart, the language used to query the NoSQL system is much less rich. This is why relational databases will always survive, and NoSQL can never fully replace SQL.

Moving from a relational database to a NoSQL database has become one of the most desired benefits for a developer or an organization today. Working with a more flexible data model and having rigid schemas is a big advantage. The difference between relational database management systems (RDBMS) and NoSQL databases is the data model used. Each record in a relational database conforms to a schema with a fixed number of fields. The data is denormalized into several tables, and so the key benefit is that there are fewer duplicate data in the database [2]. In NoSQL databases, each object can have a completely different structure from other objects. The main advantages of databases are the flexible data model since data can be inserted without a defined schema; easy scalability because the data between the servers can be propagated automatically; and finally, the advanced technologies and modern NoSQL databases that cache data, for example, and in a transparent manner.

2 From RDBMS to NoSQL Database

2.1 Literature

Currently, scientific research rarely discusses how to effectively validate the best way to migrate data from RDBMS to NoSQL databases to ensure the quality of the

data being migrated. Therefore, those needs become a major concern and an interesting research topic. According to [19], data has become an essential hub for information processing. The authors present a framework, named NoSQLayer, based on two parts: a migration module and a mapping module. The first one is a set of methods enabling seamless migration between RDBMS, and the second one provides a persistence layer to process database requests and translate and execute requests in any RDBMS. In 2016, the authors of the article [20] present an automatic transformation of a multidimensional conceptual model into two NoSQL databases: column-oriented and document-oriented models, and to validate the transformation, they implemented four data warehouses using Cassandra (as a column-oriented NoSQL system) and MongoDB (as a document-oriented NoSQL system). The results show that MongoDB with hierarchical transformation is more suitable when dealing with OLAP queries. In the same year (2016), the authors of Article [21] proposed a flexible and highly modularized data adapter for hybrid database systems by using a general SQL layer that accepts queries from application services. It also controls query flow during database transformation. Therefore, they can provide a seamless mechanism to use RDBMS and NoSQL databases at the same time. NoSQL databases are designed to solve data processing problems in volume [22]. Cloud computing also enables the database as a service model to manage large volumes of user-generated data using NoSQL data repositories. Authors in [23] have presented a NoSQL data migration framework to foster data portability across cloud-based heterogeneous NoSQL data repositories; results show that the document-based database supports efficient migration of data, whereas the graph database ensures effective management of large volumes of data [24]. In 2017, the authors of the article [25] proposed a meta-model-based data merging approach, which allows simultaneous querying of data from heterogeneous database systems. Results of database queries are translated into JSON objects, and data merging is done by concatenating separate JSON files, so the result can be represented by any data visualization source (tables, Excel, CSV files, etc.). In addition, Big Data has also become a crucial issue and one of the most important technologies in the modern world [26]. The authors have presented a study by proposing a document-oriented data model for big data and then applying this model to migrate relational database applications to NoSQL.

All of these approaches transform relational databases into NoSQL systems from the point of view of data or structures only or both. Now, a relational database has the semantic aspect, which defines how we construct the entities, the tables, and their structures, and how we can generate new information from this structure and the stored data.

2.2 Discussion

We notice that articles [4–11] have transformed just a part of the relational databases and ignored another part in their migrations. Indeed, all relational databases

have three types of data: the first is stored in tables; the second is contained in the structures of tables; and the third is implicit and deduced from the relationships between the tables, which form the semantics data in BDDRs.

To explain the third type, we will consider the following MCD model:

This MCD model will be transformed to the following MLD model (Fig. 2):

In turn, it keeps the same form as tables in a relational system database. In the example above, we notice that the MCD has three elements (student, school, and registration operation), but its transformation into MLD keeps only the first two elements, and the third has been transformed into a foreign key in one of the two tables.

For relational systems, the registration operation models the effect and action between two objects and their multiplicity. This operation is implicit in MLD; it comes from the registration relationship in MCD, which can be restored in the BDDR by joining two tables using a foreign key.

However, the NoSQL systems do not allow joining the tables, so the foreign keys do not have any meaning in these systems, and also, one cannot restore the relations between the tables, which shows losing information during the transformation and others being superfluous like foreign keys.

To this end, we consider that the relations must have a particular transformation and that the foreign keys must disappear in the transformation, or else find a way to realize the joins in the NoSQL systems.

The joining of two tables in relational systems is based on three main steps: the Cartesian product of the two tables, the restriction of the rows based on the join condition to form a new data source for the SQL query and the referential integrity constraints. This third principle prohibits the deletion of a value used as a primary key in another table, and the addition of a foreign key that is not a primary key in the other table to verify the data integrity. These principles are managed by the relational system, which respects the ACID principle, which promotes data consistency. As for the NoSQL system, the documents are very large, and therefore their product generates a file that may have zeta bytes of data and be full of bad data

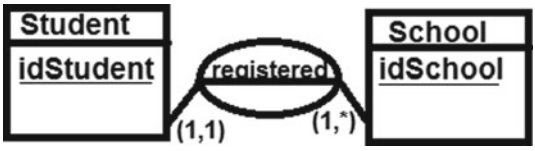

Fig. 1 Example of MCD model

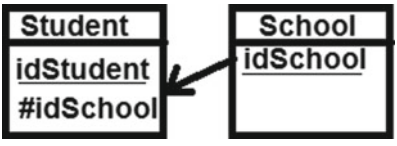

Fig. 2 The transformation of the MCD model in Fig. 1 to the MLD model

that requires restriction filtering based on a condition, which weighs down the system to simple data to extract, such as student enrollment in a school.

For NoSQL systems, data consistency can be a non-mandatory plus. They implement the BASE principle coming from the CAP theorem instead of ACID. The BASE principle favors the availability of data and the speed of responses. Therefore, the joins in relational systems must find another form during their transformation.

3 Analyzing and Rules of Transformation

3.1 Why MongoDB is Used as the Destination in Our Approach as a NoSQL System?

To complete our migration approach study, it is necessary to specify the destination category into which the old databases will be transformed. The authors of this paper have shown that document-oriented NoSQL databases are the most suitable for transforming relational databases [12] because they are similar in structure to relational systems. As a destination system, MongoDB is the favorite system because it implements the principles of the Document-oriented NoSQL category and it provides high performance in its category of NoSQL [13].

MongoDB stores data in JSON or Binary JSON (BSON) files as collections, which are structured in their files as a list of objects. Each object stored can be named "Document" and respects JSON's form. The object in JSON has a reference and an object, which contains several attributes with their values. This structure is very similar to the projection of a table onto a row in the relational system. So one row in a table can be transformed into an object in JSON, and one table storing a list of rows can be transformed by a collection into a list of objects. So we can transform all the tables into JSON files.

On the other hand, the stored functions can be transformed into programs in JavaScript, which might manipulate JSON files. Other data, which controls the structure in a relational system, can be transformed and stored in Mongoose, which controls the structure of data in MongoDB.

MongoDB can use a framework named "Mongoose" to define the structure of documents that will be stored in JSON files in their schemas. In these schemas, Mongoose can define all components of a relational database except for four elements: relationships, foreign keys, joining collections, and triggers. In the following section, we will define the rules for the transformation of relationships and foreign keys.

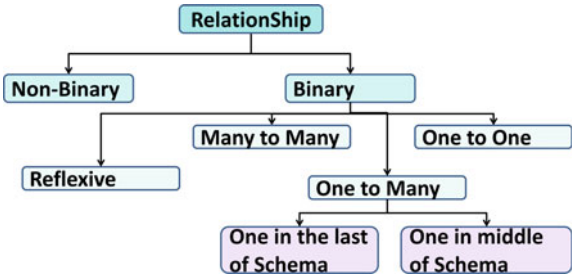

Fig. 3 Different types of Relationship in relational system Database

3.2 Rules of Transformation of Relationships and Foreign Key

This is the core of the relational system and the most demanding and interesting part of the migration. In our study, we will distinguish the relationships according to six types, as presented in Fig. 3:

In the relational model, relationships between entities are transformed into tables or foreign keys, to allow joins between tables to render the full picture of data in the relational system and to merge parts of the data dispatched in tables into a single complete data source with respecting the goal of the query. This data source is used to extract the complete data or derive other new information. But in the concept of objects, we reason by attribute. The notion of attribute makes it possible to store an entire collection of objects (equivalent to a table in the relational model) in the form of an attribute, which can be used by loops and the attribute access mode, to have a sequence of data to restore the same complete information as the relational model. This difference between the two systems clearly shows that in the relational model the connection between the data containers is done horizontally, on the other hand in the object model is done vertically.

Since the joins in the relational model are more expensive in terms of resources (RAM, CPU, etc.) and especially for large tables, their transformations in the same way in the object model will be more expensive than the relational system considering the vertical linking of collections. This situation requires the proposal of a model with fewer joins.

For this, we will use the nesting of entities (collections) in the form of an attribute when transforming relationships to create a structure giving the same calculation possibilities as the relational model but with fewer joins.

To go in this way, we will try to remove the relations and substitute them with nesting in the form of attributes, but this nesting will give a single collection extended vertically and very complex, especially in the access to the elements at the bottom of this nesting.

To have the optimum, we keep the same concept as the relational model in some cases and adopt the new approach of nesting by attributes in other cases, knowing well that this mixing must be done in such a way as to improve the system performance. To do this, we must begin by transforming the elements at the

extremities of the relational model to establish an intermediate situation. This transformation will be repeated until the final situation is reached, in which the different relations are transformed with the constraint of not exceeding two nesting levels per attribute in large objects so as not to reduce performance during the extraction operations of the data.

The rules to follow during these transformations are revealed in the following sections.

Our solution consists of proposing the following transformation rules to transform each type of relationship.

Rule 1 to Define a New Form of Foreign Key: We see that in the relational model, the foreign key is a duplicate in the current table of the field representing the primary key in another table. But in MongoDB, the identifier is attributed automatically and is different from the primary key defined in the relational model.

However, the primary key in the relational model will be transformed in MongoDB into an ordinary field whose verification of uniqueness and existence will be delegated to the Mongoose through the definition of the schema of the processed document.

According to these two situations, the foreign key will be transformed by an attribute whose value is an object made up of two attributes: the identifier of the object attributed in MongoDB, resulting from the transformation of the row of the table containing the primary key, which is called "Ref", and the attribute whose value results from the transformation of the value of the field playing the primary key in the relational model, which is called "value".

A foreign key will be represented by adding the new attribute in each document in our JSON files to represent the other entity that is part of the relationship. The name of this foreign key is the same as the collection representing the other entity, and its value is an object, whose structure must be defined in Mongoose. The objects added as the value of this key will be used to make aggregations in the file result of this transformation with fewer joins of collections.

Rule 2 for Non-binary Relations: In the relational model, this relationship is transformed into an independent table and allows self-joining to generate new information. If we attribute to it the new conception of transformation by attribute, then we lose this possibility of self-joining. As a result, we propose that this complex association be converted into an independent JSON file, with a new foreign key added to all documents in the collection of the new file JSON following the first rule. This transformation is explained by the following Fig. 4:

Rule 3 for Binary Relations of Type (Many to Many): In the relational model, this relationship is transformed into an independent table and allows self-joining to generate new information. If we attribute to it the new conception of transformation by attribute, then we lose this possibility of self-joining. Therefore, we propose that this complex association be converted into a new independent file JSON in the same way as rule 2. This transformation is explained by the following Fig. 5:

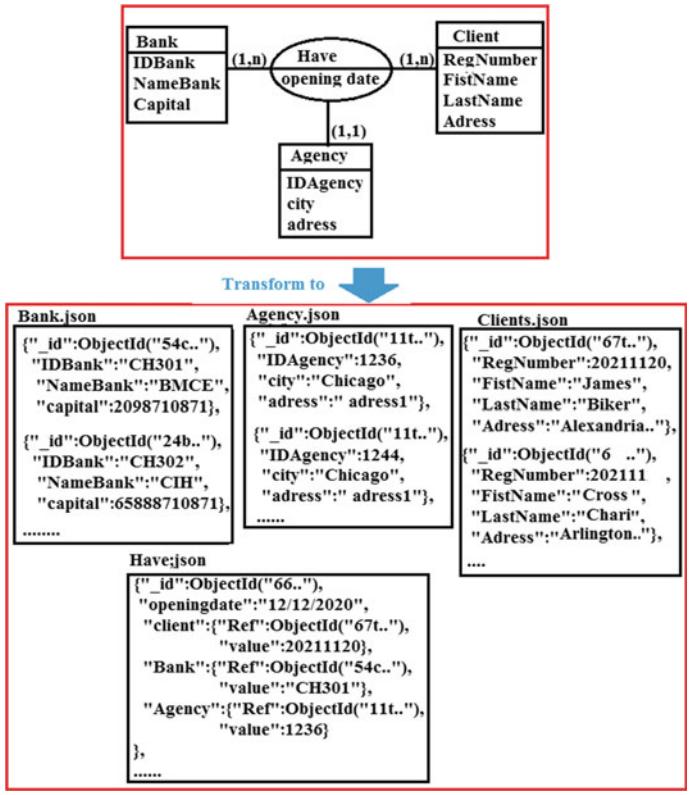

Fig. 4 Example of the practice of rule 1 and rule 2 to transform of Relationship non-binary

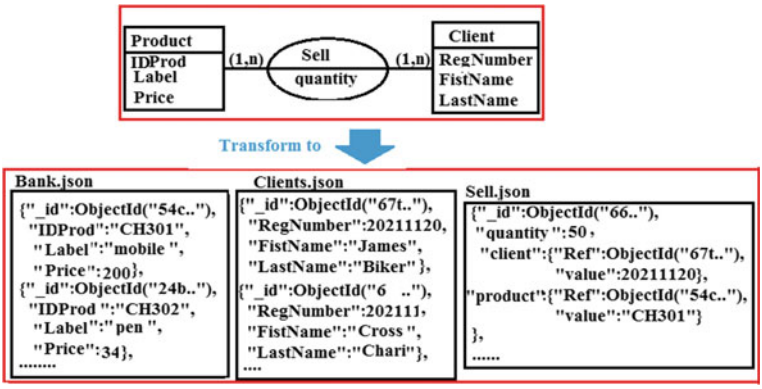

Fig. 5 Example of practice rule 3 to transform Relationship binary type: many to many

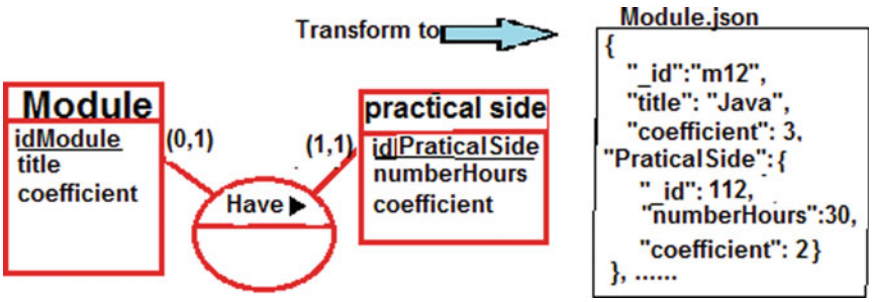

Fig. 6 Example of practice rule 4 to transform Relationship binary type: one to one

Rule 4 of Binary Relations of Type (One to One): In the relational model, this relationship is transformed into a foreign key and makes it possible to verify the existence of a correspondent on the other side. Therefore, we propose that this simple association disappear by adding another attribute to all documents of the collection that hold the relation, whose name is exactly the name of the other entity and whose value is the document of the other entity. The minimum cardinality of 0 will be expressed by the attribute ("required": false) in the schema defining the document in Mongoose and the absence of the internet object in the global object. This transformation will drastically reduce the number of joins and eliminate joins whose primary purpose is to verify the existence of a correspondent on the other side. This transformation is explained by the following figure Fig. 6:

Rule 5 for Binary Relations of Type (One to Many), Which the Entity on the Side of the Cardinality (One) Does not Participate in Another Relation: We propose for this simple association that it must disappear by adding to the documents of the collection that holds the relation another attribute whose name is exactly the name of the other entity and whose value is formed by a list of documents of the other entity. The association on the side of the cardinality "n" is the one that will contain the objects of the other entity. The minimum cardinality of 0 will be expressed in the same way as rule 4. This transformation will considerably reduce the number of joins by storing this join by default in the collection. This transformation is explained by the following Fig. 7:

Rule 6 for Binary Relations of Type (One to Many), Which the Entity on the Side of the Cardinality (One) Participates in Another Relation: We propose that this complex association must be transformed according to the rule 3 transformation presented above. This transformation is explained by Fig. 5.

Rule 7 for Reflexive Binary Relations: This type of relationship is designed in the logic of relational databases to be successively joined with the same table, building an object hierarchy according to the meaning of the relationship. This type is generally related through ordinary binary relations, but it is a very special type in its logic, modeling, and processing. Theoretically, this kind of relation favors an infinite number of joins of such a table with itself by using aliases. Also, the table

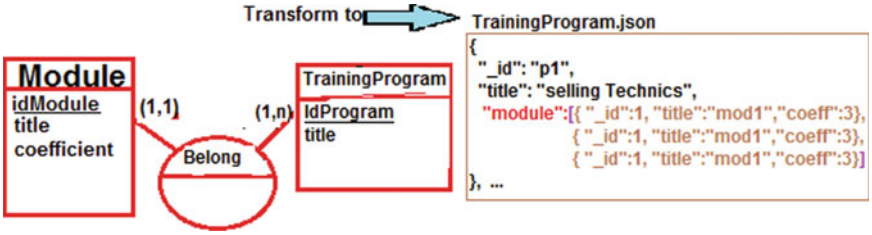

Fig. 7 Example of practice rule 5 to transform relationship binary type: one to many

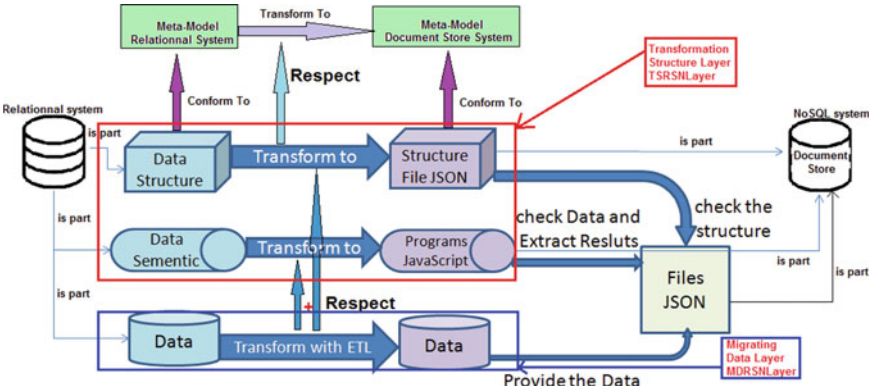

Fig. 8 The architecture of our approach "TMSDRDND Approach"

on the left has a different role than that which will be placed on the other side, even though they are from the same table. But in practice, we make a single join or double join of this table with itself. This relationship is not an ordinary binary relationship. Therefore, we will transform according to rule 3 of the transformation presented above. This transformation is explained by Fig. 5.

4 Proposed Approach

In our approach, called the "TMSDRDND Approach", we will go through two stages: the first, in the "TSRSNLayer" layer, aims to transform the structure data and semantic data into the destination NoSQL model according to a set of rules that respect the models defined previously (meta-model of both systems), and produces a file which controls the structure according to a set of schemas, and another file in JavaScript which contains the functions stored and ready to be used. The second step, in the "MDRSNLayer" layer, consists of using an ETL, which will begin its processing by extracting the data according to the models proposed in the first stage. Then it will perform the necessary transformations according to the

transformation rules of the layer. "TSRSNLayer" and finally it loads the data from the source into JSON files using the results files of the "TSRSNLayer" layer processing. Figure 8 shows the architecture of the "TMSDRDND Approach".

5 Conclusion

Currently, NoSQL is a technology that is emerging in power; it is implemented in environments handling large masses of data such as Google, Yahoo, Twitter, Facebook, etc. Search engines are the first users of these technologies because they need a lot of power. The storage and processing of these volumes of data are the same for social networks, which manage a very strong increase in load due to a large number of users and simultaneous requests.

This approach solves many problems in the previous approaches and it has two phases: the first will transform the data structure and semantics of the relational system into two files in the NoSQL system according to seven transformation rules, which transform the axis principal in a relational system into all relations; and the second phase uses an ETL tool, which we develop according to the first phase to extract, transform, and load data. In this work, we succeeded in proposing another way to transform the different relationships based on their cardinalities. This transformation adopts two models: the first is based on the notion of attributes of objects and their vertical nesting for the restoration and generation of data, while the second aims to create a new collection of data to allow it to self-join for the generation of new information. Also, we managed to find a new model to transform foreign keys, keeping the possibilities offered according to the relational model and respecting the principles of MongoDB. This transformation generates a new attribute in one of the documents with a value of type object.

In future work, we will present an implementation of our approach through a system of tools.

References

1. S. Tiwari, *Professional NoSQL* (Wiley, Hoboken, 2011)
2. G. Brun, *NoSQL vs Relationnel*, (2011)
3. Db-engines.com, DB-Engines - Knowledge Base of Relational and NoSQL Database Management Systems, (2015)
4. L. Rocha, F. Vale, E. Cirilo, D. Barbosa, F. Mourão, A framework for migrating relational datasets to NoSQL. Procedia Comput. Sci. **51**, 2593–2602 (2015)
5. R. Yangui, A. Nabli, F. Gargouri, Automatic transformation of data warehouse schema to NoSQL data base: comparative study. Procedia Comput. Sci. **96**, 255–264 (2016)
6. Y.T. Liao, J. Zhou, C.H. Lu, S.C. Chen, C.H. Hsu, W. Chen, Y.C. Chung, Data adapter for querying and transformation between SQL and NoSQL database. Futur. Gener. Comput. Syst. **65**, 111–121 (2016)

7. A. Bansel, H. González-Vélez, A.E. Chis, Cloud-based NoSQL data migration, in *2016 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)* (IEEE, 2016), pp. 224–231
8. P. Atzeni, F. Bugiotti, L. Cabibbo, R. Torlone, Data modeling in the NoSQL world. Comput. Stand. Interfaces **67**, 103149 (2020)
9. Á. Vathy-Fogarassy, T. Hugyák, Uniform data access platform for SQL and NoSQL database systems. Inf. Syst. **69**, 93–105 (2017)
10. S. Hamouda, Z. Zainol, Document-oriented data schema for relational database migration to NoSQL. in *2017 International conference on big data innovations and applications (innovate-data),* (IEEE, August 2017), pp. 43–50
11. P. Bante, K. Rajeswari, *Big Data Analytics Using Hadoop,* (2017)
12. D.C.T.A. Complete, Multi-criteria analysis between NoSQL databases categories toward a complete migration from relational database. J. Theor. Appl. Inf. Technol. **100**(1) (2022)
13. O. Hajoui, R. Dehbi, M. Talea, Z.I. Batouta, An advanced comparative study of the most promising nosql and newsql databases with a multi-criteria analysis method. J. Theor. Appl. Inf. Technol. **81**(3), 579 (2015)