Mapping rules for schema transformation

SQL to NoSQL and back

Roman Čerešňák
Faculty of Management and Informatics
University of Žilina
Žilina, Slovakia
roman.ceresnak@fri.uniza.sk

Adam Dudáš
Faculty of Natural Sciences
Matej Bel University
Banská Bystrica, Slovakia
adam.dudas@umb.sk

Karol Matiaško
Faculty of Management and Informatics
University of Žilina
Žilina, Slovakia
karol.matiasko@fri.uniza.sk

Michal Kvet
Faculty of Management and Informatics
University of Žilina
Žilina, Slovakia
Michal.kvet@fri.uniza.sk

Abstract— **Efficient way of storing data has always been a key requirement for a properly designed database system. With the growing demand for this property, the first concept of an efficient data storage called relational databases was developed in the 1960s - this type of databases is still used as the primary data storage to this day. In recent years, however, relational databases have failed to deal with two aspects of modern data: large volumes of data and unstructured data. In order to solve the mentioned problems looser databases with a flexible structure and a more efficient way of working with large volumes of data have been created. In many cases, non-relational databases also called NoSQL databases, have become a replacement for relational databases. Several applications require migration from relational to non-relational databases based on suitable properties while there is number of problems associated with this migration. Moving records from a relational database to a non-relational database requires having a structured methodology for transforming existing data. This data transformation from a relational database to a non-relational database (such as MongoDB), is more difficult due to the non-existent transmission standards. The main objective of this paper is to present a proposal for mapping rules of the relation database schema to the NoSQL database schema, specifically NoSQL database of the key-value type, such as MongoDB. The mapping is performed based on the type of relationship that occurs in the relational database, and this process can also be applied in the opposite direction, from the non-relational MongoDB database to the relational database.**

Keywords—Oracle; MongoDB; Mapping rules; ETL; undefined data

I. INTRODUCTION

The relational databases are being developed since the 1960s, which resulted in a stronger theoretical model, a larger range of features and thus a greater use of these databases. The main property of the relational database is the storage of data in highly structured tables while maintaining a normalized form. These two objectives have become a limitation, which (with the growing number of data) has become an obstacle to their use.

The problem with modern data can be identified as the diversity of data and their non-normalization, which means that objects as such are not structured with the use of same formula, number of properties or the same data types. While working with objects, the relational databases cannot be compared with the non-relational ones.

Potential of working with objects and large volumes of data was understood by notable organizations such as Google, Amazon or Microsoft, who chose NoSQL databases as their primary data storage [1]. With the increasing use of non-relational databases, it became important to find a concept of proper mapping of a schema in relational databases to schemas in various non-relational databases. Proper transformation of schemas between relational and non-relational databases enables the integration of data, which is now common practice. The problem of such mapping is currently a large number of types of NoSQL databases. This is reason for several researchers trying to define different types of rules and different mapping methods based on types of NoSQL databases.

Since we dealt with the non-relational database MongoDB in a large part of our research, we also focus on this mentioned database in the presented paper. Data in the non-relational database MongoDB are based on documents, each of which is identified by a specific key [2]. These documents are grouped into collections, which are stored sequentially, and new documents can be added to any collection at any time [3]. By inserting an object into an object, the objects are gradually nested and thus certain layers in data structure are created. There are two ways to model relationships in document-based NoSQL databases - the relationships based on references and the relationships based on insertion. Referential relationships are similar to relational databases, where user's document ID becomes a foreign key in another document. While using insertion relationships, documents are truly nested in other documents so both can be accessed together.

After applying the basic rules, we use a method for data transformation, which is well known by the abbreviation ETL (Extract, Transform, Load). We create the rule set in such a way that the program is able to apply the rules based on the input data and thus autonomously and without difficulty move the data from one type of database to another.

Since it is not possible to design and implement a general module working for all relational databases such as MySQL, MsSQL or PostgreSQL and then apply mapping rules to various types of non-relational databases, such as column-oriented database, graph model and key-value database, we present rules for mapping of relational database Oracle to the non-relational database MongoDB.

978-1-6654-3692-2/21/$31.00 ©2021 IEEE

The rest of presented paper is structured as follows:

- The works, which are focused on schema transformation between relational databases and NoSQL databases are presented in the section II.
- Section III contains proposed mapping rules for schema transformation from SQL to NoSQL and back.
- In the fourth section, we present experimental part of the paper – we experimentally tested proposed mapping rules with the use of NoSQL queries and ETL.

II. Related Work

The importance of a structured schema transformation between various types of databases has led many researchers to exploration of solutions for transformation of relational database schema, which is the most commonly used database today. In the last two decades, we have been able to follow a large number of transformation work focused on relational databases, e.g. [4, 5], in order to meet the growing need for semi-structured and unstructured data. In many works on schema transformation, not only the uniqueness of the newly created data structures is taken into account, but also the semantics that can be included in the relational databases are preserved.

Within the issue of record mapping between relational and non-relational databases, it is possible to find several works proposing techniques for transformation of relational databases into column-oriented NoSQL databases. In [6], the authors mapped entities and association relationships in an improved entity relationship diagram to the HBase database using following three rules. In the first rule, a column family is created for each table, and the primary key of each table becomes the row key in the column family. In the second rule, a new column family is added to another column family to become a supercolumn family. For a M:N association relationship, new column families are created in the relational database and inserted into HBase on both sides, which means that the join table in the relational database is deleted. The purpose of this rule is to maintain the referential integrity of the foreign key mechanisms in the relational databases. The third rule reduces foreign keys by merging them into a super column.

There are several works, in which authors proposed a method for schema transformation from relational database to a NoSQL database based on documents. In [7], the authors proposed a framework for implementing an algorithm that used metadata stored in the relational databases to automatically transform entities and association relationships. In [8], the authors used a separate application called MigDB, which parses the tables in the relational databases, creates a JSON file based on the tables, and then passes the JSON file to the neural network. In addition, the network decides the most appropriate structure for mapping of the JSON file – nested structure of referential structure. This work was done only to map association relationships.

In [9] the authors mapped the relationship of 1:M from relational databases to graph-based NoSQL, specifically the database GraphQL. The starting node in the graphs consists of multiple pages, and the primary key of one page is inserted into multiple pages by preserving the primary key as an edge property. The join table in the relational database is not used to store information as a relationship property. While mapping ternary relationship, the join table and foreign keys of the other tables were removed, but the relationship attributes were preserved as a property of the relationship between the nodes in the graph.

In [10] the authors presented the transformation of relational database into several types of NoSQL, specifically into a few key – value databases, column-oriented database, document database and graph-based database. The authors identified the concepts of each database using defined n-tuples. Subsequently, the authors presented algorithms for performing the transformation and a case study as proof of the concept. This work is complete in the sense that it includes all types of non-relational databases. However, it is not clear that all types of relationships are included in the relational database.

In [11], the authors introduced a data adapter used for querying and mapping between SQL and NoSQL databases. The adapter allows queries from the application and deals with the transformation of the database to a server with a relatively low time difference. Although this work implements a data adapter, it does not provide clear rules for the transformation between two types of databases - a mapping between various non-relational databases or a relational and non-relational database.

In addition to the data structure and relationships, several works have recently been published in the area of transformation implementation. In [12], the authors presented a framework, which supports convenient migration from relational to NoSQL database management system. The framework consists of two modules, namely the migration and data mapping modules. Since the work focuses more on implementation, it does not present a clear transformation existing within the data mapping module. Instead, the article presents the results of experiments with various database operations of the mapping results.

Since the lack of the structure can cause not only ambiguity but also the non-definition of certain values, we also had to deal with this issue. In the research, we applied the method presented in the work [13], where researchers dealt with temporal database architectures, which manage undefined values and propose a comprehensive classification system based on transactions, data reading and indexes. The article deals with techniques for modeling undefined values and covers synchronization processes using data groups. Authors also propose solutions for efficient data acquisition with emphasis on undefined values and states.

An interesting study regarding transformation was published in an article [14] where the authors proposed an approach to model transformation and data migration from a relational database to MongoDB. Their work is divided into four sections where in the first part of the work authors take into account the characteristics of the query and the data characteristics of the relational database. Subsequently, in the second part, they propose an algorithm for transforming the model based on description tags and action tags. The third part is focused on the automatic migration of data to MongoDB based on the result of the transformation of the model, and in the last – fourth – part of the paper a transformation tool is designed and implemented.

III. Proposed Mapping Rules

Although the works mentioned in the previous section brought relevant and important ideas to the problem of schema transformation from relational to non-relational databases, a large number of studies contained ideas dealing only with the associative relationship between individual types of databases. Many of these studies do not deal with the loss of referencing in the schema or loss of values, these methods do not apply backwards compatibility and also do not address the change of values after use of proposed application based on proposed rules. In order to focus on solving the problem, we suggest ways to implement mapping rules when checking undefined values, and then we carry out the process of changing values from the relational database Oracle to the non-relational database MongoDB.

In this part of the paper, we present the rules for transforming schemas from relational databases to non-relational databases. The main objective is to present three basic types of relationships between entities - these are relationships of 1:1, 1:M and M:N types.

A. *Transformation of association relationships of One-to-One (1:1) type from SQL to NoSQL*

Since One-to-One relationship is one of basic relationships, we decided to define two relations for the relational database Oracle. These are relations *student* and *college*. The *student* relation consists of three attributes, these are the primary key *Student_ID*, the name of student *Student_Name* and the address of student *Student_Address*. The *college* relation consists of two attributes and they are *College_ID* which also represents the primary key of the college and the attribute of the name of collage *College_Name*.

To connect these relations, as shown in Figure 1, a relationship called *StudyIn* is created. Based on the E-R diagram from Figure 1, we created a rule presented in the lower part of this figure. Since this is a 1:1 relationship, the ratio of student-collage relationship is in represented in the same way - one object is nested into another object.

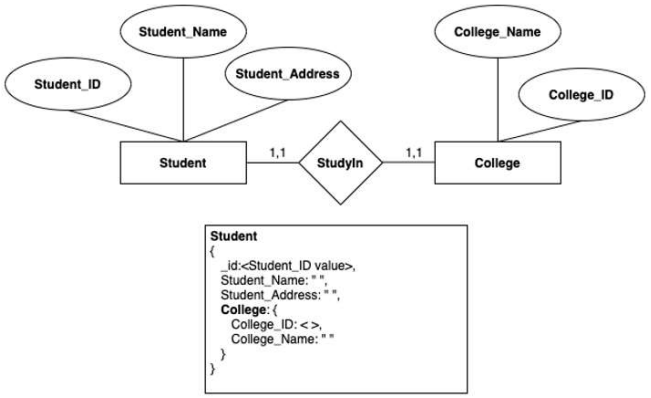

Fig. 1. 1:1 mapping rule for SQL to NoSQL

B. *Transformation of association relationships of One-to-Many (1:M) type from SQL to NoSQL*

The relationships of the type 1:M in relational databases do not complicate diametrically the relationship we presented in the previous step. Since the 1:M relationship differs only in the number of values occurring, the only difference is to increase the number of individual records in json format and apply changes. We present this relationship in the Figure 2 - it is clear, that only one change occurred (in the relationship *StudyIn* where 1 has changed to M). The presented mapping rule is proposed as follows:

- First table is created (in our case it is the table *student*)
- Subsequently, algorithm determines type of relationship based on the number of entities:
  - If number of entities is equal to one, algorithm applies relationships based on the subsection A of this section.
  - In other cases, algorithm adds new objects and creates lists in the newly created collection.

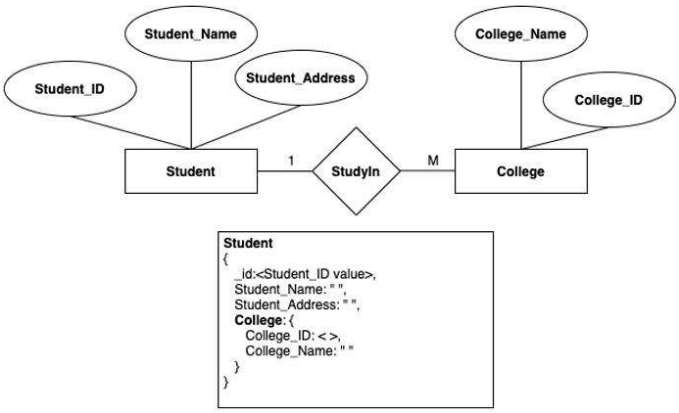

Fig. 2. 1:M mapping rule for SQL to NoSQL

C. *Transformation of association relationships of Many-to-Many (M:N) type from SQL to NoSQL*

The principle of creating a mapping rule for the relationship M:N is straightforward. First, one collection is created, in our case the collection *student*. After creating this collection, the *college* collection is created – this collection already contains values of the primary key of student (specifically the *Student_ID*). Subsequently, a new collection presenting the M:N relationship is created in the *college* collection. Since the *StudyIn* object is a nested object in the *college* collection, it can always retrieve values for references to the *student* and *college* collections. For this reason, we have defined additional values in the *StudyIn* collection – namely the value *City*.

978-1-6654-3692-2/21/$31.00 ©2021 IEEE

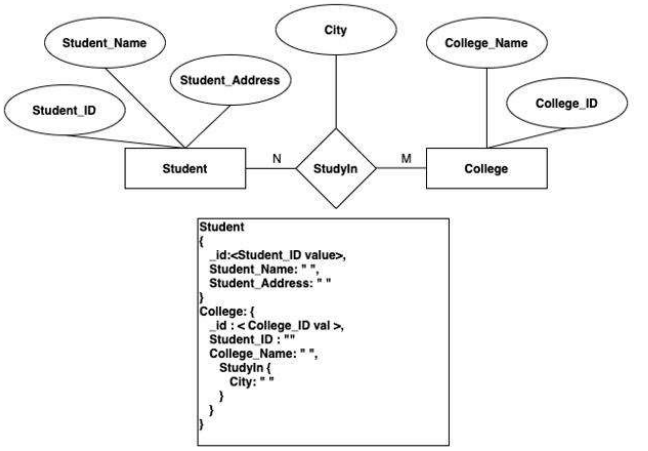

Fig. 3. M:N mapping rule for SQL to NoSQL

Since we only defined the principle of mapping from tables to collections, which means the transformation of the schema from a relational database to a non-relational database, in the next step we create mapping rules in the opposite direction. In this process, the freedom of the structure proves to be an unfavorable property in a backwards processing of schema.

D. Transformation of association relationships of One-to-One (1:1) type from NoSQL to SQL

It is more difficult to design and implement mapping rule when creating a schema from an undefined structure than in the case of creating mapping rules from a relational database to a non-relational one. Since the lack of strict structure of the schema is negative and the data types are key to the database, we need to create a universal model for the change of data types.

With the One-to-One object mapping rule, the number of nested objects in the collection student is verified. In the case the number is equal to 1, then the table student and the table college are created and a 1:1 association relationship is created between them.

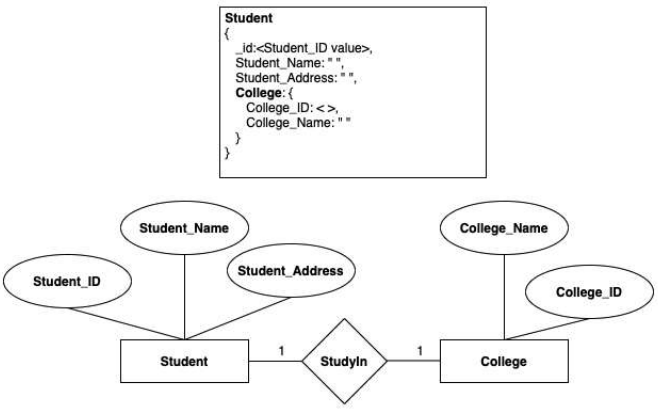

Fig. 4. 1:1 Mapping rule for NoSQL to SQL

E. Transformation of association relationships of One-to-Many (1:M) type from NoSQL to SQL

In the case, that algorithm finds an object in which the nesting of objects takes place, the object in question is verified. If the algorithm detects the number of nested objects greater than one, this suggests that the One-to-One mapping rule is not fit for use with the object - One-to-Many mapping rule is used. In such case, objects from the nested collection are read until the record corresponding to last object of original dataset is created.

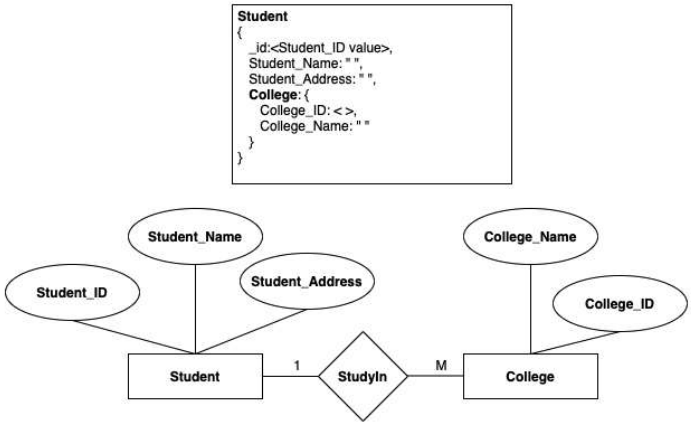

Fig. 5. 1:M Mapping rule for NoSQL to SQL

F. Transformation of association relationships of Many-to-Many (M:N) type from NoSQL to SQL

The transformation of M:N relationships is the most complex when creating a mapping rule due to multiple nesting of objects. In the case of single-layer nesting, as presented in the Figure 6, it is necessary to solve the problem with only one reference to the parent object - finding the collection in the collection and looking at the *_id* value of the parent collections. In the case of multiple nestings, there is a relationship of M:N type, which is associated with another relationship of the same type. There must be multiple use of the relationship F or other relationships applied in sections E and F according to Figures 4 and 5.

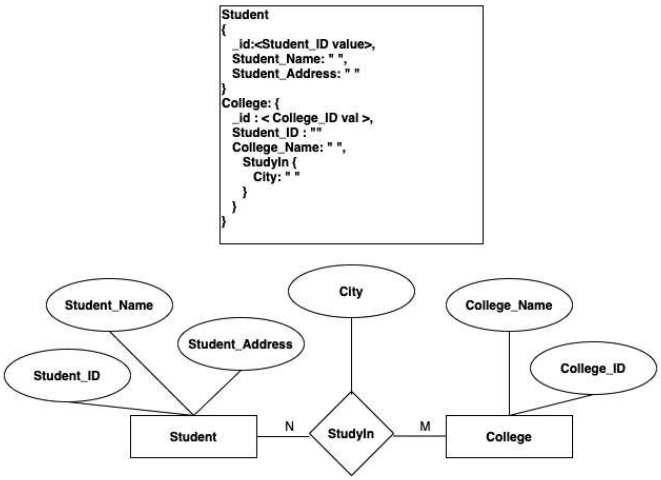

Fig. 6. *M:N Mapping rule for NoSQL to SQL*

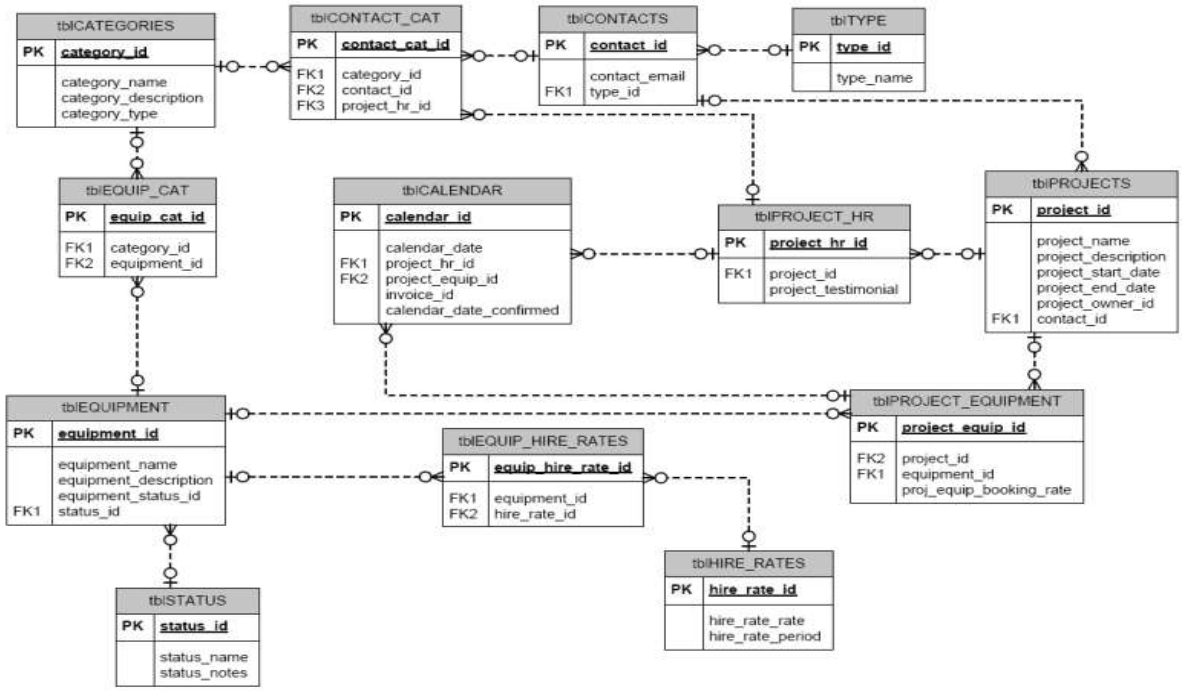

Fig. 7. Complex data model used in the testing of proposed mapping rules

G. Solution to the problem of number of attributes

The lack of strict structure of a data schema is an excellent feature in many respects, but not when creating a schema in a relational database. Since the non-relational database MongoDB does not create a schema, or said more precisely it creates it in a way that is diametrically different that relational, we needed to create a mechanism to manage the number of attributes. Since the number of attributes differs in the non-relational databases, we based our mechanism on the records containing highest number of attributes. That is, the algorithm traverses all the objects in the collection. The algorithm maintains a reference to the object and the number of its properties. If the algorithm finds an object with more attributes, it stores it in a local variable and then continues the search. At the end of the run of algorithm, it contains the object and the number of attributes. The proposed method creates the initial number of objects based on this object.

In the second cycle, the algorithm detects additional attributes and compares them with the attributes of the object in the local variable. If there is an attribute that is not contained in the local variable, the algorithm completes this attribute set and continues until all objects are verified and the remaining attributes are added to the set of attributes. This means, that an object stored in the local variable contains the attributes of all objects of the collection.

In the third cycle, objects and number of values represented in the objects are verified. Since the objects do not contain the same number of attributes (both as each other and as the model object stored in the local variable), algorithm needs to add number of attributes to all objects along with their data type and the name. This represent an operation, which tracks number of uses of various data types in given attribute and based on the number of uses decided which data type is used in the created schema. For example, if the attribute color contains values for 10 objects while six times the values is integer and four times a string, the algorithm assigns data type of the attribute as int. Remaining four values must be type consolidated by ETL.

IV. Experiments For Verification of Proposed Mapping Rules

For testing purposes, we used the model presented in the Fig. 7. The values of attributes are not significant for us at present, since in this paper we do not perform the overall transformation of data but only define the rules, it is not vital to describe them. The purpose of this model is to capture the 1:1, 1:M and M:N relationships. During the initial verification and application of the relationships when mapping the schema from chosen relational database to the non-relational database, the relationships presented in the section III were applied, specifically from subsections A, B and C. Based on these relationships, the schema transformation was performed without further additional modifications.

Since we wanted to verify the backwards compatibility and determine whether a change in structures or a change in data types can affect the whole transformation process, we decided to perform two types of experiments.

The first type of experiment consisted of us moving the schema from the relational database to the non-relational database using the rules defined in the section III. Mapping rules 1, 2 and 3 were enough for us to completely transfer the data scheme. Subsequently, we performed steps based on the proposed rules 4, 5 and 6 and transformed the schema backwards (back to relational database). While comparing pre-transformation and post-transformation schemas, we focused on consistence of schema itself and on consistence of data types.

978-1-6654-3692-2/21/$31.00 ©2021 IEEE

There were no differences - the whole structure was the same when checking the schema itself and data types.

However, the problem occurred when transforming the tables presented in the Table 1. These tables were randomly selected, and their values were randomly changed. Other than these changes, new attributes were added. In these cases, the mapping rules were able to cover all the required values when changing the schema from a relational database to a non-relational database (YES values indicate the success of the mapping (SQL to NoSQL column)).

During the experimental work, we changed and modified data types, changed values or added new attributes to the database. When changing these values, we wanted to move the schema to a relational database, and as can be seen from the results in the Table 1 (specifically NoSQL to SQL column), there were problems with this transformation. Even if the rules created by us revealed the change and worked properly, it was necessary to make additional adjustments - modifying the data type for a schema in a relational database to be compatible with original schemas, and it is also necessary to add new attributes to the database.

Our mapping rules demonstrate the detection of incompatibilities and also know how to identify collisions. Based on our research, we can already send simple SQL views to modify the data schema during data transformation, which will be related to the full compatibility of the process.

Table 1. Properties of Mapping Rules

Mappings	Results of mapping rules		
	Table	***SQL to NoSQL***	***NoSQL to SQL***
1	EQUIPMENT	YES	YES
2	PROJECT_EQUIPMENT	YES	NO
3	PROJECTS	YES	NO

CONCLUSION

In the presented paper, we proposed a set of rules for transforming a schema from relational database into non-relational database NoSQL, specifically the MongoDB. The rules cover the different types of relationships that can appear in the data stored in relational databases, namely associations, inheritance, and aggregation. Along with the types of relationships, cardinalities are also considered.

After defining the rules, we applied the mapping rules to the case study in the relational database, where we were able to create 16 documents for the non-relational MongoDB database from 14 tables with the correct mapping.

Proposed methodology works based on the following principle. The application exports the data schema and passes all tables on the basis of individual records. Then, it creates mapping rules for individual tables, where the 1:1, 1:N or M:N relationships are present. After all the rules have been created, the record mapping process is computed. The mapping works on the principle of ETL and applies the designed rules to the records which enter the system. After successfully mapping the data, the process ends and the rules are stored in additional storage space. When applying the process in reverse i.e., the process from the non-relational database to the relational one, we had to apply a control rule - the rule for monitoring undefined values, which often occurred in tables while working with the non-relational database.

In order to verify the proposed method, we created a data model which contains 1 000 records for each table. Experimental activities are divided into two parts - the first part is the tracking of records and mapping rules from the Oracle relational database to the non-relational MongoDB database, in which the rule of undefined values did not have to be applied. Otherwise, when we created mapping rules from the non-relational MongoDB database to the Oracle relational database, we had to apply the rule of tracking undefined values and we also monitored the poor compatibility between the changed values compared to the original relational database.

In our future research, we set out to improve the method of verification of undefined values and also to create improved mapper. The new mapper should address compatibility between individual data types, which in our current solution is not perfect and sometimes requires human input into the mapping process. As one of the possible variants, we propose to design and implement a process of mapping to all types of relational databases and then focus on all types of non-relational databases, as the popularity of non-relational databases is constantly growing.

ACKNOWLEDGMENT

The research was partially supported by the grant of The Ministry of Education, Science, Research and Sport of Slovak Republic - Implementation of new trends in computer science to teaching of algorithmic thinking and programming in Informatics for secondary education, project number KEGA 018UMB-4/2020.

REFERENCES

[1] L. Rocha, F. Vale, E. Cirilo, D. Barbosa, and F. Mourão, “A framework for migrating relational datasets to NoSQL,” in Procedia Computer Science, 2015.

[2] V. C. Storey and I.-Y. Song, “Big data technologies and Management: What conceptual modeling can do,” Data Knowl. Eng., vol. 108, pp. 50–67, 2017.

[3] P. Atzeni, F. Bugiotti, and L. Rossi, “Uniform access to NoSQL systems,” Inf. Syst., vol. 43, pp. 117–133, 2014.

[4] E. Pardede, W. Rahayu, and D. Taniar, Mapping Methods and Query for Aggregation and Association in Object-Relational Database using Collection. 2004.

[5] E. Pardede, J. W. Rahayu, and D. Taniar, “Object-relational complex structures for XML storage,” Inf. Softw. Technol., vol. 48, no. 6, pp. 370–384, 2006.

[6] C. Li, “Transforming relational database into HBase: A case study,” in 2010 IEEE International Conference on Software Engineering and Service Sciences, 2010, pp. 683–687.

[7] L. Stanescu, M. Brezovan, and D. D. Burdescu, “Automatic mapping of MySQL databases to NoSQL MongoDB,” in Proceedings of the 2016 Federated Conference on Computer Science and Information Systems, FedCSIS 2016, 2016.

[8] G. Liyanaarachchi, L. Kasun, M. Nimesha, K. Lahiru, and A. Karunasena, “MigDB - relational to NoSQL mapper,” in 2016 IEEE International Conference on Information and Automation for Sustainability (ICIAfS), 2016, pp. 1–6.

[9] D. W. Wardani and J. Kiing, “Semantic mapping relational to graph model,” in 2014 International Conference on Computer, Control, Informatics and Its Applications (IC3INA), 2014, pp. 160–165.

[10] M. Freitas, D. Souza, and A. C. Salgado, Conceptual Mappings to Convert Relational into NoSQL Databases. 2016.

[11] Y.-T. Liao et al., “Data adapter for querying and transformation between SQL and NoSQL database,” Futur. Gener. Comput. Syst., vol. 65, pp. 111–121, 2016.

[12] L. Rocha, F. Vale, E. Cirilo, D. Barbosa, and F. Mourão, “A Framework for Migrating Relational Datasets to NoSQL1,” Procedia Comput. Sci., vol. 51, pp. 2593–2602, 2015.

[13] M. Kvet, Š. Toth, and E. Krsak, “Concept of temporal data retrieval: Undefined value management,” Concurr. Comput. Pract. Exp., vol. 32, p. e5399, Jun. 2019.

[14] T. Jia, X. Zhao, Z. Wang, D. Gong, and G. Ding, “Model Transformation and Data Migration from Relational Database to MongoDB,” in 2016 IEEE International Congress on Big Data (BigData Congress), 2016, pp. 60–67.

 978-1-6654-3692-2/21/$31.00 ©2021 IEEE