Conceptual Mappings to Convert Relational into NoSQL Databases

Myller Claudino de Freitas[1], Damires Yluska Souza[2] and Ana Carolina Salgado[1]
[1]*Center for Informatics, Federal University of Pernambuco, Professor Luis Freire ave, Recife, Brazil*
[2]*Academic Unit of Informatics, Federal Institute of Education, João Pessoa, Brazil*

Keywords: Relational Databases, NoSQL Systems, Conceptual Mappings, Data Conversion.

Abstract: Sometimes, data belonging to Relational databases need to be transferred to NoSQL ones. However, the data conversion process between Relational to NoSQL databases is considered as not trivial, since it is necessary to have considerable knowledge about the data models at hand. Regarding the structural heterogeneity underlying this problem, we propose an approach, named as R2NoSQL, which defines conceptual mappings to enhance the data conversion process. In this paper, we present our approach and some implementation and experimental results, which show that, by using the defined conceptual mappings, we obtain a consistent target NoSQL database with respect to a source Relational one.

1 INTRODUCTION

Due to the increasing amount of data generated by user interactions on the Web or by big data requirements, some companies are focusing on using non-relational databases, usually referred to as NoSQL systems, standing for 'Not only SQL' (Han et al., 2011). This term has been used to categorize databases characterized by horizontal scalability, less constrained structure or schema-less, and faster access compared to traditional relational databases (RDBMS) (McMurtry et al., 2013).

Experts comment that despite the rise of NoSQL databases during the past years, NoSQL is not necessarily a replacement for relational databases (McMurtry et al., 2013). Instead, NoSQL databases comply with big or social data demands or specific projects which strain Relational ones. Nevertheless, sometimes, data belonging to Relational databases need to be transferred to NoSQL ones in order to be used in specific projects. However, the data conversion process is not trivial, since it is necessary to have considerable knowledge about the data models at hand.

In this scenario of structural heterogeneity, we define our research problem as follows:

> Let *RDB* be a Relational database and $NSDB = \{NSDB_1, ..., NSDB_n\}$ a set of databases belonging to NoSQL models, where each $NSDB_i$ uses one of the following NoSQL approaches $A = \{Key\text{-}value, Column, Document, Graph\}$. We need to establish conceptual mappings between *RDB* elements and the different data structures underlying $NSDB_i$ in such a way that *RDB* can be converted to $NSDB_i$.

With this in mind, and considering the structural heterogeneity between Relational and NoSQL models, this paper presents the R2NoSQL approach for converting data between the referred models. To this end, it compares the data structures belonging to the Relational model with the four main NoSQL approaches (key-value, columns, documents and graphs), identifying a set of possible conceptual mappings between *RDB* and a $NSDB_i$. Also, it provides a tool prototype, which implements a case study with a Relational database and a Document based NoSQL system. Experiments have been done to evaluate the consistency of the generated mappings by analysing the results obtained from the same set of queries executed on both systems.

This paper is organized as follows: Section 2 introduces some concepts; Section 3 presents the approach; Section 4 describes some obtained results. Related work is discussed in Section 5. Section 6 draws our conclusions and points out future work.

2 NoSQL MODELS

NoSQL systems are a category of databases that do not follow principles of the Relational Model (Han et al., 2011). The term "NoSQL" does not relate to a

Freitas, M., Souza, D. and Salgado, A.
Conceptual Mappings to Convert Relational into NoSQL Databases.
In *Proceedings of the 18th International Conference on Enterprise Information Systems (ICEIS 2016) - Volume 1*, pages 174-181
ISBN: 978-989-758-187-8

specific data model, but to a group of data models that differ from the relational approach and may have in common some features such as: they are usually open-source, distributed and horizontally scalable, and they present schema flexibility or even no schema (Han et al., 2011). NoSQL systems are classified in some categories in which the four main are: Key-value, Columns, Documents, and Graph. Indeed, their implementations may differ from each other, even when the systems belong to a similar category. In order to base our descriptions, we have chosen one example of each NoSQL category. They are briefly discussed in the following.

2.1 Key-value Model

The Key-value Model is the one with the simplest representation. Its structure consists of a list of pairs composed by a key and a value (Istvan et al., 2013).

Usually, Key-value NoSQL implemented systems allow, besides simple data types (e.g., numerals and strings), the use of lists and sets of values of simple types. This is what happens, for instance, in Redis (Redis, 2015), our example of Key-value system. A Key-value system such as Redis tends to support large volumes of data. Since it does not present data schemas, the developer may, by hand, introduce some metadata by naming the keys. On the other hand, it does not support queries to be performed on the data, but only on the search keys. Thus, all access is done through the search keys and only with the key it is possible to access the value. This access usually is accomplished with lower response times, one of its main benefits. This model does not support relationships in terms of reference keys and no referential integrity constraint.

2.2 Column Model

At a first sight, this model may be considered as similar to the Relational one, since it is also organized in terms of rows and columns. However, this approach deals with data in a non normalized way, i.e., by allowing nesting of tables inside tables (Lakshman and Malik, 2010). In this approach, rows do not store a tuple, but a set of attributes of the same type, while the set of attributes of a column contains the information from a given instance. Such feature allows queries to be performed more efficiently, although when recovering a complete instance it may become more costly.

Another important concept regards a “family of columns”, which means a set of instances of a given entity. In this structure, it is possible to have non-atomic attributes through the representation of value lists. Instances may have a different number of attributes, since there is no need to book storage space for null values. Also, there is no need to use join operations in order to query diverse entities.

2.3 Document Model

In Document Model, the data entities are grouped in documents as objects, which are composed by keys (properties) and values. These documents are usually serialized in JSON syntax (McMurtry et al., 2013).

A document is a collection of objects that are related to a data instance. The various documents belonging to the same data domain are stored in a collection of documents.Considering the MongoDB document system (Mongo, 2015), an instance key (called as an “objectId”) can be set at persistence time, or may have its value generated randomly by the database. It can provide uniqueness values for other fields by the specification of an index.

This model allows more complex queries involving different collections of documents. To this end, it is necessary that a document has a DBRef (Database Reference) to another related document or establish a reference. Despite allowing references, DBRef does not guarantee referential integrity constraint. A query may consider these references or use data embedded within the same document.

2.4 Graph Model

The Graph model is mainly concerned with representation and access, where data items are connected by relationships by means of a graph structure (McMurtry et al., 2013). The elements underlying a graph are, namely (McMurtry et al., 2013): nodes, edges and properties. Nodes correspond to data instances, edges refer to maintained relationships among node instances, and properties relate to data values. Some systems of such category allow the definition of their properties with the guarantee of unique values. One example regards the Neo4j system (Neo4j, 2015).

Nodes and edges can contain labels (terms which indicate a category) that classify them into more specific groups. For the nodes, these labels can be used to differentiate instances. On edges, labels may also be used to determine the type of relationship that is occurring.

An edge has an input node and an output node linking them. This feature, besides supporting references, also guarantees referential integrity by ensuring that the input node always makes reference

to the output node. The access keys to the nodes are automatically set by the system. However, it is possible to establish unique constraints for other node properties.

3 THE R2NoSQL APPROACH

In this section, we present some definitions along with the proposed approach.

3.1 Some Definitions

At first, we provide some definitions regarding the concepts underlying E-R and Relational models that we consider in our approach. Since we need to think about mappings between concepts, we define what we consider by "Concept" in each one of the working data models. Regarding the E-R Model, we may summarize a Concept as follows.

Definition 1 – E-R Concept. The set of concepts of an E-R conceptual model we are dealing with are C_E = {*Entity, Simple Attribute, Multi-valued Attribute, Composed Attribute, Relationship, Specialization*}.

In the light of the Relational Model, we define a Concept as follows.

Definition 2 – Relational Concept. Concepts of a relational structure are C_R = {*Table, Simple Attribute, Primary Key (PK), Foreign Key (FK)*}.

As discussed in Section 2, we observe that each NoSQL database model has specific data structures. Thereby, we provide the definition of the main Concepts of the four categories of NoSQL systems described previously.

Definition 3 – Key-value NoSQL Concept. A concept in a Key-value Model may be C_K = {*Search Key, Value, Value List, Value Set*}.

Definition 4 – Column NoSQL Concept. In a Column Model, a concept may be C_C = {*Column Family, Line, Column, Value Set, Value List, Primary Key (PK)*}.

Definition 5 – Document NoSQL Concept. In a Document Model, a concept may be C_D = {*Document Collection, Document, Field, Embedded Field, Field List, ObjectId, DBRef*}.

Definition 6 – Graph NoSQL Concept. A Graph model concept C_G = {*Label, Data Node, Property, Property Set, Id, Edge, Value Set, Value List*}.

Data indeed are instantiated differently in each one of the referred databases. Nevertheless, we can think about a data item or a data instance, in a general way, as follows.

Definition 7 - Data Item. A data item is an instance or an individual of a real entity in the data set at hand.

In the Relational Model, a data item is a tuple. In NoSQL approaches, it can be a data node, a data document, a column of a column family or simply a value from the key value model.

3.2 Our Proposal

As discussed in the previous sections, each database model has specific data structures and concepts, what provides structural heterogeneity conflicts among them. These conflicts occur because different choices of construct representation or integrity constraints are adopted in accordance with the options underlying each data model. Thereby, in this work, the task we are dealing with is concerned with what is necessary to convert concepts of a given *RDB* (a Relational database) to a $NSDB_i$ (a NoSQL one). Thus, it becomes necessary to specify conceptual mappings between concepts $C_r \in RDB$ and concepts $C_n \in NSDB_i$.

Our proposal, named as R2NoSQL approach, is based on three aspects: (i) defining conceptual mappings between *RDB* and $NSDB_i$; (ii) using these conceptual mappings to allow metadata and data conversion between the referred databases, and (iii) classifying source tables to help understanding their meaning in the database design. In the following, we provide the definitions underlying these issues.

3.2.1 Conceptual Mappings

Our approach deals with the structural heterogeneity of the data models and some aspects of database design. In order to cope with these issues, our mapping language handles the different existing concepts, which belong to the data models, but as a design reference, we consider some concepts not only from the Relational model but also from the E-R conceptual model. Thereby, we consider concepts from the Conceptual E-R model, which are not directly implemented in a Relational database, but they are close to real world and can be implemented in NoSQL systems. This conceptual mapping is the base for our conversion solution and without it, the process could not happen. These concepts regard particularly the composed and multi-valued attributes, and also specializations. By establishing that, we deal with a source model and a target model and we define the set of possible source Concepts, to be considered in a Mapping, as the following:

Definition 8 – Source Concept. A source concept C_s is the set of possible E-R or Relational concepts which may be used to compose a Mapping. Thus, $C_S = C_E \cup C_R$. Proceeding with the union operation, the final set results in C_S = {*Entity, Simple Attribute, Multi-valued Attribute, Composed Attribute, Relationships, Specialization, Table, Primary key (PK), Foreign key (FK)*}. Since a conceptual *Entity* always results in a relational *Table*, we abstract both ones only in the concept *Table*.

In the same way, we establish a target Concept, as the following.

Definition 9 – Target Concept. A target concept C_T is the set of possible NoSQL concepts which may be used to compose a Mapping. $C_T = C_K \mid C_C \mid C_D \mid C_G$.

Thus, the set of target concepts is composed by the possible concepts which belong to one of the NoSQL systems instantiated by a specific model.

With these definitions in mind, we define, firstly, in a general way, a Conceptual Mapping, as follows.

Definition 10 – Conceptual Mapping. A conceptual mapping M represents an association between a concept C_S and a concept C_T of a given $NSDS_i$, where $NSDS_i \in$ A, and A = {*Key-value, Column, Document, Graph*}. M defines a level of similarity between C_S and C_T.

A conceptual mapping M may be understood as a way of converting a given C_S into a C_T. Depending on the target $NSDS_i$, to a given C_S, there may be no corresponding C_T, i.e., there may be no concept in the target model that can be used for data conversion. When this fact happens, we point it as an empty or non existing target concept (∄).

Based on the previous definitions, we establish specific conceptual mappings between C_S and C_T, according to the $NSDS_i$ at hand. To this end, we consider the possibility of employing a table denormalization technique, which is the process of adding redundant data or grouping data, previously fragmented in a number of relational tables. Thereby, we may have nested tables or multi-valued attributes in one or more target structures, which may be sets, lists, documents or other ones, depending on the data model.

Let *RDB* be a source database composed by C_S and $NSDB_K$, a Key-value NoSQL system, composed by C_K. Specific structural conceptual mappings between C_S and C_K may be defined, as follows.

$RDB:Table \equiv NSDB_K: \nexists$
$RDB:SimpleAttribute \equiv NSDB_K:Value$
$RDB:ComposedAttribute \equiv NSDB_K:ValueList$
$RDB:Multi\text{-}valuedAttribute \equiv NSDB_K:ValueList$
$RDB:PK \equiv NSDB_K:SearchKey$
$RDB:FK \equiv NSDB_K: \nexists$
$RDB:Specialization \equiv NSDB_K:ValueSet$

Regarding data items, we may establish a mapping in the following way:

$RDB:DataItem \equiv NSDB_K:Value \mid NSDB_K:ValueList$

Although there is no corresponding concept to a *Table*, it is possible to simulate such concept by using composed search keys. In this case, keys are composed by a prefix together with the name of a given property, in such a way that we may identify to which entity it is associated. Indeed, it is not a defined standard, but one of our proposals.

The representation of relationships occurs with the storage of the search key values of a given data item inside another one. In many-to-many relationships, this happens in both sides of the data items at hand.

Now let *RDB* be a source database composed by C_S and $NSDB_C$, a Column NoSQL system, composed by C_C. Specific structural conceptual mappings between C_S and C_C are defined, as follows.

$RDB:Table \equiv NSDB_C:ColumnFamily$
$RDB:SimpleAttribute \equiv NSDB_C:Column$
$RDB:ComposedAttribute \equiv NSDB_C:ValueSet$
$RDB:Multi\text{-}valuedAttribute \equiv NSDB_C:ValueList$
$RDB:PK \equiv NSDB_C:PK$
$RDB:FK \equiv NSDB_C: \nexists$
$RDB:Specialization \equiv NSDB_K: ValueSet$

Regarding data items, we may establish a mapping in the following way:

$RDB:DataItem \equiv NSDB_C:Line$

In terms of relationships, a $NSDB_C$ allows their implementation by two options: (i) a column family may compose information from different but related tables; or (ii) data items may have a reference to other data items by storing the target primary key. The former is the most common option, since it allows a better response time.

Now let *RDB* be a source database composed by C_S and $NSDB_D$, a Document NoSQL system, composed by C_D. Structural conceptual mappings between C_S and C_D are defined, as follows.

$RDB:Table \equiv NSDB_D:DocumentCollection$
$RDB:SimpleAttribute \equiv NSDB_D:Field$
$RDB:ComposedAttribute \equiv NSDB_D:EmbeddedField$
$RDB:Multi\text{-}valuedAttribute \equiv NSDB_D:FieldList$
$RDB:PK \equiv NSDB_D:ObjectId$
$RDB:FK \equiv NSDB_D:DBRef \mid NSDB_D:EmbeddedField$
$RDB:Specialization \equiv NSDB_D: EmbeddedField$

Regarding data items, we may establish a mapping in the following way:

$RDB:DataItem \equiv NSDB_D:Document$

Relationships are implemented by defining object references between objects belonging to documents. Thereby, queries may take into account these references to get related objects information.

Now let *RDB* be a source database composed by C_S and $NSDB_G$, a Graph NoSQL system, composed by C_G. Specific structural conceptual mappings between C_S and C_G are defined, as follows.

RDB:Table ≡ $NSDB_G$*:LabelNode*
RDB:SimpleAttribute ≡ $NSDB_G$*:Property*
RDB:ComposedAttribute ≡ $NSDB_G$*:ValueSet*
RDB:Multi-valuedAttribute ≡ $NSDB_G$*:ValueList*
RDB:PK ≡ $NSDB_G$*:Id*
RDB:FK ≡ $NSDB_G$*:Edge*
RDB:Specialization ≡ $NSDB_K$*:ValueList*

Regarding data items, we may establish a mapping in the following way:

RDB:DataItem ≡ $NSDB_G$*:Node*

In the next section, we provide the way we classify identified tables in a given *RDB*.

3.2.2 Table Classification

Regarding the set of concepts $C_R \in RDB$, the main one is always a *Table*. Since a table may be the result of E-R conceptual entities, relationships, specializations, multi-valued or composed attributes, we need to understand what a *Table* means, and its importance, to the *RDB* at hand. We have defined a classification of the source Tables as follows.

- **Main Tables:** These are the main tables in a RDB design. They usually correspond to entities found in the conceptual model.
- **Subclasses:** These tables are a complement to the definition of a main table. They represent specializations of the main tables, but do not exist independently.
- **Relationships:** This classification typifies a specific kind of table, which implements a many-to-many (N:N) relationship in the conceptual model.
- **Common Tables:** The other types of tables are defined as common in a source RDB schema.

Our proposed algorithms take into account such table classification in order to identify the conceptual mappings to be used.

3.2.3 Conversion Algorithm

Based on the specified conceptual mappings between C_S and C_T, and on the table classification, some algorithms have been developed to allow data conversion. A main algorithm, named as *Algorithm1-Data Conversion*, receives a *RDB* enriched with a Table Classification as input and generates a $NSDB_i$ as output.

Algorithm1: Data Conversion.

```
Input: RDB rel;
Output: NSDSi ns;
Begin
    //Looks for main tables
1:  For Each table of rel Do
2:  If (table.classification() is "main")
    Then
3:      get all table attributes to object
4:      For Each table referencing table Do
          //looks for related tables
5:        goDeep(table, object);
            //persists object on ns
6:        persist(object) ;
7:      End For;
8:    End If;
9:  End For;
End Data_Conversion;
```

In our approach, the input RDB is composed by its metadata and data. Tables were already classified and this classification is also used as input. Based on that, the algorithm verifies the kinds of existing tables and, for each type, extracts the set of data items. Through references (FKs), data tables are traversed and analyzed. At such verification time, decisions are taken according to each type of table.

Algorithm2: goDeep.

```
Input: table, object;
Output: table, object;
Begin
    //Looks for tables that reference table
1:  table2 := findTableDeep(table);
    2:  Do Switch table2.classification
3:    case "common":
4:      get all table2 attributes to object;
5:      goDeep(table2);
6:    case "subclass":
7:      get all table2 attributes to object;
8:      goDeep(table2);
11: End Switch;
    //looks for related tables
10: goUp(table, object);
End goDeep;
```

Main tables constitute the basis for the analysis. With a main table at hand, the algorithm verifies if it is related with other ones. In this case, it calls another algorithm (Algorithm2–goDeep) where existing relating tables are identified. Regarding these selected tables, some options may be taken, namely: (i) If the table is another main table, no other procedure is accomplished because, later, the opposite direction of the relationship will be considered. At this later time, the second table will refer to the first one; (ii) If the table is a relationship

table, no action is taken because, only when all the data items are persisted, relationships can be analyzed; (iii) If the table is a common one, it is possible to add its attributes in that main table as part of its structure; (iv) If the table is a subclass, then its attributes are added to the main table. It is understood that it is a specialization of the main one.

Each time a new table is found in-depth analysis, the algorithm selects this table and repeats the process until there are no more tables, or a stop condition happens. This process is responsible for the denormalization of the data, in which the data related to the main table are included in a data item.

Algorithm3: goUp.

```
Input: table, object;
Output: object, reference_list,
referenced_list;
Begin
    //Looks for tables that reference table
1:  table2 := findTableUp(table);
2:  Do Switch table2.classification
3:    case "common":
4:      get all table2 attributes to object;
5:      goUp(table2);
6:    case "subclass":
   //saves data items to set relationship
7:      reference_list.add(table);
8:      referenced_list.add(table2);
8:      goDeep(table2);
9:    case "relationship": break;
10:   case "main":
11:     reference_list.add(table);
12:     referenced_list.add(table2);
13:  End Switch;
End goUp;
```

After the identification of in-depth relationships, the tables that the main table refers are searched up (*Algorithm3 – goUp*). According to the identified relationships, one of the following options will be considered: (i) to capture the attributes and move up or (ii) to save identified instances in a list. This function separates tables in which there could be composed or multi-valued attributes. It may also show that a new entity has been found, and a relationship should happen. The procedure is repeated until a stop condition is reached.

After the main table and its related tables of a data item are analyzed, the whole set of attributes is persisted as an entity in the target database (*Algorithm1*, line 6). Just after all instances have been persisted, the algorithm must define the relationships among them.

To establish the one-to-one or one-to-many relationships, generated lists (when a main table mentioned another one) are used. These lists are included in the data items that have references, and the data items that are referenced. For each type of implemented target database, the algorithm must implement a specific procedure. In this work, we show one regarding the MongoDB system. This algorithm, named as *oneTo,* was implemented to provide DBRef storage. It stores in the corresponding document of Table 1 a reference to list2, and in the one corresponding to Table 2, a reference to list1. In terms of many-to-many relationships, it is necessary to identify the double meaning of these references (*manyToMany* algorithm). In MongoDB case, a DBRef of each document involved in the corresponding document is stored. For instance, the student Bill Gates held a publication. Thus there is a publication document reference in the student's document, and there is a student document reference in the publication document. Other solution would be to embed all related data into one document. However, this approach can let the execution of queries harder. For instance, if student documents contain publication information, more effort would be necessary to retrieve people involved in a specific publication.

4 RESULTS AND EXPERIMENTS

In this section, we present some implementation and experimental results.

4.1 Implementation and Example of use

We have developed the R2NoSQL approach in the Java language. The main functional requirements underlying the tool's development are the following:

- **Set Source Database:** it includes the definition of the relational DBMS to be used as well as the metadata and data extraction step.
- **Classify Table:** The tables extracted from the Relational database will be classified by the user.
- **Set Target Database:** The user chooses the target NoSQL database.
- **Execute Data Conversion:** It analyzes the extracted metadata and data from the source database, verifies tables' classification and possible conceptual mappings, identifies the target NoSQL concepts and persists corresponding data in the target database.

In this current version, we have developed a prototype which deals with a *RDB* (e.g., MySQL) as a source database and a $NSDB_D$ as a target one. To the latter, we have used the MongoDB system.

We provide an example in the following. As source *RDB*, we have used a database with 15 tables

(e.g., Person, Publication, Student). Among them, there are some relationships (e.g., between Person and Publication), and some specializations (e.g., Student is a specialization of Person).

The Table classification is accomplished by the user, since, in this version, we have a semi-automated tool with respect to this functionality. Thus, after extracting the source *RDB* metadata, the tool asks the user to classify the extracted tables.

After table classification, the R2NoSQL tool is able to proceed with the data conversion, in accordance with the defined algorithms (Section 3.2.3). The tool selects instances of each table classified as main, storing the table attributes and their associated values. This happens according to the existing mappings. When this process finishes, the tool looks for instances related to what was analyzed as a complement to the main tables. These related tables can be a representation of a complex value as a composed attribute, a multi-valued attribute or even a specialization.

Considering a fragment of the source database at hand, we show some instances belonging to tables Person and Student in Table 1 and Table 2, respectively. Taking into account the data presented in Table 1 and Table 2, the tool produces a document as depicted in Figure 1. The resulting collection of documents is named by using the name of the main table of that entity. The attributes and values of the tables were converted into document fields. However, new fields were introduced as depicted in Figure 1: AR_master regards the key attribute of table Master, a specialization of Student; Person_X_Publication represents the many-to-many relationship between Person and Publication.

4.2 Experiments

We have conducted some experiments to verify the effectiveness of our approach. The goal of our experiments was twofold: (i) to check whether a set of queries formulated and executed in a *RDB* may be either formulated and executed in a generated $NSDB_D$, and (ii) to verify if the query results obtained from both databases are similar, i.e., if the used mappings and algorithms have generated a consistent target $NSDB_D$. In order to verify the latter goal, we checked both obtained query results and compared the ones obtained in a $NSDB_D$ with respect to the corresponding ones from a RDB.


```
{
    "_id": {
        "$oid": "55a6a2cc13044b9cd70b4f43"
    },
    "CPF": "74852963214",
    "RG": "10987312",
    "name": "Bill Gates",
    "Bdate": "10-28-1955",
    "Natural From": "Washington",
    "nationality": "American",
    "e_mail": "gates@ms.com",
    "url": "http://www.gatesnotes.com/",
    "user": "gates",
    "pword": "gates",
    "profile": "U",
    "type": "master",
    "AR": "790099",
    "Additional_info": "Co-author of 19 articles.",
    "AR_master": "790099",
    "Adm_semester": "1",
    "Adm_year": "1981",
    "Egressiondate": "03-19-1981",
    "Person_X_Publication": [
        {
            "Publication": {
                "$ref": "Publication",
                "$id": {
                    "$oid": "55a6a2d413044b9cd70b4f4c"
                }
            }
        }
    ]
}
```


Figure 1: Document obtained after data conversion.

The working data scenario was the same presented in the example of Section 4.1. This database was populated with 45 tuples, and queries were specified to be executed over them. Table 3 shows an examples of query used in our experiments, which shows all professor attributes according to his name. It is presented in SQL and in MongoDB query language.

The first experiment goal was accomplished, since, by using the tool, we could submit and execute the same set of queries in both source and target databases.

Regarding the latter goal, we have compared the obtained query results in terms of data items (instances) and their properties. For each query submitted and executed in MongoDB, we measured the degree of similarity of the answers with respect to the set of answers obtained in the relational database (which acted as a gold standard).

All queries returned similar instances with identical property values (100%). Differences were

Table 1: Table *Person* with a tuple.

CPF	RG	name	Bdate	Natural From	nationality	e_mail	url	user	pword	profile
74852963214	10987312	Bill Gates	10-28-1955	Washington	American	gates@ms.com	http://gatesnotes.com	gates	gates	U

Table 2: Table Student with a tuple.

AR	CPF	Additional_info	Adm_semester	Adm_year	Egressiondate
790099	74852963214	Co author of 19 articles.	1	1981	03-19-1981

obtained only in terms of query results presentation, but not regarding the set of obtained data items.

Therefore, we can see that the goals of this experiment were achieved. It was possible to require the same information from both source and target databases. In addition, it was possible to obtain the same set of query results from both ones.

Table 3: A query example used in the experiment.

SQL	MongoDB query language
select pe.*, pf.* from Person pe inner join Professor pf on pe.cpf=pf.cpf inner join IC_Professor i on pf.cpf=i.cpf where pe.cpf = '95175368429' union select pe.*, pf.* from Person pe inner join Professor pf on pe.cpf=pf.cpf inner join Invited_Professor ip on pf.cpf= ip.cpf where pe.cpf = '95175368429'	db.Person.find({cpf: "98632541754", $or:[{type: "ic"},{ type: "invited"}]}, {cpf:1, rg:1, name:1, birth_date:1, naturalness:1, nationality:1, user:1, password:1, profile:1, e_mail:1, type:1, additional_info:1})

5 RELATED WORK

Data conversion approaches regarding Relational and NoSQL models have been tackled. Zhao et al. (2014) propose an automatic approach for converting relational database schemas to NoSQL ones, which establishes conceptual rules for the denormalization of the original data. Potey et al. (2015) provide a tool to perform data conversion, in which the target database is an equivalent relational schema in a Document structure. Karnitis and Arnicans (2015) instead provide a semi-automatic approach, which allows a comprehension of the relationships that the tables carry one over the other by a classification strategy. Mpinda et al. (2015) present a data conversion process that aggregates data tables, which are analyzed along with the established relationships.

Our proposal extends some of these concepts. We provide a denormalization technique and we deal with some kinds of conceptual relationships, by producing references when possible. We have a table classification strategy to enrich the overall process. Finally, our approach may be applied to any of the target NoSQL models.

6 CONCLUSIONS

We presented the R2NoSQL approach, which allows data conversion between relational and NoSQL databases. This approach is based on conceptual mappings defined between structural concepts from relational and NoSQL ones.

Experiments have shown that obtained NoSQL database is consisted with the source relational one, by executing the same set of queries in both source and target databases. In fact, they produced similar query results.

As future work, some enhancements will be done: (i) the tool will be extended to accomplish data conversion by considering other categories of NoSQL systems, and (ii) an automated query conversion process will also be taken into account.

REFERENCES

Han, J., Haihong, E., Le, G., Du, J., 2011. Survey on NoSQL database. In: *Pervasive computing and applications (ICPCA)*, 2011 6th international conference on. IEEE. p. 363-366.

Istvan, Z., Alonso, G., Blott, M., Vissers, K., A flexible hash table design for 10Gbps key-value stores on FPGAs. In: *Field Programmable Logic and Applications (FPL)*, 2013 23rd International Conference on. IEEE. p. 1-8.

Karnitis, G. and Arnicans, G., 2015. Migration of relational database to document-oriented database: Structure denormalization and data transformation. *Communication Systems and Networks (CICSyN)*, 7th International Conference on Computational Intelligence. p. 114–118.

Lakshman, A., Malik, P., 2010. Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, v. 44, n. 2, p. 35-40.

McMurtry, D., Oakley, A., Sharp, J., Subramanian, M., and Zhang, H., 2013. *Data access for highly-scalable solutions: Using sql, nosql, and polyglot persistence. Microsoft patterns & practices*.

MongoDB, 2015. Available at https://www.mongodb.org/. Last access on December, 2015.

Mpinda, S. A. T., Maschietto, L. G., and Bungama, P. A., 2015. From relational database to columnoriented nosql database: Migration process. *International Journal of Engineering Research & Technology (IJERT)*, 4. p. 399–403.

Neo4j, 2015. Available at http://neo4j.com. Last access on December, 2015.

Potey, M., Digrase, M., Deshmukh, G., and Nerkar, M., 2015. Database migration from structured database to non-structured database. *International Conference on Recent Trends & Advancements in Engineering Technology (ICRTAET 2015)*, p. 1–3.

Redis, 2015. Available at http://redis.io/. Last access on December, 2015.

Zhao, G., Lin, Q., Li, L., Li, Z. 2014. Schema Conversion Model of SQL Database to NoSQL. In: P2P, Parallel, Grid, *Cloud and Internet Computing (3PGCIC)*, 2014 Ninth International Conference on. IEEE. p. 355-362.