data

MDPI

Article

Transformation of Schema from Relational Database (RDB) to NoSQL Databases

Obaid Alotaibi [1] and Eric Pardede [2,*]

[1] Department of Computer Science, College of Science and Arts, Sajir Campus, Shaqra University, Sajir City 11951, Saudi Arabia; obaid@su.edu.sa

[2] Department of Computer Science and Information Technology, School of Engineering and Mathematical Sciences, Melbourne Campus, La Trobe University, Victoria 3086, Australia

* Correspondence: e.pardede@latrobe.edu.au

Received: 28 October 2019; Accepted: 25 November 2019; Published: 27 November 2019

Abstract: Relational database has been the de-facto database choice in most IT applications. In the last decade there has been increasing demand for applications that have to deal with massive and un-normalized data. To satisfy the demand, there is a big shift to use more relaxed databases in the form of NoSQL databases. Alongside with this shift, there is a need to have a structured methodology to transform existing data in relational database (RDB) to NoSQL database. The transformation from RDB to NoSQL database has become more challenging because there is no current standard on NoSQL database. The aim of this paper is to propose transformation rules of RDB Schema to various NoSQL database schema, namely document-based, column-based and graph-based databases. The rules are applied based on the type of relationships that can appear in data within a database. As a proof of concept, we apply the rules into a case study using three NoSQL databases, namely MongoDB, Cassandra, and Neo4j. A set of queries is run in these databases to demonstrate the correctness of the transformation results. In addition, the completeness of our transformation rules are compared against existing work.

Keywords: relational database; NoSQL databases; transformation rules

1. Introduction

Relational database (RDB) stores data into tables in a normalized and structured form, which has since become a limitation for RDB with a fast evolution of applications. RDB cannot deal with un-normalized data and massive size, which makes companies like Google, Facebook, and Amazon choose NoSQL database as the option of their data storage [1]. In addition, NoSQL database can support object-oriented paradigm in a better way in comparison to RDB [2].

With the increasing usage of NoSQL database, it is important to find a way to map RDB schema into the schema of various NoSQL databases. A correct transformation between schemas will enable data integration, which is common in current applications. It is the aim of this paper to propose these transformation rules. The rules however, have to be differentiated depending on the NoSQL database types, which can be grouped into column-based, document-based, graph-based, and key-value-based NOSQL.

Data in column-based NoSQL is stored as columns rather than rows as in RDB [3,4]. Column-based NoSQL supports three column types, which are column, column family, and super column family. A column has a column name and value, a column family is a group of column name and value that is defined by a row key, and a super column family is a group of column family which is also defined by a row key.

Data in document-based NoSQL is stored as document, each of which is identified by a particular key [4–6]. The document is grouped into collections that are sequentially stored, with new documents able to be appended into the collection at any time [7]. There are two ways to model relationships in document-based NoSQL, which are referencing and embedding. Referencing is similar to RDB when the ID of a user document becomes a foreign key in another document. For embedding, the documents are contained in another document and can be accessed together.

Graph model consists of node, node property, label, relationship, and relationship property. Data in this model is stored as a node, and in some cases as a relationship property. The edge (relationship) is used to connect a node to another node for ensuring the referential integrity [4]. In addition, the relationship in the graph model has to contain a start node and end node [8].

We will not cover the key-value-based NoSQL in this paper as we will leave this for our future work. There is no existing work on schema transformation from RDB to key-value-based NoSQL and hence, we will not be able to do comparative analysis in this area.

The remaining of this paper is organized as follows. The related works for schema transformation from RDB into NoSQL database families is presented in Section 2. Section 3 contains our proposed transformation rules from RDB to three types of NoSQL databases. In Section 4, we examine the proposed rules by applying queries on three different NoSQL DBMSs (Database Management Systems). A summary of this paper and future work are presented in Section 5.

2. Related Works

The importance of a structured way to transform schemas between one database to another leads many researchers to investigate solutions for converting the schema of RDB, which is the most widely-used database, into various database models. In the last two decades, we have witnessed numerous works in the area of RDB schema transformation [9,10] to accommodate the increasing need for semi-structured and unstructured data. In many works of schema transformation, not only the uniqueness of the new data structures is considered, but also the semantics that can be covered in RDB are also maintained.

There have been several works that proposed techniques to transform RDB to column-based NoSQL. In [11], the authors mapped the entities and association relationships within an enhanced entity-relationship ER diagram to HBase database using three rules. In the first rule, a column family is created for each table with the primary key of each table becoming a row key in the column family. In the second rule, a new column family is added to another column family to become a super column family. For many-to-many association relationship in RDB, new column families are created and inserted on both sides in HBase, which means that the join table in RDB is deleted. This rule aims to maintain the referential integrity of foreign key mechanisms in RDB. The third rule reduces foreign keys by merging them into a super column. The decision is based on the use of data pointed by the foreign keys, whether they can be used independently or simply as an accessory to other data.

In [12], the authors mapped the one-to-one, one-to-many and many-to-many association relationships in RDB by two steps. In the first step, the row key for column families is chosen based on the expected entry pattern of users. Following that, in the second step the table that had a relationship with another table becomes merged into one super column family.

In [13], the authors used four steps for mapping the one-to-one and one-to-many relationships from RDB into HBase. Initially, the data is converted to a denormalized level, followed by merging adjacent tables, then a row key is determined to be optimal for various accessing patterns, and finally the indexing on the HBase tables are maintained.

There are also several works that proposed transformation of the schema from RDB to document-based NoSQL. In [2], the authors proposed a framework to implement an algorithm that used a metadata stored in RDB for automatic transformation of the entities and association relationships. In [14], the authors used a standalone application named MigDB that analyzes tables in RDB, creates a JSON file based on the tables, and then passing the JSON file to a neural network.

Moreover, the network made a decision on the most appropriate structure to map the JSON file, whether it will be an embedding or referencing structure. This work was done for mapping association relationships only.

In terms of the transformation to a graph-based NoSQL database, the authors in [15] mapped the one-to-many association relationship from RDB to graph-based NoSQL by creating two nodes, with the primary key of the one side inserted into the many side as a reference. For mapping the many-to-many relationship, the join table that is created in RDB is deleted, and the primary keys in the join table are inserted into each participating node.

In [16], the authors mapped the one-to-many relationship from RDB to graph-based NoSQL. The starting node in the graph is the many side and the primary key of the one side is inserted into the many side by keeping the primary key as an edge property. The join table in RDB is not used by keeping the information as a relationship property. While mapping the ternary relationship, the join table and the foreign keys of other tables were deleted, but the attributes of the relationship were kept as the relationship property between nodes on the graph.

In [17], the authors presented the transformation of RDB to several NoSQL families namely key-value, column, document, and graph. The authors identified the concepts of each database using defined tuples. Following that, algorithms to conduct the transformation are presented, and a case study is used as a proof of concept. This work is complete in the sense that it covers all NoSQL families. However, it is not clear that all relationship types in RDB are included.

Beyond the data structure and relationships, few works have recently been proposed focusing on the implementation of the transformation. In [18] the authors presented a framework that supports convenient migration from RDB to NoSQL DBMS. The framework consists of two modules, namely data migration and data mapping modules. Since the work is more focusing on the implementation, it does not present clear transformation existing inside the data mapping module. Instead, the paper presents experimentation result on various database operations of the mapping results.

In [19], the authors presented a data adapter used to query and map between SQL and NoSQL databases. The adapter enables query from application and deals with database transformation at the same time. While this work implements the data adapter, it does not provide clear transformation rules between two different databases.

As a summary, while there have been some works on schema transformation, the works focus only on the association relationship, and specifically to one particular NoSQL database only. To address the first limitation, in this paper we are going to cover transformation that includes various types of relationships that can be found in RDB, namely association, inheritance, and aggregation. In addition, to address the second limitation, in this paper we will address the transformation into various NoSQL databases.

3. Proposed Schema Transformation Rules

This Section presents the proposed rules for schema transformation from RDB to NoSQL databases. We are proposing a set of rules that follow the traditional transformation process in RDB, starting from association relationship with different constraints, and followed by inheritance relationship that is divided into specialization and union type.

3.1. Rule 1: One-to-One Association Relationship Transformation

Rule 1a (column-based NoSQL): Create a column family for each participating entity in RDB, with the primary key of one entity becoming an attribute of the other entity, and the former is the entity that has minimum cardinality of 0 if any (see Figure 1b).

Rule 1b (document-based NoSQL): There are two options based on query pattern, the first option if the information of both entities is frequently accessed together, and the second otherwise.

Option 1: Create a single collection for one participating entity in RDB and embed the other entity as a collection inside the first collection (see Figure 1c).

Option 2: Create a single collection for each participating entity in RDB, and the ID of one is inserted as a referencing ID in the other (see Figure 1c).

Rule 1c (graph-based NoSQL): Create a node for each entity in RDB, with entity that has minimum cardinality of 0 to be the start node. The ID of the start node is included as a node property in the end node (see Figure 1d). Note: if both entities have minimum cardinality 0 or both entities have minimum cardinalities of 1, the start node can be from either entity.

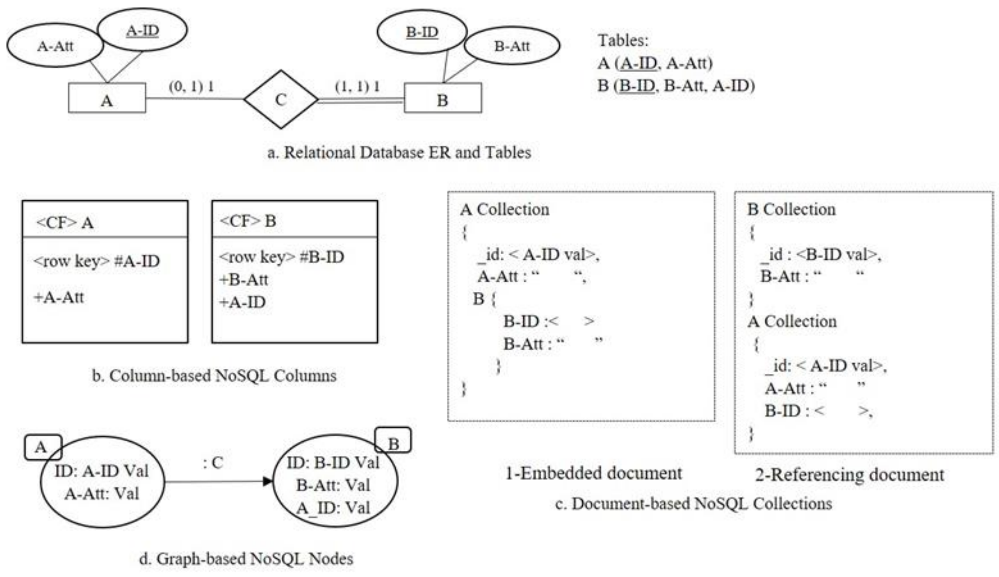

Figure 1. Transformation of one-to-one relationship from Relational database (RDB) to NoSQL Databases.

3.2. Rule 2: One-to-Many Association Relationship Transformation

Rule 2a (column-based NoSQL): Create a super column family for a participating entity in RDB that is on the one side, which includes the entity that is on the many side as a column family (see Figure 2b). The reason for adding the many side into the one side is that all data of the former can be retrieved in a single row key in the former. On the other hand, if the one side is inserted into the many side, then the information of the one side will be repeated in several columns of the many side.

Rule 2b (document-based NoSQL): There are two options based on query pattern, the first option if the information of both entities is frequently accessed together, and the second otherwise.

Option 1: Create a single collection for a participating entity in RDB that is on the one side and embed the other entity as a collection inside the first collection (see Figure 2c).

Option 2: Create a single collection for each participating entity in RDB, and the ID of the entity that is on the one side is inserted as a referencing ID in the other (see Figure 2c).

Rule 2c (graph-based NoSQL): Create a node for each entity in RDB, with entity that is on the one side becoming the start node, and the entity that is on the many side becoming the end node. The ID of the start node is included as a node property in the end node (see Figure 2d).

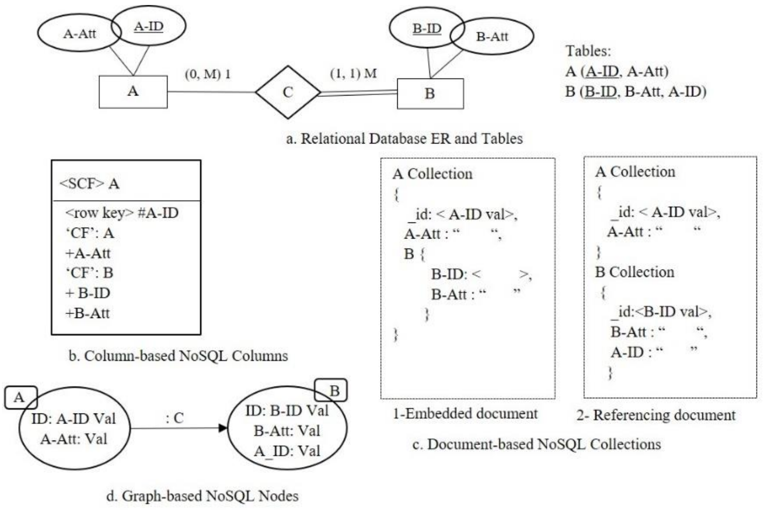

Figure 2. Transformation of one-to-many relationship from RDB to NoSQL Databases.

3.3. Rule 3: Many-to-Many Association Relationship

Rule 3a (column-based NoSQL): Create a column family for each participating entity in RDB and create a super column family for the relationship between entities. The row key of the super column family will be the combination of the relationship name and the primary key of one entity that is mostly used for the query selection (see Figure 3b). This rule is also applicable for the relationship that has more than two entities.

Rule 3b (document-based NoSQL): Create a single collection for each entity in RDB. In one collection, include the ID of another collection as a referencing ID. Additionally, include any relationship attribute inside the collection (see Figure 3c). This rule is also applicable for the relationship that has more than two entities.

Rule 3c (graph-based NoSQL): Create a node for each entity in RDB, with the ID of the start node included as a node property in the end node. The relationship and its attribute in RDB become the relationship property between the nodes (see Figure 3d). This rule is also applicable for the relationship that has more than two entities.

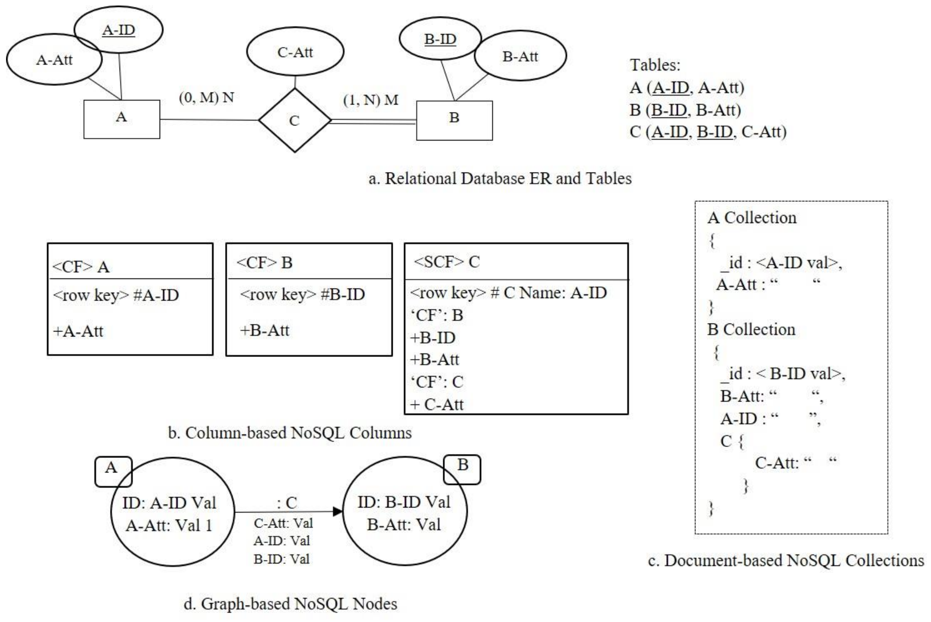

Figure 3. Transformation of many-to-many relationship from RDB to NoSQL Databases.

3.4. Rule 4: Specialization in Inheritance Relationship

Rule 4a (column-based NoSQL): Create a column family for each superclass and subclass in RDB, and the row key in the subclass column will be a combination of the subclass name and the key of the superclass column (see Figure 4b).

Rule 4b (document-based NoSQL): Create a single document for each participating entity in RDB, and by using referencing document structure, the ID of the superclass collection is included as ID in the subclasses collection (see Figure 4c).

Rule 4c (graph-based NoSQL): Create a node for each participating entity in RDB, with the super class becoming a start node and the subclasses becoming end nodes. The ID of the start node becomes the ID of the end node (see Figure 4d).

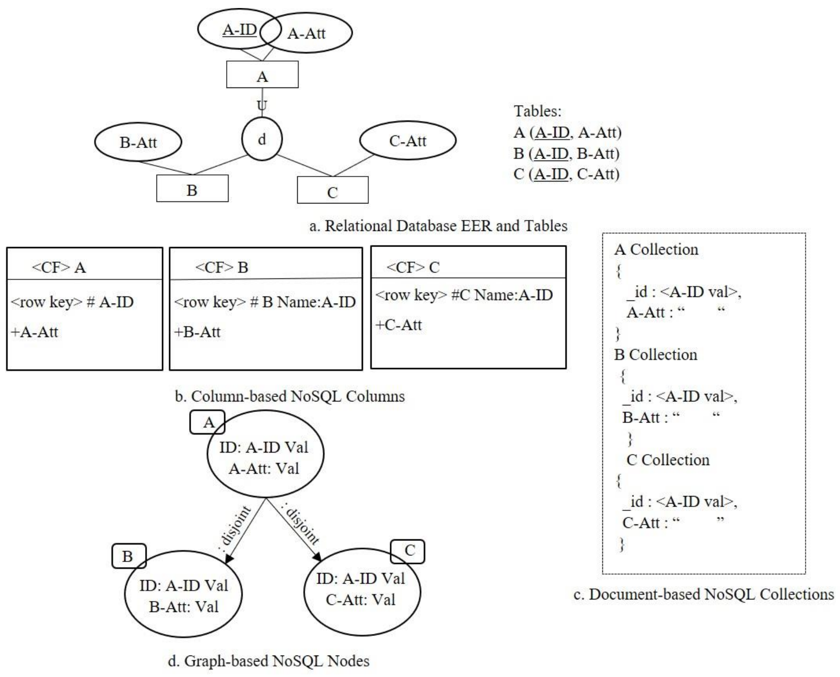

Figure 4. Transformation of specialization from RDB to NoSQL Databases.

3.5. Rule 5: Union Type in Inheritance Relationship

Rule 5a (column-based NoSQL): Create a column for the subclass and a column family for each superclass in RDB. Include the primary key of the subclass as a new attribute in the super class column family (see Figure 5b).

Rule 5b (document-based NoSQL): Create a single document for each superclass and subclass in RDB, and the primary key of the subclass is inserted in the superclass document (see Figure 5c).

Rule 5c (graph-based NoSQL): Create a node for each entity in RDB, with the subclass to be the start node and the super classes to be the end nodes, with the primary key of the subclass to be a new node property in the end nodes, and the relationship property as a union (see Figure 5d).

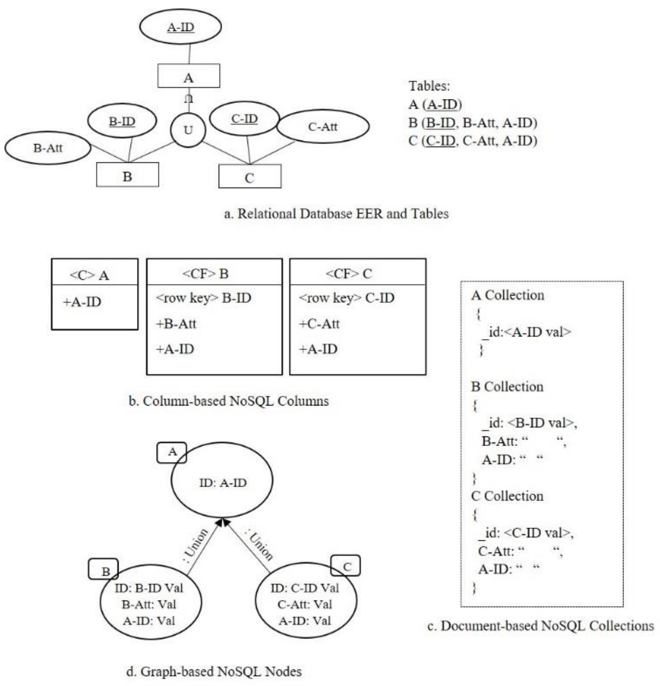

Figure 5. Transformation of union type from RDB to NoSQL Databases.

3.6. Rule 6: Aggregation Relationship

Rule 6a (column-based NoSQL): Create a super column family for a participating entity in RDB that is in the "whole" side, which includes the entity that is on the "part" side as a column family (see Figure 6b).

Rule 6b (document-based NoSQL): There are two options based on query pattern, the first option if the information from both entities are frequently accessed together, and the second otherwise.

Option 1: Create a single collection for each participating entity in RDB, and the ID of the entity that is on the "whole" side is inserted as a referencing ID in the other (see Figure 6c).

Option 2: Create a single collection for participating entity in RDB that is on the "whole" side, and embed the "part" side as a collection inside the first collection (see Figure 6c).

If aggregation relationship has one level, then both options in this rule are applicable. On the other hand, for multi-level aggregation relationships we use only the referencing document, to avoid nested embedded documents which may affect the performance of the query access.

Rule 6c (graph-based NoSQL): Create a node for each entity in RDB, with entity that is in the "whole" side becoming the start node, and the entity that is "part" side becoming the end node. The ID of the start node is included as a node property in the end node (see Figure 6d).

Table 1 lists the summary of the transformation schema between RDB to NoSQL Databases.

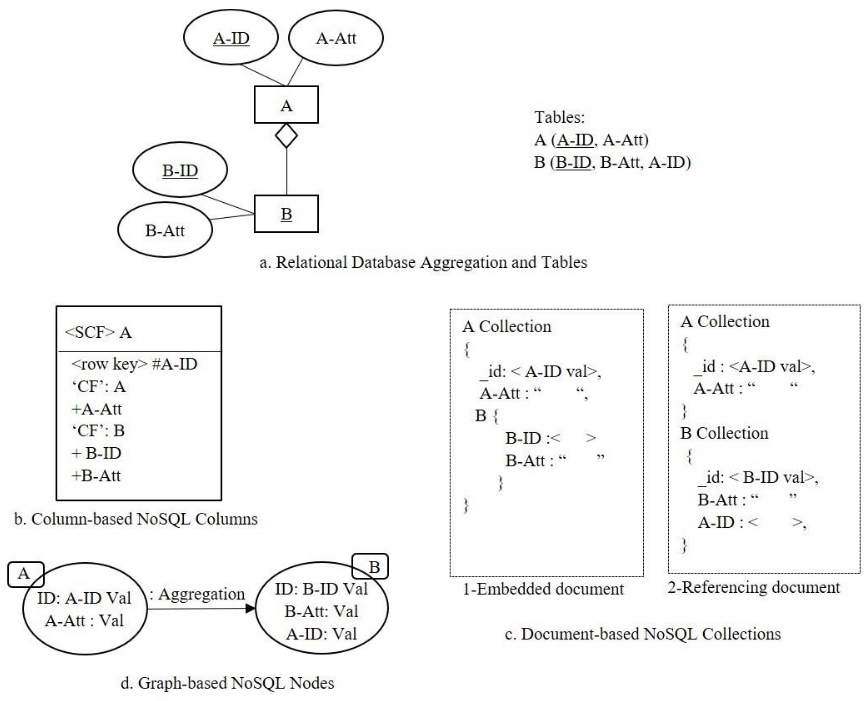

Figure 6. Transformation of aggregation from RDB to NoSQL Databases.

Table 1. Summary of proposing rules for mapping schema from RDB to NoSQL Databases.

	Column-Based			**Document-Based**		**Graph-Based**		
RDB Relationships	**Column**	**Column Family**	**Super Column Family**	**Embedding**	**Referencing**	**Node Label**	**Node Property**	**Relationship Property**
One-to-one		Y		Y	Y	Y	Y	
One-to-many			Y	Y	Y	Y	Y	
Many-to-many		Y	Y	Y	Y	Y	Y	Y
Specialization		Y			Y	Y	Y	
Union	Y	Y		Y		Y	Y	
Aggregation			Y	Y	Y	Y	Y	

4. Implementation and Evaluation

We applied the transformation rules on a case study of Service Match Company Database. The primary purpose is to provide a platform where clients can access services, needed on a regular basis, easily. These services include heavy goods transportation, domestic and commercial cleaning, landscape and lawn maintenance and design, plumbing works, electrical works, construction, and many more. Full details of the case study can be accessed on the following link (https://photos.app.goo.gl/qUDsUMWTxhtAsE4Q6). On the page, we also provide the full transformation result of the case study from original RDB to three NoSQL databases. The RDB dataset is comprised of 23 tables, each contains around 20 records. After following the proposed rules, the outcomes are 21 columns, 16 collections, and 17 nodes respectively for column-based, document-based, and graphs-based NoSQL.

We then implement the case study into four different databases: Oracle for RDB, Cassandra for column-based NoSQL, MongoDB for document-based database, and Neo4j for graph-based NoSQL. A small dataset is populated into the databases and we run six queries to check whether they return the correct results. In terms of experimentation environment, we use Oracle live SQL, Cassandra using Ubuntu, MongoDB using Cloudera, and Neo4j desktop version.

The number query outcomes for each database are shown the following Table 2.

Table 2. Number of queries outcome.

Query	Oracle	Cassandra	MongoDB	Neo4j
Q1 (one-to-one association)	5	5	5	5
Q2 (one-to-many)	3	3	3	3
Q3 (many-to-many)	7	6	4	7
Q4 (specialization)	3	3	3	3
Q5 (union)	7	7	7	7
Q6 (aggregation)	8	8	8	8

The numbers of outcome in column-based and document-based NoSQL are different only in the case of many-to-many relationship (Q3). For column-based NoSQL, for mapping many-to-many relationships, we used a super column family to map the relationship between two entities that contain the primary keys of both entities. Unlike the list and set types, the map type should be used only when creating a super column family, because the map type allows a user to store the metadata with value in the super column family. Otherwise, the data will be stored without the metadata. In addition, Cassandra does not support a negative operation such as NOT IN or not equal (!=).

For document-based NoSQL, the number of results is fewer than in RDB because we used an embedded document structure for mapping many-to-many relationships. It is obvious that one of the features of using embedded documents is that all the necessary data can be retrieved in one query. MongoDB will automatically create an object id for each document as identification for that document if the user does not specify the object id which is used as a reference to another document. This is also applicable to other document-based NoSQL databases products.

For graph-based NoSQL, the number of results is the same as the number of the result in RDB because the graph-based schema has similar features to the EER diagram in RDB. In addition, during data insertion Neo4j will discard the node properties that contain a null value. Data in Neo4j can be retrieved by one query because the relationship between two nodes is similar to join operation in RDB. Additionally, the graph model is like a grid, when each node is connected with another node, unlike MongoDB and Cassandra, which mostly need to run more than one query to restore data, and the reason is that the collections or columns are stored separately.

It is important to emphasize that the experiment is aimed to check the correctness of the transformation rules rather than focusing on the performance. Therefore, the size of the database is not the main issue, rather the database in the experiment contains different kinds of relationships that are covered by all proposed rules.

In addition to the evaluation through implementation, we also compare our proposal with existing rules as shown in Table 3. We can see that our proposed rules cover more relationship types that are usually found in RDB. While there are several existing works that have attempted to transform relational database constructs to NoSQL databases, they do not cover complete relationships, and therefore certain design complexity might not be easily transformed into NoSQL format.

Table 3. Comparative analysis with existing works.

RDB Relationships	Proposed	Column-Based			Document-Based		Graph-Based	
		[2]	[14]	[12]	[10]	[8]	[1]	[11]
One-to-one	Y	Y	Y	Y	-	Y	-	-
One-to-many	Y	Y	Y	Y	-	Y	Y	Y
Many-to-many	Y	Y	-	Y	Y	Y	Y	Y
Specialization	Y	-	-	-	-	-	-	-
Union	Y	-	-	-	-	-	-	-
Aggregation	Y	-	-	-	-	-	-	-

5. Conclusions and Future Work

In this paper, we proposed a set of rules to transform schema from RDB to three different categories in NoSQL Databases, namely column-based, document-based, and graph-based NoSQL. The rules cover different relationship types that can appear in data within RDB, namely association, inheritance, and aggregation. Along with the relationship types, the cardinalities are also considered. When discussion inheritance relationship, we further differentiate it into specialization and union type, the former involves single super-class and the latter involves multiple super-classes.

We applied these rules on a case study in RDB showing that full transformation can be applied in a real database design. From 23 tables in RDB, after applying the transformation rule, the outcomes are 21 columns, 16 documents, and 17 nodes for column-based, document-based, and graph-based NoSQL databases.

Transformation rules from one to another database format can be validated using formalization. This is something that is not yet done for this work. Our approach of validation is through experimentation by comparing query results of the original database and the query results of the target database format.

We use a small data set in different DBMS, namely Oracle Live for RDB, Cassandra for column-based NoSQL, MongoDB for document-based NoSQL, and Neo4j for graph-based NoSQL. We run queries to demonstrate the transformation still yields correct data. We also compared our proposed rule with the existing work, which shows how our work provides a more complete solution.

For the future work, work on automatic mapping can be conducted. Further investigation is needed on how our generic transformation rules can be applied to different DBMS within the same NoSQL database family. In addition, we aim to propose query mapping techniques from SQL to NoSQL databases, testing and comparing the performance of different data manipulation and retrieval operations.

NoSQL databases are commonly used to handle heterogeneous data. In this work, we are focusing on the mapping of RDB to various NoSQL database families. Mapping from different data formats, from structured formats such as object-oriented databases, and semi-structured databases such as XML database, and unstructured data such as texts or documents, is another possibility of future work. Following that, a generic framework that integrates these mapping rules can also be proposed.

Author Contributions: Validation, O.A. and E.P.; Writing—original draft, O.A. and E.P.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Rocha, L.; Vale, F.; Cirilo, E.; Barbosa, D.; Mourão, F. A Framework for Migrating Relational Datasets to NoSQL. *Procedia Comput. Sci.* **2015**, *51*, 2593–2602. [CrossRef]
2. Stanescu, L.; Brezovan, M.; Burdescu, D.D. Automatic mapping of MySQL databases to NoSQL MongoDB. In Proceedings of the 2016 IEEE Federated Conference on Computer Science and Information Systems (FedCSIS), Gdansk, Poland, 11–14 September 2016; pp. 837–840.

3. De Freitas, M.C.; Souza, D.Y.; Salgado, A.C. Conceptual Mappings to Convert Relational into NoSQL Databases. In Proceedings of the 21st International Conference on Enterprise Information Systems (ICEIS Vol.1), Roma, Italy, 25–28 April 2016; INSTICC: Lisboa, Portugal, 2003; pp. 174–181.
4. Storey, V.C.; Song, I.Y. Big data technologies and Management: What conceptual modeling can do. *Data Knowl. Eng.* **2017**, *108*, 50–67. [CrossRef]
5. Corbellini, A.; Mateos, C.; Zunino, A.; Godoy, D.; Schiaffino, S. Persisting big-data: The NoSQL landscape. *Inf. Syst.* **2017**, *63*, 1–23. [CrossRef]
6. Makris, A.; Tserpes, K.; Andronikou, V.; Anagnostopoulos, D. A classification of NoSQL data stores based on key design characteristics. *Procedia Comput. Sci.* **2016**, *97*, 94–103. [CrossRef]
7. Atzeni, P.; Bugiotti, F.; Rossi, L. Uniform access to NoSQL systems. *Inf. Syst.* **2014**, *43*, 117–133. [CrossRef]
8. Robinson, I.; Webber, J.; Eifrem, E. *Graph Databases: New Opportunities for Connected Data*; O'Reilly Media, Inc.: Sebastopol, CA, USA, 2015.
9. Pardede, E.; Rahayu, J.W.; Taniar, D. Mapping Methods and Query for Aggregation and Association Relationship in Object-Relational Database using Collection. In Proceedings of the 2004 IEEE International Conference on Information Technology: Coding and Computing (ITCC), Las Vegas, NV, USA, USA, 5–7 April 2004; pp. 539–543.
10. Pardede, E.; Rahayu, J.W.; Taniar, D. Object-Relational Complex Structures for XML Storage. *Inf. Softw. Technol. J.* **2006**, *48*, 370–384. [CrossRef]
11. Li, C. Transforming relational database into HBase: A case study. In Proceedings of the IEEE International Conference on Software Engineering and Service Sciences (ICSESS), Beijing, China, 16–18 July 2010; pp. 683–687.
12. Chen, J.K.; Lee, W.Z. Data conversion from RDB to HBase. In Proceedings of the IEEE 8th International Conference on Awareness Science and Technology (iCAST), Taichung, Taiwan, 8–10 November 2017; pp. 170–175.
13. Serrano, D.; Han, D.; Stroulia, E. From relations to multi-dimensional maps: Towards an SQL-to HBase transformation methodology. In Proceedings of the IEEE 8th International Conference on Cloud Computing (CLOUD), New York, NY, USA, 27 June–2 July 2015; pp. 81–89.
14. Liyanaarachchi, G.; Kasun, L.; Nimesha, M.; Lahiru, K.; Karunasena, A. MigDB-relational to NoSQL mapper. In Proceedings of the IEEE International Conference on Information and Automation for Sustainability (ICIAfS), Galle, Sri Lanka, 16–19 December 2016; pp. 1–6.
15. De Virgilio, R.; Maccioni, A.; Torlone, R. Converting relational to graph databases. In Proceedings of the First International Workshop on Graph Data Management Experiences and Systems, New York, NY, USA, 23 June 2013; ACM: New York, NY, USA, 25–28 April 2016; p. 1.
16. Wardani, D.W.; Kiing, J. Semantic mapping relational to graph model. In Proceedings of the IEEE International Conference on Computer, Control, Informatics and Its Applications (IC3INA), Bandung, Indonesia, 21–23 October 2014; pp. 160–165.
17. De Freitas, M.C.; Souza, D.Y.; Salgado, A.C. Conceptual Mapping to Convert Relational into NoSQL Databases. In Proceedings of the 18th International Conference on Enterprise Information Systems (ICEIS), Rome, Italy, 25–28 April 2016; SCITEPress: Setúbal Municipality, Portugal, 2004; pp. 174–181.
18. Rocha, L.; Vale, F.; Cirilo, E.; Barobosa, D.; Mourao, F. A Framework for Migrating Relational Datasets to NoSQL. In Proceedings of the 2015 International Conference on Computational Science (ICCS), Reykjavík, Iceland, 1–3 June 2015; Elsevier, B.V.: Amsterdam, The Netherlands, 1979; pp. 2593–2602.
19. Liao, Y.-T.; Zhou, J.; Lu, C.-H.; Chen, S.-C.; Hsu, C.-H.; Chen, W.; Jiang, M.-F.; Chung, Y.-C. Data adapter for querying and transformation between SQL and NoSQL database. *Future Gener. Comput. Syst.* **2016**, *65*, 111–121. [CrossRef]