Computer Systems Science & Engineering
DOI: 10.32604/csse.2022.021969
Article

Tech Science Press

An Efficient Schema Transformation Technique for Data Migration from Relational to Column-Oriented Databases

Norwini Zaidi[1], Iskandar Ishak[2,*], Fatimah Sidi[2] and Lilly Suriani Affendey[2]

[1]System Development and Engineering Center, Universiti Sains Islam Malaysia, 71800, Negeri Sembilan, Malaysia
[2]Faculty of Computer Science and Information Technology, Universiti Putra Malaysia, UPM Serdang, 43400, Selangor, Malaysia
*Corresponding Author: Iskandar Ishak. Email: iskandar_i@upm.edu.my
Received: 22 July 2021; Accepted: 23 September 2021

Abstract: Data transformation is the core process in migrating database from relational database to NoSQL database such as column-oriented database. However, there is no standard guideline for data transformation from relational database to NoSQL database. A number of schema transformation techniques have been proposed to improve data transformation process and resulted better query processing time when compared to the relational database query processing time. However, these approaches produced redundant tables in the resulted schema that in turn consume large unnecessary storage size and produce high query processing time due to the generated schema with redundant column families in the transformed column-oriented database. In this paper, an efficient data transformation technique from relational database to column-oriented database is proposed. The proposed schema transformation technique is based on the combination of denormalization approach, data access pattern and multiple-nested schema. In order to validate the proposed work, the proposed technique is implemented by transforming data from MySQL database to MongoDB database. A benchmark transformation technique is also performed in which the query processing time and the storage size are compared. Based on the experimental results, the proposed transformation technique showed significant improvement in terms query processing time and storage space usage due to the reduced number of column families in the column-oriented database.

Keywords: Data migration; data transformation; column-oriented database; relational database; big data

1 Introduction

Relational database has been the most popular database to be used by organizations around the world to manage data [1,2]. However, as data usage in information system becoming more demanding in terms of the data size, speed and variety, relational databases seems to be lacking in its ability in managing large volume of data and different types of database schema or flexible schema in which the phenomenon is known as the Big Data [2–4]. In order to overcome the relational database issues in handling the Big Data, NoSQL database is developed. NoSQL is a database approach that has unstructured format and allowing data

management operations to be faster than the relational approach [5]. In order to cope with the Big Data demand, organizations that are using relational database for their operations decided to migrate their DBMS into NoSQL database. Migration process needs to be done properly to make sure that the new NoSQL database captures the same structure and the operational tasks run smoothly.

The database migration process is defined as a process to transfer or move data from one database server; storage types or data formats to another. Commonly, a data migration involves the ETL process or Extract, Transform, and Load processes [6–9]. This process needs to be performed seamlessly without compromising the data quality content and the performance of their current operations [10–12].

As shown in Fig. 1, data transformation is among the core process in data migration. According to [13], data transformation is defined as a process to converting the data structure or data schema from source database following the new data structure in target database. However, there are no standard guidelines on how to transform the database schema from relational database and perform data migration to NoSQL database [8–13]. Database administrators' (DBA) experiences were used to manage the transformation process and the migration process to the NoSQL database since there are no definitive guidelines to perform data transformation between those two databases [13–15].

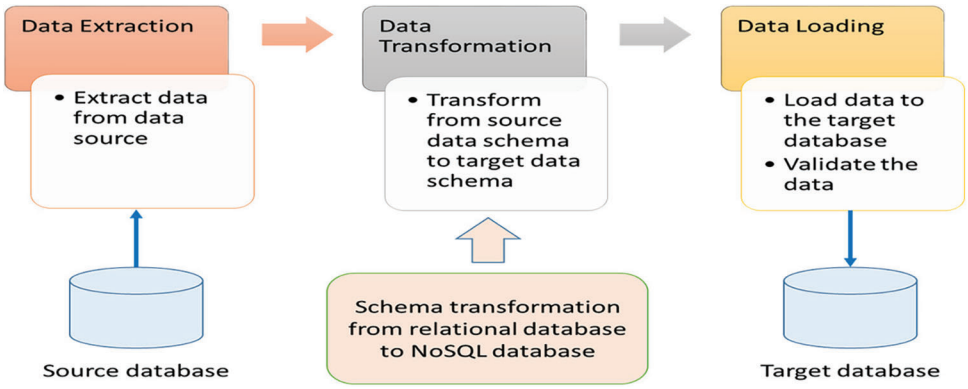

Figure 1: Data migration process

Initial challenge of performing data transformation between relational databases to NoSQL database is it requires the origin database to be analyzed properly in order to produce precise copy of NoSQL database schema when compared to the original schema. Most of the data transformation approaches had no problem of producing precise NoSQL database that is similar to its original schema as these approaches have validation component in which it is compared to the original schema. However, these transformation approaches produced unnecessary duplication that also created unnecessary storage for the destination database. Unnecessary duplication also increases query-processing time of the NoSQL databases [14–18].

In this paper, an enhanced data transformation approach that combines the use of multiple-nested schema and data access pattern is proposed for the purpose of minimizing data duplications and reduces query processing time. Rowkey design is also considered in the proposed method in order to enhance the transformation approach.

The rest of this paper is organized as follows: Section 2 highlights the related work on data transformation approaches. Section 3 discusses the key issues in data transformation approaches including data size and access time. Section 4 describes our proposed data transformation approach using the combination of query processing pattern, key design and nested, multiple nested. In this section, the architecture of the proposed model is also discussed. In Section 5, the experimental results of the proposed approach are included. Query processing time and storage size are used to evaluate and

compare the performance of the proposed model with the non-multiple nested approach. Finally in Section 6, contributions of this paper are briefly concluded in this section.

2 Related Works

Based on the literatures, denormalization is a common technique for schema or data transformation of relational database to NoSQL database [12]. Denormalization is defined as a technique to merge and duplicate related data to eliminate join relationship between tables into a single table [15,19,20]. Denormalization is used to merge and duplicate related data to eliminate join relationship between tables into a single table. There are also proposed works that involved the use of intelligent keys or rowkeys to support denormalization. Unique rowkeys or intelligent keys for identification are determined in the same NoSQL table [12,15,19–23]. Based on the differences in schema transformation techniques, there are no standard guidelines to conduct schema transformation and migration from relational database to NoSQL database [12–16].

An automatic SQL to NoSQL schema transformation technique from MySQL to HBase database is proposed in [21]. It uses denormalization technique and the NoSQL DDI (denormalization, duplication, intelligent keys) design principles to aggregate relational tables into NoSQL table. This technique also used tall-narrow design that allows a table with few columns but many rows to have better query retrieval. Its schema conversion technique parses the SQL table schema automatically and then converts the relationships among the tables into several linked lists. After parsing all the tables, the chained length of all linked list is identified. The row key with the highest cardinality is the combination of all the primary keys with the longest chained length.

Structured denormalization is another denormalization-based technique in schema transformation from relational database to NoSQL database [21]. This technique focuses on the work to autonomously denormalize and re-aggregate SQL tables into NoSQL table and also follow the NoSQL DDI (denormalization, duplication, intelligent keys) design principles. In this technique, schema analysis is performed to review all the primary keys of the SQL table. The rowkey in column oriented NoSQL table is selected from several primary keys and the relationships among the tables in SQL database. After the rowkey is defined, the migration process aggregates all the columns to the NoSQL table. Schema conversion technique in [20] improvised the work in [21] by adding a layer called persistence layer between the application and the NoSQL database for querying. They also built a component called the Mediator using Relational Database Management System (RBDMS) proxy that allows communication between the RDBMS server and the application.

There are also approaches that did not focused on using denormalization. One of the approaches is proposed in [24] by focusing on schema conversion from relational database to NoSQL database using source data model, target data model, and translation of the source data model into a target data model. In this technique, the schema of the relational database is the source data model and the column oriented NoSQL database is the target data model. A data migration algorithm is used for translating the data source model into the target model. The translation technique generates the target schema called the translation pattern and generates the data instances to the target database from the source database. The data conversion process consists of three stages, namely: data extraction to querying a relational database using SQL commands, data processing that involves transformation of data to the target database format, and injection process that injects the destination database. The proposed work in [24] also considers data extraction using existing querying SQL commands; however, without considering data access pattern, the transformation process produces data redundancy in the NoSQL database. A set of conversion rules has been proposed in [13] as a guideline for migration process from the relational to the NoSQL database. The proposed baseline conversion rules are rules on data proximity, column families, data quantity, and

access pattern. The proposed conversion rules help database practitioners to transform the relational database to the NoSQL database. However, there was no experiment conducted using the proposed conversion rule guideline and the performance of the transformed NoSQL database is not measured.

Another data migration approach that did not focus on the denormalization is proposed in [25]. In this approach, an SQL layer called SQLtoKeyNoSQL for transforming the structure of relational database to any key-oriented NoSQL database such as document store database, key value database, and column-oriented database. This schema conversion uses layer to map a relational database schema into a canonical model, which is an intermediate data model between relational database and key-oriented NoSQL database. The canonical model implements mapping strategies to map NoSQL command and SQL command using REST API access methods (Get, Put, and Delete) to the NoSQL target schema. This layer has flexible options for user to choose any targeted key oriented database. This layer also has the ability to manipulate the relational database in any key oriented NoSQL database for users to manage the data. The SQL to KeyNoSQL layer has an architecture that consists of seven modules for data migration process. The SQL to KeyNoSQL layer has an Access Interface that received the SQL instruction and sends it to the SQL Parser module. The SQL Parser module then performs semantic verifications and sends it to the Query Planner module for query execution optimizer. The Translate module generates the access method to the Execution Engine module. The Execution Engine module checks the login information, the target of NoSQL database, and the table store from the Dictionary module before filtering the data and generates the result to be sent to the Access Interface. The Communication module then executes requested access method to any key oriented NoSQL database and the data returned from the NoSQL database are sent back to the Execution Engine through Buffer. The data migration using the SQL layer focuses on the use of a special layer that helps understanding the relational database through its metadata before it can be mapped into NoSQL. This SQL layer also can execute the same SQL command to query the NoSQL database. However, this technique needs high understanding on relational database data model before a layer or framework can be developed to execute an existing SQL query for both the relational and the NoSQL databases.

One big issue related to the denormalization approaches (that also includes the use of rowkey design) is, it merges the related join relationship tables to a single table in column-oriented table to optimize read performance. It produces unnecessary duplicated data to the targeted column-oriented table [17]. This also affected query-processing time and its size as duplicated data increases processing time and data storage size. In order to improve denormalization-based approaches, data access pattern is used to be analyzed from query logs and indexing in every table of relational database [17]. Denormalization-based transformation technique that use access pattern in [26] used proposed four steps of data schema transformation process from relational database to HBase database. The steps are denormalization, extended table merging, key encoding, and view based on index. In this work, access pattern is reviewed from query logs to improve HBase schema. In the denormalization steps, the one-to-one relationship and one-to-many relationship are used to identify a pair or multiple tables that have one-to-one relationship or one-to-many relationship. If there is only one pair or multiple pairs among the tables, the tables will be merged in a separate column family based on the foreign key of the tables.

Another schema transformation technique to transform the NoSQL database using denormalization technique and data access pattern is proposed in [27]. In this schema transformation technique, a data model in the relational database is converted into the NoSQL database schema using three rules. The first rule considers the correlated data that have the same access pattern into a single column family. Then, a suitable foreign key of a table is chosen in the NoSQL database by considering the existing join relationships of one-to-one, one-to-many, and many-to-many from the origin relational database. The third and final rules merged the data tables to reduce the foreign keys based on access pattern of the application. In the second phase, the data source schema and target schema are mapped using the Tableau schema mappings.

Data transformation using multiple-nested tables merging is proposed in [18]. The multiple nested table merging merges more than three tables of the relational database into a single column-oriented table. It incorporates three cases to design column-oriented schema that includes single table, nested, and multiple-nested tables merging. The single table migrates a single table of the relational database to the column-oriented schema and selects the primary key of the table in the relational database as a rowkey in the column family of the column-oriented database. The column family contains all the data from the relational database. The nested table merging transforms two tables from the relational database into a single column-oriented table. The multiple nested table merging transforms and merges more than three tables from the relational database into a single column-oriented table. This schema integrates the data from the multiple tables of a relational database into a single column-oriented database table to improve query performance. However, this work did not consider data access pattern that can help to reduce data redundancy of HBase database.

Based on the literatures, it shows that many of the previous works focuses on denormalization and rowkey design as part of their schema transformation technique from the relational database to the NoSQL database. However, none of the previous works have considered schema transformation with read pattern, nested, and multiple nested, and rowkey design in a single solution. The combination of all mentioned criterion can improve query performance and also reduce storage size as it fits the nature of column-oriented database schema. Therefore, this study focuses on schema transformation using denormalization with read pattern, nested and multiple nested table merging and rowkey design a solution to reduce unnecessary production of redundant column families that increases storage size and to improve query performance of the resulted column-oriented database.

3 Proposed Approach

The proposed schema transformation technique has three main steps as listed as follows:

1) Step 1: Denormalization with read pattern
2) Step 2: Nested and multiple nested table merging
3) Step 3: Rowkey design

Fig. 2 below shows the steps for the proposed schema transformation technique of this study.

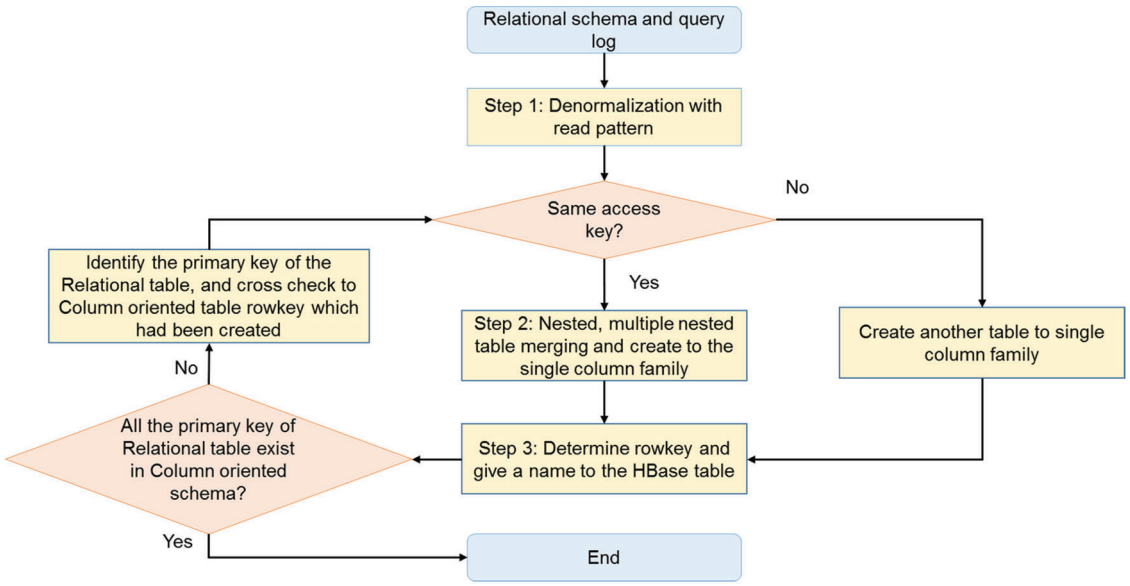

Figure 2: Proposed schema transformation technique

Step 1: Denormalization with Read Pattern

In the proposed schema transformation technique, the database schema and query logs of the relational database are analyzed to identify the origin's data access pattern. The data access pattern consists of the read pattern and the write pattern of the application system [26–31]. The read pattern is derived from the SELECT statement of the query logs while the write pattern is derived from the INSERT statement and the UPDATE statement of the query logs. In this study, in order to design the HBase database schema, the read pattern is used to determine the access key of the record from the conditional expression of a SELECT statement. This is based on the previous works in that uses data access pattern in [13,26,27] which also focuses only on the read pattern. In the proposed work, denormalization with read pattern is performed as the first step in schema transformation technique to avoid merging of unrelated relational tables that do not have the same read pattern.

The basic form of the SQL statement to identify a read pattern is shown as follows:

Given a SELECT statement for a query log Q [32]:

Q: SELECT <attribute list> FROM <table list> WHERE <condition>;

where:

i. <attribute list> is a list of attribute names whose values are retrieved by the query,
ii. <table list> is a list of relation names required to process the query,
iii. <condition> is a conditional expression that identifies the records to be retrieved by the query.

In SQL statement, the basic logical comparison operators for comparing attribute values with one another are OPR = {=, <, ≤, >, ≥, ≠}. Besides, the conditional expression can also be combined with the Boolean conditions AND, OR, and NOT. The Boolean conditions AND and OR are used to filter the records based on more than one condition. The Boolean condition AND displays a record if all the conditions separated by AND are TRUE. While the Boolean condition OR displays a record if any of the conditions separated by OR is TRUE. The Boolean condition NOT displays a record if the condition is NOT TRUE. There is also a LIMIT clause that is used in the conditional expression to specify the number of records to return [32].

In this study, the access key is identified through the <condition> clause where a <condition> clause can be in the following forms:

(1) limit ?;
(2) A_i OPR A_j;
(3) A_i OPR ?;
(4) <condition$_1$> <Boolean condition> <condition$_2$> … <condition$_n$>;

where

A_i and A_j are the attributes of a table, '?' denotes a constant value, <Boolean condition> = {AND, OR, NOT}, and < > can be in one of the above forms. Based on the forms shown above, the rules for denormalization with read pattern are as described below:

i) **Rule 1:** If the <condition> clause is in the form as specified in (1), then the access key of query Q is the primary key of the table specified in Q.
ii) **Rule 2:** If the <condition> clause is in the form as specified in (2), then the access key is either A_i or A_j.
iii) **Rule 3:** If the <condition> clause is in the form as specified in (3), then the access key is A_i.

iv) **Rule 4:** If there are n conditions as in the form specified in (4), then the access key for each condition, <condition_i>, is identified based on the form of the <condition_i> as stated in (1), (2) or (3) above and Rule 1, Rule 2, or Rule 3 are applied accordingly.

v) **Rule 5:** If the access key identified by Rule 2 or Rule 3 contains security information, then it is not recommended to be the access key to ensure information security [33]. Therefore, if the query Q has only a single access key, then the primary key of the table specified in Q is selected as the new access key of Q. Otherwise, if there are more than one access key, then the access key that contain security information is drop, while others are maintain.

Step 2: Nested and Multiple Nested Table Merging

HBase is a column-oriented database thus there are no join relationships between the HBase tables like in the relational database and it does not support multiple table queries [4,18]. After the transformation, all the related tables in the relational database are merged into a single column family table with the support of either nested or multiple nested merging [4]. The schema transformation approach proposed in [26,27] is limited to only nested table merging. The nested table merging has resulted into the creation of more column families in the HBase table that may contain redundant column families thus increases query processing time and unnecessary storage size. A multiple nested table merging in schema transformation is proposed in [4] with the aim to improve query performance and reduce data redundancy. Multiple nested table merging is important to cater merging of more than two tables in which nested table merging only cater for two tables.

Rule 1: Given a query Q_i where A_p is the access key of Q_i based on table P, and a query Q_j, where A_s is the access key of Q_j based on table S, where both access keys have been identified in Step 1. If the access key A_p = A_s, then both tables P and S are merged into a single column family, say PS. The nested table merging merged two tables of different queries if the access keys between these queries are having the same read pattern.

Rule 2: Given a query Q_i where A_j is the access key of Q_i based on tables P and S, where the access key has been identified in Step 1. Both tables P and S are merged into a single column family, to become PS. The nested table merging merged two tables of a single query that have the same read pattern.

Rule 3: Given a query Q with a list of tables, T = {T_1, T_2, …, T_n}, where the access key of T is T_x, identified in Step 1. The list of tables T are merged into a single column family say T'. The multiple nested table merging merged more than two tables of a single query into a single column family in the HBase table.

Step 3: Rowkey Design

In HBase, only one rowkey is allowed for unique identification. The data in the HBase database are sorted lexicographically by a rowkey and there is no secondary key in the HBase table [34–37]. In this study, the rowkey is selected from the access key that is identified in Step 1. There are three steps on how to produce a rowkey as explained in detail in Step 3.1, Step 3.2, and Step 3.3.

Step 3.1: In the first step of rowkey design, the access key of the table is checked whether it is the primary key of the relational table. If it is, the access key is selected as a rowkey in the HBase database schema. Otherwise, if the access key is not the primary key of the relational table, then a rowkey is determined by combining the access key and the primary key of the relational table to form a unique key. Given a table R in a relational database schema with a list of attributes = {R_1, R_2, …, R_n} where R_i is the primary key. Assume a table H derived in Step 2 with attribute list = {H_1, H_2, …, H_m} and an access key H_j. If the access key H_j is also the primary key of the relational table, where H_j = R_i, then H_j is selected as the rowkey of table H in the HBase database schema. Otherwise, H_j is combined with R_i to form a unique key H_j R_i, the rowkey of H table in the HBase database schema.

Step 3.2: After creating all the rowkeys for each column family in the HBase database in Step 3.1, the primary keys and the tables of the relational database schema are reviewed. If there is a primary key of a table in the relational database schema that does not exist as a rowkey in the HBase database schema, then a new table based on the relational database schema is created in the HBase database schema. The primary key of the table in that relational database is the rowkey in the new HBase table. This is to ensure that all tables of the relational database are migrated accordingly to the HBase database to avoid data loss. Given a relational database schema, if the primary key R_i of the R table does not exist in the HBase database schema derived in Step 2, a new table R' in the HBase database schema is created for the R table. The primary key R_i is selected as the rowkey of the new HBase table R'.

Step 3.3: In this step, a short and meaningful name is given to each table name, column family name, and column qualifier name in the HBase database. This is based on the best practices according to [26,36,38]. This is because HBase stores the column qualifier with the values and HBase does not limit the number of column qualifiers. In order to reduce storage size, short table name, column name, and column qualifier name are suggested in [26]. The name given should not be descriptive of table like in a typical relational table name [36].

4 Evaluations

In order to validate our proposed work, experiments that involved data migration process have been performed. The other related works had incomplete explanations and lack of information in their schema transformation procedure thus the experiment of the work could not be emulated. Based on the literatures reviewed; only [26] showed the detail explanation on schema transformation, experiments, and results. Therefore, the work [26] is implemented and is used as the benchmark technique.

In terms of the database platform, HBase is used as the column-oriented database and MySQL is used for the relational database. A benchmark dataset called DELL DVD dataset [39] is used in the experiment. Apache Sqoop tool is used to migrate data from the relational database (MySQL) to the column-oriented database (HBase). The computer system used in the experiment has one core processor and 4GB RAM running on Ubuntu Linux operating system.

The performance measurement used in each analysis is shown as follows:

i) Database sizes: Database size is measure to show the size of the resulted HBase database after the benchmark [26] and the proposed transformation techniques are performed. The database size is measured in gigabyte (GB).

ii) Response time: Response time is measured to measure the efficiency of query performance of the resulted HBase databases after both the benchmark and the proposed transformation techniques are performed. Time is measured in second (s). The time measurement is calculated as total time used for querying the HBase database using query for DELL DVD dataset. Each measurement represents the average time of five runs using different value of the same listing of query on HBase database of [26] as well the proposed technique.

The percentage of storage size and query performance time are measured by:

$$\text{Data size difference } (\%) = \frac{\left|\begin{array}{c}\text{Total differences of storage size resulted} \\ \text{from the benchmark work and the proposed work}\end{array}\right|}{\text{Total storage size of benchmark work}} \chi\ 100 \quad (1)$$

$$\text{Query performance time difference } (\%) = \frac{|\text{Total differences of query processing time resulted from the benchmark work and the proposed work}|}{\text{Total time taken of benchmark work}} x\ 100 \quad (2)$$

Fig. 3 describes the overview of the proposed schema transformation technique to migrate database from MySQL to HBase database. Once the data is successfully generated or download, the data are loaded into MySQL database for transformation and migration process. After the schema transformation process is performed, the data are migrated from MySQL database to HBase database using the Apache Sqoop. The MySQL Java connector is used to connect between MySQL database and HBase database. The Cdata Excel Add-In for HBase is used to export the HBase data to an excel file. The distance requirement is calculated using the Harversine formula in the excel file and then the result is imported to the HBase table. Haversine formula is the formula for calculating distance between two (2) locations [40–42] and it is used in Query 9 in the experiment.

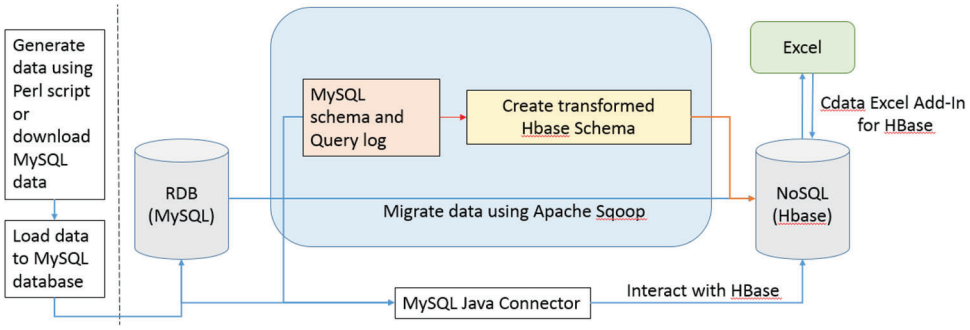

Figure 3: Overview of the migration process from MySQL to HBase

Based on the implementation that has been performed, the benchmark transformation technique produces eight tables with 23 column families as shown in Fig. 4. While, the proposed transformation method produces 10 tables with ten tables and 15 column families as shown in Fig. 5. In order to validate the reduction of data redundancy in HBase database, the database size of five tables of DELL DVD dataset and database size of three tables of Employees dataset are calculated. This is done to measure the size of the data that is used to create HBase database. In order to measure the storage size efficiency, average decrease percentage of the resulted HBase schema are calculated from the proposed work and the work done by [26] as the benchmark result.

Tab. 1 shows the database size of the table used in the experiment using the DELL DVD dataset. It shows reduction in data size between 8% and 75% in all queries performed on the resulted HBase database from the proposed technique when compared to the benchmark result. On average, it records 13.83% decrease of data size used. This shows that the proposed work has successfully reduced the data size usage in data transformation from relational database to NoSQL database.

The results of the database query processing time are shown in Tab. 2. It shows reductions between 12% and 40% in terms of processing time for every query performed. Based on the table, an average of 29.28% decrease of query processing time is recoded when it is compared to the benchmark result's work. This shows that the proposed work has successfully produced a column-oriented database that is efficient in query processing time when compared to the benchmark result.

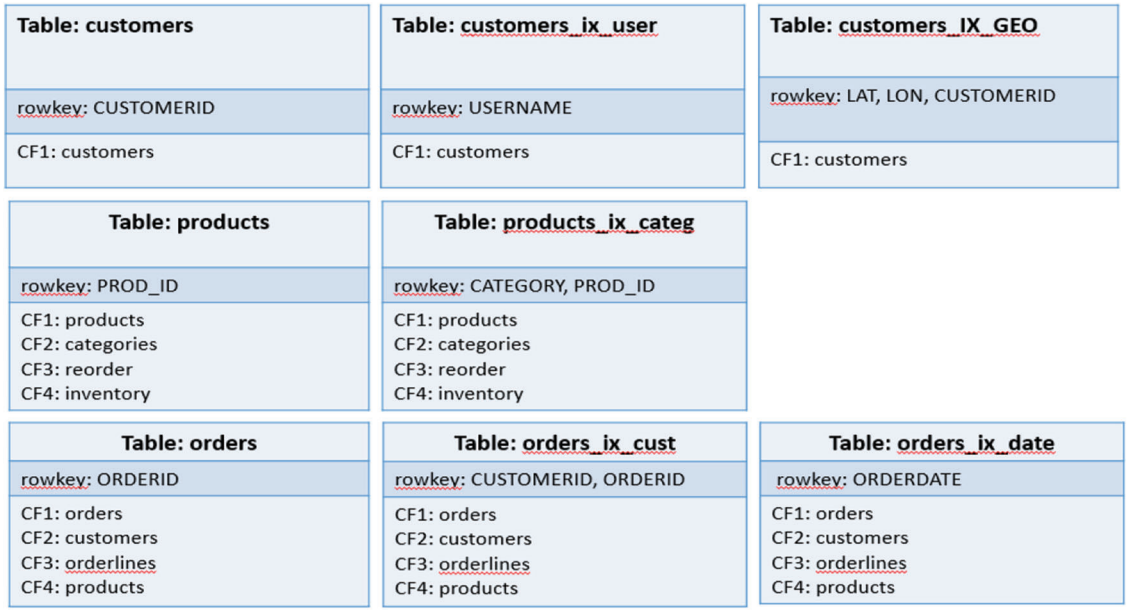

Figure 4: Resulted HBase schema after transformation using technique in [26]

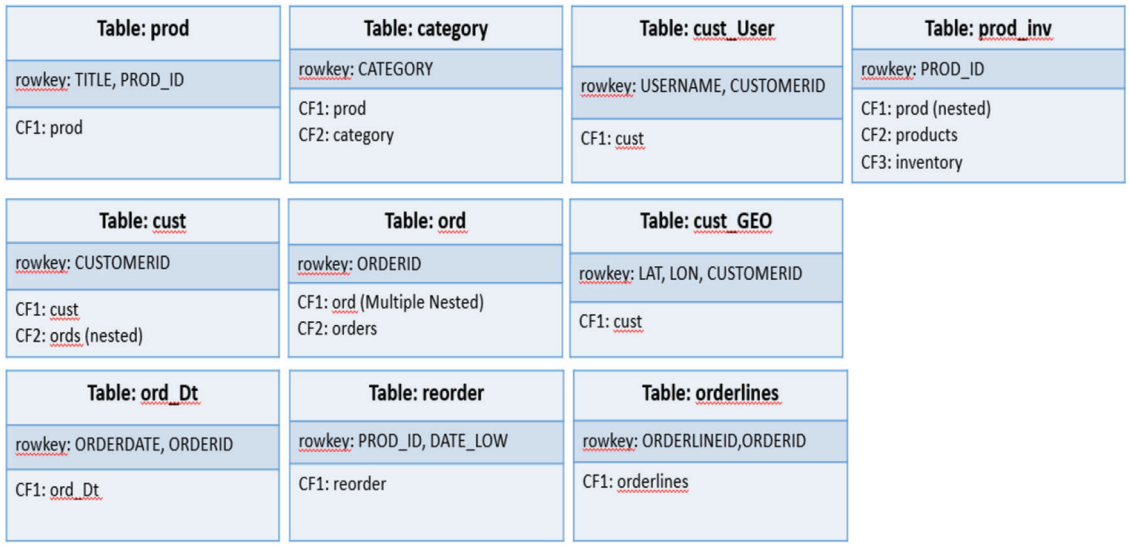

Figure 5: Resulted HBase schema after transformation using the proposed technique

Figs. 4 and 5 describe the column families generated by both transformation technique of [26] and the proposed work respectively. Fig. 4 shows the benchmark technique produced the migrated column oriented database schema with 23 column families. Fig. 5 shows the resulted column-oriented database has been produced by the proposed transformation technique with only 15 column families. This shows that the proposed transformation technique managed to reduce the column families of the transformed schema thus affected the query processing time and data size usage.

Table 1: Data size usage of the proposed and benchmark techniques

Query	Tables involved[1]	Data size used (Gb)[1]	Data size usage (Gb)[2]	Tables involved[2]	Difference in data size (Gb)	Difference in data size (%)
QH1, QH3, QH6	*customer*	19.000	15.000	*cust*	4.000	21.05
QH2	*products*	0.056	0.014	*prod_inv*	0.042	75.00
QH4, QH5, QH7	*orders*	7.700	6.500	*ord*	1.200	15.58
QH8	*orders_ix_date*	4.500	4.100	*ord_Dt*	0.400	8.89
QH9	*customers_IX_GEO*	24.00	22.000	*cust_GEO*	2.000	8.33

[1]Results from experiment performed based on [26]. [2]Results from experiment performed based on the proposed work.

Table 2: Query processing time of the proposed and benchmark techniques

Query	Query processing time[1] (s)	Query processing time[2] (s)	Difference in query processing time (s)	Difference in query processing time (%)
QH1	0.0398	0.0350	0.0048	12.06
QH2	0.0330	0.0228	0.0102	30.91
QH3	332.2102	199.0166	133.1936	40.09
QH4	0.0240	0.0194	0.0046	19.17
QH5	0.0400	0.0272	0.0128	32.00
QH6	0.1880	0.1420	0.0460	24.47
QH7	0.2194	0.1572	0.0622	28.35
QH8	1.7014	1.0416	0.6598	38.78
QH9	42.5052	26.4816	16.0236	37.70

[1]Results from experiment performed based on [26]. [2]Results from experiment performed based on the proposed work

5 Discussion

Based on the comparison results of data size, it shows that the resulted HBase tables to have smaller data size than the tables produced by the benchmark technique. Also, the resulted HBase tables produce reduction in query processing time when compared to the query processing time of the benchmark technique. This shows that the proposed schema transformation techniques had successfully achieve the objective of this study in terms of reducing data size and processing time.

The proposed schema transformation technique has successfully produced a single column family in a transformed HBase schema. This is because of the proposed technique ability to execute denormalization using read pattern, nested and multiple nested table merging and rowkey design. The identified read pattern helps to identify access key of the table and the nested and multiple nested table merging merge the related joined table into a single column family. Rowkey design used in all transformed tables is based on step 3. In addition, the short table name, column family name, and column qualifier name are given to all transformed HBase tables in which it reduces storage size as long names are always given by the previous approaches that in turn increase the data size unnecessarily. This is because the HBase

database stored the column qualifier with the values and HBase does not limit the number of column qualifier that will increase the storage size. The number of column families define in each table of the transformed HBase tables are set to maximum of three column families to avoid data retrieval performance degraded due to the HBase schema design.

6 Conclusion

In this study, the proposed schema transformation technique that utilizes denormalization with read pattern, nested and multiple nested schemas, and rowkey design has shown to be effective for schema transformation to NoSQL database. The read pattern helps to identify the access key from query logs in the application system. The same read patterns are merged together into a single column family in HBase table. This reduces data redundancy in the HBase database and reduces storage size. The nested and multiple nested schemas helps to merge related tables from relational database into a single HBase table. This is because HBase has no join relationship and it is difficult to query and retrieve data from multiple tables after data migration to the HBase database. The nested and multiple nested schema design on HBase leads to query only on a single HBase table to retrieve the data and thus provides efficient accessing time. Lastly, the rowkey design helps to produce only one rowkey for single HBase table.

As a conclusion, the proposed schema transformation technique that utilizes denormalization with read pattern, nested, multiple nested table merging, and rowkey design has proved that the technique successfully reduced the data redundancy thus reduces the storage size and also efficiently improve query processing time. Based on the comparison results of data size, it shows that the resulted HBase tables to have smaller data size than the tables produced by the benchmark technique. Also, the resulted HBase tables produce reduction in query processing time when compared to the query processing time of the benchmark technique. This shows that the proposed schema transformation techniques had successfully achieve the objective of this study in terms of reducing data size and processing time.

Acknowledgement: We thank Universiti Putra Malaysia Grant Scheme (Putra Grant) from Universiti Putra Malaysia (GP/2020/9692500) for their support.

Funding Statement: This work is supported by Universiti Putra Malaysia Grant Scheme (Putra Grant) (GP/2020/9692500).

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] J. R. Lourenco, V. Abramova, M. Vieira and B. Cabral, "NOSQL database: A software engineering perspective," *Advances in Intelligent System and Computing*, vol. 353, no. III–IV, pp. 741–750, 2015.

[2] L. Rocha, F. Vale, E. Cirilo, D. Barbosa and F. Mourão, "A framework for migrating relational datasets to NoSQL," *Procedia Computer Science*, vol. 51, pp. 2593–2602, 2015.

[3] N. Ntarmos, I. Patlakas and P. Triantafillou, "Rank join queries in NoSQL databases," in *Proc. of the 2014 VLDB Endowment*, Hangzhou, China, pp. 493–504, 2014.

[4] G. Zhao, Q. Lin, L. Li and Z. Li, "Schema conversion model of SQL database to NoSQL," in *Proc. of the 9th Int. Conf. on P2P, Parallel, Grid, Cloud and Internet Computing, 2014*, Guangdong, China, pp. 355–362, 2014.

[5] S. Ghotiya, J. Mandal and S. Kandasamy, "Migration from relational to NoSQL database," in *IOP Conf. Series: Materials Science and Engineering*, Vellore, India, vol. 263, no. 4, pp. 1–7, 2017.

[6] P. Badlani, "NoSQL in action-a new pathway to database," *International Journal of Science and Research*, vol. 5, no. 6, pp. 872–877, 2016.

[7] T. Odia, S. Misra and A. Adewumi, "Evaluation of hadoop/mapreduce framework migration tools," in *Proc. of the Asia-Pacific World Congress on Computer Science and Engineering 2014*, Nadi, Fiji, pp. 1–8, 2014.

[8] A. B. M. Moniruzzaman and S. A. Hossain, "Nosql database: New era of databases for big data analytics-classification, characteristics and comparison," arXiv Preprint, arXiv:1307.0191, vol. 6, no. 4, pp. 1–14, 2013.

[9] S. H. A. El-Sappagh, A. M. A. Hendawi and A. H. El Bastawissy, "A proposed model for data warehouse ETL processes," *Journal of King Saud University, Computer. & Information Science*, vol. 23, no. 2, pp. 91–104, 2011.

[10] M. A. Ashok, "A review to the approach for transformation of data from MySQL to NoSQL," *International Journal of Engineering, Applied and Management Sciences Paradigms*, vol. 46, no. 1, pp. 9–14, 2017.

[11] J. Ahmed and R. Gulmeher, "NoSQL databases: New trend of databases, emerging reasons, classification and security issues," *International Journal of Engineering Sciences & Research Technology*, vol. 9655, no. 6, pp. 176–184, 2015.

[12] A. Goyal, A. Swaminathan, R. Pande and V. Attar, "Cross platform (RDBMS to NoSQL) database validation tool using bloom filter," in *Proc. of 2016 Int. Conf. on Recent Trends in Information Technology*, Chennai, India, pp. 1–5, 2016.

[13] R. Ouanouki, A. April, A. Abran, A. Gomez and J. M. Desharnais, "Toward building RDB to HBase conversion rules," *Journal of Big Data*, vol. 4, no. 10, pp. 1–21, 2017.

[14] T. Jia, X. Zhao, Z. Wang, D. Gong and G. Ding, "Model transformation and data migration from relational database to MongoDB," in *Proc. of the 2016 IEEE Int. Congress on Big Data, BigData Congress 2016*, San Francisco, CA, USA, pp. 60–67, 2016.

[15] A. Gomez, A. Ravanello, R. Ouanouki, A. April and A. Abran, "Experimental validation as support in the migration from SQL databases to NoSQL databases," in *Proc. of the Cloud Computing 2015: The Sixth Int. Conf. on Cloud Computing, GRIDs and Virtualization 2015*, Nice, France, pp. 147–153, 2015.

[16] C. H. Lee and Y. L. Zheng, "Automatic SQL-to-NoSQL schema transformation over the MySQL and HBase databases," in *Proc. of the 2015 IEEE Int. Conf. on Consumer Electronics - Taiwan, ICCE-TW 2015*, Taipei, Taiwan, pp. 426–427, 2015.

[17] A. Gomez, R. Ouanouki, A. April and A. Abran, "Building an experiment baseline in migration process from SQL databases to column oriented NoSQL databases," *Journal of Information Technology & Software Engineering*, vol. 4, no. 2, pp. 1–7, 2014.

[18] G. Zhao, L. Li, Z. Li and Q. Lin, "Multiple nested schema of HBase for migration from SQL," in *Proc. IEEE Int. Conf. on P2P, Parallel, Grid, Cloud and Internet Computing*, Guangzhou, China, pp. 338–343, 2014.

[19] J. Yoo, K. H. Lee and Y. H. Jeon, "Migration from RDBMS to NoSQL using column-level denormalization and atomic aggregates," *Journal of Information Science and Engineering*, vol. 34, no. 259, pp. 243–259, 2018.

[20] N. Kuderu and V. Kumari, "Relational database to NoSQL conversion by schema migration and mapping," *International Journal of Computer Engineering in Research Trends*, vol. 3, no. 9, pp. 506–513, 2016.

[21] C. H. Lee and Y. L. Zheng, "SQL-To-NoSQL schema denormalization and migration: A study on content management systems," in *Proc. of the 2015 IEEE Int. Conf. on Systems, Man, and Cybernetics, 2015*, Hong Kong, pp. 2022–2026, 2015.

[22] L. Ho, M. J. Hsieh, J. J. Wu and P. Liu, "Data partition optimization for column-family NoSQL databases," in *Proc. of the Int. Conf. on Smart City/SocialCom/SustainCom Together with DataCom*, Chengdu, China, pp. 668–675, 2015.

[23] A. Kanade, A. Gopal and S. Kanade, "A study of normalization and embedding in MongoDB," in *Proc. of the 2014 IEEE Int. Advance Computing Conf., IACC 2014*, Gurgaon, India, pp. 416–421, 2014.

[24] S. A. T. Mpinda, L. G. Maschietto and P. A. Bungama, "From relational database to column-oriented NoSQL database : Migration process," *International Journal Engineering Research and Technology*, vol. 4, no. 5, pp. 399–403, 2015.

[25] G. A. Schreiner, D. Duarte and R. dos, S. Mello, "SQLtoKeyNoSQL: A layer for relational to key-based NoSQL database mapping," in *Proc. of the 17th Int. Conf. on Information Integration and Web-Based Applications & Services*, Brussels, Belgium, pp. 1–9, 2015.

[26] D. Serrano, D. Han and E. Stroulia, "From relations to multi-dimensional maps: Towards an SQL-to-HBase transformation methodology," in *Proc. 2015 IEEE 8th Int. Conf. on Cloud Computing*, New York, NY, USA, pp. 81–89, 2015.

[27] C. Li, "Transforming relational database into HBase: A case study," in *Proc. of the IEEE Int. Conf. on Software Engineering and Service Sciences, 2010*, Beijing, China, pp. 683–687, 2010.

[28] H. Khazaei, M. Fokaefs, S. Zareian, N. Beigi-Mohammadi, B. Ramprasad *et al.*, "How do I choose the right NoSQL solution? A comprehensive theoretical and experimental survey," *Big Data and Information Analytics*, vol. 1, no. 2, pp. 185–216, 2016.

[29] F. Zhu, J. Liu, S. Wang, J. Xu, L. Xu *et al.*, "Hug the elephant: Migrating a legacy data analytics application to hadoop ecosystem," in *Proc. of the 2016 IEEE Int. Conf. on Software Maintenance and Evolution*, Raleigh, North Carolina, USA, pp. 177–187, 2016.

[30] MongoDB, (2020, Feb). *RDBMS to MongoDB Migration Guide*. [Online]. Available: https://media16.connectedsocialmedia.com/MongoDB/14856/RDBMS_MongoDB_Migration_Guide.pdf/ (accessed on 29 December 2019).

[31] J. Klein, I. Gorton, N. Ernst, P. Donohoe, K. Pham *et al.*, "Application-specific evaluation of NoSQL databases," in *Proc. IEEE Int. Congress on Big Data*, Los Angeles, CA, USA, pp. 526–534, 2015.

[32] R. Elmasri and S. B. Navathe, "Basics of functional dependencies and normalization for relational databases," in *Fundamentals of Database Systems*, 6th ed.; Pearson: Texas, United States of America, pp. 491–532, 2011.

[33] J. Jose, T. T. Tomy, V. Karunakaran, V. A. Krishna, A. Varkey *et al.*, "Securing passwords from dictionary attack with character-tree," in *Proc. of the 2016 IEEE Int. Conf. on Wireless Communications, Signal Processing and Networking, 2016*, Chennai, India, pp. 2301–2307, 2016.

[34] O. Hajoui, R. Dehbi, M. Talea and Z. I. Batouta, "An advanced comparative study of the most promising NoSQL and NewSQL databases with a multi-criteria analysis method," *Journal of Theoretical & Applied Information Technology*, vol. 81, no. 3, pp. 579–588, 2015.

[35] V. Manoj, "Comparative study of NoSQL document, column store databases and evaluation of cassandra," *International Journal of Database Management System*, vol. 6, no. 4, pp. 29–39, 2016.

[36] A. H. Team, *Apache HBase™ Reference Guide*, 2019. [Online]. Available: http://hbase.apache.org/book/book.html (accessed on 12th November 2019).

[37] L. George, in *HBase: The Definitive Guide*, O'Reilly Media: Sebastopol, California, 2011.

[38] D. DeRoos, P. C. Zikopoulos, R. B. Melnyk, B. Brown and R. Coss, in *Hadoop for Dummies*, John Wiley & Sons: Hoboken, New Jersey, 2014.

[39] T. Muirhead (2019). *DVD store test application*, [Online]. Available: https://github.com/dvdstore/ds3/tree/master/ds3.

[40] E. Winarno, W. Hadikurniawati and R. N. Rosso, "Location based service for presence system using haversine method," in *Proc. of the 2017 Int. Conf. on Innovative and Creative Information Technology: Computational Intelligence and IoT*, Salatiga, Indonesia, pp. 1–4, 2017.

[41] V. Jovanovic, V. Lazovic and N. Minic, "SQL query execution time between two singidunum university locations," in *Proc. of the Int. Scientific Conf. of IT and Business Related Research*, Belgrade, Serbia, pp. 624–628, 2015.

[42] N. Chopde and M. K. Nichat, "Landmark based shortest path detection by using A* and haversine formula," *International Journal of Innovative Research in Computer and Communication Engineering*, vol. 1, pp. 298–302, 2013.