Relational into Non-Relational Database Migration with Multiple-Nested Schema Methods on Academic Data

Teguh Bharata Adji[1], Dwi Retno Puspita Sari[2], Noor Akhmad Setiawan[3]

***Abstract*—The rapid development of internet technology has increased the need of data storage and processing technology application. One application is to manage academic data records at educational institutions. Along with massive growth of information, decrement in the traditional database performance is inevitable. Hence, there are many companies choose to migrate to NoSQL, a technology that is able to overcome the traditional database shortcomings. However, the existing SQL to NoSQL migration tools have not been able to represent SQL data relations in NoSQL without limiting query performance. In this paper, a relational database transformation system transforming MySQL into non-relational database MongoDB was developed, using the Multiple Nested Schema method for academic databases. The development began with a transformation scheme design. The transformation scheme was then implemented in the migration process, using PDI/Kettle. The testing was carried out on three aspects, namely query response time, data integrity, and storage requirements. The test results showed that the developed system successfully represented the relationship of SQL data in NoSQL, provided complex query performance 13.32 times faster in the migration database, basic query performance involving SQL transaction tables 28.6 times faster on migration results, and basic performance Queries without involving SQL transaction tables were 3.91 times faster in the migration source. This shows that the theory of the Multiple Nested Schema method, aiming to overcome the poor performance of queries involving many JOIN operations, is proved. In addition, the system is also proven to be able to maintain data integrity in all tested queries. The space performance test results indicated that the migrated database transformed using the Multiple Nested Schema method showed a storage requirement of 10.53 times larger than the migration source database. This is due to the large amount of data redundancy resulting from the transformation process. However, at present, storage performance is not a top priority in data processing technology, so large storage requirements are a consequence of obtaining efficient query performance, which is still considered as the first priority in data processing technology.**

***Keywords*—Multiple Nested Schema, Data Transformation, Data Migration, NoSQL, Big Data.**

I. INTRODUCTION

The need for data storage and processing technology is increasing along with the development of internet technology that cannot be separated from everyday life. One of the requirements for implementing database technology is to support educational institutions in managing academic data records.

Academic data record management activities cannot be separated from the need for information system support as a forum to collect, process, store, analyze, and disseminate information [1]. The information existing in the system is often used as a reference in the decision-making process, which is often known as 'data-driven decision' [2], so it is a necessity to provide a reliable system to support educational institutions in managing academic data.

Statistics show that the data volume will increase by 40% per year, and will grow by 44 times over the period between 2009 and 2020 [3]. The massive information growth produced along with the ongoing academic activities poses challenges in the form of system reliability. Traditional database technology systems that have been used for more than three decades, face the data heterogeneity challenges, which amounts is beyond the ability of traditional databases management to record, store, manage and analyze the data, which is known as Big Data [4].

Therefore, the emergence of increasingly developing NoSQL technology, with its superior flexibility and scalability, is considered the best alternative to overcome the shortcomings found in traditional databases. In recent times, NoSQL technology has been used in various large companies, such as Facebook, Google, Twitter, and Yahoo [5], so that more and more companies are choosing to migrate to the NoSQL database.

Some of advantages in migrating data to the NoSQL database are existence of various kinds of data models that can be adjusted to the needs, scalability that is easier, faster, more efficient, more flexible, and the presence of support for hardware failure in certain NoSQL databases [6].

However, migrating data between databases with different data models is not easy. One of the complexity factors is a need to represent relations or association between tables which are characters of relational databases, into a destination database which is a non-relational database.

Some researchers have investigated the performance of the NoSQL database. According to [7], NoSQL is not a 'one size fits all' database that meets all needs in the database. Each type of NoSQL database has its own characteristics. Cassandra is more appropriate to use in applications requiring faster write operations and high availability, HBase is suitable to be used in applications requiring high performance in load and bulk read operations, while MongoDB offers advantages in document search and data aggregation functions [7].

In MongoDB, relationships representation between entities or documents is divided into two, namely reference documents and embedded documents. The reference document model maps data into several separated documents, and represents

[1,3] *Lecturer, Department of Electrical and Information Engineering, Faculty of Engineering, Universitas Gadjah Mada, Jln. Grafika 2, UGM Yogyakarta 55281 (e-mail: adji@ugm.ac.id, noorwewe@ugm.ac.id)*

[2] *Department of Electrical and Information Engineering, Faculty of Engineering, Universitas Gadjah Mada, Jln. Grafika 2, UGM Yogyakarta 55281 (e-mail: dwi.retno.p@mail.ugm.ac.id)*

relationships by storing links connecting one document to other documents [8]. While the embedded document model represents a relationship between documents by storing interconnected data in a single document structure [7]. The referred data is represented by forming an embedded document as a sub-document, in a field or array in a document [9]. Therefore, to answer the need for interrelated data representation, MongoDB is used as a destination database in the development of this migration system.

In recent times, several tools have been developed to migrate relational databases into the NoSQL database, such as Apache Sqoop and JackHare. Apache Sqoop is a tool to effectively transform large amounts of data from relational databases into Hadoop [10]. Sqoop has successfully migrated SQL into NoSQL. However, the existence of relations between tables still causes limitations in query with a large number of JOIN keys in transformation results table. While JackHare, a framework for translating SQL into NoSQL using MapReduce [11], has succeeded in converting all tables in a relational database into a single table in HBase. However, a certain additional column family is still needed to store foreign keys from relational database, which causes poor performance because it has to process a large number of JOIN operations [5].

In addition, in a research, a method has been introduced to transform SQL databases into NoSQL [5]. The method was based on the nesting table process, where a table A referenced table B, and table B referenced table C, which was called a multiple nested state. In this method, the SQL database schema was transformed following a multiple nested process, thus forming a single table in the NoSQL database. The test results in the research indicated that the proposed method successfully migrated SQL databases into NoSQL, and obtained query performance in the NoSQL database faster than in SQL databases [5]. Therefore, multiple nested methods were used in the process of transforming SQL databases into NoSQL in this work.

One aspect that needs to be considered in database migration process is data integrity, which ensures data consistency in database before and after migration process. Database migration can be said to be successful when each document model can represent the same data in both databases [12]. Therefore, in this database migration, it is necessary to test *data integrity*, to ensure that designed migration system can represent same data in database before and after migration.

II. Database Migration and Transformation

The core process of this work is to design a relational database migration system into non-relational databases using multiple nested methods. Therefore, it is necessary to review previous research regarding database migration and database schema transformation methods.

A. Database Migration Tools

Several tools have been developed to migrate relational databases into non-relational databases.

Apache Sqoop [10] is a tool developed to efficiently migrate bulk data between relational databases and non-relational databases, namely Apache Hadoop. Sqoop is able to migrate SQL databases into NoSQL. While JackHare [11] is a framework for translating SQL databases into NoSQL using MapReduce. JackHare successfully converts all tables from the SQL database into a single table in HBase. According to [5], although Sqoop and JackHare have successfully migrated SQL databases into NoSQL, these two tools still show poor performance, which is caused by the large number of foreign keys in the migration results database.

Reference [13] developed NoSQLayer, a framework to support the relational database migration process into NoSQL. The way NoSQLayer works is divided into two parts, namely the migration module and mapping module. In the migration module, source database elements such as tables, attributes, relations, and indices are automatically identified and then migrated into NoSQL. Then, the mapping module which is the application interface with the DBMS monitors all SQL transactions from the application and translates and changes operations into the NoSQL model that was created in the previous module. The framework evaluation shows that NoSQLayer can help migrate large amounts of data automatically without losing data.

In another study, MigDB was developed, a tool for automatically migrating MySQL databases into MongoDB [14]. MigDB divided the work process in several stages. Firstly, MySQL was mapped into JSON. Then the relation mapping module with the help of the neural network decided how to map relations. The management module converted SQL queries, generated MongoDB queries, and manipulated collections in MongoDB database resulted from migration. The evaluation results showed that MigDB was able to migrate tables, relations, data, and queries into MongoDB without initially requiring MongoDB knowledge.

B. Transformation Method

In addition to tools development, several studies were conducted before to study process of transforming schemes from relational databases into the NoSQL database.

In research, development of a database transformation system was carried out in the MySQL database into MongoDB by utilizing structure and relations between tables as main parameters in model formation algorithm [12]. The system was tested by data integrity testing and accessing query time that was run on MySQL and MongoDB. The test results showed that developed system could migrate database by making some adjustments.

In other studies, a Graph method has been introduced to convert SQL database schema into NoSQL [15]. The graph method was designed based on nesting table process. Testing conducted in the study resulted a finding that graph method successfully transformed SQL databases into NoSQL without losing data.

In another study, steps to change the relational scheme to the Hbase scheme were discussed [16]. In the study, nested conditions have been carried out, but it was not with multiple nested conditions, which means that tables with multilevel relations cannot be mapped to the HBase scheme.

Continuing the previous research that has not defined relations transformation between tables in a specific relational database, Multiple Nested Schema method was re-introduced [5]. The method was based on the nesting table process, where a table A referenced table B, and table B referenced table C, which was called a multiple nested state. In this method, the SQL database schema was transformed following a multiple nested process, thus forming a single table in the NoSQL database. The test results in the research indicated that the proposed method successfully migrated SQL databases into NoSQL, and obtained query performance in the NoSQL database faster than in SQL databases.

Therefore, in this development, multiple nested schema methods were applied in the process of transforming SQL databases into NoSQL. This method was carried out and tested on the academic data of the Darmajaya Institute of Informatics and Business. This work is expected to be the first step for designing data migration methods that are more effective and efficient, and are expected to contribute to the development of science.

III. System Design Method

Implementation procedure of this system design is divided into three stages as follows.

A. Initial Processing

1) Confidential Data Incognito: This confidential data incognito aimed to protect data owner's privacy. Data incognito process was carried out by creating an incognito data formula, which was then executed on the SQL query on table needed to be disguised, i.e., 'Mahasiswa' Table. To disguise student's name with NPM 04030348, a disguised data with a 'namamhs' + NPM formula in SQL query, producing value 'namamhs04030348', was formulated. A similar thing was done on these attributes: 'alamat', 'telpon', 'namaortu', 'alamatortu', 'telponortu', and 'Nosttb' in the 'Mahasiswa' Table.

2) Identification of Relations: Before starting the design of SQL database transformation schemes into NoSQL, it was necessary to identify relations or connections between entities or tables from SQL databases. Obtained relations between tables were used as a reference in the transformation process at a later stage.

B. Design of Transformation Schemes

The design of SQL database transformation scheme into NoSQL was conducted by referring to a Multiple-Nested Schema method [5]. Steps of NoSQL schemes design are shown in Fig. 1.

1) Single Nested Conversion: 'Mahasiswa' and 'Jurusan' Tables: In this single nested conversion, 'Mahasiswa' Tables refer to 'Jurusan' Tables in SQL database. To convert those to tables into one table in NoSQL, the following rules were followed [5].

a. Table name in NoSQL was main table same as in SQL database, namely 'Mahasiswa'.

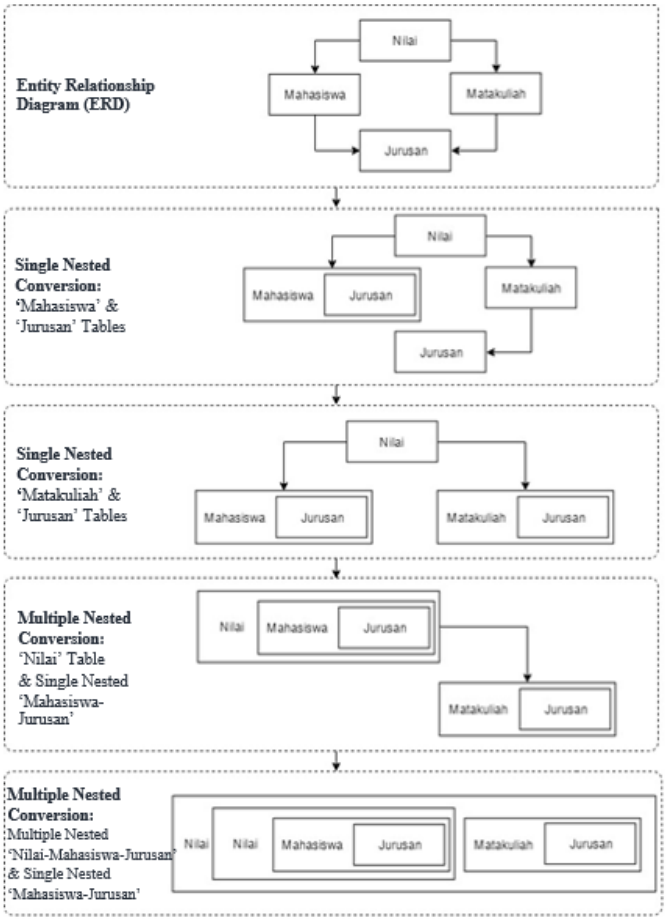

Fig. 1 Database scheme conversion steps.

b. Rowkey was primary key of 'Mahasiswa' Table in SQL database, which was 'mNPM'.
c. Column family name was the name of the main table and tables referenced in the SQL database, namely 'Mahasiswa' and 'Jurusan'.
d. In the 'Mahasiswa' Table in the SQL database, a foreign key 'mKodeJurusan' referenced the table to 'Jurusan' Table. In NoSQL, 'Mahasiswa' Table contained two column families, the first was 'mahasiswa' containing all attributes except the table's primary key in SQL database. The second family column was 'jurusan' containing all attributes of 'Jurusan' Table in SQL database.

Fig. 2 is the conversion of single nested schemes, namely 'Mahasiswa' Tables and 'Jurusan' in SQL database into 'Mahasiswa' Tables in NoSQL.

2) Single Nested Conversion: 'Matakuliah' and 'Jurusan' Tables: Following a same rules as single nested conversion in previous section, 'Matakuliah' and 'Jurusan' Tables from SQL database were converted into NoSQL 'Matakuliah' Table.

3) Multiple Nested Conversion: 'Nilai' Tables and Single Nested 'Mahasiswa'-'Jurusan': In this multiple nested scheme conversion, results of single nested conversion in previous step were nested in a 'Nilai' Table.

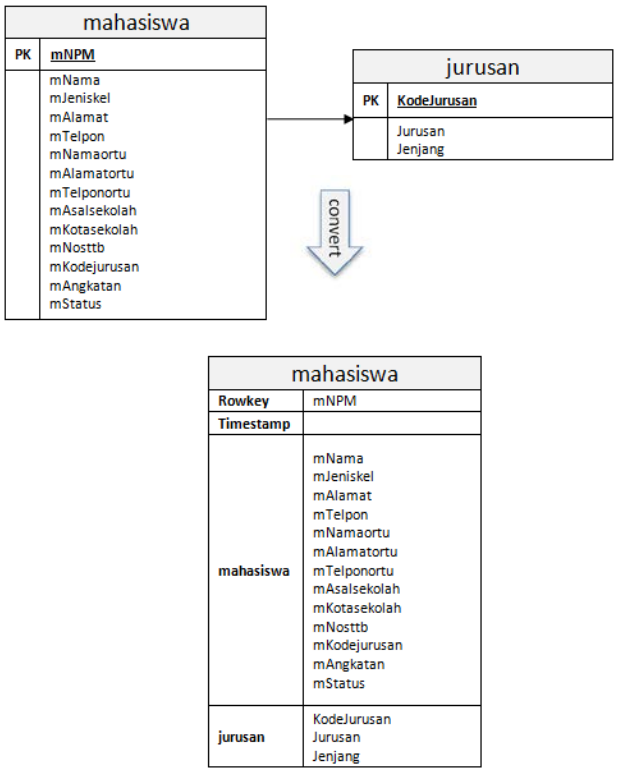

Fig. 2 Single nested scheme conversion designs for 'Mahasiswa' and 'Jurusan' Tables.

4) Multiple Nested Conversion: Final: In this step, results of multiple nested conversions in previous step were nested with results of single nested 'Matakuliah'-'Jurusan' conversion, thus it formed a single table scheme in NoSQL.

C. Design Implementation

The created database schema conversion was then be implemented in the transformation making process in PDI/Kettle.

1) Single Nested Conversion: 'Mahasiswa' and 'Jurusan' Table: There are three steps in this process.

a. Table Input. The transformation process began by creating table input steps on transformation canvas. In this step, the utilized inputs were two tables from SQL database, namely 'Mahasiswa' Tables and 'Jurusan' Tables in the Darmajaya database.
b. Sort Rows. This step was required as a condition to conduct a merge join, that was tables to be merged had to be sorted by the same key or key field. In 'Mahasiswa' and 'Jurusan' Tables, the same key field was 'KodeJurusan', which was a primary key from 'Jurusan' Table, and foreign key in 'Mahasiswa' Table. Then, 'KodeJurusan' attribute was selected as key field in Sort Rows step.
c. Merge Join. Merge join step was used to convert two single nested SQL tables into one table, namely 'Mahasiswa' Table nest 'Jurusan' Table. In this conversion, the 'Mahasiswa' Table acted as the left table, and the 'Jurusan' Table acted as a right table, so to obtain all data from the 'Mahasiswa' Table along with data from the related 'Jurusan' Tables, LEFT JOIN operations were used to conduct merge join.

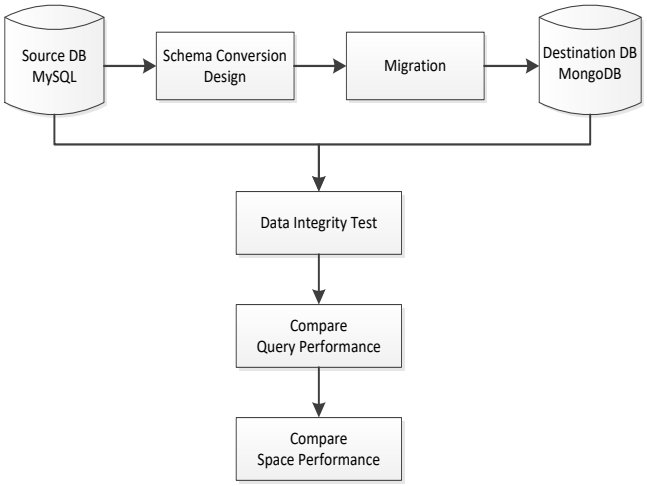

Fig. 3 Testing scenario.

2) Single Nested Conversion: 'Matakuliah' and 'Jurusan' Tables: With the same rules as the previous conversion process, in this step, the 'Matakuliah' and 'Jurusan' Tables from the SQL database were used in the input. Then sort rows used 'KodeJurusan' key field, and LEFT JOIN operation to conduct merge joins

3) Multiple Nested Conversion: 'Nilai Table and Single Nested 'Mahasiswa'-'Jurusan': In this step, the utilized inputs were a 'Nilai' Table from SQL and a table, conversion result of single nested 'Mahasiswa'-'Jurusan'. Sort rows stage used NPM key field, and LEFT JOIN operation at the merge join step.

4) Multiple Nested Conversion: Final: In this step, the utilised input was derived from conversion results in the previous stages, namely the multiple nested 'Nilai'-'Mahasiswa'-'Jurusan' conversions results and single-nested of 'Matakuliah'-'Jurusan' conversion results. Sort rows used 'KodeMK' key field and 'KodeJurusan', with LEFT JOIN operation to merge joins.

5) MongoDB Output: The last step in this transformation was loading process into the migration destination database, namely MongoDB output. In this step, configuration of MongoDB server connection, 'darmajaya' database connection, 'nilai' collection, and Mongo document path, as well as index creation, which resulted in MongoDB document structure, were carried out.

D. Testing

Testing was carried out on three aspects, namely data integrity, query performance, and space performance. Testing was carried out by comparing measurement results of the three parameters in each database, namely MySQL as the source database, and MongoDB as the destination database. Testing scenario is shown in Fig. 3.

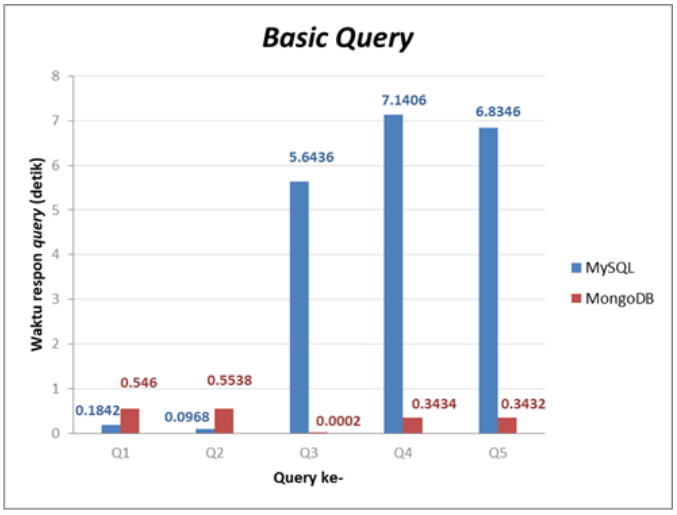

Fig. 4 Graph of basic query response time comparison in MySQL and MongoDB.

IV. Results and Discussion

A. Test Results for Query Response Time

1) Basic Query: Fig. 4 is a chart on comparison of basic query testing result based on query response time on source database and migration destination. At the same time, a comparison summary of response time testing result on basic query is shown in Table I.

From graph in Fig. 4 and Table I, it can be seen that Q1, which is a query only involving one table, namely 'Mahasiswa' Table, with SELECT COUNT, WHERE, and AND operations, MySQL migration source database actually shows faster performance than migration result database in MongoDB, which is equal to 0.1842 seconds for MySQL and 0.546 seconds for MongoDB. Or it can be said that MySQL is 2.96 times faster than MongoDB.

Similar to Q1, basic query Q2 is also a query that only involves one master table in MySQL, with the SELECT DISTINCT, WHERE, and AND operations, MySQL migration source database showing performance of 5.72 times faster than the migration results database.

This happens because in MySQL, basic queries Q1 and Q2 only access 'Mahasiswa' Tables, which are master tables, with records number of 11,236 rows. Whereas in MongoDB, Q1 and Q2 access 'Nilai' collection, containing entire master table and transaction table from migration source database, with records number of 401,712 documents, as shown in Table I. In other words, computational process of queries Q1 and Q2 loaded to each database is not balanced, causing faster query response time in MySQL of 2.96 times and 5.72 times that of the MongoDB database.

Furthermore, Q3, Q4, and Q5 are basic queries involving only transaction tables, namely 'Nilai' Tables in MySQL, with SELECT COUNT operations in Q3, SELECT DISTINCT and WHERE in Q4, and SELECT COUNT and WHERE in Q5. Testing results show that MongoDB migration results database provide a faster query response time than migration source database of 28,218 times for Q3, 20.79 times for Q4, and 19.91 times for Q5, as shown in Table I.

Table I
Summary of Basic Query Response Time Testing Results

	Involved SQL Table	Number of Records	Involved MongoDB Collection	Number of Records	Comparison Response time
Q1	'Maha-siswa'	11,236	'Nilai'	401,712	MySQL 2.96 times faster
Q2	'Maha-siswa'	11,236	'Nilai'	401,712	MySQL 5.72 times faster
Q3	'Nilai'	401,712	'Nilai'	401,712	MongoDB 28,218 times faster
Q4	'Nilai'	401,712	'Nilai'	401,712	MongoDB 20.79 times faster
Q5	'Nilai'	401,712	'Nilai'	401,712	MongoDB 19.91 times faster

Therefore, it can be said that in basic queries involving transaction tables, MongoDB migration results database shows query response time 28.6 times faster than MySQL's migration source database. Vice versa, in basic queries that do not involve transaction tables, MySQL shows query response time of 3.91 times faster than the migration results database.

From comparison of basic query test results, it was discovered that total average response time for processing five basic queries in source database was 19.8998 seconds, while result of migration database was 1.7866 seconds. Therefore, it was discovered that required total time to perform basic queries on the migration results in MongoDB was 11.13 times faster than migration source database in MySQL.

2) Complex Query: Fig. 5 is a comparison chart of complex query testing results based on query response time in source database and migration destination. A summary of comparison of complex query response time testing results is shown in Table II.

From graph in Fig. 5 and Table II, it can be noted that in Q1, which is a query involving four SQL tables, with SELECT DISTINCT operation, three JOIN operations, WHERE, and three AND operations, migration results databases delivers performance 7.041 times faster than source database. This happens because the query process running on MySQL requires system to find information on four tables combined through a JOIN operation. Whereas when querying the same information in migration results table in MongoDB, system only queries one table that has been combined through a transformation process.

Likewise in Q2, which involves four SQL tables, with SELECT operations, three JOIN operations, WHERE, AND, SUBQUERY, and ORDER BY, MongoDB database provides performance of 20.471 times faster than the MySQL database. In Q3, which involves three SQL tables, with SELECT operations and two LEFT JOIN operations, the MongoDB database provides performance of 51.692 times faster than MySQL. Then in Q4 and Q5 involving two SQL tables, with SELECT DISTINCT, INNER JOIN, and WHERE operations, and AND in Q4, MongoDB database provides 11.483 times performance and 13.046 times faster than MySQL.

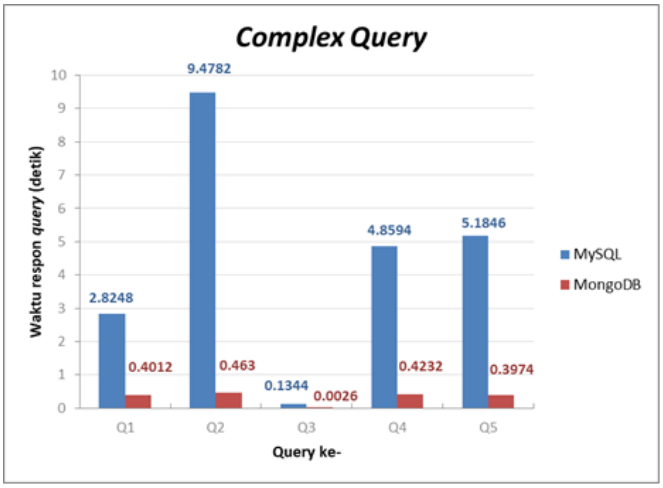

Fig. 5 Graph on comparison of complex query response time in MySQL and MongoDB.

TABLE II
SUMMARY OF COMPLEX QUERY RESPONSE TIME TESTING RESULT

	Involved SQL Table	Number of Records	Involved MongoDB Collection	Number of Records	Comparison Response time
Q1	'Nilai' 'Mahasiswa' 'Matakuliah' 'Jurusan'	401,712 11,236 1,678 9	'Nilai'	401,712	MongoDB 7.041x faster
Q2	'Nilai' 'Mahasiswa' 'Matakuliah' 'Jurusan'	401,712 11,236 1,678 9	'Nilai'	401,712	MongoDB 20.471x faster
Q3	'Nilai' 'Mahasiswa' 'Jurusan'	401,712 11,236 9	'Nilai'	401,712	MongoDB 51.692x faster
Q4	'Nilai' 'Mahasiswa'	401,712 11,236	'Nilai'	401,712	MongoDB 11.483x faster
Q5	'Nilai' 'Matakuliah'	401,712 1,678	'Nilai'	401,712	MongoDB 13.046x faster

This happens because MySQL database requires system to look for required information on more than one table combined with JOIN operations, which makes searching process takes longer time [5]. Unlike the migration results table in MongoDB, 'Nilai' collection has provided all the data in one table, which makes the search process proven faster in the five complex queries tested, as shown in the summary in Table II.

From the comparison of complex query test results, it was found that total average response time for processing five complex queries in source database was 22.4814 seconds, while migration result on database was 1.6874 seconds, so that required total time to perform complex queries on migration results in MongoDB was 13.32 times faster than migration source database in MySQL.

Therefore, it can be said that, in MongoDB database the database transformation result using multiple nested schema, complex queries or queries involving the existence of relations between tables or JOIN states, has a performance of 13.32 times faster than MySQL database, at all tested complex queries.

TABLE III
SUMMARY OF DATA INTEGRITY TESTING RESULTS IN MYSQL AND MONGODB

Query	Number of Query Results (rows)		Comparison of Query Results
	MySQL	MongoDB	
Complex Q1	20	20	Identical
Complex Q2	39	39	Identical
Complex Q3	401,712	401,712	Identical
Complex Q4	837	837	Identical
Complex Q5	11	11	Identical
Basic Q1	476	476	Identical
Basic Q2	213	213	Identical
Basic Q3	401,712	401,712	Identical
Basic Q4	7	7	Identical
Basic Q5	1	1	Identical

TABLE IV
COMPARISON ON SPACE PERFORMANCE TESTING RESULT OF SOURCE AND MIGRATION RESULTS DATABASES

Basis Data	Table	Storage Size	Total Storage Size
Migration Source	'Nilai' 'Matakuliah' 'Mahasiswa' 'Jurusan'	25.6 MB 336 KB 3.5 MB 16 KB	29,452 MB
Migration Result	'Nilai'	310,833 MB	310,833 MB

B. Data Integrity Testing Results

Table III is a summary of data integrity testing result on migration source and destination databases. From testing result in Table III, it can be seen that both databases display same query result, with same records number in every tested query, both complex query and basic query. It shows that database transformation process with multiple nested schema method is able to maintain data integrity in all tested queries.

C. Space Performance Testing Result

Table IV shows comparison of space performance testing result of source and migration result databases. Comparison on space performance testing result in Table IV shows that database resulted from transformation with multiple nested schema method in MongoDB requires data storage 10.53 time greater than migration source database in MySQL. This happens because the MongoDB database stores all related information in one record, resulting in a lot of data redundancy in migration result collection. However, at present, storage performance is not a top priority in data processing technology, so large storage requirement is a consequence of obtaining efficient query performance, which is still considered as the first priority in data processing technology [5].

V. CONCLUSIONS

Based on the work that has been done, several conclusions can be drawn as follows. A database transformation system design with Multiple Nested Schema has been successfully carried out. Relational database migration into non-relational database has also been successfully carried out, from MySQL into MongoDB. Result evaluation of database migration was

successfully carried out, with the following conclusions. Complex queries or queries involving existence of a relation between tables or JOIN state in SQL shows response time 13.32 times faster on migration results database in the MongoDB, rather than MySQL migration source database. Then, basic queries involving SQL transaction tables show response times 28.6 times faster in migration results database in MongoDB than MySQL migration source database. In contrast, basic queries that do not involve SQL transaction tables, MySQL shows query response times 3.91 times faster than MongoDB. Furthermore, database transformation process with Multiple Nested Schema method is able to maintain data integrity on the entire tested queries. Finally, database resulted from transformation with Multiple Nested Schema in MongoDB shows a storage requirement of 10.53 times greater than the MySQL source migration database. This is due to the large amount of data redundancy resulting from the transformation process. However, at present, storage performance is not a top priority in data processing technology, so large storage requirement is a consequence of obtaining efficient query performance, which is still considered as the first priority in data processing technology.

REFERENCES

[1] R.M. Stair and G.W. Reynolds, *Fundamentals of Information Systems*, Boston, USA: Courses Technology, 2008.

[2] K. Hallgren (2016) "How to Approach Data-Driven Decisions in Education," [Online] https://www.mathematica-mpr.com/commentary/data-driven-decisions-in-education, access date: 08-Nov-2018.

[3] J. Manyika, M. Chui, B. Brown, J. Bughin, R. Dobbs, C. Roxburgh, and A.H. Byers, "Big Data: The Next Frontier for Innovation, Competition, and Productivity," McKinsey Co. Tech. Report, pp. 1-156, 2011.

[4] S. Yin and O. Kaynak "Big Data for Modern Industry: Challenges and Trends," *Proc. of the IEEE*, Vol. 103, No. 2, pp. 143–146, 2015.

[5] G. Zhao, L. Li, Z. Li, and Q. Lin, "Multiple Nested Schema of Hbase for Migration from SQL," *Proc. - 2014 9th Int. Conf. P2P, Parallel, Grid, Cloud Internet Comput. 3PGCIC 2014*, 2014, pp. 338–343.

[6] A. Nayak, A. Poriya, and D. Poojary, "Type of NOSQL Databases and Its Comparison with Relational Databases," *Int. J. Appl. Inf. Syst.*, Vol. 5, No. 4, pp. 16–19, 2013.

[7] V. Manoj, "Comparative Study of Nosql Document, Column Store Databases and Evaluation of Cassandra," *Int. J. Database Manag. Syst.*, Vol. 6, No. 4, pp. 11–26, 2014.

[8] K. Chodorow, *MongoDB: The Definitive Guide 2nd Edition*. USA: O'Reilly Media, Inc., 2013.

[9] J. Speelpenning, J. Lounsberry, and A. Price-budgen, "Data Modeling and Relational Database Design Publishers," Oracle, Student Guide, Vol. 1, pp. 1-320, July, 2001.

[10] M. Dagar, S. Mittal, and M. Singh, "Conversion from Relational-Based Database to Column-Based Database," *Int. J. Sci. Res. Comput. Sci.*, Vol. 1, No. 1, pp. 29–35, 2013.

[11] W.C. Chung, H.P. Lin, S.C. Chen, M.F. Jiang, and Y.C. Chung, "JackHare: a Framework for SQL to NoSQL Translation Using MapReduce," *Autom. Softw. Eng.*, Vol. 21, No. 4, pp. 489–508, 2014.

[12] I.G. Winaya and A. Ashari, "Transformasi Skema Basis Data Relasional Menjadi Model Data Berorientasi Dokumen pada Mongodb," *Indonesian Journal of Computing and Cybernetics Systems (IJCCS)*, Vol. 10, No. 1, 2015.

[13] L. Rocha, F. Vale, E. Cirilo, D. Barbosa, and F. Mourão, "A Framework for Migrating Relational Datasets to NoSQL," *Procedia Comput. Sci.*, Vol. 51, No. 1, pp. 2593–2602, 2015.

[14] G. Liyanaarachchi, L. Kasun, M. Nimesha, K. Lahiru, and A. Karunasena, "MigDB - Relational to NoSQL Mapper," *2016 IEEE Int. Conf. Inf. Autom. Sustain. Interoper. Sustain. Smart Syst. Next Gener. (ICIAfS 2016)*, 2016, pp. 1-6.

[15] G. Zhao, Q. Lin, L. Li, and Z. Li, "Schema Conversion Model of SQL Database to NoSQL," *Proc. - 2014 9th Int. Conf. P2P, Parallel, Grid, Cloud Internet Comput. 3PGCIC 2014*, 2014, pp. 355–362.

[16] C. Li, "Transforming Relational Database into HBase: A Case Study," *Proc. 2010 IEEE Int. Conf. Softw. Eng. Serv. Sci. (ICSESS 2010)*, 2010, pp. 683–687.