Enhanced Graph Transforming V2 Algorithm for Non-Simple Graph in Big Data Pre-Processing

by Dr. Sutedi, S.kom., M.t.i

Submission date: 27-Sep-2021 06:30PM (UTC+1000)
Submission ID: 1658591161
File name: V2_Algorithm_for_Non-Simple_Graph_in_Big_Data_Pre-Processing.pdf (3.42M)
Word count: 7540
Character count: 38277

Enhanced Graph Transforming V2 Algorithm for Non-Simple Graph in Big Data Pre-Processing

Sutedi, Noor Akhmad Setiawan, and Teguh Bharata Adji, *Member, IEEE*

Abstract—Incapability of relational database in handling large-scale data triggers the development of NoSQL database that becomes part of a big data ecosystem. NoSQL database has different characteristics compared to the relational database. However, NoSQL database requires data from the relational database as one of the structured data sources. Therefore, data pre-processing is required to ensure proper data migration from a relational database to NoSQL database. This data pre-processing is normally called data transformation. One of the simple and understandable transformation algorithms is graph transforming algorithm. However, the algorithm has a problem in solving a non-simple graph (multigraph). This research proposes an algorithm to overcome several multigraph problems. The experimental work confirms that the algorithm proposed in this research is able to transform data from a relational database to NoSQL schema that has a minimum number of redundant attributes while the data completeness is still maintained.

Index Terms—SQL to NoSQL, multiple edges, loop, multigraph, graph

1 Introduction

The massive and continuous growth of varied data in a computation system in a cyber world results in an enormous collection of data [1]. In fact, the explosion of data goes beyond the capacity of storage media and the ability of the existing relational database management systems. This limited technology capability in data management leads to the existence of big data technology. The term of big data is used to describe a massive dataset [2] comprising of structured, semi-structured, and unstructured data [3].

In an attempt to handle the massive dataset, NoSQL database is developed in a big data ecosystem. This database is increasingly popular [4] due to its simplicity of access, velocity, and scalability [5]. Those things become the trigger factors for corporates to put an attention on utilizing NoSQL database. Based on the way of storing data, NoSQL database can be classified into four categories [6], that is, key-value oriented, column-oriented, document-oriented, and graph-oriented. These NoSQL database categories have distinctive features compared to relational database management system (RDBMS) [7]. Relational database is designed based on normalization concept and not particularly prepared for handling data partition and replication. Meanwhile, NoSQL database is designed based on denormalization concept and thus prepared for handling the issues in data partition and replication [8].

A significant different characteristic between RDBMS and NoSQL database leads to the need for a good and reliable method for data pre-processing [9]. In such data pre-processing, the data from RDBMS is transformed to a NoSQL database. The transformation process must be correct so that data can be integrated into other data sources in a NoSQL database.

One of the data transformation methods from SQL database to NoSQL database developed in [9] is graph transforming algorithm. The algorithm is simpler compared to other transformation methods and the resulted NoSQL database schema can store the required information completely. Nevertheless, graph transforming algorithm developed in [9] results in a schema that has high redundancy level so that an effort was done in [10] to develop an algorithm (i.e., enhanced graph transforming algorithm) which is able to reduce high redundancy levels.

The problem of both algorithms in [9] and [10] is that they have no mechanism for transforming relational database that has non-simple graph characteristic (multigraph), i.e., multiple edges and loop. The existence of multiple edges and loop can make a graph without a leaf node, which makes the algorithms developed in [9] and [10] ineffective because they always started the transformation process from the leaf node. Therefore, this current research aims to develop additional rules that enable the enhanced graph transforming algorithm to handle multigraph and yield a schema which guarantees data completeness and faster, easier queries. Another research issues include efforts to accelerate the query process and to develop schema designs for various types of NoSQL databases that are less redundant.

- *Sutedi is with the Department of Electrical and Information Engineering, Universitas Gadjah Mada, Yogyakarta, Indonesia, and the Department of Information System, Institut Informatika dan Bisnis Darmajaya, Bandar Lampung, Indonesia. E-mail: sutedi.s3te15@mail.ugm.ac.id.*
- *N.A. Setiawan and T.B. Adji are with the Department of Electrical and Information Engineering, Universitas Gadjah Mada, Yogyakarta, Indonesia. E-mail: {noorwewe, adji}@ugm.ac.id.*

Manuscript received 12 Feb. 2018; revised 18 Sept. 2018; accepted 1 Nov. 2018. Date of publication 12 Nov. 2018; date of current version 5 Dec. 2019. (Corresponding author: Teguh Bharata Adji.)
Recommended for acceptance by J. Levandoski.
Digital Object Identifier no. 10.1109/TKDE.2018.2880971

2 Related Works

Several previous researches had been conducted in relation to data transformation process from SQL database to NoSQL database and one of them was conducted in [9]. In the study, the SQL database was converted into a graph and then each vertex was merged gradually. The output was a process sequence for integrating relational tables into a NoSQL database schema. The scheme proved to be able to provide all the data needed. Unfortunately, the algorithm allowed redundancy in an effort to improve query efficiency. This took too many spaces and therefore, some way to reduce the redundancy is required.

The study in reference [10] attempted to overcome the weakness in [9] by developing an enhanced graph transforming algorithm. The algorithm was developed by adding rules to detect and delete transitive dependency among vertices in a graph before integrating vertices and deleting edges. The implementation of these rules can actually decrease the redundancy level that previously occurred in the NoSQL database schema produced by graph transforming algorithm. However, the developed algorithm cannot handle multigraph transformation.

The development of data transformation methods from an SQL database to NoSQL database was also performed in [11]. The study developed a multiple-nested schema based on the idea of a fully nested concept. The multiple-nested schema allowed data queries in one NoSQL table. The schema could integrate data from several tables in a relational database to a single table in the NoSQL database with the data remaining complete. The basic idea of the process in [11] is to transform each table in a relational database into a column family in the NoSQL table. The query performance on the NoSQL table was dramatically enhanced for a join query. However, this schema design took many space storages. Therefore, it needs revising to achieve lower redundancy level and better query performance.

The research in reference [12] developed a transformation method based on the principal of DDI (Denormalization, Duplication and Intelligent Keys) design. The NoSQL database design pattern used in this mechanism was tall narrow in which the table consisted of fewer column with more rows. Every data row in NoSQL table was given an intelligent key to identify the row. The selected row keys had to be able to keep the highest cardinality. Thus, cross-table query to represent data from the NoSQL database schema was no longer needed. Nevertheless, the row key determination mechanism and the dependency among column families in the research needs some improvement.

The research in reference [13] has completed the process in [12] by introducing automatic row key determiner. This was done to reduce overhead occurring during data migration from SQL database to NoSQL database. However, the imprecision problem in choosing row key and the disappearance of relational dependency among column families remain a weakness in the research. Therefore, the data migration and transformation method needs to be continuously developed.

In an attempt to make the data migration process from relational (MySQL) to NoSQL (MongoDB) database easier, reference [8] developed NoSQLayer framework by presenting migration and mapping modules. In accordance with the experiment, it was revealed that those modules in the NoSQLayer framework could function correctly to handle data with a big volume. However, the framework needs to be developed further so that its implementation will not be concerned only with MongoDB. In addition, efforts to reduce overhead costs in the abstraction layer are highly needed, so that the framework can be applied to efficiently migrate small volumes of data.

In relation to the data migration process, the study in [14] utilized Amazon cloud services for migrating data from a structured database (SQL database) to unstructured one (NoSQL database). The study proposed a system comprising three main modules: user registration module, database upload/download module, and database conversion module. Registered users could upload the database and download the conversion results from the system. Nevertheless, the system did not have a mechanism to migrate the SQL database, which has characteristics of multiple and unary relationships.

The research in [15] reviewed a mechanism to migrate data from SQL database to column-based NoSQL database. It utilized SQOOP to migrate data from Oracle 11g to Hadoop Distributed File System (HDFS). The result of data conversion was stored in XML/CSV format or other data storage formats. Afterwards, HBase, which is actually a column-based NoSQL database, would map the table based on the data access needed from a related application. Unfortunately, the executed table mapping mechanism is still general.

In accordance with the previously mentioned researches, there has been no mechanism to transform data from a relational database that has multiple and unary relationships. This paper will discuss the transformation mechanism proposed to overcome the issue.

3 Proposed Method

For most cases, the enhanced graph transforming algorithm can transform data from an SQL database to an NoSQL database. However, the algorithm is not able to handle relational database transformation with multiple and unary relationships. Multiple relationships on the graph are represented in multiple edges, while a unary relationship is denoted in a loop. To solve those cases, our current research proposes various additional rules, which will be elaborated in Sections 3.1, 3.2, and 3.3.

3.1 Rule to Transform Multiple Relationships

Table T1 has multiple relationships with Table T2 when there is more than one attribute which connects both tables. Multiple relationships can leave a graph without leaf nodes if it is formed of two tables that refer to each other. Take an example with Table Lecturer (T1) and Table Department (T2) in an Academic database (see Fig. 1). There are multiple relationships between both tables because they refer to each other. In that case, Table Lecturer refers to Table Department through the foreign key Dept_ID, which is the primary key of Table Department. Meanwhile, Table Department also refers to Table Lecturer via the foreign key Dept_Head_ID, which is the alias attribute of Lec_ID, the primary key of Table Lecturer. On one hand, a lecturer works only at one department in a college, while one department can

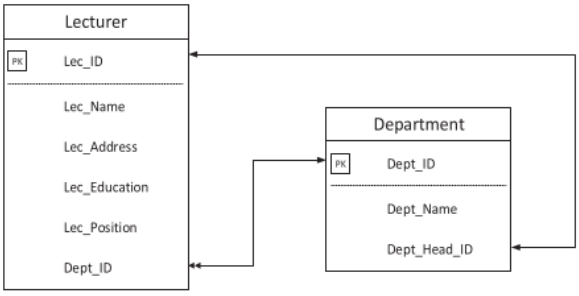

Fig. 1. Example of multiple relationships.

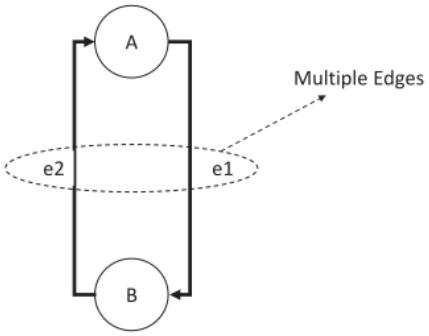

Fig. 2. Example of multiple edges in a graph.

house many lecturers. Thus, the degree of cardinality from Table Lecturer to Table Department is many to one (in Fig. 1, it is represented by a double arrow to a single arrow). On the other hand, the head of one department in a college is a lecturer, while a lecturer can become the head of only one department. Therefore, the degree of cardinality from Table Department to Table Lecturer is one to one (in Fig. 1, it is expressed in a single arrow to a single arrow).

In a graph, a relational table is indicated with a circular symbol known as "node" or "vertex", while the relationship between relational tables in a graph is shown by an arrow symbol called "edge". The edge direction in the graph shows that a table that becomes a predecessor vertex refers to another table that becomes a successor vertex. Predecessor vertex is a node on an arrow tail, while successor vertex is a node on an arrowhead. Thus, the relationship between Table Lecturer and Table Department as shown in Fig. 1 can be transformed into a graph as in Fig. 2. Vertex A in Fig. 2 represents Table Lecturer, while Vertex B indicates Table Department. The relationship between both tables is represented by Edges e1 and e2, showing multiple edges between the two vertices.

Multiple edges in Fig. 2 make the graph lacking any leaf node. This node is needed to get the transformation process started. Therefore, the initial phase to overcome the problem is by comparing the number of instance entities from both vertices. If the number of instance entities of Vertex A (denoting Table Lecturer) is more than the number of instance entities of Vertex B (denoting Table Department), then an additional vertex (Vertex A1) is formed to accommodate the attributes of Table Lecturer (Vertex A) that will be integrated into Table Department (Vertex B). Vertex A1 is a subset of Vertex A that contains some or all of the attributes of Table Lecturer (Vertex A). The selection of these attributes depends on the additional

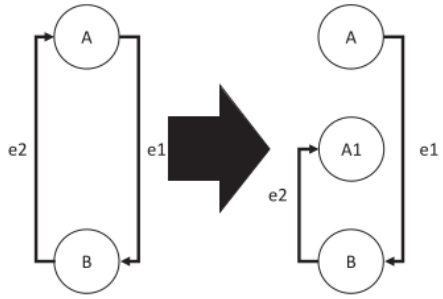

Fig. 3. The first phase of multiple edges transformation.

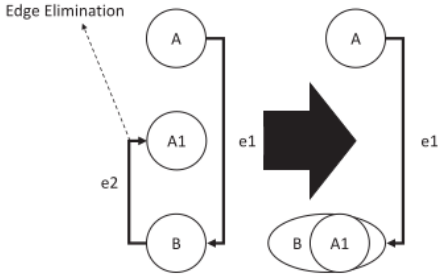

Fig. 4. The second phase of multiple edges transformation.

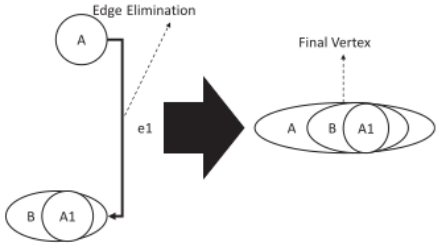

Fig. 5. The third phase of multiple edges transformation.

information needed in the Table Department (Vertex B). Meanwhile, if the number of instance entities of Vertex B is more than the number of instance entities of Vertex A, then Vertex B1 as an additional vertex is created. Vertex B1 is a subset of Vertex B to accommodate the attributes of Table Department (Vertex B) which will be integrated into Table Lecturer (Vertex A). In this case, the instance entities of Table Lecturer are more than those of Table Department, so that the additional vertex formed in Fig. 3 is Vertex A1. Next, Edge e2 that initially leads to Vertex A is converted to Vertex A1.

After multiple edges in the graph no longer exist, vertices integration and edges elimination start from the leaf node (Vertex A1). In Fig. 4, the second phase is performed by integrating Vertex A1 into Vertex B and eliminating Edge e2, which connects Vertex A1 with Vertex B.

In the next step, the result of integration between Vertex B and Vertex A1 is combined with Vertex A, and Edge e2 connecting Vertex A to Vertex B is eliminated. In Fig. 5, the iteration stops as the final vertex has both indegree and outdegree that equal zero.

The database table is mathematically noted as a set, in which the members of a set are the representation of attributes from related tables. Thus, Table Lecturer in Fig. 1 can be noted as Set A with its members $\{\underline{\boldsymbol{a}}1, a2, a3, a4, a5, \boldsymbol{b}1\}$,

```
X= {x | x ← ∀ instance entities A};
Y= {y | y ← ∀ instance entities B};
  If |X|>|Y| then
   {
     A1 = {x|(x ∈ A) ∧ (A1 ⊆ A) ∧ (A1 ⊂ B)};
     B = B ∪ A1;
     A = A ∪ B;
   }
  Else
   {
     B1 = {y|(y ∈ B) ∧ (B1 ⊆ B) ∧ (B1 ⊂ A)};
     A = A ∪ B1;
     B = B ∪ A;
   }
```

Fig. 6. Proposed rule to overcome multiple edges.

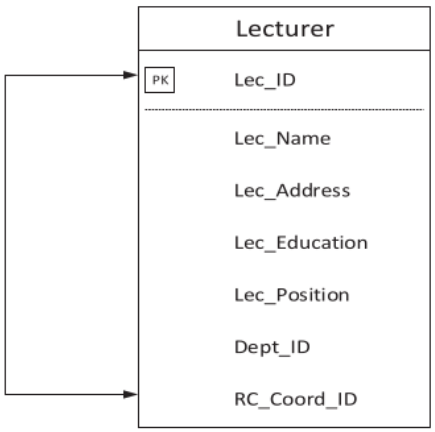

Fig. 7. The example of unary relationship.

while Table Department is noted as Set B with its own members $\{\underline{\boldsymbol{b1}}, b2, \boldsymbol{a1}\}$. Multiple relationships that occur between those two tables can be mathematically considered the same as the function, which is defined as a surjective-injective function. The function between Set A and Set B is defined as a surjective function from Set A to Set B and an injective function from Set B to Set A. Multiple relationships between those two tables, which are represented by set notation, are expressed in Equation (1) as follows:

$$|A \cap B| \geq 2. \tag{1}$$

The relationship between two sets that is represented by at least two intersect elements shows that there are multiple foreign keys (multiple relationships) which connect the two tables. In the case of Fig. 1, the multiple relationships occur on Set A (Table Lecturer) and Set B (Table Department), in which there are two elements of intersection between those sets, i.e., a1, which represents the attribute Lec_ID as well as Dept_Head_ID as an alias attribute to Lec_ID, and b1, which refers to the attribute Dept_ID. These two intersect elements show surjective-injective function between Set A and Set B, which means there are multiple relationships between both tables that make the graph lose leaf nodes. Thus, the solving-problem phase of the case is stated in an algorithm as shown in Fig. 6. At the beginning of the algorithm, there is Set X definition ($X = \{x|x \leftarrow \forall$ instance entities $A\}$), defined as a temporary set to accommodate all instance entities from Table Lecturer represented by Set A. Afterwards, the definition of Set $Y(Y = \{y|y \leftarrow \forall$ instance entities $B\})$, defined as a temporary set to accommodate all instance entities from Table Department represented by Set B. Then, the absolute values from Set X and Set Y are compared to get the number of instance entities that is bigger between them. If the absolute value of Set X is bigger than Y ($|X| > |Y|$ values '*true*') then Set A1 is formed to accommodate elements of Set A that is going to be merged with Set B $(A1 = \{x|(x \in A) \wedge (A1 \subseteq A) \wedge (A1 \subset B)\};)$. It is then followed by merging process with consecutive steps started with $B = B \cup A1$ and followed by A = A ∪ B. In contrast, if the absolute value of Set X is smaller than the absolute value of Set Y ($|X| > |Y|$ values '*false*' then Set B1 is formed to accommodate the elements of Set B that will be merged with Set A $(B1 = \{y|(y \in B) \wedge (B1 \subseteq B) \wedge (B1 \subset A)\};)$. It is then continued to the merging process through consecutive steps initiated with $A = A \cup B1$ and followed by $B = B \cup A$. In a case sample in Fig. 1, the number of instance entities of Table Lecturer represented by Set A is bigger than it is in Table Department, represented by Set B. Therefore, Set A1 is formed $(A1 \subseteq \{a2, a3, a4, a5\})$ and subsequently integrated into Set B $(B = B \cup A1 = \{b1, b2, a1, a2, a3, a4, a5\})$. Then, Set B is integrated into Set A $(A = A \cup B = \{a1, a2, a3, a4, a5, b1, b2, a1.1, a1.2, a1.3, a1.4, a1.5\})$ in which $\{a1.1, a1.2, a1.3, a1.4, a1.5\}$ are the alias attributes from $\{a1, a2, a3, a4, a5\}$. In order to handle this case, there is an anomaly in integrating the sets in which the same element between the sets is not merged into one as it is normally. Conversely, they are written in different names. In database concept, such case is known as "alias attributes", in which these attributes have identical characteristics to the genuine attributes but written in different names and with different data values. In its implementation, the number of alias attributes formed during the transformation process is adjusted to the user's need.

The implementation of the proposed algorithm for a graph that has multiple edges as elaborated in Fig. 6 can decrease the potential risk of data loss. For a relational database containing multiple relationships as shown in Fig. 1, the percentage of data loss that may arise due to incorrect transformation is calculated using Equation (2) below:

$$R = (100 - ((|RA|/|RB|) * 100)\,\%. \tag{2}$$

where R is defined as the risk percentage of data loss, $|RA|$ as the number of records from the table that has fewer instance entities and $|RB|$ as the number of records from the table that has more instance entities.

3.2 Rule to Transform Unary Relationship

The unary relationship may occur when a table in a relational database refers to itself by using a foreign key that is an alias attribute of the primary key from the table. This condition is illustrated in Fig. 7, which shows Table Lecturer comprising the attributes Lec_ID, Lec_Name, Lec_Address, Lec_Education, Lec_Position, Dept_ID, and RG_Coord_ID (Research Group Coordinator ID). This case indicates that every

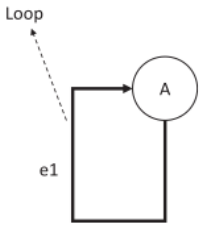

Fig. 8. Example of loop in a graph.

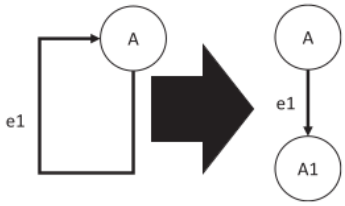

Fig. 9. The first phase of loop transformation.

lecturer has a research group coordinator that is also a lecturer. Thus, Table Lecturer has a foreign key: RG_Coord_ID, referring to primary key: Lec_ID in the table. RG_Coord_ID is an alias attribute of Lec_ID.

In a graph, a unary relationship is visualized into a loop as presented in Fig. 8. The graph in Fig. 8 comprises Vertex A that represents Table Lecturer, and Edge e1 that forms loop showing that Table Lecturer refers to itself. Edge e1 represents the foreign key: RG_Coord. The transformation process for the loop as presented in Fig. 8 started by forming Vertex A1 is used to accommodate attributes identified by the foreign key: RG_Coord. Afterwards, Edge e1 is directed to Vertex A1. The initial phase of the transformation process is illustrated in Fig. 9. Vertex A1 is an additional vertex that contains part or all attributes from Table Lecturer (Vertex A) identified by the foreign key: RG_Coord (Edge e1). This additional vertex formation is important to guarantee the completeness data from the transformation result. The following phase is vertices integration and edges elimination, illustrated in Fig. 10. In that case, Vertex A1 is combined with Vertex A, and Edge e1 is eliminated. The handling of loop transformation will make the query easier and will make the information more complete. The complexity of the information searching on the resulted schema is $O(1)$ until $O(n)$, in which $O(n)$ is the complexity of the information searching on a worst-case scenario. However, if the proposed rule is not used, the complexity is $O(2)$ to $O(2n)$, where $O(2n)$ is the complexity of the worst-case scenario. The complexity of the information searching is lower because the attributes from every record in the transformation result are already available in one record and thus there is no need to refer to other records.

In a mathematical model, the case of a unary relationship as in Fig. 7 can be expressed as a set, in which its elements are the representation of the available attributes in Table Lecturer. The mathematical model for that case is expressed in the following Equation (3):

$$A = \{a1, a2, a3, a4, a1.1\}. \tag{3}$$

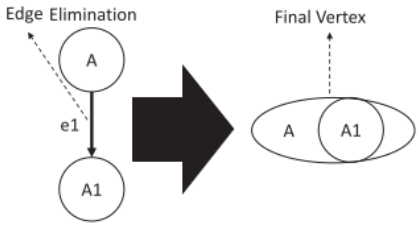

Fig. 10. The second phase of loop transformation.

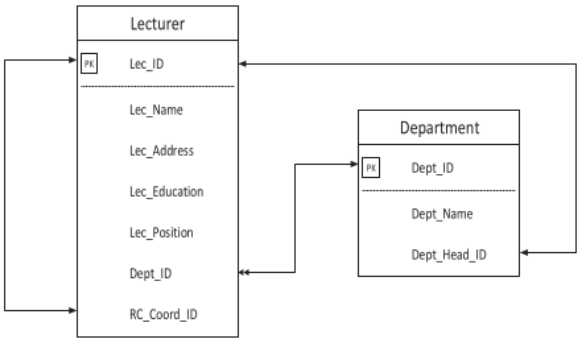

Fig. 11. Multiple relationships and unary relationship cases.

In the equation, the a1.1 element is an alias of the a1 element and it is an element forming the loop condition. The accomplishment of the transformation process starts with the formation of Set A1 to accommodate attributes from Set A identified by element a1.1. Thus, it becomes $A1 \subseteq \{a2, a3, a4\}$. The definition of Set A1 is expressed in Equation (4):

$$A1 = \{x | (x \in A) \Lambda (A1 \subseteq A).\tag{4}$$

After Set $A1$ is formed, it is combined with Set A. The integration process is stated in Equation (5):

$$A = A \cup A1.\tag{5}$$

The result of the integration is $A = \{a1, a2, a3, a4, a1.1, a1.2, a1.3, a1.4\}$ in which the elements $\{a1.1, a1.2, a1.3, a1.4\}$ are the alias elements of $\{a1, a2, a3, a4\}$. In a practical implementation, the number of resulted aliases is adjusted to the user's need.

3.3 Rule to Transform the Combination of Multiple Relationships and Unary Relationship

The combination of multiple relationships and unary relationship may occur when there is a table having one or more relationships with another table in a relational database and there is a table referring to itself. The example of a relational database in a combination of multiple relationships and unary relationship is illustrated in Fig. 11. In Fig. 11, there exists multiple relationships between Table Lecturer and Table Department, and there exists a unary relationship in Table Lecturer. Such condition is visualized in a graph as in Fig. 12.

On the graph, there are two vertices: Vertex A (Table Lecturer) and Vertex B (Table Department). Both vertices have multiple edges relation: Edge e1 as the representation of the foreign key: Dept_Head_ID and Edge e2 as the

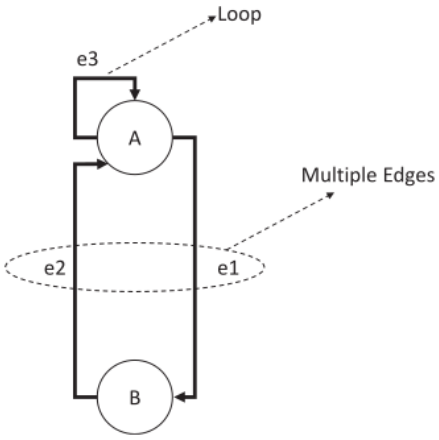

Fig. 12. The combination example of multiple edges and loop in a graph.

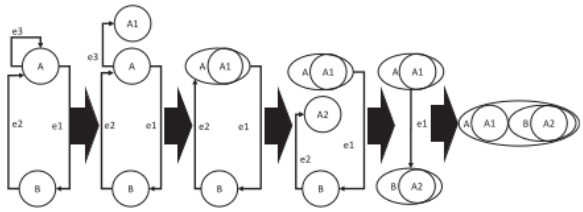

Fig. 13. A loop followed by multiple edges transformation.

representation of the foreign key: Dept_ID. Besides, in Vertex A, there is Edge e3 that forms loops. Edge e3 is the representation of the foreign key: RG_Coord_ID (the alias attribute of Lec_ID), which refers to the primary key: Lec_ID. In that case, we still follow the rules elaborated previously in Sections 3.1 and 3.2. The sequence of transformation processes is performed in three alternatives as follows:

a. Complete loop transformation first, followed by multiple edges transformation. The transformation process is illustrated in details in Fig. 13. The transformation process is commenced by forming an additional Vertex A1 ($V \leftarrow V + [A1]$). Afterwards, the Vertex A attributes based on the foreign key that forms the loop is identified. The attributes are then stored on the Vertex A1 ($A1 = \{y | (y \in A) \wedge (A1 \subseteq A)\}$). The next phase is integrating Vertex A1 to Vertex A ($A = A \cup A1$) and eliminating Edge e3. Since the number of instance entities of Vertex A representing Table Lecturer is bigger than it is in Vertex B representing Table Department, additional vertex A2 is added ($V \leftarrow V + [A2]$). It is followed by identifying attributes of Vertex A that becomes the additional attributes of Vertex B, and those attributes are stored on Vertex A2 ($A2 = \{x \,|\, x \in A) \wedge (A2 \subseteq A) \wedge (A2 \subset B)\}$). The next step is integrating Vertex A2 to Vertex B ($B = B \cup A2$) and eliminating Edge e2. Vertex B is then re-integrated to Vertex A ($A = A \cup B$) and Edge e1 is eliminated. Thus, the final vertex has outdegree and indegree that equal zero, and then the iteration is stopped.

b. Accomplish multiple edges transformation first, followed by completing the loop transformation. The transformation process is illustrated in detail in Fig. 14. This alternative of the transformation process begins with the formation of an additional Vertex A1 ($V \leftarrow V + [A1]$). The attributes of Vertex A that function as the additional attributes in Vertex B are then identified. Afterwards, the attributes are stored on Vertex A1 ($A1 = \{x \,|\, x \in A) \wedge (A1 \subseteq A) \wedge (A1 \subset B)\}$). It is performed because the number of instance entities of Vertex A representing Table Lecturer is bigger than it is in Vertex B representing Table Department. Vertex A1 is then integrated to Vertex B ($B = B \cup A1$) and Edge e2 is eliminated. It is followed by integrating Vertex B to Vertex A ($A = A \cup B$) and eliminating Edge e1. The next step is the formation of the additional Vertex A2 ($V \leftarrow V + [A2]$), followed by identifying the attributes of Vertex A using the foreign key that forms the loop. The attributes are then stored in Vertex A2 ($A2 = \{y | (y \in A) \wedge (A2 \subseteq A)\}$). Later, Vertex A2 is combined with Vertex A ($A = A \cup A2$) and Edge e3 is eliminated. In this step, the iteration is terminated because the final vertex has outdegree and indegree that equal zero.

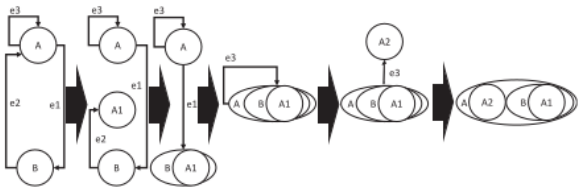

Fig. 14. Multiple edges followed by a loop transformation.

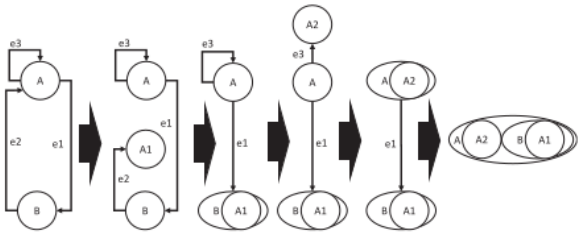

Fig. 15. Partial edges transformation on multiple edges followed by a loop and other partial multiple edges transformation.

c. Complete transformation of partial edges on multiple edges and accomplish loop transformation, followed by the transformation of other partial multiple edges. The transformation process is illustrated in Fig. 15. This transformation process is started by forming an additional Vertex A1 ($V \leftarrow V + [A1]$). Afterwards, the Vertex B attributes derived from those of Vertex A are identified. These attributes are then stored in Vertex A1 ($A1 = \{x \,|\, x \in A) \wedge (A1 \subseteq A) \wedge (A1 \subset B)\}$). Next, the integration of Vertex A1 to Vertex B ($B = B \cup A1$) and the elimination of Edge e2 are performed. The following step is forming an additional Vertex A2 ($V \leftarrow V + [A2]$). The attributes of Vertex A based on the foreign key that forms the loop are identified. The attributes are then stored into Vertex A2 ($A2 = \{y | (y \in A) \wedge (A2 \subseteq A)\}$). Next, Vertex A2 is integrated into Vertex A ($A = A \cup A2$) and Edge e3

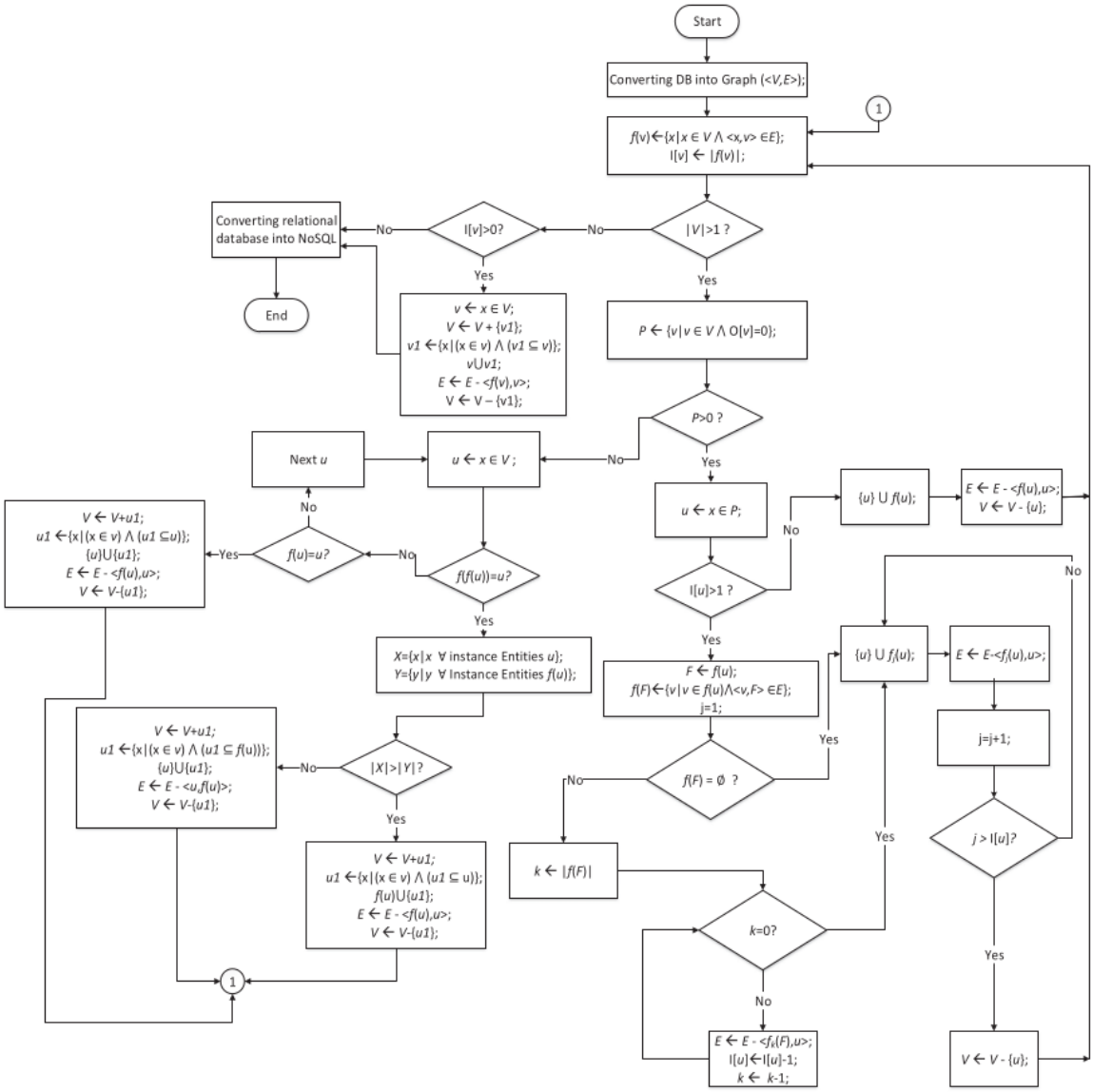

Fig. 16. Enhanced graph transforming v2 algorithm.

is eliminated, and Vertex B is integrated into Vertex A ($A = A \cup B$) and Edge e1 is also removed. Hence, the final vertex is obtained and the iteration is stopped. Although these three alternatives to transformation processes have different sequences, they always use the same pattern in their integration processes, in which the Vertex A and Vertex B are integrated after Vertex B acquired additional attributes from Vertex A. It is intended to make the resulted data remain intact.

The rules discussed in Sections 3.1, 3.2, and 3.3 are elaborated in an algorithm as shown in Fig. 16. The algorithm is implemented to transform data from relational database with simple and non-simple graph characteristics. The first step of this algorithm is performed by converting the relational database structure into a graph. If the number of Vertex in the graph is only one and loop occurs ($|V| = 1$ and $I[v] > 0$), the graph is added with a vertex used to store attributes identified by a foreign key that forms the loop ($V \leftarrow V + v1; v1 \leftarrow \{x|x \in V \wedge v1 \subseteq v;\}$. Next, vertices integration is completed and edges that form a loop is eliminated ($\{v\} \cup \{v1\}; E \leftarrow E - < f(u), v >; V \leftarrow V - v1;$). When the final vertex has outdegree and indegree that equal zero, the iteration is terminated. If the number of vertices in the graph is more than one ($|V| > 1$), identification of leaf node in the graph is done ($\mathrm{P} \leftarrow \{v \,|\, v \in V \wedge O[v] = 0\}$). If the graph has the leaf node ($|V| > 1;$ and $P > 0;$), the vertices integration process and edges elimination are completed based on the enhanced graph transforming algorithm proposed in [10]. However, when there is no leaf node ($|V| > 1;$ and $P = 0$), it is possible that there are multiple edges and/or loop in the graph. When there are multiple edges in the graph, the comparison of the number of instance entities between vertices related to

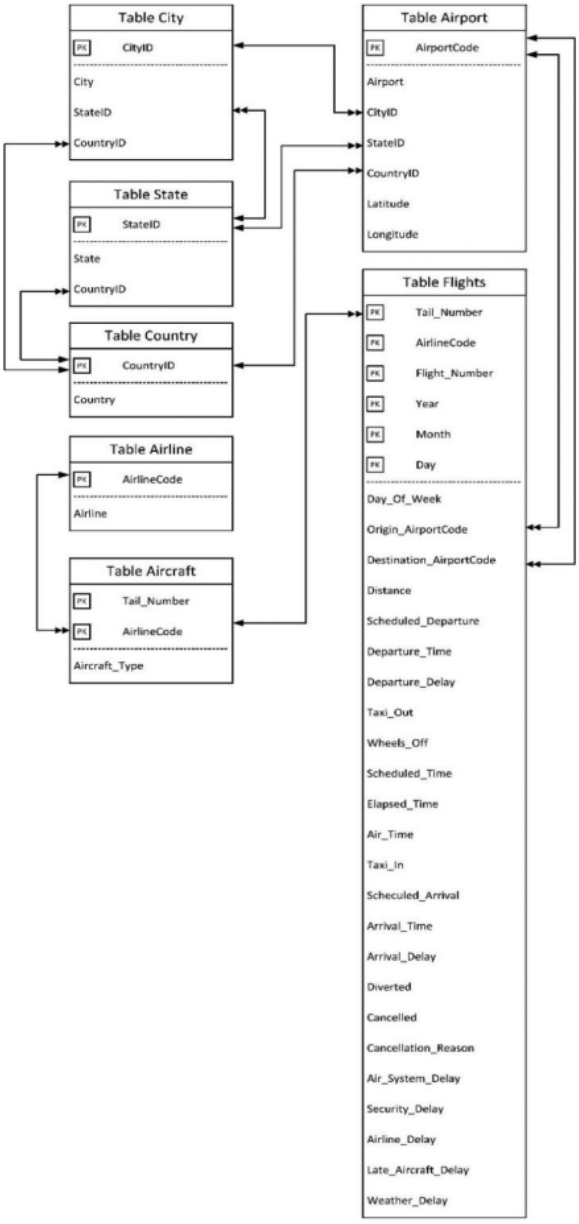

Fig. 17. Relational database extracted from the flight dataset.

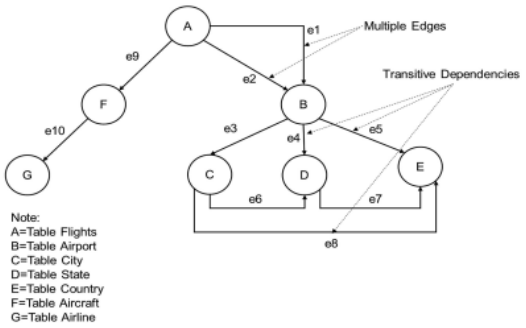

Fig. 18. The graph resulted from the conversion of the relational database in Fig. 17.

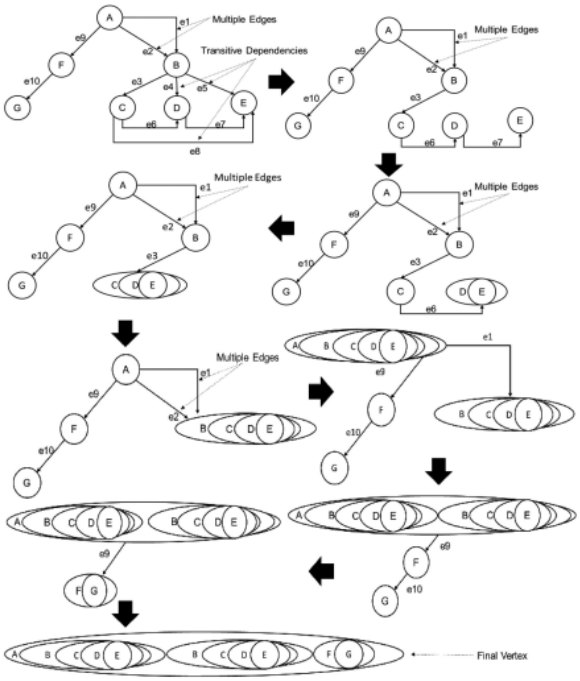

Fig. 19. Details of the transformation process.

multiple edges is made ($X = \{x|x \leftarrow \forall$ instance entities $u\}$; $Y = \{y|y \leftarrow \forall$ instance entities $f(u)\}$; |X| > |Y|?). Then, a vertex is added to accommodate attributes identified by the vertex that has a smaller number of instance entities $(V \leftarrow V + u1; u1 \leftarrow \{x|(x \in V) \wedge (u1 \subseteq u)\};$ or $u1 \leftarrow \{x|(x \in V) \wedge (u1 \subseteq f(u))\};)$. The next process is the integration of additional vertex to the one that has a smaller number of instance entities ($\{u\} \cup \{u1\}$; or $f(u) \cup \{u1\}$; $E \leftarrow E- < u, u1 >; V \leftarrow V - u1;)$. Afterwards, the following transformation process is repeated.

When there is a loop on a graph, a vertex is added to the graph to accommodate attributes identified by the foreign key forming the loop $V \leftarrow V + u1; u1 \leftarrow \{x|x \in V \wedge u1 \subseteq u\};)$. Then, it is integrated into the vertex that contains a loop, followed by the removal of edge that results in a loop ($\{u\} \cup \{u1\}$; E $\leftarrow$ E$- < u, u1 >; V \leftarrow V - u1;)$. This iteration of transformation is kept running until the final vertex has outdegree and indegree that equal zero.

4 Experimental Design, Result, and Discussion

Graph transforming algorithm proposed in [9] and enhanced graph transforming algorithm proposed in [10] initiate the transformation process from a leaf node, that is, a vertex that has outdegree = 0 and indegree $\neq$ 0. The existence of multiple edges and/or loop causes the value of outdegree to be more than zero. Therefore, the algorithms proposed in [9] and [10] face a problem in handling graph transformation that has multiple edges and/or loop. These issues are resolved after adding the rules proposed in our current research. This is important because failure to determine the order of vertices integration characterized by multiple edges

```
format_mongo =
        { "Tail_Number": Tail_Number,
          "AirlineCode": AirlineCode,
          "Flight_Number": Flight_Number,
          "Year": Year,
          "Month": Month,
          "Day": Day,

          "Aircraft_Type": Aircraft_Type,
          "Airline": Airline,
          "Day_Of_Week": Day_Of_Week,
          "Origin_AirportCode": Origin_AirportCode,
          "Origin_Airport": Origin_Airport,
          "Origin_CityID": Origin_CityID,
          "Origin_City": Origin_City,
          "Origin_StateID": Origin_StateID,
          "Origin_State": Origin_State,
          "Origin_CountryID": Origin_CountryID,
          "Origin_Country": Origin_Country,
          "Origin_Latitude": Origin_Latitude,
          "Origin_Longitude": Origin_Longitude,
          "Destination_AirportCode": Destination_AirportCode,
          "Destination_Airport": Destination_Airport,
          "Destination_CityID": Destination_CityID,
          "Destination_City": Destination_City,
          "Destination_StateID": Destination_StateID,
          "Destination_State": Destination_State,
          "Destination_CountryID": Destination_CountryID,
          "Destination_Country": Destination_Country,
          "Destination_Latitude": Destination_Latitude,
          "Destination_Longitude": Destination_Longitude,
          "Distance": float (Distance),
          "Scheduled_Depature": Scheduled_Depature,
          "Departure_Time": Departure_Time,
          "Departure_Delay": Departure_Delay,
          "Taxi_Out": Taxi_Out,
          "Wheels_Off": Wheels_Off,
          "Scheduled_Time": Scheduled_Time,
          "Elapsed_Time": Elapsed_Time,
          "Air_Time": Air_Time,
          "Taxi_In": Taxi_In,
          "Scheduled_Arrival": Scheduled_Arrival,
          "Arrival_Time": Arrival_Time,
          "Arrival_Delay": Arrival_Delay,
          "Diverted": Diverted,
          "Canceled": Canceled,
          "Cancellation_Reason": Cancellation_Reason,
          "Air_System_Delay": Air_System_Delay,
          "Security_Delay": Security_Delay,
          "Airline_Delay": Airline_Delay,
          "Late_Aircraft_Delay": Late_Aircraft_Delay,
          "Weather_Delay": Weather_Delay

        }
```

Fig. 20. NoSQL schema generated using enhanced graph transforming V2 algorithm.

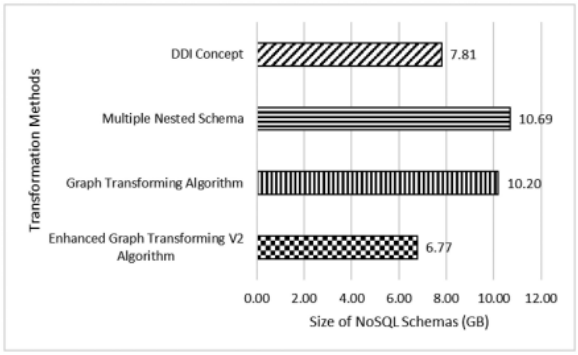

Fig. 21. Schema size (storage size) comparison.

can lead to significant information loss on the database. It should be understood that merging Vertex B to Vertex A ($A = A \cup B$) will produce different data compared to merging Vertex A to Vertex B ($B = B \cup A$). Both processes produce a different row key, so the number of records identified is also different. Meanwhile, database transformation with loops requires proper handling so that the desired information can be generated easily and completely. This is also important because the attribute that becomes a foreign key in the case of a loop is an alias attribute that causes the loop, which is frequently overlooked in a relational database. The handling of graph transformations that have a combination of multiple edges and loop must consistently comply with the rules described in Fig. 16.

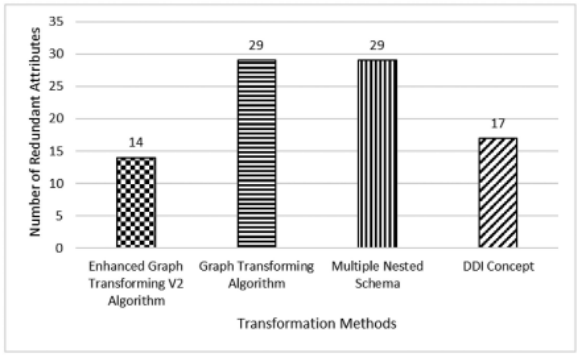

Fig. 22. The number of redundant attributes comparison.

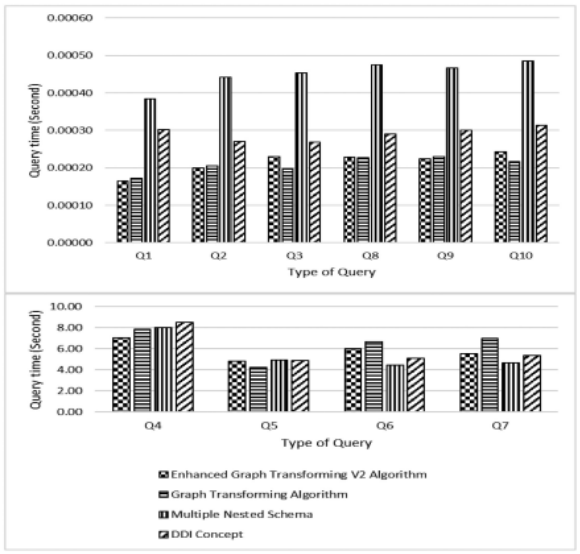

Fig. 23. Query time comparison.

4.1 Experimental Design

Experiment in this study is conducted to measure the performance of the proposed algorithm in handling data transformation into a document-oriented NoSQL database. The experiment uses public dataset from the Kaggle repository. The dataset is the flight delay and cancellation data in 2015, which is collected and published by The U.S. Department of Transportation's (DoT) Bureau of Transportation Statistics. In this experiment, we perform a simulation in which the data is migrated from the relational database to the NoSQL database. Therefore, the dataset of flight delay and cancellation that was originally stored in the CSV file is imported into the SQL Server database. After extraction, transformation and loading (ETL), the dataset is changed into seven relational tables, namely: Table City (318 records), Table State (54 records), Table Country (1 record), Table Airport (322 records), Table Airline (14 records), Table Aircraft (5,244 records), and Table Flights (5,332,914 records). In detail, the relationships and structure of the relational tables are shown in Fig. 17. In this case, there is no unary relationship but there are transitive dependencies between Table Airport and Table State, Table Airport and Table Country,

TABLE 1
List of Queries Tested

No	Query Commands
1.	db.getCollection('Flight').find({"Origin_City":"Chicago"})
2.	db.getCollection('Flight').find({"Airline":"Southwest Airlines Co.","Destination_City":"New York"})
3.	db.getCollection('Flight').find({"Day_Of_Week":"1","Origin_City":"Charlotte"})
4.	db.getCollection('Flight').distinct("Airline")
5.	db.getCollection('Flight').distinct("Destination_Airport",{"Destination_City":"New York"})
6.	db.getCollection('Flight').find({"Airline":"Skywest Airlines Inc."}).count()
7.	db.getCollection('Flight').find({"Airline":"Skywest Airlines Inc.","Origin_City":"New York","Destination_City":"Chicago"}).count()
8.	db.getCollection('Flight').find({"Distance":{$gt:3000},"Airline":"Delta Air Lines Inc."})
9.	db.getCollection('Flight').find({"Distance":{$lte:3000},"Origin_City":"New York"})
10.	db.getCollection('Flight').find({$or:[{"Origin_City":"Minneapolis"},{"Origin_Airport":" LaGuardia Airport (Marine Air Terminal)"}],"Destination_City":"Chicago"})

and Table City and Table Country. In addition, there are also multiple relationships between Table Airport and Table Flights, but the relationships are in the same direction. Therefore, the completion of the multiple relationships does not require comparing the number of instance entities between the two tables. The initial step in solving this case is to convert the relational database into a graph. The conversion results are shown in Fig. 18. The next step is to remove edges that cause transitive dependencies in the graph, followed by combining the vertices and removing the next edges starting from the leaf node. Details of the transformation process is illustrated in Fig. 19. Afterwards, the NoSQL schema is generated according to the sequence of combined vertices in the final vertex as shown in Fig. 19. The resulted schema is shown in Fig. 20. The row keys in the schema are adopted from the primary key of the root table in the relational database. After that, the data is migrated from the relational database to the NoSQL database schema. This experiment also runs the transformation process using graph transforming algorithm, multiple nested schema, and DDI concepts. The results of each method are compared in Section 4.2.

4.2 Experimental Results and Discussion

After the data has been migrated to each schema, the schemas are compared in size as shown in Fig. 21. The figure shows that the schema generated from the enhanced graph transforming V2 algorithm has the smallest size. This means that the schema is the most efficient compared to the others. Fig. 22 shows a comparison of the number of redundant attributes among these schemas. In the figure, it appears that the number of redundant attributes in the proposed schema is the least compared to the other schemas. The difference in schema size resulted from this study with the schemes generated from other studies is very significant. It can be seen from the experimental results that enhanced graph transforming V2 algorithm generates a schema that has the lowest amount of storage size with 6.77 GB. This size is 36.67 percent lower than the schema created with Multiple Nested Schema (10.69 GB), 33.63 percent lower than the schema created with graph transforming algorithm (10.2 GB), and 13.32 percent lower than the schema produced using DDI Concept (7.81 GB). In addition, this experiment also measures the speed of queries against data on each generated schema. The measurement uses a computer device with a Core i7 processor and 8 GB RAM and runs on applications that were built using Python programming language. In the measurement, 10 types of query commands (details are shown in Table 1) were executed. The results of the query speed measurement are listed in Fig. 23. From the measurement results, the average query speed in the proposed schema is relatively good. This shows that reducing the number of redundant attributes does not significantly affect the query performance. Thus, the transformation method proposed in this study is suitable for use.

5 Conclusion

Enhanced graph transforming v2 algorithm is able to transform data from simple-graph or non-simple-graph relational database to NoSQL database. The main advantage of the algorithm is that it can reduce the number of redundant attributes, while the data completeness is still maintained. In addition, the speed of the query in the proposed schema is fairly desirable. However, it must be kept in mind that the algorithm cannot transform data optimally in the case of relational databases that have subtype-supertype structures. Therefore, further research is needed to address the problem.

Acknowledgments

The researchers would like to address their gratitude to Universitas Gadjah Mada, Indonesia, and the Informatics and Business Institute Darmajaya, Indonesia. This research was conducted under the support of both institutions.

References

[1] 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.
[2] P. Russom, "Managing big data," *TDWI Best Pract. Report, TDWI Res.*, pp. 1–40, 2013.
[3] A. Gadkari, V. B. Nikam, and B. B. Meshram, "Implementing joins over HBase on cloud platform," in *Proc. IEEE Int. Conf. Comput. Inf. Technol.*, 2014, pp. 547–554.
[4] Y. Li and S. Manoharan, "A performance comparison of SQL and NoSQL databases," in *Proc. IEEE Pacific RIM Conf. Commun. Comput. Signal Process.*, 2013, pp. 15–19.
[5] M. J. Mior, K. Salem, A. Aboulnaga, and R. Liu, "NoSE: Schema design for NoSQL applications," *IEEE Trans. Knowl. Data Eng.*, vol. 29, no. 10, pp. 2275–2289, Oct. 2016.
[6] J. Bhogal and I. Choksi, "Handling big data using NoSQL," in *Proc. 29th Int. Conf. Adv. Inf. Netw. Appl. Workshops*, 2015, pp. 393–398.
[7] J. Han, E. Haihong, G. Le, and J. Du, "Survey on NoSQL database," in *Proc. 6th Int. Conf. Pervasive Comput. Appl.*, 2011, pp. 363–366.

[8] L. Rocha, F. Vale, E. Cirilo, D. Barbosa, and F. Mouroa, "A framework for migrating relational datasets to NoSQL," in *Proc. Int. Conf. Comput. Sci.*, vol. 51, 2015, pp. 2593–2602.
[9] G. Zhao, Q. Lin, L. Li, and Z. Li, "Schema conversion model of SQL database to NoSQL," in *Proc. 9th Int. Conf. P2P Parallel Grid Cloud Internet Comput.*, 2014, pp. 355–362.
[10] Sutedi, T. B. Adji, and N. A. Setiawan, "Enhanced graph transforming algorithm to solve transitive dependency between vertices," in *Proc. IEEE Int. Conf. Signals Syst.*, 2017, pp. 250–255.
[11] G. Zhao, L. Li, Z. Li, and Q. Lin, "Multiple nested schema of HBase for migration from SQL," in *Proc. 9th Int. Conf. P2P Parallel Grid Cloud Internet Comput.*, 2014, pp. 338–343.
[12] C. H. Lee and Y. L. Zheng, "SQL-to-NoSQL schema denormalization and migration: A study on content management systems," in *Proc. IEEE Int. Conf. Syst. Man Cybern.*, 2015, pp. 2022–2026.
[13] C. H. Lee and Y. L. Zheng, "Automatic SQL-to-NoSQL schema transformation over the MySQL and HBase databases," in *Proc. Int. Conf. Consum. Electron.*, 2015, pp. 426–427.
[14] M. Potey, M. Digrase, G. Deshmukh, and M. Nerkar, "Database migration from structured database to non- structured database," in *Proc. Int. Conf. Recent Trends Adv. Eng. Technol.*, 2015, pp. 1–3.
[15] V. Seshagiri, M. L. Vadaga, J. J. Shah, and P. Karunakaran, "Data migration methodology from SQL to column oriented databases (HBase)," *Int. J. Adv. Res. Comput. Eng. Technol.*, vol. 5, no. 11, pp. 2631–2635, 2016.

Sutedi received the master's degree (M.T.I), in information technology from Universitas Indonesia, in 2009. He is a doctoral student in the Department of Electrical and Information Engineering, Universitas Gadjah Mada. His research interests include software engineering, data warehouse, data mining, and big data.

Noor Akhmad Setiawan received the bachelor's and master's degree in electrical engineering from Universitas Gadjah Mada, in 1998 and 2003, respectively, and the PhD degree in electrical and electronics engineering from Universiti Teknologi Petronas, in 2009. He is with the Department of Electrical and Information Engineering Universitas Gadjah Mada. His research interests include soft computing and machine learning.

Teguh Bharata Adji received the PhD degree from the Department of Computer and Information Science, Universiti Teknologi Petronas, Perak, Malaysia, in 2010. He is an associate professor in Electrical and Information Engineering at Universitas Gadjah Mada. His reseach interests include natural language processing, computational linguistic, artificial intelligence, big data, data warehouse, data mining, and S/W testing & reliability.

Enhanced Graph Transforming V2 Algorithm for Non-Simple Graph in Big Data Pre-Processing

ORIGINALITY REPORT

4%	2%	6%	2%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Monash University Student Paper	2%
2	Sutedi, Teguh Bharata Adji, Noor Akhmad Setiawan. "Enhanced Graph Transforming Algorithm to solve transitive dependency between vertices", 2017 International Conference on Signals and Systems (ICSigSys), 2017 Publication	2%

Exclude quotes	On	Exclude matches	< 2%
Exclude bibliography	On		

Enhanced Graph Transforming V2 Algorithm for Non-Simple Graph in Big Data Pre-Processing

GRADEMARK REPORT

FINAL GRADE

/0

GENERAL COMMENTS

Instructor

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11