2022 9th International Conference on Advanced Informatics: Concepts, Theory and Applications (ICAICTA) | 978-1-6654-5668-5/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICAICTA56449.2022.9932986

Conversion of Entity-Relationship Model to NoSQL Document-Oriented Database Logical Model Using Workload Information and Entity Update Frequency

Ananda Yulizar Muhammad
School of Electrical Engineering and Informatics
Institut Teknologi Bandung
Bandung, Indonesia
13518088@std.stei.itb.ac.id

Fazat Nur Azizah
School of Electrical Engineering and Informatics
Institut Teknologi Bandung
Bandung, Indonesia
fazat@staff.stei.itb.ac.id

***Abstract*—Although dubbed schema less, NoSQL databases, especially the document-oriented ones, can still make use of effective data modeling techniques in order to provide high-performance databases. In this paper, we present a proposal of converting Entity-Relationship Model (ER Model) into NoSQL document-oriented database logical model based on a workload-driven logical modeling algorithm by Lima & Mello [6]. This work improves the algorithm by considering, not only database workload, but also entity update frequency. The workload information and the entity update frequency are used to determine whether to refer or to embed in the logical model. Embedding or referencing are two prominent "relationships" among collections in document-oriented databases. We tested the improved conversion algorithm into a case study and the performance of the resulting logical model was tested on MongoDB in comparison to the logical model resulting from the original algorithm. The performance tests show that the logical model resulting from the improved algorithm provides better performance in read operations, but worse performance in create and update operations in collections with embedded relationships.**

Keywords— ER Model, database, document-oriented, NoSQL, query, workload

I. INTRODUCTION

NoSQL databases were introduced to augment the features of traditional relational databases with new concepts such as schema flexibility, scalability, high performance, partition tolerance, and other new extended features [1]. It is generally recognized that NoSQL databases can be classified in four categories: the key-value stores, document-oriented databases, column-family stores, and graph databases, each suited to different kinds of task. In NoSQL document-oriented databases, a database contains a set of documents. A document is a structured and complex value of a set of attribute-value pairs, in which the value can be simple value, list, and even nested document. Documents are considered schema-less: each document can have its own attributes, defined at runtime [2].

The schema-less notion of NoSQL document-oriented databases often implies that it is not necessary to create a data model prior development of a database [3]. Nevertheless, it is believed that data modeling has an impact in providing better query performance, consistency, usability, software debugging and maintainability, and many other aspects [4], even in NoSQL databases. Several works attempt to provide data modeling frameworks for NoSQL databases, including the work by Rossel & Manna [3], Chebotko et.al. [5], and by Lima & Mello [6]. All emphasize the importance to start from conceptual modeling, using techniques such as Entity-Relationship Model (ER model), and transform it into proper logical model.

Lima & Mello [6] provides a workload-driven data modeling algorithm for NoSQL document-oriented databases based on ER model. The algorithm uses the estimation frequency of access by a query to an entity within a database (aka. the database workload) as a mechanism to decide whether to use embedding or referencing for the relationship among two document collections. As suggested in [1] and [3], the performance of the NoSQL databases may be influenced also by the rate of change of the entities involved. Therefore, in this paper, we propose an improvement of the workload-driven data modeling algorithm using not only database workload information, but also the update frequency of the entities.

II. RELATED WORKS

A. Data Modeling

Generally, data modeling is the first step in designing a database. It refers to the process of creating a specific data model for a determined problem domain. A data model is a simple representation of real-world data structures. Within a database environment, a data model represents data structures and their characteristics, relations, constraints, transformations, and other constructs with the purpose of supporting a specific problem domain [7]. According to [8], there are three levels in data models: conceptual, logical, and physical.

B. Entity Relationship Modeling

Entity Relationship Modeling (ER modeling) is conceptual data modeling technique in which data modelers provides the representation of the problem domain in terms of entities and their relationship. ER model is graphically represented by ER diagram (ERD) [7]. ER model is created to facilitate designing database through representing the logical structure of a database. Three main concepts in an ER model are: entity sets, relationship sets, and attributes [8]. Refer to [8] for detailed description of the technique.

C. NoSQL Databases

NoSQL databases are new emerging database technologies which differ from the traditional relational database in the structures and scaling capabilities [9]. In relational databases, the structure/schema of a database must be properly created before the database can be operated. On the other hand, NoSQL databases are notorious for more flexible schema that can accommodate changes and new data

types. Nevertheless, although NoSQL database schema can be changed dynamically, we need to take also a full consideration of the impact of the changes to the applications related to the database which may introduce more complications and costs.

1) NoSQL Document-Oriented Database

In NoSQL document-oriented database, a database stores information in the form of a collection of documents. A document contains semi-structured data in the form of a set of attribute-value pairs. Each attribute can have a different data type on each document [11]. The documents do not necessary to have a fixed schema. Therefore, each document can have different attributes defined at runtime [2].

2) Data Modeling for NoSQL Document-Oriented

NoSQL brings new features compared to relational database such as flexible schema, scalability, high performance, can be distributed, and other features [1]. The fact that the data structure does not need to be defined in advance has many advantages for exploratory development, but as the database grows bigger and used by applications, there will be a need to organize the database. Therefore, it is necessary to do a data modeling to define the structure of the database [3].

Rossel & Manna [3] defines a methodology to convert ERD into NoSQL document-oriented logical model called the Document Interrelation Diagram (DID). Their approach takes a conceptual model in ERD and query pattern as an input. The query patterns contain list of entities and attributes that will be need in a query.

Similarly, Chebotko et. al. [5] defines a data modeling modeling approach starting from a conceptual model and an application workflow to provide a logical model for NoSQL column-oriented database. The application workflow describes the query that the application requires.

3) Workload-Driven Logical Design

In [6], Lima & Mello propose an algorithm for modeling logical model of NoSQL document-oriented database that starts with a conceptual schema and workload information. The workload is estimated based on the conceptual schema, which are then used to convert the model into a logical model. The conversion to the logical schema is conducted through several algorithms they proposed.

The database workload information is required for the conversion to logical model process. There are three concepts of workload information defined in [6]: the volume of data, the application load, and the general access frequency.

a) Volume of Data

Given an EER schema Ɛ, the volume of data of Ɛ is defined by $V = \{N(t), Avg(\tau,r)\}$ with $N(t)$ is the average number of occurrences of a conceptual type t, and given a n-tuple $\tau = <t1,...,tn>$ of entity types associated through a relationship type r, $Avg(\tau,r)$ is the average of the cardinality of entities related through r. Fig. 1 shows an example of an EER with the data volume information. The numbers within the entity and relationship types depict N (the number of occurrences of the entity/relationship type), and the numbers above the symbols *avg* are the average cardinality of each side of relationship types.

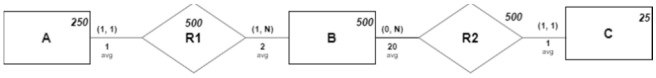

Fig. 1. ERD Schema with Volume Data [6]

b) Application Load

The application load on an EER schema Ɛ is defined by a set of operations. An operation is an interaction with the database. An operation is composed by two functions: (i) $f(o_i)$, which is the average access frequency in operation o_i on a given period of time, and (ii) $v(o_i,t_j)$, which is the volume of the instance t_j that are accessed in operation o_i. Table 1 shows an example of application load from the schema in Fig. 1.

TABLE I. EXAMPLE OF APPLICATION LOAD

Operation	Frequency per day	Concept Accessed	Access Volume
O1	900	C	900
		R2	18000
		B	18000
O2	300	A	300
		R1	600
		B	600

c) General Access Frequency

General Access Frequency (GAF) is the total access frequency of an instance t from all operations to the database. GAF of an instance t is defined as follows:

$$GAF(t) = \sum_{i=0}^{n} v(o_i, t) \quad (1)$$

To evaluate GAF, Minimal Access Frequency (MAF) is used as the threshold of which an access frequency to be considered significant. An access frequency value that is less than MAF are considered to be insignificant. MAF is a percentage of the total GAF of all instances and is provided by the database designer.

III. THE IMPROVED ALGORITHM FOR CONVERSION FROM ER MODEL TO DOCUMENT-ORIENTED LOGICAL SCHEMA

The conversion process consists of two main algorithms: the algorithm to convert generalization types and algorithm to convert the relationship types. The details of the algorithm can be seen in [6]. Fig. 2 shows an example of the logical model used in [6]. In this research, we adopt the same logical model representation for document-oriented logical model.

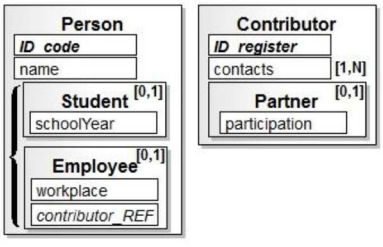

Fig. 2. Example of NoSQL document-oriented Logical Schema [6]

The algorithm by Lima & Mello uses the estimation frequency of access by a query to an entity within a database (aka. the database workload) as a mechanism to decide whether to use embedding or referencing to translate the relationship among two document collections [6]. As suggested in [1] and [3], the performance of the NoSQL databases may be influenced also by the rate of change of the

entities involved. In this section, we present our proposal to improve the workload-driven data modeling algorithm using not only database workload information, but also the update frequency of the entities.

A. Adding Functions to Operation

It was stated that operation is composed of two functions, which are $f(o_i)$ and $v(o_i,t_j)$. To accommodate the improved algorithm to take the entity's update frequency into consideration, we add 2 new functions: type of operation and involved attributes.

1) Type of Operation ($i(o_i)$)

$i(o_i)$ returns the type of operation o_i. The type of operation can either be create, read, update, or delete.

2) Involved Attributes ($a(o_i,t_j)$)

$a(o_i,t_j)$) returns the attributes of t_j that are involved in the operation o_i.

B. Update Access Frequency (UAF)

Because the decision to use embedding or referencing to define a relationship in document-oriented database is influenced by the rate of change of an entity, the algorithm is equipped with the information of the rate of change. A new concept derived from GAF, called the Update Access Frequency (UAF) which is the total access frequency of an entity only for the update operation, is added to the algorithm.

C. Modification of Algorithm

The original algorithm is modified in order to take entity update frequency into consideration.

1) Primitive Function

The following primitive functions are used within the new modified algorithm.

a) GenerateBlock(E, A, L)

The function GenerateBlock takes an input of entity E dan a set of attribut A as well as a set of root blocks L. The function makes a root block b_E that contains attribute A in L. The function returns the block b_E. If E is a weak entity, the function will not make the block as a root block and will only return the block

b) EmbedBlockIn(b1, b2, L)

The function embedBlockIn takes an input of block b_1 and block b_2 and a set of root blocks L. The function writes block b_1 as a nested block in block b_2 in a set of root block L.

c) ReferenceBlockFrom(b1, b2, L)

The function referenceBlockFrom takes an input of block b_1 and block b_2 and a set of root blocks L. The function writes a reference attribute in block b_2 that points to block b_1 in a set of root block L.

2) EER-NoSQL Algorithm Modification

The EER-NoSQL algorithm is modified in order to accommodate the new modifications as the following:

EER-NoSQLModified Algorithm

```
Input   : (EER Schema with load data information) Ɛ;
          (Minimal Access Frequency) MAF;
Output  : null;

U ← listUpdatedAttributes(Ɛ);
H ← convertHierarchies(Ɛ, MAF);
L ← convertRelationshipModified(Ɛ, MAF, H, U);
NF ← listOfCollections(L);
defineRootBlockIDs(NF);
```

3) listUpdatedAttributes Algorithm

The following listUpdatedAttributes algorithm is added to list all of the attributes that is involved in an update operation. This algorithm takes an EER schema and returns a set of attributes.

listUpdatedAttributes Algorithm

```
Input   : An EER Schema with load data information Ɛ;
Output  : A set of attributes U;

U ← {};
O ← List of operations of Ɛ;
for each oi ∈ O (1 ⩽ i ⩽ n) do
        if (i(oi) = 'update') then
                for each tj ∈ oi (1 ⩽ i ⩽ n) do
                        U ← U ∪ a(oi, tj);
                end for
        end if;
end for;
return U;
```

4) ConvertRelationships Algorithm Modification

The following convertRelationships algorithm is modified to take entity update frequency into consideration in its rules. convertRelationshipsModified takes an extra input which is a set of attributes U from the listUpdatedAttributes algorithm. The algorithm is also modified so that the entities involved in a relationship is always sorted ascending by GAF, and each rule condition is simplified to a 1-to-1 binary relationship for rule 4, 1-to-many binary relationship for rule 5, and the rest is for rule 6 to focus on its cardinality, as explained below.

convertRelationshipsModified Algorithm

```
Input   :   (EER Schema with load data information) Ɛ;
            (Minimal Access Frequency of Ɛ) (MAF);
            (A set of blocks) H;
            (A set of attributes) U;
Output  :   (A set of root blocks) L;

L ← {} ∩ H;
R ← the list of relationship types of Ɛ;
R' ← sort R ascending by GAF;
for each ri ∈ R' do
        ES ← the set of entities related by ri;
        ES' ← Sort ES ascending by GAF;
        if ((ri is 1:1) AND (ri is binary)) then
                applyRule4Modified(ri, ES', U, L);
        else if ((ri is 1:N) AND (ri is binary))then
                applyRule5Modified(ri, ES', U, L);
        else
                applyRule6Modified(ri, ES', U, L);
        end if;
end for;
return L;
```

5) applyRule4Modified Algorithm

The rules from [6] are written in the form of algorithms. The applyRule4Modified algorithm is a 1-to-1 binary relationship. Embedding is done to the the lower GAF entity which does not have a higher UAF than MAF. However, if they do, the algorithm creates a reference to the entity from the higher GAF entity. If there is a weak entity involved, but that weak entity is not weak to the other entity in this relationship, the weak entity is treated as the higher GAF entity to avoid embedding the entity separately from its strong entity.

applyRule4Modified Algorithm

```
Input   :   (relationship type) r;
            (set of entities) ES;
            (set of attributes that are updated) X;
            (set of root blocks) L;
Output  :   null;

If ( ∃ Ei ∈ ES: Ei is a weak entity with the highest GAF but not in r) then
        Etemp ← E1;
        E1 ← Ei;
        Ei ← Etemp;
end if;

bE1 ← generateBlock(E1, E1, L);
bE2 ← generateBlock(E2, E2 ∩ r, L);

if (X ∩ E2 ≠ {} AND UAF(E2) > MAF) then
        ReferenceBlockFrom(E2, E1, L);
else
        EmbedBlockIn(E2, E1, L);
end if;
return null;
```

6) applyRule5Modified Algorithm

The applyRule5Modified algorithm is for a 1-to-many binary relationship. If both entities have higher UAF than MAF, the algorithm creates reference to the lower GAF entity from the higher GAF entity. However, if only the lower GAF entity has a higher UAF than MAF, the higher entity will be embedded in the lower GAF entity. This option is done to minimize the need of using array of reference from the one side to the many side. If the lower GAF entity does not have a higher UAF than MAF, the higher GAF entity will embed the lower GAF entity. If there is a weak entity involved, but that weak entity is not weak to the other entity in this relationship, the weak entity will be treated as the higher GAF entity to avoid embedding that entity separately from its strong entity.

applyRule5Modified Algorithm

```
Input   :   (relationship type) r;
            (set of entities) ES;
            (set of attributes that are updated) X;
            (set of root blocks) L;
Output  :   null;

if ( ∃ Ei ∈ ES: Ei is a weak entity with the highest GAF but not in r) then
    Etemp ← E1;
    E1 ← Ei;
    Ei ← Etemp;
endif;
if (X ∩ E2 ≠ {}) then
    if (UAF(E2) > MAF) then
        if ((X ∩ E1 ≠ {} AND UAF(E1) > MAF) OR
            (E1 is a weak entity)) then
            bE1 ← generateBlock(E1, E1, L);
            bE2 ← generateBlock(E2, E2 ∩ r, L);
            referenceBlockFrom(bE2, bE1, L)
        else
            bE1 ← generateBlock(E1, E1 ∩ r, L);
            bE2 ← generateBlock(E2, E2, L);
            embedBlockIn(bE1, bE2, L);
        end if;
    else
        bE1 ← generateBlock(E1, E1, L);
        bE2 ← generateBlock(E2, E2 ∩ r, L);
        EmbedBlockIn(bE2, bE1, L);
    end if;
else
    bE1 ← generateBlock(E1, E1, L);
    bE2 ← generateBlock(E2, E2 ∩ r, L);
    EmbedBlockIn(bE2, bE1, L);
end if;
return null;
```

7) applyRule6Modified Algorithm

The applyRule6Modified algorithm is for relationships other than 1-to-1 and 1-to-many, such as many-to-many and ternary relationships. Entities other than the highest GAF entity are embedded in the highest GAF entity if their UAF is lower than MAF. If their UAF is higher than MAF, they will be referenced from the highest Entity. If there is a weak entity involved, but that weak entity is not weak to the other entity in this relationship, the weak entity will be treated as the higher GAF entity.

applyRule6Modified Algorithm

```
Input   :   (relationship type) r;
            (set of entities) ES;
            (set of attributes that are updated) X;
            (set of root blocks) L;
Output  :   null;

if ( ∃ Ei ∈ ES: Ei is a weak entity with the highest GAF but not in r) then
        Etemp ← E1;
        E1 ← Ei;
        Ei ← Etemp;
end if;

bE1 ← generateBlock(E1, E1, L);
for each Ei ∈ ES (1 ⩽ i ⩽ n)  do
        bEi ← generateBlock(Ei, Ei ∩ r, L);
        if (X ∩ Ei ≠ {} AND UAF(Ei) > MAF) then
                referenceBlockFrom(bEi, bE1, L);
        else
                embedBlockIn(bEi, bE1, L);
        end if;
end for;
return null;
```

IV. TESTING

We test the improved conversion algorithm into a case study and the performance of the resulting logical model was tested on MongoDB in comparison to the logical model resulting from the original algorithm. Testing is done to evaluate the performance of the improved algorithm through its data model result. Testing are carried out with modeling entity relationship diagram with its estimated workload information test case to a database for each test case. Each test case consists of an EER schema, list of operations, and GAF of each type. Each test case is assumed to have a 1.15% MAF. The execution time for each operation accumulatively and individually is compared for all resulting logical models. Accumulative operation means that the operation that is run according to the frequency of the operation. Each operation is run five times to avoid an outlier in the execution time. The

testing is run on MongoDB accessed through NodeJS with the library Mongoose.

The test case 1 consists of a conceptual schema about an -e-commerce application from [6] that is modified to have an update operation. EER schema of test case 1 can be seen on Fig. 3. Table 2 shows the list of operation for test case 1. The types accessed from the list of operations will be omitted. Table 3 shows the GAF and UAF of test case 1. The MAF of test case 1 is 466.

Table 4 shows the accumulative comparison of the operation, while Table 5 shows the individual comparison in millisecond.

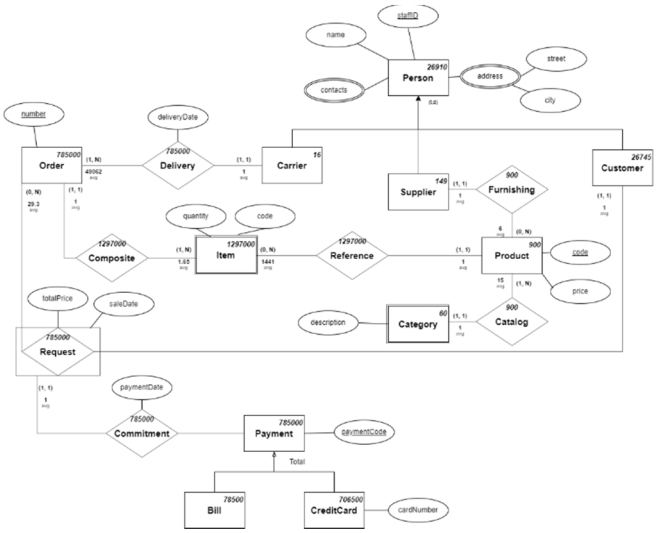

Fig. 3. EER Schema of Test Case 1

TABLE II. LIST OF OPERATIONS IN TEST CASE 1

Operation	Frequency per Day	Type of Operation
O1	1500	Create
O2	1500	Create
O3	450	Create
O4	100	Read
O5	1000	Read
O6	300	Update
O7	2000	Update

TABLE III. LIST OF GAF AND UAF IN TEST CASE 1

Concept	GAF	UAF
Order	6450	2000
Customer	4750	0
Product	4725	0
Request	4450	0
Composite	4125	0
Reference	4125	0
Item	4125	0
Payment	2100	300
Carrier	1450	0
Delivery	1450	0
Catalog	600	0
Furnishing	600	0
Category	600	0
Supplier	100	0

Fig. 4 shows the data model from original algorithm while Fig. 5 shows the data model from the improved algorithm.

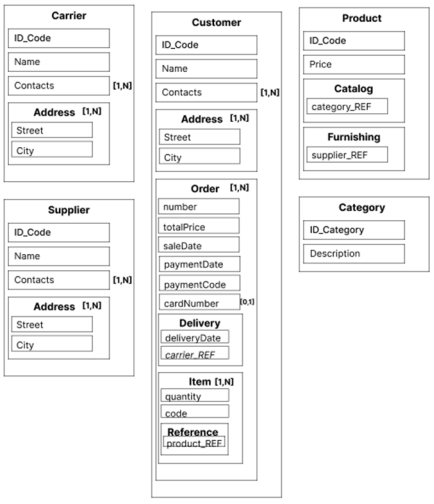

Fig. 4. Data Modeling Result of Lima & Mello Algorithm for Test Case 1

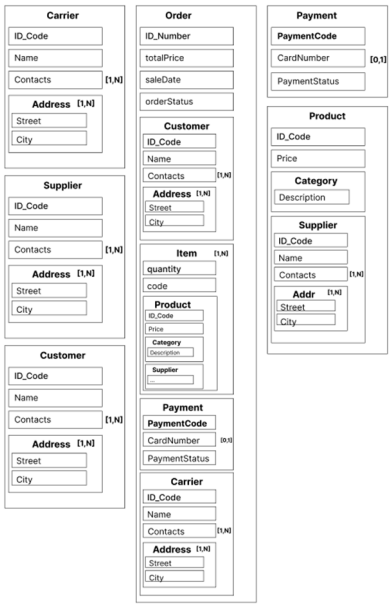

Fig. 5. Data Modeling Result of the Modified Algorithm for Test Case 1

TABLE IV. ACCUMULATIVE EXECUTION TIME COMPARISON FOR TEST CASE 1

Operation	Original Algorithm	Improved Algorithm
O1	1653	2845
O2	2260	3394
O3	590	761
O4	378	141
O5	3786	2031
O6	390	470
O7	3306	5475
Total	**10710**	**12272**

TABLE V. INDIVIDUAL EXECUTION TIME COMPARISON FOR TEST CASE 1

Operation	Original Algorithm	Improved Algorithm
O1	17	23
O2	4	4
O3	5	4
O4	8	4
O5	11	4
O6	3	3
O7	3	3
Total	**51**	**45**

From the testing results, we can see that the improved algorithm has better performance on operation O4 and O5 but has worse performance on operation O1, O2, O3, O6, and O7. This is because O4 and O5 is a read operation that involves reference in the original algorithm while the improved algorithm only used embedding. The reference made extra time to find the reference in another collection and that causes a longer execution time. Meanwhile, because of the embedding, the volume of data stored in the database is bigger and it causes a longer execution time for create/insert and update since it has to traverse the data in the collection to find the document to update. Therefore, the improved algorithm has better performance on read operation that involves reference, while it has a worse performance in create and update operations.

V. CONCLUSIONS AND FUTURE WORKS

Based on the testing results, we can conclude that the improved algorithm has better performance on read operation that has reference in it, although it has worse performance on create and update operation and the overall execution time because of the large volume of data. For future work, using partial embedding instead of fully embedding a nested document may improve the performance because it decreases the volume of data. We also need to specify a method to define the MAF percentage effectively.

REFERENCES

[1] A. A. Imam, S. Basri, R. Ahmad, J. Watada, M. T. Gonzlez-Aparicio dan M. A. Almomani, “Data Modeling Guidelines for NoSQL,” (IJACSA) International Journal of Advanced Computer Science and Applications,, 2018.

[2] P. Atzeni, F. Bugiotti, L. Cabibbo dan R. Torlone, “Data Modeling in the NoSQL World,” Computer Standards and Interfaces, Elsevier, 2020, 67, pp. 103-149, 2017.

[3] G. Rossel dan A. Manna, “A Big Data Modeling Methodology for NoSQL Document Databases,” Database Systems Journal, vol. XI/2020, 2020.

[4] P. Gómez, R. Casallas dan C. Roncancio, “Data schema does matter, even in NoSQL Systems,” 2016 IEEE Tenth International Conference on Research Challenges in Information Science (RCIS), pp. 1-6, 2016.

[5] A. Chebotko, A. Kashlev dan S. Lu, “A Big Data Modeling Methodology for Apache Cassandra,” 2015.

[6] C. d. Lima and R. d. S. Mello, "A Workload-Driven Logical Design Approach," pp. 1-10, 2015.

[7] P. Rob and C. Coronel, Database Systems: Design, Implementation, and Management (8th ed.), Massachusetts: Course Technology, Inc., 2007.

[8] A. Silberschatz, H. F. Korth dan S. Sudarshan, Database System Concepts (6th ed.), New York: Mcgraw-Hill, 2011.

[9] R. T. Mason, "NoSQL Databases and Data Modeling Techniques," Proceedings of Informing Science & IT Education Conference (InSITE) 2015, pp. 259-268, 2015.

[10] A. B. M. Moniruzzaman dan S. A. Hossain, “NoSQL Database: New Era of Databases for Big data Analytics -,” International Journal of Database Theory and Application, 2013.

[11] B. Sethi, S. Mishra dan P. K. Patnaik, “A Study of NoSQL Database,” International Journal of Engineering Research and Technology, 2014.