Advances in Engineering Research, volume 198
International Seminar of Science and Applied Technology (ISSAT 2020)

Experimental Evaluation of Relationships Model Between Documents in MongoDB

Ade Hodijah[1,*] Urip Teguh Setijohatmo[1]

[1]*School of Information Engineering, Politeknik Negeri Bandung, West Java40559, Indonesia*
[*]*Corresponding author. Email: adehodijah@jtk.polban.ac.id*

ABSTRACT
MongoDB is a document-oriented database technology that supports unstructured data processing. In general, there are two approaches to modeling data relationships between entities in MongoDB, namely the Embedded and Normalized data models. This study analyzes the data processing time performance of both models and explores other data modeling possibilities. The performance analysis process was carried out by testing each type of data model using a sufficient dataset as a scientific study of the simple Q&A application representation. The test results show that the Normalized model produces the longest data processing time of all test scenarios, namely almost 27000 times longer in retrieving data than other models with the number of processed lines by 100 K dataset. This condition occurs because of the growing array of documents_ID as a reference attribute. Another model is needed that supports the ease of processing data retrieval from various possible criteria besides using the Embedded model, which is the parent document model that is seen as having relationships between entities without any further search criteria. The mechanism for making relations from the Embedded model is by embedding an object from the complete child document. This study proposes the development of the Normalized data model, which is supporting the search criteria as references attributees between documents, which is called the Redundant data model.

Keywords: *MongoDB, Embedded data model, Normalized data model, Redundant data model*

1. INTRODUCTION

A table in Relational Database Management System (RDBMS) is known as a collection in MongoDB, a row as a document, and a field as a column. In RDBMS, a row of record or row is a grouping of columns within the same structure in a table [1], but a record in MongoDB is a document consisting of field and value pairs [2] as loosely structured. The field can be structured in a nested document or array with no more than 100 levels of nesting [2].

The types of relationships between tables or mapping cardinality in RDBMS can be constructed in one-to-one, one-to-many, and many-to-many [1] with relations in RDBMS using primary or foreign keys. Whereas in MongoDB, there are two approaches to forming relationships between documents based on data model design, namely the Embedded and the Normalized models, with its cardinality mapping, namely one-to-one relationships with Embedded documents, one-to-many relationships with Embedded documents, and one-to-many relationships with Normalized document [2].

The Embedded data model is generally known as the denormalized model that stores related data in the same record. The Normalized data model describes the relationships using references between documents like primary key value associations between tables in a relational database.

The experimental analysis in this research examined the structure of the MongoDB data model, using test data obtained by a simple application to generate documents as a representation of the answer and question datasets of the knowledge-sharing system [6]. Some of the problems in this research are: (1) What is the mechanism for accessing Question list shared by Person in the knowledge-sharing system; (2) What considerations are used as the basis for modeling relationships between documents in MongoDB; and (3) How is the schema of the documents of built-in MongoDB.

1.1. Related Work

In [3] tested ten non-relational database technology known as NoSQL databases of reading, update, and mixed read/update operation using the dataset of Yahoo! Cloud Serving Benchmark (YCSB). MongoDB presented a better performance in read-operation than update and mixed read/update operation. The evaluation

discussed memory and disk patterns analysis. The MongoDB stores databases In-Memory Database (IMDB) that processing large volumes of data fundamentally by processors manner based on the 64-bit architecture since volatile memory is used to map records so the execution time performance increases due to the use of locking mechanisms to perform update operations.

[4] tested SQL and NoSQL databases of query process operation in a single environment. There are two measurement results evaluated by four types of scenarios and three types of queries. The scenario consists of 1 million records, 10 million records, 25 million records, and 50 million records. The query consists of key (ID)value, WHERE clause data set, and aggregation function of SUM().The results revealed that a hybrid model which combines both the SQL databases to maintain greater consistency and NoSQL databases to provide immediate queries performance.

1.2. Our Contribution

[3, 4, 5] studied performance comparison for reading and update operations of NoSQL databases without explaining how the structure of the database schema is constructed, while execution time with respect to the schemes in Embedded and Referenced documents has been analyzed in [11] without conducting further analysis when combining the Embedded and Referenced models known as Redundant model. This paper presents experimental results to reveal how efficient is the run time of query using Embedded, Normalized, and Redundant data model.

1.3. Paper Structure

The following figure shows the stages of this research in detail.

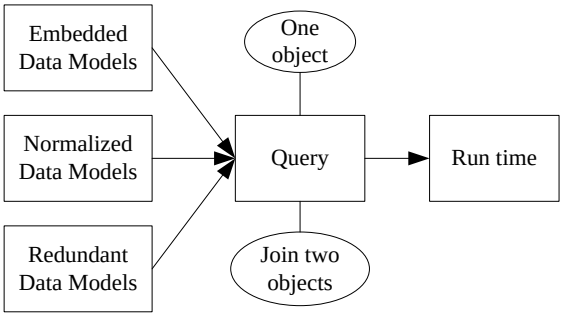

Figure 1 Research methodology

2. RESULT

MongoDB uses collections to store documents in the BSON (Binary JSON) format. The collection in this research is Person and Question with relationships as follows.

– Each Person can own zero or more than one Question

– Each Question can only be owned strictly by one Person

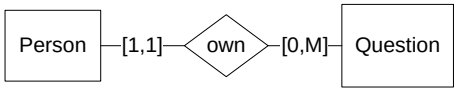

Figure 2 The relation of Person and Question

There are two types of searching to retrieve both Question and Person data: (1) Query is searching Question by criteria of Person_Name;(2) Query is searching Question by criteria of Question_Subject or Question_AcademicLevel. The data model design to support the first and second query is explained as follows.

A. The Embedded Data Model

This model connects Person and Question by storing Question object in Person collection as a parent document to provide speed to find out the Question owned by a certain Person_Name. However, this model is less efficient when searching Question by criteria of Question_Subject or Question_AcademicLevel because the process has to read Person documents, then getting a Question object list and check whether the attribute of Question_Subject or Question_AcademicLevel matches with the searching criteria. The Embedded model eases to get all Question data from each Person at once in one data reading process. However, this mechanism can reduce server speed, especially when the size of a particular attribute of a Question or Person object is large, such as video or image data. The greater the data size of an attribute, the more it will affect the performance of the server to respond to the data load process for the first time when the application starts. When the size of a document is getting bigger, the server performance degradation may occur in applications using the Embedded model to store data, either using a cardinality mapping type, which is one-to-one relationships or one-to-many relationships [2]. Therefore, it must be ensured that the attributes of a Question object embedded in the Person document are the ones that are required for obtaining information from the searching process. The logical form approach of the document-oriented model [9], which is used as the basis for designing the data structure in this study, can be seen in Figure 3.

```
public class Question {
    private String ID;
    boolean status;
    private String category;
    private String grade;
    private String question;
    private int report;
    private String time;

   public class Person {
       private String ID;
       private String username;
       private String password;
       private String name;
       private String academicLevel;
       private int point;
       private List<Question>
       embeddedQuestion = new ArrayList<>();
```

embed Question object

Figure 3 Declaration of Person and Question Collection in theEmbedded data model

B. The Normalized Data Model

This model splits the collection of Embedded data models into more than one collection. In this research, it connects Person and Question by creating a list that refers to the ID or primary key of Question in Person document. Each Person and Question is stored in different collections. It provides speed to find only a Question list or only a Person list without any criteria that connect each other. However, this model is less efficient when searching Question by criteria of Person_Name because it has to traverse the collection of Question ID list owned by a certain Person in Person collection then getting a complete Question object in Question collection. If the number of one-to-many relationships with document references that Question per Person is small with limited growth, storing the Question ID reference inside the Person document may sometimes be useful, but this data model will lead to mutable if the array of Question ID list is unbounded growing [2]. The logical document-oriented model [9] of this model is declared in the programming approach, as seen in Figure 4.

```
public class Person {
    private String ID;
    private String username;
    private String password;
    private String name;
    private String academicLevel;
    private int point;
    private List<String>
    referenceQuestionID = new ArrayList<>();
```

refers to Question object

```
public class Question {
    private String ID;
    boolean status;
    private String category;
    private String grade;
    private String question;
    private int report;
    private String time;
```

Figure 4 Declaration of Person and Question Collection in the Normalized data model

A. The Redundant Data Model

There is a need for better query performance from the data search process, scenario 1 and scenario 2, so another data model that can support these query needs is called the Redundant data model. There are two collections in this model. One collection is Person collection in Normalized data model without containing reference Question ID attribute. Another is Question collection in Normalized data model with reference attributes are all criteria of the query except the Question object (Question_Subject) that is Person_Name or Person_AcademicLevel and adding a Person ID attribute as a foreign key which refers to a Person collection. The redundant data model idea is adopted by the Multiple Table Inheritance approach in a relational data model [7]. However, not all structures of documents are in the normalized form where unstructured data are also stored in a non-relational data model [8]only criteria for searching. However, this model is less efficient in memory usage when the same Person creates a new Question, so Person data, which is attributes of the search criteria that refer to Person collection, is duplicated in the Question document as much as the number of Questions belongs to that Person (PersonID). When a new Person creates a new Question, the application also stores complete Person data in Person collection to get generated Person ID, which is a reference attribute stores in Question collection along with complete question data and other attributes of the search criteria. The logical document-oriented model [9] of this model is declared in the programming approach, as seen in Figure 5.

```
public class Question {
    private String ID;
    boolean status;
    private String category;
    private String grade;
    private String question;
    private int report;
    private String time;
    private String Person_ID;
    private String Person_Name;
    private String Person_AcademicLevel;
```

Redundant attributes of Person object

```
public class Person {
    private String ID;
    private String username;
    private String password;
    private String name;
    private String academicLevel;
    private int point;
```

Figure 5 Declaration of Person and Question Collection in Redundant data model

[10] studied the response time of the query in the tree Database Management System (MongoDB, Postgresql, Oracle) for a different number of processed lines in 100 K, 1M, 5M, and 10 M with 100 K dataset. Testing was carried out five times in a single server

environment PC with Intel (R) Core (TM) specifications i5-8250U CPU @ 1.60GHz 1.80 GHz uses 12.0 GB RAM, MS Windows 10. The experiment was carried out in two searching scenarios with the data set that are not generated comprehensively so that it just reached as sufficient data sources for scientific studies in knowing which data model design produces the best time performance.

Table 1. Scenario_1: experimental results for searching Question by criteria of Person_Name in milliseconds.

Model	1	2	3	4	5	Average
Embedded	323	339	325	309	322	**324**
Normalized	7514747	12285120	7662866	9887754	6329713	8736040
Redundant	383	341	364	349	388	365

Table 2. Scenario_2: experimental results for searching Question by criteria of Person_Name and Question_Subjectin milliseconds.

Model	1	2	3	4	5	Average
Embedded	289	280	289	317	308	**297**
Normalized	14274081	8333911	10364804	7126261	29155823	13850976
Redundant	401	413	379	359	406	392

The variation of testing above shows that the Embedded data model is better than the Redundant data model in Scenario_1 and Scenario_2 because it performs one query to retrieve Person and Question data, as seen in Figure 3.Meanwhile, the Normalized data model is the latest in all searching scenarios with the response time by3 hours 50 minutes 50 seconds, as seen in Table 2, that is the longest run time since it joins two documents to retrieve Person and Question data base on Question ID, as seen in Figure 4.

There are two possibilities in storing the Question ID list in Person document stored as an array. First, the response time result at about 3 hours was conducted by performing a query by selecting 100 K Questions of 1 Person. It means that there is only one record in the Person document, which consists of 100 K Question ID lists of the array. There is something to consider when using an Embedded data model since the maximum BSON document size is 16 MB. The GridFS API is used to store documents larger than the maximum size [2]. According to the maximum size, the data in this paper are generated twice of 50,000 with a total is 100 K Question ID list of array.

The other experimental results below are performed by searching Question by criteria of Person_Nameof 100 K records stored in Question document that refer to 100 K records of Question ID, so each record in Person document only has one Question ID. It designed in a Normalized data model, as seen in Figure 4. The first Normalized data model experiment shows an average result of 8736040 ms or about 2 hours 25 minutes 36 seconds, as seen in Table 1. The second experiment result shows an average of 8191163 ms or about 2 hours, 16 minutes, 31 seconds, as seen in Table 3.

Table 3. Scenario_3: experimental results of the Normalized data model implementation for joining two documents of 100 K Question with 100 K Personwhere the maximum list in each record in Person document consists of 1 string of Question ID in milliseconds.

Model	1	2	3	4	5	Average
Normalized	5663530	7576200	10494152	9035775	8186159	8191163

The other experiment was conducted for the Embedded data model to search Question by criteria of Person_Name, which has the same data structure to store Question ID list in Person document using an array of objects, as seen in Figure 3. The first Embedded data model experiment shows an average result of 324 ms, as seen in Table 1, while the second experiment shows an average result of1318ms or about 1 second, as seen in Table 4.

Table 4. Scenario_4: experimental results of the Embedded data model implementation for searching 100 K Question in 100 K Person where the maximum Question list in each record of Person document consists of one object in milliseconds.

Model	1	2	3	4	5	Average
Embedded	842	1753	1625	1536	833	1318

The other experiment was conducted for the Redundant data model to search Question by criteria of Person_Name, which has the same data structure to store Question ID list in Person document using an array of objects, as seen in Figure 5. The first Redundant data model experiment, as seen in Table 1, shows the average result by 365ms, while the second experiment shows the average result by752ms, as seen in Table 5.

Table 5. Scenario_5: experimental results of the Redundant data model implementation for searching 100 K Question in 100 K Person where the maximum Question list in each record of Person document consists of one object in milliseconds.

Model	1	2	3	4	5	Average
Redundant	905	1073	808	484	491	752

The response time of the Normalized data model (as seen in Table 3) and Embedded data model (as seen in Table 4) to search Question by criteria of Person_Name and Question_Subject in 100 K records at each Question and Person document is longer than the results shown in table 1 which consists of 100 K records of Question list of the array in one record of Person document.

Table 6. Scenario_6: Experimental results of the Normalized, the Embedded, and the Redundant data model comparisons for Scenario_A: 100 K Question list in 1 record of Person document, Scenario_B: 100 K Question in 100 K records of Person documents where maximum list in each record of Person document consists of 1 Question in milliseconds.

Model	Scenario _A	Scenario _B	Average	Slower	Faster
Embedded	324	1318	1:4	Table 4	Table 1
Normalized	8736040	8191163	1:1	Table 1	Table 3
Redundant	365	752	1:2	Table 5	Table 1

The response time of the Normalized, Embedded, and Redundant data model of searching Question by criteria of Person_Name and Question_Subject in 100 K records at each Question and Person document (scenario_A) is lower than searching Question by criteria Person_Name and Question_Subject (scenario_B). The fastest response time of scenario_A is the Embedded data model. Otherwise, the Redundant data model is faster than the Embedded data model in scenario_B, where the Redundant data model is 752 ms, and the Embedded data model is 1318 ms.

```
public class Question {
    private String ID;
    boolean status;
    private String category;
    private String grade;
    private String subject;
    private int report;
    private String time;
    private List<Person>
    embeddedPerson = new ArrayList<>();
}

public class Person {
    private String ID;
    private String username;
    private String password;
    private String name;
    private String academicLevel;
    private int point;
}
```

Redundant of Person in Embedded schema

Figure 6 Redundant data model in Embedded schema approach for the Person object

The results of searching experimental in scenario_1 and scenario_2 show that the Embedded data model is the fastest, but no in scenario_6 where the Redundant data model is the fastest for Scenario_B. Therefore, further testing was carried out by designing a Redundat data structure using an embedded data model approach [11], as seen in Figure 6.

Table 7. Scenario_7: experimental results of the Redundant data model implementation using the Embedded approach for 100 K Question in 100 K records of Person documents where maximum list in each record of Person document consists of 1 Question in milliseconds.

Model	1	2	3	4	5	Average
Redundat in Embedded	784	809	788	826	815	805

The response time of the Redundant data model using the Embedded approach [11], as seen in Table 7 shows that the average response time is 805 ms, so it is longer than the response time of the Redundant data model is 752 ms as seen in Table 6.

Table 8. Scenario_8: experimental results of the Redundant data model implementation using the Embedded approach and does not use the Embedded approach for 2 million of Question in 1 million records of Person documents where maximum list in each record of Person document consists of 1 Question in milliseconds.

Model	1	2	3	4	5	Average
Redundat	10002	7629	9650	8785	9899	9193
Redundat in Embedded	10629	9011	8036	9089	8489	9051

The response time of the Redundant data model using the Embedded approach in Table 8 is faster than the result in Table 7. The result of it is not consistent. Based on these results can be concluded that it does not matter to design the Redundant data model with or without using the Embedded approach, but it will provide two directions of the searching process if we designed in using the Embedded approach. First, searching the list of Questions based on the criteria of the point value of Person object which is embedded in the Question document, as seen in Figure 7. Second, searching the list of Questions based on the criteria of the subject value of the Question object which is embedded in the Person document as seen in Figure 8.

```
public class Question {
    private String ID;
    boolean status;
    private String category;
    private String grade;
    private String subject;
    private int report;
    private String time;
    private List<String>
    referencePersonID = new ArrayList<>();

public class Person {
    private String ID;
    private String username;
    private String password;
    private String name;
    private String academicLevel;
    private int point;
    private List<String>
    referenceQuestionID = new ArrayList<>();
```

Figure 7 Redundant data model in Embedded schema approach for Person and Question object with a point value as searching criteria

```
public class Question {
    private String ID;
    boolean status;
    private String category;
    private String grade;
    private String subject;
    private int report;
    private String time;
    private List<String>
    referencePersonID = new ArrayList<>();

public class Person {
    private String ID;
    private String username;
    private String password;
    private String name;
    private String academicLevel;
    private int point;
    private List<String>
    referenceQuestionID = new ArrayList<>();
```

Figure 8 Redundant data model in Embedded schema approach for Person and Question object with subject value as searching criteria

One last data structure of the Redundant data model can be designed in the Normalized data model, as seen in Figure 9. The experimental results of the Normalized data model are always the longest, among others. But, when the inserted data is predicted to grow, it is better to use the Normalized data model than the Embedded data model since the máximum BSON document size.

```
public class Question {
    private String ID;
    boolean status;
    private String category;
    private String grade;
    private String subject;
    private int report;
    private String time;
    private List<String>
    referencePersonID = new ArrayList<>();

public class Person {
    private String ID;
    private String username;
    private String password;
    private String name;
    private String academicLevel
    private int point;
    private List<String>
    referenceQuestionID = new ArrayList<>();
```

Figure 9 Redundant data model in Normalized schema approach for Person and Question object

Table 9. Experimental comparison for scenario A: searching Question by criteria of Person_Name (one object) and B: searching Question by criteria of Person_Name and Question_Subject (two objects) with dataset in C: 100 K Questions of 1 Person and D: 100 K Questions of 100 K Person.

Scenario	Dataset	The fastest	Experimental result
A	C	Embedded	Table 1
B	C	Embedded	Table 2
A	D	Redundant	Table 6
A	D	Redundant without using the Embedded approach	Table 5
A	D	Redundant with using the Embedded approach	Table 8

Table 9 shows that the Embedded data model is faster than the Redundant data model in 100 K Questions of 1 Person. But, when the dataset is spread in all records in Person document (100 K Questions of 100 K Person) and when the search process wants to be done in two directions, so the Redundant data model is the fastest with or without using the Embedded approach.

3. CONCLUSION

There are three possibilities to retrieve the question list asked by a certain Person or vice versa using a query based on data model design in MongoDB: (1) Does it search the Question in Person using an Embedded data model, or (2) Does it search the Person references to Question using Normalized data; or (3) Both of them use the Redundant data model. The variation of the dataset in the unstructured data model is required to get the best experimental results as a follow up of this research.

REFERENCES

[1] IBM

[2] MongoDB

[3] Abramova, V., Bernardino, J., Furtado, P.: Experimental evaluation of NoSQL databases, International Journal of Databases Management Systems (IJDMS), 2014, 6, (3). DOI: http://dx.doi.org/10.5121/ijdms.2014.6301

[4] Duran-Cazar, J.W., Tandazo-Gaona, E.J.: Performance of columnar database, In Scientific Paper INGENIUS, 2019. DOI: https://doi.org/10.17163/ings.n22.2019.05

[5] Cooper, B., Silberstein, A., Tam, E., Ramakrishnan, R., and Sears, R.: Benchmarking cloud serving systems with YCSB, In Proceedings of the 1st ACM Symposium on Cloud Computing (SoCC, 2010), pp. 143-154. DOI: https://doi.org/10.1145/1807128.1807152

[6] Brainly

[7] Maowa, J., SajedulHoque, A.H.M., Mustafa, R., Rahman, M.O.: A comparative study on big data handling using relational and non-relational data model, International Journal of Data Mining & Knowledge Management Process (IJDKP), 2017, 7, (3). SSRN: https://ssrn.com/abstract=3672094

[8] James, B.E., Asagba, P.O.: Hybrid database system for big data storage and management, International Journal of Computer Science, Engineering and Applications (IJCSEA), 2017, 7, (3/4). SSRN: https://ssrn.com/abstract=3544670

[9] Benmakhlouf, A.: NoSQL implementation of a conceptual data model: UML class diagram to document-oriented model, International Journal of Database Management Systems (IJDMS), 2018, 10, (2). DOI: http://dx.doi.org/10.5121/ijdms.2018.10201

[10] Hiyane, Y., Benmakhlouf, A., Marzouk, A.: Storing data in a document-oriented database and implemented from structured nesting logical model, International Journal of Database Management Systems (IJDMS), 2020, 12, (2). DOI: http://dx.doi.org/10.5121/ijdms.2020.12202

[11] Gomez, P., Casallas, R., Roncancio, C.: Data schema does matter, even in NoSQL systems!, IEEE Tenth International Conference on, 2016. https://hal.archives-ouvertes.fr/hal-01482250