software

MDPI

Article

A MongoDB Document Reconstruction Support System Using Natural Language Processing

Kohei Hamaji [1,*] and Yukikazu Nakamoto [2]

[1] Honda Motor Co., Ltd., Haga 321-3321, Tochigi, Japan
[2] Department of Information and Data Science, Nortre Dame Seishin University, Okayama 700-8516, Okayama, Japan; nakamoto@m.ndsu.ac.jp
* Correspondence: kohei_hamaji@jp.honda

Abstract: Document-oriented databases, a type of Not Only SQL (NoSQL) database, are gaining popularity owing to their flexibility in data handling and performance for large-scale data. MongoDB, a typical document-oriented database, is a database that stores data in the JSON format, where the upper field involves lower fields and fields with the same related parent. One feature of this document-oriented database is that data are dynamically stored in an arbitrary location without explicitly defining a schema in advance. This flexibility violates the above property and causes difficulties for application program readability and database maintenance. To address these issues, we propose a reconstruction support method for document structures in MongoDB. The method uses the strength of the Has-A relationship between the parent and child fields, as well as the similarity of field names in the MongoDB documents in natural language processing, to reconstruct the data structure in MongoDB. As a result, the method transforms the parent and child fields into more coherent data structures. We evaluated our methods using real-world data and demonstrated their effectiveness.

Keywords: MongoDB; natural language process; Has-A relationship; semantic distance

Citation: Hamaji, K.; Nakamoto, Y. A MongoDB Document Reconstruction Support System Using Natural Language Processing. *Software* **2024**, 2, 206–225. https://doi.org/10.3390/software3020010

Academic Editor: Alessio Ferrari

Received: 12 March 2024
Revised: 15 April 2024
Accepted: 24 April 2024
Published: 2 May 2024

1. Introduction

As the size, type, and generation rate of data that computer systems handle increase, the shortcomings of traditional data stores, such as performance degradation and inflexible data structures, become apparent for many reasons, including structural rigidity and poor responsiveness. Such data becomes larger in scale and more complex, resulting in the following phenomena: (1) The fixed-table schema of the existing relational database (RDB) cannot cope with complex data structures. (2) The cost of data retrieval and modification increases as the data structure becomes more complex and sufficient performance cannot be ensured. RDB assumes ACID characteristics that requires strong data consistency. A strict schema enforces rules about data structure and relationships to ensure data consistency and prevent inconsistencies. On the other hand, scalability is sacrificed in data storage and access. This leads to difficulties in processing large and diverse data sets. However, in new data-intensive applications, where performance and availability are paramount, efficient data management becomes increasingly important. Consequently, Not Only SQL (NoSQL) databases, which have a different data model from traditional RDBs, are widely used. In particular, document-oriented NoSQL databases have received attention owing to their flexibility, scalability, and high performance.

MongoDB is a document-oriented NoSQL database [1]. A document in a document-oriented NoSQL is equivalent to a record in an RDB and has the flexibility to dynamically change its structure, making it suitable for semi-structured and unstructured data. This simplifies data modeling for complex data structures. MongoDB is open-source software that is widely used by companies and startups across multiple industries. MongoDB also

features a schema that defines the document structure in advance. However, MongoDB can flexibly handle data without such a schema. It is frequently used to import schemaless data whose structure may be inconsistent. In such cases, the documents are highly flexible. Nevertheless, this flexibility makes it difficult to design and maintain an appropriate MongoDB document structure. For developers familiar with the RDB design, it is more difficult to design MongoDB documents effectively and fully use their functionalities because document design in MongoDB differs from legacy RDB schema design in many aspects.

The differences between the RDB and document data in MongoDB in terms of database design are as follows:

(1) In MongoDB, document design depends on how an application program accesses data, that is, the data usage patterns of a particular application. A developer must structure the data to match the manner in which the application queries and updates the data. Therefore, data modeling is frequently created not by the database manager but by application program developers who have limited skills in database design. Owing to the large size and complexity of the data being handled, the immaturity of the designer's data modeling skills and inadequate modeling guidelines tend to result in low-quality MongoDB document structures.
(2) In the case of a MongoDB database used by multiple applications, data design becomes more complex because all applications require sufficient performance and well-designed data structures that developers can easily share. This has resulted in an increase in inappropriate database modeling, design, and low-performance systems.

Based on the above observations, we believe that the following problems can occur when storing data in MongoDB's document structure:

P1: MongoDB is a database that stores data in the JSON format with a tree-like structure whose node is called a field in the MongoDB document. This tree structure implies that the upper field involves lower fields; in particular, the upper parent field is closely related to the lower child fields of the parent field. Moreover, fields with the same parent are also related. One feature of MongoDB is that schemaless data can be dynamically stored in an arbitrary location without explicitly defining a schema in advance. Therefore, if the data are placed in an arbitrary location, the close relationship between parent–child fields and fields at the same level in the document violates the above property. Consequently, the cost of searching and modifying the data increases because there is no clues to detect the target fields.
P2: In MongoDB, it is common for data to be stored in a single-level, flat format without a hierarchical structure. This is because data in the CSV format, which is widely used in open data, are imported directly into MongoDB.

Problem P1 increases the maintenance efforts for reusing MongoDB documents in the development of other application programs. P2 results in a longer access time because a program may access too much unrelated data when it reads certain data.

To solve these problems, this study proposes a method that improves document design in MongoDB by reconstructing it and a tool to support the method, focusing on the data model. We hypothesize that the parent–child relationship of fields and child fields under the parent field in the JSON data are closely associated. The proposed method reconstructs the data structure in MongoDB documents based on the distance between field names using a method that measures word distances in natural language processing based on the hypothesis. The authors have already shown the correctness of the hypothesis, that is, the relationship between a table name and the column names in the table, as well as that among column names in a table, in the RDB [2]. This study applies this hypothesis to data design in MongoDB. Although this study primarily presents the application of this method in the data design of MongoDB documents, it can also be applied to the design of MongoDB schemas.

The remainder of this paper is organized as follows: Section 2 describes the problems in the parent–child relationship of the field name and child fields under the parent field in

the JSON data, as well as a method to solve the problem of the violation of the relationships. Section 3 presents an overview of the proposed method. The specifications of this tool are presented in Section 4. In Section 5, we apply the tool to the JSON and CSV data to verify the hypotheses above and demonstrate its effectiveness. Section 6 describes the related studies. Section 7 provides the conclusions.

2. Requirement Specification

In this section, we describe problems with field names in MongoDB using examples. Subsequently, we present functions in a method to solve the problems.

2.1. Definition of Terms

MongoDB is a document-oriented database that stores data in the JSON format. Data are stored in collections, and each document is stored in the collection. Here, we next introduce the terms used in MongoDB [3].

MongoDB Data Modeling Introduction

Collection: A collection has one or more documents in MongoDB, which is equivalent to a table in the RDB.
Document: A document comprises of one or more fields. A document is equivalent to a row in a table.
Field: A field is a data element in a document consisting of a field name and values. A field is equivalent to a column in the RDB.
Nested document: A field can have a document in its field value in nested or embedded form. This is called a nested document and is equivalent to linking another table with foreign keys in an RDB. A nested document enables a hierarchical data representation using a tree structure.

Consider the following document (https://www.mongodb.com/docs/manual/tutorial/model-embedded-one-to-one-relationships-between-documents/ (accessed on 28 February 2024)).

```
{
   name: "Joe Baker",
   age: 20,
   address: {
              street: "123 Fake Street",
              city: "Faketon",
              state: "MA",
              zip: "12345"
            },
   phone: [ ‘‘090-1234-5678‘‘,’’090-5678-1234’’]
 }
```

This document consists of four fields: `name`, `age`, `address`, and `phone`. The `address` field is a nested document and the `phone` field has multiple values in the form of an array. These fields comprise MongoDB documents.

MongoDB also provides features to specify the MongoDB schema to define the field names of data and data types of the fields. The JSON schema defines where to place the fields, how to name the fields, and what is the data type of the fields.

2.2. Problems in MongoDB Document Design

As mentioned in Section 1, MongoDB's flexibility allows us to easily modify the document structure, and we consider that the following problems may easily occur:

(1) Improper placement of fields
In the hierarchical structure that stores data in MongoDB, data fields are placed inappropriately. This can occur in the following cases:

(a) A field hierarchy in a document tree is inappropriate. This occurs in the fields under collection and is stored within a given nested document. Among these fields, some have different meanings from other fields and have little relation to the fields in the upper hierarchy. Alternatively, the fields that should be placed in a nested document may not be appropriate. In such cases, it becomes difficult for a programmer to locate the data because it is not placed in the expected location. This decreases the program's readability and maintainability. The following is an example:

```
//shopdata
{
   "saleDate": "20220313",
   "item": {
         "name": "notepad",
         "tags": [ "office", "writing", "school" ],
         "price": { "$numberDecimal": "35.29" },
         "quantity": { "$numberInt": "2" },
         "customerName":"Smith"
    },
   "storeLocation": "Denver",
   "customer": {
      "gender": "M",
      "age": { "$numberInt": "42" },
      "email": "cauho@witwuta.sv",
      "satisfaction": { "$numberInt": "4" }
   },
   "couponUsed": true,
   "purchaseMethod": "Online"
}
```

A problem with this example is that if a programmer wants to obtain customer information, the programmer must access both the `customerName` and the `customer` subtrees in the nested document. To access these two sets of data, MongoDB requires the programmer to specify the location of the data to be accessed through the query APIs. The following is a query in which `customerName` is Smith's customer information:

```
customerdb.shops.find({"item.customerName":"Smith"},{"customer.gender":1,
  "customer .age": 1,"customer.email":1,"customer.satisfaction":1})
```

It is impossible to write such a program without knowing the rules of the document structure and where its related fields, such as customer information, are located. Such examples reduce the readability and maintainability of the data structures.

(b) When fields that are related to the parent field in one document but are not related to the fields at the same level in a nested document subtree exist, it becomes difficult to grasp the meaning of the data in the document and write the appropriate queries. In the following example, the fields related to the address of the restaurant and those related to the restaurant's rating are placed at the same level without a hierarchy:

```
//shopdata
{
 "shopName": "streetRestrant".
 "shopInfo"{
    "building": "8825",.
    "coord": [-73.8803827, 40.7643124],.
    "street": "Astoria Boulevard", "street".
    "zipcode": "11369","
    "borough": "Queens", "borough".
    "cuisine": "American", "American".
    "date": "2014-11-15T00:00:00.000Z", "date".
    "grade": "Z",.
    "score": 38
  }
}
```

The fields of `building`, `coord`, `street`, and `zipcode` are related to addresses, while `date`, `grade`, and `score` are related to rating groups. If these two groups are appropriately grouped and structured, the data structure can be improved.

(c)
```
//shopdata
{
   "saleDate": "20220313",
   "item":  {
         "name": "notepad",
         "tags": [ "office", "writing", "school" ],
         "price": { "$numberDecimal": "35.29" },
         "customerName":"Smith"
      },
   "storeLocation": "Denver",
   "customer": {
      "name": "Smith",
      "gender": "M",
      "age": { "$numberInt": "42" },
      "email": "cauho@witwuta.sv",
   },
}
```

In this example, we assumed that the `customerName` field in the `item` field and the `name` field in the `customer` field are identical. In this case, if a programmer wants to obtain `customer` information, the programmer must write a search query to obtain the `customerName` under the `item`, as well as issuing a query for the `customer`. The following search query must be specified as follows:

```
db.shopdata.find(
  {},
  {"item.customer.name": 1, "customer": 1}
)
```

(2) Data are stored in fields with only one level, and nested documents are not used. This can occur when data in the CSV format are imported directly into MongoDB. An example of this is as follows:

```
{
   "uniq_id": "34709b14bed89e484f902363f4878e55",
   "hotel_id": "5530755",
   "hotel_name": "Luxurious 1BR Dwelling in Kochi",
   "default_rank": "250",
   "price_rank": "12",
   "ota": "booking.com",
   "name": "Apartment",
   "price": 895.0,
   "occupancy": 2,
   "breakfast": "Room Only",
   "cancellation": "free cancellation",
   "checkin_date": "2020-03-26",
   "crawled_date": "2020-03-20 09:43:15 +0000"
}
```

The following query provides a `hotel_name` of the above document. A programmer must specify several other fields in order to obtain a specific description.

```
db.hotel.find("uniq_id": 0,"hotel_id": 0,"hotel_name": 1," default_rank ": 0,
" price_rank ": 0," ota ": 0,"name": 0,"price": 0," occupancy ": 0,
"breakfast": 0,"cancellation": 0," checkin_date": 0,"crawled_date":0)
```

In addition, access to the fields with such a flat data structure will be time-consuming when the number of fields is large because other unrelated fields must be read until the field is accessed.

2.3. Functions in a Proposed Solution Method

This section describes the functionality of the MongoDB document reconstruction support method for solving the aforementioned problems. The method enforces the assignment of a field name to a certain rule based on natural language processing while

retaining the flexibility of MongoDB document structure. In this study, in the MongoDB document tree, we define the parent and child field names as (i) the collection name and field names of documents belonging to the collection, respectively, or (ii) the name of a field having a nested document and field names in the nested document, respectively. The MongoDB document reconstruction support method is based on the following two hypotheses: First, in the MongoDB data in the JSON format, the hierarchical relationship between a parent field name and a child field name is closely related. Moreover, there are many cases of Has-A relationships in relationships, where a Has-A relationship is a relationship where one is a part of another. This is because the hierarchy of the parent and child fields in the JSON tree represents a link to tables in the RDB and columns in Table [4]. Second, fields under the same parent field are related and formed into one group. Otherwise, the fields belonging to multiple domains are expected to be mixed at the same level. We call the former a hierarchical relationship and the latter a horizontal relationship.

Based on these hypotheses, the proposed method evaluates the strength of the Has-A-relation relationships and the distances between field names in a document tree using natural language processing and presents the fields to be reconstructed.

We describe the necessary functions of this method as follows:

F1: To show child field names that are not closely related to their parent field name. Typically, child field names are related to the parent field name. This is because the parent and child fields in a nested document are used to construct a table. However, if child field names have little relationship with the parent name, these fields may be better placed in another nested document. This method supports the reconstruction of the data structure in MongoDB by extracting the fields to be reconstructed. Reconstruction improves the document's impedance. The extraction of fields is shown by presenting the clustering results based on the strength between a parent field name and its child field names.

F2: If many fields are located at the same level in a document tree, fields belonging to multiple domains are expected to be mixed at the same level. In this case, the method clusters the fields using the semantic distance of field names and proposes that the clustered fields belong to the same domain to form another nested document. This method also indicates the fields for reconstruction.

F3: Reconstruction by F1 and F2 changes the complexity of the data structure of a MongoDB data tree. As an index of complexity (coherence), we calculated the average semantic distance between each field name. This index was calculated before and after the field reconstruction to evaluate the modified document structure. If the modified document structure is inappropriate, the reconstruction may adversely affect the application program design. This tool visualizes the modified document structure and presents the differences in the complexity index.

The above reconstruction functions do not necessarily consider the circumstances and query performance of the application programs that access the data in MongoDB. Therefore, the decision to reconstruct is left to the user.

These functions detect phenomena and solve the problems described in Section 2.2. For example, in the case of inappropriate field placement, the parent–child relationship of the field in the document tree decreases and can be detected using the above functions.

Moreover, the two reconstructions above, F1 and F2, can find fields that violate the principle of one datum by one place, although this is executed indirectly.

3. Data Reconstruction Support Using Natural Language Processing

The purpose of this study is to support document reconstruction by presenting the semantic distance of two types of relationship strengths between field names in MongoDB JSON-structured documents. The proposed method measures two types of relationship fields in the MongoDB documents. The first measures the relationship between the parent and child fields. As described in Section 2, the parent and child fields are considered to have a hierarchical relationship. This relationship has often been considered as a Has-A

relationship in MongoDB usage. We evaluated the Has-A-relation relationship between words from a text corpus and calculated the strength of the Has-A-relation relationship between fields using natural language processing. We used Sketch Engine to identify words with a Has-A-relationship and calculated their strengths.

The second is the so-called horizontal relationship among fields at the same level in MongoDB documents. We calculated the similarity of field name words in addition to the strength of the Has-A-relation relationship. We calculated the cosine similarity by transforming a vector of words used in field names in documents using natural language processing. A cosine similarity is a similarity measure between two vectors and the cosine values of the vectors. We used fastText (https://fasttext.cc/ (accessed on 28 February 2024)) for the calculations and the resulting cosine similarity to show the semantic distance between words. According to this similarity, we cluster the field names and present the fields to be reconstructed to the user.

This tool prompts users to restructure based on hierarchical and horizontal relationships. The user moves the fields to be reconfigured into subtrees that already exist in the document or are newly created as a new nested document.

3.1. Hierarchical Relationship

We used Sketch Engine to detect words in a Has-A relationship between the words of the parent and child fields in MongoDB. The nested document in MongoDB corresponds to linking to a table in the RDB, and the field names of the nested document correspond to the column names in the table. Thus, the hierarchical relationship of a nested document is a Has-A relationship. The parent field name and child field name are calculated using Sketch Engine [5] to check whether their words form this Has-A relationship. Sketch Engine is a powerful corpus builder and parser for natural language processing and analysis. Sketch Engine provides features such as word frequencies, word co-occurrence, grammar, and scanning and is used to study grammatical patterns and lexical usage. Sketch Engine also has the ability to create a user's own corpus and special query language, the corpus query language (CQL) [6], which can search for flexible and complex lexical patterns for the corpus. Table 1 lists the patterns in which words represent the Has-A relationship to be detected, as defined by the CQL. For example, if a sentence is "A has B", which indicates that the relationship between A and B is a Has-A relationship, CQL defines the following pattern to detect:

```
[tag="N.*"][lenma ="have"][word ="a"]?[][tag="N.*"]
```

An expression `[tag="N.*"]` specifies the parts of speech and indicates that words that can be applied to nouns is present, `[lenma ="have"]` indicates that a verb whose base form is "have" is present, and `[word ="a"]?` represents that "a" is optional. Refer to other descriptive methods [6].

If a sentence "A car has an engine" is the given text, Sketch Engine detects the pair of words, car and engine, using the above CQL definition, and we consider that the words "car" and "engine" are in a Has-A relationship. Moreover, Sketch Engine provides the strength value of words using measurements such as LogDice and MI scores. Both the LogDice and MI (Mutual Information) coefficient are statistical measures used to identify co-occurrence (i.e., two items occurring together). LogDice is more useful because its values are not affected by corpus size. Thus, Sketch Engine uses LogDice to evaluate the relationship between the parent and child fields [7].

Table 1. The patterns of a Has-A relationship in CQL.

Pattern to Be Detected	**An Example**
CQL definition	
A have B.	A car has an engine.
[tag= "N.*"][lenma ="have"][word ="a"]?[][tag="N.*"]	
A is a part of B.	Fingers are a part of hand.
[tag="N.*"][word ="is"][word ="a"][word ="part"][word ="of"] []1,3[tag="N.*"]	
A inside B	Walls inside the building
[tag="N.*"][word ="inside"][word ="a" or the] [tag="N.*"]	
A above B	His leg above the knee
[tag="N.*"][word ="above"][word ="a" or the] [tag="N.*"]	
A is a member of B	Iceland is a member of NATO
[tag="N.*"][word ="is"][word ="a"][word ="member"][word ="of"] [tag="N.*"]	
A of B	Legs in a table
[tag="N.*"][word ="of"][word ="a" or the] [tag="N.*"]	
A in B	An engine in a car
[tag="N.*"][word ="in"][word ="a" or the] [tag="N.*"]	

3.2. Horizontal Relationship

We used fastText and DBSCAN to evaluate the horizontal relationships at the same level, creating clusters of field-name words. FastText is a neural-network-based method used to obtain a distributional representation of words (a representation using low-di mensional dense vectors characterizing a word concept) [8]. It uses a language model known as the skip-gram model to obtain a distributed representation of words [9]. It is characterized by the fact that it considers word sub-words. Specifically, a vector of words is generated by adjusting the vector values such that the vectors of words that have similar meanings and co-occur in the context of the training document are assigned closer to each other. Otherwise, vectors of words that do not have similar meanings are assigned further from each other. The distance between vectors was calculated using cosine similarity. DBSCAN clusters field names with these vectors, which creates high-density regions as clusters, and words with long distances are clustered as outliers. For this purpose, elements with distance are used as parameters (epsilon or eps) for the upper bound of the distance in the neighborhood [10].

3.3. Reconstruction Process

We have developed a document reconstruction support tool for easy use in MongoDB. Figure 1 shows the reconstruction steps of the document reconstruction support tool.

S1:
- Read the data in the target MongoDB document.
- Set a corpus to be used in Sketch Engine to detect word pairs with a Has-A relationship in CQL. The English version of TenTen was used as the corpus. The TenTen corpus is a text corpus created on the Web [11]. This corpus was built using technology that specializes in collecting only linguistically valuable Web content. It is the largest corpus in Sketch Engine and has been observed to be effective for analyzing a wide range of domains (https://www.sketchengine.eu/blog/build-a-corpus-from-the-web/ (accessed on 28 February 2024)).
- Extract a pair of words with a Has-A relationship using CQL in Sketch Engine.

S2:
- Read the word pairs of the Has-A relationship and calculate their strength, which is measured using LogDice. We used LogDice coefficients [7] to extract the indicators of word Has-A relationship feasibility from Sketch Engine using the method described in Section 3.1. The LogDice coefficient is a statistical measure used to identify co-occurrence (i.e., two items occurring together). This represents

the typicality of collocations. Because LogDice is not affected by corpus size, it can be used to compare scores across different corpora and is an effective statistical measure for large corpora.

S3: • Calculate the word vector with fastText
Cosine similarity is calculated for the vectors of words corresponding to child field names at the same level in a nested document and is presented as similarity using the similarity method of fastText. The cosine similarity indicates the closeness of the vectors.

S4: • Cluster field names used the vector representation of field names obtained in S3 so that the field names were clustered by DBSCAN.

S5: • There are two methods to achieve this. The first moves the fields to be reconstructed into an existing, appropriate nested document. The second is the creation of a nested document field. In the latter case, the user creates a new field for the separated fields and moves them to the new field as a nested document, leaving the other fields in their original positions. The tree with clearer roles than the original tree is separated. The user then specifies the name of the new field. To support this, the tool recommends candidate names. The candidates are words contained in the names of the moved fields and words that co-occur with the moved field names.

• The effect of reconstruction was evaluated using the cohesion of documents before and after reconstruction. We used the sum of the squares of intracluster errors (SSE) as a measure of tree cohesion. This is expressed by the following equation [12]:

$$SSE = \Sigma_{j=1}^{K} \Sigma_{x_i \in C_j} ||x_i - \mu_j||^2$$

where $\mu_j = \frac{1}{n_j} \Sigma_{x_i \in C_j} x_i$ denotes the average of cluster C_j, and n_j denotes the number of elements in cluster C_j.

• The user examines the cohesion values, SSE. If the user is satisfied with the value, they perform an actual reconstruction of the extended schema. Otherwise, the user performs steps S1–S5 repeatedly, and the document evaluation measures consisting of LogDice, cosine similarity, and clustering results are output.

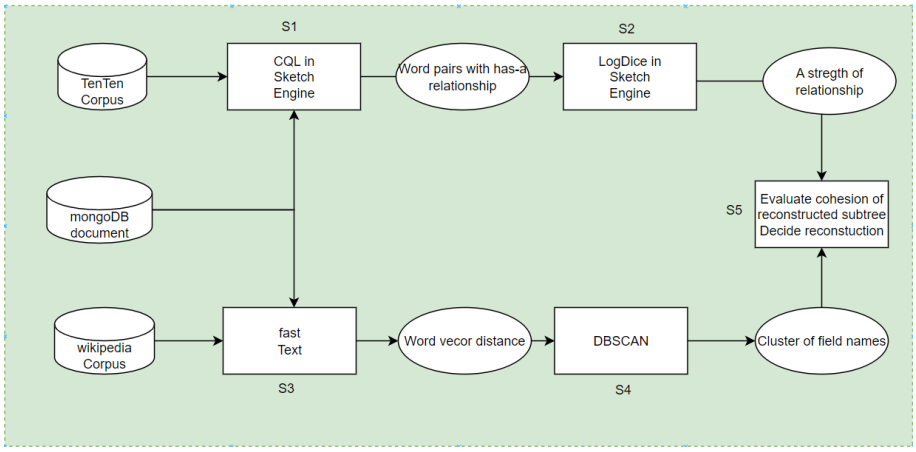

Figure 1. Reconstruction steps in the document reconstruction support tool.

Note that the above reconstruction does not necessarily consider the circumstances and query performance of the application program accessing data in MongoDB. Therefore, the decision to reconstruct was left to the user.

4. Overview of Support Tools

The following are the features of the document reconstruction support tool in terms of the user interface: The tool consists of three types of pane. Each shows the JSON data structure, hierarchical relationships between fields, and similarity relationships between fields.

- To display the tree document structure of the target MongoDB database.
  This is an overview of the data structure of the target MongoDB document with field names from the JSON document.
- To display parent–child fields in the tree.
  The hierarchical strength of the parent–child fields was calculated. The LogDice value between each field was attached to the edges of the tree. The larger the LogDice value, which implies that it has a stronger relationship, the wider the line of the edges. In addition, an edge is not displayed if the strength of the relationship is weaker than a threshold. These were the reconstruction candidates.
- To display similarity between fields in a subtree.
  This tool shows the results of field clustering based on the cosine similarity of the field names in the subtree. The similarity between field names leads to candidate reconstruction.
- Data structure reconstruction.
  When the field nodes to be reconstructed are identified, the tool displays the original tree to which the fields belong to and the target tree to which they are moved simultaneously. Users can also create new trees to accommodate these fields.
- Evaluation of reconfiguration.
  Based on this information, the user can make changes to the existing document structure on the screen, and the results of the changes are displayed. The tool also recalculates the strength of the relationship between the parent and child fields, similarity between the fields, and degree of cohesion as document evaluation measures, in addition to the history of the changes. The user can use these results to decide whether to modify the data structure in MongoDB.

Details of the screens and their operation are described below.

Figure 2 shows the basic properties of the MongoDB documents. The top part shows the names of the connected MongoDBs, and the collection list shows the collections stored in the connected MongoDBs. The user then selects the collection to be reconstructed from the collection list. The JSON DATA section (right part of the pane) shows the raw data of the collection. The graph section in the center of the pane presents the document portions extracted from the current JSON DATA as a tree structure. Users can obtain a general view of the database structure.

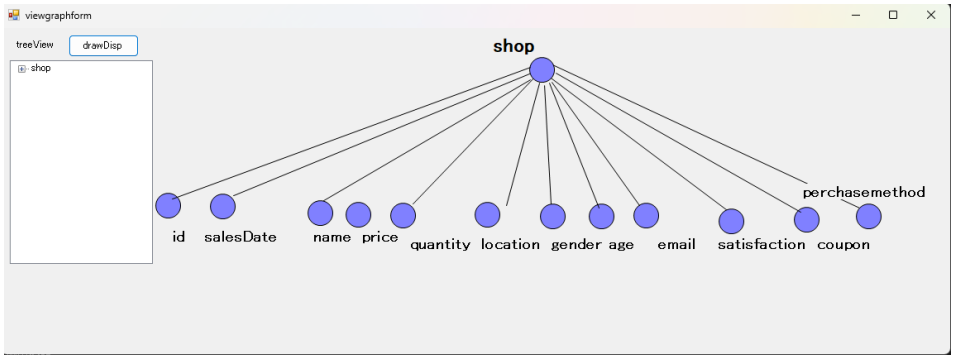

Figure 2. Displaying MongoDB document.

Figures 3–5 show panes that recommend document design changes and display the predicted results. In Figure 3, the graph display section in the center of the pane shows the clustering result, indicating the strength of the relationship between the parent field and child fields. Field nodes belonging to the same cluster have the same color. The closeness of the hierarchical relationships between the nodes is indicated by the width of the edge

with a closeness value. In Figure 4, the user creates a new node under the parent field node. The user can then input a new node name with the input dialog. The tool recalculates the document evaluation measures after the user has reconstructed a document. Figure 5 shows the measures and tree structures of the reconstructed document.

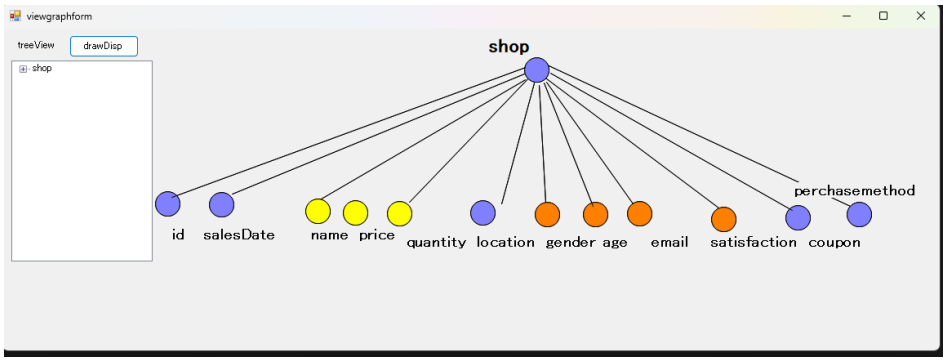

Figure 3. Clustering of field names with the strength of the Has-A relationship.

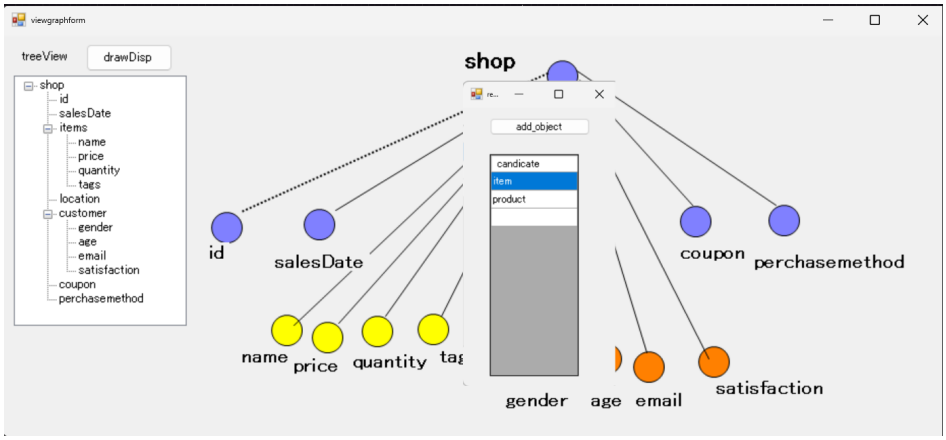

Figure 4. Adding field names to the embedded documents.

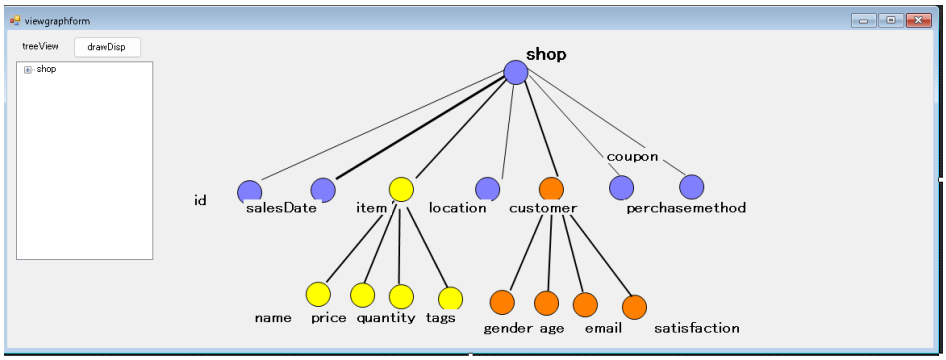

Figure 5. Document structure after creating the embedded document.

The tool was implemented using windows application Microsoft.NET frame-work 4.6. The DBSCAN and fastText computing environments were implemented as back-ends by installing Linux (ubuntu) on an AWS EC2 instance. The tool uses Sketch Engine through Web APIs.

5. Evaluations

To evaluate the effectiveness of the proposed document reconstruction support tool, we used open data and performed experiments to determine whether the document structure in MongoDB was coherently reconfigured. As evaluation data, we used e-commerce

purchase data (shop.json) from the open-data site OpenDatahub (https://opendatahub.com/datasets/ (accessed on 14 February 2024)).

We experimented to determine whether we could reproduce an original document from a document damaged from the original. The original document is shown on the right-hand side in Figure 6. We generated the damaged document shown on the left-hand side in Figure 6 by moving the `gender`, `age`, and `email` fields originally under the `customer` fields to the `item` field. We attempted to reproduce the damaged document to give the original document using the proposed method. Figures 7 and 8 show the LogDice values calculated using Sketch Engine between the parent collection names and child field names in the shop data.

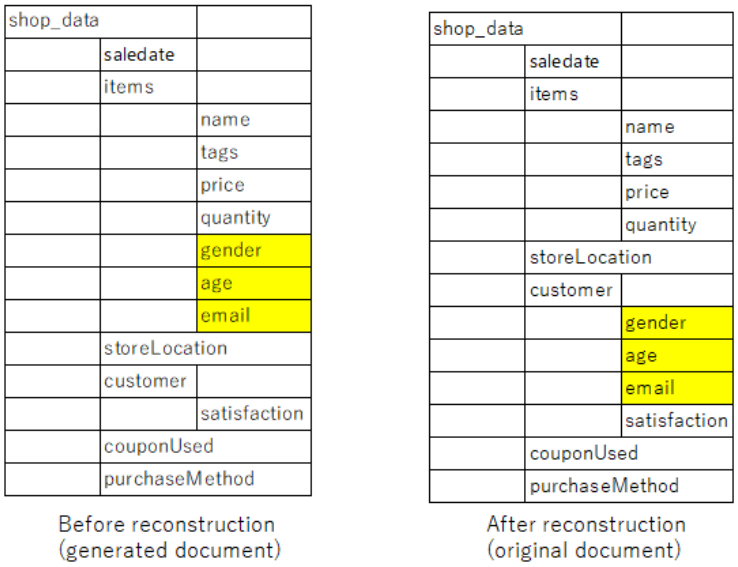

Figure 6. Generated document before and after the reconstruction of the shop data.

			LogDice
shop_data			
	saledate		6.98
	items		6.34
		name	6.34
		tags	6.45
		price	6.56
		quantity	6.55
		gender	5.98
		age	5.75
		email	6.01
	storeLocation		6.67
	customer		6.76
		satisfaction	6.34
	couponUsed		6.45
	purchaseMethod		6.5

Figure 7. LogDice values before reconfiguration of the shop data.

If we set the threshold of the LogDice value to approximately 6.5, we can separate the `gender`, `age`, and `email` fields into different groups because the values are under 6.5 and those of the other fields are above 6.5. This suggests that we should consider placing these three fields in separate subtrees during the reconstruction. This was consistent with the structure of the original document.

Figure 7 and Figure 8 shows the LogDice values before and after reconstruction, respectively. In Figure 8, the three fields have comparable values under the `customer` field.

			LogDice
shop_data			
	saledate		6.98
	items		6.34
		name	6.34
		tags	6.45
		price	6.56
		quantity	6.55
	storeLocation		6.67
	customer		6.76
		gender	6.87
		age	6.54
		email	7.12
		satisfaction	6.34
	couponUsed		6.33
	purchaseMethod		6.5

Figure 8. LogDice values after reconfiguration of the shop data.

Next, we measured the similarity of the field names in the `items` subtree of the damaged document. We measured the semantic distance between the child fields and clustered them using DBSCAN by setting eps to 0.1. The results are shown in Figure 9. The results suggest that the `gender`, `age`, and `email` fields were separated into clusters, and the remaining fields (`name`, `tags`, `price`, and `quantity`) were separated into two clusters. The similarity of the separated fields was approximately 0.2–0.4, which is significantly higher than the other fields. This result is also consistent with the original document structure.

	name	tags	price	quantity	gender	age	email
name	-						
tags	0.11	-					
price	0.13	0.01	-				
quantity	0.16	0.1	0.41	-			
gender	0.11	0.1	0.21	0.24	-		
age	0.19	0.23	0.17	0.17	0.44	-	
email	0.15	0.17	0.2	0.02	0.39	0.22	-

Figure 9. Results of the cluster analysis of the field names.

According to this result, the fields of `gender`, `age`, and `email` were moved from under `item` to `customer`. Figure 10 shows the changes in the coherence values of the subtrees using the SSE. The smaller the SSE values, the better the coherence. A tree coherence close to one indicates that the field nodes in the subtree are distributed. Before the modification, coherence values in the `shop_data` subtree were 0.91, 0.53, and 0.43, while after the modification, the values decrease to 0.88, 0.36, and 0.45, respectively. The field names became more cohesive, indicating that these modifications were effective.

parent field name	before reconstruction	after reconstruction
shop_data	0.91	0.88
item	0.53	0.36
customer	0.43	0.45

Figure 10. Changes in cohesion before and after reconstruction of shop date.

Secondarily, we tested whether it was possible to construct an appropriate hierarchy for data without a hierarchical structure. We used JSON data (person lists) containing the personal information of one person. JSON documents before and after the reconstruction are shown in Figure 11. Figure 12 shows the LogDice values between the field and collection

names in the person list document calculated using Sketch Engine. Considering that LogDice is approximately 6.5, the fields `address`, `street`, `city`, `state`, and `zip` can be divided into different groups. This implies that we should consider combining these five fields in a different hierarchy.

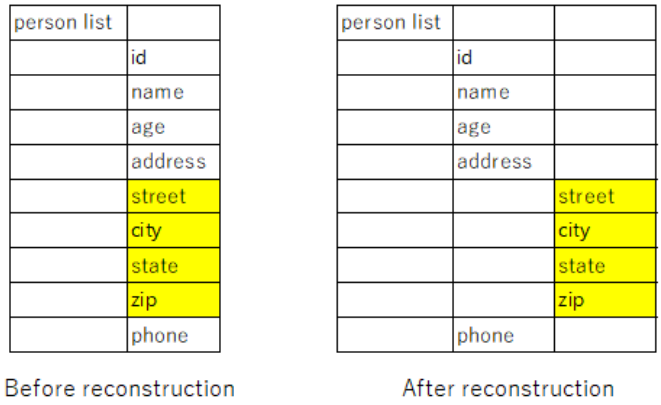

Figure 11. Before and after the reconstruction of the person list.

			LogDice
person list			
	id		6.71
	name		6.65
	age		6.57
	address		6.45
	street		6.45
	city		6.45
	state		6.31
	zip		6.35
	phone		6.45

Figure 12. LogDice values before reconfiguration of the person list.

We measured the semantic distance between the child fields and clustered them using DBSCAN with eps set to 0.1. The results are shown in Figure 13. Fields `address`, `street`, `city`, `state`, and `zip` were separated as a cluster and divided into two clusters. Similarity of the separated fields was approximately 0.2–0.4, which is significantly higher than the other fields. The results of the cluster separation were consistent with the results of the LogDice separation of the Has-A relationship. This indicates that the fields `address`, `street`, `city`, `state`, and `zip` can be separated into separate subtrees. The results are shown in Figure 14.

	name	age	address	street	city	state	zip	phone
name	-							
age	0.11	-						
address	0.13	0.11	-					
street	0.16	0.1	0.39	-				
city	0.11	0.1	0.41	0.4	-			
state	0.16	0.11	0.29	0.35	0.39	-		
zip	0.15	0.11	0.25	0.33	0.41	0.28	-	
phone	0.11	0.12	0.11	0.16	0.11	0.11	0.11	-

Figure 13. Clustering results with DBSCAN.

			LogDice
person list			
	id		6.71
	name		6.65
	age		6.57
	address		6.45
		street	7.56
		city	7.86
		state	7.77
		zip	7.98
	phone		6.45

Figure 14. LogDice values after reconfiguration of the person list.

Finally, we examined whether a JSON hierarchical structure can be created using open CSV data published on the Internet. Many of these open data sources are in the CSV format and are not necessarily published in the JSON structure. JSON documents before and after the reconstruction are shown in Figure 15. Figure 16 shows the fields and LogDice values of the CSV file containing hotel information. The `room_type_name` field was separated from the `room_type_cancellation` field at seven points of the LogDice value. Thus, we generated a separate subtree whose name was `room_type` and whose members ranged from `room_type` to `room_type_cancellation`. The schema and LogDice values of the reconstructed `hotel_list` are shown in Figure 17.

hotel list	
	record
	uniq_id
	hotel_id
	hotel_name
	default_rank
	price_rank
	room_name
	room_price
	occupancy
	breakfast
	cancellation
	checkin_date
	crawled_date

Before reconstruction

hotel list		
	record	
	uniq_id	
	hotel_id	
	hotel_name	
	default_rank	
	price_rank	
	booking	
		room_name
		room_price
		occupancy
		breakfast
		cancellation
	checkin_date	
	crawled_date	

After reconstruction

Figure 15. Before and after the reconstruction of the hotel list.

We measured the similarity of the field names in this CSV file with semantic distance and clustered them in DBSCAN with eps set to 0.1. The results are shown in Figure 18. The `room_name`, `room_price`, `occupancy`, `breakfast`, and `cancellation` fields were separated into clusters that could be further divided into two clusters. Similarity of the separated fields was approximately 0.2–0.4, which is significantly higher than the other fields. The results of the cluster separation were consistent with the results of the LogDice separation of the Has-A relationship. Based on these results, we created a new subtree `room_type`, and the child fields in the subtree were `room_type_name`, `room_price`, `occupancy`, `breakfast`, and `cancellation`.

		LogDice
hotel list		
	record	6.71
	uniq_id	6.65
	hotel_id	7.77
	hotel_name	7.88
	default_rank	6.44
	price_rank	6.43
	room_name	7.74
	room_price	7.43
	occupancy	7.46
	breakfast	7.22
	cancellation	7.65
	checkin_date	6.43
	crawled_date	6.38

Figure 16. LogDice values before the reconstruction of the hotel list.

			LogDice
hotel list			
	record		6.71
	uniq_id		6.65
	hotel_id		7.77
	hotel_name		7.88
	default_rank		6.44
	price_rank		6.53
	booking		8.56
		room_name	7.81
		room_price	7.67
		occupancy	8.12
		breakfast	8.11
		cancellation	8.21
	checkin_date		6.82
	crawled_date		6.54

Figure 17. LogDice values after the reconstruction of the hotel list.

We measured the coherence after the reconstruction (See Figure 19). Before the reconstruction, the single tree, `hotellist`, was 0.82; however, after the reconstruction, the `hotellist` and newly created subtree were smaller, 0.54 and 0.31, respectively, indicating that document name cohesion increased and the reconstruction was effective. In other words, fields that should belong to individual rooms were separated from those that represented the status of the hotel. This indicates that the data structure in the JSON document was improved.

	record	uniq_id	hotel_id	hotel_name	default_rank	price_rank	room_name	room_price	occupancy	breakfast	cancellation	checkin_date	crawled_date
record	-												
uniq_id	0.11	-											
hotel_id	0.13	0.11	-										
hotel_name	0.16	0.1	0.23	-									
default_rank	0.11	0.1	0.24	0.24	-								
price_rank	0.11	0.11	0.25	0.17	0.12	-							
room_name	0.11	0.15	0.33	0.33	0.11	0.24	-						
room_price	0.11	0.16	0.34	0.34	0.11	0.23	0.34	-					
occupancy	0.1	0.11	0.35	0.35	0.15	0.22	0.35	0.33	-				
breakfast	0.1	0.1	0.38	0.38	0.15	0.22	0.32	0.34	0.33	-			
cancellation	0.13	0.1	0.41	0.41	0.16	0.25	0.33	0.35	0.32	0.35	-		
checkin_date	0.1	0.17	0.32	0.23	0.11	0.14	0.1	0.14	0.1	0.41	0.18	-	
crawled_date	0.1	0.1	0.33	0.1	0.1	0.14	0.1	0.14	0.1	0.1	0.11	0.41	-

Figure 18. Clustering results from DBSCAN.

parent field name	before reconstruction	after reconstruction
hotellist	0.82	0.54
room_type		0.31

Figure 19. Changes in cohesion before and after reconstruction of hotel list.

Performance Evaluation

We evaluated the performances that were affected by the reconstructions using the computer shown in Table 2.

Table 2. Computer specifications for the performance evaluation.

CPU	Intel(R) Core(TM) i5-8365U CPU @ 1.60 GHz1.90 GHz
Memory	16GB
OS	Windows 10

To check the performance of the tool after reconfiguration, we perform queries on the documents containing the changed fields for the data models used in this section and measure the performance. For the shop data used in Figures 7 and 8 and the hotel list document in Figures 16 and 17, we prepared data sets with 10 to 5000 records whose values were random in the fields. We created queries that returned 10 records by specifying the field that was moved in the reconstruction and measured the time performed by the queries. The results of the shop data and the hotel list are shown in Table 3 and Table 4, respectively. We can say there was almost no difference in the time to return the search results before and after the reconstruction for both the shop data and hotel list; thus, we can conclude that there are significant changes in performance. As for the hotel list, the execution times slightly increased because the hierarchy level increased.

Table 3. Performance in the shop data.

The number of query	100	1000	10,000	50,000
Before reconstruction	5	102	380	1801
After reconstruction	3	99	365	1656

Table 4. Performance in the hotel list data.

The number of query	100	1000	10,000	50,000
Before reconstruction	10	115	401	2340
After reconstruction	13	121	403	2400

6. Related Works

MongoDB is characterized by its flexibility in that it can be used without having to specify a schema in advance. This has led to various problems, and solutions to these problems have been studied. The selected studies were categorized into three groups. The first aims to support the handling of MongoDB documents, while the second aims to generate schemas from MongoDB documents. The third is a modeling guideline for developing well-formed MongoDB documents.

First, data visualization tools are presented; the most general software tools for MongoDB management are MongoDB Compass and NoSQL Manager. MongoDB Compass provides tools for visualization in list and table structures [13]. NoSQL managers for MongoDB allow users to view and manipulate documents in three ways: tree view, which allows users to see and manipulate documents like a horizontal tree; view, which views

documents such as tables; and text, which views documents such as JSON documents, which can be represented and handled by the user [14].

Concerning the second research category, tools have also been developed to analyze the structure of MongoDB documents. jHound is a tool used to profile large collections of JSON documents [15]. It shows key indicators of data in the MongoDB document tree, such as tree depth, the levels at which many fields are gathered, and the number of fields, allowing a user to determine the structure of a MongoDB document. This helps to detect structural anomalies in documents.

Attempts have also been made to extract and automatically generate schemas for existing MongoDB documents. Wang et al. developed a schema management framework for document storage [16]. The framework discovers the schema of the JSON records and stores it in a repository. In addition, it supports schema summarization. The main challenge in schema management is that the diverse data structures cause the evolution of data structures in a schemaless data model. The authors proposed an algorithm and data structure called eSiBuTree based on equivalent subtrees in the schema discovery phase and generated heterogeneous schemas from a single summarized representation. To present a single summary representation for the heterogeneous schema of a record, the authors introduced the notion of a "skeleton" and proposed to use this as a relaxed form of the schema.

Klettke et al. also proposed a process for extracting the schema from the document [17]. The authors proposed a data structure called a structure-identification graph (SG) attached to a field. SG represents data in which the field is either mandatory or optional and is a data type of the field. For example, fields with mandatory attributes must appear in all the JSON documents. A schema can also specify whether an attribute supports multiple data types.

Ruiz et al. proposed a reverse engineering process for inferring schemas in aggregate-oriented NoSQL data using a meta-model [18]. The process has three steps: (1) to extract a set of JSON objects for each version of a real-world entity using a MapReduce-based procedure, (2) to put the objects into the JSON model, and (3) to transform the JSON model into a model that confirms the NoSQL-schema meta-model.

Izquierdo et al. proposed a model-based process to generate schemas from JSON documents that return values from a Web service [19]. The process consists of the following three steps: (1) a pre-discovery phase to extract low-level JSON models from JSON documents, (2) a single-service discovery phase to obtain schema information for concrete services (inferred from a set of low-level JSON models that are output from a series of Web service calls), and (3) a multi-service discovery phase to synthesize the schema information obtained in (2) to obtain an overall picture of the application domain. This process produces a schema refined from a group of JSON documents.

Baazizi et al. introduced a JSON-type language to express schema inferences, as well as an algorithm to infer the schema from a JSON data set using this language [20]. This algorithm consists of the following two steps: (i) A map step to process JSON values as an input with a map function that infers a simple data type for each value. (ii) A reduce step to fuse the inferred data types and identify the mandatory, optional, and repeated data types that output a JSON schema.

In the third category, there has also been an attempt to develop guidelines for modeling document structures. Imam et al. proposed modeling guidelines for NoSQL document storage databases [21]. These guidelines span both logical and physical phases of database design. The guidelines are design recommendations in terms of embedding, referencing, and bucketing in database design, which have been used in previous studies. The authors observed the database design of industry experts regarding how they prioritized the guidelines. The authors heuristically evaluated five NoSQL databases for guidelines.

7. Conclusions

In this study, we proposed a method for supporting tree structure reconstruction in MongoDB to solve the problems caused by the flexibility of the schemaless feature of MongoDB. This method is based on the hypothesis that the field names of the JSON trees have a strong relationship. In the proposed method, we built a process that shows reconstruction candidates using a method that measures the distance and degree of association between words using natural language processing and developed a MongoDB document reconstruction support tool to execute this process. To evaluate the effectiveness of the support tool, we used evaluation data from open data sources and conducted a reconstruction. The evaluation results show that appropriate document structures could be created through reconstruction. It was also shown that an appropriate hierarchy can be constructed from data that do not have a hierarchical structure.

In future work, it will be necessary to assign appropriate field names, because our method relies on field names in MongoDB documents. To do so, we must understand the background of database usage and the business domain of the applications to incorporate this knowledge into the tool. By applying this method to real-world projects, we can improve it. In addition, we would like to open our tool for the wide usage.

Author Contributions: Conceptualization, K.H. and Y.N.; methodology, K.H. and Y.N.; software, K.H.; validation, K.H.; formal analysis, K.H.; investigation, K.H.; resources, Y.N. and K.H.; data curation, K.H.; writing—original draft preparation, K.H.; writing—review and editing, Y.N.; visualization, K.H.; supervision, Y.N.; project administration, K.H. and Y.N. All authors have read and agreed to the published version of the manuscript.

Funding: Not applicable.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Publicly available datasets were in this study. Sketch engine Corpus is available at https://www.sketchengine.eu/documentation/tenten-corpora/ (accessed on 14 February 2024). Json data for evaluation is available at https://opendatahub.com/datasets/ (accessed on 14 February 2024). Internal generated dataset is available on request from the corresponding author.

Conflicts of Interest: Author Kohei Hamaji was employed by the company Honda Motor Co., Ltd. The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

References

1. Davoudian, A.; Chen, L.; Liu, M. A Survey on NoSQL Stores. *Acm Comput. Surv.* **2017**, *51*, 40. [CrossRef]
2. Hamaji, K.; Nakamoto, Y. Development for DB schema reconstruction support tool using natural language processing. *Comput. Softw.* **2022**, *39*, 29–38. [CrossRef]
3. MongoDB. *MongoDB Manual*. Available online: https://www.mongodb.com/docs/manual/ (accessed on 2 February 2024).
4. Bourhis, P.; Reutter, J.L.; Suárez, F.; Vrgoč, D. JSON: Data Model, Query Languages and Schema Specification. In Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, Chicago, IL, USA, 14–19 May 2017; pp. 123–135.
5. Kilgarriff, A.; Baisa, V.; Bušta, J.; Jakubíček, M.; Kovář, V.; Michelfeit, J.; Rychlý, P.; Suchomel, V. The Sketch Engine: Ten Years on. *Lexicography* **2014**, *1*, 7–36. [CrossRef]
6. Sketch Engine Team. *CQL—Corpus Query Language*. Available online: https://www.sketchengine.eu/documentation/corpus-querying/ (accessed on 2 February 2024).
7. Rychlý, P. A Lexicographer-Friendly Association Score. In Proceedings of the Recent Advances in Slavonic Natural Language Processing, Karlova Studánka, Czech Republic, 5–7 December 2008; pp. 6–9.
8. Bojanowski, P.; Grave, E.; Joulin, A.; Mikolov, T. Enriching Word Vectors with Subword Information. *Trans. Assoc. Comput. Linguist.* **2017**, *5*, 135–146. [CrossRef]
9. Mikolov, T.; Chen, K.; Corrado, G.; Dean, J. Efficient Estimation of Word Representations in Vector Space. In Proceedings of the International Conference on Learning Representations, Scottsdale, AZ, USA, 2–4 May 2013.
10. Parimala, M.; Lopez, D.; Senthilkumar, N.C. A Survey on Density Based Clustering Algorithms for Mining Large Spatial Databases. *Int. J. Adv. Sci. Technol.* **2011**, *31*, 59–66. [CrossRef]

11. Sketch Engine Team. *TenTen Corpus Family*. Available online: https://www.sketchengine.eu/documentation/ (accessed on 2 February 2024).
12. Su, T.; Dy, J. A deterministic method for initializing K-means clustering. In Proceedings of the International Conference on Tools with Artificial Intelligence, Boca Raton, FL, USA, 15–17 November 2004; pp. 784–786. [CrossRef]
13. MongoDB. *Compass. The GUI for MongoDB*. Available online: https://www.mongodb.com/products/tools/compass/ (accessed on 2 February 2024).
14. NoSQL Manager Group. *How the NoSQL Manager Helps You to Work with MongoDB*. Available online: https://www.mongodbmanager.com/ (accessed on 2 February 2024).
15. Möller, M.L.; Berton, N.; Klettke, M.; Scherzinger, S.; Störl, U. jHound: Large-Scale Profiling of Open JSON Data. In Proceedings of the 15th Conference on Database Systems for Business, Technology and Web, Dresden, Germany, 4–8 March 2019; pp. 555–558. [CrossRef]
16. Wang, L.; Zhang, S.; Shi, J.; Jiao, L.; Hassanzadeh, O.; Zou, J.; Wangz, C. Schema management for document stores. *Proc. Vldb Endow.* **2015**, *8*, 922–933. [CrossRef]
17. Klettke, M.; Störl, U.; Scherzinger, S. Schema extraction and structural outlier detection for JSON-based NoSQL Data Stores. In Proceedings of the 16th Conference on Database Systems for Business, Technology and Web, Brussels, Belgium, 2–6 March 2015; pp. 425–444.
18. Ruiz, D.S.; Morales, S.F.; Molina, J.G. Inferring Versioned Schemas from NoSQL Databases and Its Applications. In Proceedings of the International Conference on Conceptual Modeling, Stockholm, Sweden, 19–22 October 2015; pp. 467–480. [CrossRef]
19. Izquierdo, J.L.C.; Cabot, J. Discovering implicit schemas in JSON data. In Proceedings of the International Conference on Web Engineering, Aalborg, Denmark, 8–12 July 2013; pp. 68–83.
20. Baaziz, M.A.; Lahmar, H.B.; Colazzo, D.; Ghelli, G.; Sartiani, C. Schema Inference for Massive JSON Datasets. In Proceedings of the Extending Database Technology, Venice, Italy, 14–18 March 2013; pp. 222–233.
21. Imam, A.A.; Basri, S.; Ahmad, R.; Watada, J.T.; Ahmad, M. Data Modeling Guidelines for NoSQL Document-Store Databases. *Int. J. Adv. Comput. Sci. Appl.* **2018**, *9*, 544–555. [CrossRef]