computation

MDPI

*Article*

# Creating Collections with Embedded Documents for Document Databases Taking into Account the Queries

**Yulia Shichkina * and Muon Ha**

Faculty of Computer Science and Technology, Saint Petersburg Electrotechnical University, 197022 Saint Petersburg, Russia; muon.ha@mail.ru

* Correspondence: strange.y@mail.ru

Received: 31 March 2020; Accepted: 13 May 2020; Published: 15 May 2020

**Abstract:** In this article, we describe a new formalized method for constructing the NoSQL document database of MongoDB, taking into account the structure of queries planned for execution to the database. The method is based on set theory. The initial data are the properties of objects, information about which is stored in the database, and the set of queries that are most often executed or whose execution speed should be maximum. In order to determine the need to create embedded documents, our method uses the type of relationship between tables in a relational database. Our studies have shown that this method is in addition to the method of creating collections without embedded documents. In the article, we also describe a methodology for determining in which cases which methods should be used to make working with databases more efficient. It should be noted that this approach can be used for translating data from MySQL to MongoDB and for the consolidation of these databases.

**Keywords:** NoSQL; database query; collection; document; database structure optimization; embedded documents

## 1. Introduction

Access to accurate information in the modern era is a major challenge that organizations have to face. For example, a police officer must know whether he has the right to apply the law when a new situation arises. The social worker must have accurate information about the applicant. The doctor wants to know all the information about the patient who could be treated in any hospital and with any diagnosis.

The aforementioned situations and many other situations require quick access to either a single common source of information, or to a data collection system from various sources. The main problem is that each of the sources of information usually allows obtaining specific information stored in it and this, as a result, entails a loss of understanding of the requested object as a whole. The latter leads to the fact that the information obtained does not reflect the full content of the object. Therefore, it is important to create systems for the automatic collection of data from databases of various types and structures.

Relational databases are the most common forms of data storage. In addition to these databases, there are NoSQL and NewSQL databases. These new forms of databases today are used by more and more companies, including such large ones as Google or Amazon [1]. Chen et al. [2] describe fifteen categories of NoSQL databases and some principles and examples for choosing a suitable NoSQL database for different industries. Diogo et al. [3] analyze and compare the consistency model implementation on five popular NoSQL databases: Redis, Cassandra, MongoDB, Neo4j, and OrientDB.

The problem of combining information is not only that data is collected from databases of various types. Even if the databases are of the same type, the databases can have a different structure.

When combining information, it is necessary to analyze the structure of the new database for the combined information. In this part, there is again a problem: the lack of a NoSQL database formalized apparatus for determining the number and composition of collections, in contrast to relational databases with methodologies based on relational algebra and functional dependencies. Each database administrator analyzes the structure and composition of the future database based on their experience and recommendations of leading experts.

The aim of our research was to find a method that allows creating the optimal collection structure with embedded documents with an orientation to future database queries. We have created such a method. Our research has shown that this method can also be used to transform databases such as a family of columns and even relational databases. The purpose of this article is to familiarize readers with this method and demonstrate its work on document databases, as these databases are very often used in practice and well described in a large number of publications, for example, [4–7].

We propose a new formalized method for constructing a NoSQL document database, taking into account the structure of queries planned for execution to the database. This method consists of steps and rules that allow creating a database structure that will speed up database queries and make the database smaller by storing fewer duplicates.

The method has two key properties: it takes into account the structure of queries and the relationship between database objects. The first property allows not to use "join" operations in database queries. The second property, due to the organization of embedded documents, reduces the number of duplicate data in the database, and makes the database smaller. The method can be used when creating new databases or restructuring existing document databases, as well as when converting relational databases to document databases.

The article is structured as follows. The second section provides an overview of related work, shows the diversity of existing solutions and their focus. In the third section, based on the analysis of existing solutions, it shows the relevance of the task of finding a formalized method for converting data from one format to another, taking into account the structure of queries. The fourth section provides a methodology for transforming the data structure depending on the initial state of the database, the fifth section describes a method for converting data from one format to another, taking into account the structure of queries and relationships between objects. The sixth section presents the results of testing the method. In conclusion, the research results are summarized and the eighth section contains the main directions for continuing research.

## 2. Related Work

Database consolidation is an important task in many areas of database applications, such as heterogeneous database integration, e-commerce, data warehousing, or semantic query processing. A lot of research has been and is being done in the world today by individual scientists as well as research organizations and, as a result, various solutions are offered for this problem. Among the known results in the field of database consolidation are the systems SemInt [8,9], Learning Source Descriptions (LSD) in [10,11], Semantic Knowledge Articulation Tool (SKAT) [12], TranScm [13], Palopoli in [14], ARTEMIS [15,16], and Mediator environment for Multiple Information Sources (MOMIS) [17,18]. However, all of these systems focus on consolidating relational databases and do not solve the problem of integrating a relational database with NoSQL or NewSQL. In our research, we created a method that can be used to consolidate relational databases and NoSQL with the resulting NoSQL database.

It should be noted that many studies over the past thirty years have been devoted to the problems of transforming a relational database schema [19,20]. No matter how the scheme of the relational database has been improved, it has its limitations, which contributed to the development of other forms of databases, such as key-value and their varieties [4–7] or graph databases [21].

The integration of relational databases and NoSQL is a new problem that is relevant to the urgency of the solution and studied little today.

Chickerur [22] proposed a way to transfer relational databases to MongoDB by converting tables to CSV files and import the converted files using the built-in MongoDB command. He also compared the performance of MySQL and MongoDB using queries to add, update, and delete data from an airline database. Performance tests showed that MongoDB performs queries well on large amounts of data. However, this method only directly translates tables into collections without regard to the relationships between tables. MongoDB performs worse queries on a set of collections. Our method differs in that it takes into account the structure of database queries.

Hanine et al. [23] developed an approach to transferring data from relational databases to MongoDB, which consists of three stages: extracting data from the source database, converting data, and transferring the converted data to the target database. Unfortunately, this approach does not solve the problem of the dependence of query performance on database schemas and relationships between objects. In our approach, we take into account the relationships and show that query performance increases from this.

Li et al. [24] proposed a model for converting a relational schema into a NoSQL database schema based on a data structure and data queries. This model has a three-phase structure: a description of the structure of the relational database and the requirements for data queries; query-oriented data modeling for NoSQL database; a query-oriented description of the NoSQL database schema. In this model, there is no allowance for dependencies between database objects during the transition from a relational database to a NoSQL database. The new NoSQL database structure is based only on metadata about objects and queries. The method described in [25] is also based on information about queries. In our approach, we show how relationships between objects affect the structure of a document with embedded documents.

Zhao et al. [26] proposed an approach to data migration from a relational database to a NoSQL database based on relational algebra methods. In this study, the authors propose adapting the theory and methods of relational algebra to the development of the MongoDB database schema. For the same purposes, we use set theory, which allows us to construct formalized rules for translating a relational database into a MongoDB database.

Alotaibi et al. [27] propose six rules for translating a relational database into a NoSQL database of three types (Column-Based, Document-Based, and Graph-Based). Moreover, in all three types, NoSQL database subspecies are considered. Each rule is associated with the type of relationship between the tables (1-1, 1-M, M-M) or with a special operation carried out in one of the tables (for example, aggregation). This article demonstrates how easy it is to translate a relational database into NoSQL if it is possible to ignore the structure of the queries. The problem is that NoSQL databases are query-specific. Those queries that refer to several tables in the relational database will work very slowly in the NoSQL database, if there was a direct translation of the relational database into NoSQL. The main problem is the lack of NoSQL databases of the operations of join data sets (collections, tables, families, etc.). The authors solve this problem by creating a single collection in the NoSQL database. This is not a good solution: combine all the tables into one collection. This leads to an increase in data volume and a memory problem. At the end of our article, we show, based on the results of testing, that our approach based on building a certain number of collections is better than the approach based on creating a single collection.

Celesti et al. [28] accelerate queries that use operations «join» at the application level. They implement their approach also in MongoDB. Their approach is good because it allows saving data models. This approach will not be very good in case of data consolidation, when the same data will appear in different models in one way or another and, as a result, during consolidation, structure modification will still be necessary. We solve this problem in our approach. When consolidating to an existing database of another database, objects and new object attributes appear in the current database. Therefore, database queries in the previous form cannot be performed or are executed for a very long time. The method proposed by us allows restructuring the database schema so that all queries are executed quickly.

One possible solution is the NoSQLayer software module, which can consist of two modules [29,30]: a data transfer module that identifies and analyzes the NoSQL database schema and converts it into a relational database schema, and a data mapping module that allows users to create queries in SQL, selecting data from the NoSQL database. This program is not intended for data migration and many queries work more slowly than if there were direct access to the NoSQL database in the built-in query language [31]. In our approach, we optimize the database query processing using the internal query language.

Feng et al. [32] developed an approach to transforming UML class diagrams into a Cassandra database schema. This approach does not provide for optimizing the NoSQL schema for executing queries. Currently, we tested and proved that the query structure when creating a collection affects performance in the MongoDB database, and we are currently testing our approach on Cassandra.

Kartinis et al. [33] provided a technology that allows translating relational database systems into document-oriented databases. This technology consists of two stages: description of the data of the existing relational database and transfer to the NoSQL database. This technology does not provide for optimizing the NoSQL schema for queries. Our method takes into account the relationships between tables to create embedded documents and reduce the size of the database.

According to [34], relations in a document-oriented database model can be represented in the form of embedded documents and relationships between these documents. However, firstly, embedded documents can be used for a limited amount of data, and secondly, it remains a problem to determine the form of embedding. The studies described in [34] confirm this. A positive characteristic of our method is not only its ability to make decisions about the need to use embedded documents, but, very importantly, the definition of the form of embedded documents and the place where they should be embedded.

Gu et al. [35] applied the traditional rules of normalizing the relational database schema theory (normal form theorems: second and third normal forms (2NF, 3NF) and the Boyce–Codd normal form (BCNF)) to develop the MongoDB schema. For example, if there is a functional relationship between two attributes, then both data attributes are converted to a single data item in MongoDB. The same principle applies to partial and transitive dependencies. Testing showed that, as a result of applying this approach, it is possible to actually increase query performance for related database objects. However, testing was carried out on a small scheme. The approach does not take into account relationships "Many to Many", primary keys and foreign keys. In this article, we show how relationships of this type between objects can be taken into account.

Thus, the analysis of works in the field of optimizing the structure of NoSQL databases, in particular, document databases, showed that there are many different approaches to translating databases, but there is no integrated approach that would simultaneously take into account the types of relationships between objects and the structure of queries to objects databases and would allow building such a number of collections and such an internal structure that queries to select data from a document database are executed as quickly as possible.

In this article, examples of tests of the proposed method used MongoDB databases of various structures and sizes.

MongoDB is a database that is used very often in various subject areas. In the literature, there are many good methodologies for its application for various practical problems. For example, Wang et al. [36] describe the sharding technology of MongoDB and the new approach to managing large-scale remote sensing data; Marrara et al. [37] describe a method for constructing blind queries to JSON document stores using MongoDB as an example; the method for processing queries in the space-time range based on the index using the example of a large database MongoDB is described by Qian et al. [38]. Many works are devoted to describing the application of the MongoDB system in various subject areas: to create a prototype of a three-dimensional cadastral system and three-dimensional visualization [39]; predicting the consumed heat capacity of buildings [40], to control groundwater flow and contaminant transport [41], in a personalized healthcare monitoring system [42] and, of course, IoT applications [43].

## 3. Formulation of the Problem

Transferring data between different data sources is a necessary step for many data mining tasks. However, the differences between the storage methods in the two relational and non-relational (NoSQL) forms pose many problems in the areas of data translation, transformation, and consolidation. One of these problems is matching NoSQL collections to relational database tables. For translation, transformation, and consolidation of various types of databases, it is also necessary to take into account the lack of structure in some databases and the features of the query language.

For relational databases, there are methods for creating relationships based on declared functional dependencies between attributes. For NoSQL and NewSQL databases, such formalized methods do not exist, only recommendations. For example, when converting a relational database into a document database, the database administrator has little benefit from knowing the relational database schema. He needs to consider many options for the form of a document database in order to choose the best one.

In this case, the following options for formatting are possible (the list is not complete):

- To each table in a relational database to assign a separate collection of documents in MongoDB;
- From all tables in the relational database to make one collection of documents in MongoDB, by applying the join operation to the tables;
- To create such a set of document collections in MongoDB, so that they most fully fit the queries being executed.

These three options can be effective with one database design and not be effective with another database. In our article [25], we presented a method based on set theory, which allows determining which of the three options will be effective. In this article, we show how it is possible to make the selected collection structure even more effective if you create collections with attached documents. At the same time, the choice of the number of collections and the determination of the contents of the collections (with embedded documents or without embedded documents) is based on such parameters as: attributes of objects in the database, structure of queries to the database and types of relationships between database objects.

## 4. The Methodology for Determining Collections in a Document Database

In the method of creating collections without embedded documents, the new MongoDB database is created on the basis of the existing relational database. We have shown that the use of a formalized set-based approach helps to find the optimal collection of collections in MongoDB based on information about the executed queries to the database.

If the new MongoDB database is not created from a relational database then it is necessary to use information about the attributes of objects that are planned to be stored in the database; apply relational algebra methods to these attributes and create a database schema that satisfies the normal form.

The methodology for preparing input data for applying the method of searching for the optimal form of a document database is shown in Figure 1.

Figure 1 shows that if the final result should be a document database, then for optimal performance of query execution it is necessary to apply our two methods sequentially to the set of attributes of database objects:

- Determination of the optimal number of collections [25];
- Determination of the internal structure of documents in the collection: the number and order of embedded documents. We describe this method in this article.

These two methods are applicable when translating a relational database into a document database, consolidation of a relational database with a document database, document database restructuring, and creating a new document database.

To determine the need to create embedded documents, it is better if the scheme is initially reduced to normal form using relational algebra. This will allow compromising between query performance and the minimum amount of memory for storing data.

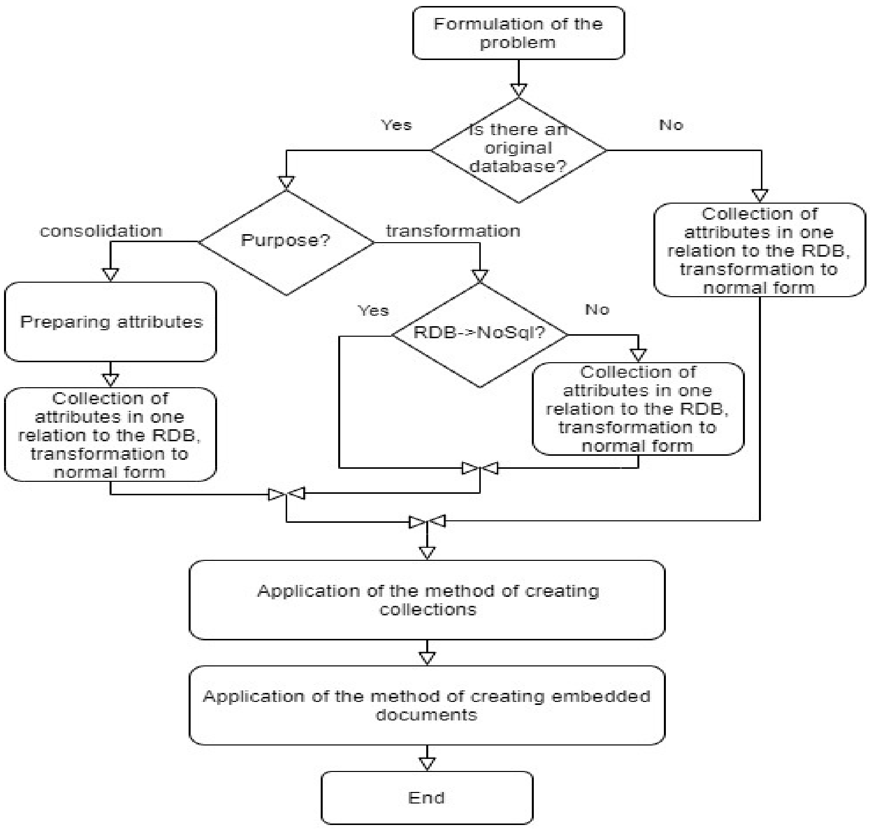

Formulation of the problem
Is there an original database?
Yes
No
Purpose?
consolidation
transformation
Collection of attributes in one relation to the RDB, transformation to normal form
Preparing attributes
Collection of attributes in one relation to the RDB, transformation to normal form
RDB->NoSql?
Yes
No
Collection of attributes in one relation to the RDB, transformation to normal form
Application of the method of creating collections
Application of the method of creating embedded documents
End

**Figure 1.** The scheme of creating input data for applying the method of building embedded documents.

To determine the structure of embedded documents, it is necessary to have information about the relationships between objects. One way to obtain this information is to search for functional dependencies between the attributes of objects, then construct a relational database schema and bring the schema to normal form using the relational algebra [44]. At the same time, the data remain in the same form; only the schemes are studied. Therefore, this process is not time-consuming and does not affect data. The relational database schema is intended in this case to describe the relationships between objects in order to understand which documents should be embedded.

## 5. Methods for Determining the Structure of Embedded Documents in the Database

The method for determining the number and order of embedded documents is based on the type of relationships between tables in the relational model. Depending on the type of link, it is necessary to apply a specific rule to create a document in the collection.

The first step in applying this method as shown in Figure 1 it is necessary to collect all the attributes that are in the document database into one collection (an analog of many fields of one table). Then, convert the schema of this table to normal form (preferably BCNF). The input to the search for the structure and number of collections in the document database is a set of these tables.

Below we consider the main types of relationships (one-to-one «$1-1$», one-to-many «$1-M$», many-to-many «$M-M$») for two tables and three tables. The rules for the three tables are based on the rules for two tables. The rules for a model of more than three tables will be based on the rules for two and three tables, so it makes no sense to consider them.

We show that when applying the method for determining the number of collections from [25], embedded documents can only exist when there is a $1-M$ relationship between tables whose attributes are included in the collection. Therefore, for two tables, only one rule is deduced.

When determining the form of embedded documents based on three tables, the sequence of relationships between the tables and the number of primary and secondary tables is important.

### *5.1. Defining Embedded Documents for Two Tables*

This subsection presents the rules for determining the form of embedded documents for document database collections consisting of documents compiled on the basis of a two-table relational database schema. At the beginning, the necessary information on input data is given, on the basis of which a decision is made on the need and form of embedded documents, then the rules and an explanation of these rules are given.

Input data:

- Two tables of the relational model, on the basis of which a non-relational document database schema is created—$T_1$ and $T_2$:

$$T_1\{T_{11},\ T_{12},\ \ldots, T_{1k}\}$$

$$T_2\{T_{21},\ T_{22},\ \ldots, T_{2n}\}$$

  where $T_{ij}$ is the $j$-th field of the $i$-th table, $k$ is the number of fields in relation to $T_1$ and $n$ is the number of fields in the table $T_2$.
- According to the method of determining collections in the document database, let a collection of documents be obtained (the structure of these documents depends on the structure of the queries):

$$Q\{T_{11},\ \ldots, T_{1m},\ T_{21},\ \ldots, T_{2r}\},\ m \le k,\ r \le n$$

- Tables $T_1$ and $T_2$ have $1-M$ relationships: $\boxed{T_1}_{1-\mathrm{M}}\boxed{T_2}$
- The tables have keys that belong to this collection:

$$T_1 :\ T_{11},\ T_{12},\ \ldots, T_{1s1},\ \text{where } 1 \le s1 \le m$$

$$(1) T_2 :\ T_{21},\ T_{22},\ \ldots, T_{2s2},\ \text{where } 1 \le s2 \le r$$

- There is a collection of queries $S_2$ to the collection $Q$:

$$S_{21}\Big\{T_{11},\ \ldots, T_{1s1}, \ldots, T_{1i1}, T_{21},\ \ldots, T_{2s2}, \ldots, T_{2j1}\Big\};$$

$$\ldots$$

$$S_{2t}\Big\{T_{11},\ \ldots, T_{1st}, \ldots, T_{1it}, T_{21},\ \ldots, T_{2s2}, \ldots, T_{2jt}\Big\};$$

  and the set of queries $S_1$ to the attributes of the table $T_2$ from the collection $Q$:

$$S_{11}\{\ldots, T_{2i1}, \ldots\};$$

$$\ldots$$

$$S_{1r}\{\ldots, T_{2ir}, \ldots\}.$$

Queries in the sets $S_1$ and $S_2$ are queries only for data selection, i.e., in terms of SQL, queries of the form “select ... from . . . where ...”. These queries can have multi-level subqueries, there can be operations of aggregation, grouping, sorting. In them, in terms of SQL, there can also be operations of joining, intersecting, and subtracting tables, since the method [25] was previously applied. Under database queries are not considered database queries to delete, modify, and insert data.

- $T'_2(S_2)$ is the set of all the attributes involved in the queries $S_2$:

  $$T'_2(S_2) = \left\{T_{21}, \ldots, T_{2s2}, \ldots, T_{2j1}\right\} \cup \ldots \cup \left\{T_{21}, \ldots, T_{2s2}, \ldots, T_{2jt}\right\} = \left\{T_{21}, \ldots, T_{2s2}, \ldots, T_{2j1}, \ldots, T_{2jt}\right\}$$

- Total queries $S_1$ equal to $V_s$,

  Output data: Collection $Q' \approx Q$.
  Rule 1. An embedded document should consist of attributes of the set:

$$T'_2 = T'_2(S_2) \cup \overset{Vs}{\underset{i=1}{\cup}} (S_{1i} - T'_2(S_2))_{S_{1i} - T'_2(S_2) \neq \varnothing \text{ and } S_{1i} - T'_2(S_2) \neq S_{1i}},$$

and the new structure of the $Q$ collection has the form:

$$Q'\{T_{11}, \ldots, T_{1s1}, \ldots, T_{1m}, T'_2 : \{T'_2(S_2) \cup \\ \cup \bigcup_{i=1}^{Vs} (S_{1i} - T'_2(S_2))_{S_{1i} - T'_2(S_2) \neq \varnothing \text{ and } S_{1i} - T'_2(S_2) \neq S_{1i}}\}, T''_2 : \left\{ \bigcup_{i=1}^{Vs} (S_{1i})\ _{S_{1i} \cap T'_2(S_2) = \varnothing} \right\}\} \quad (1)$$

where $T'_2$ and $T''_2$ are the names of the new keys for the embedded documents.

Explanation. The principle of creating embedded documents: to group those attributes that are in the queries executed simultaneously to two tables. These attributes are the set $T'_2(S_2)$. However, the remaining attributes from table $T_2$, which are in the collection $Q$, are obviously those attributes that are included only in queries to table $T_2$, i.e., these are attributes from the sets $S_1$. Among the sets $S_1$ there may be sets that contain the attributes $T'_2(S_2)$. To separate them from the embedded document is to complicate the query. Therefore, they must be subtracted from the sets $S_1$. Thus, the set $(S_{1i} - T'_2(S_2))$, these are the attributes from the $i$-th query $S_1$, which are only in queries to the table $T_2$ and at the same time contain attributes from queries $S_2$ if $S_{1i} - T'_2(S_2) \neq \varnothing$ *and* $S_{1i} - T'_2(S_2) \neq S_{1i}$. In total, such attributes will be $\overset{Vs}{\underset{i=1}{\cup}} (S_{1i} - T'_2(S_2))_{S_{1i} - T'_2(S_2) \neq \varnothing \text{ and } S_{1i} - T'_2(S_2) \neq S_{1i}}$. These attributes must be added to the embedded document along with the attributes $T'_2(S_2)$.

If attributes from the set $\{T_{21}, \ldots, T_{2r}\}$ in the $Q$ collection that are not included in the embedded document are left in the main document, this means increasing the amount of data in the $Q$ collection. Therefore, they must be collected in a separate subdocument. These attributes can be included in the queries of the set $S_1$ along with other attributes that are included in the first embedded document. These attributes are set: $T''_2 = \overset{Vs}{\underset{i=1}{\cup}} (S_{1i})\ _{S_{1i} \cap T'_2(S_2) = \varnothing}$.

Thus: the attributes from (1) will be included in the embedded document.

Note 1. If $T''_2 = \overset{Vs}{\underset{i=1}{\cup}} (S_{1i})\ _{S_{1i} \cap T'_2(S_2) = \varnothing} = \varnothing$, then the new collection structure $Q$ will have the form: $Q'\{T_{11}, \ldots, T_{1s1}, \ldots, T_{1m}, T'_2 : \{T_{21}, \ldots, T_{2r}\}\}$. It is this option that is often implemented when translating tables of a relational database with $1 - M$ relationships in MongoDB according to the principle: all tables must be assigned one collection.

Note 2. Above, only the relation of tables of type 1-$M$ was considered. If the relational database schema initially satisfied the normal form of BCNF, 3NF, or 4NF, then there can be no other relationships between tables $T_1$ and $T_2$. If for some reason, such as the denormalization of the relational database schema, this relationship exists, then:

- There can be no embedded documents for relationship between tables of the database of type $1 - 1$, due to the correspondence to each record from table $T_1$ of a single record from table $T_2$.
- For an $M - M$ relationship, embedded documents are built on the same principle as for $1 - M$ relationship.

## *5.2. Defining Embedded Documents for Three Tables Not Interconnected Sequentially*

This subsection presents the rules for determining the form of embedded documents for document database collections based on a three-table relational database schema. In this database schema, one table is the main table, and two other tables are associated with this table.

Input data:

- Three tables of the relational model, on the basis of which the non-relational document database schema is built: $T_1$, $T_2$, and $T_3$:

$$T_1\{T_{11},\ T_{12},\ \ldots,T_{1k}\}$$

$$T_2\{T_{21},\ T_{22},\ \ldots,T_{2n}\}$$

$$T_3\{T_{31},\ T_{32},\ \ldots,T_{3m}\}$$

  where $T_{ij}$ is the $j$-th field of the $i$-th table, $k$ is the number of fields in relation to $T_1$, $n$ is the number of fields in the table $T_2$, and $m$ is the number of fields in the table $T_3$.
- By the method of defining collections in a document database, a collection has been obtained:

$$Q\{T_{11},\ \ldots,T_{1k'},\ T_{21},\ \ldots,T_{2n'},T_{31},\ \ldots,T_{3m'}\},\ k' \le k, n' \le n,\ m' \le m$$

- Tables $T_1$, $T_2$, and $T_3$ have relationships of type $1-M$: $\boxed{T_3}_{\mathrm{M-1}}\boxed{T_1}_{\mathrm{1-M}}\boxed{T_2}$ or more schematically in Figure 2.
- In the tables the keys belonging to this collection are defined:

$$T_1:\ T_{11},\ T_{12},\ \ldots,T_{1s1},\ where\ 1 \le s1 \le k'$$

$$T_2:\ T_{21},\ T_{22},\ \ldots,T_{2s2},\ where\ 1 \le s2 \le n'$$

$$T_3:\ T_{31},\ T_{32},\ \ldots,T_{3s3},\ where\ 1 \le s3 \le m'$$

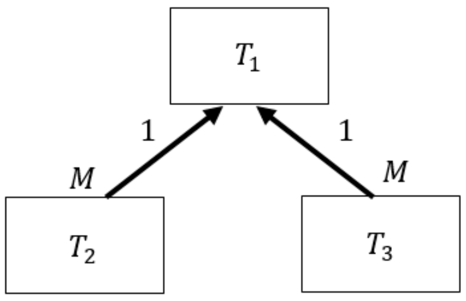


**Figure 2.** The scheme of relationships between tables $T_1$, $T_2$, and $T_3$ for the case when the first table is the main, second, and third tables are secondary, associated with the first table.

Output data: Collection $Q' \approx Q$.

Rule 2. If $S_1\{T_{11},\ \ldots,T_{1s1},\ldots,T_{1i}\}$ is the query that refers only to the attributes of one table, for Example $T_1$, then from Section 5.1, it follows that there cannot be embedded documents in this collection.

Rule 3. If $S_2\left\{T_{11},\ \ldots,T_{1s1},\ldots,T_{1i},T_{21},\ \ldots,T_{2s2},\ldots,T_{2j}\right\}$ is the query that refers to the attributes of two tables $T_1$, $T_2$ or $T_1$, $T_3$, then in Section 5.1, it is shown that the new structure of the collection $Q$ will have the form (1).

Rule 4. If $S_3\left\{T_{11}, \ldots, T_{1s1}, \ldots, T_{1i}, T_{21}, \ldots, T_{2s2}, \ldots, T_{2j}, T_{31}, \ldots, T_{3s3}, \ldots, T_{3r}\right\}$ is the database query that refers to the attributes of all three tables $T_1$, $T_2$, and $T_3$, then the new structure of the collection $Q$ will have the form:

$$\begin{gathered} Q'\{T_{11}, \ldots, T_{1m}, T'_2 : \{T'_2(S_2) \cup \\ \left.\cup \bigcup_{i=1}^{Vs} (S_{1i} - T'_2(S_2))_{S_{1i} - T'_2(S_2) \neq \varnothing \textit{ and } S_{1i} - T'_2(S_2) \neq S_{1i}}\right\}, T''_2 : \left\{\bigcup_{i=1}^{Vs} (S_{1i})_{\ S_{1i} \cap T'_2(S_2) = \varnothing}\right\}, T'_3 : \{T'_3(S_3) \cup \\ \left.\cup \bigcup_{i=1}^{Vs} (S_{1i} - T'_3(S_3))_{S_{1i} - T'_3(S_3) \neq \varnothing \textit{ and } S_{1i} - T'_3(S_3) \neq S_{1i}}\right\}, T''_3 : \left\{\bigcup_{i=1}^{Vs} (S_{1i})_{\ S_{1i} \cap T'_3(S_3) = \varnothing}\right\}\} \end{gathered} \tag{2}$$

where $T'_2$, $T'_3$ are the names of new keys for embedded documents; attributes $T'_2(S_3)$—these are all attributes of the table $T_2$ included in queries of type $S_3$ for the collection $Q$; attributes $T'_3(S_3)$ are all attributes of the table $T_3$ included in queries of type $S_3$ to the collection $Q$, $V_s$ is the number of queries of type $S_1$, $T''_2$ is the set of attributes that are not included in $T'_2$ of all the attributes of table $T_2$ included in the collection $Q$, $T''_3$ is the set of attributes that are not included in $T'_3$ of all the attributes of table $T_3$ included in the collection $Q$.

## *5.3. Defining Embedded Documents for Three Tables Interconnected Sequentially*

This section presents the rules for determining the form of embedded documents for document database collections based on a three-table relational database schema. Unlike the previous section in this database schema, the tables are connected in series: the first table is the main one in conjunction with the second table, the second table is the main one in conjunction with the third table.

Input data:

- Three tables of the relational model, on the basis of which the non-relational document database schema is built: $T_1$, $T_2$, and $T_3$:

$$T_1\{T_{11}, T_{12}, \ldots, T_{1k}\}$$

$$T_2\{T_{21}, T_{22}, \ldots, T_{2n}\}$$

$$T_3\{T_{31}, T_{32}, \ldots, T_{3m}\}$$

  where $T_{ij}$ is the $j$-th field of the $i$-th table, $k$ is the number of fields in relation to $T_1$, $n$ is the number of fields in the table $T_2$, and $m$ is the number of fields in the table $T_3$.
- By the method of defining collections in a document database, a collection has been obtained:

$$Q\{T_{11}, \ldots, T_{1k'}, T_{21}, \ldots, T_{2n'}, T_{31}, \ldots, T_{3m'}\},\ k' \leq k, n' \leq n,\ m' \leq m$$

- Tables $T_1$, $T_2$, and $T_3$ have relationships of type $1 - M$: $\boxed{T_3}_{1-M}\boxed{T_1}_{1-M}\boxed{T_2}$ or more schematically in Figure 3.
- In the tables the keys belonging to this collection are defined:

$$T_1 :\ T_{11}, T_{12}, \ldots, T_{1s1}, \textit{ where } 1 \leq s1 \leq k'$$

$$T_2 :\ T_{21}, T_{22}, \ldots, T_{2s2}, \textit{ where } 1 \leq s2 \leq n'$$

$$T_3 :\ T_{31}, T_{32}, \ldots, T_{3s3}, \textit{ where } 1 \leq s3 \leq m'$$

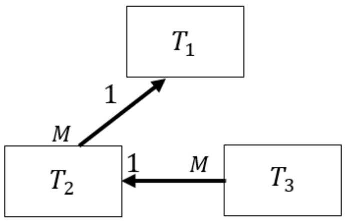


**Figure 3.** The relationship between the tables $T_1$, $T_2$ and $T_3$ for the case when: (1) first table is the main, (2) the second table is simultaneously secondary for the first table and the main for the third table, and (3) the third table is secondary, related to the second table.

Output data: Collection $Q' \approx Q$.

Consider the possible query options for the collection $Q$:

- $S_1\{T_{11}, \ldots, T_{1s1}, \ldots, T_{1i}\}$ is the database query that refers only to the attributes of one table, for example, $T_1$.

This case is considered in Section 5.1 when defining embedded documents for a collection constructed from the attributes of two tables and it is shown that there cannot be embedded documents in this collection.

- $S_2\left\{T_{11}, \ldots, T_{1s1}, \ldots, T_{1i}, T_{21}, \ldots, T_{2s2}, \ldots, T_{2j}\right\}$—a query that refers to the attributes of two tables $T_1$, $T_2$, or $T_2$, $T_3$.

This case is considered in Section 5.1 when defining embedded documents for a collection built from the attributes of two tables and it is shown that the new structure of the collection $Q$ will take the form according to Rule 1.

- $S_3\left\{T_{11}, \ldots, T_{1s1}, \ldots, T_{1i}, T_{21}, \ldots, T_{2s2}, \ldots, T_{2j}, T_{31}, \ldots, T_{3s3}, \ldots, T_{3r}\right\}$—a query that refers to the attributes of all three tables $T_1$, $T_2$ and $T_3$.

This case is a generalization of Case B from two tables to three tables. In this case, the new structure of the $Q$ collection will have the form:

Rule 5.

$$\begin{aligned} Q'\Bigg\{ & T_{11}, \ldots, T_{1m}, T'_2 : \Bigg\{ T'_2(S_2) \cup \cup \bigcup_{i=1}^{Vs} (S_{1i} - T'_2(S_2))_{S_{1i}-T'_2(S_2)\neq\varnothing \textit{ and } S_{1i}-T'_2(S_2)\neq S_{1i}}, \\ & T'_3 : \Bigg\{ T'_3(S_3) \cup \bigcup_{i=1}^{Vs} (S_{1i} - T'_3(S_3))_{S_{1i}-T'_3(S_3)\neq\varnothing \textit{ and } S_{1i}-T'_3(S_3)\neq S_{1i}} \Bigg\}\Bigg\}, \\ & T''_2 : \Bigg\{ \bigcup_{i=1}^{Vs} (S_{1i})\ _{S_{1i}\cap T'_2(S_2)=\varnothing} \Bigg\}, T''_3 : \Bigg\{ \bigcup_{i=1}^{Vs} (S_{1i})\ _{S_{1i}\cap T'_3(S_3)=\varnothing} \Bigg\}\Bigg\} \end{aligned} \quad (3)$$

where $T'_2$, $T'_3$ are the names of new keys for embedded documents; attributes $T'_2(S_3)$—these are all attributes of the table $T_2$ included in queries of type $S_3$ for the collection $Q$; attributes $T'_3(S_3)$ are all attributes of the table $T_3$ included in queries of type $S_3$ to the collection $Q$, $V_s$ is the number of queries of type $S_1$, $T''_2$ is the set of attributes that are not included in $T'_2$ of all the attributes of table $T_2$ included in the collection $Q$, and $T''_3$ is the set of attributes that are not included in $T'_3$ of all the attributes of table $T_3$ included in the collection $Q$.

- $S_4\{T_{11}, \ldots, T_{1s1}, \ldots, T_{1i}, T_{31}, \ldots, T_{3s3}, \ldots, T_{3r}\}$—a query that refers to the attributes of two tables $T_2$ and $T_3$.

Because database query to the attributes of tables $T_1$ and $T_3$ can be performed only through the attributes of the table $T_2$, then this case as a result reduces to **Case C**).

Note 3. If the collection $Q$ is created for more than three tables, then the form of embedded documents is determined according to Rules 1–5.

The scheme for applying the rules for the three tables is given in Appendix A.

## 6. Testing the Effectiveness of the Methodology for Determining the Structure of Embedded Documents in a Document Database

To test the effectiveness of the developed methodology for determining the structure of embedded documents in document databases, we performed queries to databases filled with the same attribute values, but having different collection structures: with and without embedded documents.

Testing was carried out on a personal computer: Intel (R) Core (TM) i7-8700 CPU @ 3.20GHz, with Windows operating system, MySQL Community Server Version 8.0.17, MongoDB Community Server v. 4.0.5.

During the experiments, all additional functions, such as cache, indexes, and multi-threading, were disabled. The used data is synthetic. Each database query is executed 25 times. The difference in the execution time of each of 25 times for one query varied within e = 0.01 ms (for example: t1 = 0.3526 ms, t2 = 0.3401 ms, etc.). The execution time of each database query, given in all tables below, is the average execution time of a given query 25 times.

Below are three examples from the testing.

Example 1. To establish relationships between objects in the MongoDB database, it was built a schema of a relational database consisting of three tables, one of which is the main and two associated with it (Figure 4). Data volume: main table $T_1$—10,000 records, related tables: $T_2$—100,000 records, $T_3$—100,000 records.

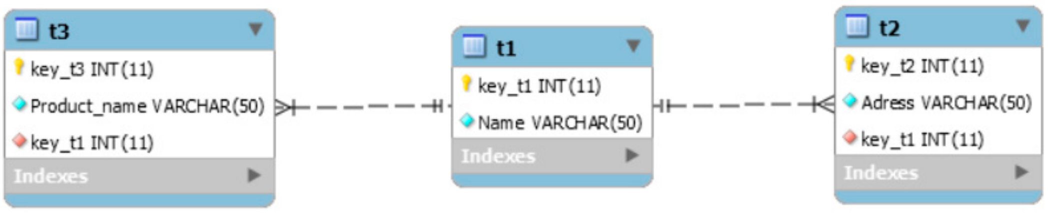


**Figure 4.** An example of a database of three tables. This is the case when the first table is the main, second, and third tables are secondary, associated with the first table.

The database schema shown in Figure 4 has two MongoDB databases in accordance with it:

(1). A database consisting of one collection $Q$, including all fields of tables $T_1$, $T_2$ and $T_3$. A fragment of this collection is shown in Figure 5.

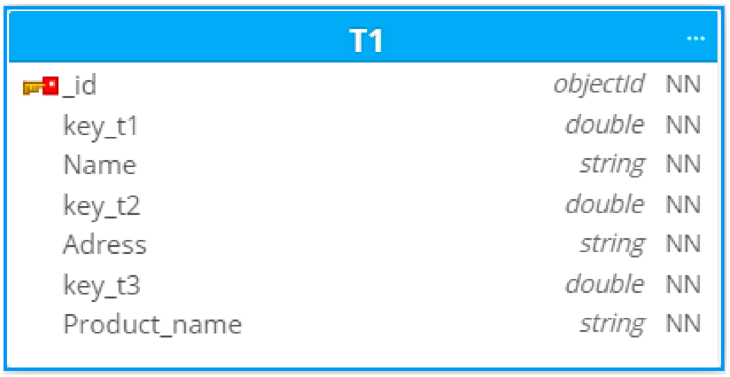
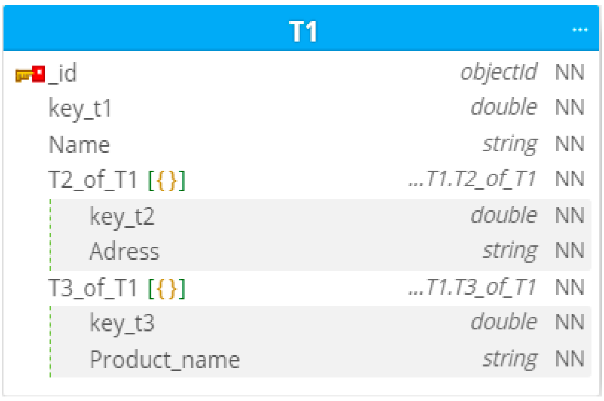


**Figure 5.** A single MongoDB collection without embedded documents.

(2). The database, built in accordance with the Rules 1–4 described in Section 5. The new structure of the collection *Q′* will have the form:

*Q′{key_t1,Name,T2_of_T1{key_t2, Adress},T3_of_t1{key_t3,Product_name}}*

The structure of this collection is shown in Figure 6.



| T1 | | |
|---|---|---|
| _id | *objectId* | NN |
| key_t1 | *double* | NN |
| Name | *string* | NN |
| T2_of_T1 [{}] | *...T1.T2_of_T1* | NN |
| key_t2 | *double* | NN |
| Adress | *string* | NN |
| T3_of_T1 [{}] | *...T1.T3_of_T1* | NN |
| key_t3 | *double* | NN |
| Product_name | *string* | NN |

**Figure 6.** New collection with sequentially embedded documents.

To test these two variants of the MongoDB database, eight queries were executed, differing in their structure. The average value of the query execution time is presented in Table 1 and in Figure 7.

**Table 1.** The speed of execution of queries to document databases: without embedded documents and with embedded documents.

| Query | Option 1 DB (without Embedded Documents), Time, ms | Option 2 DB (with Embedded Documents), Time, ms |
|---|---|---|
| Q1 | 0.1106 | 0.0698 |
| Q2 | 0.1026 | 0.0716 |
| Q3 | 0.142 | 0.124 |
| Q4 | 0.3526 | 0.01 |
| Q5 | 0.346 | 0.013 |
| Q6 | 0.3366 | 0.014 |
| Q7 | 0.3666 | 0.043 |
| Q8 | 0.369 | 0.041 |

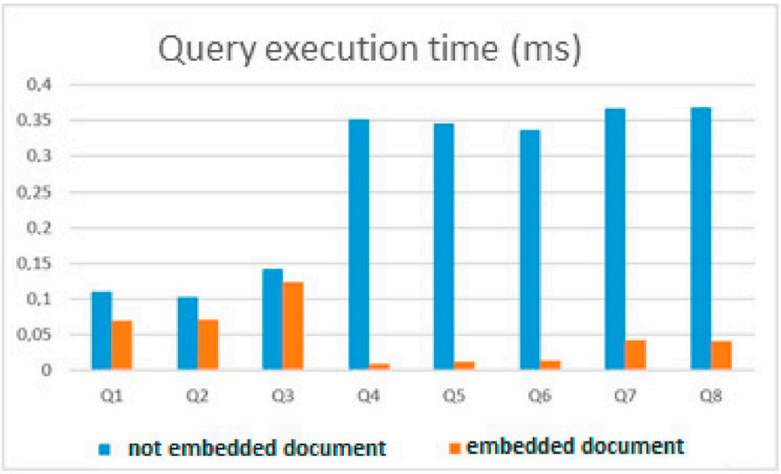


**Figure 7.** A diagram showing the execution time of eight queries to a database of two types: a single database and a database with embedded documents.

Example 2. To establish relationships between objects in the MongoDB database, it was built a schema of a relational database consisting of three tables connected in series (Figure 8). Data volume: main table $T_1$—10,000 records, $T_2$—60,000 records, $T_3$—100,000 records. The volume of $T_2$ in this example is less than in Example 1. This is because $T_2$ in Example 1 was a secondary table, but in this example, $T_2$ is a secondary table for $T_1$ and the main for $T_3$. Therefore, the data volume in it should be more than in $T_1$ and less than in $T_3$.

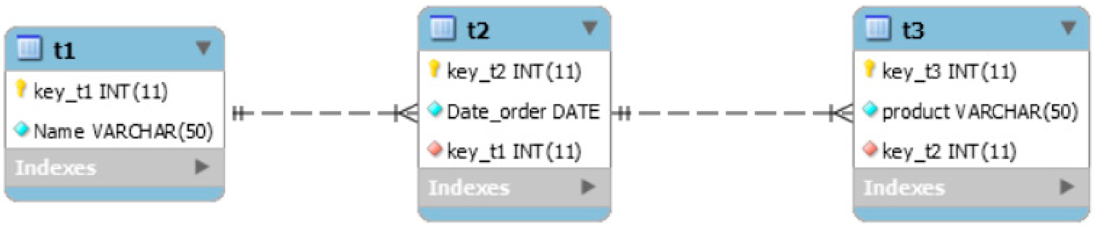


**Figure 8.** The case of sequentially related tables. This is case when first table is the main, the second table is simultaneously secondary for the first table and the main for the third table, and the third table is secondary, related to the second table.

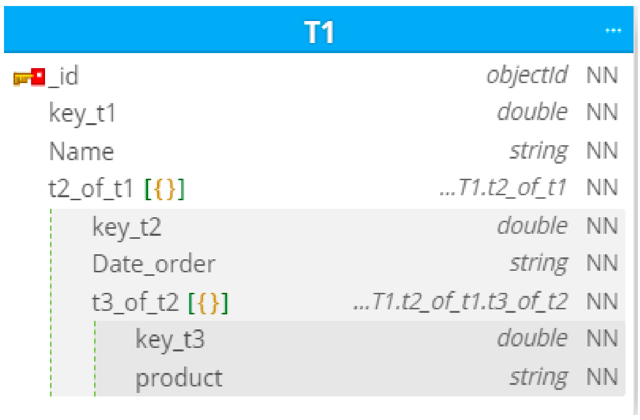
Let two MongoDB databases be set in accordance with the database schema shown in Figure 8:

(1). A database consisting of one collection $Q$, composed of all the fields of tables $T_1$, $T_2$, and $T_3$. A fragment of this collection is shown in Figure 8.

(2). The database, built in accordance with Rules 1–4 described in Section 5. The new structure of the collection $Q'$ will have the form:

*Q′{key_t1,Name,T2_of_T1:{key_t2, Adress,T3_of_T2:{key_t3,Product_name}}}*

The structure of this collection is shown in Figure 9.



| | | |
|---|---|---|
| _id | objectId | NN |
| key_t1 | double | NN |
| Name | string | NN |
| t2_of_t1 [{}] | ...T1.t2_of_t1 | NN |
| key_t2 | double | NN |
| Date_order | string | NN |
| t3_of_t2 [{}] | ...T1.t2_of_t1.t3_of_t2 | NN |
| key_t3 | double | NN |
| product | string | NN |

**Figure 9.** New collection with two levels of embedded documents.

To test these two variants of the MongoDB database, six queries were executed, differing in their structure. The average value of the query execution time is presented in Table 2 and in Figure 10.

**Table 2.** The speed of execution of queries to document databases: without embedded documents and with embedded documents.

| Query | Option 1 DB (without Embedded Documents), Time, ms | Option 2 DB (with Embedded Documents), Time, ms |
|---|---|---|
| Q1 | 0.108 | 0.07 |
| Q2 | 0.171 | 0.126 |
| Q3 | 0.162 | 0.052 |
| Q4 | 0.036 | 0.007 |
| Q5 | 0.038 | 0.027 |
| Q6 | 0.037 | 0.025 |

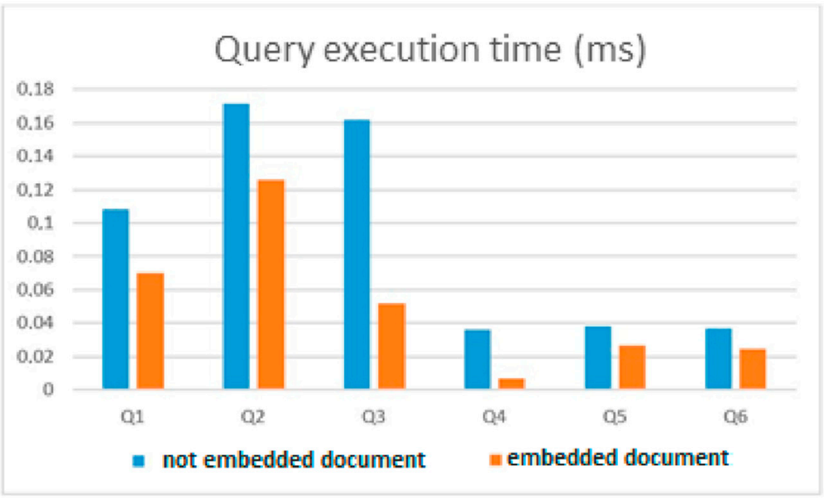


**Figure 10.** A diagram showing the execution time of six queries to a database of two types: a single database and a database with embedded documents.

Example 3. To establish relationships between objects in the MongoDB database, it was built a schema of a relational database consisting of four tables. Two tables are the main and two tables are secondary (Figure 11). Data volume: T1—15,000 records, T2—50,000 records, T3—110,000 records, and T4—200,000 records.

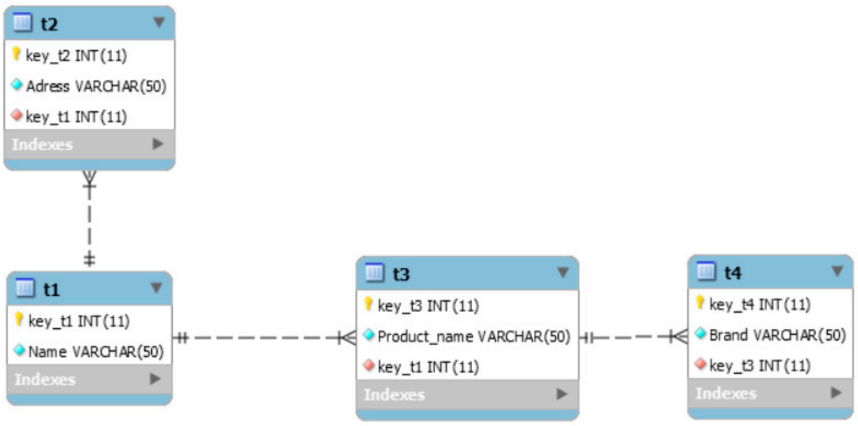


**Figure 11.** An example of a database of four tables.

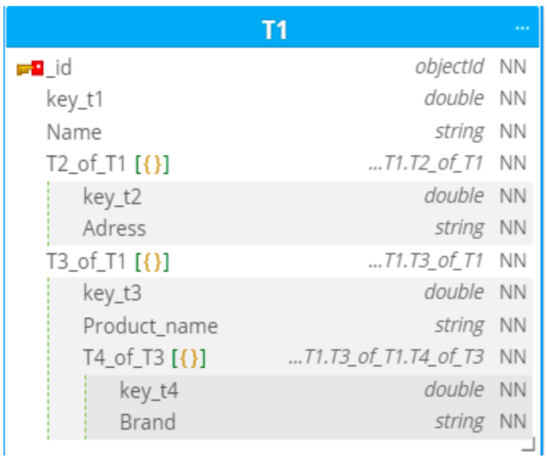
Let two MongoDB databases be set in accordance with the database schema shown in Figure 8:

(1). A database consisting of one collection $Q$, composed of all the fields of tables $T_1$, $T_2$, $T_3$ and $T_4$. A fragment of this collection is shown in Figure 8.

(2). The database, built in accordance with Rules 1–4 described in Section 5. The new structure of the collection *Q'* will have the form:

*Q'{key_t1,Name,T2_of_T1:{key_t2,Adress},T3_of_T1:{key_t3,Product_name, T4_of_T3: {key_t4, Brand}}}}*

The structure of this collection is shown in Figure 12.



| | | |
|---|---|---|
| **T1** | | |
| _id | objectId | NN |
| key_t1 | double | NN |
| Name | string | NN |
| T2_of_T1 [{}] | ...T1.T2_of_T1 | NN |
| key_t2 | double | NN |
| Adress | string | NN |
| T3_of_T1 [{}] | ...T1.T3_of_T1 | NN |
| key_t3 | double | NN |
| Product_name | string | NN |
| T4_of_T3 [{}] | ...T1.T3_of_T1.T4_of_T3 | NN |
| key_t4 | double | NN |
| Brand | string | NN |

**Figure 12.** The new collection with sequential and multi-level embedded documents.

To test these two variants of the MongoDB database, eight queries were executed, differing in their structure. The average value of the query execution time is presented in Table 3 and in Figure 13. Examples of queries are given in Appendix B.

**Table 3.** The speed of execution of queries to document databases: without embedded documents and with embedded documents.

| Query | Option 1 DB (without Embedded Documents), Time, ms | Option 2 DB (with Embedded Documents), Time, ms |
|---|---|---|
| Q1 | 0.078 | 0.055 |
| Q2 | 0.175 | 0.163 |
| Q3 | 0.169 | 0.135 |
| Q4 | 0.198 | 0.08 |
| Q5 | 0.234 | 0.109 |
| Q6 | 0.199 | 0.012 |
| Q7 | 0.209 | 0.022 |
| Q8 | 0.206 | 0.026 |

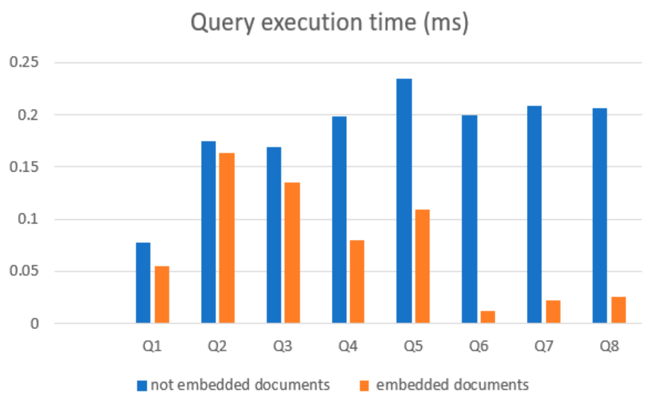
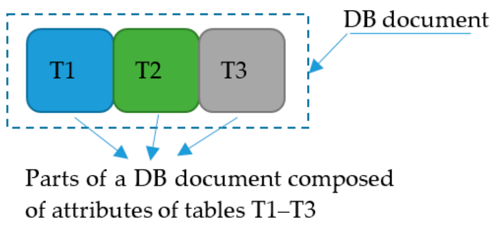
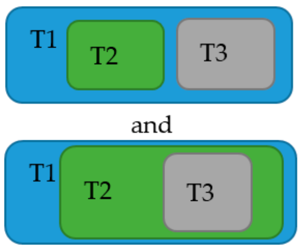
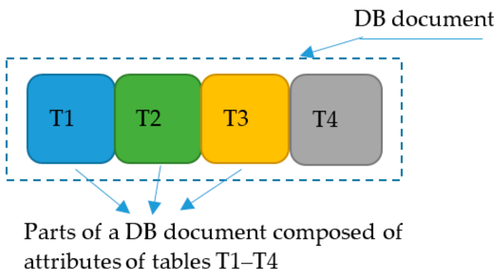
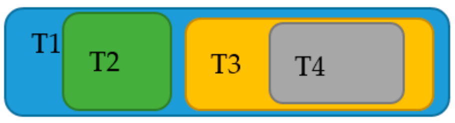


**Figure 13.** A diagram showing the execution time of six queries to a database of two types: a single database and a database with embedded documents.

We have tested the method on databases of various contents and structure. In this article, we have demonstrated three of the most revealing examples that can be schematically represented as follows (Table 4):

**Table 4.** Schematic representation of a document in a document database composed of attributes corresponding to the fields of different tables.

| Original Document Database Schema | A New Scheme with Embedded Documents, Obtained Taking into Account the Structure of Database Queries and Relationships between Objects |
|---|---|
|   DB document<br>T1<br>T2<br>T3<br>Parts of a DB document composed of attributes of tables T1–T3 |  |
|   DB document<br>T1<br>T2<br>T3<br>T4<br>Parts of a DB document composed of attributes of tables T1–T4 |  |

For databases whose documents contain attributes of more than two tables, many other options for embedded documents are possible. The method described above allows uniquely determining the form of embedding documents.

The diagrams in Figure 8, Figure 10, and Figure 13 show that depending on the structure of the queries, the speedup of the query execution in the database created by the proposed method (this is a database with embedded documents in the figures) can be from 10% to 70%, but all queries with embedded documents were faster. This can be explained by the fact that the volume of the collection when using embedded documents is much smaller than in the case of building a collection without embedded documents. For example, in Example 1, when transferring data from a MySQL database to MongoDB, a new collection with embedded documents consisted of 20,000 documents and had a volume of ≈20 MB, and a collection without embedded documents consisted of 100,770 documents and had a volume of ≈138 MB. Therefore, it is very important to evaluate in which collections documents should be embedded and in which not. These examples show the effectiveness of the application of the methodology and rules for constructing collections described in Sections 4 and 5, taking into account the relationships between the objects in the database and the structure of the queries that are executed to them.

## 7. Conclusions

As a result of the research, a formalized method based on set theory was developed that allows automating the process of building a document database for a given set of properties of objects and the relationships between them. Information on the structure of database queries allows determining the number and composition of collections for the MongoDB database so that it is possible not to perform "join" operations in database queries. Taking into account the relationships between objects allows assessing the need to create embedded documents, their number and composition.

For example, let two tables be obtained from the results of constructing a relational schema based on database attributes. Let each table have five attributes. Then, it is possible to create one collection of nine attributes (taking into account the relationship of the tables). It is possible to create two collections of four and five attributes. It is possible to create three collections of three attributes. It is possible to create four collections of two and three attributes. There are other options. It is possible to create collections with embedded documents, or without embedded documents. In total, we get more than 10 options. Which to choose? It all depends on what queries will be performed most often on tables and how tables are related. This is what our method does. The method unambiguously answers these questions: how many and which collections.

We thoroughly tested our method on document databases of various structures with different relationships between tables. In each test, we measured the average query execution time. We showed some of these measurements in Section 6. Having analyzed all the measurements of query execution time, we conclude that organizing a database using the method proposed in this article allows us to speed up query execution by 10–50% compared with other forms of database organization. The percentage of increasing the speed of query execution depends on two parameters: the initial database schema (before applying the method described in this article to this schema) and the structure of the query.

This method can be used when translating a database from relational format to MongoDB format, to define collections in a new MongoDB database, and when consolidating databases of various structures.

In general, the proposed method can be applied to:

- Creating an effective structure of a new document database;
- Data translation from a relational database to a document database;
- Data translation from an arbitrary database into a document database;
- Database consolidation.

There are other possibilities for applying this method, such as creating temporary collections, etc.

It should also be noted that the data is transferred to the new structure last, and this is usually done using software modules or using the tuple algebra, which is partially described in [44].

## 8. Further Research

As shown in the examples, the method works well for all queries. However, for some queries, the method works better, while for others it is worse. For example, it can be seen from Appendix B that Q3 and Q6 queries are performed on tables T1, T2, and T3. The query Q3 in the new form of the database does not work much better than in the previous one. The query Q6 in the new form is much faster than in the previous one. We have not yet found an explanation for this fact. This is the subject of our further research.

The main problem in document databases was that there was no formalized method that would unequivocally give an answer to the administrator about the optimal form and structure of documents taking into account database queries. Among NoSQL databases, there are databases of the type of a column family, which are also used to store related data and are also oriented to specific database queries. Therefore, for these databases, the decision on the optimal organization of column families is also important. At the moment, we are conducting research on the adaptation of the proposed method to the databases of the column family (using Casandra as an example).

When restructuring the database, the next step after defining a new database structure for the database administrator is to rewrite the queries from the old form to the new form (if these queries are in stored procedures or scripts). To speed up the process of converting a database from one form to another, we have created an approach to the automatic conversion of queries. The input for the query conversion method is information about the form and structure of the document, which is output by the method described in this article. We are currently testing this method.

One way to improve query performance is to create indexes. When changing the form of organization of the database, the use of indexes to the previous attributes may not give their initial effectiveness. Therefore, it is necessary to conduct research to supplement the method (presented in this article) with information about indexes. This we plan to do in a future study.

**Author Contributions:** Conceptualization, Methodology and Writing—original draft, Y.S. Software, Formal analysis, Visualization, M.H. All authors have read and agreed to the published version of the manuscript.

**Funding:** The paper was prepared in Saint-Petersburg Electrotechnical University (LETI) and is supported by the Agreement No. 075-11-2019-053 dated 20.11.2019 (Ministry of Science and Higher Education of the Russian Federation, in accordance with the Decree of the Government of the Russian Federation of April 9, 2010 No. 218)

**Conflicts of Interest:** The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Appendix A

The scheme for determining the form of embedded documents based on three tables is presented in Figure A1.

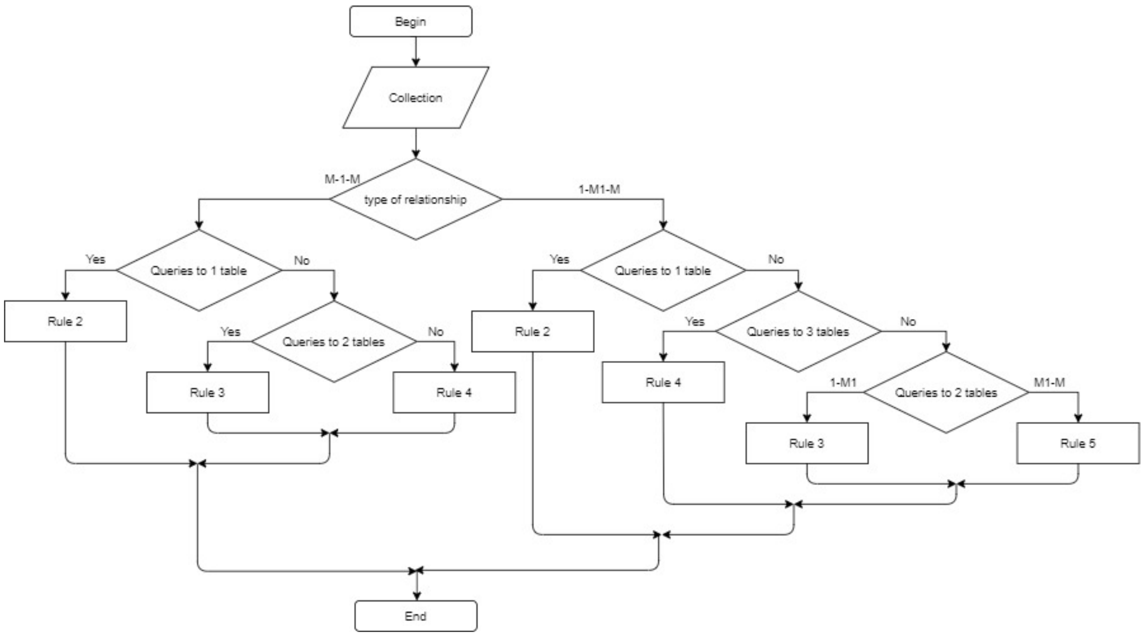


**Figure A1.** A diagram showing the execution time of six queries to a database of two types: a single database and a database with embedded documents.

## Appendix B

Table A1 shows examples of queries that were declared in Example 3 in this article and in accordance with which the database was restructured.

**Table A1.** Schematic representation of a document in a document database composed of attributes corresponding to the fields of different tables.

| Query | Option 1 DB (without Embedded Documents) | Option 2 DB (with Embedded Documents) |
|---|---|---|
| Q1 | Display all the data of people with the given last name and address from tables T1, T2: SQL: select name, address from t1, t2 where t1.key_1 = t2.key_1 | |
| | db.T1.find({},{‘Name’:1,’T2_of_T1.Adress’: 1,’_id’:0}) | db.getCollection(“T1”).find({},{‘Name’:1, ‘Adress’:1,’_id’:0}) |
| Q2 | Display a list of all objects with the given values of the attributes “product” and “brand” from tables T2, T4: SQL: select product_name, brand from t2, t4 where t2.key_t2 = t4.key_t2 | |
| | db.T1.find({},{‘T3_of_T1.Product_name’:1, ‘T3_of_T1.T4_of_T3.Brand’: 1,’_id’:0}) | db.getCollection(“T1”).find({},{‘Product_name’:1, ‘Brand’:1,’_id’:0}) |
| Q3 | List all people with the given attributes “Last name”, “product” and “brand” from tables T1, T3, T4: SQL: select name, product, brand from t1,t3,t4 where t1.key_t1 = t3.key_t1 and t3.key_t3 = t4.key_t3 | |
| | db.T1.find({},{‘Name’:1,’T3_of_T1.Product_name’:1,’T3_of_T1.T4_of_T3.Brand’: 1,’_id’:0}) | db.getCollection(“T1”).find({},{‘Name’:1,’Product_name’:1, ‘Brand’:1,’_id’:0}) |
| Q4 | Find all the addresses of this person Charity Dickerson: SQL: select adress from t1,t2 where t1.key_t1 = t2.key_t1 and name = ‘charity dickerson’ | |
| | db.T1.find({‘Name’:’Charity Dickerson’}, {‘T2_of_T1.Adress’:1,’_id’:0}) | db.getCollection(“T1”).find({‘Name’:’Charity Dickerson’},{‘Adress’:1,’_id’:0}) |
| Q5 | Search for all products that have the brand name: ‘Sureraquistor International Corp.’: SQL: select t3.product_name from t3,t4 where t3.key_t3 = t4.key_t3 and t4.brand = ‘sureraquistor international corp.’ | |
| | db.T1.find({‘T3_of_T1.T4_of_T3.Brand’:’Sureraquistor International Corp.’}, {‘T3_of_T1.Product_name’:1,’_id’:0}) | db.getCollection(“T1”).find({‘Brand’:’Sureraquistor International Corp.’},{‘Product_name’:1,’_id’:0}) |
| Q6 | Search all products and brands for the man ‘Mindy Garcia’ SQL: select Product_name, Brand from t1,t3,t4 where t1.key_t1 = t3.key_t1 and t3.key_t3 = t4.key_t3 and Name = ‘Mindy Garcia’ | |
| | db.T1.find({‘Name’:’Mindy Garcia’}, {‘T3_of_T1.Product_name’:1,’T3_of_T1.T4_of_T3.Brand’:1, ‘_id’:0}) | db.getCollection(“T1”).find({‘Name’:’Mindy Garcia’},{‘Product_name’:1,’Brand’:1,’_id’:0}) |
| Q7 | Search all products and customers at 662 South Second Drive: SQL: select Name, Product_name from t1,t3,t2 where t1.key_t1 = t2.key_t1 and t1.key_t1 = t3.key_t1 and Adress = ‘662 South Second Drive’ | |
| | db.T1.find( {‘T2_of_T1.Adress’:’662 South Second Drive’}, {‘T3_of_T1.Product_name’:1,’Name’:1, ‘_id’:0}) | db.getCollection(“T1”).find({‘Adress’:’662 South Second Drive’},{‘Product_name’:1,’Name’:1,’_id’:0}) |
| Q8 | Search for all products, addresses and customer brands of Sonia Chang: SQL: select Adress, Product_name, Brand from t1,t2,t3,t4 where t1.key_t1 = t2.key_t1 and t1.key_t1 = t3.key_t1 and t3.key_t3 = t4.key_t3 and Name = ‘Sonia Chang’ | |
| | db.T1.find({ ‘Name’: ‘Sonia Chang’}, {‘T3_of_T1.Product_name’:1,’T2_of_T1.Adress’:1,’T3_of_T1.T4_of_T3.Brand’:1,’_id’:0}) | db.getCollection(“T1”).find({‘Name’: ‘Sonia Chang’}, {‘Product_name’:1,’Adress’:1,’Brand’:1,’_id’:0}) |

## References

1. Padhy, R.P.; Patra, M.R.; Satapathy, S.C. RDBMS to NoSQL: Reviewing some next-generation non-relational database’s. *Int. J. Adv. Eng. Sci. Technol.* **2020**, *11*, 15–30.
2. Chen, J.; Lee, W. An Introduction of NoSQL Databases Based on Their Categories and Application Industrie. *Algorithms* **2019**, *12*, 106. [CrossRef]
3. Diogo MCabral, B.; Bernardino, J. Consistency Models of NoSQL Databases. *Future Internet* **2019**, *11*, 43. [CrossRef]
4. Storey, V.C.; Song, I.Y. Big data technologies and Management: What conceptual modeling can do. *Data Knowl. Eng.* **2017**, *108*, 50–67. [CrossRef]
5. Corbellini AMateos CZunino AGodoy, D.; Schiaffino, S. Persisting big-data: The NoSQL landscap. *Inf. Syst. J.* **2017**, *63*, 1–23. [CrossRef]
6. Makris ATserpes KAndronikou, V.; Anagnostopoulos, D. A classification of NoSQL data stores based on key design characteristics. *Procedia Comput. Sci.* **2016**, *97*, 94–103. [CrossRef]
7. Atzeni PBugiotti, F.; Rossi, L. Uniform access to NoSQL systems. *Inf. Syst. J.* **2014**, *43*, 117–133. [CrossRef]
8. Li, W.; Clifton, C. Semantic Integration in Heterogeneous Databases Using Neural Networks. In Proceedings of the 20th International Conference on Very Large Data Bases, Santiago, Chile, 12–15 September 1994; pp. 1–12.
9. Li WClifton, C.; Liu, S. Database Integration Using Neural Networks: Implementation and Experiences. *Knowl. Inf. Syst.* **2000**, *2*, 73–96.
10. Doan ADomingos, P.; Levy, A. Learning source description for data integration. *WebDB* **2000**, *81*, 2000.
11. Doan ADomingos, P.; Halevy, A.Y. Reconciling schemas of disparate data sources. *SIGMOD Rec.* **2001**, *30*, 509–520. [CrossRef]
12. Miller RJHaas, L.M.; Hernández, M.A. Schema Mapping as Query Discovery. In Proceedings of the 26th International Conference on Very Large Data Bases, Cairo, Egypt, 10–14 September 2000; pp. 77–88.

13. Milo, T.; Zohar, S. Using schema matching to simplify heterogeneous data translation. *Vldb* **1998**, *98*, 1–21.
14. Palopoli, L.; Saccà, D.; Ursino, D. An automatic technique for detecting type conflikts in database shemes. In Proceedings of the seventh international conference on Information and knowledge management, Yokohama, Japan, 22–23 November 1998; pp. 306–313.
15. Castano, S.; De Antonellis, V. A schema analysis and reconciliation tool environment for nheterogeneous databases, Proceedings. IDEAS'99. In Proceedings of the Int. Database Engineering and Application Symposium, Montreal, QC, Canada, 2–4 August 1999. Cat. No.PR00265.
16. Castano SDe Antonellis, V.; Di De Vimercati, S.C. Global viewing of heterogeneous data sources. *IEEE Trans. Knowl. Data. Eng.* **2001**, *13*, 277–297. [CrossRef]
17. Bergamaschi, S.; Castano, S.; Vincini, M.; Beneventano, D. Semantic integration of heterogeneous information sources. *Data Knowl. Eng.* **2001**, *36*, 215–249. [CrossRef]
18. Bergamaschi, S.; Castano, S.; Vincini, M. Semantic integration of semistructured and structured data sources. *SIGMOD Rec.* **1999**, *28*, 54–59. [CrossRef]
19. Pardede, E.; Rahayu, J.W.; Taniar, D. Mapping Methods and Query for Aggregation and Association Relationship in Object-Relational Database using Collection. In Proceedings of the 2004 IEEE International Conference on Information Technology: Coding and Computing (ITCC), Las Vegas, NV, USA, 5–7 April 2004; pp. 539–543.
20. Shichkina, Y.; Kupriyanov, M.; Shevsky, V. The Application of Graph Theory and Adjacency Lists to Create Parallel Queries to Relational Databases. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*; Springer: Berlin, Germany, 2018; Volume 10963, pp. 61–77.
21. Robinson, I.; Webber, J.; Eifrem, E. *Graph Databases: New Opportunities for Connected Data*; O'Reilly Media, Inc.: Sebastopol, CA, USA, 2015.
22. Chickerur, S. Comparison of Relational Database with Document-Oriented Database (MongoDB) for Big Data Applications. In Proceedings of the 8th International Conference on Advanced Software Engineering and Its Applications ASEA, Jeju Island, Korea, 25–28 November 2015; pp. 41–47.
23. Hanine, M.; Bendarag, A.; Boutkhoum, O. Data Migration Methodology from Relational to NoSQL Databases, World Academy of Science, Engineering and Technology. *Int. J. Comput. Electr. Autom. Control Inf. Eng.* **2015**, *9*, 2566–2570.
24. Li, X.; Ma, Z.; Chen, H. QODM: A Query-Oriented Data Modeling Approach for NoSQL Databases. In *Advanced Research and Technology in Industry Applications (WARTIA)*; IEEE: Piscataway, NJ, USA, 2014; pp. 338–345.
25. Ha, V.M.; Shichkina, Y.A.; Kostichev, S.V. Determination of the collection of collections for databases of the key-document type for a given set. *Comput. Tools Educ.* **2019**, *3*, 15–28. [CrossRef]
26. Zhao, G.; Huang, W.; Liang, S.; Tang, Y. Modeling MongoDB with Relational Model. In *Emerging Intelligent Data and Web Technologies (EIDWT)*; IEEE: Piscataway, NJ, USA, 2013; pp. 115–121.
27. Alotaibi, O.; Pardede, E. Transformation of Schema from Relational Database (RDB) to NoSQL Databases. *Data* **2019**, *4*, 148. [CrossRef]
28. Celesti, A.; Fazio, M.; Villari, M. A Study on Join Operations in MongoDB Preserving Collections Data Models for Future Internet Applications. *Future Internet* **2019**, *11*, 83. [CrossRef]
29. Rocha, L.; Vale, F.; Cirilo, E. A Framework for Migrating Relational Datasets to NoSQL. *Procedia Comput. Sci.* **2015**, *51*, 2593–2602. [CrossRef]
30. Liang, D.; Lin, Y.; Ding, G. Mid-model Design Used in Model Transition and Data Migration between Relational Databases and NoSQL Databases. In Proceedings of the IEEE International Conference on Smart City, Chengdu, China, 19–21 December 2015; pp. 866–869.
31. Hamid, S.; Rezapour, M.; Moradi, M.; Ghadiri, N. Performance evaluation of SQL and MongoDB databases for big e-commerce data. In Proceedings of the IEEE Pacific RIM Conference on Communications, Computers, and Signal Processing, Victoria, BC, Canada, 24–26 August 2015; pp. 1–7.
32. Feng, W.; Gu, P.; Zhang, C.; Zhou, K. Transforming UML Class Diagram into Cassandra Data Model with Annotations. In Proceedings of the IEEE International Conference on Smart City, Chengdu, China, 19–21 December 2015; pp. 798–805.

33. Karnitis, G.; Arnicans, G. Migration of Relational Database to Document-Oriented Database: Structure Denormalization and Data Transformation. In Proceedings of the 7th International Conference on Computational Intelligence, Communication Systems and Networks (CICSyN), Riga, Latvia, 3–5 June 2015; pp. 113–118.
34. Mason, R.T. NoSQL Databases and Data Modeling Techniques for a Document-oriented NoSQL Database. *Comput. Sci.* **2015**, *3*, 259–268.
35. Gu, Y.; Shen, S.; Wang, J.; Kim, J. Application of NoSQL Database MongoDB. In Proceedings of the IEEE International Conference on Consumer Electronics-Taiwan (ICCE-TW), Taipei, Taiwan, 6–8 June 2015; pp. 158–159.
36. Wang, S.; Li, G.; Yao, X.; Zeng, Y.; Pang, L.; Zhang, L. A Distributed Storage and Access Approach for Massive Remote Sensing Data in MongoDB. *ISPRS Int. J. Geo-Inf.* **2019**, *8*, 533. [CrossRef]
37. Marrara, S.; Pelucchi, M.; Psaila, G. Blind Queries Applied to JSON Document Stores. *Information* **2019**, *10*, 291. [CrossRef]
38. Qian, C.; Yi, C.; Cheng, C.; Pu, G.; Wei, X.; Zhang, H. GeoSOT-Based Spatiotemporal Index of Massive Trajectory Data. *ISPRS Int. J. Geo-Inf.* **2019**, *8*, 284. [CrossRef]
39. Višnjevac, N.; Mihajlović, R.; Šoškić, M.; Cvijetinović, Ž.; Bajat, B. Prototype of the 3D Cadastral System Based on a NoSQL Database and a JavaScript Visualization Application. *ISPRS Int. J. Geo-Inf.* **2019**, *8*, 227. [CrossRef]
40. Acquaviva, A.; Apiletti, D.; Attanasio, A.; Baralis, E.; Bottaccioli, L.; Cerquitelli, T.; Chiusano, S.; Macii, E.; Patti, E. Forecasting Heating Consumption in Buildings: A Scalable Full-Stack Distributed Engine. *Electronics* **2019**, *8*, 491. [CrossRef]
41. Marchiori, A.; Li, Y.; Evans, J. Design and Evaluation of IoT-Enabled Instrumentation for a Soil-Bentonite Slurry Trench Cutoff Wall. *Infrastructures* **2019**, *4*, 5. [CrossRef]
42. Alfian, G.; Syafrudin, M.; Ijaz, M.F.; Syaekhoni, M.A.; Fitriyani, N.L.; Rhee, J. A Personalized Healthcare Monitoring System for Diabetic Patients by Utilizing BLE-Based Sensors and Real-Time Data Processing. *Sensors* **2018**, *18*, 2183. [CrossRef]
43. Syafrudin, M.; Alfian, G.; Fitriyani, N.L.; Rhee, J. Performance Analysis of IoT-Based Sensor, Big Data Processing, and Machine Learning Model for Real-Time Monitoring System in Automotive Manufacturing. *Sensors* **2018**, *18*, 2946. [CrossRef]
44. Shichkina, J.; Degtyarev, A.; Kulik, B.; Fridman, A. Optimization of relational databases schemas by means of n-tuple algebra. *AIP Conf. Proc.* **2017**, *1863*, 110008. [CrossRef]