Translating a Distributed Relational Database to a Document Database

Muon Ha¹ · Yulia Shichkina¹

Received: 3 March 2021 / Revised: 17 January 2022 / Accepted: 4 February 2022 / Published online: 3 March 2022
© The Author(s) 2022

Abstract

The problem of transferring data from a database of one type to a database of another type is relevant today due to the increase in data volumes and the complexity of tasks solved in various fields of human activity. The existing databases are less and less satisfying the needs of users. New types of databases appear that are more suitable for solving large-scale problems. This article proposes an algorithm for solving the problem of optimizing the document structure of a database based on metadata about the structure of a distributed relational database from which data are transferred. The approach also takes into account information about the structure of the priority database queries. The priority of database queries is user-defined. The system of automatic translation of the database, taking into account these metadata, allows the user to create a distributed document database that is optimal in two parameters: in terms of the volume of stored data and in terms of the execution time of priority database queries.

Keywords NoSQL · Database query · Collection · Document · Database structure optimization · Embedded documents

1 Introduction

Over the past decades, the variety of database applications has increased, and with them the importance of database features such as simplicity, robustness, flexibility, performance, scalability, and compatibility has increased. And with the increase in the number of databases, one feature becomes more important than all the others. It is scalability. As more and more applications run under high load conditions, their scalability requirements can change and grow very quickly. Relational databases scale well only if they are located on a single server. When the resources of this server run out and it becomes necessary to increase the number of computational nodes and distribute the load between them, then problems arise with the growth of the complexity of relational databases, which directly depends on the amount of scalability. So, if there is an increase in the number of servers not to a few, but to a hundred or a thousand, then

the complexity will increase by an order of magnitude, and the characteristics that make relational databases so attractive rapidly reduce to zero the chances of using them as a platform for large distributed systems.

The solution to this problem is to transfer systems to other types of databases that have a higher scalability, but with the loss of other capabilities available in relational databases.

It must be said that this solution has a number of difficulties: the choice of a database type suitable for an applied task, in the case of a document database or a database of the column family type; choice of a global database structure in an existing distributed system and a local structure of objects in a database, transferring data from an existing database to a new type of database.

The problem is that document database management systems do not perform "join" operations well, so it is impossible to translate a distributed relational database into a document database format on a one-to-one principle. The use of this principle will lead to the opposite effect: a decrease of performance due to complication of queries and their partial implementation in client applications (those parts of queries that cannot be implemented by a document DBMS, e.g., the "join" operation). Currently, there is no formalized way to build an optimal distributed document database structure based on relational database metadata for translating data from one format to another. In this paper we propose a solution to this problem.

✉ Yulia Shichkina
strange.y@mail.ru

Muon Ha
muon.ha@mail.ru

¹ Faculty of Computer Science and Technology, Saint Petersburg Electrotechnical University "LETI", St. Petersburg, Russia

In earlier articles, we considered the problems associated with the optimization of the structure of non-distributed document databases and databases such as a family of columns, taking into account the structure of the planned queries [1]. We also described the methodology for applying methods for optimizing the structure of a database, depending on the stage at which this optimization is carried out: at the initial stage of creating a new database, at the stage of translating data from relational databases to NOSQL databases, or when consolidating databases of various types. But, all these solutions concerned only centralized databases.

Currently, due to the large increase in data volumes, distributed databases are increasingly used. Therefore, in this article, we show how methods designed for centralized databases would work when used to translate distributed relational databases to distributed NOSQL databases. According to our methodology described in [xxx], these methods can also be used when creating new document databases.

The proposed methods are based on graph theory, set theory, and parallel computing theory:

- Elements of set theory allow to formalize the process of choosing the optimal structure of documents in collections of the MongoDB database, taking into account the architecture of the computing system, the set of elements stored in the database, and queries to the database that are most often executed or require the highest execution speed. This is the first necessary step to achieve an optimal balance between the amount of stored data and the speed of execution of queries to the database.
- Elements of graph theory allow visualizing and formalizing the query execution plan depending on the architecture of the computing system and the database schema.
- Elements of the theory of parallel computing allow to formalize the process of optimizing the query execution plan. We use the theory of parallel computing based on information graphs [2, 3].

Thus, our proposed approach to translating a distributed relational database into a distributed document database is based on sequential steps: optimizing the document database schema based on metadata about the relational database, building query execution plans, optimizing query execution plans, translating queries from SQL format to MongoDB format.

The article is structured as follows. The second section provides an analysis of existing studies close to the topic of our research. This section shows that despite a huge number of publications on the topic of relational databases and NOSQL and their mutual transfer from one format to another, there is no research devoted to formalized methods of data translation in distributed databases. There are separate publications in which the authors talk about their experience of translating data in distributed databases to solve problems in a narrow

specific subject area. The third section provides a methodology for applying methods of optimizing the structure of document databases, taking into account the structure of planned queries for the case of a distributed database. The fourth section presents the results of testing the proposed methodology on databases of various sizes with queries of various structures and degrees of their embeddedness. In the conclusion, the main results of the research are summarized.

2 Overview of Related Works

The problem of translating relational databases into NoSQL is relevant and solved by many researchers. The main difficulty in solving this problem is the difference in data structures in relational databases (RDBs) and NoSQL.

The paper [4] offers a solution for quick data migration from a relational database into a document-oriented database. Authors have created semi-automatically two logical levels over physical data. Users can refine generated logical data model and configure data migration template for each needed document. Data migration features are implemented into relational database browser Dig Browser. But, the document does not present solutions to the problems of optimizing the database schema and database queries.

The paper [5] describes some approaches of migration and proposes an approach of model transformation from object relational database to NoSQL document database (MongoDB).

In article [6], the authors describe a semi-automated approach to migrate highly dynamic SQL-based web applications to ones that use document-oriented NoSQL databases such as MongoDB using source analysis and transformation techniques. Authors demonstrate our semi-automated framework on the analysis and migration of three existing web applications to extract, classify, translate, migrate, and optimize the queries and migrate the PHP code to interact with the migrated database. Authors approach provides migrating and optimizing the embedded SQL queries to interact with the new database system and changing the application code to use the translated queries. But, this approach does not take into account distributed database schemas and query optimization based on this schema.

Approaches to the translation of databases from the RDB format to the format of documentary NoSQL databases are described in various publications [7–12]. Articles [7, 8] describe automatic solutions in which the input data are RDB metadata and ER diagrams. In [8], an algorithm is presented that takes into account the dependencies between tables and the number of primary keys and foreign keys in the conversion process. In [9], an algorithm is presented that uses dependencies between tables, but data embeddedness is used only in relation to a foreign key to a primary key. In

article [7], tables are classified according to four types: main, subclass, related, and general. The algorithm presented by the authors uses this classification to include subclass and general tables in the main table. Table of relationships is converted by using the links. This research mainly focuses on transforming a relational concept (e.g., table, relationships, etc.) into a NoSQL (document database) concept (document, subdocuments, etc.). The research proposed by the authors [10] has a different classification, consisting of four classes: codifier, simple entity, complex entity, and N:N-relationship. In addition to classification, the user must provide a “focused table” that represents the target NoSQL entity. The algorithm creates a tree with related tables. In [11], the authors propose the use of tags to control the translation of databases. The user tags the ER diagram with tags that represent the characteristics of the data and queries. Based on these tags, the algorithm decides to use embedded documents or links in the conversion.

The authors [12] used formal concept analysis, conceptual scaling, and relational concept analysis to create a conceptual model that helps users define the structures of a NoSQL database. This model provides translation rules for all types of relationships between relational data tables (1–1, 1- n , n - n).

In connection with the increase in the volume of data, there has recently been an increased attention to the research of translation processes and optimization of queries when translating data from one format to another.

The document [13] proposes an application with a graphical user interface for data transfer and automatic query conversion. The system that authors propose can be subdivided into an online database application and a query conversion utility. The client application acts as an environment that allows the users to select and convert the databases from SQL to MongoDB. The article also discusses the structure, data types, and keys. The query conversion utility provides the user with graphical user interface that allows him to choose from some basic predefined SQL queries or write his own SQL query. This way the user can implement his queries even if he is not familiar with the MongoDB database.

The article [14] is devoted to the problems of translation of queries from a distributed relational database into NoSQL. This article discusses column-oriented NoSQL DBMS, HBase, because it is widely used by many Internet enterprises such as Facebook, Twitter, and LinkedIn. Because HBase does not support SQL, authors use Apache Phoenix as an SQL layer on top of HBase. The authors also confirm that important unsolved problems are supporting complex SQL queries, automatic index selection, and optimizing SQL queries for NoSQL.

In article [15], authors present a generic standards-based architecture that allows NoSQL systems, with specific focus on MongoDB, to be queried using SQL and seamlessly

interact with any software supporting JDBC. A virtualization system is built on top of the NoSQL sources that translates SQL queries into the source-specific APIs. The virtualization architecture allows users to query and join data from both NoSQL and relational SQL systems in a single SQL query.

In [16], it is proposed to analyze log files to determine which tables are frequently involved in a query with the “join” operator. The authors suggest placing these tables on the same node, if possible. This makes it possible not to use the “join” operator.

In general, the task of translating queries between the types of RDB and NoSQL databases is not a solved problem when translating databases. Analysis of publications shows that the following directions of its solution can be distinguished:

- Development of middleware software that executes SQL commands to process data for NoSQL databases. This approach was proposed in [17–19].
- Translation of queries from SQL to NoSQL format. For example, [20] is a web translator that takes a SQL query and generates an equivalent MongoDB query. The translation is based solely on the syntax of the SQL query. The approach does not take into account any data or schemas. There is no explanation for the approach to translation. Russell [21] describes a library that provides an API for translating SQL queries to MongoDB. The translation is based on SQL query syntax only. The system from [22] requires the user to provide a MongoDB schema expressed in relational form using tables, procedures, and functions.

Among the studies directly related to the optimization of queries to relational databases, it is possible to single out [23]. This document describes the methods used to optimize relational queries in an SDD-1 distributed database system. Queries are submitted to SDD-1 in a high-level procedural language called Datalanguage.

The following papers describe solutions for query performance:

The article [24] describes the Perfopicon system, which is an interactive query profiling tool. This tool allows you to quickly identify performance bottlenecks and data skew. Perfopicon combines interactive visualizations of (1) query plans, (2) overall query execution, (3) data flow among servers, and (4) execution traces. These views coordinate multiple levels of abstraction to enable detection, isolation, and understanding of performance issues.

The article [25] provides an overview of query optimization systems. In particular, it was considered that AQUA [26] is a query optimizer which aims at reducing the cost of storing intermediate results by optimizing the order of

join operations. QMapper [27] is an intelligent translator to enable automatic rule-based SQL-to-HiveQL mapping as well as cost-based optimizing of the translated queries.

There are also a number of articles devoted to translating databases into other types of NoSQL databases. Among them are the following.

The approach described in [28] is for creating data structures for column-oriented databases such as HBASE. The authors propose to analyze the primary key and foreign key of relational tables and then create a large table that stores all related information. The main focus of this approach is to create a key for a large table. The authors [11] also formulated some rules for converting a table with one embedded and several embedded DBMS tables to a column-oriented HBase database.

The easiest way to translate a relational database into a graph database is to simply convert each table record to a node, grouping the set of nodes by label name. Foreign keys between tables are converted into graph edges connecting two nodes [29]. The authors of [30] proposed the transformation of the 3NF-form of the RDB into the form of the 3EG graph model of the database. The authors used a relational database that exists in 3NF as input data and proposed four rules for transforming relational database tables into a graph database.

The authors of [31] also proposed an algorithm for converting a relational database into a graph database. They combine key values from more than one row in a single node, so that when a user needs relevant information, he can get it by visiting only one node. They define some rules for different cases of grouping information from different rows of a table in one node. They created a description of the full path along the graph, which allows to go from the source node to the destination node, and find information from a given node, which must be duplicated in another node.

Our analysis shows that the problem of translating data from one format to another is currently a very actual problem. However, a good solution to this problem has not yet been found. Many researchers are looking for a solution. There are specific solutions for individual areas, for direct translation of tables into collections, for transferring RDBs to document databases, taking into account the types of relationships between tables, for extracting data from document databases using SQL queries. However, the researchers have not yet proposed a general approach that would take into account the database schema, the computing system architecture, and the query structure. Our research is devoted to finding such a general approach.

This article proposes a formalized approach that takes into account RDB schemas, SQL query schemas, and sharding and replication schemes to create a distributed document database.

3 Methods on Which Distributed Database Translations are Based

The method for optimizing the schema of document databases taking into account the structure of executed queries for the case of a distributed database is based on similar methods for centralized databases described in [32] and methods for optimizing the graph of information dependencies.

The essence of the methods for optimizing the structure of document databases, taking into account the structure of executed queries for centralized databases, consists in the following postulates:

1. It is necessary to present a relational database in the form of a single set of fields, and all queries are also in the form of a single set of fields. Then, by iteratively applying set subtraction and set intersection operations, it is possible to obtain collections of fields that will correspond to the collections in the document database.
2. It is necessary to take into account during these operations the presence of types of links between tables of the relational database. Then, based on the collections obtained from the results of the first postulate, it is possible to optimize the structure of the document database by creating embedded documents.

These two methods of optimizing the structure of a document database without and with embedded documents are described in detail in [2].

The aim of the current research is to find approaches to account for the structure of a distributed database for accelerated query execution.

A database query is a set of instructions that describe the order of actions of the executor to achieve a certain result. After the development of parallelism in the operation of computers, the word "sequence" began to be replaced by the more general word "order." It is possible to say that a query to a database is a determination of the order and conditions for selecting data, and the process of creating a query is a decomposition of a query into elementary subqueries. An elementary subquery can be understood as a construct «Select» without embedded subqueries. If a query contains an elementary query (subquery) inside it, then it is called the parent, and the subquery is called the child.

An information graph is further understood as a directed acyclic multigraph, the vertices of which correspond to elementary queries within the main query, and the edges correspond to data transfer between them.

The principle of efficient organization of parallel query processing implies the search for subqueries that are independent of each other according to the data. When creating

a new query model, it is important that query properties such as computational accuracy and query robustness are preserved. For this, it is important to know which subqueries have input data that result from the execution of other subqueries. These links are reflected in the information graph. An important role in this is played by the concept of a parallel graph form.

When modifying the parallel form of the information graph, the structure of the graph does not change, but only its projection on the plane. Due to this, a new query created from the modified parallel form of the information graph is equivalent to the original query. But, with this approach, a global extremum may not be achieved in optimizing the solution to the problem in terms of certain parameters, for example, in time. This approach will find the most optimal query among all equivalent queries, taking into account data relationships between subqueries.

To construct a parallel plan of queries in the first approximation, it is possible to use the parallelization method of classical algorithms, based on the adjacency lists of the information graph of the algorithm [14]. Unlike classical algorithms, an information query graph is created much easier and has much fewer vertices.

Our research has shown that all methods for optimizing an information graph for various parameters (time, computational nodes, volume of interprocess transfers, etc.), developed for classical algorithms, are completely suitable for queries. The article [32] shows how it is possible to modify the database query information graph in order to increase the degree of query parallelism.

4 Creating a Document Database Schema Taking into Account the Information Graph of the Database Query

In our study, we used an approach based on information graphs to find the dependences of subqueries on the location and data availability in accordance with a given distributed database schema and information about shards and replicas.

A method for constructing an information query graph for a distributed relational database taking into account information about table fragmentation.

1. To each RDB table put in correspondence the input vertex of the information graph.
2. Associate each elementary subquery with the vertex of the information graph.
3. Connect the vertices of the information graph with directed edges in accordance with the rule: if subquery A receives data from table (or subquery) B, then connect vertices A and B with a directed edge from B to A.

4. Adaptation of the graph for horizontal sharding for the case when the table "T" of the RDB is distributed into k shards:
 - a. Create k input vertices of the graph, each of which will correspond to a certain part of the table on a certain shard: $T_i, i = 1 \dots k$.
 - b. If table T was connected by an edge with query Q , then create k graph vertices, each of which will correspond to query Q executed for a certain part of the table on a certain shard: $Q_i, i = 1 \dots k$.

Further, one of three options is possible:

Option 1.

- i. Connect all the vertices that were previously connected to the vertex Q with the vertices $Q_i, i = 1 \dots k$;
- ii. Connect each vertex corresponding to a part of the table T with a vertex corresponding to the query Q and executed for this part of the table T , i.e., the T_i part must be connected to Q_i (an example of this option is shown in Fig. 1b);
- iii. If the vertex Q was not the output vertex of the graph, then: create a vertex Q' , which will correspond to the query on data aggregation from queries Q_i over parts of the table T_i .

Option 2.

- i. Connect each vertex corresponding to a part of the table T with a vertex corresponding to the query Q and executed for this part of the table T , i.e., the T_i part must be connected to Q_i (an example of this option is shown in Fig. 1c);
- ii. If the vertex Q was not the output vertex of the graph, then: create a vertex Q' , which will correspond to the query on data aggregation from queries Q_i over parts of the table T_i ;
- iii. If there are vertices (except T) that were previously connected to the vertex Q , then create a node Q'' and connect all the vertex and the vertex Q' with the vertex Q'' ;

Option 3.

- i. Connect all the vertices that were previously connected to the vertex Q with the vertices $Q_i, i = 1 \dots k$;
- ii. Connect each vertex corresponding to a part of the table T with a vertex corresponding to the query Q and executed for this part of the table T , i.e., parts T_i must be connected to Q_i (an example of this option is shown in Fig. 1d);

Fig. 1 Information graphs of a query to the database: **a** initial graph, **b** graph corresponding to option 1, **c** graph corresponding to option 2, **d** graph corresponding to option 3

- iii. If the vertex Q was not the output vertex of the graph b and was connected by an edge to the query Q_1 , then create k vertices of the graph, each of which will correspond to the query Q_1 , executed for a certain part of the query Q_i on a certain shard: Q_{1i} , $i = 1 \dots k$.
- iv. Connect each vertex corresponding to the part of the query Q with the vertex corresponding to the part of the query Q_1 , i.e., connect Q_i with Q_{1i} ;
- v. If the vertex Q_1 was not the output vertex of the graph, then: create a vertex Q'' that will correspond to the query on data aggregation from queries Q_{1i} ;
- c. If there are more tables in the graph that are also distributed across shards, then step b) is repeated for each such table.

An example of creating an information query graph for the case of a distributed database. Let there be a database query information graph (Fig. 1a). Let the table $T1$ be distributed over 2 shards. Then the information graph of the database query, taking into account the sharding scheme, can have one of the forms shown in Fig. 1b–d according to options 1–3 described in the method.

Note. For the case of presence of replicas of tables on different shards, the same method of constructing an information graph of a query for a distributed relational database

is used, taking into account information about fragmentation of tables with modification. The modification consists in not doing sub-steps related to data aggregation in step b. This is step iii for options 1 and 2, step v for option 3.

Note. The choice of option 1–3 depends on the types of operators that are in the database query. So, for example, for a database query:

Select f1 from (select f1, f2 from t1, t2 where t1.key = t2.key and condition) as Q, t3 where Q.f1 = t3.key and condition;

option 3 will be optimal for execution.
To query the database:

Select f1 from t3 where f1 > (select count() from t1, t2 where t1.key = t2.key and condition) and condition;*

option 2 will be optimal for execution.
To query the database:

Select f1 from t3 where f1 in (select count() from t1, t2 where t1.key = t2.key and condition) and condition;*

option 3 will be optimal for execution.

A method for constructing a document database schema taking into account the structure of a distributed relational database:

1. Design an information graph without sharding and replication.
2. Transform the information graph to a parallel form that minimizes the transfer of data from shard to shard using the method of optimizing parallel algorithms by the number of communications [30–32].
3. Perform step 1–2 for all queries, whose structure should be taken into account when optimizing the document database schema in order to speed up the execution of these queries.
4. Based on the metadata about tables, fields, queries and relationships using methods of optimization of the document database schema with nested or without embedded documents, form the structure of document collections.
5. Improve the structure of documents, taking into account the graphs of information dependencies of queries, minimizing the number of data joining operations.

5 Testing the Approach to Translating Distributed Databases into Document Databases

To test the approach to translating distributed databases into document databases, a test relational database of three tables (DB_TEST) was created, the structure of which is shown in

Fig. 2. Table T1 is the main table in the link between tables T1 and T2 (Fig. 2). Table T2 is the main table in the link between tables T2 and T3.

Consider a query that finds the number of values in field T1.A that correspond to values in field F that are a multiple of 5 (listing 1):

Listing 1

Select SQL_NO_CACHE count(*) From t1 where A in (Select A from t2 where D in (Select D from t3 where F%5=0));

Fig. 3 Subqueries in the original query

Fig. 2 The structure of the test database

Fig. 4 Option A: information graph of a query to a centralized relational database

This query works with three tables. If the query structure shown in Fig. 3. highlight subqueries Q1, Q2, Q3, it is possible to design the information graph of the query for the case of a centralized database (Fig. 4).

The following describes testing a query to a MongoDB database without embedded documents based on a schema with sharding and replicas and in accordance with option 1 of the method for constructing an information query graph.

The first step was to define the structure of the collection in MongoDB by the method of optimizing the document database schema without embedded documents.

Three tables were used as input data: $T1(A, B, C)$, $T2(D, A, E)$, $T3(D, F, G)$.

The output data—the document database collections:

$V1 = \{T1.B, T1.C, T2.E, T3.G\}$

$V2 = \{T1.A, T2.D, T2.A, T3.D, T3.F\}$

Further, there is an option for this database in a distributed form with the presence of sharding, but without replications. Figure 5 shows the schema of the DB_TEST database, in which the $T3$ table is distributed over two shards. In this case, a horizontal fragmentation scheme is used, when records from table $T3$ contain values for all fields of table $T3$ and are distributed among different shards according to some rule, for example, according to a hash key. The case of vertical fragmentation can be thought of as dividing one table into several projections in the sense of relational algebra, linked by a one-to-one relationship. Therefore, this case is a special case of a centralized database and is not considered by us when looking for a formalized approach to translating distributed databases from one format to another.

Figure 6 shows a possible information graph for a database query from Listing 1, executed on a database with the schema in Fig. 5. This graph corresponds to the query to the distributed database without replication, but with sharding. The original query (Listing 1) has been split into subqueries that can be executed in parallel on shards:

Fig. 5 Database schema DB_TEST, in which table $T3$ is distributed over two shards

Select D as Q1_1 from t3 where $F \% 5 = 0$ // This is a subquery Q1 to the first part of table $T3$. The first part of the table $T3$ is indicated on the graph: $T3$ (Partition 1);

Select D as Q1_2 from t3 where $F \% 5 = 0$ // This is a subquery Q1 to the second part of table $T3$. The second part of the $T3$ table on the graph is designated as $T3$ (Partition 2);

In order to execute query Q2 on the results of query Q1, which consists of two parts, it is necessary to combine these parts of the result into a single part. For this, another subquery Q4 is created, which is executed in the application and corresponds to the code:

*Select * from Q1_1 union Select * from Q1_2*

Select A from t2 where D in Q4 // This is a subquery Q2 to the results of query Q1 and table $T2$;

Select SQL_NO_CACHE count() From t1 where A in Q2* // This is a subquery Q3 to the results of query Q2 and table $T1$.

Note: that the information graph can be different. Optimization methods for the classical algorithm represented by an information graph are described in [32].

If we add a replica for one of the tables to option C of a distributed relational database, for example, for table $T2$ (Fig. 7), then a possible information graph of the query from Listing 1, executed on DB_TEST, can be as in Fig. 8.

In this variant, the original query is divided into subqueries, which are executed in parallel to shards, including a shard with a replica:

Select D from t3 where $F \% 5 = 0$ // This is a query Q1 for the first part of table $T3$;

Select D from t3 where $F \% 5 = 0$ // This is a query Q1 to the second part of the table $T3$;

Fig. 6 Option B: information graph of a query to a distributed relational database with several shards, but no replications

Fig. 7 Database schema DB_TEST, in which table T2 has a replica

Select A from t2 where D in Q1 // This is a query Q2 to the results of query Q1 on the first part of table T3 and to table T2;

Select A from t2_copy where D in Q1 // This is a query Q2 to the results of query Q1 on the second part of table T3 and to table T2;

Select SQL_NO_CACHE count() From t1 where A in Q4* // This is a Q3 query to the results of Q4 query, which combines the results of previous queries, and to table T1.

To test these three options A, B, C, MariaDB was used with the Spider storage engine, which allows you to create distributed databases from a standard MariaDB installation and ensures data consistency when sharded. The table T3 was sharded in accordance with the hash algorithm on the field F.

Fig. 8 Option C: information graph of a query to a distributed relational database with several shards and replicas

Table 1 Volumes of the test database DB_TEST

	<i>T1</i>	<i>T2</i>	<i>T3</i>
Volume 1	1000	3000	9000
Volume 2	10 000	30 000	90 000
Volume 3	100 000	300 000	900 000
Volume 4	1000 000	3 000 000	9 000 000

To test the database DB_TEST, data were generated in four different volumes, the sizes of which are shown in Table 1.

Table 2 shows the volumes of the distributed database DB_TEST in accordance with the diagram in Fig. 5.

The table *T2_copy* is a Node2 replica of the table *T2* from Node1.

Since MariaDB with the Spider storage engine does not support a function for making database queries to individual shards, a separate program was created for testing. This program is developed in the Python programming language with the multiprocessing library.

The way in which the program performs a database query on the MariaDB database depends on the variant of the database structure A-C.

- For variant A of the DB_TEST database schema (not distributed), the execution of the program consisted in the direct execution of the query, and the execution time is

calculated from the moment the query was sent until the moment the results were returned.

- For variants B–C of the DB schema, the program executes sub-queries to shards in parallel. Then the program combines the results obtained from the shards according to the principle of the "Union" mechanism and sends the result to the input of the main query. The processing of a database query for variants B–C of the database schema can be represented as follows (Fig. 9):

Table 3 shows the query execution time (in seconds) using the developed program:

In Fig. 10 shows an example of the program execution result for the case of a distributed database structure with a volume of 9,000,000 records.

Query execution time graphs for different variants of database schemas and data volumes are shown in Fig. 11.

Figure 11 shows that for small amounts of data, a query to all three variants of the database scheme works almost the same, but with an increase in data volumes, the time difference between a sequential query and a parallel query increases. In this case, the fastest query is performed to the database, which contains not only sharding, but also replication.

Testing the approach to translating a database with embedded documents, taking into account the structure of the information graph, is shown below.

Table 2 Volumes of the distributed database DB_TEST

	Node 1			Node 2		
	<i>T1</i>	<i>T2</i>	<i>T3</i>	<i>T1</i>	<i>T2_copy</i>	<i>T3</i>
Volume 1	1000	3000	4500	Her	3000	4500
Volume 2	10,000	30,000	45,000	Her	30,000	45,000
Volume 3	100,000	300,000	450,000	Her	300,000	450,000
Volume 4	1000,000	3000,000	4500,000	Her	3000,000	4500,000

Fig. 9 Processing database query for variants B–C database schema

Table 3 Query execution time

	Volume 1	Volume 2	Volume 3	Volume 4
Option 1	0.00498	0.0651	0.708	6.37
Option 2	0.001966	0.01898	0.207	3.655
Option 3	0.00097	0.00898	0.1027	1.351

The first step was to determine the structure of the collection in MongoDB using the method of optimizing the schema of a document database with embedded documents.

Three tables were used as input data: $T1(A, B, C)$, $T2(D, A, E)$, $T3(D, F, G)$.

The output data—the document database collections:

$V1 = \{T1.B, T1.C, T2_of_T1 \{T2.E, T3_of_T2 : \{T3.G\}\}\}$.

$V2 = \{T1.A, T2_of_T1 :$

$\{T2.A, T2.D, T3_of_T2 : \{T3.D, T3.F\}\}\}$

In this testing, Option 1 or Option 2 of the information query graph method using an example of a database with embedded documents was evaluated to be faster.

A distributed database according to the scheme of the second variant of the method for constructing an information graph of a database query is shown in Fig. 12:

Note. In this case, duplication of fields was not done because the union of the results is done after the first subquery is executed.

Translation of subqueries Q1, Q2, Q3 for option B (Fig. 6) from SQL format to MongoDB format was carried out taking into account the execution plan of the relational query. An example of a Q1 subquery obtained as a result of translation from SQL to MongoDB format for one shard is shown below:

```

C:\Users\MaiAnh\Google Drive\NCS\K1 5\Queries_distrubute\Version 2>python Option3.py
query data from shard1 in 1607256804.764285
query data from shard2 in 1607256804.7912164
read data from shards done in 1607256804.81717
Input query
SELECT SQL_NO_CACHE COUNT(*) FROM t1 WHERE A IN (SELECT A FROM t2 WHERE D IN (SELECT D FROM t3 WHERE F %5=0))
-----
Subquery by the Node:
+-----+-----+-----+
| Shard | IP      | Query                                                                 |
+-----+-----+-----+
| Node1 | 192.168.136.141 | SELECT A FROM t2 WHERE D IN (SELECT D FROM t3 WHERE F %5=0) |
| Node2 | 192.168.136.142 | SELECT A FROM t2_copy WHERE D IN (SELECT D FROM t3 WHERE F %5=0) |
+-----+-----+-----+
-----
+-----+-----+
| Result | Time      |
+-----+-----+
| (488442,) | 1.350393533706665 |
+-----+-----+
  
```

Fig. 10 An example of program execution

Fig. 11 Query execution time for different variants of the database schema and data volume

Fig. 12 DB schema with embedded documents

```

db.collection.aggregate(
[{"$unwind": "$t2_of_t1"},
 {"$unwind": "$t2_of_t1.t3_of_t2"},
 {
   "$match": {"$and": [{"Location": "Node1"}, {"t2_of_t1.t3_of_t2.F": {"$mod": [5, 0]}}]}
 },
 {
   "$project": {"new_D": "$t2_of_t1.t3_of_t2.D", "_id": 0}
 }
])

```

The final query Q3 (taking into account that the results of subquery Q2 and table T2 are on the shard) is:

```

db.find({"$and": [{"Location": "Node1"}, {"t2_of_t1.D": {"$in": new_data}}]})

```

Taking into account the specification of the operation of the MongoDB DBMS, the Q1, Q2, Q3 subqueries are executed in the MongoDB query language, and the query to combine the results of the Q1 subqueries to the first part of the T3 table and Q1 to the second part of the T3 table is executed in a Python program. This is because MongoDB does not support join operations. The query execution plan is shown in Fig. 13:

The Python code for combining query results is:

```

#Read from shards
procs.append(Process(target=select_from_shard1()))
procs.append(Process(target=select_from_shard2()))
map(lambda x: x.start(), procs)
map(lambda x: x.join(), procs)
# Union results from Shards
new_data = Union(list_data1, list_data2)
# Execute general query
data = mycol.find({"$and": [{"Location": "Node1"}, {"t2_of_t1.D": {"$in": new_data}}]})

```

The distributed database according to the scheme of variant C of the method for constructing the information query graph is shown in Fig. 14:

Note. Because the second subquery works with the results of the first query and with fields from external documents in relation to embedded documents, then it is necessary in this case to duplicate all fields on both shards.

As in variant B, in this variant the subqueries Q1, Q2, Q3 were translated into the MongoDB query language. Query Q4 to combine the results of subqueries Q1, Q2 was executed by a Python program.

The subquery codes that are executed on each shard are shown below:

Fig. 13 Query execution plan using the program

Fig. 14 DB schema with embedded documents

Fig. 15 Query execution plan using the program

Table 4 Volumes of the test database DB_TEST:

	<i>T1</i>	<i>T2</i>	<i>T3</i>
Volume 1	1000	3000	9000
Volume 2	10,000	30,000	90,000
Volume 3	100,000	300,000	900,000
Volume 4	1000,000	3000,000	9000,000

Table 5 Volumes of database data DB_TEST on relational database shards

	Option 2		Option 3	
	Node 1	Node 1	Node 2	Node 2
Volume 1	1000	1000	849	849
Volume 2	10,000	10,000	8682	8682
Volume 3	10,000	10,000	87,461	87,461
Volume 4	1,000,000	1,000,000	874,525	874,525

Table 6 Volumes of database DB_TEST on MongoDB database shards

Volume	Option A	Option B		Option C	
		Node 1	Node 2	Node 1	Node 2
Volume 1	344,7 κб	267 κб	183 κб	213,2 κб	203,9 κб
Volume 2	3,4 мб	2,61 мб	1,8 мб	2,1 мб	2,0 мб
Volume 3	33,7 мб	26,12 мб	18,06 мб	20,8мб	20,1мб
Volume 4	336,6 мб	261,3 мб	180,69 мб	208,37мб	200,71 мб

Table 7 Time of execution of queries to DB_TEST

	Volume 1	Volume 2	Volume 3	Volume 4
Option A	0.0464	0.1423	1.297	13.445
Option B	0.0189	0.112	1.089	11.531
Option C	0.0114	0.0818	0.7502	8.884

The query execution plan is shown in Fig. 15:

To test queries for the DB_TEST database, data were generated in four different volumes, the sizes of which are shown in Table 4.

Table 5 shows the volumes of the distributed database DB_TEST in accordance with the scheme in Fig. 5.

Option 1 is a variant of a centralized database, option 2 is a distributed database with an information query graph according to option 2 of the method for constructing an information query graph, option 3 is a database with an information graph of queries according to option 3 of a method for constructing an information query graph.

Table 6 shows the volumes of the distributed database DB_TEST in accordance with the scheme in Fig. 5.

Table 7 shows the query execution time (in seconds) using the developed program:

Query execution time graphs for different variants of the database schema and data volume are shown in Fig. 16

From the diagram in Fig. 16, it can be seen that the maximum parallelization of a query in the database before combining the result works fastest with the application of an external program.

In our early article [32] we showed that queries to a centralized database with an optimized structure using a set theory method are significantly faster than a database built manually without optimizing the structure. The results of this study show that it is possible to further speed up query execution if the system is distributed and the architecture of the distributed system is taken into account when constructing queries. So in Figs. 11 and 16 it is possible to see that the centralized database even with an optimized structure (Option 1) has the longest execution time. Options 2 and 3 allow you to execute some of the subqueries in parallel and therefore work faster.

It should also be noted that the accelerated version requires more memory to store duplicate data. It is possible to see the results of testing all three options for the use of RAM and permanent memory, which are presented in Table 8.

```

db.collection.aggregate(
  [{"$unwind": "$t2_of_t1"},
   {"$unwind": "$t2_of_t1.t3_of_t2"},
   {
     "$match": {"$and": [{"Location": "Node1"}, {"t2_of_t1.t3_of_t2.F": {"$mod": [5, 0] }}}}]
  },
  {
    "$project": {"New_A": "$A", "_id": 0}
  }
])

```

Fig. 16 Query execution time for different variants of the database schema and data volume

The table shows the amount of used memory for two sizes of the database. For the other two database volumes, the picture of used memory is similar. More clearly the use of RAM is shown in Fig. 17.

Figure 17 shows that the fastest Option 3 is more memory intensive than Option 2. This is due to the fact that more duplicate data are placed in memory to speed up the execution of the query. Nevertheless, Option 3 is much more economical compared to Option 1.

To assess which operations in the query take the longest time with different sharding and replication options, the test was conducted for a database with a volume of 1000 000 records. Based on the test results a diagram was built (Fig. 18).

The length of each block on the diagram corresponds to the execution time of separate query operation (Q1–Q4). The execution time of each operation was measured using a specially designed program. Twenty-five tests were carried out. The diagram shows the average results. From the diagram in Fig. 18 it is possible to see that in Option 3 the execution time of parallel subqueries (Q1–Q2) on the shards is longer than in Option 2. But due to the fact that most of the work has been parallelized the "join" operation (Q4) in Option 3 is performed for less data and therefore executed faster than in Option 2. Paralleling most of the work is not achieved by performing more suboperations, but by processing more data in parallel, as it is possible to see from the RAM occupancy diagram in Fig. 17. It is in Option 3 that more memory is occupied, because more data are processed. The last subquery Q3 is also applied to a smaller amount

of data, so its execution speed is also higher. In Option 3, most of the data is processed at the first stage (execution of queries Q1, Q2) in parallel.

6 Conclusion

This article has shown that when creating a distributed document database, there is a need to consider:

1. The structure of queries, relationships between tables. In accordance with these data, it is necessary to create a document database model with or without embedded documents.
2. The presence of shards and replicas. The presence of replicas significantly speeds up the process of executing queries, because allows to maximally parallelize a complex query based on the information graph of the query.
3. Internal dependencies and query operations. This determines the way the information graph is presented and, subsequently, the structure of the document database collections.
4. Consensus between memory size and query execution speed. Database queries in which maximum parallelization is done directly by the DBMS are usually faster. However, in this case, it is necessary to make not a minimum amount of memory and store duplicate records in different collections on different shards.

Table 8 Results of testing memory

Volume	Option	Permanent memory (Kb)	RAM	Maximum RAM (Mb)
1,000,000	Option A—centralized database	976		24.9
	Option B—distributed database with sharding without replication	1972		96.5
	Option C—distributed database with sharding and replication	2103		204.2

Fig. 17 Graph of used memory changes during query execution

Fig. 18 Diagram of execution of operations of a query on a database of 1000 000 records

Acknowledgements Not applicable.

Author Contributions Conceptualization, methodology, formal analysis, writing—review and editing, project administration were performed by YS; MH helped in software, validation, formal analysis, original draft preparation, visualization. All authors have read and agreed to the published version of the manuscript.

Funding This work was supported by the Ministry of Science and Higher Education of the Russian Federation by the Agreement No 075-15-2020-933 dated 13.11.2020 on the provision of a grant in the form of subsidies from the federal budget for the implementation of state support for the establishment and development of the world-class scientific center «Pavlov center «Integrative physiology for medicine, high-tech healthcare, and stress-resilience technologies»

Data Availability Data available upon request.

Declarations

Conflict 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.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

- Ha M, Shichkina Y (2020) Creating collections without embedded documents for document databases taking into account the queries. *Computation* 8(2):45. <https://doi.org/10.3390/computatio8020045>
- Shichkina Y (2015) Approaches to speed up data processing in relational databases. *Procedia Comput Sci* 150:131–139
- Shichkina Y, Kupriyanov M, Al-Mardi M (2016) Optimization algorithm for an information graph for an amount of communications, *Lecture Notes in Computer Science* (including subseries *Lecture Notes in Artificial Intelligence* and *Lecture Notes in Bioinformatics*), 9870 LNCS, c. 50–62, ISSN: 03029743 ISBN: 978-331946300-1, https://doi.org/10.1007/978-3-319-46301-8_5
- Karnitis G, Arnicans G (2015) Migration of relational database to document-oriented database: structure denormalization and data transformation. In: 2015 7th international conference on computational intelligence, communication systems and networks, Riga, Latvia, pp. 113–118 <https://doi.org/10.1109/CICSyN.2015.30>
- Toufik F, Bahaj M (2019) Model transformation from object relational database to nosql document database. In: *Proceedings of the 2nd international conference on networking, information systems & security (niss19)*. Association for computing machinery, New York, NY, USA, Article 49, pp. 1–5 <https://doi.org/10.1145/3320326.3320381>
- Al Mahruqi RS, Alalfi MH, Dean TR (2019) A semi-automated framework for migrating web applications from SQL to document oriented NoSQL database. In: *Proceedings of the 29th annual international conference on computer science and software engineering (CASCON '19)*. IBM Corp., USA, pp. 44–53
- Stanescu L, Brezovan M, Burdescu DD (2016) Automatic mapping of MySQL databases to NoSQL MongoDB. In: 2016 FedC-SIS, pp. 837–840
- Zhao G, Lin Q, Li L, Li Z (2014) Schema conversion model of SQL database to NoSQL. In: 2014 Ninth 3PGCIC, pp. 355–362
- Freitas MC, Souza DY, Salgado AC (2016) Conceptual mappings to convert relational into NoSQL databases. In: *Proceedings of the 18th ICEIS*
- Karnitis G, Arnicans G (2015) Migration of relational database to document-oriented database: structure denormalization and data transformation. In: 2015 7th ICCICSN, pp. 113–118
- Jia T, Zhao X, Wang Z, Gong D, Ding G (2016) Model transformation and data migration from relational database to MongoDB. In: *IEEE BigData*, pp. 60–67
- Varga V, Jánosi-Rancz KT, Kálmán B (2016) Conceptual design of document NoSQL database with formal concept analysis. *Acta Polytech Hung* 13(2):229–248
- Shaikh NF (2017) Data migration from SQL To Mongodb. *Int J Eng Res Dev* 13(11):32–36
- Kim HJ, Ko EJ, Jeon YH, Lee KH (2018) Migration from RDBMS to column-oriented NoSQL: lessons learned and open problems. In: Lee W, Choi W, Jung S, Song M (Eds.), *Proceedings of the 7th International Conference on Emerging Databases. Lecture Notes in Electrical Engineering*, vol 461. Springer, Singapore. https://doi.org/10.1007/978-981-10-6520-0_3, https://link.springer.com/chapter/10.1007/978-981-10-6520-0_3#citeas
- Lawrence R (2014) Integration and virtualization of relational SQL and NoSQL systems including MySQL and MongoDB. In: 2014 international conference on computational science and computational intelligence, Las Vegas, NV, USA, 2014, pp. 285–290 <https://doi.org/10.1109/CSCI.2014.56>
- Hsu MJC, Hsu CH, Chen SC, Chung YC (2014) Correlation aware technique for SQL to NoSQL transformation. In: 7th international conference on ubi-media computing and workshops (UMEDIA). IEEE Press, Ulaanbaatar, pp. 43–46
- Zhang H, Wang Y, Han J (2011) Middleware design for integrating relational database and NOSQL based on data dictionary. In: *Proceedings 2011 international conference on transportation, mechanical, and electrical engineering (TMEE)*, Changchun, China, pp. 1469–1472 <https://doi.org/10.1109/TMEE.2011.6199485>
- Ma K, Abraham A (2013) Toward lightweight transparent data middleware in support of document stores. In: 2013 third world congress on information and communication technologies (WICT 2013), Hanoi, Vietnam, 2013, pp. 253–257 <https://doi.org/10.1109/WICT.2013.7113144>
- Adriana J, Holanda M (2018) NoSQL2: SQL to NoSQL databases. In: Rocha Á, Adeli H, Reis L, Costanzo S (eds) *Trends and advances in information systems and technologies. WorldCIST'18 2018. Advances in intelligent systems and computing*, vol 746. Springer, Cham
- QueryMongo. Query mongo: Mysql to mongodb query translator, 2019. URL <http://www.querymongo.com>.
- Russell V (2016) sql-to-mongo-db-query-converter URL <https://github.com/vincentrussell/sql-to-mongo-db-query-converter>.
- Reddy R (2015) Mongodb translator - teiid 9.0 (draft), URL <https://docs.jboss.org/author/display/TEIID/MongoDB+Translator>
- Bernstein PA, Goodman N, Wong E, Reeve CL, Rothnie JB (1981) Query processing in a system for distributed databases (SDD-1). *ACM Trans Database Syst* 6(4):602–625. <https://doi.org/10.1145/319628.319650>
- Moritz D, Halperin D, Howe B, Heer J (2015) Perfopticon: visual query analysis for distributed databases. *Comput Graph Forum*. <https://doi.org/10.1111/cgf.12619>
- Dai J (2019) SQL to NoSQL: what to do and How. *IOP Conf Ser Earth Environ Sci* 234:012080. <https://doi.org/10.1088/1755-1315/234/1/012080>
- Wu S, Li F, Mehrotra S, et al. (2011) Query optimization for massively parallel data processing. In: 2nd ACM symposium on cloud computing. ACM, p. 12
- Xu Y, Hu S (2013) QMapper: a tool for SQL optimization on hive using query rewriting. In: 28th international conference on world wide web companion. pp. 211–212
- Lee C, Zheng Y (2015) Automatic SQL-to-NoSQL schema transformation over the MySQL and HBase databases. In: *IEEE international conference on consumer electronics-Taiwan*, pp. 426–427

29. <https://neo4j.com/developer/guide-importing-data-and-etl/>
30. Park Y, Shankar M, Park B-H, Ghosh J (2014) Graph databases for large-scale healthcare systems: A framework for efficient data management and data services. In: IEEE 30th international conference on data engineering workshops. Chicago, IL, USA, pp. 12–19
31. De Virgilio R, Maccioni A, Torlone R (2013) Converting relational to graph databases. In: First international workshop on graph data management experiences and Systems. ACM
32. Ha VM, Shichkina YA, Kostichev SV (2019) Determining the composition of collections for key-document databases based on a given set of object properties and database queries. *Computer Tools in Education*, no. 3, pp. 15–28. <https://doi.org/10.32603/2071-2340-2019-3-15-28> (in Russian)