

NoSql Database Optimization Based on Metadata About Queries and Relationships Between Objects

Muon Ha, Yulia Shichkina(✉), and Roza Fatkieva

St. Petersburg State Electrotechnical University, St. Petersburg, Russia
Strange.y@mail.ru

Abstract. Most experts in the field of big data agree that the volume of data generated by various devices will increase exponentially in the future. Therefore, there is no doubt the relevance of solving the problem of data storage in such a way that access to the necessary information would be as quick as possible. In this article, we propose an approach based on set theory, which is suitable for databases that store information about objects with many relationships. Such databases can include both relational and NoSQL databases. This approach takes into account the relationships between objects and the structure of the most frequently executed database queries when designing the database architecture. This allows to remove operations "join" from queries and, at the same time, to minimize the data redundancy anomaly in a separate database structural unit. We demonstrated the application of this approach on three types of databases: relational, document, and columns family.

Keywords: NoSQL · Database query · Collection · Document · Data conversion · Data format · Database structure optimization · MongoDB · Cassandra

1 Introduction

Among the databases, despite the intensive development of NoSQL and NewSQL databases, relational databases still hold the lead (Fig. 1). It is known that relational DBMSs allow to create complex data systems, and NoSQL databases and DBMSs do not support internal relationships between objects. NoSQL databases are not based on one model, and each database uses different models de- pending on the purpose of creation.

All NoSQL databases today can be divided into 4 large classes:

- key-value (Redis, MemcacheDB, etc.);
- column family - Cassandra, HBase, etc.;
- document - MongoDB, Couchbase, etc.;
- graph - OrientDB, Neo4J, etc.

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2022
O. Gervasi et al. (Eds.): ICCSA 2022 Workshops, LNCS 13380, pp. 588–602, 2022.
https://doi.org/10.1007/978-3-031-10542-5_40

A lot of educational books, articles have been written about all these systems, and many different professional websites and forums on the Internet have been devoted [3, 10, 13].

In fact, a review of systems that are based on NoSQL databases allows to conclude that, for one reason or another, many developers today choose not only relational databases to create complex data systems. It is also worth noting that if the theory of designing relational databases based on functional dependencies and theorems for normalizing relational schemes has existed since the 70s [5], then for NoSQL databases there is no such theory yet, but there is a lot of research in this area. Some of the work we consider in the next section.

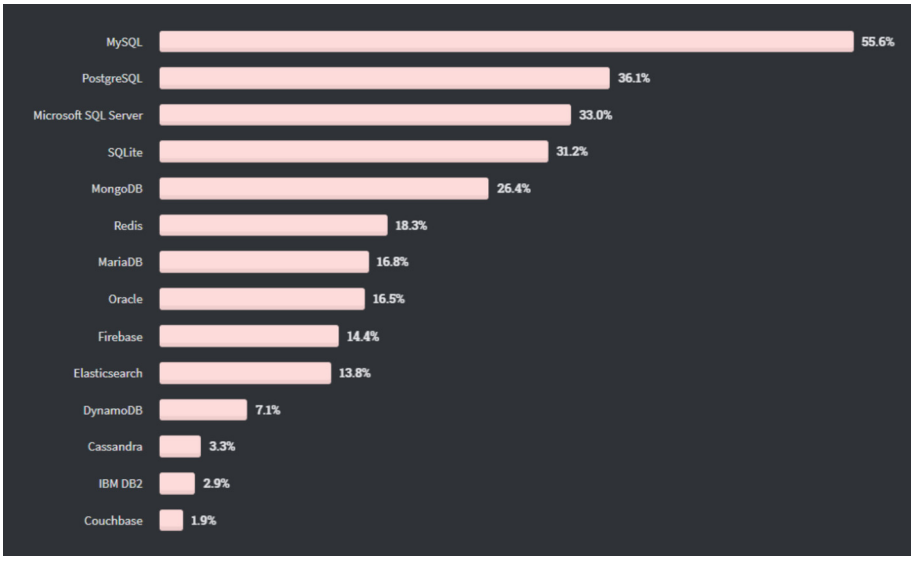

Fig. 1. Database rating [6]

And it's worth noting that many developers choose another way instead of using NoSQL to speed up access to data: denormalizing a relational database. Unfortunately, there is also no theory of optimizing data access by denormalizing a relational database, but there are many works on the Internet in which the authors describe their results of database denormalization.

The approach that we propose in this article is an attempt to create a universal apparatus based on set theory, which, in accordance with the latest proven trends in orienting databases to specific queries, allows to choose a database architecture based on formalization. We tested this approach on the three most popular in its class data type, included in the top fifteen databases [6]: MySQL, MongoDB and Cassandra. As a result, we confirmed the efficiency of the approach in queries of various complexity and showed that the approach provides the best local solution for optimizing the database in terms of query execution time. This also allowed us to compare the obtained solutions

for these three databases with each other. The results of these studies are presented in this article.

The article is structured as follows. The Sect. 2 provides an overview of related work in recent years; the Sect. 3 describes an approach to creating a database architecture. The Sect. 4 presents the experimental results. In conclusion, the results of the research are summarized.

2 Related Works

Among the studies devoted to optimizing data access in various databases, the following directions can be identified:

1. Research on relational databases
   In [12], there are proposed new requirements for functional dependencies that can improve the design results of a relational database schema. For this purpose, the authors assign to tuples the value of the probability with which these tuples are in a relation, assign to functional dependencies a confidence value that indicates to which tuples these dependencies correspond. This allows, within reasonable limits, to carry out the decomposition of relations in relational databases.

   In [17], there is an algorithm for searching for possible functional dependencies between non-key attributes. As a result, the database developer can use the found functional dependencies to improve the structure of the database.

   We think that these approaches are very good when, when creating a database, queries to the database are unknown and the administrator needs to provide any options for queries. This will speed up many queries, but it will also create a redundancy anomaly where it might not have. In our case, the approach to denormalization is query-oriented and this allows us to reduce the redundancy anomaly.
2. Research on data migration between databases.
   The article [2] proposes rules for transforming an RDB schema to various NoSQL database schema, namely document-based, column-based and graph-based databases. The rules are applied based on the type of relationships that can appear in data within a database. All the rules are demonstrated by the authors on the three most popular NoSQL databases: MongoDB, Cassandra and Neo4j. We believe that the authors of [2] offer a good, but rather universal method. This method without taking into account the properties of queries will lead to the fact that on the basis of the M-1 relationship, for example, in a document database, very large documents will be created, which will slow down the work with the database. If the most frequent queries to the database are not applied to all the attributes of relations with the M-1 relationship type, then it is better to create several collections with smaller documents instead of one collection. Our approach allows to do this.

It should be noted that the topic of data migration from a relational database format to NoSQL is quite popular today, and most publications in well-known citation databases are devoted to this topic. Among such publications over the past year are [1, 8, 15] for MongoDB, an approach to transforming a model from an object-relational database to a database based on the NoSQL column family (Cassandra) [9] and many others. But, in all these studies, the structure of database queries is not taken into account.

3. Research that focuses on data indexing and query optimization. These works, for example [4, 14], are aimed at speeding up data access, but they do not directly relate to the database architecture and therefore the results of their analysis are not presented in this article.
4. Applied research on the construction of databases of various types in a given subject area. These studies are interesting and useful because the authors share their experience in creating databases, the problems of their construction, and found particular solutions to these problems. Such articles include the use of NoSQL in the Internet of Things (IoT), where a large number of sensors are used in everyday life [7]; collection and processing of data on various weather parameters for a smart home [11]; food supply chain traceability [16] and others.

The analysis of world research in the field of formalization of the approach to finding the optimal database architecture in terms of the search query execution time showed that there are currently no methods that take into account the metadata about queries and relationships between objects and are applicable to relational, documentary and column-family databases.

In this article, we present the results of a generalized approach to optimizing database architecture based on set theory that we developed and tested on three types of databases: relational, documentary, and column family.

3 Generalized Approach to Defining Database Architecture

By analogy with a relational database, we mean by a relation a finite set of values of object properties; schema of relation is a set of object attributes, object properties, data about which should be stored in the database. Functional dependence is dependence in the classic definition from the relational model [5]. Suppose there is a set of attributes X and for the attributes of a set X there is a set of declared functional dependencies.

Initially, it is assumed that the database schema consists of one relation $T = X$. Further, this scheme should be, taking into account the declared functional dependencies, reduced to the third normal form or the Boyce-Codd form. As a result, there are will be obtained the first group of sets - these are the sets of attributes belonging to a separate relation T_r, where r is number of relation, $r = 1...k$, k is quantity of relations, $T_{(r,j)}$ is an attribute in relation T_r, j is number of attribute, $j = 1...r_n$, r_n is quantity of attributes in the r-th relation Eq. 1:

$$T_r = \{T_{r,j}, j = 1, 2, ..., r_n\} \tag{1}$$

Then the set of all database attributes will be determined by the formula Eq. 2:

$$X = \{T_{r,j}, r = 1, 2, ...k | T_{r,j} \neq T_{q,i}, \forall r, q \leq k, j \leq r_n, i \leq q_n\} \quad (2)$$

where: r is the number of relation, $r = 1...k$, k is quantity of relations, j is number of attribute in the relation, $j = 1...r_n$, r_n is quantity of attributes in the r-th relation.

The length of a set is the quantity of its elements. Designation: $|X|$ - the length of the set X.

In order not to do "join" operations in the future to perform queries, it is necessary to understand what attributes are accessed by queries in the database. This requires the creation of a second group of sets - these are the sets of attributes included in the queries Eq. 3:

$$S_i = \{T_{r,j}, r \leq k, j \leq r_n\} \quad (3)$$

where: i is the number of queries $i = 1, 2, \ldots m$, m is the quantity of queries to the database.

Obviously, those attributes that are not referenced in the most frequent queries can be collected into a separate relation. This will be the first step towards denormalizing the resulting relation. This is because the data will enter the new relation from different relations, including those between which there was a relationship of type "$M-1$" (Many to one), and this will lead to the creation of duplicates.

There are two ways to find such attributes:

(1) First, for each attribute, create a set of queries in which this attribute participates in any part of the construction:

$$T'_{r,j} = \{S_i, i = 1...p, p \leq m | T_{r,j} \in S_i\} \quad (4)$$

where m– quantity of queries, r – number of relations, $r = 1...k$, k – quantity of relations, j – number of the attribute in relation, $j = 1...r_n$, r_n– quantity of attributes in r-th relation. Then, among the sets that we have obtained, choose those for which the length of the set is equal to zero: $|T'_{r,j}|$. The attributes corresponding to these sets are those attributes that do not participate in queries.

(2) These same attributes can be defined as the difference between the original set of X attributes and the set of attributes included in the queries Eq. 5:

$$V_1 = X - \bigcup_{i=1}^{p} s_i \quad (5)$$

It should be noted that the first method is preferable, since in the future, we will also need to find attributes that are involved in only one database query, and formula 4 will make it easy to do this.

So, these attributes can be included in a new relationship (6):

$$V_1 = \{T_{r,j}, r \leq k, j \leq r_n || T'_{r,j}| = 0\} \tag{6}$$

where: m - the quantity of queries, r - the ordinal number of relations, $r = 1...k$ the quantity of relation, j - the ordinal number of the attribute in relation, $j = 1...r_n$, r_n the quantity of attributes in r-th relation.

It is possible not to follow the path of denormalization and create new relations based on these attributes (7):

$$V_i = T_i - \bigcup_{j=1}^{p} S_j \tag{7}$$

In this case, it should be noted that, firstly, a relationship cannot consist of one attribute, and secondly, if the attribute is not key, then the new relationship must include key attributes from the previous relationship. The latter is necessary in order to reduce the anomaly of lost information, which can be in any type of database.

The choice of a way to represent attributes that are not involved in queries is shown in the scheme in Fig. 2.

Note 1. The new relations should be formed taking into account the relationships between the original relations.

Example: In terms of relational algebra, composing a relation V_1 from four relations T1–T4 might look like this:

$$V_1 = \pi(T1 \triangleright\triangleleft T2 \triangleright\triangleleft T3 \triangleright\triangleleft T4)_{T1_4, T2_2, T2_5, T3_5, T3_6, T4_2, T4_5, T4_6}$$

where, the operations $\triangleright\triangleleft$ on are the natural join of the relations, $\pi(X)_{y,z}$ is the projection or vertical selection in the relation X of the attributes y, z.

With attributes that are involved in only one database query, i.e. $|T'_{r,j}| = 1$, it is possible to act in the same way as with attributes that are not involved in database queries. All the attributes $T_{r,j} \in S_i$ for which the condition $|T'_{r,j}| = 1$, is satisfied, must be separated into a new relation 8 or they can be composed of separate relations:

$$V_p = \{T_{r,j}, r \leq k, j \leq r_n | T_{r,j} \in S_i \& |\{T_{r,j}\}| = 1\}, p > 1 \tag{8}$$

After all attributes are removed from consideration that did not participate in database queries or participated in queries once, i.e. included in the new relations V1 and V_p, it remains to distribute those attributes that are involved in several database queries over the new relations.

This can be done iteratively by finding a new relation at each iteration as the difference of sets (9), (10):

$$V_p = I - P \tag{9}$$

$$I = P \tag{10}$$

where:

- m - the number of queries;
- $I = \bigcup_{i=1}^{m} S_i$ - the set of attributes included in database queries;
  $S_k^{'} = S_i \cap S_j, \forall i \neq j; i, j = 1, 2, ..., |I|; k = 1, 2, ..., C_{|I|}^2$ - the pairwise intersections of sets $S_i \in I$;
- $C_{|I|}^2 = \frac{|I|!}{|(|I|-2)|!2!} = \frac{|I|(|I|-1)}{2}$ - the number of pairwise intersections of sets $S_i \in I$;

The iterative process continues as long as $|I| > 1$.

When the length of the set I becomes equal to 1, the iterative process will be stopped and it is necessary to include in the new relation all the attributes from the last single intersection that are not present in other intersections (11):

$$V_{(p+1)} = X - \bigcup_{i=1}^{p} V_i \tag{11}$$

As a result of this approach, a set of new relations with attributes will be obtained (regardless of the type of database) (12):

$$V_i = \{T_{r,j}, r \leq k, j \leq r_n\} \tag{12}$$

satisfying conditions (13), (14):

$$\bigcup_{i=1}^{p} V_i = X = \bigcup_{r=1}^{k} T_r, \tag{13}$$

$$(\forall S_i)(\exists V_j)(S_i \in V_j, S_i \notin V_i, i \neq j) \tag{14}$$

where i is the ordinal number of relation ($i = 1, 2, ..., l$), l is the quantity of relations.

Note 2. The new relation could be a column family for a Cassandra database, or a collection for a MongoDB document database, or a denormalized relation for a relational database.

Note 3. In [18] we considered the second method, which is a continuation of this method for MongoDB databases and which allows you to create document collections with embedded documents of any depth.

The scheme of the described approach is shown in Fig. 2.

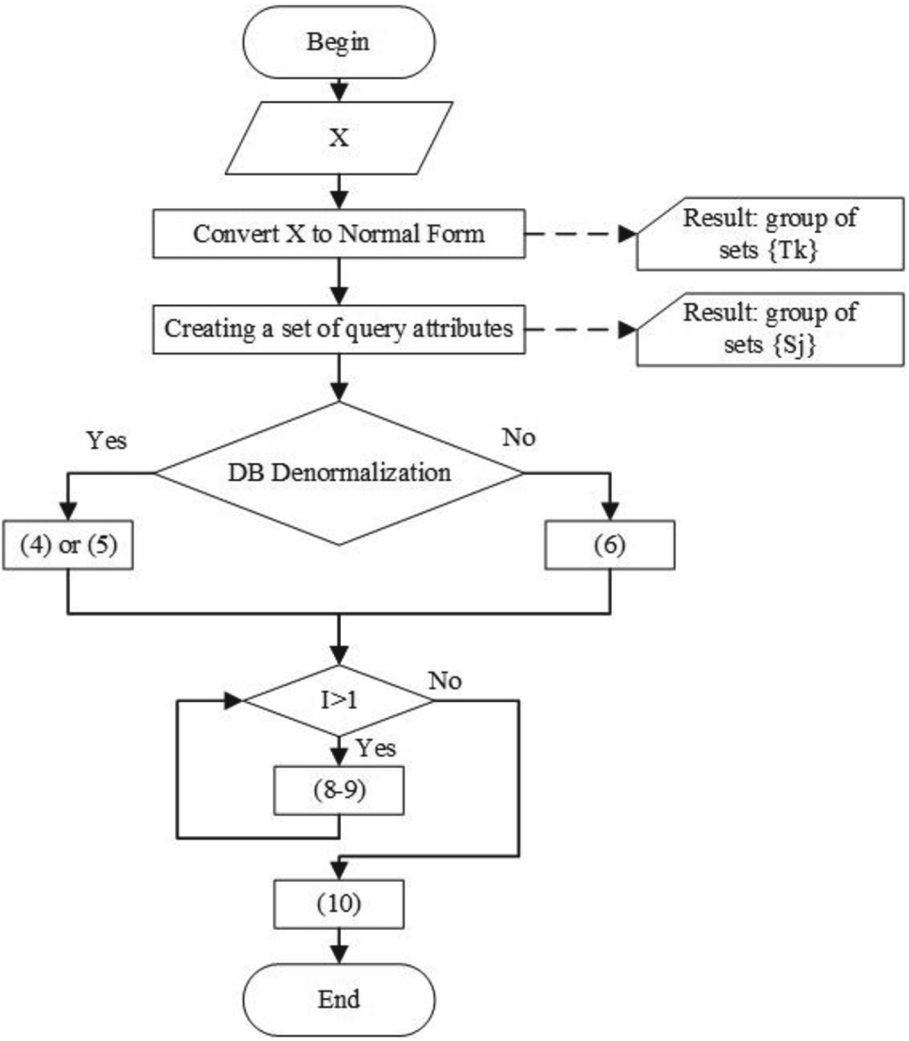

Fig. 2. Scheme of creating relations taking into account database queries

4 Testing the Approach on MySQL, MongoDB and Cassandra Databases

To test this approach, we chose a database, which in the relational model consisted of the following five relations:

- *T1(Output_phone_number, Start_time_of_the_call, Input_phone_number, End_time_of_the_call, Date) (or T1(f1, f2, f3, f4, f5))*
- *T2 (Output_phone_number, SMS, Input_phone_number, Time_sending, Volume, Date) (or T2(f1, f2, f3, f4, f5, f6))*

- *T3 (Phone_number, Tariff, Id_User, Contract_number, Start_date_of_the_contract, Start_date_of_the_contract) (or T3(f1, f2, f3, f4, f5, f6))*
- *T4 (id_User, Birthday, Passport, Registration, Email, Fullname) (or T4(f1, f2, f3, f4, f5, f6))*
- *T5 (Contract_number, Payment, Date_of_payment) (or T5(f1, f2, f3))*

Relationships between relations are shown in Fig. 3. To assess the effectiveness of this approach, testing was carried out on various volumes of the database:

- Volume 1: User(10.000), Number_phone(10.000), Payment(25.000), SMS(43.000), Call(32.000)
- Volume 2: User(50.000), Number_phone(50.000), Payment(65.000), SMS(81.000), Call(32.000)
- Volume 3: User(100.000), Number_phone(100.000), Payment(123.000), SMS(230.000), Call(320.000)

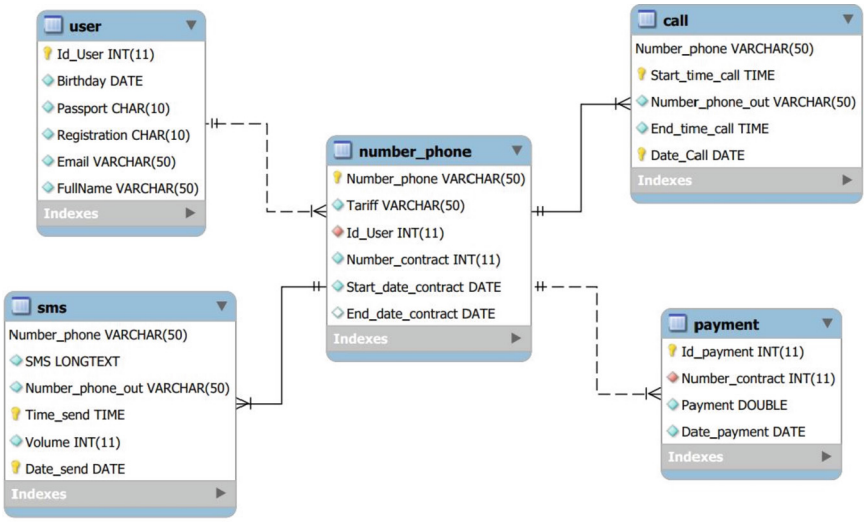

Fig. 3. Relationships between relations

Testing was carried out for 9 search queries, which were prepared in such a way that among them there were simple queries to select data from only one relation, queries to select data from several relations, queries with aggregation functions:

5. Select sum(f4 − f2); f1 From T1 Where f5 > a and f5 < b Group by f1;
6. Select count(∗); f1 From T2 Where f6 > a and f6 < b Group by f1;
7. Select T3.f1; T3.f2 From T3; T4 Where T3.f3 = T4.f1 and T4.f3 = a and T3.f6 = null;

8. Select T1.f2; T1.f3 From T1 Where T1.f1 = a and f5 > a and f5 < b Union Select T2.f3; T2.f4 From T2 Where T2.f1 = a and f6 > a and f6 < b;
9. Select T1.f2; T1.f1 From T1 Where T1.f3 = a and f5 > a and f5 < b Union Select T2.f1; T2.f4 From T2 Where T2.f3 = a and f6 > a and f6 < b;
10. Select T3.f1; T3.f2 From T3; T4 Where T3.f3 = T4.f1 and T4.f3 = a and T3.f4 ≥ a and T3.f4 ≤ b Order by T3.f4;
11. Select distinct T3.f1; T4.f6 From T3; T4 Where T3.f1 = a and T3.f3 = T4.f1;
12. Select T 5.f2; T5.f3 From T 3; T 5 Where T3.f1 = a and T 5.f3 > a and T5.f3 < b and T3.f4 = T5.f1;
13. Select T5.f2; T5.f3 From T5 Where T5.f3 in Select max(T5.f3) From T3; T5 Where T3.f1 = a and T3.f4 = T5.f1;

For the objectivity of the results, each query was tested 25 times. The table, chart, and graphs below show the average query execution time for 25 times. As a result of applying the approach described in Sect. 3, three relations were obtained:

$$V_1 = \{T2_2;\ T2_5;\ T3_5;\ T4_2;\ T4_4;\ T4_5\}$$

$$V_2 = \{T1_1;\ T1_2;\ T1_4;\ T1_5;\ T2_1;\ T2_6;\ T1_3;\ T2_3;\ T2_4\}$$

$$V_3 = \{T3_1;\ T3_2;\ T3_3;\ T3_6;\ T4_1;\ T4_3;\ T3_4;\ T4_6;\ T5_1;\ T5_2;\ T5_3\}$$

These three relations are three collections in MongoDB (Figs. 4 and 5) and three column families in Cassandra (Fig. 6).

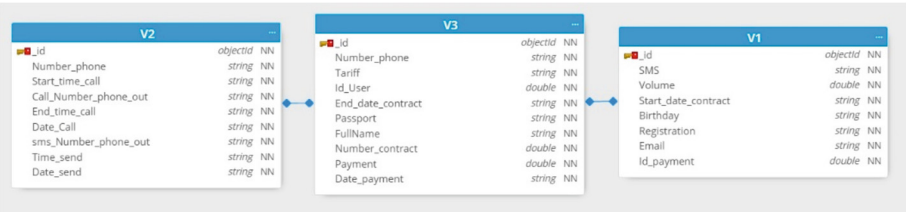

Fig. 4. MongoDB database structure without embedded documents

Table 1 shows the results of testing databases of various architectures and types. Times are in milliseconds. These results are presented for volume 3. Columns I–VI contain the time to execute queries in the following databases: I -MySQL, the initial relational model of 5 relations, II - MongoDB database obtained from the relational model by direct data translation and therefore consisting of 5 collections, III - MongoDB database obtained using the above approach and consisting of three collections without embedded documents, IV - MongoDB database obtained using the above and additional methods [18] and consisting of three collections with embedded documents, V - Cassandra database obtained from a relational model with direct data translation and therefore consisting of 5 column families, VI - Cassandra database obtained using the above approach and consisting of three column families.



Fig. 5. MongoDB database structure with embedded documents

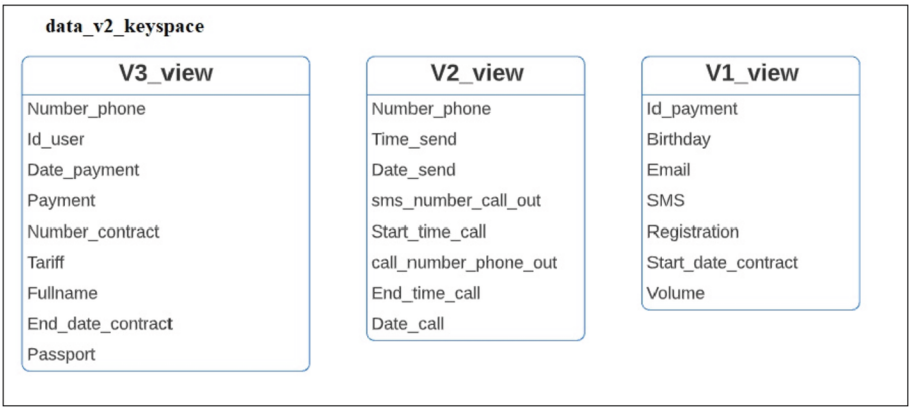

Fig. 6. Cassandra database structure

Table 1. Test results

Queries	Database type					
	I	II	III	IV	V	VI
Q1	184	114	121	104	132	106
Q2	172	84	158	125	144	127.3
Q3	465	363	117	97	121.2	103
Q4	389	282	188	110	137	119
Q5	245	219	174	127.1	125	121
Q6	718	620	179	156	148.4	94.3
Q7	652	407	172	133	166	142.8
Q8	712	457	223	145	172	157
Q9	846	550	327	221	367	181.7

Based on the test results, we created a diagram (Fig. 7), which reflects the difference in the execution time of queries for the database, depending on the type of database, query, database architecture. The diagram is built for data volume 3. In Fig. 8 there are some of the most interesting graphs illustrating the effectiveness of this approach depending on the amount of data. From the diagrams in Fig. 7 it can be seen that the approach presented in this article is the best for all cases.

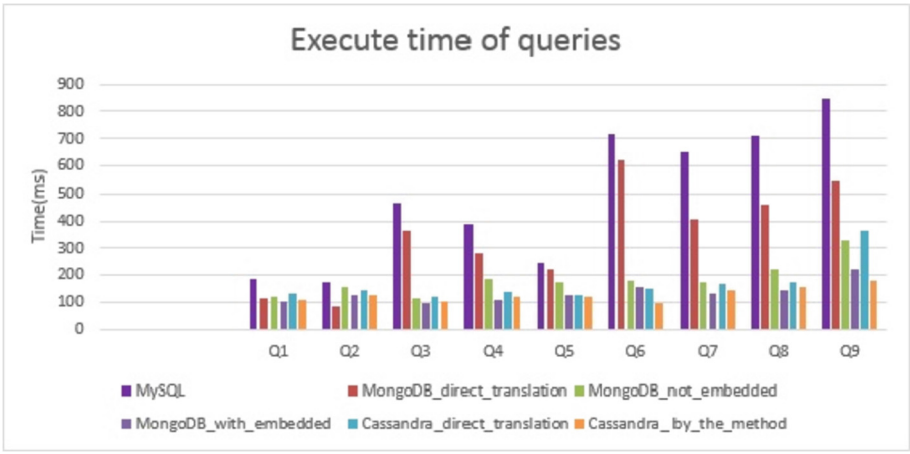

Fig. 7. Query execution time chart

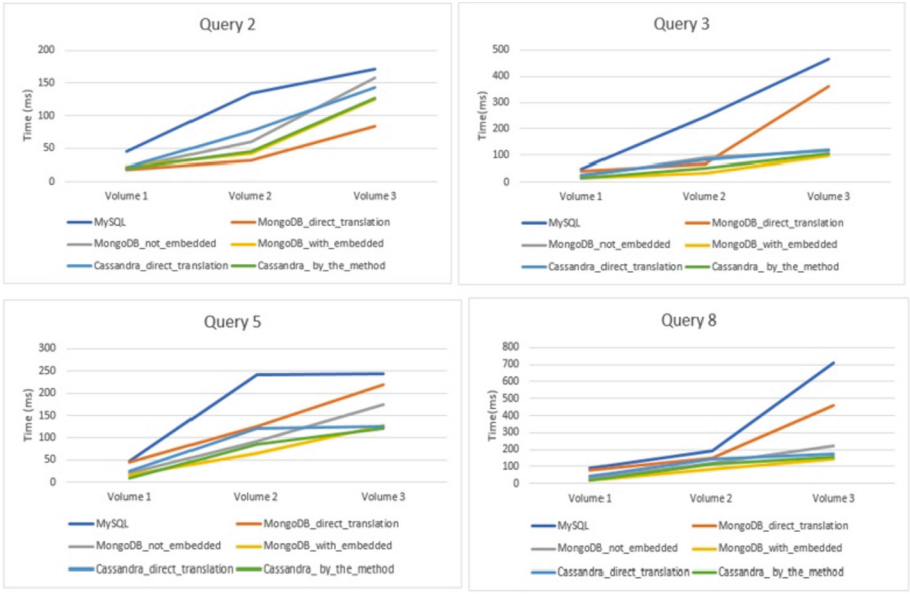

Fig. 8. Execution time of queries depending on the size of the database

It is possible to see from the graphs in Fig. 8 that the approach works differently for different queries. And in general, for a very large amount of data, the results obtained for a MongoDB database with embedded documents are much better than for a database without embedded documents, and even more so a direct correspondence of collections of a relational database in normal Boyce-Codd form. But, unfortunately, it can also be seen that with an increase in the amount of data, the approach to denormalization and adaptation of the database for queries in the Cassandra DBMS approaches in its performance a database similar in terms of the relational database scheme in the Boyce-Codd normal form.

5 Conclusion

As a result of our research, we obtained an approach that allows, based on information about the relationships between the attributes of objects in the database and the queries that will be performed most often, to build an optimal database architecture in terms of execution time for models: relational, document type, and column families. The presented approach is based on set theory and allows to automate the process of creating a database of these types. Taking into account the structure of queries to the database allows to determine the number and composition of collections for the MongoDB DBMS, tables for relational databases or column families for Cassandra, so that it is possible not to do "joins" of database structural units in the future. This method can be used when translating a database from SQL to MongoDB or Cassandra, to define collections or column families in a new MongoDB or Cassandra database, and when consolidating databases of different structures.

This method can also be applied to distributed databases. In the latter case, the distributed system architecture is taken into account besides the database schema. This approach developed by us is described in [20].

There are other possibilities for using this method, such as creating temporary collections, etc. The scientific group, which includes the authors of this article, also developed methods for automatic translation of queries from the format of one database to the format of another database (including conversion of database types), as well as methods for parallelizing queries and indexing data based on the CW-tree [19].

Further research. At the moment, research continues in the direction of supplementing the developed methods with information about a distributed database, in particular about the sharding and replication system. In the near future, we will also consider the problem of reindexing databases obtained using this approach. And finally, it is planned to take into account information about the possible parallelization of queries using methods developed by the authors based on graph theory.

Acknowledgment. The work was done within the research "Methods of constructing co-evolving intelligent systems and their prototyping in applied fields" in order to implement the ETU "LETI" Strategic Academic Leadership Program "Priority-2030", Agreement N 075-15-2021-1318.

References

1. Aggoune, A., Namoune, S.: A method for transforming object-relational to document-oriented databases. In: 2020 2nd International Conference on Mathematics and Information Technology (ICMIT), pp. 154–158 (2020). https://doi.org/10.1109/ICMIT47780.2020.9047011
2. Alotaibi, O., Pardede, E.: Transformation of schema from relational database (RDB) to NoSQL databases. Data **4**, 148 (2019). https://doi.org/10.3390/data4040148
3. Ceresnak, R., Kvet, M.: Comparison of query performance in relational a non-relation databases. Transp. Res. Procedia **40**, 170–177 (2019). https://doi.org/10.1016/j.trpro.2019.07.027
4. Chopade, R. , Pachghare, V.: MongoDB indexing for performance improvement. In: Tuba, M., Akashe, S., Joshi, A. (eds.) ICT Systems and Sustainability. Advances in Intelligent Systems and Computing, vol. 1077. Springer, Singapore (2020). https://doi.org/10.1007/978-981-15-0936-056
5. Codd, E.F.: A relational model of data for large shared data banks. Assoc. Comput. Mach. **13**, 377–387 (1970). https://doi.org/10.1145/362384.362685
6. Most popular databases. https://insights.stackoverflow.com/survey/2020#technology-databases-all-respondents4. Accessed 18 Mar 2022
7. Eyada, M.M., Saber, W., El Genidy, M.M., Amer, F.: Performance evaluation of IoT data management using MongoDB versus MySQL databases in different cloud environments. IEEE Access **8**, 110656–110668 (2020). https://doi.org/10.1109/ACCESS.2020.3002164
8. Fouad, T., Mohamed, B.: Model transformation from object relational database to NoSQL document database. In: Proceedings of the 2nd International Conference on Networking, Information Systems & Security, p. 5. Association for Computing Machinery, New York (2019). https://doi.org/10.1145/3320326.3320381
9. Fouad, T., Mohamed, B.: Model transformation from object relational database to NoSQL column based database. In: Proceedings of the 3rd International Conference on Networking, Information Systems & Security, p. 5. Association for Computing Machinery, New York (2020). https://doi.org/10.1145/3386723.3387881
10. Jose, B., Abraham, S.: Performance analysis of NoSQL and relational databases with MongoDB and MySQL. Mater. Today Proc. **24**, 2036–2043 (2020). https://doi.org/10.1016/j.matpr.2020.03.634
11. Khine, S.N., Tun, Z.: MongoDB on cloud for weather data (Temperature and Humidity) in Sittway. In: 2020 IEEE Conference on Computer Applications (ICCA), pp. 1–6, Yangon, Myanmar (2020). https://doi.org/10.1109/ICCA49400.2020.9022825
12. Sebastian, L., Henri, P.: Relational database schema design for uncertain data. Inf. Syst. **84**, 88–110 (2019). https://doi.org/10.1016/j.is.2019.04.003
13. Diogo, M., Cabral, B., Bernardino, J.: Consistency models of NoSQL databases. Future Internet **11**, 43 (2019)
14. Ramya, M., Thirumahal, R.: Hybrid query system. Int. J. Comput. Sci. Eng. **7**, 8–11 (2020). https://doi.org/10.14445/23488387/IJCSE-V7I5P103
15. Leonardo, R., Fernando, V., Elder, C., D´arlinton, B., Fernando, M.: A framework for migrating relational datasets to NoSQL1. Procedia Comput. Sci. **51**, 2593–2602 (2015). https://doi.org/10.1016/j.procs.2015.05.367
16. Singh, S., Jenamani, M.: Cassandra-based data repository design for food supply chain traceability. VINE J. Inf. Knowl. Manag. Syst. (2020). (ahead-of-print), https://doi.org/10.1108/VJIKMS-08-2019-0119
17. Sug, H.: Efficient checking of functional dependencies for relations. J. Phys. Conf. Ser. **1564**, 012011 (2020). https://doi.org/10.1088/1742-6596/1564/1/012011

18. Yulia, S., Muon, H.: Creating collections with embedded documents for document databases taking into account the queries. Computation **8**, 45 (2020). https://doi.org/10.3390/computation8020045
19. Shevskiy, V.S., Shichkina, Y.A.: Investigation of CW-tree for parallel execution of queries to database. In: Gervasi, O., et al. (eds.) ICCSA 2021. LNCS, vol. 12956, pp. 324–335. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-87010-2_23
20. Ha, M., Shichkina, Y.: Translating a distributed relational database to a document database. Data Sci. Eng. **7**, 136–155 (2022). https://doi.org/10.1007/s41019-022-00181-9