ISSN 2522-9818 (print)
ISSN 2524-2296 (online) Innovative technologies and scientific solutions for industries. 2022. No. 2 (20)

UDC 004.657; 004.051; 004.413.5 DOI: https://doi.org/10.30837/ITSSI.2022.20.052

O. MAZUROVA, I. SYVOLOVSKYI, O. SYVOLOVSKA

NOSQL DATABASE LOGIC DESIGN METHODS FOR MONGODB AND NEO4J

Modern developers of gaming mobile and Internet applications almost do not imagine themselves without the use of NoSQL databases, if they pursue the goal of creating scalable databases with high-performance and wide functionality. When designing a database for any NoSQL system, the developer needs a clear understanding of the logic of such databases and the capabilities of the tools offered by the corresponding DBMS. However, unfortunately, they do not have unified methods of logical design of such models, as in relational databases. Thus, there is a problem of developing effective methods for the logical design of such databases that would provide the necessary performance when implementing the business logic of the corresponding applications. The **subject** of the research is approaches to the logical design of NoSQL document and graph databases. The **goal** of the work is to propose unified logical modeling methods for MongoDB and Neo4j NoSQL systems based on an experimental study of their performance. The following tasks are solved in the work: analysis of current approaches to the logical design of document and graph databases\, the development of logical design methods for them; planning and experimental study of the performance of the proposed methods on the example of models developed with their help. The following **methods** are used: database design methods, database performance evaluation methods, development methods are based on MongoDB 5.0.5, Neo4j 4.4.3 DBMS, Visual Studio 2022 development environment. The following **results** are obtained: unified logical design methods for MongoDB and Neo4j NoSQL systems are proposed; on their basis, the corresponding logical models have been developed; experimental measurements of the number of resources required working with the developed models; recommendations on the proposed methods are formed. **Conclusions:** The proposed modeling methods for MongoDB have their own aspects of their effective use for different types of applications. The strengths and weaknesses of both methods were identified, but a mixed method based on a combination of modeling through normalization and denormalization was recommended. Even though Neo4j lost out to MongoDB in terms of consumed resources in most experiments, both DBMS's' demonstrate good productivity, taking into account the orientation to different tasks.

Keywords: database; logical design method; DB DESIGN; Neo4j; NoSQL; MongoDB.

Introduction

The amount of data on the Internet is growing at an enormous rate as active users add hundreds of gigabytes of data to social networks every second. Relational databases cannot cope with such modern masses of information, although data processing tasks have been successfully implemented for several decades.

This problem has led to the need to introduce new approaches to information processing in large systems. To date, NoSQL databases have met this challenge [1 - 2], which have made it possible to replace costly vertical scaling with efficient horizontal scaling on clusters. In addition, they have higher performance, more flexible data model, and open source DBMS code.

Now, the most popular NoSQL databases are document databases, in particular MongoDB, rapidly catching up with popular relational databases of Microsoft SQL Server, Oracle, MySQL and PostgreSQL [2]. In addition, when creating large systems, particularly for social networks, well proven graph DBMS, namely the most common DBMS Neo4j [3], which has a very wide functionality.

When designing a database for any NoSQL system, a developer is required to have a clear understanding of the logic of the database and the tools that DBMS offers [4]. Since this understanding may not happen in practice, many commercial projects hesitate to switch to new NoSQL databases, because the implementation of such a switch requires a lot of time for performance modeling and information migration.

Algorithms for transition from ER diagrams to logical models in the context of relational DB [5] have long been formalized. However, these algorithms are not applicable to NoSQL databases, which are based on data structures other than tables (relations).

Consequently, to solve the problem faced by developers of NoSQL databases, the task of developing more unified methods of logical modeling of such databases and their experimental study in order to identify more productive design methods and form certain recommendations on their application for different tasks and applications is relevant.

Analysis of recent research and publications

Fast and widespread distribution of NoSQL (DB) databases is due to the ease of working with them. NoSQL DB is convenient to use for many modern applications that aim to use scalable databases with high performance, wide functionality, ability to provide maximum usability [4, 6]. For such as mobile, gaming, Internet applications, etc.

Document-oriented DBs are quite common among NoSQL systems. They allow developers to store and query data in the DB using the same document model they used in the program code. Each stored record looks like a separate document with its own set of fields. Documents are flexible and hierarchical, allowing them to evolve to meet the increasing needs of applications. MongoDB, CouchDB, and Couchbase are all examples of the most common document DBMSs. They aim to provide functional and intuitive APIs for agile development. Among them, MongoDB is not only one of the most popular (or widespread), but also very attractive for developers due to the availability of drivers for different programming languages [1 - 2].

Another rather popular type of NoSQL DB is graph DB [3]. Graph DBs implement data representation in the form of nodes and edges, which are relations between

ISSN 2522-9818 (print)
Сучасний стан наукових досліджень та технологій в промисловості. 2022. № 2 (20) ISSN 2524-2296 (online)

nodes. Such DBs implement easy processing of complex related data and computation of specific properties of graphs, such as the path from one node to another and its length. Common examples of the use of graph DBs include social networks and services; Neo4j is currently the most common on this class of DBMS, since it supports a purely graph model and is already a proven development for production solutions.

DB theory and practice have long established a stage-by-stage approach to their design through conceptual, infological (or ER-) modeling to logical and then physical modeling [7 - 9]. For relational DBs, all transitions from one model to another all transitions from one model to another have been long formalized and unified. However, unfortunately, for NoSQL systems such unified methods of logical design, where it is necessary to take into account the peculiarities of the logic of such systems, do not exist today. For example, usually NoSQL DBs do not involve relational links, so the implementation of similar logic and data integrity mechanism is entirely up to the developers of the corresponding DB.

The current recommendations and approaches do not give developers for NoSQL systems any knowledge about how to model entities and relationships effectively for a particular data model, which data indexes work best, and so on [2, 6, 8]. For example, MongoDB recommends using the "Manual reference" method to create similar logic to links [10 - 11]. [10 - 11], which involves saving the "_id" field of one document to the field of another object, similar to the foreign key in relational DBs, but without supporting the link itself. This method forms a 0:M relation, which can be used by developers as 1:1, 1:M relations and derivatives thereof. But this leads to the "N+1" problem, as it requires an additional query or join data through a JOIN-like operation. Accordingly, document DBs need to use composition in the form of nested objects or arrays of objects to solve such problems. This approach is suitable if the relationship between objects can be expressed by the word "includes".

This approach can be used to model relationships:

- 1:1 type, but it should be considered that embedded object would increase the weight of the document, which slows down its unloading from DB to the client.
- 1:M type, but if M is not a particularly large number and embedded objects should not be too large.

Keep in mind that MongoDB has a maximum nesting size of 100 levels; the maximum document size is 16 MB. Consequently, if new records are constantly being added to the document field (array), the document size will keep growing. This can cause performance problems by moving the document to a different memory location, because there is no place for it to grow in the current location, so defragmentation is performed.

No joins means no JOINs in the relational sense. However, later, MongoDB added two ways of combining data:

- $lookup – an operation that works analogous to LEFT OUTER JOIN in relational DB (added in version 3.2);
- $graphLookup – creates a collection of records showing a hierarchy of objects from some to the current one, similar to lookup in graph DBs (added in version 3.4).

This approach can be used to model relationships:

- 1:1 type, but you have to consider that an embedded object will increase the weight of the document, which slows down its unloading from DB to the client.
- 1:M type, but if M is not a particularly large number and embedded objects should not be too large.

Things are more complicated with entities that have an M:M relation. It is known that an M:M relation can be defined as two 1:M relations and an intermediate object containing identifiers of those two referenced entities [12]. This approach can be implemented in MongoDB without much trouble, except for creating field indexes with identifiers. With this approach, all the auxiliary attributes of the M:M link will be located in a separate object.

But if a developer needs to connect such a link data, he would have to use two JOIN-like operations ($lookup), which in the context of document DB is very expensive.

To solve such a problem, MongoDB practically always uses another approach: compositing this intermediate object into one of the M:M link objects in the form of an array. Figure 1 shows both approaches: via auxiliary entity (top) and reduced composition approach (bottom).

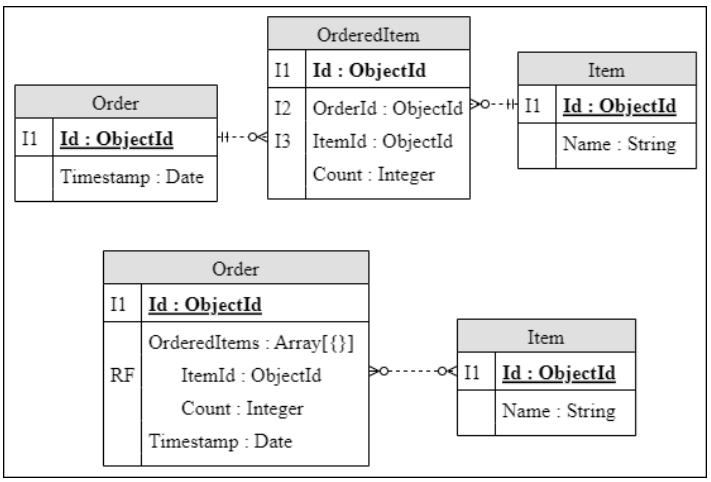

Fig. 1. Approaches to designing M:M communication in document DBs

ISSN 2522-9818 (print)
ISSN 2524-2296 (online) Innovative technologies and scientific solutions for industries. 2022. No. 2 (20)

The widespread use of the second approach is due to the fact that in 2017 the $lookup operation gained support for using arrays of identifiers as inputs to connect data. This approach requires only one JOIN operation instead of two to connect the data. But, when using it, you need to clearly identify the "main" M:M connection object, which will contain the identifier array.

Unlike document databases, graph databases support links, although they are quite different from relational links [13]. In Neo4j, each relationship is an entity of a special type that preserves a reference to the outgoing and incoming entities. Thus, links have names, can contain attributes, and indexes can be created on them. Like all NoSQL databases, this DBMS has no integrity restriction mechanisms; this must be decided by the developer at the application level only. But, each relationship in the graph must have a source and an input entity.

It follows that every Neo4j relationship has a default cardinality of 1:M, which can be "transformed" into a 1:1 relationship due to uniqueness constraints or at the software level. Thus, an M:M relationship can potentially be modeled in two ways: through an auxiliary entity (as in relational DBs) and directly by storing additional data as attributes of the relationship. Figure 2 illustrates these modeling approaches graphically: as it looks in relational DB (top), auxiliary entity (middle), and via link attributes (bottom).

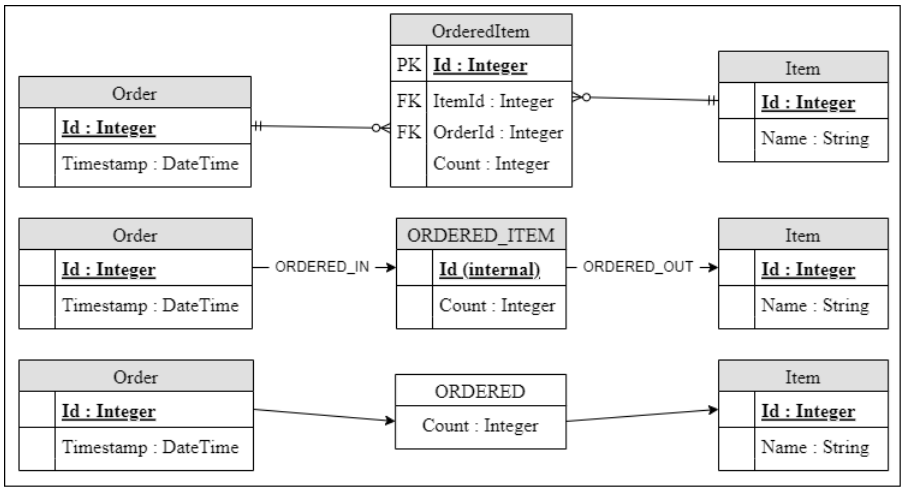

Fig. 2. Comparison of M:M communication design methods in graph DBMS

It should be noted that almost all graph DBMSs have only unidirectional links. The standardized query language Gremlin, supported by all graph DBMSs, also has no support for bidirectional links. Thus, to model bidirectional links, you need to make two bindings in both directions. But considering that a link is also one of the DBMS objects, the option with the intermediate entity is ineffective from the very beginning, as it strongly clogs the DB with redundant entities and links, increases the weight of the DB due to redundant objects and potentially increases the execution time even for basic queries.

However, the existence of the considered recommendations and approaches does not provide NoSQL DB developers with unified methods of logical design of such DB, which would unambiguously indicate the effectiveness of the model obtained in the end. Thus, the study of NoSQL database logical modeling methods and approaches is relevant.

The **aim** of this article is the development of unified logical design methods for NoSQL systems MongoDB and Neo4j based on the analysis of existing design approaches, as well as experimental study of their performance.

This research requires:

- development of unified logical design methods for selected NoSQL databases MongoDB and Neo4j;
- analysis and infological modelling of a certain applied subject area of creation of complex server systems for further experimental research;
- developing of the logical models for the selected DBMS on the basis of the developed unified logical design methods;
- experimental study of the performance of the obtained models and development of recommendations on the feasibility of using the proposed methods in the design of NoSQL databases.

Evaluation of the effectiveness of the methods should be made taking into account such criteria as: disk space occupied by DB (MB); query execution time (ms); operating memory consumption (MB); CPU time consumption (%).

Materials and methods

For further study, the applied subject domain of an arbitrary game server system was chosen. A multiplayer action-adventure game with RPG elements and a dedicated server was chosen as the subject domain object. In games of similar genre and implementation of multiplayer, in any case, it is necessary to implement DB for storing world state and player progress. Consequently, the database must store the following information:

- player account information (currency, player data);
- status and information about the characters in the game world and their abilities;
- a list of the character's tasks and their status;

Сучасний стан наукових досліджень та технологій в промисловості. 2022. № 2 (20)
ISSN 2522-9818 (print)
ISSN 2524-2296 (online)

- a list of enemies (monsters) in the game and related information (or information about their location, if the server part generates them);
- a list of non-player characters (NPCs) and related information;
- history of events in the game (buying in-game currency, defeating enemies, completing tasks, etc.).

A general diagram of the domain classes, describing the essence of the game system and the relationship between them, is given in figure 3.

Based on this diagram, as well as the identified integrity constraints and attributes of the subject area, an ER diagram [12] of DB (fig. 4) based on the "Crow's foot" notation was developed. [9].

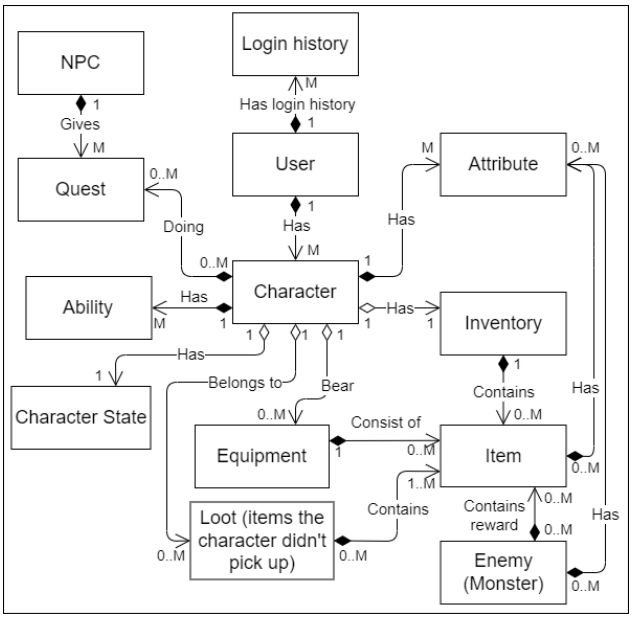

Fig. 3. General class diagram of the subject area

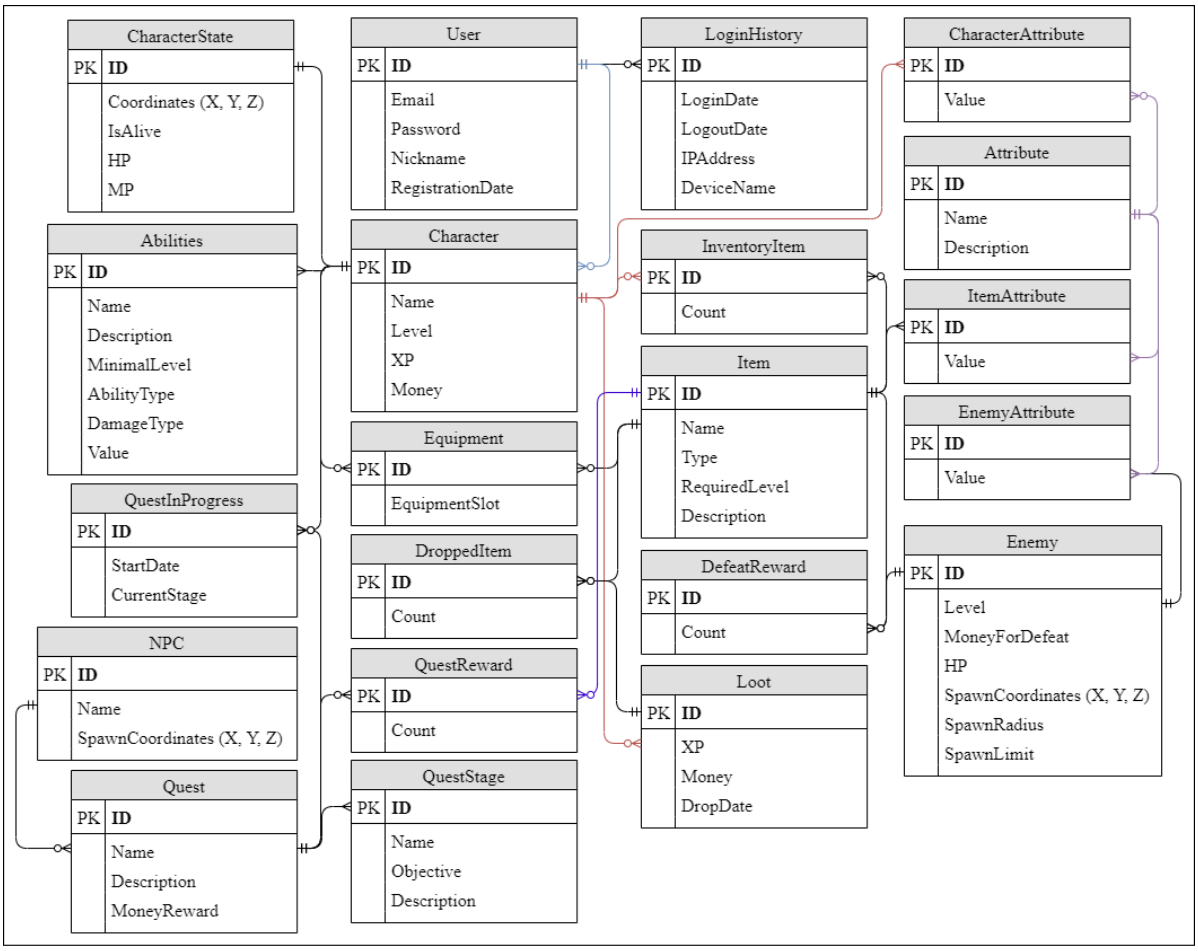

Fig. 4. ER-diagram of the subject area database

ISSN 2522-9818 (print)
ISSN 2524-2296 (online) Innovative technologies and scientific solutions for industries. 2022. No. 2 (20)

So, let us consider the methods by which the logical DB design for DBMS MongoDB can be performed. Recall that there is no standardized notation for the visualization of this model. What also complicates the process is that MongoDB objects can have up to 100 levels of nesting and is problematic to reflect visually, although this does not happen very often in practice. Therefore, a modification of the notation for relational logical models with additional functionality inherent in document DB is proposed to describe the document logical model.

Consequently, the mentioned "Manual reference" approach essentially makes the model more similar to relational DBs, so it can be denoted as a "normalizing" method. In contrast, the nested document approach reduces the level of normalization through composition, so it can be called "denormalizing".

Let us consider a unified method for turning an ER diagram into a normalized document logical model, in which the following steps have been proposed:

- modeling entities participating in a 1:1 relationship: add a field with the identifier of one document (master) to another document (dependent).

- modeling entities participating in a 1:M relationship: add a field with the identifier of the main document (1) to the dependent ones (M);

- modeling entities participating in the M:M relationship should use one of the previously mentioned approaches: either through isolating an intermediate entity with identifiers of objects referring to it (more often inefficient), or through composing this intermediate entity. Entity in the "main" object as an array (usually efficient).

The normalized logic model designed by this method is shown in figure 5. All "conditional" external keys constructed by "manual reference" are marked with RF in the figure. The resulting links have purely conditional character due to the fact that MongoDB has no integrity restriction mechanisms and document binding functionality in general. The task of data integrity control is entirely up to the developer.

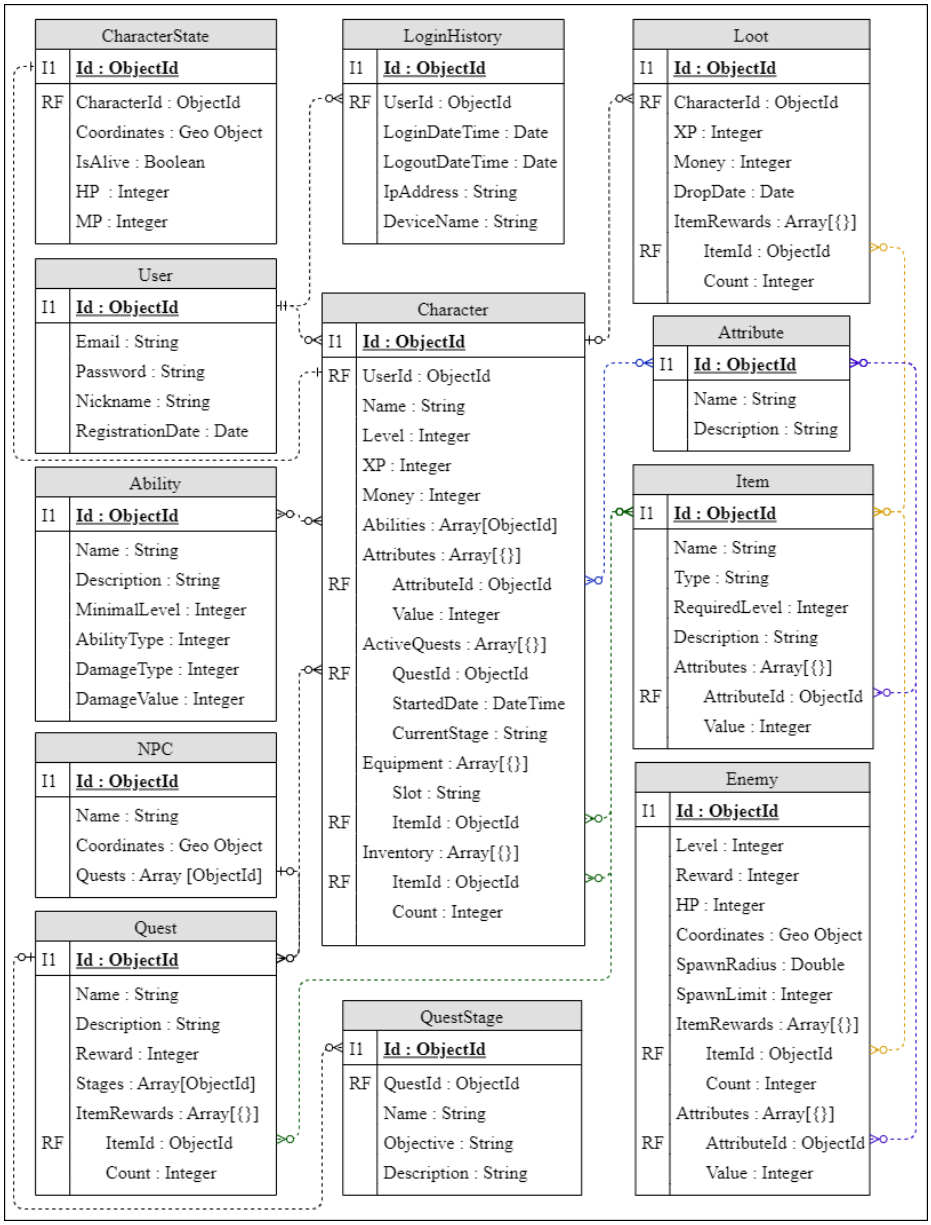

Fig. 5. Normalized DB document logical model of the subject area

ISSN 2522-9818 (print)
Сучасний стан наукових досліджень та технологій в промисловості. 2022. № 2 (20) *ISSN 2524-2296 (online)*

Consider a unified method for transforming an ER diagram into a denormalized document logical model (fig. 6), in which the following steps have been proposed:

- modeling entities participating in 1:1 relationships: create a field in the main document and nest the dependent document in it, followed by deleting the dependent entity; it is recommended to add an index to it if you plan to select these entities separately from the main entity;

- modeling entities participating in 1:M relationships: add a field-array to the main document (1) containing all dependent (M) entities. If semantically the main entity may have no relations to all dependent entities (no "owns" relation), a separate collection without relations must be created to contain all instances of dependent entities. Otherwise, an additional entity is not required.

- modeling entities that participate in M:M relationships: to add a field-array to the main document that contains all dependent documents. The cases in which an additional collection needs to be created are similar to 1:M.

Guided by this method, a denormalized document logical model of the domain was designed (fig. 6).

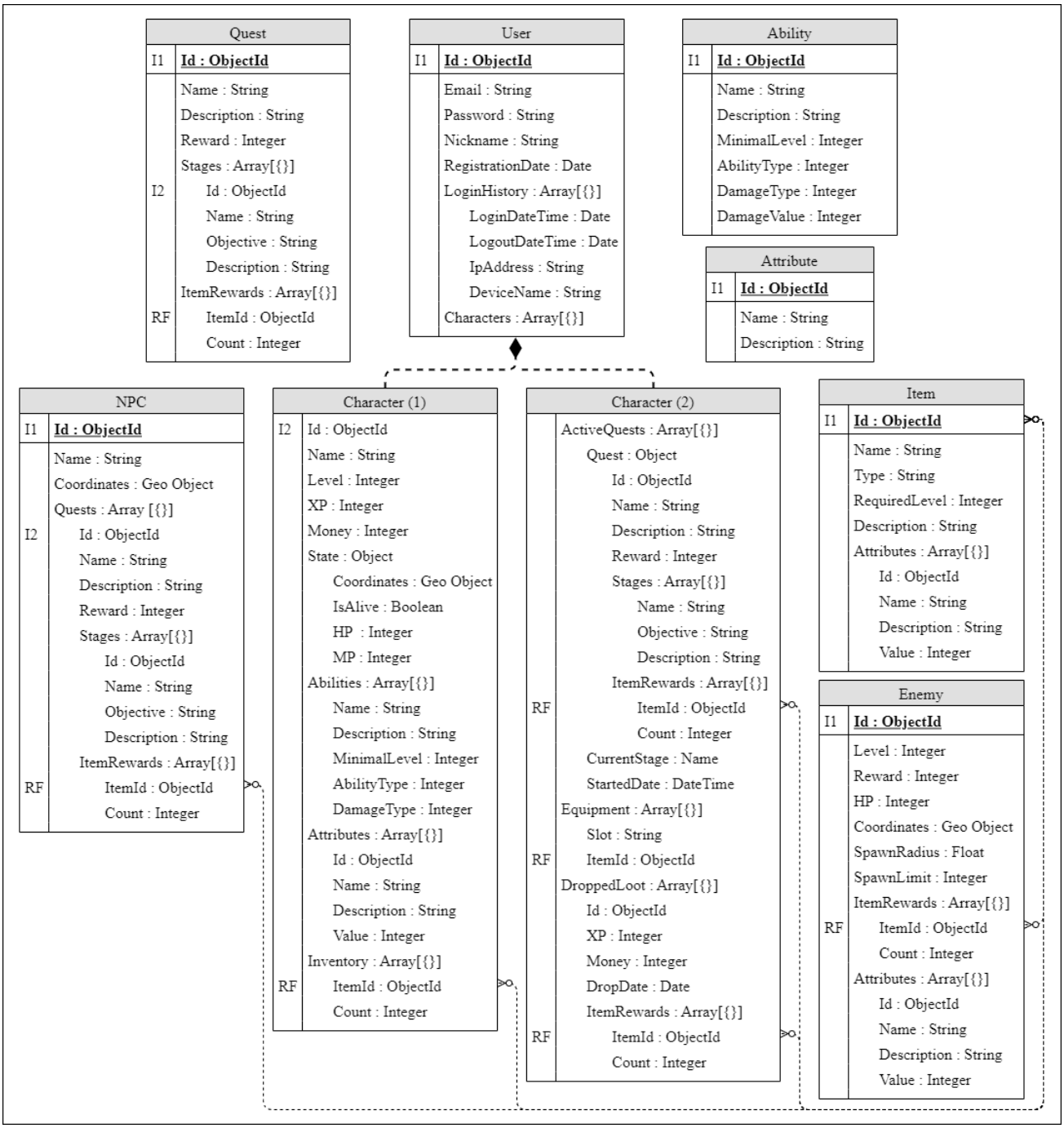

Fig. 6. Denormalized DB document logical model of the subject area

For a more convenient visual representation of the model, the nested Character object has been separated and split into two parts. To denote the nesting of Character in User, the connection with UML class diagrams "including" was used. The only entity that did not undergo denormalization was Item. This is due to the fact that in

ISSN 2522-9818 (print)
ISSN 2524-2296 (online) Innovative technologies and scientific solutions for industries. 2022. No. 2 (20)

game applications of this genre all items are accessed by its identifier and the set of all items is unloaded at the start of the game. Thus, JOIN-like operations with the Item entity will not be performed in practice and there will be no difference in performance either.

Next, let's propose a logical design method for the graph model. There is no standardized notation for constructing a logical model of this type now either, so a relational modification will be used. It should be noted that Neo4j supports attributes in relationships, which can significantly reduce the number of entities and simplify the model.

The algorithm for turning an ER diagram into a graphical logic model is very different from the previous ones because of the different structure of data storage. The following steps are proposed for it:

- to combine entities that have 1:1 relationships with each other into one entity;
- to turn 1:M links into graph links without attributes.
- to replace intermediate entities that create M:M links with graph links (with attributes, if any).

Figure 7 shows the graphical logic model obtained as a result of the proposed method.

In the developed model there are two types of links: with and without attributes. A separate notation in the form of a transparent block was proposed to display links with attributes. The use of attribute relationships eliminated all the entities that were used to model the M:M relationship, which reduced the model considerably. But since graph DB does not support nesting of entities, the 1:1 link must be maintained at the program level [13], such as the link between a character and its state.

Thus, as a result of the analysis and modeling of the subject area, logical models were developed: normalized and denormalized document and graph models. Based on these models, the corresponding physical DB models for the corresponding DBMS MongoDB and Neo4j were developed for further study.

For the experimental study the clusters from DB servers or source-replica type replication were used, as this approach is suitable for game servers with large read specificity. Consequently, all measurements were performed on clusters of database servers regardless of configuration. They were located in the Azure cloud service on virtual machines of different sizes.

Thus, the following DB server configurations with their characteristics were chosen for the experiments:

- configuration type Small: machine name – Standard_B2s; vCPU - 2; RAM - 4 GB; number of nodes - 2; number of connections 20;
- configuration type Medium: machine name is Standard_B4ms; vCPU - 4; RAM - 16 GB; number of nodes - 4; number of connections 50;
- configuration type Large: machine name – Standard_B8ms; vCPU - 8; RAM - 32 GB; number of nodes - 6; number of connections 100.

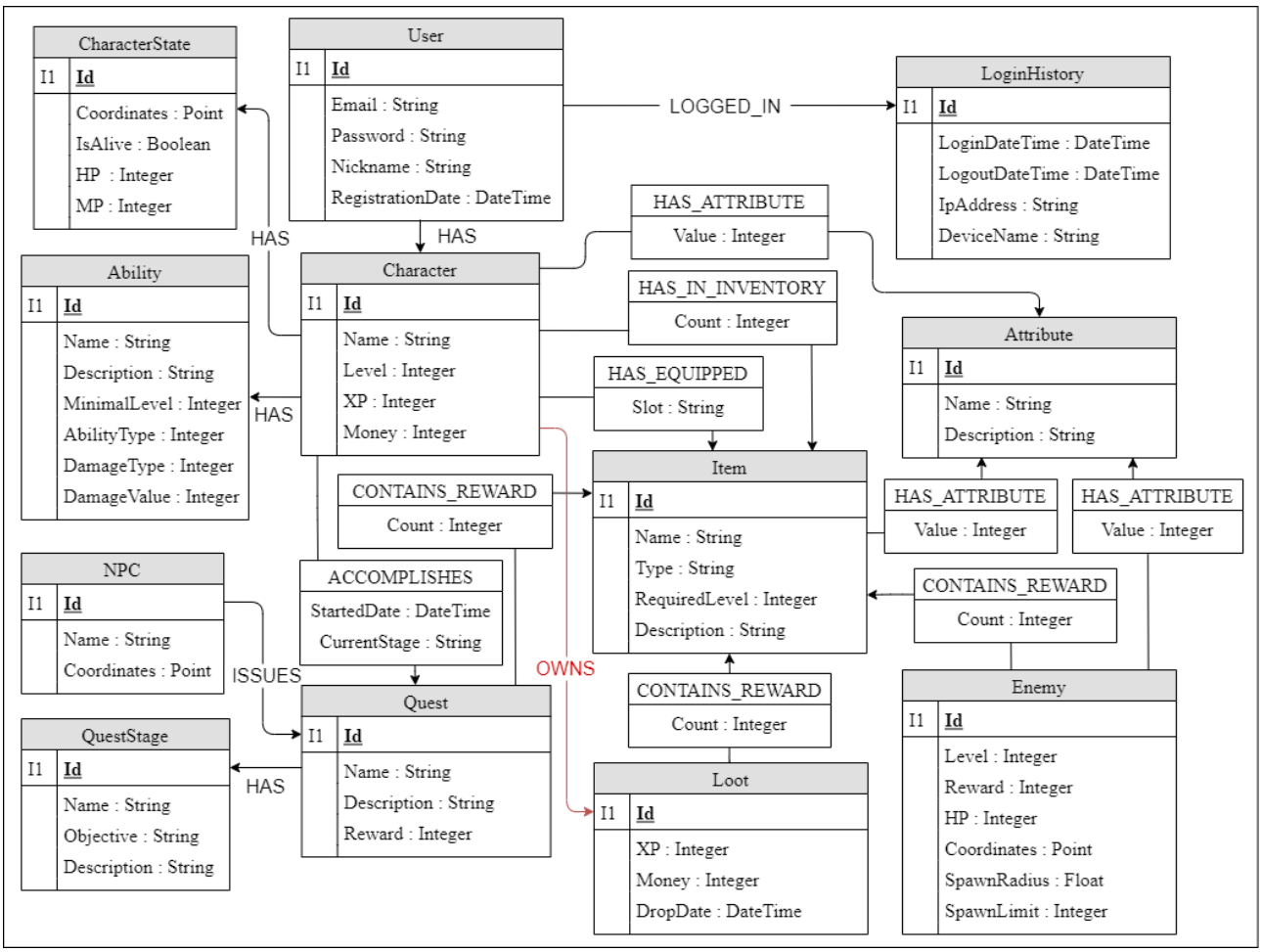

Fig. 7. Graphical logic model of the subject area database

ISSN 2522-9818 (print)
Сучасний стан наукових досліджень та технологій в промисловості. 2022. № 2 (20) ISSN 2524-2296 (online)

Ubuntu 20.04 LTS Minimal was used as the operating system to minimize the consumption of resources by the system. The type of machine configuration also affects the number of entities in the DB to be used in the experiments (table 1).

Table 1. *Number of DB entities for experiments on configurations*

Type of configuration	Small	Medium	Large
Users / characters per user	3000 / 1	5000 / 2	8000 / 3
Number of entries in the game per user	25	50	75
Total items in inventory of the players	100 / 50	200 / 100	300 / 150
Skill / equipment slots	10 / 4	25 / 6	50 / 8
NPCs/tasks they issue	50 / 100	150 / 200	300 / 500
Enemies	100	200	500
Loot / number of items within it	5000 / 3	15000 / 5	30000 / 8
Number of DB items in the "worst case" (denormalization)	776 967	4 701 983	16 362 659

Based on games of a similar genre, it was taken into account that some entities cannot be in large numbers and do not change depending on the configuration, for example: Attribute (a constant number - 6 was chosen); CharacterState (one entity per character).

When performing each step of the experimental study, it was decided to collect metrics that are quite often used to investigate DB performance [14 - 15]:

- S – space occupied by DB on disk (MB);
- M – RAM consumption (MB);
- C – processor time consumption (%);
- T – query execution time (ms) (the results of experiments to measure this metric will be given in the further publications);

Results of research and their discussion

Let us consider the main most interesting performance trends of the experiments to study the designed models for NoSQL DBMS MongoDB and Neo4j.

First of all, let's compare DB sizes with filled test data, which is shown in figure 8.

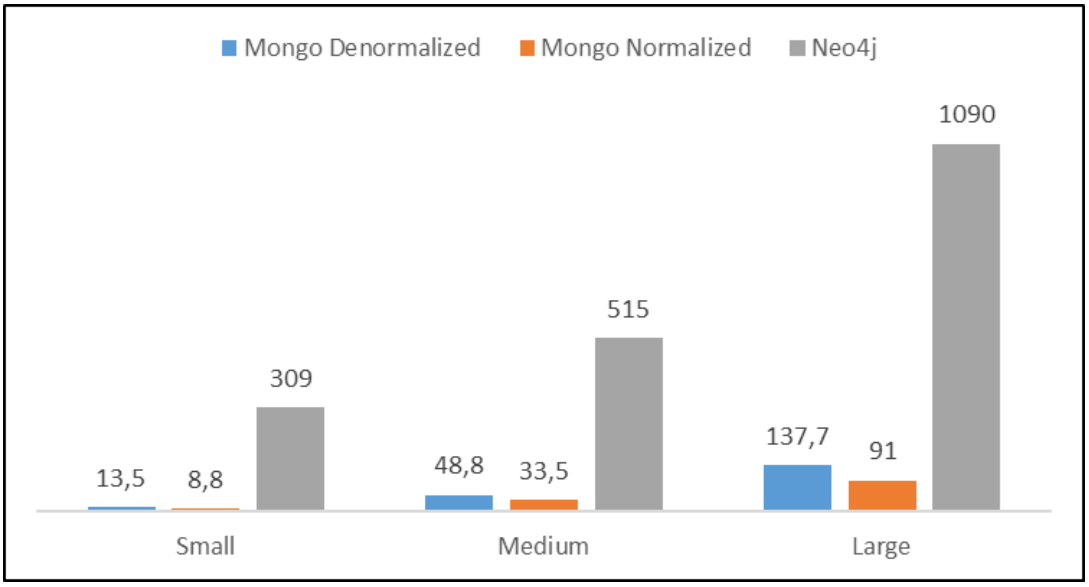

Fig. 8. DB size comparison (on disk)

The diagram shows that DBMS Neo4j consumes a huge amount of disk space, and this growth is almost linear to the number of entities. In the course of experiments it was determined that this "DB weight" is formed by entities, the connections themselves practically do not take up disk space. The denormalized MongoDB model weighs 30-35% more than the normalized one, which is obviously caused by data redundancy. Nevertheless, in terms of DB weight MongoDB clearly wins over Neo4j.

The results of the comparison of RAM consumption of the DBMS server are shown in figure 9.

ISSN 2522-9818 (print)
ISSN 2524-2296 (online) Innovative technologies and scientific solutions for industries. 2022. No. 2 (20)

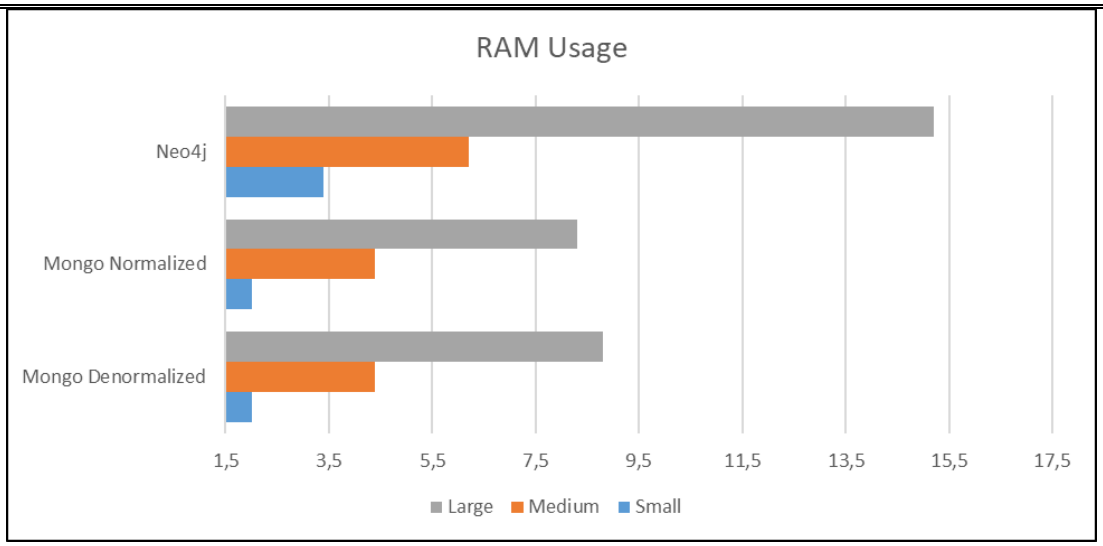

Fig. 9. RAM usage

The histogram shows that Neo4j uses even more RAM than MongoDB. In this situation, the MongoDB memory allocation logic still plays a role: DBMS cannot use more than 50% of the system RAM, while Neo4j, if necessary, can use almost all available to it. Also, Neo4j has conditionally minimal amount of RAM for the correct work - 2 gigabytes, when the recommended amount is about 8 gigabytes.

Let's consider the results of measurements of CPU usage by the DBMS server. Three configurations were used in the research. So, the experiments have shown that for small projects or projects at MVP stage Neo4j is not especially effective. Let's take a closer look at the Medium configuration (fig. 10), which more corresponds to the real machines configurations for medium-sized projects.

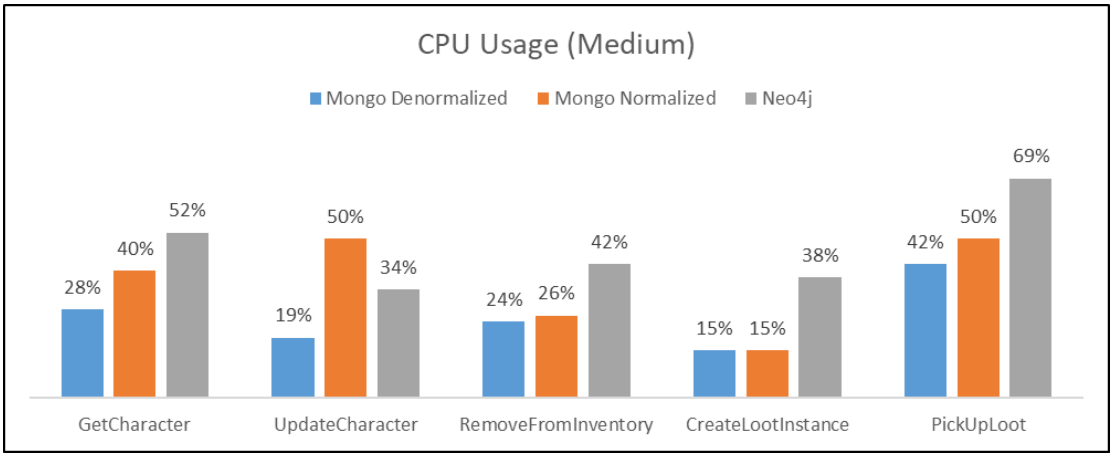

Fig. 10. CPU usage (Medium configuration)

With so many resources, the situation for Neo4j has leveled off relative to MongoDB, now the DBMS data is about equal. In general, it was on this configuration that Neo4j started working "without limitations". We can even conclude that the resources allocated to it are even a bit much for the load that was allocated to it.

Figure 11 shows the results of the measurements for the Large configuration. In general, the situation is very similar to the preliminary results. The resources increased, but the DB size and load increased proportionally to the resources.

We see that the allocated DBMS resources are more than they need for stable operation on these loads.

We also investigated the performance of the models when executing queries. But this will be a topic for another publication. Note only that during all comparisons we could see certain pattern - Neo4j consumes more resources compared to MongoDB, denormalized model works faster than normalized model in context of the queries studied as well as requires less resources.

After all experiments, we can unequivocally say that the denormalized MongoDB model is the most preferable option for the studied domain (DB model and queries). This scheme resulted in the lowest consumption of machine resources with satisfactory performance.

After comparing all the results obtained, we can draw some conclusions and develop some recommendations for the use of one or another method in a particular situation.

So DBMS Neo4j. It has been in development for 15 years, during which it has acquired a large set of functionalities, significant performance improvements and so on. But in practice, it is not so good: the limitation of Community version, huge consumption of CPU and RAM

ISSN 2522-9818 (print)
Сучасний стан наукових досліджень та технологій в промисловості. 2022. № 2 (20) ISSN 2524-2296 (online)

resources, large DBMS weight compared to other DBMS and average performance in trivial tasks make this DBMS not particularly attractive for small or medium sized projects. It is also should be noted that the demanding DBMS is also associated with its implementation of JVM, which immediately impose restrictions on the smallest RAM for the DBMS server.

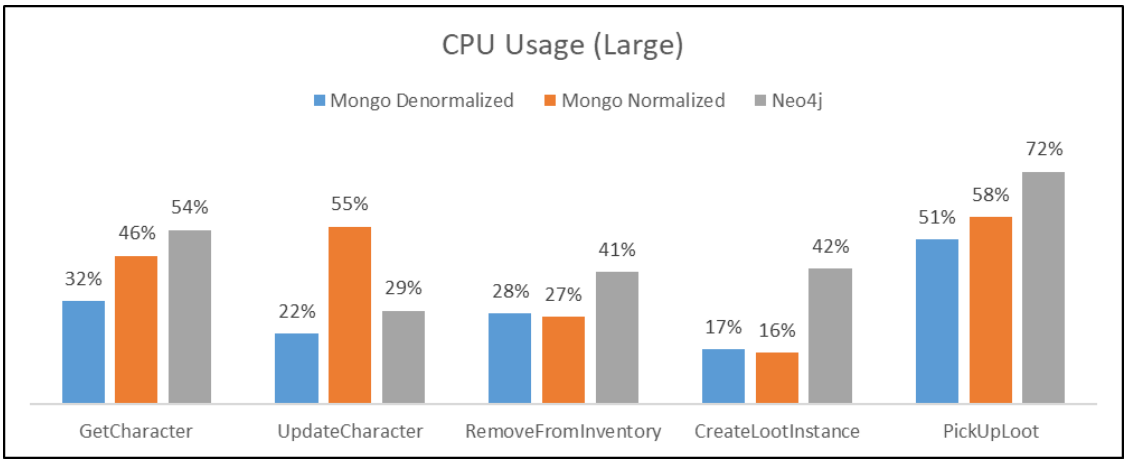

Fig. 11. CPU usage (Large configuration)

One of the peculiarities of Neo4j is that it uses the maximum of its allocated resources, so they have to be strictly limited to certain values. But it should be noted that this DBMS is able to easily perform operations that are difficult or impossible to perform in other DBMS.

Thus, it is recommended to use graph model logic design method and use Neo4j in cases when:

- ER graph DB contains a large number of M:M links and the server-side logic involves frequent fetching of several linked data simultaneously;
- an ER diagram DB has a small fraction of entities on a large fraction of links, and the application logic is mostly about deleting and adding links between DB objects;
- the system is large, has a large number of users and the company has a large amount of resources;
- the server side needs the specifications of graph DB, such as finding the depth of relationships.

Now let's move on to DBMS MongoDB. Consider first the model normalization method. Using this method resulted in zero data redundancy in the DB, which had a positive effect on weight. Also, since the DB objects are much smaller than the denormalized model, they have more "similarity" between them, DBMS more effectively applied the compression mechanism of the stored data (on average, by 5-15%).

But the analyzed operations in the server system under study often required either joining data or performing operations on several collections simultaneously, which required the use of transactions or JOIN-like operations. This resulted in reduced performance compared to the denormalized model.

Thus, the normalization method should be used if:

- the links in the schema are predominantly cardinality "0", which eliminates the need to artificially maintain data integrity through transactions (the traditional "eventually consistent" approach);
- in a 1:M relation, the number M is expected to be large (and/or the weight of the object is large). This is due to an object size limit of 16 MB;
- the system was previously using a relational RDBMS and a quick migration to MongoDB is required.

The denormalization method used in the server system under study proved to be the most efficient in terms of performance. Also data redundancy increased the weight of DB noticeably. Also some operations of the system were quite difficult to implement using array operations (and some potentially impossible), which is not typical for normalized model. This method should be used if the number of "M" objects in a 1:M relationship is not particularly large (up to 1000) or dependent objects cannot exist without the main one (simpler "artificial" data integrity support);

Conclusions and prospects for further development

In this study, NoSQL DB logical design methods were proposed and investigated in terms of performance using DBMS MongoDB and Neo4j examples. A series of experiments were conducted to measure the resources consumed.

Based on the analysis of logical design approaches, unified logical design methods for NoSQL systems MongoDB and Neo4j were proposed. For the experiment, based on the proposed methods, logical models were designed, the performance of which was investigated. The experiments used metrics on the resources required to handle such models.

The study showed that none of the proposed modeling methods for MongoDB could be called unambiguously best. The best would be a mixed method - a combination of modeling through normalization and denormalization. In general, it can be unambiguously said that both studied DBMS have good performance, although they are oriented to different tasks.

If you don't know in advance how fast the system will grow, how many users it will have, and so on, a universal choice is to use MongoDB. This DBMS has a very wide functionality and the ability to scale

ISSN 2522-9818 (print)
ISSN 2524-2296 (online) Innovative technologies and scientific solutions for industries. 2022. No. 2 (20)

horizontally and vertically, which makes it a good choice for prototypes and newly created systems.

Thus, based on the results of the experimental study, recommendations for the use of the proposed methods have been formed. These recommendations can be used to design real systems, in particular in the area of game servers.

References

1. Maran, M. M., Paniavin, N. A., Poliushkin, I. A., (2020), "Alternative Approaches to Data Storing and Processing", *V International Conference on Information Technologies in Engineering Education (Inforino)*, P. 1–4, DOI: https://doi.org/10.1109/inforino48376.2020.9111708
2. Meier, A., Kaufmann, M. (2019), *SQL & NoSQL Databases: Models, Languages, Consistency Options and Architectures for Big Data Management*, Springer Vieweg, , 248 p. DOI: https://doi.org/10.1007/978-3-658-24549-8
3. Wood, P. T. (2018), "Graph Database", In: Liu, L., Özsu, M.T. (eds), *Encyclopedia of Database Systems. Springer*, New York, NY, P. 1639–1643. DOI: https://doi.org/10.1007/978-1-4614-8265-9_183
4. Acharya, B., Jena, A. K., Chatterjee, J. M., Kumar, R., & Le, D. (2019), "NoSQL Database Classification: New Era of Databases for Big Data", *International Journal of Knowledge-Based Organizations (IJKBO)*, 9 (1), P. 50–65. DOI: http://doi.org/10.4018/IJKBO.2019010105
5. Halpin, T., Morgan, T, (2008), "Information Modeling and Relational Databases (Second Edition) ", *The Morgan Kaufmann Series in Data Management Systems*, P. 305–343. DOI: https://doi.org/10.1016/B978-0-12-373568-3.X5001-2
6. Kuzochkina, A., Shirokopetleva, M., Dudar, Z. (2018), "Analyzing and Comparison of NoSQL DBMS", *International Scientific-Practical Conference Problems of Infocommunications. Science and Technology (PIC S&T)*, P. 560–564. DOI: https://doi.org/10.1109/INFOCOMMST.2018.8632133
7. Sanders, G. L., Shin, S. K. (2001), "Denormalization effects on performance of RDBMS", *Proceedings of the 34th Annual Hawaii International Conference on System Sciences*, P. 9–15.
8. Sahatqija, K., Ajdari, J., Zenuni, X., Raufi, B., Ismaili, F., (2018), "Comparison between relational and NOSQL databases", *41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, P. 216–221. DOI: https://doi.org/10.23919/mipro.2018.8400041
9. Date, C. J. (2019), *Database Design and Relational Theory: Normal Forms and All That Jazz*, Apress, 470 p, ISBN 978-148-425-539-1. DOI: https://doi.org/10.1007/978-1-4842-5540-7
10. Palanisamy, S., SuvithaVani, P. (2020), "A survey on RDBMS and NoSQL Databases MySQL vs MongoDB", *International Conference on Computer Communication and Informatics (ICCCI)*. DOI: https://doi.org/10.1109/iccci48352.2020.9104047
11. Chodorow, K., (2016), *MongoDB: The Definitive Guide: Powerful and Scalable Data Storage, 3rd Edition*, O'Reilly Media, 514p.
12. Bagui, S., Earp, R. (2011), *Database Design Using Entity-Relationship Diagrams (Foundations of Database Design)*, Auerbach Publications, 371 p. DOI: https://doi.org/10.1201/9781439861776
13. Vukotic, A., Watt, N., Abedrabbo, T., Fox, D., Partner, J. (2014), *Neo4j in Action*, Manning, 304 p.
14. Mazurova, O., Naboka, A., Shirokopetleva, M. (2021), "Research of ACID transaction implementation methods for distributed databases using replication technology", *Innovative technologies and scientific solutions for industries*, № 2 (16), P. 19– 31. DOI: https://doi.org/10.30837/ITSSI.2021.16.019
15. Gomes, C., Borba, E., Tavares, E., Junior, M. N. de O. Performability (2019), "Model for Assessing NoSQL DBMS Consistency", *IEEE International Systems Conference (SysCon)*. DOI: https://doi.org/10.1109/syscon.2019.8836757

Received 30.06.2022

Відомості про авторів / Сведения об авторах / About the Authors

Мазурова Оксана Олексіївна – кандидат технічних наук, доцент, Харківський національний університет радіоелектроніки, доцент кафедри програмної інженерії, м. Харків, Україна; e-mail: oksana.mazurova@nure.ua; ORCID ID: https://orcid.org/0000-0003-3715-3476.

Мазурова Оксана Алексеевна – кандидат технических наук, доцент, Харковский национальный университет радиоэлектроники, доцент кафедры программной инженерии, г. Харьков, Украина.

Mazurova Oksana – PhD (Engineering Sciences), Associate Professor, Kharkiv National University of Radio Electronics, Associate Professor of the Department of Software Engineering, Kharkiv, Ukraine.

Сиволовський Ілля Михайлович – Харківський національний університет радіоелектроніки, магістр кафедри програмної інженерії, м. Харків, Україна; e-mail: illia.syvolovskyi@nure.ua; ORCID ID: https://orcid.org/0000-0002-4592-0965.

Сиволовский Илья Михайлович – Харьковский национальный университет радиоэлектроники, магистр кафедры программной инженерии, г. Харьков, Украина.

Syvolovskyi Illia – Kharkiv National University of Radio Electronics, Master of the Department Of Software Engineering, Kharkiv, Ukraine.

Сиволовська Олена Вікторівна – кандидат економічних наук, доцент, Український державний університет залізничного транспорту, доцент кафедри маркетингу, м. Харків, Україна, e-mail: alenasvl3@gmail.com; ORCID ID https://orcid.org/0000-0002-9317-9307.

Сиволовская Елена Викторовна – кандидат экономических наук, доцент, Украинский государственный университет железнодорожного транспорта, доцент кафедры маркетинга, г. Харьков, Украина.

Syvolovska Olena – PhD (Economics Sciences), Associate Professor, Ukrainian State University of Railway Transport, Associate Professor of the Department of Marketing, Kharkiv, Ukraine.

ISSN 2522-9818 (print)
Сучасний стан наукових досліджень та технологій в промисловості. 2022. № 2 (20) *ISSN 2524-2296 (online)*

МЕТОДИ ЛОГІЧНОГО ПРОЕКТУВАННЯ NOSQL БАЗ ДАНИХ ДЛЯ MONGODB ТА NEO4J

Сучасні розробники ігрових мобільних та інтернет-додатків майже не уявляють себе без використання NoSQL баз даних, якщо вони мають на меті створення масштабованих баз даних, які мають високу продуктивність та широкі функціональні можливості. При проєктуванні бази даних для будь-якої NoSQL-системи від розробника вимагається чітке розуміння логіки таких баз даних та можливостей інструментів, які пропонує відповідна СКБД. Але, на жаль, уніфікованих методів логічного проектування таких моделей, як є в реляційних базах даних, вони не мають. Отже існує проблема розробки ефективних методів логічного проектування NoSQL баз даних, які б забезпечували необхідну продуктивність під час реалізації бізнес-логіки відповідних додатків. **Предметом** дослідження є підходи до логічного проектування NoSQL документних та графових баз даних. **Мета** роботи – запропонувати уніфіковані методи логічного моделювання для NoSQL систем MongoDB та Neo4j на основі експериментального дослідження їх продуктивності. В роботі вирішуються наступні **завдання:** аналіз актуальних підходів до логічного проектування документних та графових баз даних, розробка методів логічного проектування для них; планування та експериментальне дослідження продуктивності запропонованих методів на прикладі моделей, що розроблено за їх допомогою. Використовуються такі **методи:** методи проектування та оцінки продуктивності баз даних, методи розробки базуються на СКБД MongoDB 5.0.5, Neo4j 4.4.3, середовищі розробки Visual Studio 2022. Отримано наступні результати: запропоновано уніфіковані методи логічного проектування для NoSQL систем MongoDB та Neo4j; на їх основі розроблено відповідні логічні моделі; проведено експериментальні заміри кількості **Висновки:**ресурсів, що необхідні для роботи з розробленими моделями; сформовано рекомендації щодо запропонованих методів. запропоновані методи моделювання для MongoDB мають власні аспекти ефективного використання для різних типів додатків; були виявлені сильні та слабкі сторони обох методів, але рекомендовано змішаний метод на базі комбінації моделювання через нормалізацію та денормалізацію; незважаючи на те, що Neo4j в більшості експериментів програла MongoDB за споживаними ресурсами, але обидві СКБД мають хорошу продуктивність орієнтовно до різних завдань.

Ключові слова: база даних; метод логічного проектування; СКБД; Neo4j; NoSQL; MongoDB.

МЕТОДЫ ЛОГИЧЕСКОГО ПРОЕКТИРОВАНИЯ NOSQL БАЗ ДАННЫХ ДЛЯ MONGODB И NEO4J

Современные разработчики игровых мобильных и интернет-приложений почти не представляют себя без использования NoSQL баз данных, если они преследуют цель создания масштабируемых баз данных, имеющих высокую производительность и широкие функциональные возможности. При проектировании базы данных для любой NoSQL-системы от разработчика требуется четкое понимание логики таких баз данных и возможностей инструментов, предлагаемых соответствующей СУБД. Но, к сожалению, унифицированных методов логического проектирования таких моделей, как в реляционных базах данных, они не имеют. Таким образом, существует проблема разработки эффективных методов логического проектирования таких баз данных, которые обеспечивали бы необходимую производительность при реализации бизнес-логики соответствующих приложений. **Предметом** исследования являются подходы к логическому проектированию NoSQL документных и графовых баз данных. **Цель** работы – предложить унифицированные методы логического моделирования для NoSQL систем MongoDB и Neo4j на основе экспериментального исследования их производительности. В работе решаются следующие **задачи:** анализ актуальных подходов к логическому проектированию документных и графовых баз даных, разработка методов логического проектирования для них; планирование и экспериментальное исследование производительности предложенных методов на примере моделей, разработанных с их помощью. Используются следующие **методы:** методы проектирования и оценки производительности баз данных, методы разработки базируются на СУБД MongoDB 5.0.5, Neo4j 4.4.3, среде разработки Visual Studio 2022. Получены следующие **результаты:** предложены унифицированные методы логического проектирования для NoSQL систем MongoDB и Neo4j; на их основе разработаны соответствующие логические модели; проведены экспериментальные замеры количества ресурсов, необходимых для работы с разработанными моделями; сформированы рекомендации по предложенным методам. **Выводы:** предложенные методы моделирования для MongoDB имеют собственные аспекты эффективного их использования для разных типов приложений; были выявлены сильные и слабые стороны обоих методов, однако рекомендовано смешанный метод на базе комбинации моделирования через нормализацию и денормализацию; несмотря на то, что Neo4j в большинстве экспериментов проиграла MongoDB по потребляемым ресурсам, обе СУБД демонстрируют хорошую продуктивность с учетом ориентации на разные задания.

Ключевые слова: база данных; метод логического проектирования; СУБД; Neo4j; NoSQL; MongoDB.

Бібліографічні описи / Bibliographic descriptions

Мазурова О. О., Сиволовський І. М., Сиволовська О. В. Методи логічного проектування NoSQL баз даних для MongoDB та Neo4j. *Сучасний стан наукових досліджень та технологій в промисловості*. 2022. № 2 (20). С. 52–63. DOI: https://doi.org/10.30837/ITSSI.2022.20.052

Mazurova, O., Syvolovskyi, I., Syvolovska, O. (2022), "NoSQL database logic design methods for MongoDB and Neo4j", *Innovative Technologies and Scientific Solutions for Industries*, No. 2 (20), P. 52–63. DOI: https://doi.org/10.30837/ITSSI.2022.20.052