Mechanics Transport Communications
Academic journal http://www.mtc-aj.com

ISSN 1312-3823 (print)
ISSN 2367-6620 (online)
volume 21, issue 2, 2023
article № 2363

RESEARCH OF MODERN NOSQL DATABASES TO SIMPLIFY THE PROCESS OF THEIR DESIGN

Volodymyr Lysechko[1]
volodymyr.lysechko@gmail.com
Illia Syvolovskyi[1]
ilyasvl95@gmail.com
Bohdan Shevchenko[1]
shevchenkobn@gmail.com
Alla Nikitska[2]
nikaalla91@gmail.com
Galina Cherneva[3]
cherneva@vtu.bg

[1]Ukrainian State University of Railway Transport, Kharkiv, Feuerbach Square 7, UKRAINE
[2]Military serviceman, military base A5617, UKRAINE
[3]Todor Kableshkov University of Transport, Sofia, Geo Milev Str. 158, BULGARIA

Keywords: *database, experimental research, DBMS, NoSQL, performance testing, MongoDB, Neo4j.*

Abstract: *The amount of data on the Internet is growing at a tremendous rate, as users add thousands of gigabytes of data to social networks every second. It is not surprising that relational databases cannot cope with such modern arrays of information, even though they have been successfully dealing with data processing tasks for several decades. This problem has led to the need to introduce new approaches of data storing and processing in large systems. NoSQL databases coped with this task, as they allowed replacing expensive vertical scaling with efficient horizontal scaling. They also had better performance, a more flexible data model, and open-source code.*

However, without unified approaches to database selection and schema implementation, application developers make mistakes at the system design stage, which can lead to additional costs and problems.

Given the urgency of the problem, the paper explores modern NoSQL databases, which may be the key to improving the overall performance of server systems.

INTRODUCTION

The world is currently experiencing an "information explosion". In the previous five years, humanity has produced more information than in the previous history. Relational databases, which have been used for more than 40 years in projects of various profiles, can no longer cope with such data flows. This is also reinforced by the fact that they do not effectively support the important functionality for big data today: sharding and replication.

Therefore, in the current situation, non-relational NoSQL databases are increasingly being used to effectively store unstructured and poorly structured data.

The rapid and widespread use of NoSQL databases is due to the ease of their development at any scale, the functionality and performance of these databases. NoSQL databases are conveniently used for many modern applications that aim to use scalable databases that have high performance, wide functionality, and the ability to provide maximum usability. These are mobile, gaming, Internet applications, etc.

THE MAIN PART

The research found that the main advantages of NoSQL databases are:

- lack of a clear scheme and flexibility, which leads to faster development and provides opportunities for phased implementation of projects;
- scalability: NoSQL databases do not need to be installed on expensive, reliable servers, as they are designed to scale using distributed clusters. Also, all cloud service providers implement such operations in the background to provide a fully managed service;
- high performance: unlike relational databases, NoSQL databases allow for higher performance because they are optimized for specific access patterns and data models;
- wide functionality: NoSQL databases provide APIs that have wide functionality and are specially designed for the respective data models. Depending on the data model, NoSQL databases are divided into several types.

The main types of NoSQL databases include the following:

- key/value databases: they have the ability to store data in any format by a specific key. They provide high distributivity and support unprecedented horizontal scaling, which is unattainable with other types of databases. These databases are most often found in projects with a high read load. That's why they require caching of information blocks. Examples of such databases are: Redis, Amazon DynamoDB;
- are document-oriented; they are databases that allow developers to store and query data in the database 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 characterized by flexibility and hierarchy, which allows them to evolve in response to the growing needs of applications. MongoDB, CouchDB, and Couchbase are all examples of the most common document databases. They aim to provide functional and intuitive APIs for agile development;
- graph databases are databases in which the data representation is realized in the form of nodes and edges, which are relations between nodes. Such databases provide easy processing of complex data and calculation of specific properties of graphs, such as the path from one node to another and its length. Typical examples of graph databases include social networks and services;
- columnar (column-oriented) databases; these are databases where data is stored in the form of cells that are grouped not in rows of data, as in relational databases, but in columns. These columns are logically grouped into families that have an almost unlimited number of columns. Columns, not rows, are used for reading and writing. Such databases are usually used to store and process analytical information. Google BigTable is an example of the first columnar DBMS. Nowadays, modern columnar databases include: Cassandra, HBase, and ClickHouse.

At the moment, the most popular NoSQL databases are document databases, in particular MongoDB, which is rapidly catching up with popular relational databases: Microsoft SQL Server, Oracle, MySQL, and PostgreSQL [1].

When designing databases, it is necessary to create a database schema and determine the necessary integrity constraints. The main tasks of database design include the following:

a) ensuring that all necessary information is stored in the database;
b) reducing duplication and redundancy of data;
c) ensuring the possibility of obtaining information on all necessary requests;
d) ensuring data integrity, i.e., eliminating data loss and contradictions.

Database design includes four main stages.

Conceptual design is the process of creating a conceptual (semantic) model of a subject area that does not rely on a DBMS or an existing data model.

The conceptual model should contain:

- description of the information objects of the system and the relationships that arise between them;
- description of integrity constraints, i.e., requirements for acceptable data values and relationships that arise between them, and so on.

To implement this stage of database design, a textual description and free-form notations similar to the UML object diagram are usually used. If the subject area model is not very complex or large, this stage can be skipped.

The next stage of database design is infological design. This is the process of creating an ER model of the subject area. The model can be either in the form of a diagram or in text form. The model includes: entities, their attributes and keys (primary), relationships between entities and their cardinality.

The artifact of this stage is an ER diagram in Chen's [2], Barker's, Crow's Foot [3], or IDEF1X notation.

Logical or datalogical design is the creation of a specific logical data model, namely, a relational, document, or graph model, but without taking into account the specifics of a particular DBMS. Often the purpose of this stage is to create a database schema. As for the relational database model, it is a set of relationships with the primary keys and their relationships, the graph model is a set of nodes, links and their attributes, and so on.

Physical design is the process of creating a physical database model for a particular DBMS. That is, at this stage of design, you specify the type of data to be supported, create indexes, manage the physical data storage space, and so on. The result of this design stage is a diagram and/or a database creation script.

If a database model is relational, it must be normalized according to normal forms [4]. But, in the context of non-relational data models, such as document or graph models, the opposite transformation is used in practice – denormalization [5].

Denormalization is the process of modifying a database, in which the order of its normalization is reduced. The need to perform this process is motivated by the need to improve database performance.

In real-world applications, the number of read operations is usually much higher than the number of write operations. Also, the entity data for the required operations are often joined to reduce the number of queries using the JOIN operator in relational DBMSs [6] or its equivalent in NoSQL databases.

The denormalization process is performed using the following set of rules:

- combining entities with 1:1 relationship;
- duplication of non-key attributes in entities to reduce the number of relationships;
- creation of aggregate entities containing data from other entities.

When designing a database for any NoSQL system, the developer is required to have a clear understanding of the database operation and the tools offered by the DBMS. Since this understanding may not be present in practice, many commercial projects are hesitant to switch to new NoSQL databases because the implementation of such a transition requires a lot of time for performance modeling and information migration.

Also, quite often, developers do not know how to model the schema (entities, relationships) more efficiently for a particular data model, which data indexes work more efficiently, etc.

Thus, the study of methods and approaches to logical modeling of NoSQL databases is relevant today. For further research, the target DBMSs need to be selected.

As for the popular documentary DBMSs that can be used for server applications, they include the following: MongoDB, DynamoDB, CosmosDB (in document database mode), Couchbase.

When choosing the optimal database for research, it is advisable to use the following criteria:

- popularity (or prevalence) of the DBMS - the more popular the DBMS, the more information about it can be found and the greater the chance that it will be supported for a longer time. To determine the popularity of a DBMS, we suggest using the db-engines.com resource;
- license - whether the DBMS is an open-source project or has a commercial license;
- type of DBMS deployment - whether the DBMS is a cloud-only solution or has the option of deployment in the cloud;
- availability of drivers for different programming languages.

The lack of binding to a particular cloud service is also a great advantage, as it makes the system architecture more flexible and allows you to build your infrastructure on a separate server (or set of servers) for any project needs.

From the license point of view, open-source databases have an advantage over commercial ones due to the openness of the source code. The availability of an optional enterprise version of the DBMS with additional data protection and backup functionality is a significant advantage in large projects.

The availability of drivers for many programming languages gives more flexibility in choosing technologies when building the server side.

Based on these criteria, we can compile a table of the most widespread and relevant document DBMSs at the moment (Table 1).

Table 1. The most widely used document DBMSs

	Popularity on db-engines	**License**	**Deployment type**	**Drivers**
MongoDB	435.49	Open Source + Enterprise	Standalone + Cloud-based	5 / 5
DynamoDB	78.81	Commercial	Cloud-based only	2 / 5
CosmosDB	36.49	Commercial	Cloud-based only	5 / 5
Couchbase	25.14	Open Source + Enterprise	Standalone + Cloud-based	3 / 5

It should be noted that, at the moment, all the presented DBMSs support such important concepts as:

- support for data sharding;
- support for replication (source-replica, multi-source);
- support for ACID transactions.

When it comes to graph databases, the number of possible DBMS options is quite low. At the moment, there are the following graph databases: Neo4j, CosmosDB (in graph database mode), Amazon Neptune, ArangoDB, OrientDB, TigerGraph.

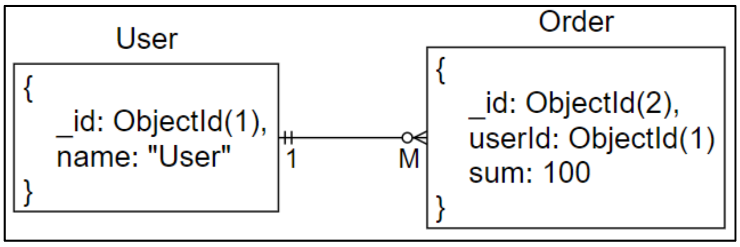
However, it should be noted that only Neo4j, Amazon Neptune and TigerGraph support a purely graph model, while the rest of the DBMSs are multi-model. Supporting many types of data models does not guarantee the required performance and functionality of the DBMS.

Also, Amazon Neptune is a proprietary database that can only be deployed in the cloud, which can be an obstacle for many developers. Many of the DBMSs on the list, such as ArangoDB, OrientDB, and TigerGraph, are still quite new and are still being developed, making them a risky choice for production solutions. In such a situation, only Neo4j is an acceptable choice if the server application needs to work quickly with data that can be represented as a graph.

Now that the databases to investigate have been identified, we should move back to the database design process. Before the logical design of the database, the infological design takes place, i.e., the construction of the logical model is based on the ER-diagram. Algorithms for transition from ER diagrams to logical models in the context of relational databases have long been formalized. However, these algorithms are not applicable to NoSQL databases, which are based on data structures other than tables (relationships). But even so, the ways of modeling the entities and relationships of relational databases can be applied to NoSQL databases, albeit with amendments and additions [7].

Traditionally, document databases do not provide for relationships in the relational sense, so the implementation of such functionality and data integrity lies entirely with the developers of the server (or client, if it is a local database) part that uses the database.

To create functionality similar to relationships, the «Manual reference» method is used, which involves saving the "_id" field of one document in the field of another object, similar to the foreign key in relational databases, but without the relationship itself (Fig. 1) [8].



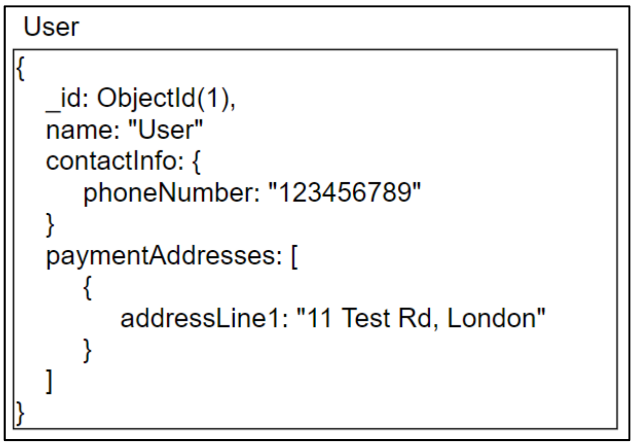
```
{
    _id: ObjectId(1),
    name: "User"
}
```

```
{
    _id: ObjectId(2),
    userId: ObjectId(1)
    sum: 100
}
```

Fig. 1. The method of «Manual reference»

Using this method, a 0:M relationship is formed, which can already be used by developers as 1:1, 1:M, and derivative relationships. Thus, using an additional query, you can get another document to which the current one refers. However, this leads to the «N+1» problem, because it requires an additional query or joining data through a JOIN-like operation.

To solve such problems, document databases offer the use of compositions in the form of nested objects or arrays of objects (Fig. 2).



```
{
    _id: ObjectId(1),
    name: "User"
    contactInfo: {
        phoneNumber: "123456789"
    }
    paymentAddresses: [
        {
            addressLine1: "11 Test Rd, London"
        }
    ]
}
```

Fig. 2. The method of object composition

This approach works well if the relationship between objects can be expressed with the word «includes». This method can be used to model relationships:

- 1:1, but you need to keep in mind that a nested object will increase the weight of the document, which slows down its upload from the database to the client;
- 1:M, but if M is not a particularly large number and the nested objects are not very large.

But, with this approach, you need to remember that:

- the maximum nesting size is 100 levels;
- the maximum size of a document is 16 MB;
- if new records are constantly being added to the document field (array), the document size will constantly grow. This can cause performance problems because the document is moved to another memory location because it no longer has room to grow in the current location (defragmentation).

No relationships also mean no JOIN in the relational sense. But, later, MongoDB added two ways to join data, namely:

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

However, it should be noted that these methods have potentially low performance because they perform additional queries for the required data. Therefore, in the context of document databases, composition is a higher priority. But, very often, you cannot do without this functionality, so you need to create the right index to replace the «COLLSCAN» (full search) stage with «IXSCAN» (index search). Thus, if we draw an analogy with relational databases, we turn the slowest type of JOIN (LOOKUP) into a fast MERGE JOIN.

Also, MongoDB has added functionality that has an external resemblance to relationships - DBRef. However, this functionality is not a relationship, it is an auxiliary function for client drivers that indicates that a 1:M relationship is possible in this field, so it is possible to make an additional request to retrieve entities. However, this functionality has a rather low performance, is not supported by all drivers, and is intended more for "linking" entities from different collections in a nested array, since the link contains the name of the collection and, optionally, the name of the database.

The situation with entities that have an M:M relationship is more complicated. It is known that an M:M relationship can be expressed as two 1:M relationships and an intermediate object that contains the identifiers of the two referenced objects. This approach can be implemented in MongoDB without any problems, except for creating indexes on fields with identifiers. With this approach, all auxiliary attributes of the M:M relationship will be located in a separate object.

But if the developer needs to join the data of such a relationship, he will have to use two JOIN-like operations ($lookup), which is very expensive in the context of document databases.

To solve such a problem, MongoDB almost always uses a different approach: composing this intermediate object into one of the M:M relationship objects as an array. Figure 3 shows both approaches: through an auxiliary entity (top) and a shortened approach with composition (bottom).

The widespread use of the second approach is due to the fact that in 2017, the $lookup operation received support for using arrays of identifiers as input data for joining data. With this approach, only one JOIN operation is needed to join data instead of two. But, when using it, it is necessary to select the "main" object from the M:M relationship, which will contain an array of identifiers.

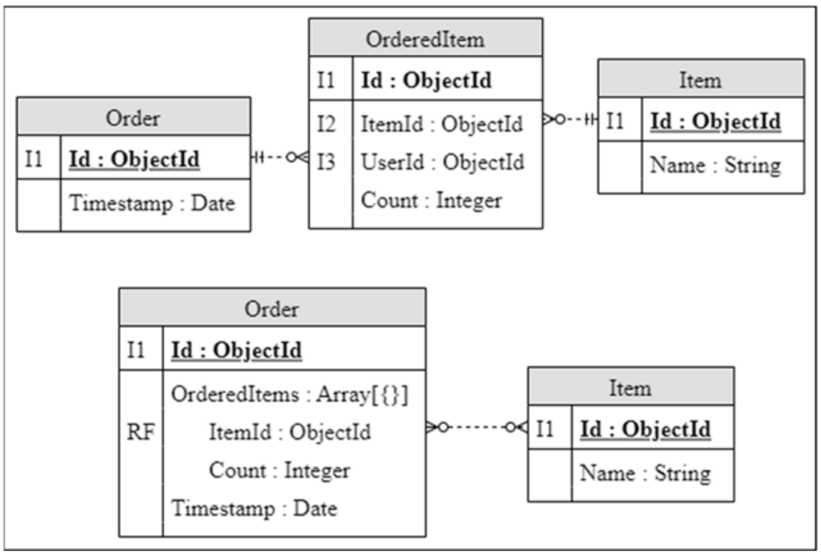


Fig. 3. Methods of designing M:M communication in document database

In fact, the «Manual reference» method makes the schema more similar to relational databases, so it can be labeled as a «normalizing» method. The Nested Document method, on the other hand, reduces the level of normalization through composition, so it can be called «denormalizing». The aforementioned M:M relationship design method is a «mixed» method, leaning towards «normalizing» because it uses both concepts but still depends on the JOIN-like operation.

The next object of research is the methods of logical design for graph DBMSs, in particular Neo4j. In this DBMS, the database is one big graph without dividing entities by type (like tables or collections). This graph consists of: vertices (entities) and edges (links). Unlike document databases, graph databases support relationships, although they differ significantly from relational relationships.

In Neo4j, each relationship is a special type of entity that stores references to the source and target entities. Thus, relationships have their own names, can contain attributes, and can be indexed.

Like all NoSQL databases, this DBMS has no integrity restriction mechanisms, this is up to the developer to decide at the application level. However, each relationship in the graph must have an output and an input entity.

It follows that each Neo4j link has a 1:M cardinality by default, which can be "converted" to a 1:1 link through uniqueness restrictions or at the program level.

Thus, the M:M relationship can potentially be modeled in two ways: through an auxiliary entity (as in relational databases) and directly, storing additional data as attributes of the relationship. Figure 4 illustrates these modeling methods graphically: how it looks like in relational databases (top), through a helper entity (middle), and through relationship attributes (bottom).

It should be noted that almost all graph DBMSs have only unidirectional relationships. The standardized query language Gremlin, which is supported by all graph DBMSs, also does not support bidirectional relationships. Thus, to create a bidirectional relationship, you need to make two connections in both directions.

Taking into account that a relationship is also one of the DBMS objects, the option with an intermediate entity is inefficient from the very beginning, because it heavily clogs the database with unnecessary entities and relationships, increases the weight of the database due to unnecessary objects, and potentially increases the execution time of even basic queries.

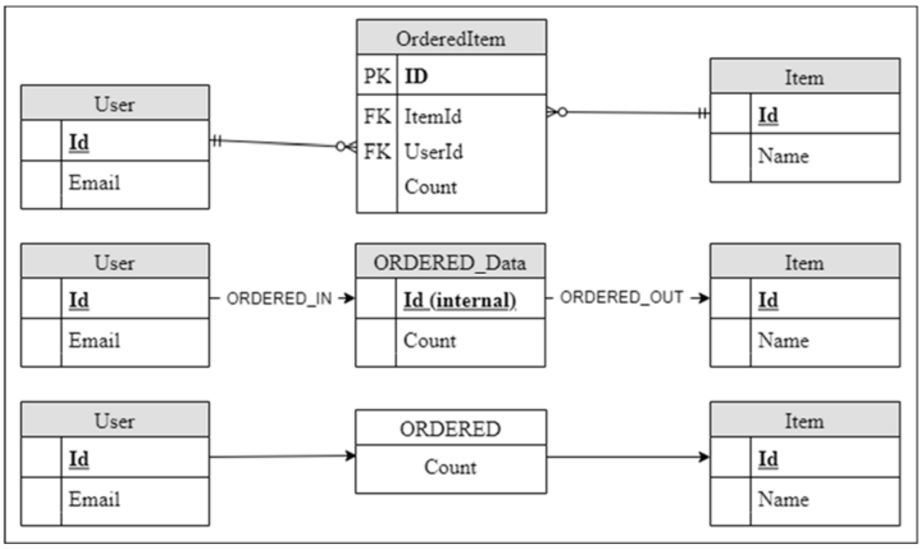


Fig. 4. Comparison of methods for designing M:M relationships in graph DBMSs

CONCLUSIONS

As a result of the analysis of modern NoSQL databases, it can be noted that they provide high efficiency of application to increase the performance of server systems.

As a result of analyzing the field of modern relational and NoSQL databases, it was found that NoSQL databases are finding more and more widespread every day and in various tasks. This gives a reason to conduct a deeper study in the field of NoSQL database design. The most relevant types of NoSQL databases were investigated and the most common NoSQL DBMSs at the moment were selected. Having analyzed the approaches of modelling relationships in document and graph databases, it was found that NoSQL databases can also effectively cope with this task, even with more flexibility. The results obtained are a solid background for further research.

LITERATURE

[1]. Graph Database. What's the Big Deal? Towards Data Science. - URL: https://towardsdatascience.com/graph-databases-whats-the-big-deal-ec310b1bc0ed.
[2]. Halpin T. Entity Relationship modeling from an ORM perspective: Part 1. Object Role Modeling. - URL: http://www.orm.net/pdf/JCM11.pdf.
[3]. Data, K. Introduction to database systems. / K. Data. - Moscow: Nauka, 1980. - 464 c.
[4]. Teorey T., Lightstone S., Nadeau T. Database Modeling and Design. – Elsevier, 2006. – 296 P. – ISBN 978-0-12-685352-0.
[5]. Wikipedia: Denormalization. - URL: https://en.wikipedia.org/wiki/Denormalization
[6]. Sanders G. L., Shin S. K. Denormalization effects on performance of RDBMS/ Proceedings of the 34th Annual Hawaii International Conference on System Sciences. 2001.9 P.
[7]. Meier A., Kaufmann M. SQL & NoSQL Databases: Models, Languages, Consistency Options and Architectures for Big Data Management. – Springer Vieweg, 2019. – 248 P. – ISBN 978-3658245481.
[8]. MongoDB Manual. Data Model Design. - URL: https://www.mongodb.com/docs/manual/core/data-model-design/#embedded-data-models.