Efficiency of Data Caching when Building Heterogeneous Databases

Igor Grebennik[1,†], Oksana Mazurova[1,*,†], Oleksandr Samantsov[1,†] and Denys Semko[1,†]

[1] *Kharkiv National University of Radio Electronics, Nauky Ave., 14, Kharkiv, 61166, Ukraine*

Abstract
An extended infological (ER) model is proposed, which takes into account deeper knowledge of data flows in the subject area due to the added maps of data processing operations. This model is the basis of the method of designing heterogeneous databases due to the possibility of better breaking down a monolithic data model into separate logical structures and selecting a heterogeneous set of appropriate logical database models. In the example of designing a heterogeneous database in the field of e-commerce, the application of the model at the main stages of designing such a database is shown. The issues of choosing data organization logics for parts of a heterogeneous database and database management systems (DBMS) for their physical implementation are considered, taking into account the possibilities of data caching in the corresponding DBMS. The results of the study of data caching mechanisms in the NoSQL systems MongoDB and Redis, as well as the relational system MSSQL, are given. Recommendations for the use of researched database management systems for database development, including for distributed heterogeneous databases, are formulated.

Keywords
Heterogeneous database, caching, design method, DBMS, ER-model, NoSQL, MongoDB, MSSQL, operations execution map, Redis

1. Introduction

As we're present in the time of active usage of various Internet applications, including e-commerce systems, the issue of scaling such systems is important. The use of distributed heterogeneous databases (HDB) [1, 2] is becoming increasingly common in the direction of creating highly scalable systems. Such databases use the advantages of different logical database models and different database management systems (DBMS) for data storage and processing.

The existing approaches to the modeling of distributed HDBs are mainly presented in the form of general recommendations for the division of a database monolith when designing systems for a microservice architecture [3], "best practices" of using NoSQL [4] and other DBMSs to solve specific practical problems, or methods of integrating existing DB into a heterogeneous system [2, 5]. It does not allow solving the problem of designing HDBs "from scratch" and in the same methodological way as during the process of designing relational databases, which has long become convenient for developers. The successful implementation of the classical approach to the design of relational databases [6, 7] requires the expansion of existing data models, which would allow for a continuous process of designing the logic of the HDBs and would provide reasonable recommendations for the selection of appropriate DBMSs for physical implementation of parts of HDB.

Information Technology and Implementation (IT&I-2025), November 20-21, 2025, Kyiv, Ukraine
[*]Corresponding author.
[†] These authors contributed equally.
igor.grebennik@nure.ua (I. Grebennik); oksana.mazurova@nure.ua (O. Mazurova); samantsov.aa@gmail.com (O. Samantsov); dench327@gmail.com (D. Semko)
0000-0003-3716-9638 (I. Grebennik); 0000-0003-3715-3476 (O. Mazurova); 0000-0002-4788-4144 (O. Samantsov); 0009-0006-9519-3248 (D. Semko)

Choosing a DBMS for physical implementation, including parts of the DBMS, requires the developer to clearly understand not only the logic of its operation, but also additional capabilities and tools offered by the DBMS [8-10]. One such tool, which quite often leads to heterogeneous solutions in the database, is the caching mechanism that some DBMSs offer to efficiently reuse previously retrieved data, thereby reducing the time spent accessing the same data. Many existing DBMSs implement their own caching mechanism and provide their own built-in logic that is quite unique. So, when choosing a DBMS, there is a question as which of them provides work with a large volume of data using its own mechanics, including caching, better than others.

Therefore, to solve the identified problems that have arisen before the developers of GBD, the task of choosing adequate logical models and corresponding effective DBMS for the implementation of individual parts of GBD is urgent. Such task requires not only the development of a general information technology for DBMS design, but also an experimental study of the performance of special mechanisms, such as caching, in modern DBMSs.

2. Review of the literature

Modern information systems require the use of high-performance databases and scalable solutions. That is why developers use HDBs more often [2, 7]. At the same time, the use of distributed HDBs covers more and more different industries, among which, along with traditional ones (e.g., e-commerce [11, 12]), new ones are emerging - social networks, space engineering, rapid response in emergency situations, etc. Such databases allow us to make maximum use of the features of new logics of database organization and new DBMSs, especially in the direction of NoSQL systems, which have appeared quite intensively in recent years [8]. Thanks to heterogeneous solutions, modern software applications cope with those problems that relational databases are no longer able to effectively solve.

In the theory of databases, an end-to-end approach to their design through conceptual, info-logical (or ER-) modeling to logical, and then physical modeling has been developed a long time ago [6, 7, 13]. For relational databases, all transitions from one model to another have been formalized a long time ago. Unfortunately, for NoSQL systems that do not have such a thorough mathematical basis as relational databases, there are basically no methods for their logical design [14, 15]. For example, the design of connections, which in NoSQL models traditionally do not coincide with the relational approach, and the implementation of the data integrity mechanism are completely up to the developers of the respective databases.

It should be noted that with the advent of non-relational models, including NoSQL models, the importance of the ER model in the design process has increased. The ER-model continues to be the basis for further modeling of database logic, as it is as general as possible and does not contain details inherent, for example, to relational databases or other logics [2, 7, 13]. The ER model can be considered as an excellent informational framework that does not contain the description of foreign keys, which in most NoSQL models are not used to model relations. However, the capabilities of ER-diagrams for modeling entities ((E)entity) and relations ((R)relationship) are still not sufficient to design a heterogeneous database on its basis, for which an important element is considering the peculiarities of data processing.

Currently, approaches to modeling distributed HDBs are mainly developing in the direction of integration of existing databases into a heterogeneous system [1, 2, 5], synthesis of new databases during their reengineering [16], or are presented in the form of general recommendations for the design of microservice architectures [3] with a corresponding cut-off from more often, a homogenous monolith of the database is divided into individual NoSQL systems that are popular today. Database developers are forced to first study the tools of new NoSQL models and then, relying more often on their "relational" experience, physically implement certain database options and evaluate the quality of the resulting solutions [9, 15]. A single methodology for logical modeling of HDBs should take into account the variety of data models existing today and get rid of the existing separation of business logic design (namely, database transactions) from the development of a logical database model.

NoSQL systems, which provide high performance and wide functionality [14], are actively used for many modern applications that aim to use scalable databases, for example, for mobile and game applications, applications to support e-commerce, etc. E-commerce systems today are flexible, scalable and highly loaded systems due to the presence of a large volume of data and a complex structure [11]. E-commerce systems must respond to a request with maximum speed and accuracy, as the load on the system can reach millions of requests per second. Caching minimizes such speed needs through a wide variety of mechanisms and strategies.

NoSQL DBMSs of the "key-value" type are quite widespread in use when it comes to data caching [17]. A "key-value" is a data structure that consists of a unique key to identify the data and a value that acts as system data. Among key-value DBMSs, Redis is one of the most popular databases due to its high performance, easy scalability, and high reliability. Redis is designed for fast read and write operations, using a variety of data structures such as hash tables, sets, strings, and others that store data at the level of RAM. In addition, Redis supports a persistence feature that allows data to be saved to disk and restored in the event of a system crash or server shutdown. Due to the ability to quickly process data, effectively control its preservation, and ensure high system performance, this DBMS is most often chosen when data caching is needed. Among DBMSs with caching mechanisms, Redis stands out with the ability to automatically delete unnecessary data from the cache after a certain period, check which cache elements are the least used, and provide support for client-side caching.

Another quite popular type of NoSQL system is document-oriented DBMS. Each saved record looks like a separate document with its own set of fields. Documents are characterized by flexibility and hierarchy, which allows them to develop in accordance with the growing needs of applications. Among document-oriented DBMS, MongoDB [14, 18] is not only one of the most popular, but also provides a functional and intuitive API for flexible development and is attractive to developers due to the availability of drivers for various programming languages. In addition, the widespread use of MongoDB has led to the appearance of many studies devoted, including approaches to logical design for this NoSQL system [15]. MongoDB has its own WiredTiger caching engine that uses a caching technique to store frequently used data in RAM. This approach significantly improves data reading performance and reduces data access time. Using compression and checkpointing functionality, WiredTiger minimizes memory usage, thereby allowing more data to be loaded into memory for caching [14].

We should not forget relational DBMSs, while researching data caching mechanisms, which even in heterogeneous solutions are the basis for storing and processing a large part of data. The model of storing data in the form of structured tables provides efficient caching strategies because data is easily identified based on tables and indexes. Relational databases are characterized not only by structured data but also by the use of indexing, support for ACID transactions [19], and concurrency control. Under such conditions, caching can be implemented thanks to various strategies and mechanisms. Among the relational databases, the most popular one for use and interaction is MS SQL Server, which provides a wide range of possibilities for analyzing each query. This DBMS divides the caching capabilities into two separate mechanisms - a buffer cache and a procedure cache. A buffer cache deals with caching pages of data in memory to reduce I/O and improve performance by keeping frequently accessed data in memory. The procedure cache is a place where query execution plans are stored, which contain information about exactly how MS SQL Server will execute the query and what logical steps are involved in the efficient execution of the query.

Meanwhile, the available documentation on NoSQL DBMS and existing recommendations do not provide developers of heterogeneous DBs with an end-to-end methodology for their design and selection of appropriate DBMSs, which would clearly indicate the effectiveness of the resulting DB. Solving such questions requires not only a deep understanding of the capabilities of such DBMSs but also experimental studies of additional mechanisms available in them. Thus, the study of the effectiveness of caching mechanisms in relational and NoSQL systems is relevant.

3. Problem statement

The purpose of this article is to study the effectiveness of data caching in NoSQL and relational DBMS with the aim of developing qualitative recommendations regarding the selection of DBMS for the implementation of appropriate business logic when solving the problems of building heterogeneous databases.

To achieve the goal, the following tasks must be solved:

- To develop an extended infological (ER) model as a basis for designing HDB.
- On the basis of the proposed model, design HDB in the field of electronic commerce.
- To plan an experimental study, which includes the development of performance evaluation criteria, the possibility of configuring data in a certain format at the same time for all DBMSs, and the development of stress situations for verification.
- Conduct an analysis and implement caching mechanisms in the selected for the heterogeneous database DBMSs.
- Conduct an experimental study and formulate recommendations for selecting DBMSs that effectively use the caching mechanism when building HDBs.

4. Models and methods

4.1. Development of an extended conceptual model for the design of a distributed HDB

Existing studies and practices of designing HDBs have shown that when dividing a monolithic database into local heterogeneous parts (DBs), it is important to consider the business logic of the system, namely, data processing operations [3], in the execution of which those or other database entities. The main part of such operations are queries and transactions, which are a sequence of CRUD operations in the database (according to the standard classification of data manipulation functions introduced by James Martin in 1983). With the advent of NoSQL systems based on the Eventually consistent BASE architecture [4, 9], the business logic of some database operations sometimes adds data consistency stages involving humans or other external factors. An example of such an external operation is the reconciliation of orders by an operator in e-commerce systems. Therefore, modeling business logic for HDB also requires considering such external operations (we will call them E-operations).

Query and transaction modeling is not included in the design methodology of a classic relational database, but the use of different semantic models to represent data processing operations is noted by many researchers [3, 10, 20] and is important when designing a heterogeneous database. One of the interesting models for representing transactions is the transaction execution map, which, based on DB entities, displays the sequence of CRUD operations that make up the essence of the transaction, as well as the capacity of the corresponding entities. Such a model can be quite easily integrated with an ER diagram, which is an important component of the database design process. And if we extend such maps to E-operations modeling (in this case, we will call them operation execution maps) and add the requirements for distributed data processing, which are important for distributed databases, defined by the CAP theorem [10], and requirements for the performance of their execution, this will allow to use such a model as an effective basis for designing distributed HDBs.

The paper proposes an extended ER-model (EER), which can be presented in the form of a tuple

$$EER = < \mathrm{E}, \mathrm{R}, \mathrm{O} >, \tag{1}$$

where $E = \{E_i | i = \overline{1, n}\}$ – set of entities of the subject area; R – set of relations (interconnections) between entities; $O = \{O_k | k = \overline{1, p}\}$ – set of data processing operations.

Each entity E_i can be represented as a set of corresponding attributes A_{ij} entity - $E_i = \{A_{ij} | j = \overline{1,m}\}$. Set of relations (interconnections) R between entities $E_i \in E$ represents as a matrix $R = \|R_{ij}\|$ $(i,j = \overline{1,n})$, the elements of which determine the type of relationship directed from the entity E_i to the entity E_j.

An important component of the EER model, which extends the traditional ER model, is the set of data processing operations. Every operation O_k has certain characteristics that set requirements for the nature of its processing and describe the essence of this operation in the database. So, every operation O_k can be represented as a tuple

$$O_k =< OM_k, \mathrm{CAP}_k, Fr_k, rT_k >, \tag{2}$$

where a vector-string $OM_k = \|om_{ki}\|$ specifies the execution map with k-th operation; CAP_k sets requirements for transaction processing in accordance with CAP theory; Fr_k - estimated frequency of the operation; rT_k - requirements for the operation time (в ms).

Vector string elements OM_k are indexed along the axes themselves with k-th operation and a set of entities where the elements $om_{ki} = \{CRUDE_{kij} | j = \overline{1,q}\}$ is a sequence of CRUD or E-operations executed on the corresponding entity E_i.

The description of the CRUDE operation can be represented by a tuple

$$CRUDE_{kij} =< poz_{kij}, CRUDE\_Name_{kij}, ex\_max_{kij} > , \tag{3}$$

where poz_{kij} - the order in which the j-th operator is executed within the k-th operation; $CRUDE\_Name_{kij} \in \{C, R, U, D, E\}$ - letter indicating the type of CRUD or E-operation; ex_max_{kij} - the maximum number of instances of the i-th entity that will be processed by the operator.

The requirements for processing an operation according to the CAP theorem can be given by a vector

$$\mathrm{CA}P_k = \|p_{kj}\|, j = \overline{1,3}. \tag{4}$$

Vector's elements p_{kj} will take the value 1, if in the requirements for the processing of the k-th operation there is, respectively, the requirement "(C)onsistency" for the component p_{k1}, requirement "(A)valability" for the component p_{k2}, "(P)artitioning" requirement for a constituent p_{k3}. In the absence of a corresponding requirement, $p_{kj} = 0$.

The proposed EER model (1)-(4) can be used as a basis for the heterogeneous database design process, which should include the following stages:

1. Selection and preliminary preparation of input data: at this stage, the analysis is carried out for the selected subject area, as well as the stages of conceptual and informational database modeling, which correspond to the traditional database design methodology; namely, UML and other diagrams are constructed that provide the results of the analysis to the domain (for example, a generic class diagram for preliminary modeling of the relationships between the main entities of the domain, a data flow diagram for a general representation of business processes, etc.), and for a more formalized representation of entities and the relationships between them, an ER diagram is developed [6, 7, 15] (for the convenience of the next stages, it is better to use Barker's notation or its modification Crow's Foot).
2. Construction of the EER-diagram: at this stage, models of data processing operations inherent in the defined business process are added to the ER-diagram in the form of operation performance maps (2).
3. EER decomposition: at this stage, the EER is decomposed into a system of interconnected local EER-models based on the analysis of a set of operation execution maps.
4. Determination of types of logical models for further design of local EERs: at this stage, types of logical DB models are determined (in the case of using NoSQL models, specific

recommendations for NoSQL DBMSs are possible) for selected local EER models, taking into account the peculiarities of logical models.

5. Logical design of a heterogeneous database: at this stage, the construction of local logical models is carried out, considering the peculiarities of data organization and limitations of the integrity of the selected data models or DBMS.

The proposed model (1)-(4) makes it possible to form a matrix on the basis of which the best options for dividing the database into separate fragments can be algorithmically found. The algorithm includes some aspects of subjectivity and requires presentation in a separate work.

4.2. Designing a heterogeneous database in the field of e-commerce

The applied field of e-commerce was chosen for the research of caching mechanisms. Let us consider some points regarding using the EER model and the proposed approach to designing HDB in this area. So, the analysis and conceptual modeling of the e-commerce industry was carried out. Let us consider a fragment of the business process consisting of searching for goods, working with the shopping cart, and creating an order.

The basic ER diagram was developed using "Crow's foot" notation. Based on the basic ER diagram, an extended EER model (Fig. 1) was developed, which simulates maps of the execution of operations. Figure 1 shows the map of the execution of the operation O_1 (the transaction of adding the product to the cart).

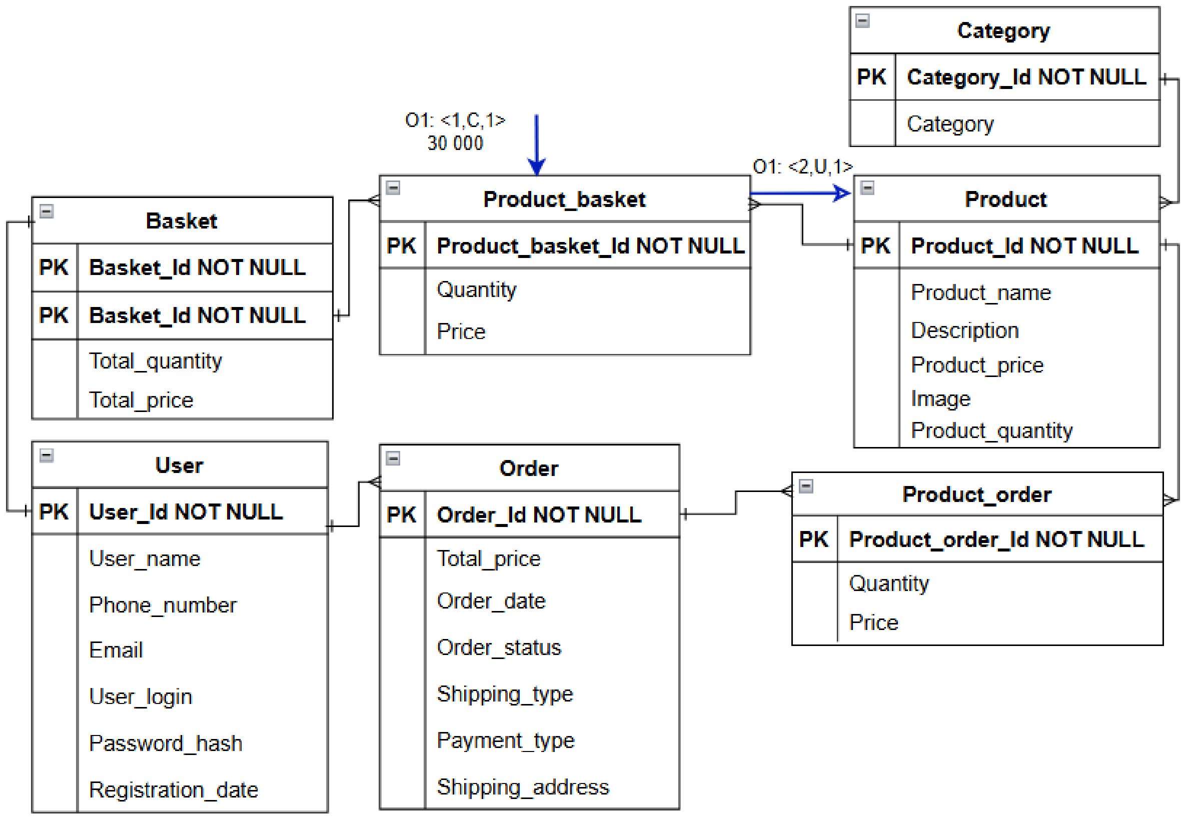

Figure 1: A fragment of the EER diagram

Figure 2 separately, for clarity, shows a map of the execution of several operations that are part of the developed EER model for the task of working with a shopping cart, which simulates:

- Data processing flows in the form of consecutive CRUD operators (3) over the relevant entities.
- The intensity of operations in units of transactions per day (the value is shown near the entry point of the operation).
- The minimum, average and maximum strength of relationships between entities (shown next to relationships).
- The average power of entities (values are given under the relevant entities).

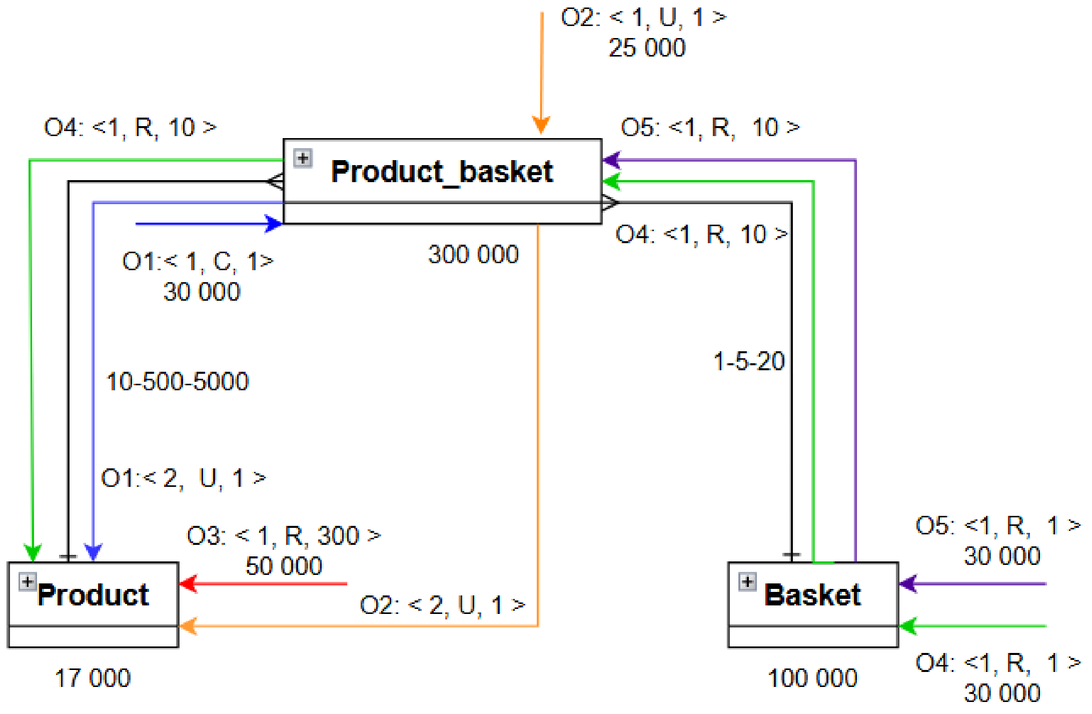

Figure 2: A fragment of the operation execution map for the business process of working with the shopping cart

The map contains next operations:

- O_1 – the transaction of adding the product to the cart.
- O_2 – the transaction of editing the quantity of goods in the cart.
- O_3 – request to filter products by brand and price.
- O_4 – request to receive a list of products from the basket by the unique Basket identifier.
- O_5 – request to calculate the total amount for the products selected for the basket by the unique Basket identifier.

A transaction O_2 starts with editing (operation U – UPDATE) product's quantity in the basket (Product_basket entity is used) and ends with the appropriate editing of the quantity of the available product in the entity Product. The estimated intensity of execution of such a transaction is 25,000 operations per day. Similarly, other operations related to viewing and searching for products on the website of the online store were modeled, which include operations for filtering products, counting the total number of selected products, analyzing completed orders, etc.

At the next stage of the proposed approach to HBD design, the expanded info model was decomposed into local EER models. Based on the analysis of maps of the execution of operations and the requirements for their processing, the following sets of entities were selected, which require further logical design based on a common logic of working with them (Fig. 3):

- Product and Category entities (local EER Catalog) involved in joint operations to search for goods that are critical from the point of view of their intensity (more than 50,000 executions per day) and Consistency and Partition Tolerance requirements regarding their processing.
- Entities User, Order and Product_order (local EER Ordering), involved in joint operations to form an order and transfer goods from the basket to the order, which are not so critical in terms of the intensity of execution (up to 5000 transactions per day), but have enhanced requirements regarding Availability and Consistency of data.
- Entities Basket and Product_Basket (local EER Basket) involved in joint basket operations, which have similar performance intensity characteristics (30,000 transactions per day) and the same Partition Tolerance and Consistency requirements for their processing.

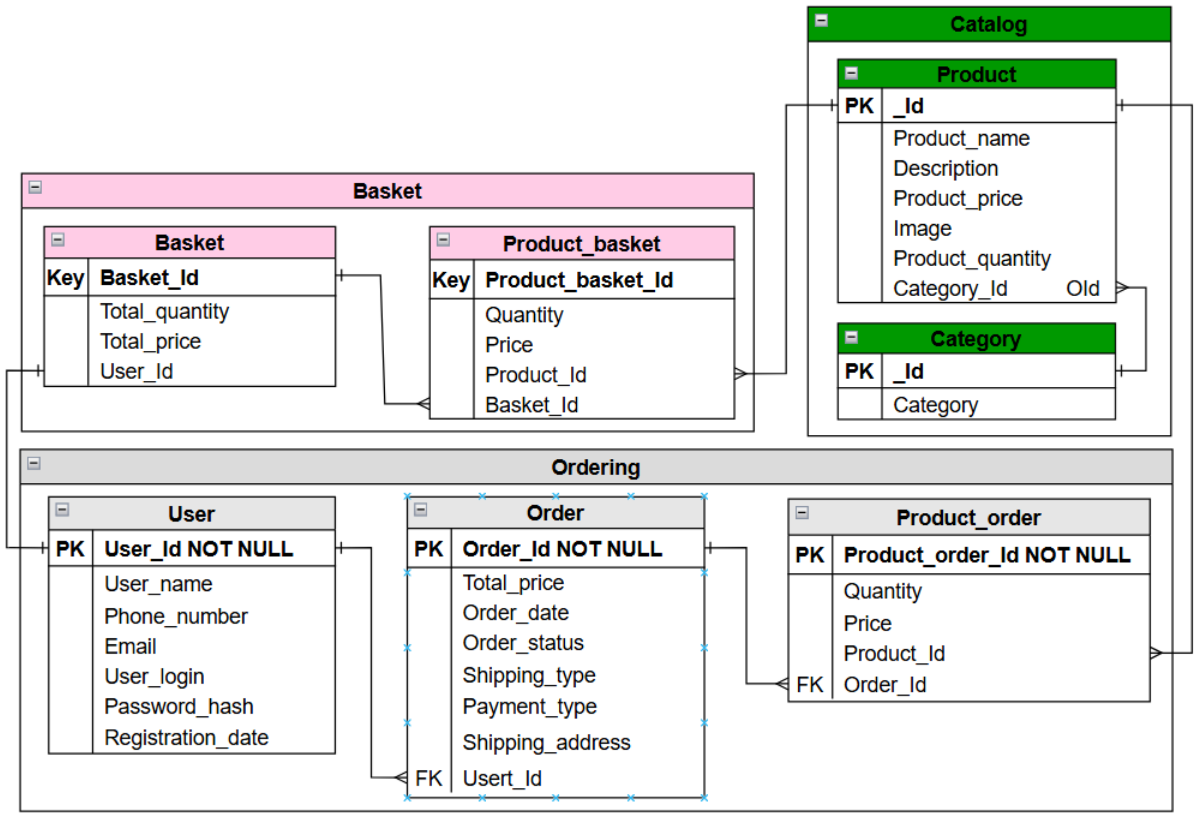

Figure 3: A fragment of a logical model of a heterogeneous database

It should be noted that during the decomposition of the global EER model, the analysis of operation execution maps allowed for interesting and more efficient redesign of some transactions and database entities, but this is a topic for a separate publication.

At the next stages, the most adequate logical database models and corresponding DBMS were selected for the selected local EER models for further implementation, as well as a logical model of a heterogeneous database was designed, namely:

- Document-oriented model and NoSQL DBMS MongoDB for the Catalog model
- Key-value NoSQL model and the corresponding Redis DBMS for the Basket model
- Relational data model and MS SQL Server DBMS for the Ordering model.

The design decisions made at the last stages are based on the peculiarities of data organization in the selected logical database models, on the relation of the selected DBMS to the CAP theorem, as well as on the presence of certain advantages indicated by the developers of these DBMS. Yes, Redis was chosen, among other things, because of the small amount of data that needs to be processed in one operation and its short-term nature. Also, the biggest need in the Basket database (getting the list of products in the basket) can be solved by storing data under a single key and the ability not to waste time on the "JOIN"-combination of data provided by the "key-value" logic. Document-oriented databases are better if there is a need for complex "JOIN" queries, which can be quickly processed due to built-in collections, in the form of which products with many properties are usually presented. Therefore, the MongoDB DBMS will allow, when working with the catalog of goods (with a more complex and real structure of this local model), to increase the speed of query execution due to getting rid of complex JOIN queries, which is possible in the case of nesting collections one into another [15].

Meanwhile, all DBMSs selected for a heterogeneous database support data caching to one degree or another, which left open the question of the effectiveness of the selected DBMSs and led to the study of caching mechanisms in them.

4.3. Analysis of Redis, MongoDB and MS SQL Server caching mechanisms

An analysis of caching mechanisms in Redis, MongoDB, and MS SQL Server DBMSs was carried out, which allowed to develop the necessary software tools for conducting research.

So, according to its structure, Redis stores data in the form of key-value pairs in RAM. This allows for very fast access to data, as requests are processed directly in memory. Redis uses several separate functions to directly operate the caching mechanism. A deletion policy where Redis deletes keys that have not been used for a significant period of time is one such [17]. Thus, DBMS allows you to store only the most relevant data in memory. Redis supports a distributed cache through replication and sharding. Replication allows you to have copies of data on several servers, which ensures higher availability and data security. Sharding allows you to distribute the caching load between several Redis servers.

MongoDB provides different caching mechanisms than Redis that allow for complex queries and data analysis, while Redis is better suited for cases where very fast access to data with low latency is required. The main data storage mechanism in MongoDB is WiredTiger [14], which uses in-memory caching to improve performance. It supports data compression and cache usage for quick access to frequently used data.

WiredTiger uses two types of caching:

- Internal cache for storing recently used data pages.
- Query plans to avoid recompiling plans for the same queries.

Support for data replication and sharding functions allows you to increase the amount of cache by distributing data between different physical machines and reducing the load on each individual server.

MS SQL Server provides a more complex caching implementation structure compared to other DBMS. The main caching mechanism is a buffer cache that stores pages of data and indexes in RAM [20]. This allows you to significantly speed up access to data, since reading from memory is much faster than from disk. The second mechanism is a cache of executed query plans. Like MongoDB, MS SQL Server stores executable query plans to avoid recompiling frequently executed queries. At the same time, optimization of the use of query plans is achieved by checking the effectiveness and relevance of the plan.

MS SQL Server supports memory-optimized tables and indexes that are stored entirely in RAM. In-Memory OLTP technology is used to support memory-optimized tables, which uses various data structures and algorithms to store tables entirely in memory.

4.4. Planning of experimental research

The following criteria for assessing the quality of caching mechanisms were chosen for conducting an experimental study:

- Cache memory hit rate (%), which estimates the share of successful cache memory accesses; the higher the coefficient, the more effective the caching mechanism in the DBMS.
- Cache miss rate (%), which determines the proportion of unsuccessful attempts to access the cache memory.
- Experiment execution time (ms); each DBMS has its own characteristics of the implementation of the experiment, so this indicator will be able to determine which of the DBMSs copes with the implementation of the experiment faster.
- The use of system resources during the execution of a request from the cache, including the use of disk space (KB) and the time of using the processor or driver (ms); the more the request from the cache consumes resources, the more bottlenecks there are in the performance of the database.
- The size of the cache memory (Mb), which is allocated for working with data; shows how sufficient the level of allocated memory is; by its values, it will be possible to determine what consequences the corresponding volume can lead to.

Based on the received logical model of the heterogeneous database (Fig. 3), for experiments it was decided to use 3 homogeneous databases built on the basis of a common ER model (Fig. 1): relational DB scheme, document-oriented logical model under MongoDB (with a "normalized" approach to the design of relationships [14]) and a logical model of the "key-value" type for the Redis DBMS. Now it is possible to provide cleaner conditions for conducting experiments. Examples of modeling for the corresponding models can be seen on the corresponding fragments of the logical model of the heterogeneous database in Figure 3.

A data set was designed to be uploaded to each DBMS. Since each of the selected DBMSs has its own data storage structure, the JSON format has become the best option for data loading. The data set contains information about users who have selected products and wish to place an order, or those who have already placed an order. Thus, such a data set reflects the widespread main stages of using the online ordering system.

The hardware and software were chosen to organize the experiments. The hardware equipped with an Intel Core i7-1185G7 CPU (4 cores, 8 threads) and 16 GB of RAM was chosen to provide sufficient performance for the conducted experiments. Windows 11 64-bit was picked as an operating system. MS SQL Server 2022, MongoDB 8.0 and Redis 7.2 were picked as the latest supported versions of chosen DBMSs.

To conduct the experiments, the operations, namely the read requests, on which the measurements will be performed (some of them are depicted on the operation execution map in Figure 2) were defined. As a rule, at the stage of adding goods to the basket, system users are interested in sorting and filtering functions according to the provided categories. Therefore, read operations for the Product entity were selected for the study, among which are:

- The operation of finding a list of goods by category with sorting by price (exp. No. 1).
- The operation of finding a list of goods in the range of the price per unit of goods (exp. No. 2).
- The operation of finding a list of products with filtering by several attributes, namely by brand and price range, with sorting by product name (exp. No. 3).
- Such operations have both simple queries and complex ones, with the addition of several filtering and sorting attributes. Thus, we were able to check at which levels the DBMS uses caching mechanisms and at which levels it does not.

Operations for Basket and Product_basket entities are also considered, which include:

- Selecting goods added to the basket by the unique Basket identifier (exp. No. 4).
- Each user's selection of goods added to the basket (exp. No. 5).
- The operation of using the aggregate function to calculate the total amount of goods added to the basket by the unique Basket identifier (exp. No. 6).

For the study, a more complex operation for the Order entity (exp. No. 7) was also included, which includes the calculation of the total cost of paid orders grouped by each user, which is suitable for analytical data and for understanding how much money was spent on goods.

5. Experiments and their discussion

Experiments were conducted on seven queries for each DBMS separately, considering the availability of symmetric data for the subject area. The results of the experiments according to the "cache hit rate" metric are shown in Table 1.

Table 1
Results of experiments using the "cache hit rate" metric (%)

№	MSSQL	Redis	MongoDB
1	99.75	100	98.72
2	0	100	98.73
3	99.93	100	98.8
4	99.77	100	98.68
5	99.81	100	98.69
6	99.78	100	98.7
7	99.80	100	98.73

According to these results, it should be concluded that Redis is the most effective for this indicator, since a value less than 100 is possible only in the absence of a suitable key for conducting the experiment. At the same time, MSSQL has a zero value for experiment number 2. Such a result signals the non-use of caching mechanisms for a query that was recognized by MSSQL as very simple. We also measured the “cache miss ratio” metric.

It should be noted the tendency for MSSQL and MongoDB not to reach the maximum values in both metrics due to the peculiarities of the implementation of caching mechanisms. Both DBMSs have built-in capabilities to check if old plans are needed, if they need to be deleted, or if an existing one needs to be replaced, and therefore it is very difficult to get a 100% result, even if the query has already been executed several times before.

The results of the experiments according to the "experiment execution time" metric are shown in Table 2.

It should be noted a significant difference for the worse in the execution time of Redis experiments compared to other DBMSs. Such numbers are due to the complexity of implementing experiments for key-value DBMSs, the main goal of which is to return data by the selected key as quickly as possible. However, such DBMSs are not capable of processing requests for filtering, sorting or any other actions related to data manipulation. Therefore, the capabilities of the StackExchange. Redis library of .NET technology were used to perform the experiments.

The results of the experiments according to the "system resource utilization" metrics are shown in Table 3 and Table 4. According to this metric, we obtained the results of the execution time of the request by the processor or DBMS driver and the amount of disk space occupied by the request. In the case of the Redis DBMS, the amount of disk space is calculated based on the specified number of keys that were used for the experiment and the occupied memory of each. For other DBMSs, this is the occupied memory of the query execution plan.

Table 2
Results of the experiments according to the "experiment execution time" metric (ms)

№	MSSQL	Redis	MongoDB
1	1.0255	25.785	2.748
2	1.750	47.489	0.8419
3	3.173	58.528	0.9769
4	1.6856	109.495	1.1563
5	2.2742	50.376	2.346
6	2.3635	85.673	1.178
7	4.7286	62.363	1.02

Table 3
The results of the experiments according to the "volume of disk space" metric (Kb)

№	MSSQL	Redis	MongoDB
1	32	2.63	0.29
2	-	4.67	0.16
3	24	1.094	0.21
4	32	6.979	0.35
5	40	3.146	0.53
6	32	4.751	0.41
7	32	2.016	0.39

Table 4
The results of the experiments according to the "CPU execution time" metric (ms)

№	MSSQL	Redis	MongoDB
1	2	9.11	6
2	-	7.79	3
3	7	6.4	11
4	18	5.2	5
5	8	7.04	8
6	6	4.45	6
7	21	13.66	11

As MSSQL did not use caching for the second experiment, the corresponding resources were not calculated. It is possible to conclude that MongoDB consumes the least amount of memory for storing query execution plans. The results of the experiments according to the "cache size" metric are shown in Table 5. The metric determines how much cache memory is allocated by each DBMS during a request to the cache. The results show that MSSQL spends quite a lot of memory on cache usage, which is due to the complex and multifaceted structure of caching mechanisms. In turn, Redis uses the least amount of memory to store cache pages and keys.

Experiments conducted for the Redis DBMS proved the effectiveness of caching for simple queries that do not require data processing from several tables. The speed of finding the necessary keys and the amount of allocated memory for caching is a significant advantage for choosing this DBMS if distributed caching is needed.

MongoDB performed best for experiments that query data from multiple DB entities. This is due to the feature of the logical model and the DBMS itself, which returns data in the form of collections and stores them in the BSON format, thereby guaranteeing the speed and efficiency of data filtering and searching. The best results were obtained despite the fact that when modeling under MongoDB, a "normalized" approach was used, which did not involve nesting collections into each other. That is why this DBMS is best suited for services related to obtaining a large volume of data. It should also be noted that MongoDB uses a relatively small amount of memory for caching, has a significant query execution speed, and shows good performance.

Table 5
Results of experiments on the "cache size" metric (Mb)

№	MSSQL	Redis	MongoDB
1	904	0.38	1.09
2	0	0.39	1.1
3	7656	0.40	1.1
4	7952	0.41	1.12
5	8272	0.42	1.12
6	8744	0.43	1.13
7	9104	0.43	1.14

MSSQL has the best structure for building caching and opportunities for obtaining data about this process. The speed of operation, minimal use of system resources and one of the best indicators of "hitting" in the cache memory make MSSQL an example of a reliable database for using caching. However, you should also be careful with the amount of data to cache, as the DBMS takes quite a large amount of memory for caching, which can reach up to 9 GB, while other DBMSs use no more than 2 MB for caching. Therefore, it is recommended to use MSSQL for caching applications with a small amount of data.

The results of the experiments made it possible to develop the following recommendations regarding the choice of DBMS for the implementation of appropriate business logic during the construction of distributed, including heterogeneous, databases.

The Redis DBMS is recommended for use in databases and operations (queries) that do not require data processing from several entities. This is due to the fact that Redis only stores data in key-value format, so the execution time of the experiments was significantly longer, ranging from 25 to 109 ms. It is recommended to use Redis for data structures that are frequently updated and do not store "archived data". This is due to the support of the TTL (Time To Live) mechanism, which sets the lifetime of the keys, as well as the involvement of LFU (Least Frequently Used) and LRU (Least Recent Used) algorithms for caching, which automatically free memory and delete the least used or oldest keys. Another key component of choosing a given DBMS is the amount of allocated memory for caching. Experiments showed a range from 0.38 to 0.43 MB, which is the best indicator among DBMS and is due to the use of ZIPLIST and INTSET storage technologies, which allow to minimize the use of memory for small arrays and sets.

Recommendations regarding the use of Redis for distributed caching should be expressed separately, since simple queries related to key searches are performed in Redis 2 times faster than in other DBMSs. Thereby, minimizing the response time to discover data in the cache.

The MongoDB DBMS is recommended to be used for databases and operations (queries) with a complex structure, which involve the combination of several entities and the possibility of a filtering or search function. This is confirmed by measurements based on the system resource metric, according to which requests occupied a very small percentage of memory, up to 1KB each. As the number of entities in the query increases, the effect of using DBMS will increase, thanks to the support of data replication and sharding functions. This allows you to increase the amount of cache by distributing data between different physical machines and reduce the load on each server. Separately, it should be noted that the design of this database in a non-normalized form will increase the efficiency in the execution of requests, since WiredTiger has a caching mechanism that uses the Least Recently Used (LRU) algorithm for cache management.

The MSSQL DBMS did not show the best results in the area of caching, but we should not forget about such strengths of MSSQL as data security, support of data integrity and ACID-properties of transactions, because of which this DBMS is chosen more often. An important feature of DBMS is the division of simple and complex queries, determining whether to use caching or not. Nevertheless, it should be noted separately the high rate of hitting the cache memory, which reaches almost 100%.

6. Conclusions

The paper presents the results of the study of the effectiveness of data caching in NoSQL DBMS Redis and MongoDB, as well as relational MSSQL, to solve the problem of choosing the most effective DBMS for database development, including heterogeneous ones.

An extended infological model of EER was proposed as a basis for the design of the HBD and the general principles of its design. The paper provides a mathematical representation of the model and a visual presentation of its extension in the form of operation execution maps, which is a convenient tool for database developers. The use of the proposed model, which considers more in-depth knowledge of data flows in the subject area, and the corresponding approach to the design of the HBD will allow developers to obtain the following advantages:

- The possibility of designing more effective data processing operations for the business logic of applications and the corresponding reduction of their execution time.
- The ability to design (decompose) databases for microservice architecture with a reasonable choice of logical database models and corresponding DBMS.
- As a result, the high scalability of the information systems based on the developed HBD is ensured.

In the future, the EER model can be supplemented with the possibility of modeling data integrity restrictions and the approach to the design of the HBD - the stage of considering such restrictions during constructing a logical model. A heterogeneous database in the field of electronic commerce was designed, and experiments were planned and carried out to study caching mechanisms in selected DBMSs in the paper. The results of the experiments made it possible to develop detailed recommendations regarding the choice of DBMSs for the implementation of distributed parts, including HDBs. Developers can use these recommendations to design real systems, particularly in electronic commerce, space engineering, and emergency risk reduction.

Declaration on Generative AI

The authors have not employed any Generative AI tools.

References

[1] Y. Zhu, Y. Li, A Data sharing and integration technology for heterogeneous databases, International journal of circuits, systems and signal processing 16 (2022) 232-238. doi:10.46300/9106.2022.16.28.

[2] H. Ayu, M. Zuhar, A. Teuku, A. Zalfie, A.B. Zuriana, R. Zulaida, Mutiawati, Development of heterogeneous database synchronization algorithm based on entity relationship model approach, in: Proceedings of the 1st International Postgraduate Conference on Ocean Engineering Technology and Informatics 2021 (IPCOETI 2021), volume 2484(1), Kuala Terengganu, Malaysia, 060016 (2023). doi:10.1063/5.0115791.

[3] K. Munonye, P. Martinek, Evaluation of data storage patterns in microservices archicture, in: 2020 IEEE 15th International Conference of System of Systems Engineering (SoSE), Budapest, Hungary, 2020, pp. 373– 380. doi:10.1109/SoSE50414.2020.9130516.
[4] A. Meier, M. Kaufmann, SQL & NoSQL Databases: Models, Languages, Consistency Options and Architectures for Big Data Management, Springer Vieweg, 2019.
[5] M. Asfand-E-Yar, R. Ali, Semantic integration of heterogeneous databases of same domain using ontology, in: IEEE Access, volume 8, 2020, pp. 77903-77919. doi:10.1109/ACCESS.2020.2988685.
[6] C.J. Date, Database Design and Relational Theory: Normal Forms and All That Jazz, Apress, 2019.
[7] A. A. Mhawes, M. A. Hassan, Methods to implement homogeneous and heterogeneous distributed database, International journal of novel research in computer science and software engineering 7(2) (2020) 8-19.
[8] A. Kuzochkina, M. Shirokopetleva, Z. Dudar, Analyzing and comparison of NoSQL DBMS, in: International scientific-practical conference Problems of Infocommunications. Science and Technology (PIC S&T), 2018, pp. 560-564. doi:10.1109/INFOCOMMST.2018.8632133.
[9] C. Gomes, E. Borba, E. Tavares, M. N. de O. Junior, Performability model for assessing NoSQL DBMS consistency, in: 2019 IEEE International Systems Conference (SysCon), Orlando, FL, USA, 2019, pp. 1-6, doi:10.1109/SYSCON.2019.8836757.
[10] M. Kleppmann, Designing Data-Intensive Applications, O'Reilly Media, 2017.
[11] G. John, Building a Hybrid SQL + NoSQL E-Commerce Data Model, 2021. URL: https://dev.to/fabric_commerce/building-a-hybrid-sql-nosql-e-commerce-data-model-3fc3.
[12] K. Melnyk, N. Borysova, V. Melnyk, The Hierarchical Information System for Management of the Targeted Advertising, in S. Subbotin (Eds.), Proceedings of The Fifth International Workshop on Computer Modeling and Intelligent Systems (CMIS-2022), Zaporizhzhia, Ukraine, 2022, pp. 288-302. URL:https://ceur-ws.org/Vol-3137/paper24.pdf.
[13] S. Bagui, R. Earp, Database Design Using Entity-Relationship Diagrams (Foundations of Database Design), Auerbach Publications, 2011.
[14] K. Chodorow, MongoDB: The Definitive Guide: Powerful and Scalable Data Storage, 3rd. ed., O'Reilly Media, 2019.
[15] O. Mazurova, I. Syvolovskyi, O. Syvolovska, NOSQL database logic design methods for MONGODB and NEO4J, Innovative technologies and scientific solutions for industries 2(20) (2022) 52–63. doi:10.30837/ITSSI.2022.20.052.
[16] V. Filatov, V. Semenets, Methods for synthesis of relational data model in information systems reengineering problems, in: 2018 International scientific-practical conference Problems of Infocommunications. Science and Technology (PIC S&T), Kharkiv, Ukraine, 2018, pp. 247-251, doi:10.1109/INFOCOMMST.2018.8632144.
[17] L. Atchison, Caching at Scale with Redis, Redis Labs, 2021.
[18] S. Palanisamy, P. SuvithaVani, A survey on RDBMS and NoSQL Databases MySQL vs MongoDB, in: 2020 International Conference on Computer Communication and Informatics (ICCCI), Coimbatore, India, 2020, pp. 1-7. doi: 10.1109/ICCCI48352.2020.9104047.
[19] O. Mazurova, A. Naboka, M. Shirokopetleva, Research of ACID transaction implementation methods for distributed databases using replication technology, Innovative technologies and scientific solutions for industries 2 (16) (2021) 19-31. doi:10.30837/ITSSI.2021.16.019.
[20] V. Filatov, V. Semenets, O. Zolotukhin, Data mining in relational systems, Innovative technologies and scientific solutions for industries 3(13) (2020) 65–76. doi:10.30837/itssi.2020.13.065.