Comparation of the Performance of SQL and NoSQL Databases in Modern Business Applications

Aniela BORCAN, Diana CAPTARI, Emanuela-Cristina CARP
Bucharest University of Economic Studies
Faculty of Cybernetics, Statistics and Economic Informatics
Bucharest, Romania
borcananiela20@stud.ase.ro, captaridiana20@stud.ase.ro, carpemanuela20@stud.ase.ro

Databases represent a fundamental element of modern business applications, directly influencing performance, scalability, and data management. This article explores the differences between SQL and NoSQL databases, analyzing the advantages and disadvantages of each technology based on criteria of performance, consistency, and specific applications. Databases are essential structures for storing and managing information, having a major impact on the functioning of business applications. The choice between a relational database (SQL) and a non-relational database (NoSQL) depends on several factors, including the type of data being handled, scalability requirements, and the data access model.

Keywords: *SQL databases (Structured Query Language), NoSQL databases (Not Only SQL), relational databases, SQL vs NoSQL, NoSQL usage, NoSQL performance, data consistency, scalability.*

1 Introduction

In the era of digitalization, the volume of data generated by companies has increased exponentially, necessitating the use of efficient solutions for storing and managing information. Databases have evolved significantly from traditional relational models to more flexible alternatives, such as NoSQL databases. This evolution has been driven by the need for companies to manage unstructured and semi-structured data, as well as to support high performance and scalability requirements [1].

The choice between a relational database (SQL) and a non-relational database (NoSQL) depends on several critical factors, including:

- Type of data used;
- The complexity of the queries;
- The need for strict consistency;
- The requirements for horizontal or vertical scalability.

SQL databases, based on the relational model, offer well-defined structures and support for ACID transactions (Atomicity, Consistency, Isolation, Durability), being widely used in critical fields such as finance and healthcare [2]. On the other hand, NoSQL databases are designed to handle large volumes of distributed data and are used in scalable web applications, Big Data analytics, and recommendation systems.

The fundamentals of databases SQL and NoSQL

SQL databases: Relational databases (SQL) are structured on a tabular model, using the SQL (Structured Query Language) query language. SQL is a language used for managing and processing data in a relational database. In a relational database, information is organized in the form of tables, each having rows and columns that represent the attributes of the data and the relationships between them. Through SQL instructions, users can store, update, delete, search, and extract data from the database. At the same time, SQL allows

for the administration and optimization of database performance.

The SQL query language is essential due to its universal use in various software applications. Data analysts and developers learn and adopt SQL due to its excellent compatibility with multiple programming languages. For example, SQL queries can be integrated into applications developed in programming languages such as Java, contributing to the creation of robust solutions for data processing within large-scale relational database systems like Oracle or MS SQL Server.

SQL is also an accessible language, considering that it uses familiar keywords in English, thus facilitating its learning. For example, the keyword "SELECT" is used to extract data from a database, "FROM" to specify the source table, "WHERE" to apply filtering conditions, and "ORDER BY" to organize the results in a specific order [3].

NoSQL databases: The term "NoSQL" refers to non-relational databases, which use a data storage format different from that of relational tables. However, NoSQL databases can be queried through APIs, using idiomatic languages, declarative structured query languages, or example-based query languages. For this reason, these databases are often referred to as "not only SQL."

NoSQL databases are widely used in real-time web applications and Big Data management, their main advantage being the ability to provide superior scalability and availability.

Additionally, NoSQL databases are preferred by developers due to their flexibility, being able to quickly adapt to the ever-changing requirements of projects. These databases allow for data storage in a more intuitive and understandable way, being closer to the format in which they are used in applications, which reduces the need for additional transformations for storage or retrieval through specific NoSQL APIs. Moreover, NoSQL databases are optimized to fully leverage cloud infrastructure, ensuring continuous and efficient operation [4].

When to choose a NoSQL database

Considering the necessity for companies and organizations to innovate rapidly, the ability to remain flexible and support operations at any scale becomes essential. NoSQL databases offer flexible schemas and support a variety of data models, making them ideal solutions for developing applications that require managing large volumes of data and low latency or reduced response times, such as web applications for online gaming and e-commerce platforms [4].

When not to choose a NoSQL database

NoSQL databases are generally built on a denormalized data model, supporting applications that use a small number of tables (or containers) and do not rely on references to correlate data, but rather on embedded records (or documents). Many traditional back-office applications of companies, such as finance, accounting, and resource planning, rely on highly normalized data to prevent anomalies and data duplication. Generally, these applications are not suitable for using a NoSQL database.

Another important aspect of NoSQL databases is the complexity of queries. Although these databases work excellently for simple queries involving a single table, when the complexity of the queries increases, relational databases become a more suitable choice. NoSQL databases usually do not offer advanced functionalities for complex joins, subqueries, or nesting queries in a WHERE clause.

In certain cases, however, it is not necessary to exclusively choose between a relational database and a non-relational one. In many situations, companies opt for hybrid databases, which allow the simultaneous use of relational and non-relational data models. This approach offers greater flexibility in managing

different types of data, while ensuring read and write consistency without compromising performance.

How does a NoSQL database work?

NoSQL databases use various data models to manage and manipulate information. These databases are designed for applications that process large amounts of data and require low latency, as well as flexible data models. This is achieved by relaxing the strict data consistency requirements characteristic of other types of databases.

To illustrate this, let's consider a simple database model for managing books.

In a relational database, information about a book is usually divided into multiple tables (a process called "normalization"), and the relationships between them are established through primary and foreign key constraints. For example, a table named "Books" might contain columns such as ISBN, Title, and a foreign key referencing an "Authors" table that contains columns like Author Name and Author ID. The relational model is created to ensure referential integrity between tables, and the data is normalized to reduce redundancy and to be optimized for efficient storage.

In contrast, in a NoSQL database, a book record is usually stored as a JSON document. In this case, for each book, information such as ISBN, Title, Edition Number, Author's Name, and Author's ID is stored together as attributes of a single document. This model is optimized for more intuitive development and horizontal scalability, allowing for efficient data management in a more flexible manner [5].

Types of NoSQL databases

NoSQL databases are used in situations where storing data in tables is not optimal. They use various storage formats, each with specific applications. There are six main types of NoSQL databases, each with distinct characteristics.

- **Key-value pair-based databases**
  These databases allow for high separability and extensive horizontal scalability, which is impossible to achieve in a similar manner with other types of databases. They are ideal for applications that require high performance and low latency, such as gaming, advertising, and IoT applications. A notable example is Amazon DynamoDB, which offers consistent performance with a latency of a few milliseconds, even at large scale. These features were essential for migrating Snapchat Stories to DynamoDB, given the high volume of writes recorded.
- **Document-type databases**
  In this category, data is stored in the form of documents in formats such as JSON, which are directly compatible with the objects used in the application's code. These databases are extremely flexible and allow for the storage and querying of data in an intuitive way, like how data is structured in the application's code. Document models are useful for applications that involve catalogs, user profiles, or content management systems. Examples of document databases are Amazon DocumentDB (compatible with MongoDB) and MongoDB, which offer intuitive APIs for agile development.
- **Graph databases**
  Graph databases are used for applications that work with complex datasets, such as social networks, product recommendations, fraud detection, and knowledge graphs. These databases are optimized for storing and manipulating relationships between entities. Amazon Neptune is a fully managed service that supports Property Graph and Resource Description Framework (RDF) graph models. Other examples include Neo4j and Giraph.

- **In-memory databases**
  These databases are used in applications that require quick responses in microseconds, such as gaming and online advertising, where traffic can spike suddenly. Amazon MemoryDB for Redis is an in-memory database service that reduces data read latency and ensures reliability at scale, making it ideal for microservices-based applications. Amazon ElastiCache, compatible with Redis and Memcached, serves workloads that require low latency and high throughput, while Amazon DynamoDB Accelerator (DAX) enhances its performance by reading data several times faster.
- **Databases for searches**
  Many applications generate logs to assist with troubleshooting and problem analysis. Amazon OpenSearch is a service dedicated to the real-time analysis and visualization of data streams, indexing, aggregating, and searching information from logs and semi-structured data. OpenSearch also offers high-performance full-text search capabilities, being used, for example, by Expedia, which manages 30 TB of data and 30 billion documents for cost optimization and operational monitoring.

SQL Data Types

Databases SQL uses a well-defined set of data types, which are applied in a structure of tables with rows and columns. Each column in a table is defined with a specific data type, and these types are very rigid and normalized.

– Numbers:
  - INT: For storing integer numbers.
  - DECIMAL or NUMERIC: For numbers with high precision decimals.
  - FLOAT or DOUBLE: For floating-point numbers.
– Strings:
  - VARCHAR: For variable-length character strings.
  - CHAR: For fixed-length character strings.
  - TEXT: For long character strings.
– Date and time:
  - DATE: Used to store only the date (year, month, day).
  - TIME: Used to store only the time.
  - DATETIME or TIMESTAMP: Used to store both the date and time in a single field.
– Boolean:
  - BOOLEAN: For storing values like true/false.
– Binary:
  - BLOB: For binary data, such as images or files.
– Others:
  - ENUM: To define a limited list of possible values for a field.
  - SET: To store a set of values.

Tools and Frameworks for Managing NoSQL Databases

MongoDB Atlas is a fully managed platform for MongoDB, which offers cloud services for managing MongoDB databases. It allows developers to create, monitor, and scale MongoDB-based applications without having to manage the infrastructure [6].

Use cases:

- Development of modern web applications.
- Mobile and IoT applications.
- Content management systems and user profiles.

Apache Cassandra is a distributed NoSQL database that is extremely scalable and fault-tolerant. It is ideal for applications that require a large volume of distributed data and cannot tolerate service interruptions.

Use cases:

- Applications that manipulate Big Data, such as real-time data analysis.

- Write-heavy distributed systems.
- Social media and e-commerce applications that require massive scalability.

Redis is an in-memory NoSQL database that offers high performance for storing and manipulating temporary data, such as sessions, queues, and caches. Redis is widely used in applications that require low latency and fast response.

Use cases:

- Storing sessions in web applications.
- Caching to accelerate data access.
- Message queues and jobs in distributed applications.

Couchbase is a document-oriented NoSQL database that combines the features of document databases with those of key-value databases. It is optimized for applications that require high scalability and performance.

Use cases:

- Web and mobile applications with high scalability and performance requirements.
- Applications for managing semi-structured data.
- Authentication and user management systems.

Amazon DynamoDB is a fully managed NoSQL database service offered by AWS, which ensures automatic scalability and high performance for applications that require a large volume of data and low latency [6].

Use cases:

- Mobile and web applications that require rapid scalability and low costs.
- IoT and Big Data applications.
- Management of structured and semi-structured data.

Comparing performance between SQL and NoSQL

Performance in CRUD operations:

CRUD operations (Create, Read, Update, Delete) are essential for any database management system. Their performance varies depending on the database architecture.

SQL: Read efficiency, write difficulties:

- SQL databases are optimized for complex queries and read operations. Due to advanced indexing and normalization, they allow for rapid data retrieval.
- Write operations (Create, Update) can be slower due to strict referential integrity checks and asynchronous index updates.

NoSQL: Optimized for writing, high flexibility:

- NoSQL databases are faster for write operations because they do not require strict integrity checks.
- They use techniques such as partitioning (sharding) and replication to efficiently distribute the workload.

Practical example:

Financial analysis applications: SQL is preferred due to complex queries.

Logging systems: NoSQL is ideal due to its high data write speed.

Consistency Models and Transaction Management:

Data consistency is a critical factor in choosing a database, influenced by the ACID and BASE models.

SQL: The ACID Model (Atomicity, Consistency, Isolation, Durability):

- Offers reliable and secure transactions.
- It is essential in financial, medical, or other fields where data integrity is crucial.
- Cost: Performance can be affected by the bottlenecks caused by maintaining strict consistency.

NoSQL: The BASE Model (Basically Available, Soft-state, Eventual consistency):

- It offers high availability but allows eventual consistency.
- It is used in distributed applications, such as social networks or caching systems.
- Cost: Data can be temporarily inconsistent.

Practical example:

- Banks and hospitals: They require strict consistency, so SQL is preferred.
- Social networks: They can tolerate eventual consistency, so NoSQL is more suitable.

Scalability and Availability:

Scalability is essential for managing the growth of data volume and the number of users.

SQL: Vertical scalability, limitations on horizontal:

- Performance improvement is achieved by adding hardware resources (vertical scalability).
- Horizontal scalability (adding nodes) is difficult due to the complex relationships between tables.

NoSQL: Horizontal scalability, efficient distribution:

- Allows for the easy addition of new nodes.
- Techniques such as sharding and replication ensure balanced data distribution.
- It is used in large systems, such as Amazon, Facebook, or Google.

Practical example:

- Banking systems: The vertical scalability of SQL is sufficient for many financial applications.
- E-commerce: NoSQL allows for rapid scaling and handling of high traffic [7].

Current trends and the future of databases in modern business applications

In today's digital era, databases play an essential role in managing information for modern business applications. Current applications require high scalability, optimized performance, and great flexibility to handle the volume of data in an ever-changing world. In this context, recent and future trends in the field of databases, both SQL and NoSQL, are directed towards solutions that meet the complex needs of modern businesses.

Another important trend in database development is the migration to cloud databases and serverless databases. Currently, many business applications are migrating to cloud-based solutions to reduce costs and the complexity of infrastructure management. Cloud databases, such as AWS Aurora or Google Cloud SQL, offer flexibility and scalability, being managed by cloud service providers. Additionally, the concept of serverless databases, exemplified by Firebase Firestore and AWS Aurora Serverless, allows applications to automatically scale based on traffic needs, without the need to manage a server or fixed resources. These solutions are extremely attractive for startups and businesses that do not want to invest heavily in infrastructure but need performance and scalability as their business grows.

At the same time, another significant trend in database usage is the adoption of graph databases, especially for analyzing complex data. Graph databases, such as Neo4j or Amazon Neptune, are increasingly used by businesses that need to analyze complex relationships between entities in their data systems, such as users, products, transactions, or connections in social networks. These databases allow for a much more intuitive representation of relationships between data, facilitating analyses and the discovery of relevant information, such as fraud detection, product recommendations, or supply chain optimization.

In addition to these, Artificial Intelligence (AI) and Machine Learning

(ML) are having an increasingly significant impact on the management and optimization of databases. AI algorithms are used to analyze query behavior and optimize database performance by dynamically adjusting allocated resources (storage, CPU, memory) based on application requirements. Additionally, AI is used to identify anomalies and predict potential system failures before they impact operations, thereby contributing to reduced downtime and increased reliability.

Another important direction for the future of databases in business applications is the integration of multi-model databases. Businesses that have diverse requirements regarding data types (relational, graph, document, etc.) can adopt database solutions that integrate multiple data models into a single platform. This allows them to meet various storage and processing requirements, providing greater flexibility in data management.

One of the most notable aspects of the evolution of databases in modern business applications is the transition to solutions capable of providing horizontal scalability. While traditional SQL databases were excellent for managing complex relationships between data, they had limitations in terms of horizontal scalability, meaning they could not efficiently scale as the volume of data grew exponentially. In contrast, NoSQL databases, such as Cassandra and MongoDB, are built to support excellent horizontal scalability, allowing business applications to manage large volumes of data distributed across multiple servers. These solutions are ideal for applications that handle semi-structured or unstructured data, such as those in the field of IoT (Internet of Things) or Big Data, making them a popular choice for businesses that need to process data quickly in a distributed manner [8, 9].

Case study: Migration from SQL to NoSQL

A notable example of migration from SQL to NoSQL is the case of Netflix, a global video streaming platform, which migrated from a relational database (SQL) architecture to an infrastructure based on NoSQL databases, such as Cassandra and Amazon DynamoDB. The decision was motivated by the need to scale rapidly and efficiently as their storage and data processing requirements grew exponentially, in the context of an increasing number of users and video content.

Data Migration: The migration was a multi-step process, involving an initial massive migration ("forklift"), followed by incremental replication and data consistency validation. These stages aimed to transfer data from SimpleDB to Cassandra without interrupting the service and without degrading performance:

- Forklift (Initial Migration): In the first stage, all data from SimpleDB was copied to Cassandra. The process was carried out over the course of 30 hours, and the number of threads and instances was adjusted to minimize the impact on SimpleDB's performance during the migration.
- Incremental Replication: After the forklift, changes from the Queue continued to be written only in SimpleDB. Incremental replication ensured that the changes made to the data in the post-migration period were also reflected in Cassandra.
- Consistency Checker: To verify if the migration and incremental replication were correct, a consistency checker was used. It compared the data between SimpleDB and Cassandra, updating only the records that did not match between the two databases.
- Shadow Writes: At this point, data continued to be read from SimpleDB,

but for each Queue update, the data was also written to Cassandra. This allowed for the validation of performance and consistency of writes in Cassandra, without disrupting the existing user flow.

- Shadow Reads (Parallel Reads): After shadow writes were implemented, the shadow reads functionality was also activated, where the data from the Queue was read from both SimpleDB and Cassandra. The differences between the data from the two sources were monitored and corrected.
- Ending the use of SimpleDB: After the discrepancies between SimpleDB and Cassandra were minimal (<0.01%), SimpleDB was eliminated, and Cassandra became the primary source of truth for the data in Queue. All reads and writes to Queue were now exclusively managed by Cassandra.

Results: The migration from SimpleDB to Cassandra allowed Netflix to improve the scalability, performance, and availability of the Queue service, without interruptions and with minimal impact on users.

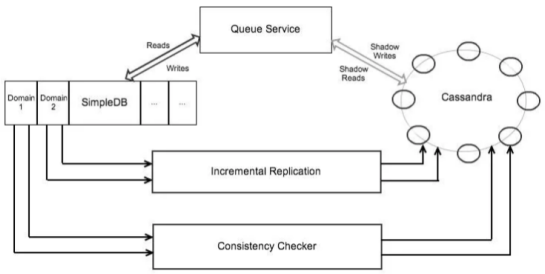

Fig. 2. Migration from SQL to NoSQL

Advantages of migration:

- Horizontal scalability: NoSQL allows the addition of nodes to manage large volumes of data, unlike SQL, which has limitations in this regard.
- Improved write performance: NoSQL is more efficient for write operations, ideal for high-traffic platforms like Netflix.
- Flexibility: NoSQL allows for the storage of semi-structured and unstructured data, which is suitable for movie metadata and user interactions.
- Availability and fault tolerance: NoSQL, such as Cassandra, offers continuous availability even in the event of hardware failures.

Disadvantages of migration:

- Costs and complexity in migration: Migrating from SQL to NoSQL required significant investments in resources and time, and development teams had to learn new paradigms.
- Lack of ACID transactions: NoSQL does not guarantee the same complex transactions and data integrity as SQL, which can be problematic for financial applications.
- Challenges with complex queries: NoSQL databases are not ideal for complex queries, such as joins between tables, requiring alternative solutions like Apache Kafka and ElasticSearch.
- Eventual consistency: NoSQL uses the eventual consistency model, which can lead to temporary data inconsistencies.

Migration issues and solutions:

- Team adaptability: Teams had to learn the new architectures and paradigms. The solution was to form specialized teams in NoSQL and conduct training sessions.
- Data inconsistency during migration: A gradual migration process was used, with both databases running in parallel, to minimize risks.
- Monitoring: An internal monitoring system was implemented to track the performance and health of the distributed NoSQL database [10, 11, 12].

What are hybrid solutions and interoperability between SQL and NoSQL?

Hybrid solutions allow the integration of both relational and non-relational databases into a single technological architecture, to meet a wide range of requirements related to performance, scalability, flexibility, and data consistency. These solutions aim to utilize the best features of each type of database according to the specific needs of the applications, without sacrificing performance or data integrity.

Challenges of integration:

- Implementation complexity: Integrating SQL and NoSQL databases into a hybrid solution can be technically complex. This can involve managing multiple data storage solutions and query mechanisms, which can lead to increased operational costs and the need for specialized technical teams.
- Data consistency: In some cases, NoSQL databases allow for a weaker consistency model (in accordance with the eventual consistency model), while SQL databases guarantee consistency through ACID transactions. In a hybrid system, it can be difficult to maintain uniform consistency between the two databases, which can lead to issues related to data integrity.
- Integrating queries and APIs: Another challenging aspect is the integration of queries between SQL and NoSQL databases. Each type of database has its own query language (SQL for SQL and custom APIs for NoSQL), and combining them may require additional development effort.

Examples of hybrid systems that combine SQL and NoSQL:

- **Microsoft Azure Cosmos DB**:A notable example of a hybrid solution is Microsoft Azure Cosmos DB, which supports both SQL and NoSQL data models. Cosmos DB offers the integration of a relational database model based on SQL, as well as the possibility to use various NoSQL models, such as document, graph, and key-value models. It thus offers a scalable platform that can manage both structured and semi-structured or unstructured data as applications develop [13].
- **MongoDB with SQL integration:** Although MongoDB is a document-based NoSQL database, there are solutions like MongoDB Atlas that allow integration with SQL databases. This allows developers to manage both relational and semi-structured data simultaneously. Additionally, MongoDB includes advanced search and indexing functionalities that are compatible with SQL queries, and users can also use APIs to interact with the data in a hybrid manner [14].
- **PostgreSQL with NoSQL extensions:** PostgreSQL, one of the most popular SQL relational databases, has implemented extensions to support NoSQL functionalities. For example, with the HStore extension, PostgreSQL can store semi-structured data in a key-value pair format. Additionally, with the JSONB extension, PostgreSQL can store and query JSON documents, making it compatible with the data models used by NoSQL databases [15].
- **Amazon Aurora and DynamoDB:** In the AWS ecosystem, Amazon Aurora (a relational database compatible with MySQL and PostgreSQL) can be used together with Amazon DynamoDB, a NoSQL database. Aurora is used for managing transactions and relational data, while DynamoDB is used for managing data that

requires scalability and low latency. This combination offers a robust hybrid solution that can be rapidly scaled according to the application's needs [16].

- **Couchbase and SQL:** Couchbase is a document-oriented NoSQL database that provides support for SQL queries through the N1QL language. This allows users to write standard SQL queries to query data stored in JSON format, thus combining the advantages of a NoSQL model with the familiarity of SQL queries [17].

Modern database architectures: Polyglot Persistence and microservices

In modern business applications, database performance is closely tied to how they are integrated into the overall system architecture. The microservices model, increasingly adopted, assumes that each functional module can use a database specific to its needs. This principle is called Polyglot Persistence and allows the combination of relational databases (SQL) with non-relational ones (NoSQL) [18].

A practical example is in e-commerce: the SQL database is used for processing orders and transactions where consistency and integrity are critical while NoSQL is preferred for comments, reviews, or recently viewed products, where flexibility and speed are essential.

This approach brings increased flexibility and allows for independent scaling of components. At the same time, technologies like Apache Kafka facilitate data exchange between microservices, eliminating bottlenecks between components and providing a more adaptable and stable architecture.

According to Fowler [18], using a combination of databases within a complex ecosystem helps avoid the limitations specific to each type of technology, thereby increasing the application's reliability.

Artificial Intelligence and Machine Learning in databases

The performance of modern databases no longer depends solely on indexes or well-designed structures, but also on the use of advanced technologies such as artificial intelligence (AI). More and more solutions use machine learning algorithms to automatically optimize how queries are executed or resources are managed [19].

For example, Oracle Autonomous Database can automatically adjust the execution plan of a query or decide when and how to create indexes, without human intervention. On the NoSQL side, systems are becoming increasingly intelligent in resource management: they learn application behaviors and pre-load the most frequently accessed data, optimizing cache memory and response times.

This AI-based automation not only optimizes performance but also helps reduce human errors. In a large-scale organizational environment, where data is critical and updates are frequent, AI becomes a reliable ally for ensuring service continuity [20]. Özsu and Valduriez explain in detail this type of AI integration in distributed systems in their seminal work [19].

Real-time data management

We live in an era of instantaneity, users and businesses expect real-time reactions and analyses. In this context, databases must be capable of processing large volumes of data continuously and quickly [21].

NoSQL solutions are often the primary choice for such scenarios. They easily integrate with real-time processing platforms like Apache Kafka or Apache Flink and can handle real-time data streams, such as bank transactions, messages from IoT sensors, or user activities on a website.

On the other hand, SQL has also evolved, and some modern solutions (such as

Snowflake or ClickHouse) offer very good performance in the near-instant processing of semi-structured data. However, these usually come with a higher cost and a steeper learning curve.
As Tyler Akidau shows in the work "Streaming Systems" [21], systems capable of handling real-time data are essential for modern applications that cannot afford processing delays.

Cost comparison: SQL vs NoSQL in the cloud

Beyond performance, cost is one of the most sensitive criteria for any company. In the cloud, the differences between SQL and NoSQL become evident from the perspective of pricing and flexibility. SQL is usually billed based on reserved resources (processor, RAM, storage space), which means you pay even when the database is inactive. In contrast, many NoSQL services such as DynamoDB, allow flexible billing based on the actual number of operations performed [22].
NoSQL also has the advantage of open-source implementations that can completely eliminate licensing costs. However, in such cases, other hidden costs come into play: maintenance teams, security, backup. It is important for a Total Cost of Ownership (TCO) analysis to be conducted before the final decision is made [23].
According to Joe Weinman, in "Cloudonomics" [23], a balanced approach between performance and costs is essential to achieve a sustainable competitive advantage in dynamic digital environments.

The impact of database choice on user experience

The choice of database not only affects the technical aspect of the application but also the way users interact with it. If the data loads slowly, if there are consistency errors, or if certain functions are unavailable at key moments, the user will perceive the application as weak, regardless of how well the code is written.
NoSQL databases, by their distributed nature and optimization for fast reads, can provide an extra level of fluidity. This is especially evident in social applications (e.g., Facebook with TAO) or messaging apps (e.g., Snapchat with DynamoDB), where responses need to come in milliseconds.
In contrast, applications where accuracy is more important than speed, such as financial or medical reporting will benefit more from an SQL database, which offers strict consistency and secure transactions [24].

Performance benchmarks: SQL vs NoSQL in practice

To truly understand the performance differences between SQL and NoSQL, it is important to analyze concrete data from independent benchmarks. For example, in the tests conducted by the Yahoo! Cloud Serving Benchmark (YCSB), it was found that MongoDB (NoSQL) has superior performance in distributed write operations, while PostgreSQL (SQL) stands out for its efficiency in complex JOIN queries [25, 26].
These benchmarks highlight that the choice of database is not universal: for applications involving complex aggregations and multiple relationships between data, SQL is superior. In contrast, for applications that require fast writes and horizontal scalability, NoSQL is more efficient. The study conducted by Zdravevski et al. (2023) [26] confirms this conclusion, indicating that MySQL offers stable performance in OLTP tasks, while MongoDB excels in handling semi-structured data in Big Data tasks.

Choosing the database based on the domain

The selection between SQL and NoSQL databases must be guided by the specific requirements of the application domain.

For example, in the financial sector, where data consistency and integrity are essential, SQL databases are often preferred due to their compliance with ACID properties [27]. In contrast, in fields such as social networks or video game applications, where scalability and flexibility are crucial for managing large volumes of data and high interaction rates, NoSQL databases offer significant advantages.

A notable example is Netflix's use of Cassandra for managing personalized recommendations, demonstrating NoSQL's ability to quickly process large amounts of unstructured data and provide real-time responses. In e-commerce, a hybrid approach is often preferred: SQL is used for transactions, while NoSQL is used for the product catalog and user interactions.

Challenges in adopting NoSQL

Although NoSQL databases offer numerous benefits, their adoption also comes with significant challenges. One of the main difficulties is the lack of native support for ACID transactions, which can affect data consistency in critical applications [28]. To address this issue, researchers such as Alflahi et al. (2023) have proposed transaction management models in MongoDB that include specialized locking algorithms and logical separation of read/write operations, improving both performance and data consistency [28].

Another challenge is the lack of standardization among different NoSQL solutions, which can lead to interoperability issues and risks related to vendor lock-in. Therefore, it is essential for organizations to conduct pilot tests and choose vendors with active support and a strong technical community.

NewSQL and multi-model databases

To combine the advantages of relational databases with the flexibility and scalability offered by NoSQL, hybrid solutions such as NewSQL and multi-model databases have emerged. NewSQL aims to provide the performance and scalability of NoSQL while maintaining the ACID properties characteristic of SQL [29]. These systems are designed to efficiently manage online transaction processing (OLTP) tasks in the context of Big Data.

On the other hand, multi-model databases, such as ArangoDB and OrientDB, allow for the storage and querying of data using multiple models (documents, graphs, columns), providing increased flexibility in managing various types of data and complex relationships [27]. These solutions represent a bridge between the advantages of both worlds, allowing developers to choose the most suitable data model for each component of their application.

Consistency models in distributed systems: trade-offs and options

In modern distributed architectures, the choice of consistency model has a significant impact on the performance and behavior of the application. NoSQL databases like Cassandra or DynamoDB adopt the eventual consistency model, which offers availability and scalability at the expense of immediate consistency. On the other hand, traditional SQL systems promote strong consistency, ideal for critical applications where data must always be accurate [30].

More and more modern databases offer configurable consistency options. For example, Cosmos DB allows the choice between five consistency levels (including bounded staleness and session consistency), giving developers fine control over the balance between performance and accuracy [31]. This flexibility comes with direct implications for architectural design: stricter consistency involves increased latency and higher replication costs, while weak consistency can lead to users viewing outdated data.

Another important aspect is the integration of consistency models based on the type of data and usage. In critical applications, such as banking or healthcare systems, strict consistency is preferred to ensure the absolute correctness of transactions. In contrast, for social network applications, a small temporary lag between the states of replicas is acceptable, as long as availability and scalability are maximized.

The CAP theorem (Consistency, Availability, Partition tolerance) remains a fundamental theoretical benchmark in this context. In a distributed environment, only two of the three characteristics can be guaranteed simultaneously. The choice of the right strategy thus depends on the application's priorities: performance, reliability, or absolute consistency.

Query and index optimization in dynamic environments

Query optimization is a constant challenge in managing large and dynamic databases. Traditional query optimization algorithms rely on estimated costs and the analysis of data statistics, but in modern environments characterized by constantly changing data, this approach is not always sufficient.

PostgreSQL, for example, uses a cost-based planner that evaluates all possible data access strategies and selects the one with the lowest estimated cost. However, in distributed environments, costs can vary over time, and dynamic query reoptimization becomes a necessity.

NoSQL databases, such as MongoDB and Couchbase, offer powerful tools for performance optimization through compound indexes, geospatial indexes, or TTL indexing. For example, in location-based applications, geospatial indexes dramatically reduce the response time for queries involving coordinates.

Another growing trend is the use of "adaptive indexing" algorithms, which dynamically adjust the structure of the indexes based on user access patterns. This technique is effective in applications with variable traffic, as it eliminates the need for constant manual reconfiguration.

Modern observability and profiling tools, such as Query Profiler in MongoDB or EXPLAIN ANALYZE in PostgreSQL, provide granular visibility into the time spent at each stage of execution. Thus, developers can quickly identify bottlenecks and propose targeted or strategic optimizations [32, 33].

Database security in modern architectures

Database security is a constantly evolving field, considering the increase in the number and complexity of cyberattacks. In modern cloud-native architectures, data circulates through multiple layers and systems, significantly increasing the risk of interception, loss, or unauthorized modification.

In the case of relational databases, protection mechanisms are well-defined and include column or table-level encryption, role-based access control (RBAC) policies, and detailed operation logging. Systems like Oracle or SQL Server offer advanced auditing functionalities, including for preventing internal threats.

NoSQL databases have also evolved significantly in this regard. MongoDB, Couchbase, or Cassandra now offer support for full encryption (end-to-end), certificate-based authentication, and integration with complex identity management systems (LDAP, Kerberos). The new versions also include audit functions, protection against injection attacks, as well as field-level security controls.

Additionally, distributed systems require the protection of communication channels between nodes. The TLS 1.3 protocol is used for traffic encryption, and network access control is achieved through security groups, isolated subnets, and declarative firewall policies.

A key element is also context-based rights management. Modern systems implement models such as Attribute-Based Access Control (ABAC), which allow flexible policies based on user, device, or location attributes. Thus, a user can access a specific database only under certain conditions (e.g., only within the company's network and during working hours).

In the future, a closer integration between databases and cybersecurity threat analysis platforms is expected. These will enable the automatic identification and blocking of suspicious behavior in real-time, through behavioral analysis and machine learning [34, 35, 36].

Integration of databases into Big Data and Artificial Intelligence ecosystems

An increasingly important development direction in modern business applications is the integration of databases with Big Data processing platforms and artificial intelligence, aspects also addressed in the recent works of Abadi and Pavlo [37, 38]. In this context, both SQL and NoSQL play essential roles.

SQL databases are used in the stages of data preprocessing and cleaning, where the rigid structure and transactionality ensure data integrity. On the other hand, NoSQL databases are preferred for storing and quickly accessing large volumes of unstructured data, used in training machine learning algorithms.

For example, Apache Spark offers native connectors for both PostgreSQL and MongoDB or Cassandra, allowing for distributed in-memory processing. Thus, data can be extracted from relational databases, transformed, and then combined with NoSQL sources for predictive analytics or real-time recommendation systems.

More and more modern databases are starting to integrate artificial intelligence-based functionalities directly into their structure. Oracle Autonomous Database and Microsoft SQL Server use machine learning algorithms for auto-tuning and anomaly detection, according to the technical documentation and research published by Microsoft and Oracle [39] for automatic query optimization, anomaly detection, and index proposal.

Integration with artificial intelligence also entails challenges: massive volumes of data must be managed, processing ethics ensured, and traceability of automated decisions guaranteed. For this reason, hybrid systems that combine the flexibility of NoSQL with the stability offered by SQL become ideal solutions for complex organizations with diverse data management needs.

Conclusions

The choice between SQL and NoSQL depends on the application's context. There is no universal solution. Relational databases are essential for critical applications that require complex transactions and strict consistency, while NoSQL offers flexibility and scalability for modern applications that handle large volumes of unstructured data.

Migrating to NoSQL brings benefits, but also challenges. Cases like Netflix highlight the advantages of horizontal scalability and superior performance for distributed data. However, the transition from SQL to NoSQL involves significant technical challenges, particularly in maintaining data consistency and adapting the architecture.

Hybrid and multi-model architectures are increasingly being adopted. Combining the flexibility of NoSQL with the reliability of SQL, along with approaches like Polyglot Persistence and microservices, offers an adaptable solution for complex organizations, allowing for independent scaling and varied technological integration.

Artificial intelligence fundamentally changes the way databases are designed and managed. AI integration allows for automatic query optimization, dynamic resource adjustment, and early error

detection, reducing human intervention and increasing operational efficiency.

AI and machine learning are becoming integral to the future direction of databases. Machine learning algorithms are increasingly used for predictive analysis, intelligent resource allocation, and anomaly detection, while graph databases are gaining ground in analyzing complex relationships and information networks.

Cloud-native technologies and serverless architectures are redefining database infrastructure. Migrating to cloud-based solutions and pay-per-use models allows companies to reduce operational costs and dynamically manage resources without compromising performance or security.

References

[1] "The Evolution of Database Technologies", *IEEE Journals*, 2023

[2] "SQL vs NoSQL: Choosing the Right DBMS," SimeonOnSecurity. [Online]. Available: https://ro.simeononsecurity.com/articles/sql-vs-nosql-choosing-the-right-database-management-system. [Accessed: March 24, 2025]

[3] "Ce este SQL?" *NewTech*. [Online]. Available: https://www.newtech.ro/blog-ce-este-sql/. [Accessed: March 24, 2025].

[4] Oracle, "What is NoSQL?" [Online]. Available:https://www.oracle.com/ro/database/nosql/what-is-nosql/. [Accessed: March 24, 2025].

[5] CodeGym, "Hibernate: SQL vs NoSQL." [Online]. Available: https://codegym.cc/ro/quests/lectures/ro.questhibernate.level19.lecture00. [Accessed: March 24, 2025].

[6] Guru99, "SQL Tools Overview." [Online]. Available: https://www.guru99.com/ro/sql-tools.html. [Accessed: March 25, 2025].

[7] "Conf-TehStiint-UTM-StudMastDoct-2024-V1-p585-588.pdf," *Proceedings of UTM Conference*, pp. 585–588, Chisinau, Moldova, 2024. [Online]. Available: https://repository.utm.md/bitstream/handle/5014/27928/Conf-TehStiint-UTM-StudMastDoct-2024-V1-p585-588.pdf. [Accessed: March 25, 2025].

[8] Budibase, "Data Management Trends." [Online]. Available: https://budibase.com/blog/data/data-management-trends/. [Accessed: March 24, 2025].

[9] "Database Management Trends in 2024," *Dataversity*. [Online]. Available: https://www.dataversity.net/database-management-trends-in-2024/. [Accessed: March 24, 2025].

[10] Netflix Tech Blog, "Netflix Queue Data Migration for a High-Volume Web Application." [Online]. Available: https://netflixtechblog.com/netflix-queue-data-migration-for-a-high-volume-web-application-76cb64272198. [Accessed: March 26, 2025].

[11] Netflix Tech Blog, "NoSQL at Netflix." [Online]. Available: https://netflixtechblog.com/nosql-at-netflix-e937b660b4c. [Accessed: March 26, 2025].

[12] P. Krill, "Big Movies, Big Data: Netflix Embraces NoSQL in the Cloud," *InfoWorld*, 2012. [Online]. Available: https://www.infoworld.com/article/2171162/big-movies-big-data-netflix-embraces-nosql-in-the-cloud.html. [Accessed: March 26, 2025].

[13] Microsoft, "Azure Cosmos DB Documentation." [Online]. Available: https://learn.microsoft.com/en-us/azure/cosmos-db/. [Accessed: March 26, 2025].

[14] MongoDB, "Atlas Database Platform." [Online]. Available: https://www.mongodb.com/products/

platform/atlas-database. [Accessed: March 26, 2025].
[15] PostgreSQL, "JSON Data Types." [Online]. Available: https://www.postgresql.org/docs/current/datatype-json.html. [Accessed: March 26, 2025].
[16] Amazon Web Services, "AWS Home." [Online]. Available: https://aws.amazon.com/. [Accessed: March 27, 2025].
[17] Couchbase, "Documentation Home." [Online]. Available: https://docs.couchbase.com/home/index.html. [Accessed: March 27, 2025].
[18] M. Fowler, "*NoSQL and Polyglot Persistence*," martinfowler.com, 2012.
[19] T. Özsu, P. Valduriez, *Principles of Distributed Database Systems*, Springer, 2020
[20] Oracle, "*AI-powered database optimization*," oracle.com, 2023.
[21] T. Akidau et al., *Streaming Systems: The What, Where, When, and How of Large-Scale Data Processing*, O'Reilly Media, 2018.
[22] Amazon Web Services, "Amazon DynamoDB Pricing." [Online]. Available: https://aws.amazon.com/dynamodb/pricing/ [Accessed: March 27, 2025].
[23] J. Weinman, *Cloudonomics: The Business Value of Cloud Computing*, Wiley, 2012.
[24] M. Stonebraker, "*The Case for NewSQL Databases*" *Communications of the ACM*, 2018.
[25] Yahoo! Cloud Serving Benchmark, "*YCSB Project*" GitHub Repository, 2023.
[26] E. Zdravevski et al., "*Comparison of SQL and NoSQL databases with different workloads: MongoDB vs MySQL evaluation*," *International Journal of Database Management Systems*, vol. 6, no. 3, pp. 1-14, 2023.
[27] S. Patro, "*SQL vs. NoSQL: Choosing the Right Database for Your Data Needs*," *LinkedIn*, 2024.
[28] A. A. E. Alflahi et al., "*An Enhanced Model for Transactional Consistency in MongoDB*," *arXiv preprint arXiv:2308.13921*, 2023.
[29] A. B. M. Moniruzzaman, "*NewSQL: Towards Next-Generation Scalable RDBMS for Online Transaction Processing (OLTP) for Big Data Management*," *arXiv preprint arXiv:1411.7343*, 2014.
[30] E. A. Brewer, *CAP twelve years later: How the 'rules' have changed*, *Computer*, 2012.
[31] Microsoft Azure Cosmos DB, *Consistency levels documentation*, 2024
[32] A. Pavlo et al., *Self-tuning database systems: a review*, *ACM Computing Surveys*, 2021
[33] M. Stonebraker et al., *Readings in Database Systems*, 5th Edition, MIT Press, 2015.
[34] A. Gupta et al., *Security in Modern Databases: Challenges and Techniques*, *ACM Computing*
[35] S. Ghazali et al., *A Survey on Secure NoSQL Databases in Cloud Environment*, *Journal of Network and Computer Applications*, 2020.
[36] Microsoft Docs, *SQL Server Security Features*, Microsoft.com, 2024.
[37] D. Abadi, *Data Management in the Cloud: Limitations and Opportunities*, *IEEE Data Eng. Bull.*, 2020.
[38] A. Pavlo, *Big Data meets Big Query: Lessons from Real-World Systems*, *Proceedings of the VLDB Endowment*, 2021.
[39] Oracle & Microsoft, *AI in Databases: Technical Briefs and Innovations*, Whitepapers, 2023.

Aniela BORCAN graduated in 2023 with a Bachelor's degree in Economic Cybernetics from the Faculty of Cybernetics, Statistics and Economic Informatics at the Bucharest University of Economic Studies. She continued her studies at the same institution, earning a Master's degree in Databases – Support for Business in 2025. With a background in both IT and the financial industry, Aniela has contributed to the development of digital solutions aimed at enhancing business efficiency and enabling data-driven decision-making. She has practical experience in backend development and database management, with key interests in scalable data infrastructures, process automation and the integration of secure and efficient digital banking systems.

Diana CAPTARI completed her undergraduate studies in 2023, earning a degree in Economic Cybernetics from the Faculty of Cybernetics, Statistics and Economic Informatics at the Bucharest University of Economic Studies. She further pursued graduate studies at the same university, obtaining a Master's degree in Data Bases – Support for Business in 2025. Currently active in the banking industry, she works as a programmer, with hands-on experience in software development and database technologies tailored to financial services. Her professional interests focus on digital solutions in banking, optimization of data systems, and the integration of programming tools in business processes.

Emanuela-Cristina CARP graduated in 2023 with a Bachelor's degree in Economic Cybernetics from the Faculty of Cybernetics, Statistics and Economic Informatics of the Bucharest University of Economic Studies. She obtained her Master's degree in *Data Bases – Support for Business* from the same institution in 2025. She is currently employed in the private sector, with professional experience in databases, artificial intelligence, and software development. Her research interests include database systems, intelligent data processing, and the integration of AI technologies into modern business solutions.