SDLC-INDEPENDENT PYTHON-BASED QUERY PERFORMANCE BENCHMARKING APPROACH AND PRACTICAL OPTIMAL DATABASE SELECTION GUIDELINES

Katarina Milojković*,
[0000-0001-8658-8973]
Petar Spalević,
[0000-0002-6867-7259]
Nikola Vasić,
[0000-0003-2713-4589]
Nikola Milojković,
[0009-0001-0106-9662]
Hristina Milojković
[0000-0003-4150-3301]

Singidunum University,
Belgrade, Serbia

Abstract:
One of the critical decisions of software development teams in application development is choosing the most optimal database. Modern business conditions require software development teams to continually improve application performance, and a common dilemma is whether to transition from a relational database to a non-relational database or vice versa. Changes during the application development phase can slow down and complicate the process, so it is crucial to empirically approach data analysis and decision-making before implementing any changes. This research aims to facilitate the optimal database selection by following established practical guidelines for optimal database selection and implementing an SDLC-independent Python-based query performance benchmarking approach. This benchmarking approach is a crucial part of the optimal database selection process particularly useful in the early stages of development or when considering a migration to an existing project. The research methodology includes qualitative and quantitative methods: analytical-synthetic, experimental, comparative analysis, and hypothetical-deductive methods. The results of this research include the practical optimal database selection guidelines, the process of conducting the benchmark, and the utilization of both the guidelines and benchmark results for optimal database selection of an application where changing the initially selected MySQL database to MongoDB is being considered.

Keywords:
SQL vs NoSQL, MySQL vs MongoDB, Optimal Database Selection Guidelines, Query Performance Benchmark, Python.

INTRODUCTION

In modern business conditions, software development teams frequently encounter challenges such as difficult application development, data consistency and integrity, processing complex queries, real-time fraud detection, robustness, efficiency, scalability, agility, transaction consistency, complex relationships and transactions, unstable workloads, real-time analytics and updates, managing user-generated data, and many more. The main reasons for those challenges can be wrong initial database selection, requirements evolving over time, increased application popularity, unpredictable traffic spikes, large or constantly evolving data sets, and changed market conditions and consumer preferences. In order to resolve those challenges, it is vital to select the most optimal database for the application's architecture and requirements.

Correspondence:
Katarina Milojković

e-mail:
katarina.milojkovic.21@singimail.rs

DOI: 10.15308/Sinteza-2025-536-543

All of the above confirms the extreme importance of choosing the right database solution at every stage of the Software Development Life Cycle (SDLC). Developers can choose a relational (SQL) or a non-relational (NoSQL) database solution for the sake of establishing, maintaining, and improving performance, security, reliability, and user satisfaction. Changes during the application development phase can slow down and complicate the process, especially if that change is a wrong database selection because replacing it will be time-consuming and costly. In order to avoid that, it is crucial to have a deep understanding of SQL and NoSQL pros and cons, as well as empirically approach data analysis and decision-making before implementing any changes. Organizations conduct benchmarks using a framework or script created by a programming language such as Python in order to analyze the data flow and performance in real-time as well as simulate other database solutions working in an application and test their performance. The goal of this research is to share insights and knowledge regarding optimal database selection facilitation by following established practical guidelines for optimal database choice and implementing an SDLC-independent Python-based query performance benchmarking approach.

The hypothesis tested in this research states that MongoDB is the optimal database solution for the "BooksByHM" application. The content-sharing web application "BooksByHM" allows authors to publish chapters, books, images, and audio versions while enabling users to interact through likes, ratings, comments, and purchases. The application is currently in a mid-developing phase where changing the initially selected MySQL database to MongoDB is being considered.

In the 'Literature review' section, relevant research was summarized and presented to provide a deeper insight into this research. This section serves as a foundation for practical optimal database selection guidelines. The methodology used for this research is thoroughly explained in the 'Methodology' section, while the implementation of the Python-based query performance benchmarking approach is in the 'Implementation' section. The results of qualitative and quantitative methods used in this research and the discussion of those results are shown in the 'Results and Discussion' section, whereas the 'Conclusion' section presents the conclusions drawn from the research.

2. LITERATURE REVIEW

Database selection is one of the most crucial decisions developers must make to develop a system and maintain performance, reliability, and user satisfaction. [1] [2] Selecting the wrong database will make application development difficult because replacing it will be time-consuming and costly, so it must be done with careful consideration of long-term support and sustainability. [1] Before choosing, switching, or migrating databases, organizations conduct benchmarks using a framework or script created by a programming language such as Python in order to analyze the data flow and performance in real-time. [3] [4] [5] Benchmarking is the process of running a specific program or workload on a machine or system to evaluate its performance for that workload accurately. [6] The reason behind making database changes is to improve efficiency, maintainability, scalability, and security. [5] With the Python migration script, data from one database can be present in another database. [5] Python is a dynamically typed programming language frequently used for scientific research, web development, machine learning, artificial intelligence, and data analysis. [7] Python is usable in various fields because it has a powerful standard library and wide module support. [7] Working with MySQL and MongoDB is possible in Python if libraries such as mysql-connector-python and pymongo are installed and MySQL and MongoDB servers are connected. [8]

In order to choose the most optimal database, relational (SQL) or non-relational (NoSQL), several factors must be taken into account such as the data and database structure, schema flexibility, data scalability, query language for defining and manipulating the data, performance indicators, guaranteed properties (ACID, BASE, CAP), and cost. [9] Another important factor can be security, licensing, and its capability with different tools. [5] SQL offers extensive integration support, while NoSQL provides modern APIs and flexible data formats for seamless integration with microservices and cloud-native architectures. [2] These factors can be more harmful than helpful if not utilized efficiently within the software architecture [5] and requirements. [8]

Based on the data structure of stored data, database models are examined as relational databases (SQL databases), where structured data is stored in a predefined schema, and non-relational databases (NoSQL databases), where unstructured data is not stored in a predefined schema. [7] The advantages of storing data in a predefined schema are predicting entities and values the application expects, validating data based on exist-

ing or new records, using database constraints, and organizing data through normalization. [1] On the other hand, NoSQL schemas enhance flexibility and scalability, simplify Big Data management, and allow developers to focus on software application development, and database optimization. [9] SQL databases, as a tabular relational model, use tables where data is stored in the forms of rows (records) and columns (attributes). [10] [8] Popular SQL databases that have become industry standards are MySQL, PostgreSQL, Oracle Database, and Microsoft SQL Server. [11] NoSQL databases are non-tabular databases, which is why they use data models such as document-based, key-values, column-based, or graph-based. [3] Document-based databases store data in document forms inside a collection. [1] Since they do not implement field validations and constraint checks, they are swift. [1] Because the same entity with different fields can be stored multiple times, developers need to pay attention to the application side to avoid making mistakes. [1] Document-based databases have a multi-server architecture. [1] They are ideal for application development, flexible data structures, scalability, and dynamic schemas. [1] Popular document-based NoSQL solutions are MongoDB, CouchDB, CosmosDB, DynamoDB [1] [10], and Firebase. [12] MongoDB is an open-source NoSQL database that uses a document-oriented approach. [4] MongoDB uses XML, JavaScript Object Notation (JSON), or Binary JSON (BSON) to encode and store data in a document format. [10] Queries are performed over collections or using map-reduce. [4]

MySQL has a networked client-server architecture which has two components, MySQL server and client programs. [4] MySQL architecture is a web of task-related functions that work to finish the job of a database server. [4] Relational databases can have a multi-server architecture by using shared storage technology. [1] MySQL uses master-slave replication through a cluster-based architecture, whereas MongoDB uses master-slave replication through replica sets. [4]

Most relational databases are vertically scalable, meaning that the load on a server will be increased by increasing/upgrading the server's hardware components like RAM, HDD/SSD, and CPU. [1] Non-relational databases are horizontally scalable, meaning they can support increased traffic by adding servers and instances. [10] NoSQL databases can be more cost-effective than SQL databases due to horizontal scalability and open-source free solutions. [9] The scalability of MongoDB is easy to implement and performs better whereas MySQL maintains data integrity. [4] [13] [14] However, MySQL can improve scalability by using cloud-based technologies. [14]

SQL databases use Structured Query Language (SQL), a declarative and standardized query language for defining, manipulating, querying, and managing data. [9] [10] [11] NoSQL databases use Not Only Structured Query Language (NoSQL), a custom query language tailored to their specific data models. [3]

The performance evaluation (query runtime, memory used, CPU used, and storage size) of different SQL and NoSQL databases resulted in MongoDB outperforming almost all tests with a large data volume [13] [3] for example, 10,000+ records. [10] The result is due to the way data is stored, complex joins, and data normalization. [10] NoSQL select operations are 3 times faster, delete operations are 6 times faster, update operations are 9 times faster, and insert operations are 15 times faster than SQL. [10] MySQL shows better performance for small datasets (a few thousand records) and few database operations (a hundred operations daily). [15] SQL databases have a better join query performance [13] and use less CPU resources and memory usage for task completion compared to NoSQL databases. [15] However, the process of storing and retrieving complex data types (images as byte data) was faster inside NoSQL databases like MongoDB. [15] MongoDB outperformed MySQL in terms of Latency, Throughput, scalability, security, performance, and availability. [4]

SQL databases follow ACID (Atomic, Consistent, Isolated, and Durable) transaction principles [7] and CAP (Consistency, Availability, and Partition Tolerance) theorem. [11] Instead, NoSQL databases follow the BASE (Basically Available, Soft-State, and Eventual Consistency) transaction principles [7] and CAP (Consistency, Availability, and Partition Tolerance) theorem. [11] NoSQL databases often do not follow ACID principles, such as strong data consistency, which makes processing complex SQL queries challenging. [14] This can be overcome with automatic machine-learning classification techniques such as SVM, K-means, and NBC. [14] Distributed systems can only prioritize two out of three CAP principles. [11] SQL databases prioritize consistency and availability, whereas NoSQL databases prioritize tolerance and availability which makes them offer eventual consistency instead of strict consistency. [11] NoSQL databases lack full support for atomicity, consistency, isolation, and durability features found in SQL databases. [14] NoSQL databases sacrifice some robustness to achieve more speed and scalability. [1] Relational databases managed by RDBMS assure data integrity and transaction consistency. [9] In relational databases, the data storage performance degrades as the data volume increases. [9]

SQL databases are used for e-commerce [8], transaction applications, financial systems, enterprise resource planning (ERP) systems, customer relationship management (CRM) systems, applications that require complex queries [11], relationships and transactions, strong data integrity, inventory control, applications with stable workloads [2], applications with multi-row transactions [9] , payment processing, and core banking systems. [16]

NoSQL databases are used for big data analytics, recommendation systems, Internet of Things (IoT) [8], cloud-based applications [4], real-time analytics, content management systems, social networks [11], applications with large or constantly evolving data sets [9], applications focused on rapid data access, flexible data models, scalability, high traffic loads, unpredictable traffic spikes, managing user-generated data, real-time updates, product catalogs, and inventory data [2], agile applications where requirements evolve over time [17] [11], data mining applications [13], real-time fraud detection, and personalized finance services. [16]

A hybrid database approach combines the strengths of SQL and NoSQL systems allowing organizations to tailor their database solutions to their needs. [11] [8] [2] However, this approach requires careful planning and implementation because it can complicate data management and integration. [11] A hybrid solution (SQL for transaction processing and core banking systems, and NoSQL for big data analytics, real-time fraud detection, and customer management) was chosen for the FinTech application founded on large-scale data processing, transactional integrity, and real-time analytics, and warrants robust and highly scalable database solutions. [16] Netflix uses SQL for billing and subscriber data while using NoSQL for viewing history and recommendations. Similarly, Uber uses SQL for transactional accuracy in rides and payments, while NoSQL handles real-time tracking for high availability. [11] PayPal uses Apache Cassandra to power its real-time fraud detection systems. [16] JP Morgan Chase uses SQL databases in its core banking operations. [16] Countries with a higher number of train stations and stops such as Germany, Netherlands, and others, use non-relation databases. Whereas smaller size train stations in Slovakia use relation databases. [10] Google, Facebook, Twitter, and Amazon prefer NoSQL database systems because they have very large datasets, and they need to implement their solutions on multiple servers and NoSQL are horizontally scalable databases. [1] [4] [10]

3. METHODOLOGY

The research combines qualitative and quantitative methods such as analytical-synthetic, experimental, comparative analysis, and hypothetical-deductive methods. The analytical-synthetic method was used to conduct a comprehensive analysis of SQL and NoSQL databases and synthesize those findings in the form of practical guidelines for choosing the optimal database. The experimental method was used to conduct an experiment with the SDLC-independent Python-based query performance benchmarking approach to measure the query execution time of MySQL and MongoDB under the same conditions. The benchmark results were compared and analyzed using the comparative analysis method. A hypothetical-deductive method was used for testing the set hypothesis with the results of analytical-synthetic, experimental, and comparative analysis methods, and reaching deductions on optimal database choice for the "BooksByHM" application based on the obtained results.

4. IMPLEMENTATION

The process of conducting an SDLC-independent Python-based benchmark for measuring query execution time consists of database and data preparation, setting up benchmarking conditions, executing benchmarking queries, and measuring performance metrics.

The experiment is set under the same initial benchmarking conditions:

- Hardware and system specifications:
  - Processor: AMD Ryzen 7 3700U with Radeon Vega Mobile Gfx (8 CPUs), ~2.3GHz
  - RAM: 8.00 GB
  - Operating System: Windows 11 Pro
  - System type: 64-bit operating system, x64-based processor
- Software and database versions:
  - Python version: 3.13.2
  - PIP version: 24.3.1
  - MySQL version: 10.4.28-MariaDB
  - MongoDB version: 8.0.0
  - Python libraries: mysql-connector-python, pymongo, random, and time.

- Dataset description:
  - The smaller dataset contains 10 book records and 11 user records.
  - The bigger dataset contains 10.000 book records and 1.001 user records.
  - MySQL tables: books, users, user_books (many-to-many relationship)
  - MongoDB collections: books, users (embedded book list)
  - MySQL indexing: primary keys on id fields, foreign keys in user_books
  - MongoDB indexing: default index on _id.

The databases were prepared by duplicating the existing MySQL database used in the "BooksByHM" application and creating an equivalent MongoDB database. Working with MySQL and MongoDB databases in Python was made possible by installing mysql-connector-python and pymongo Python packages and connecting MySQL and MongoDB servers in the Python script. Installing the MySQL Connector package for Python on Windows using PIP was done with the command: *pip install mysql-connector-python*. On the other hand, the command used for installing the PyMongo package for Python on Windows using PIP was: *pip install pymongo*. Listing 1 shows the Python code used to establish the connection of specific MySQL and MongoDB databases.

```
mysql_conn = mysql.connector.connect(
    host="localhost",
    user="root",
    password="",
    database="books_by_hm"
)
mysql_cursor = mysql_conn.cursor()

mongo_client = MongoClient("mongodb://localhost:27017/")
mongo_db = mongo_client["books_by_hm"]
mongo_books = mongo_db["books"]
mongo_users = mongo_db["users"]
```

Listing 1. Python code is used to establish a connection between the MySQL database and the MongoDB database

```
def measure_mysql():
    start = time.time()
    mysql_cursor.execute("SELECT * FROM books WHERE name='Book5000'")
    mysql_cursor.fetchall()
    end = time.time()
    print(f"MySQL – simple query: {end - start:.6f} seconds")

    start = time.time()
    mysql_cursor.execute("""
        SELECT users.*
        FROM users
        JOIN user_books ON users.id = user_books.user_id
        JOIN books ON books.id = user_books.book_id
        WHERE books.name = 'Book5000'
    """)
    mysql_cursor.fetchall()
    end = time.time()
    print(f"MySQL – complex query: {end - start:.6f} seconds")

def measure_mongodb():
    start = time.time()
    mongo_books.find({"name": "Book5000"})
    end = time.time()
    print(f"MongoDB – simple query: {end - start:.6f} seconds")

    start = time.time()
    mongo_users.find({"books": {"$elemMatch": {"name": "Book5000"}}})
    end = time.time()
    print(f"MongoDB - complex query: {end - start:.6f} seconds")
```

Listing 2. measure_mysql() and measure_mongodb() functions written in Python

Tables and collections were created in the Python script. Also, the data used for the benchmark was prepared by automatically inserting the small and large datasets in the Python script.

Based on the application's needs, the chosen simple benchmarking query was finding a book by title, and the complex benchmarking query was finding users who own a specific book. Execution time in seconds was measured using Python's time.time() function before and after query execution. The scalability tests were performed by scaling datasets from 10 to 10,000 books and 11 to 1,001 users to analyze the performance fluctuation. Listing 2 shows measure_mysql() and measure_mongodb() functions written in Python used to execute queries and measure execution time.

5. RESULTS AND DISCUSSION

The measured query execution time of MySQL and MongoDB with different dataset sizes are shown in Table 1.

As shown in Table 1, MySQL simple query execution time is 11.9864 times faster for smaller datasets and 3.2198 times faster for bigger datasets than MongoDB. The result is due to the way data is stored, database constraints, and data normalization. MySQL is faster than MongoDB because it uses an index and searches directly on the table. As the data volumes increase, query execution time increases for both databases with MySQL remaining to be faster. This indicates that indexed search in MySQL scales is better for simple queries than in MongoDB.

Table 1. Benchmark results

	MySQL		**MongoDB**	
Dataset size	Smaller dataset	Bigger dataset	Smaller dataset	Bigger dataset
Simple query execution time (seconds)	0.000516	0.003316	0.006185	0.010677
Complex query execution time (seconds)	0.003165	0.011179	0.000073	0.000045

Table 2. Practical guidelines for optimal database selection

SQL database characteristics	tabular data models for storing structured data in a fixed schema; vertical scalability (increasing/upgrading the server's hardware components); scalability can be improved by using cloud-based technologies; client-server architecture; can have a multi-server architecture by using shared storage technology; using Structured Query Language (SQL) for defining, manipulating, querying, and managing data; better performance for small/limited datasets and few database operations; better join query performance; less CPU resources and memory usage for task completion; master-slave replication through a cluster-based architecture; ACID database transaction model; follows CAP theorem (prioritizing consistency and availability); strong data consistency; assure data integrity and transaction consistency; the data storage performance degrades as the data volume increases; offers extensive integration support;
NoSQL databases characteristics	non-tabular data models such as document-based, key-values, column-based, or graph-based for storing unstructured data in a flexible schema; cost-effective due to horizontally scalable (increasing/upgrading servers and instances) and open-source free solutions; scalability is easy to implement; multi-server architecture; using Not Only Structured Query Language (NoSQL) tailored to their specific data models for defining, manipulating, querying, and managing data; better performance for large datasets; faster storing and retrieving complex data types (images as byte data); master-slave replication through replica sets; BASE database transaction model; follows CAP theorem (prioritizing tolerance and availability); eventual data consistency; lack full support for ACID features; less robustness; high-velocity and scalability; provides modern APIs and flexible data formats for seamless integration with microservices and cloud-native architectures;
SQL is suitable for	e-commerce applications; transaction applications; financial systems; enterprise resource planning (ERP) systems; customer relationship management (CRM) systems; applications that require complex queries, relationships, and transactions; applications that require strong data integrity; inventory control; applications with stable workloads; applications with multi-row transactions; payment processing; core banking systems;
NoSQL is suitable for	big data analytics; recommendation systems; Internet of Things (IoT); cloud-based applications; real-time analytics; content management systems; social networks; applications with large or constantly evolving data sets; applications focused on rapid data access, flexible data models, and scalability; high traffic loads; unpredictable traffic spikes; managing user-generated data, real-time updates, product catalogs, and inventory data; agile applications where requirements evolve over time; data mining applications; real-time fraud detection; personalized finance services;

As shown in Table 1, MongoDB complex query execution time is 43.3562 times faster for smaller datasets and 248.4222 times faster for bigger datasets than MySQL. The result is due to the way data is stored, complex join, and data normalization. MongoDB is significantly faster than MySQL because the search is performed directly, while MySQL requires joining data from multiple tables and filtering the results. When the database is large, MongoDB remains consistently fast (even faster than with a smaller database), while MySQL's execution time increases significantly. This suggests that MongoDB is better for complex queries as it does not need expensive JOIN operations like MySQL. As data grows, MySQL's complex queries become significantly slower, while MongoDB remains efficient.

Utilizing both the guidelines, shown in Table 2, and benchmark results, shown in Table 1, the most optimal database solution for the 'BooksByHM' application is a hybrid database approach that combines the strengths of MySQL and MongoDB. For optimal performance, MySQL would be used for structured data (users, purchases, and ratings), transaction processing, core banking operations, and dynamic content generation (SEO optimization). On the other hand, MongoDB would be used for content storage and interactions (books, comments, and likes), real-time analytics, real-time fraud detection, customer management, large-scale data processing, personalized recommendations, storing and retrieving images (as binary data) and audiobook files, searching books, managing nested comments, and generating statistical insights.

6. CONCLUSION

The research contributes by presenting practical guidelines for optimal database selection and an SDLC-independent Python-based query performance benchmark. The hypothesis was successfully tested with both the guidelines and benchmark results. The hybrid approach can utilize the strengths of both MySQL and MongoDB by allowing tailored database solutions to the "BooksByHM" application's needs. The benchmarking approach highlighted in this research can be used to analyze the data flow and performance in real-time, as well as simulate database solutions and test their performance without needing to have a concrete application. This is the reason why this benchmarking approach is SDLC-independent. Following the established practical guidelines for optimal database selection and implementing the SDLC-independent Python-based query performance benchmarking approach is particularly useful in the early stages of development or when considering a migration to an existing project. The research highlights the importance of selecting the right database solution and demonstrates that the benchmarking process can be both simple and efficient using Python. It aims to encourage and motivate developers to experiment with and test different database solutions. Additionally, it serves as an accessible and engaging experiment that anyone can try in their spare time. However, this study represents just the tip of the iceberg in the field of benchmarking. Future directions of this research could explore additional benchmarking methods using Python to further enhance database performance evaluation.

REFERENCES

[1] H. Paci, "SQL vs NoSQL databases from developer point of view," *Industry 4.0,* vol. VII, no. 3, pp. 95-97, 2022.

[2] T. Ramzan and G. Alwin, "Comparative Study of SQL vs. NoSQL for High-Performance E-commerce Databases," pp. 1-18, 2023.

[3] M. Z. Khan, F. U. Zaman, M. Adnan, A. Imroz, M. A. Rauf and Z. Phul, "Comparative case study: An evaluation of performance computation between SQL and NoSQL database," *Journal of Software Engineering*, vol. I, no. 2, pp. 14-23, 2023.

[4] R. Pandey, "Performance benchmarking and comparison of cloud-based databases MongoDB (NoSQL) vs MySQL (Relational) using YCSB," *Nat. College Ireland, Dublin, Ireland, Tech. Rep*, 2020.

[5] D. Liberman, "Migration from NoSQL to SQL," 2023.

[6] R. H. Saavedra and A. J. Smith, "Analysis of benchmark characteristics and benchmark performance prediction," *ACM Transactions on Computer Systems (TOCS)*, vol. XIV, no. 4, pp. 344-384, 1996.

[7] M. Yeşilyurt and Y. Z. Ayik, "Comparison of C# and Python programming languages in terms of performance and coding on SQL server DML operations," *NanoEra*, vol. IV, no. 1, pp. 23-33, 2024.

[8] Є. С. Тимошенко, "Робота з базами даних в python: SQL та NOSQL," 2024.

[9] A.-G. Babucea, "SQL OR NoSQL DATABASES? CRITICAL DIFFERENCES," *Annals of Constantin Brancusi' University of Targu-Jiu. Economy Series/ Analele Universităţii' Constantin Brâncuşi' din Târgu-Jiu Seria Economie*, no. 1, 2021.

[10] R. Čerešňák and M. Kvet, "Comparison of query performance in relational a non-relation databases," *Transportation Research Procedia*, vol. XL, pp. 170-177, 2019.

[11] Y. Jani, "The role of sql and nosql databases in modern data architectures," *International Journal of Core Engineering & Management*, vol. VI, no. 12, pp. 61-67, 2021.

[12] K. Milojković, M. Živković and N. B. Džakula, "Agile Multi-user Android Application Development With Firebase: Authentication, Authorization, and Profile Management," *Sinteza 2024-International Scientific Conference on Information Technology, Computer Science, and Data Science*. Singidunum University, pp. 405-412, 2024.

[13] J. Antas, R. R. Silva and J. Bernardino, "Assessment of SQL and NoSQL systems to store and mine COVID-19 data," *Computers*, vol. XI, no. 2, p. 29, 2022.

[14] R. A. Kadir, E. S. M. Surin and M. R. Sarker, "A Systematic Review of Automated Classification for Simple and Complex Query SQL on NoSQL Database," *Computer Systems Science & Engineerin*g, vol. XLVIII, no. 6, 2024.

[15] S. A. FADHEL and E. A. JAMEEL, "A Comparison between NoSql and RDBMS: Storage and Retrieval," *International Journal of Applied Sciences and Technology,* vol. IV, no. 3, pp. 173-184, 2022.

[16] P. Gowda and A. N. Gowda, "SQL vs. NoSQL databases: Choosing the right option for FinTech," *Journal of Scientific and Engineering Research*, vol. VII, no. 8, pp. 100-104, 2020.

[17] D. Milojković and K. Milojković, "Improving the Business Resilience of an Organization by Applying Agile Project Managemet Approach," *FINIZ 2022-Business Resilience in a Changing World*, pp. 98-103, 2022.