2023 Second International Conference On Smart Technologies For Smart Nation (SmartTechCon) | 979-8-3503-0541-8/23/$31.00 ©2023 IEEE | DOI: 10.1109/SmartTechCon57526.2023.10391694

Scalability and Performance Evaluation of Graph Database Systems: A Comparative Study of Neo4j, JanusGraph, Memgraph, NebulaGraph, and TigerGraph

André Lopes, Diogo Rodrigues, João Saraiva
Department of Informatics
Polytechnic of Viseu
Viseu, Portugal
pv26447@alunos.estgv.ipv.pt,
pv26448@alunos.estgv.ipv.pt,
pv26721@alunos.estgv.ipv.pt

Maryam Abbasi
Applied Research Institute
Polytechnic of Coimbra
Coimbra, Portugal
maryam@dei.uc.pt

Pedro Martins, Cristina Wanzeller
Research Centre in Digital Services
Polytechnic of Viseu
Viseu, Portugal
pedromom@estgv.ipv.pt,
cwanzeller@estgv.ipv.pt

***Abstract*—This paper presents a comprehensive analysis and evaluation of the performance and scalability characteristics of graph databases. The study focuses on the leading graph databases, namely Neo4j, JanusGraph, MemGraph, NebulaGraph, and TigerGraph, as ranked by DB-Engines. In the face of the ever-increasing daily data volumes, traditional database management systems (DBMS) are struggling to cope with the growth, necessitating the exploration of alternative solutions. Graph databases, along with other types such as document and key-value databases, have emerged as promising alternatives that may offer more efficient handling of data-related challenges faced by DBMS.**

The objective of this research is to evaluate and compare the performance and scalability of these graph databases using various metrics, including query response time, data loading time, and memory usage. Through our experimentation, we have obtained compelling results. Among the top-five graph databases considered, Neo4j has exhibited superior performance, especially when handling larger datasets. It has consistently outperformed the other engines, showcasing its potential as a robust and efficient solution for managing graph-based data.

***Index Terms*—Graph databases, Performance evaluation, Scalability analysis, Database management systems (DBMS), Neo4j, Data handling efficiency**

I. Introduction

Graph databases have gained significant attention in recent years due to the increasing complexity and volume of data in modern applications. Traditional relational databases struggle to efficiently handle interconnected data, making graph databases a promising solution that leverages the inherent structure and relationships of data entities [1]. This paper focuses on analyzing and evaluating the performance and scalability characteristics of graph databases to address the challenges associated with managing and querying highly connected data.

Previous studies have explored the advantages of graph databases in various fields such as social networks, recommendation systems, and fraud detection [2]. For example, research has shown that graph databases outperform standard database systems in traversing complex relationships and answering graph-based queries [3]. However, there is a lack of comprehensive comparison studies that systematically evaluate the efficiency and scalability of different graph database technologies.

This research aims to fill this gap by conducting an in-depth analysis and comparison of multiple graph database technologies. The performance evaluation will consider metrics such as query response time, data loading time, and memory usage. The scalability assessment will focus on measuring the database's ability to handle increasing data size.

We will analyze the performance and scalability characteristics of the top 5 graph database systems as of May 2023, according to the rankings by DB-Engines. This includes Neo4j, JanusGraph, Memgraph, NebulaGraph, and TigerGraph. Evaluating these leading graph database systems will provide insights into their performance and scalability characteristics.

This work contributes significantly by offering valuable perspectives on the strengths and limitations of the top five NoSQL graph databases based on their rankings in DB-Engines. Furthermore, the study includes a thorough experimental evaluation of these graph databases using an established benchmark, aiming to select the best NoSQL graph database based on various performance measures such as data loading time, query execution times, and RAM and CPU memory usage.

This paper is organized into five sections. Section II provides a review of related work in the field of graph databases. Section III discusses the architecture of the graph database systems under evaluation. Section IV describes the experimental setup and methodology. Section V presents the results of the performance evaluation. Finally, Section VI concludes the paper and discusses future work.

II. Related work

In recent years, the use of graph databases has significantly increased due to their efficient handling of data with complex interconnections. Researchers have conducted various studies and comparisons to explore the performance, scalability, and suitability of different graph database systems. This section provides an overview of the existing literature and related work in the field of graph databases, including indexing techniques, performance evaluation methodologies, query languages, and applications across different domains.

Wang et al. conducted a comparative empirical investigation of the effectiveness of modern graph database systems, such as Neo4j, OrientDB, Titan, and Virtuoso [4]. Their study assessed the usefulness, performance, and scalability of these systems. By simulating real-world applications using benchmark datasets and workloads, the authors found that each system had its own advantages and disadvantages. They emphasized the importance of considering the specific demands and requirements of the application when selecting a graph database system.

Macak et al. performed a benchmarking study comparing the performance of OrientDB, Neo4j, and MongoDB [5]. They focused on query execution time and data loading speed to compare the multimodel capabilities of OrientDB with its single-model counterparts, Neo4j and MongoDB. Their findings provided insights into the performance characteristics of these databases and their effective usage for different purposes.

Timón-Reina et al. provided an overview of graph databases and their applications in the biomedical field [6]. They discussed the advantages of graph databases in managing complex biological and biomedical data, such as protein-protein interactions, gene regulatory networks, and disease networks. The authors emphasized the importance of graph databases in facilitating efficient data integration, querying, and visualization for biomedical research and healthcare applications.

Erdemir et al. compared the querying capabilities of Neo4j for graph and hyper-graph models [7]. They evaluated the response times of various query types on both models and analyzed the findings. Their study provided insights for researchers dealing with complex interconnected data by highlighting the advantages and disadvantages of the graph and hyper-graph models in terms of query efficiency.

These studies contribute to the field of graph databases, indexing methods, system performance, and query languages. They offer useful insights for researchers and developers seeking to optimize and effectively utilize graph database systems.

III. Architecture

Graph databases have emerged as a powerful solution for efficiently managing highly interconnected data in various domains, including social networks, network and asset monitoring, fraud detection, and knowledge graphs [8]. Unlike traditional relational databases optimized for tabular data, graph databases are specifically designed to handle and navigate through interconnected structures resembling graphs. They excel in representing complex relationships between entities and enable powerful queries based on graph structures [9].

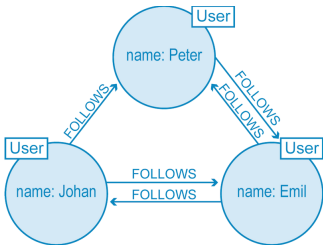

Fig. 1: Example of a graph database model [11]

In a graph database, each node represents an entity or object, such as a person, category, or data point. Nodes can possess properties that provide additional information about the represented entity. Edges, also referred to as relationships or connections, establish associations between nodes and depict interactions or dependencies between entities [10].

Figure 1 illustrates a simplified example of a graph database representing a subset of Twitter users. The nodes in the graph represent individual users, and the connections between them depict their relationships. For instance, Peter and Emil are mutual followers, and Emil and Johan are connected as well. However, Johan follows Peter, while Peter has not reciprocated the follow yet.

This graph database offers a succinct visual depiction of the social connections within this group of Twitter users. By utilizing nodes to represent users and edges to denote relationships, the graph provides an intuitive understanding of the network's structure. Graph databases enable the exploration of various aspects of social networks, such as identifying influential users, detecting communities, studying information diffusion, and predicting user behavior [11].

The key advantages of graph databases include:

1) **Relationship focus**: They excel in representing and querying relationships between data points. Traversing and retrieving connected data is efficient and flexible.
2) **Flexibility**: They are schema-less or schema-flexible, allowing the data structure to evolve and change over time without significant schema modifications. This suits dynamic data scenarios and complex, evolving relationships between entities.
3) **Performance**: Graph databases offer high performance for queries involving relationship traversal. Index structures and optimized graph algorithms enable fast query execution, even with large and interconnected datasets.
4) **Analytical capabilities**: They support complex analytical queries, enabling tasks such as pattern identification, cluster detection, and path finding between nodes.
5) **Scalability**: Graph databases are designed for horizontal scalability, allowing them to handle growing datasets by distributing the load across additional servers. This enables efficient management of large data volumes and high concurrency scenarios.

Below, we present an overview of the key features of the top 5 graph database systems for study and analysis according to DB-Engines: Neo4j, JanusGraph, Memgraph, NebulaGraph, and TigerGraph. Each of these databases has its own unique characteristics and features, making them suitable for different use cases and scalability requirements.

Neo4j is a widely used graph database with horizontal and vertical scalability. It implements the property graph data model and offers a robust set of indexing features. Neo4j's Cypher query language enables expressive interaction with graph data, supported by an ecosystem of tools and libraries. However, it does not directly handle RDF-formatted data and consumes substantial memory.

JanusGraph is an open-source graph database known for its scalability and flexibility. It employs a distributed, master-slave architecture and supports multiple storage backends. The Gremlin query language allows for complex graph analytics tasks. However, JanusGraph relies on third-party storage and indexing backends, which introduces external dependencies and makes future development prediction challenging.

Memgraph is a high-performance, in-memory graph database that combines transaction management, query execution, and indexing. It excels at real-time graph queries and high-throughput transactional workloads. Memgraph supports the Cypher query language and ensures low latency and high throughput for real-time insights. However, its reliance on in-memory storage may limit its ability to handle extremely large datasets.

NebulaGraph is a distributed graph database designed for large-scale applications. It follows a master-slave architecture and provides horizontal scalability. The nGQL query language integrates graph pattern matching, aggregation, and traversal operations. NebulaGraph's disk space consumption and limitations on cross-cluster backup restoration should be considered.

TigerGraph is a parallel graph database optimized for high-performance graph analytics. It employs a native parallel graph concept and a distributed design. The GSQL query language supports complex graph algorithms and iterative computation models. TigerGraph has platform limitations, as it only runs on Linux servers and has an expensive cloud service.

These graph databases offer diverse features and capabilities, catering to different use cases and requirements. Understanding their strengths and limitations can guide the selection of the most suitable database for specific applications.

TABLE I: Overview of the Graph Databases

Database	Architecture	Query Language	Key Feature
Neo4j	Layered design	Cypher	High Performance
JanusGraph	Distributed	Gremlin	Fault Tolerance
Memgraph	In-memory	Cypher	Real-time Analytics
NebulaGraph	Master-slave	nGQL	High Scalability
TigerGraph	Parallel graph	GSQL	Parallel Processing

IV. Experimental setup

After reviewing various graph databases and their characteristics, we conducted a comprehensive evaluation of their performance and scalability. To facilitate this evaluation, we utilized the Linked Data Benchmark Council's Social Network Benchmark (LDBC SNB) dataset [12]. This dataset is widely recognized as a benchmark for graph databases and is designed to simulate real-world social network scenarios, capturing the intricacies of highly interconnected data.

The LDBC SNB dataset includes entities such as users, posts, comments, as well as relationships like friendships and likes. It provides a valuable graph schema that represents a social network, encompassing a wide range of graph features and properties commonly encountered in real-world settings. This comprehensive dataset enables a realistic assessment of graph database systems. It is available in different scale factors, ranging from 0.1 to 1000, allowing evaluation across varying dataset sizes with increasing numbers of nodes and relationships. By evaluating the graph databases at multiple scale factors, we gained insights into their performance and scalability across diverse data sizes and workloads.

The LDBC SNB dataset is structured into two main components: static and dynamic. The static component represents the fundamental elements of the social network that remain constant across different scale factors [13]. It encompasses entities like tags and countries, along with their corresponding relationships. On the other hand, the dynamic component represents user-generated content and interactions within the social network. It includes entities such as persons and comments, as well as their relationships. As the scale factors increase, the dynamic component of the schema expands in terms of the number of nodes and edges [13].

Our evaluation focuses on various performance metrics, including query response time, memory usage, and scalability. By measuring these metrics across different scale factors and query workloads, we can gain valuable insights into the efficiency, effectiveness, and capacity of the database systems in handling growing data sizes.

In the subsequent subsections, we provide a detailed description of the experimental setup, including hardware and software configurations. We outline the process of loading the LDBC SNB dataset at various scale factors, executing the benchmark queries, and establishing specific performance goals and requirements for each graph database system.

A. Hardware and Software

The experimental evaluations were conducted on a test environment consisting of an Apple laptop powered by an Apple M1 CPU featuring 8 cores, 16 GB of RAM, and a 512 GB SSD disk. The operating system utilized for the experiments was macOS Ventura 13.4. This hardware setup

TABLE II: Database Scale Factors

	SF 0.1	SF 0.3	SF 1	SF 3	SF 10
Dataset size	108MB	336MB	1.21GB	3.66GB	10.4GB
Total nodes	416.3K	1.1M	3.9M	11.4M	36M
Total edges	2.0M	6.0M	23.0M	69.4M	231M

provided an appropriate platform for executing the graph database systems and performing the benchmark queries.

The following versions of the graph database systems were employed in the experiments: JanusGraph 0.6.3, Nebula Graph 3.4.0, Neo4j 4.4.17, TigerGraph 3.9.0, and Memgraph 2.7.0. These specific versions were selected based on their stability and compatibility with the experimental setup. Each graph database system was installed and configured in accordance with its respective documentation.

B. Procedure

The experimental procedure involved several steps, including dataset loading, query execution, and performance measurement.

First, the LDBC SNB dataset was loaded into each graph database system at the specified scale factors: 0.1, 0.3, 1, 3, and 10. Due to hardware limitations, these scale factors were chosen to ensure that the experiments could be executed successfully on the given hardware configuration. The dataset loading process involved importing the dataset files into the respective graph databases, configuring the schema, and optimizing the database settings for performance.

Table II presents the scale factors and corresponding dataset characteristics for the evaluated graph databases. The dataset size ranges from 108MB for SF 0.1 to 10.4GB for SF 10, reflecting the increasing data volume. The total number of nodes and edges also grows with the scale factor, indicating the expanding complexity of the datasets.

After the dataset was successfully loaded, the 29 benchmark queries provided by the LDBC SNB benchmark were executed for each graph database system. The SBN consists of 3 types of queries:

1) **Complex read-only:** 14 queries that retrieve information about the social environment of a specific user, including groups joined by friends and recent hashtags used.
2) **Simple read-only:** 7 queries that involve looking up information. Profile view queries retrieve information about a user, including their name, city, age, and a list of up to 20 friends and their posts.
3) **Transactional update:** 8 queries that involve inserting new data into the SNB dataset, such as creating user accounts, adding friendships, forums,

These queries covered a variety of graph traversal, pattern matching, and aggregation tasks, allowing for a comprehensive evaluation of the database systems performance. The specifications of the benchmark queries can be found in the LDBC SNB benchmark documentation [13].

The evaluation was performed for each scale factor, starting from the smallest (0.1) and gradually increasing to the largest (10). This approach allowed us to observe the performance and scalability characteristics of the graph database systems as the dataset size and workload complexity increased.

During the experimental procedure, the hardware, and software environment remained consistent to ensure fair comparison and prevent interference between different runs.

V. Results and analysis

The results of our performance evaluation of graph database systems at various scale factors are presented in this section. We analyse each database system performance and scalability by looking at different metrics such as data loading time, query response time, and memory consumption.

Table III presents the loading times for populating each graph database with the initial set of nodes at different scale factors. Neo4j shows the fastest loading time, ranging from approximately 1 minutes and 21 seconds to 48 minutes and 44 seconds. TigerGraph also demonstrates competitive loading times, ranging from 1 minutes and 30 seconds to 55 minutes and 48 seconds. JanusGraph, NebulaGraph, and Memgraph present slightly higher loading times.

A. Query Response Time

Table IV shows the query execution times for different databases at a scale factor of 0.1. The response times for complex read range from 0.23s to 18.93s, for short read range from 7,48s to 17,96s, and for updates range from 0,13s to 1,48s. Among these databases, Neo4j consistently outperforms the others in terms of response times. It has the lowest response times for complex read, short read, and updates among all the databases. Short queries take longer time to complete since they involve looking through numerous nodes with different characteristics.

Table V shows the query execution times for different databases at a scale factor of 10. The response times for complex read range from 18.37s to 49.51s, for short read range from 54.14s to 293.96s, and for updates range from 1.03s to 46.18s. Neo4j consistently exceeds the other databases in terms of response times at scale factor 10, just as it does at scale factor 0.1. It has the fastest response times for the three different query types. Response times have increased significantly as compared to the scale factor of 0.1. This is to be expected given that the dataset size has grown, resulting in higher query execution times. The longer execution times are most noticeable for short read queries, since they cover a larger number of nodes with diverse characteristics.

B. Memory Usage

Figure 2 presents the CPU usage (%) for each graph database at different scale factors. Neo4j consistently demonstrates lower CPU usage across all scale factors, with an average of 25%. JanusGraph and TigerGraph exhibit moderate CPU usage, with an average of 38% and 35% respectively. NebulaGraph shows higher CPU usage, especially at larger

TABLE III: Data Loading Time (hh:mm:ss) for Graph Databases

Database	SF 0.1	SF 0.3	SF 1	SF 3	SF 10	Total
JanusGraph	00:03:06	00:05:02	00:11:16	00:19:48	01:03:18	01:42:30
NebulaGraph	00:02:24	00:04:54	00:12:36	00:22:54	01:18:54	02:01:42
Neo4j	00:01:21	00:03:12	00:09:06	00:17:30	00:48:44	01:20:53
TigerGraph	00:01:30	00:03:36	00:10:12	00:18:06	00:55:48	01:29:12
MemGraph	00:01:52	00:04:06	00:11:36	00:19:18	00:56:42	01:33:34

TABLE IV: Query Response Times (Scale Factor: 0.1)

Database	Complex Read Times			Short Read Times			Update Times		
	Lowest	Highest	Average	Lowest	Highest	Average	Lowest	Highest	Average
Neo4j	0,23s (Q1)	13,12s (Q13)	3,43s	7,48s (Q19)	14,39s (Q16)	12,27s	0,13s (Q22)	0,94s (Q26)	0,38s
JanusGraph	0,42s (Q2)	16,35s (Q13)	4,01s	9,26s (Q19)	15,25s (Q16)	14,76s	0,15s (Q22)	1,09s (Q26)	0,42s
MemGraph	0,39s (Q1)	14,48s (Q13)	3,92s	7,93s (Q19)	15,09s (Q21)	14,09s	0,18s (Q22)	1,07s (Q26)	0,46s
NebulaGraph	0,48s (Q1)	18,93s (Q13)	4,73s	11,86s (Q19)	17,96s (Q21)	16,67s	0,32s (Q22)	1,48s (Q26)	0,71s
TigerGraph	0,38s (Q2)	13,96s (Q13)	3,87s	7,84s (Q19)	16,18s (Q16)	14,01s	0,17s (Q22)	0,98s (Q26)	0,39s

TABLE V: Query Response Times (Scale Factor: 10)

Database	Complex Read Times			Short Read Times			Update Times		
	Lowest	Highest	Average	Lowest	Highest	Average	Lowest	Highest	Average
Neo4j	18.37s (Q1)	40.14s (Q13)	28.63s	54.14s (Q19)	220.18s (Q21)	128.08s	1.03s (Q22)	37.30s (Q25)	3.19s
JanusGraph	22.52s (Q1)	44.77s (Q11)	33.09s	147.16s (Q18)	274.15s (Q21)	195.36s	1.95s (Q22)	43.76s (Q25)	6.32s
Memgraph	20.89s (Q1)	43.68s (Q13)	32.12s	128.93s (Q19)	251.19s (Q21)	191.39s	1.68s (Q22)	43.03s (Q25)	6.16s
NebulaGraph	24.68s (Q1)	49.51s (Q11)	38.13s	221.16s (Q19)	293.96s (Q15)	278.37s	2.36s (Q22)	46.18s (Q25)	8.92s
TigerGraph	19.93s (Q1)	41.32s (Q13)	30.17s	119.34s (Q19)	249.28s (Q21)	170.21s	1.67s (Q22)	42.28s (Q25)	5.19s

scale factors, reaching up to 76% at SF = 10. Memgraph maintains competitive CPU usage throughout, with an average of 24%. The results highlight the varying efficiency of different graph databases in terms of CPU utilization, with Neo4j and Memgraph showcasing better CPU utilization compared to the other databases.

Figure 3 presents the RAM consumption (%) of different graph databases, with a total of 16 GB of RAM available. JanusGraph exhibits the highest RAM usage across all scale factors, with an average consumption of 55%. Neo4j and TigerGraph show comparable RAM usage, with averages of 43% and 54%, respectively. Memgraph and NebulaGraph maintain moderate RAM consumption, with average values of 49% and 60%, respectively. JanusGraph shows the highest RAM usage, while Neo4j performs the most efficiently in terms of RAM utilization.

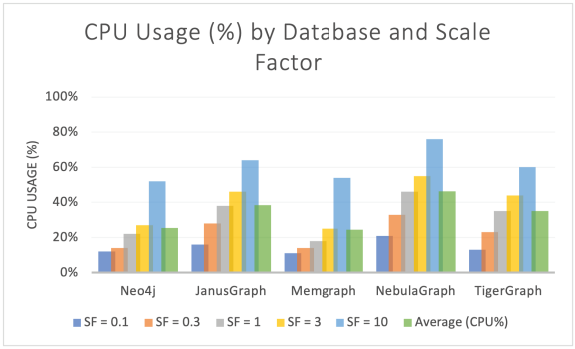

Fig. 2: Comparison of CPU Usage (%)

VI. Conclusions

This study involved the evaluation of five graph database systems: Neo4j, JanusGraph, Memgraph, NebulaGraph, and TigerGraph, with a specific focus on scalability and per-

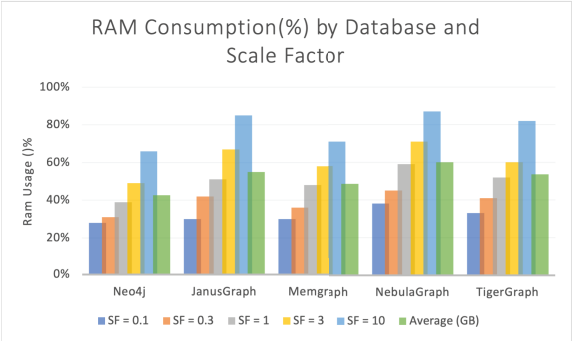

Fig. 3: Comparison of Ram Usage (%)

formance. Our objective was to analyze and assess these databases based on their capacity to handle large data volumes and deliver efficient query execution.

Among the graph databases evaluated, Neo4j emerged as the top-performing system across various metrics, demonstrating exceptional scalability. It exhibited superior performance in terms of node load times and query execution, outperforming the other databases in our experiments. TigerGraph also showcased strong performance and scalability, positioning it as a reliable option for applications that require efficient data access and processing. While Memgraph offered similar features to Neo4j, it exhibited slower performance when dealing with larger datasets, similar to JanusGraph. NebulaGraph showed relatively weaker performance and scalability compared to the other systems, particularly when dealing with larger datasets.

Acknowledgements

"This work is funded by National Funds through the FCT – Foundation for Science and Technology, I.P., within the scope of the project Ref. UIDB/05583/2020. Furthermore, we would like to thank the Research centre in Digital Services (CISeD) and the Instituto Politécnico de Viseu for their support."
Maryam Abbasi thanks the National funding by FCT - Foundation for Science and Technology, P.I., through the institutional scientific employment program-contract (CEECINST/00077/2021)

References

[1] Brandon T. Barclay. *Graph Databases Will Replace Relational Databases: Why Laggards Are Trailing Behind*. 2023. URL: https://www.linkedin.com/pulse/graph-databases-replace-relational-why-laggards-trailing-barclay.

[2] Min. Wu. *What is a graph database and what are the benefits of graph databases*. 2023. URL: https://www.nebula-graph.io/posts/what-is-a-graph-database.

[3] Petri Kotiranta, Marko Junkkari, and Jyrki Nummenmaa. "Performance of graph and relational databases in complex queries". In: *Applied Sciences* 12.13 (2022), p. 6490.

[4] Ran Wang et al. "An empirical study on recent graph database systems". In: *Knowledge Science, Engineering and Management: 13th International Conference, KSEM 2020, Hangzhou, China, August 28–30, 2020, Proceedings, Part I 13*. Springer. 2020, pp. 328–340.

[5] Martin Macak et al. "How well a multi-model database performs against its single-model variants: Benchmarking OrientDB with Neo4j and MongoDB". In: *2020 15th Conference on Computer Science and Information Systems (FedCSIS)*. IEEE. 2020, pp. 463–470.

[6] Santiago Timón-Reina, Mariano Rincón, and Rafael Martınez-Tomás. "An overview of graph databases and their applications in the biomedical domain". In: *Database* 2021 (2021).

[7] Mert Erdemir et al. "Comparison of Querying Performance of Neo4j on Graph and Hyper-graph Data Model." In: *KDIR*. 2019, pp. 397–404.

[8] Andrew Terrel. *Graph Databases: Seeking Structure in Data*. 2021. URL: https://www.ardentmc.com/2021/11/04/graph-databases-seeking-structure-in-data/.

[9] Alex Williams. *NoSQL database types explained: Graph*. 2021. URL: https://www.techtarget.com/searchdatamanagement/tip/NoSQL-database-types-explained-Graph.

[10] Neo4j Documentation. *Graph database concepts*. 2023. URL: https://neo4j.com/docs/getting-started/appendix/graphdb-concepts/.

[11] Neo4j. *Graph Databases for Beginners: Why Graph Technology Is the Future*. 2019. URL: https://neo4j.com/blog/why-graph-databases-are-the-future/.

[12] LDBC. *LDBC Graph Schema*. 2018. URL: http://ldbcouncil.org/benchmarks/snb/.

[13] LDBC. *The LDBC Social Network Benchmark Specifications*. 2018. URL: https://ldbcouncil.org/ldbc_snb_docs/ldbc-snb-specification.pdf.