Graph Databases: An Alternative to Relational Databases in an Interconnected Big Data Environment

Robert Pavliš
University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
robert.pavlis@fer.hr

***Abstract* — As the global volume of data continues to rise at an unprecedented rate, the challenges of storing and analyzing data are becoming more and more highlighted. This is especially apparent when the data are heavily interconnected. The traditional methods of storing and analyzing data such as relational databases often encounter difficulties when dealing with large amounts of data and this is even more pronounced when the data exhibits intricate interconnections. This paper examines graph databases as an alternative to relational databases in an interconnected Big Data environment. It will also show the theoretical basis behind graph databases and how they outperform relational databases in such an environment, but also why they are better suited for this kind of environment than other NoSQL alternatives. A state of the art in graph databases and how they compare to relational databases in various scenarios will also be presented in this paper.**

Keywords – Graph Databases; Relational Databases; NoSQL Databases; Interconnected Data; Big Data

I. INTRODUCTION

Relational databases have been the powerhouse of storing and analyzing data for decades. Their reliability, simplicity, and sheer power [1], but also a huge community of users have been an integral part of the reason why they have been at the top of their field for an unprecedented amount of time. At their core, relational databases are simply a series of previously defined tables which can usually be connected to other tables via foreign keys in order to retrieve related data. Those tables can then be used to store data and access it in order to gain certain knowledge. Although they have been steadily developing and advancing through the years, their core has always remained the same and that simple concept was enough to meet most of the market's data needs for decades.

However, in recent years, the Big Data revolution has been gaining huge popularity in the field of data management and with it a great number of new challenges have emerged. Most of these new challenges stem from the fact that there is a very large quantity of data that is seldom structured and that needs to be analyzed in order to make sense of it. Additionally, these data are often highly interconnected, which means that the entities within the database are very densely connected to each other. An example of highly interconnected data would be a social network where most users have a lot of friends, to which they are often connected in many different ways. Being able to handle that kind of interconnection will make analyzing the data much swifter and more efficient. Using more traditional tools like relational databases to deal with these new challenges is, of course, possible, but the problem with such attempts is that they are inefficient. The reason for that is because a large amount of interconnected data with lots of large tables often translates into a lot of join operations, which slow down queries significantly and are extremely resource expensive, which is something that should be avoided in a Big Data environment. In this context, graph databases present themselves as a viable solution for this kind of scenario.

Graph databases are NoSQL (Not only SQL) databases that specialize in dealing with connections. They are based on graphs, which consist of nodes and the relationships that connect them [2]. They have been around for a few decades, in one shape or another, but only recently, with the emergence of cheaper hardware and growing amount of interconnected data, but also data in general, have they gained popularity.

The rest of this paper is organized as follows. Section II describes the other NoSQL databases that exist and the way they handle large amounts of interconnected data. Graph databases, graph database management systems and graph database properties are defined in section III. Integrity constraints in graph databases are also included there. Section IV shows a state-of-the-art overview of graph databases being compared to relational databases, which includes analyzing various papers to determine how they compare to relational databases in different scenarios. Section V concludes this paper.

II. ON NOSQL DATABASES AND HOW THEY HANDLE INTERCONNECTED DATA

A. About NoSQL databases

Although there is no commonly agreed upon definition of what exactly NoSQL databases are, there are a number of things many of these different definitions have in common [3]. NoSQL databases are commonly considered to be non-relational databases with flexible schemas which are capable of handling a large amount of load and data. They are known for their comparative advantages in performance and scalability over relational databases in a large number of scenarios, often offering a bigger variety of data types they can store.

Graph databases, which will be analyzed in detail in the next section of this paper, are considered one of four major types of NoSQL databases [4]. The other three are key-value databases, column-oriented databases, and document databases.

Key-value databases are the simplest of all NoSQL databases – they basically function as a dictionary. There is a key that is unique for each record and there can be any number of fields within it. The records are found within the database using this key. Key-value databases do not have an SQL-like language for querying data, which means that the key management has to be paid a lot of attention to. Some databases offer certain more sophisticated search capabilities in order to compensate for that.

Column-oriented databases are the most similar to relational databases out of all NoSQL databases, but there are several crucial differences between those two because of which they are considered NoSQL databases. The biggest one of these differences is that column-oriented databases store records by column as opposed to a relational table which stores everything in rows. The upside of that is that accessing data is much faster and more efficient. One can only query a subset of columns which eliminates reading from columns that are not relevant. However, that also means that inserting data is going to take much more than it usually would, which is why these databases are used mostly for querying data.

The data in document databases are stored within documents, not tables. These documents can be of different formats like XML, JSON or BSON. Each document has a unique key which is used to access it and a flexible schema which means that fields can vary between each document. That schema flexibility allows for the structure of a document to be changed at any time. They are very similar to key-value databases. In fact, they are something of an extension of them. They offer complex querying and more room for better record organization, but they sacrifice the simplicity of key-value databases in order to accomplish that.

B. The drawbacks of different NoSQL databases handling interconnected data

Each of the three databases mentioned in the previous section have reasons why they can be picked as an alternative to relational databases in a Big Data environment. The mentioned databases, however, start having issues when the data are heavily interconnected.

Key-value databases are very simple in their design and handling relationships is not really what they are supposed to do. There are certain workarounds that can be done in order for them to handle relationships to other records [5], but they are not ideal for an interconnected environment as they can be slow and ineffective.

Column-oriented databases are capable of handling relationships between data similarly to relational databases – by using joins, but that also means that they are struggling with similar problems. These join operations are faster than the joins in a relational database because of the way column-oriented databases function [6], but they will still take a lot of time to execute if there are a lot of them.

Document databases also do not have any foreign keys in the traditional sense. Instead, they can, if they have a relationship to another document, either have other documents nested inside of them or they can hold a reference to another document [7]. Both of these approaches have certain drawbacks. Having whole documents inside of other documents can use a lot of memory space, especially if there are going to be a lot of connections. Having a reference to another document may be spatially more effective, but it would drastically increase query times because there would have to be as many lookup operations as there are references.

After analyzing how the other three NoSQL database types handle interconnected data, it can be concluded that none of them can handle this type of data without compromising on either storage, query time, or both. Graph databases, on the other hand, were made specifically with interconnected data in mind and that is why they are the NoSQL database being proposed as an alternative to relational databases in an interconnected Big Data environment.

III. GRAPH DATABASES

A. Graph database features

Graphs in graph databases consist of nodes and edges. The nodes represent individual records, while edges represent relationships between them. This makes adding or altering relationships or records very simple because changing the database schema is not required in order to do so. In fact, graph databases do not even have a schema – nodes and relationships can be added, changed, or deleted however and whenever the need for that arises. This is one of the reasons why graph databases are considered to be NoSQL databases. The absence of a schema also allows for easier data modeling and data management, which are incredibly important, especially in larger, more complex database systems.

Another important feature of graph databases is performance. They are optimized for traversing nodes using relationships between them, which makes them excel at finding related data, which especially comes to shine in large data sets.

Horizontal scalability is another graph database feature of significance. It allows for larger datasets and more complex data to be distributed among multiple nodes within a cluster [8], which makes querying much faster and more efficient. This feature allows scaling out to be much easier.

Visualization is a feature that allows the user to visualize the data within a graph database using visualization tools which many graph database management systems have built in. Graphs are much easier to understand when visualized, so using such tools can greatly help users understand relationships between nodes and find connections that would otherwise be difficult to spot.

A crucial feature to have in order to work with other types of systems is integration. Graph databases are able to be integrated with other systems like relational databases.

The relational database can, for example, be used for the purpose of data transaction, while the graph database can be used for the purpose of data analysis.

B. Graph database management systems

Graph database management systems (GDBMSs) are software tools which are used for working with graph databases. Each GDBMS provides unique features, which make them useful for different applications and use-cases. Before choosing a GDBMS, the user must consider their needs and choose one which meets those needs best.

One of the most important differences between each GDBMS is based on which kind of graph data model they support. There are two main types of graph data models: the property graph data model and the RDF (resource description framework) graph data model. The RDF graph data model stores values in the form of triplets (subject-predicate-object). Each subject and predicate have a unique URI (uniform resource identifier), which is used to reference them, while the object can be a URI or a literal, such as a person's name. Such a standardized way of storing and referencing objects makes it easy to integrate the data from multiple sources, but it also has certain drawbacks, such as offering less flexibility and being less intuitive for users, which are addressed in the property graph data model.

The property graph data model [9] is much simpler than its counterpart – it uses local data identifiers to reference objects. These identifiers are mere strings which reference objects that are collections of different data structures which can point at other data structures. Such a structure makes working with graph database management systems that use it much easier by making everything simpler to understand and implement. Since property graph data models are a lot different than RDF data models, the user will have to choose which one of them they will use based on the user's use case and their needs and resources.

Another important difference between GDBMSs refers to graph languages, which are used to query the graph databases. While they may differ in syntax and are often tailored to specific GDBMSs, there are graph query languages, such as Cypher and Gremlin, which are compatible with multiple GDBMSs.

C. Graph database integrity constraints

Integrity constraints ensure that every record within the database follows a certain user-defined ruleset in order for the database to be in a consistent state [10]. These constraints are extremely important for databases that are used for data transactions. By defining integrity constraints, the user can be certain that inserting and altering records is not going to violate the defined ruleset. Adhering to this ruleset ensures that there are no issues during the data transaction process and to make sure that data quality is maintained, while also ensuring that there are no inconsistencies or errors in the data.

The lack of schema in graph databases, although useful in many use cases, makes implementing integrity constraints substantially more difficult. The reason for that is that there is no predefined structure to enforce the constraints upon. The data can be more flexible and varied, so a more complex logic and mechanisms are required to validate and maintain data, such as creating an additional layer for validation. Using such a solution, however, will result in performance overhead, which can significantly increase transactional data query times.

IV. STATE OF THE ART IN GRAPH DATABASES AND HOW THEY COMPARE TO RELATIONAL DATABASES

This section will analyze various research papers that compare graph databases and relational databases in different aspects and scenarios to determine their relative strengths and weaknesses. The motivations behind the research, the methods used, and the conclusions made in the research papers will also be analyzed in order to provide a fuller picture of the papers in question. Additionally, the state-of-the-art findings will be summarized at the end of this section.

A. Graph databases and relational databases comparison

The authors of [13-17] compared graph databases to relational databases in various scenarios in order to see where graph databases perform better and where they do not. In order for the relational and graph databases to be comparable, usually one of them has to be converted into the other. Specifically, the data structures and the data itself have to be migrated. There are different approaches that can be used to do that, from manual mapping to using automated tools or even tools built into the GDBMS itself. Most of these tools convert relational tables and their columns into nodes and their properties, while the relationships between those nodes are created based on the foreign keys used in the tables. Depending on the schema complexity, manual mapping may be necessary after the automated mapping is completed in order to correctly convert some of the more complex data models. Although many different relational databases are present in the papers, most of the papers use Neo4j as the representative for graph databases because it is the most mature and commonly used graph database on the market [11; 12]. The papers in question are analyzed below.

In [13] relational and graph databases were compared starting from database modeling. That part involved conceptualizing the data model for both the relational and graph database to be used in the rest of the paper. They concluded that both database types are equally simple and intuitive to model, with the graph data model being more flexible, which is an obvious consequence of graph databases being a NoSQL database. The next step was creating both databases and importing the data. The dataset used was generated in .csv format, which was easy to import to the relational database. The procedure for importing a .csv file into a graph database, however, took a lot more time than for the relational database. The last part of the paper consists of measuring two aspects: queries and modifications. The experiments were performed on a dataset that involves telecommunications, with all the steps and procedures being well explained and documented. The relational database in the experiments was implemented using Oracle DB, while Neo4j was used to implement the

graph database used. The comparison was made in terms of query execution time and the number of full database reads. The results show that, on the dataset used in the paper, relational databases are faster at executing queries that require less joins, but graph databases are a lot faster when the join count is higher. Data modification was also faster in the graph database and was constant, while the record modification in the relational database depended on the number of records, with the modifications taking longer, the more records there were.

The authors of [14] compared graph databases to relational databases using a cluster of computers to conduct the experiments. The relational database used in the experiments was PostgreSQL, while the graph database used was Neo4j. There were several types of queries used for the benchmark and there were several queries for each query type. Each query was run five times, and the authors removed the highest and lowest value for each query while averaging the remaining three values to get the final result. The results show that the graph database had a similar performance to the relational database in most queries, but there were queries where it was a lot better. The authors made a recommendation in which scenario to use graph databases. The authors, however, did not include data insertion and modification into the benchmark query types, but they mention that and other issues as they assess the construct, internal and external threats to the validity of their own results.

In [15], graph databases and relational databases were compared based on their performance on various query types with a greater emphasis on complex queries. MariaDB and MySQL were the relational databases used in the experiments, while Neo4j was the graph database used. Queries were run with and without indexes in order to see what difference using them makes on the performance. The indexes on relational databases made a huge difference in performance, but the indexes in the graph database did not, other than helping find the starting point in the graph faster. There was a large variety of queries used in the experiments and the authors have documented both the SQL queries and their graph query equivalents written using the Cypher graph query language. The results show that Neo4j outperformed the two relational databases in the simpler queries, but the relational databases were faster on more complex queries. The authors, however, conclude that it is not possible to generally state which one is better because it largely depends on the complexity of the data and individual queries.

The author of [16] compared a graph database and a relational database on a social shopping application example. Neo4j was the graph database used in the experiments, while MSSQL was the relational database used. The dataset and the database modeling process, both for the relational database and graph database, were fully explained, and documented. The author provides graphical interpretations of the results of the graph database queries, and they claim that the graph database queries were faster, but they do not provide any numerical data to support that claim. As such, readers should take these results with caution until further evidence can be presented. Although the number of queries that were run in the experiments was limited, the author intends to add more query types and a larger variety of data to cover in their future work.

Graph databases and relational databases were compared based on different query types in [17], with the query types including selection, recursion, aggregation, and pattern matching. Neo4j was the graph database used for the experiments, while MySQL was the relational database used. The authors included where to find the test database used and the SQL codes for the relational databases and their Cypher equivalents in order for the test results to be fully reproducible. The results of the experiments indicate that the graph database performs better than the relational database on every query type. On recursive query types, which involve querying relationships between nodes in a loop, the graph database was a hundred times faster than the relational database on each individual query. This was to be expected considering graph databases are a lot better in these types of queries than relational databases, but the graph database was vastly superior in almost every other query type as well. Although the experiments only included one dataset, the authors plan to add more tests and datasets in future research.

B. Graph database integrity constraints implementation

Integrity constraints are certainly an important factor to consider in order to determine if graph databases can be used as an alternative to relational databases. They are a lot easier to implement in relational databases where the schema is known and predefined, as opposed to graph databases (or NoSQL databases in general) where it is not. The authors of [18-20] analyze the integrity constraints available in graph databases and discuss ways of implementing new ones.

In [18], several types of integrity constraints are listed such as nullability, uniqueness and range constraints. The authors explain what they are, after which they analyze two graph query languages and the constraints they support: Cypher and Gremlin. Although there are not many integrity constraints available by default, the authors introduce two new approaches for how integrity constraints can be achieved: integrated and layered. The difference between the two is that the integrated approach changes the system's source code in order to implement integrity constraints, while the layered approach creates a new layer where the constraints are implemented without changing the source code. Both ways are described in detail, together with the advantages and disadvantages of each of them. The authors also implemented their own node attribute uniqueness constraint in Neo4j's Gremlin graph query language, and they describe in detail how they implemented it, together with the problems they had along the way and how they managed to overcome them.

The authors of [19] and [20] discuss integrity constraint challenges in graph database modeling. Their focus was mostly on Neo4j's integrity constraints and possibilities. They analyze the integrity constraints Neo4j has by default and list the constraints that can be useful to implement. The constraints that were implemented at the end were: uniqueness on one node's attribute, uniqueness on several of the node's attributes and mandatory property value. They managed to implement them by manipulating the

folders where the metadata for the nodes and relationships are being held. They proved that implementing these integrity constraints is possible, however, they conclude that they were only looking for a feasible way to implement them and that there is a lot of space for improvements.

C. Graph data warehouses

The authors of [22;23] focus on graph databases in data warehousing. Data warehouses are structures which store and manage a large amount of data from different sources in order to gain a detailed view of various parts of a business or the whole business itself [21]. Data warehouses are usually implemented using relational databases, but the authors of the beforementioned papers have come to the constatation that using relational databases for analytical processing in certain data warehouses focusing on interconnected data has its challenges, which are a direct consequence of using relational databases for implementing data warehouses. Most of these challenges are similar to the ones mentioned in this paper and include performance issues and overall ineffectiveness while using relational databases for working with heavily interconnected data.

A state of the art in NoSQL graph data warehouses for Big Data social network analysis is captured in [22]. While explaining the drawbacks of using other NoSQL database types for creating a data warehouse for analyzing social networks and backing those claims up with research that has been done on those fields, the authors also introduce a new architecture for creating a NoSQL graph data warehouse for social networks. This architecture involves using document-oriented databases to store information, while using ETL tools to migrate parts of the network that require analysis into the graph data warehouse, which would be based on Social Graph Cubes – a concept proposed by the authors of [24]. Such architecture would be able to support OLAP (on-line analytical processing) on multidimensional social networks. This architecture, however, is not described in detail and is instead simply a concept which has been proposed by the authors.

In [23], a set of rules is proposed in order to turn a multidimensional model into a graph data model, which can be used to implement a star-like and snowflake-like graph data warehouse. The authors included the hardware specifications of the machine used in order for the experiments to be fully reproducible. They used several datasets of varying size and several types of queries divided into several categories based on the complexity of the queries and the number of tables involved in the query. The graph data warehouse was implemented using Neo4j, while the relational data warehouse was implemented using MariaDB. The same query could not be used on both the data warehouses due to graph databases using their own query language, so after running one query, the equivalent of the query had to be written and run on the other type of data warehouse in order to compare the two. The summarized results of the research performed in this paper are shown in Table I. The query times in the table are based on the queries performed on the largest dataset, which comprises 7GB of data. However, it is important to note that the findings of the paper remain consistent across datasets of varying sizes. The results show that the graph data warehouse takes a lot more time to write records than the relational data warehouse, which is due to the fact that it has to create the nodes, but also the relationships between them. Additionally, it also performs worse on non-hierarchical queries, which are queries that do not involve a specific hierarchical structure and are a more general type of query used for various data retrieval tasks. On the other hand, the results also show that the graph data warehouse performs a lot better on hierarchical queries, which are queries where the structure typically involves parent-child relationships, where each child record is linked to a parent record, and which can often be very complex to query. Additionally, the graph data warehouse also performs better on queries which are both hierarchical and cumulative, requiring aggregation in addition to the complexity of the hierarchical queries. The results generally show that the relational data warehouse performs better on queries which are less complex, while the graph data warehouse performs better on hierarchical and more complex queries. A surprising find in the research is that the star and the snowflake schema of both data warehouses show similar results throughout most of the experiments. The conclusion section of the paper notes that, based on the results of the experiments, it would be beneficial to use graph data warehouses for implementing OLAP (online analytical processing) systems and performing analytical queries in a data warehouse kind of structure.

TABLE I. SUMMARIZED RESULTS OF QUERIES USED IN [23] BASED ON QUERY TYPE

	Relational data warehouse average query times [sec]		***Graph data warehouse average query times [sec]***	
Query type	***Star schema***	***Snowflake schema***	***Star schema***	***Snowflake schema***
Data insertion queries	1052	1088	11391	14083
Non-hierarchical queries	8,92	9,01	83,12	83,39
Hierarchical queries	84,08	85,43	4,57	4,69
Hierarchical and cumulative queries	83,93	85,11	10,25	12,38

D. Critical summary on the state-of-the-art findings

There are many papers that compare graph databases to relational databases in one way or another. However, while comparing the relational database to the graph database, most of the research papers use Neo4j to implement the graph database used in the comparison. Although it is considered the most developed and complete GDBMS on the market, there should be more papers that focus on other GDBMSs in order to have a fuller picture of the whole field. Most papers' results show that graph databases provide faster query results than relational databases in most of the analyzed cases. These cases usually involve datasets with highly interconnected data, which is something that puts them perfectly within the scope of this paper. The queries used, however, include mostly select queries and very few of them include insertion and update queries, which are equally as important and there should definitely be more emphasis on those aspects of querying in future works.

In order to analyze the comparability of graph databases to relational databases in the data transaction aspect, the options for enforcing integrity constraints in graph databases were examined. The analyzed papers show that further work is needed to improve the implementation of integrity constraints in the native GDBMSs. Although there are ways of getting around it by implementing the constraints on a higher application layer, this approach is usually less efficient than the constraints being built into the graph management system itself, which can potentially introduce significant performance overhead.

Data warehousing focuses on storing and analyzing a large amount of data from different sources. The papers analyzed in this field show various ways and methods of transforming relational data warehouses into graph data warehouses, but only the authors of [22] offer concrete metrics that compare the two in various aspects. The authors show that graph data warehouses offer a selection of analytical tools that relational data warehouses cannot compete with in terms of speed and efficiency when working with a large quantity of interconnected data. Those tools, however, come at the price of slower data insertion. Due to the limited number of studies conducted in this field, it would be beneficial to conduct additional research in order to substantiate or refute the claims made in the papers in this field, but also find new ways of implementing data warehouses using graph databases and comparing them to relational data warehouses.

V. CONCLUSION

In this paper, an alternative to relational databases in an interconnected Big Data environment was given in the form of graph databases. The reasons for choosing graph databases were thoroughly explained, but the alternative NoSQL solutions and the explanations on how they would perform in such an environment were also given. An analysis of graph databases was provided which included defining the most important graph database features and types of graph database models. To provide a full picture of the state of the art in this field, several research papers including graph databases and their comparison to relational databases in various scenarios were analyzed.

With both the theory presented and the research papers analyzed, it can be concluded that graph databases are indeed a valid alternative to relational databases in an interconnected Big Data environment. The degree of their suitability, however, depends on the users' needs from the database itself. The majority of the available data show that graph databases are faster in analyzing data in such an environment, however, the data also show that using graph databases for transactional purposes is slower and less effective. Although the research papers analyzed cover many scenarios and situations, further research is needed in order to fully understand the differences between graph databases and relational databases. In addition to that, even though they were not included in the scope of this paper, there are relational database management systems, which have certain graph database capabilities as part of their solution, such as PostgreSQL or Oracle DB, which could allow for the user to leverage the benefits of both relational and graph databases for certain use cases, potentially proving to be a good future research opportunity.

REFERENCES

[1] P Baxendale and E F Codd, "A relational model of data for large shared data banks," Communications of the ACM, (1970), 377-387

[2] Robinsson I, Webber J and Eifrem E, "Graph Databases: New Opportunities for Connected Data," O'Reilly Media, (2015)

[3] He C, "Survey on NoSQL Database Technology," Journal of Applied Science and Engineering Innovation, (2015), 2(2)

[4] Gupta A et al., "NoSQL databases: Critical analysis and comparison," 2017 International Conference on Computing and Communication Technologies for Smart Nation, 2017, 293-299

[5] Van Hieu D, Smanchat S and Meesad P, "MapReduce join strategies for key-value storage," 2014 11th Int. Joint Conf. on Computer Science and Software Engineering 2014 (2014), 164-169

[6] Abadi D, Madden S and Hachem N, "Column-stores vs. row-stores: How different are they really?," Proceedings of the ACM SIGMOD International Conference on Management of Data, (2008), 967-980

[7] Celesti A, Fazio M and Villari M, "A Study on Join Operations in MongoDB Preserving Collections Data Models for Future Internet Applications," Future Internet 2019, Vol. 11, Page 83, (2019)

[8] Fernandes D and Bernardino J, "Graph Databases Comparison: AllegroGraph, ArangoDB, InfiniteGraph, Neo4J, and OrientDB," DATA 2018 - Proceedings of the 7th International Conference on Data Science, Technology and Applications, (2018), 373-380

[9] Angles R, "The property graph database model," Alberto Mendelzon Workshop on Foundations of Data Management, (2018)

[10] Codd E, "Data models in database management," Proceedings of the 1980 Workshop on Data abstraction, Databases and Conceptual Modeling, ACM Press, (1980), 112-114

[11] "DB-Engines Ranking - ranking of graph DBMS", https://db-engines.com/en/ranking/graph+dbms (accessed April 21. 2023).

[12] "Best Graph DBs in 2023", https://www.g2.com/categories/graph-databases (accessed April 21. 2023).

[13] Lazarska M and Lamch O, "Comparative study of relational and graph databases," INFORMATICS 2019 - IEEE 15th International Scientific Conference on Informatics, Proceedings, (2019), 363-370

[14] Macak M, Stovcik M and Buhnova B, "The Suitability of Graph Databases for Big Data Analysis: A Benchmark," Proceedings of the 5th International Conference on IoT and Big Data (2020)

[15] Kotiranta P et al., "Performance of Graph and Relational Databases in Complex Queries," Applied Sciences 2022, Vol. 12, Page 6490

[16] Uzunbayir S, "Relational Database and NoSQL Inspections using MongoDB and Neo4j on a Big Data Application," Proceedings - 7th International Conference on Computer Science and Engineering, UBMK 2022, (2022), 148-153

[17] Do T et al., "Query-based Performance Comparison of Graph Database and Relational Database," ACM International Conference Proceeding Series, (2022), 375-381

[18] Šestak M, Rabuzin K and Novak M, "Integrity constraints in graph databases - implementation challenges," Proceedings of Central European Conference on Information Systems, (2016), 23-30, 2016

[19] Pokorný J, Valenta M and Kovačič J, "Integrity constraints in graph databases," Procedia Computer Science, (2017), 975-981, 109

[20] Pokorný J, "Conceptual and database modelling of graph databases," ACM International Conference Proceeding Series, (2016), 370-377, 11-13-July-2016

[21] Gardner S, "Building the data warehouse," Communications of the ACM, (1998), 52-60, 41(9)

[22] Akid H and Ayed M, "Towards NoSQL graph data warehouse for big social data analysis," Advances in Intelligent Systems and Computing, (2017), 965-973, 557

[23] Akid H et al.,"Performance of NoSQL Graph Implementations of Star vs. Snowflake Schemas," IEEE Access, (2022), 48603-48614

[24] De Virgilio R, Maccioni A and Torlone R, "R2G: a Tool for Migrating Relations to Graphs," Proceeding of the 17th International Conference on Extending DB Technology (2014)