Query-based Performance Comparison of Graph Database and Relational Database

Thi-Thu-Trang Do
Faculty of Information Technology
Hung Yen University of Technology and Education
Hung Yen, Vietnam
trangdtt@utehy.edu.vn

Thai-Bao Mai-Hoang
Faculty of Information Technology
Hung Yen University of Technology and Education
Hung Yen, Vietnam
maihoangthaibao01@gmail.com

Van-Quyet Nguyen
Faculty of Information Technology
Hung Yen University of Technology and Education
Hung Yen, Vietnam
quyetict@utehy.edu.vn

Quyet-Thang Huynh*
School of Information and Communication Technology
Hanoi University of Science and Technology
Ha Noi, Vietnam
thanghq@soict.hust.edu.vn

ABSTRACT

A graph database is a type of NoSQL database that uses graph structure for semantic queries with nodes, edges, and properties to represent and store data. It has been applied in many fields, such as education, health, business, and social network, with many famous applications such as Google, Facebook, and eBay. One of the main advantages of the graph database is its effective performance in data queries. This paper presents a comprehensive comparison of the performance based on the execution time of a NoSQL graph database named Neo4J with a popular relational database system, MySQL, which is used as the underlying technology in developing a software system. Query types are categorized into four groups: selection/ search, recursion, aggregation, and pattern matching. We examined representative questions for each group and executed them on Neo4j and MySQL using a real-life dataset named Career Village. The results show that Neo4j's data query performance is better than MySQL in most results.

CCS CONCEPTS

• **Information systems → Data management systems**.

KEYWORDS

knowledge graph, graph database, relational database, complex queries, query performance

ACM Reference Format:
Thi-Thu-Trang Do, Thai-Bao Mai-Hoang, Van-Quyet Nguyen, and Quyet-Thang Huynh. 2022. Query-based Performance Comparison of Graph Database and Relational Database. In *The 11th International Symposium on Information and Communication Technology (SoICT 2022), December 01–03, 2022, Hanoi–Halong bay, Vietnam.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3568562.3568648

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SoICT '22, December 01–03, 2022, Hanoi–Halong bay, Vietnam
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9725-4/22/12…$15.00
https://doi.org/10.1145/3568562.3568648

1 INTRODUCTION

In recent years, knowledge graph has been applied in many fields, from social applications to science [18]. Big companies like Google, Facebook, eBay, Microsoft, NASA, and others use knowledge graphs to improve their performance. It helps improve user experience, provide more accurate recommendations, enhance search engines, content management, impact analysis, real-time analysis, financial services, retail services, health care, security, traffic, media, IoT, and AI/ machine learning. A knowledge graph represents a structure consisting of entities, relationships, and semantics, similar to human knowledge. Graph databases often store knowledge graph data and the logic that describes interconnections and context. Graph databases can provide a unique and efficient chart store structure, including some graph databases like Neo4j, ArangoDB, Amazon Neptune, Dgraph, OrientDB, and GraphDB, where Neo4J is used most widely.

One of the powers of graph databases is the data query performance, especially in model-fit and query tasks exploiting relations between nodes. It differs from the SQL data model using relationships between tables.

There have been several studies on the comparison of graph databases, especially Neo4j, with traditional relational databases, mainly MySQL. These studies analyzed certain query aspects such as recursion, partial matching, clustering, and path queries [1, 4, 6, 8, 9, 14]. Other studies have analyzed queries in specific areas like Web-based applications[5], Heterogeneous IoT Data Management[15], CRM Systems[17]. Furthermore, most of these studies mainly work with quite simple queries.

This study evaluates the data query performance on four queries: select/search, recursion, aggregation, and pattern matching between Neo4J and MySQL databases[3]. Moreover, complex queries on graphs and relational databases also were realized to compare the performance of the two query methods since working with a large dataset. A complex query is a type of query whereby data must be fetched from multiple joined tables in an SQL database or by using a path with various links to different types of nodes that can be recursive in graph databases. In aggregate computation, complex queries are computed values on a large dataset.

The rest of the paper is organized as follows. Section 2 presents our survey of previous work related to performance analysis between MySQL and Neo4j. Section 3 represents the methods to evaluate the performance of two database systems, graph databases (Neo4J) and relational databases (MySQL). Section 4 introduces the test environment and test results. Finally, the conclusions and discussion are described in section 5.

2 RELATED WORK

Several studies have been conducted to compare query performance between SQL and graph databases, especially Neo4j. Study results concluded that the performance of the graph databases is better than that of the conventional databases. [2, 7, 16]. However, these studies were performed on either simple queries or small-sized datasets. In [2], S. Batra et al. used datasets with 100, 500 rows/objects stored in four tables (including users, friends, movies, and favorite actors), three types of nodes, and three relationships corresponding to MySQL and graph database, respectively. The derived results showed that the query time on MySQL was 2-30 times slower than Neo4j. In [16], a comparison between MySQL and Neo4j is presented. The authors used three queries with 10-10,000 records, and one was used to feed the probabilistic database data capable of handling unstructured data. Study results showed that MySQL was faster and saved memory than Neo4j; Neo4j was better than MySQL in terms of flexibility. However, the relational schema on MySQL only includes two join tables, and the schema on the graph consists of 5 connected entities. The queries use up to two join tables/related entities and focus on the selection/search query type. Holzschuher et al.[7] tested and compared the performance of Neo4j and MySQL using different backend solutions. The authors used Cypher, Gremlin, and SQL in this research to write queries. The study results showed Neo4j significantly improved performance comparison since the increased database size. Gremlin and Cypher query on Neo4j were executed faster than those using MySQL with JPA.

Also many studies focused on this topic, for example: [2, 7, 10, 11, 13, 16] used search/selection query and the aggregate query method; [12], the author used five query tasks for job price, job price with items, invoice price, invoice price for a given customer…; [9] the Recursive queries are presented; In [6], a comparison between Neo4J and relational databases in pattern matching queries is described; Clustering queries are introduced in [8], and path queries are in [1, 8, 14]; and the following studies were several areas were mention like Social Network Analysis [4], Web-based applications [5], Heterogeneous IoT Data Management[15], CRM Systems [17].

3 PERFORMANCE COMPARISON OF GRAPH DATABASE AND RELATIONAL DATABASE USING QUERY-BASED TECHNIQUES

In database systems, data queries have many different purposes. Based on query questions, classification is split into four queries: selection/search, related or recursive data, aggregation, and pattern matching[3]. We take three illustrative examples for each query type and compare the performance between Neo4j and MySQL.

3.1 Test Database

The test database is a general example of Web application information systems. Different types of information are associated with professionals and students. The dataset is taken from CareerVillage.org, a nonprofit that crowdsources career advice for underserved youth. This platform uses questions and answers similar to the formulation method of the StackOverflow/Quora website. The purpose is to provide answers in many career fields for students.

Table 1: The numbers of the generated rows/objects in MySQL and Neo4j

Table/Object	Row in MySQL	Object in Neo4j
professionals	28,152	28,152 nodes
students	30,971	30,971 nodes
questions	23,931	23,931 nodes
answers	51,123	51,123 nodes
comments	14,966	14,966 nodes
tags	16,269	16,269 nodes
tag_questions	76,553	76,553 edges
tag_users	136,663	135,907 edges
groups	49	49 nodes
group_memberships	1,038	1,038 edges
emails	1,850,101	1,850,101 nodes
matches	4,316,275	1,116,275 edges
schools_memberships/ schools	5,638	2,706 nodes

The relational database has 13 tables corresponding to students, professionals, questions, answers, comments, tags, groups, and items, which contain the professional information, student information, the question asked by the professional or student, the answer information, and the comment for the question. The tags and the groups are categorized questions, professionals, and students into topics of interest. Figure 1 shows the database structure as a relational database schema. Symbols illustrate how the tables are associated together.

Our graph database schema represents entities as nodes and relationships as directed edges. Professional, student, question, answer, comment, tag, group, and several objects are represented as nodes. The attributes of relationships are represented on edges. Figure 2 introduces to the graph database structure. Table 1 shows the numbers of rows/objects generated for the dataset. Each row in the table has two columns. One presents the number of rows in the relational database, and another column represents the number of nodes in the graph database.

3.2 Test Queries

This section presents representative query questions per group on MySQL and Neo4J.

- Selection/Search

Queries in this grouping focus on finding and selecting a group of data from a database by one or more criteria. This type of question does not require the use of rich relationships in the data. Examples of queries are:

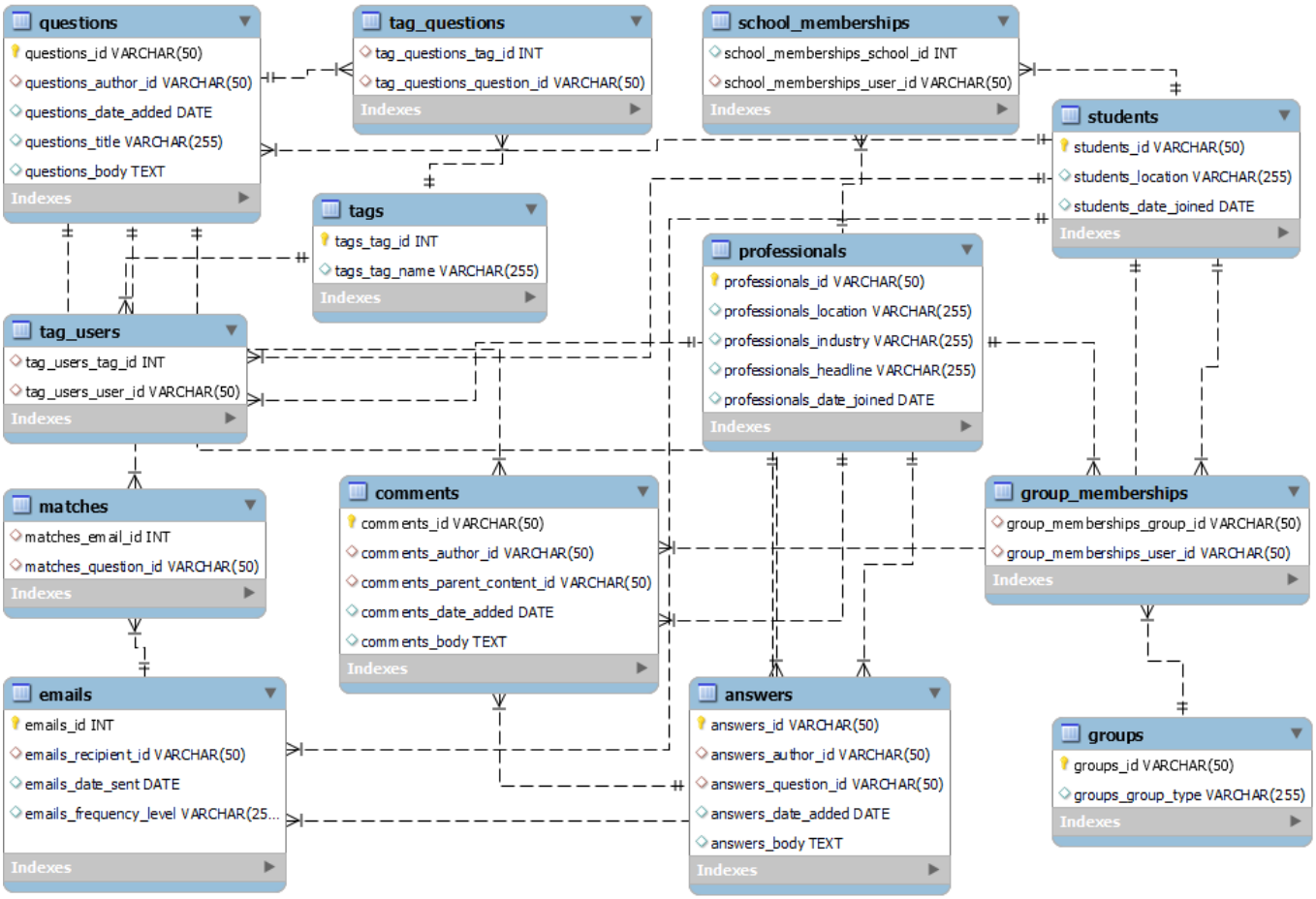

Figure 1: Relational database structure

Q1: Looking for professionals in a specific tag?

SQL

```
SELECT p.* FROM professionals p
 JOIN tag_users tu ON
   p.professionals_id = tu.tag_users_user_id
 JOIN tags t ON tu.tag_users_tag_id = t.tags_tag_id
WHERE tags_tag_name = 'college';
```

Cypher

```
MATCH (p:professionals)-[]->(t:tags)
WHERE t.tags_tag_name='college'
RETURN p,t
```

Q2: Looking for students in a specific group and interested in a specific tag?

SQL

```
SELECT * FROM students s
 JOIN group_memberships gm ON
   students_id = gm.group_memberships_user_id
 JOIN groups g ON
   g.groups_id = gm.group_memberships_group_id
 JOIN tag_users tu ON
   tu.tag_users_user_id = s.students_id
 JOIN tags t ON t.tags_tag_id = tu.tag_users_tag_id
WHERE t.tags_tag_name = 'college'
 AND g.groups_group_type = 'youth program';
```

Cypher

```
MATCH (t:tags)<-[:HAS_TAG]-(s:students)-
   [:MEMBER_IN]->(b)
WHERE t.tags_tag_name='college'
 AND b.groups_group_type='youth program'
RETURN s,t,b
```

Q3: Looking for all emails received by a particular professional?

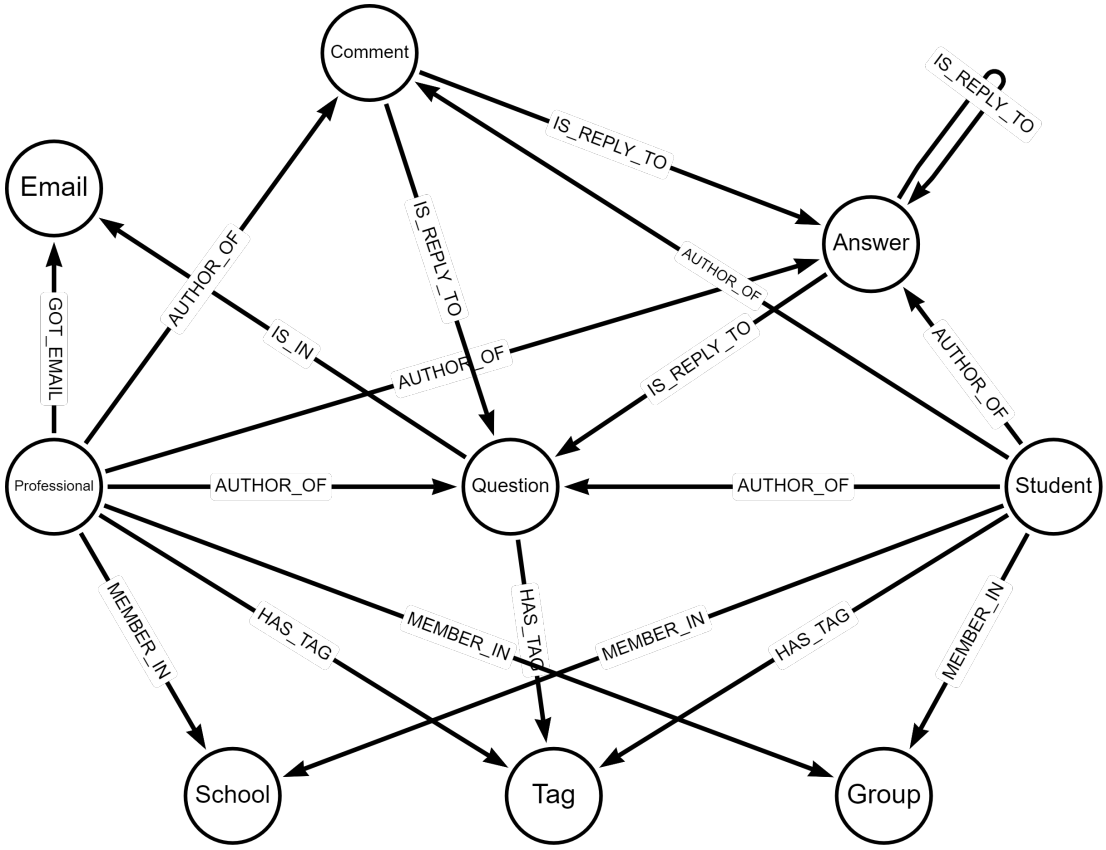

Figure 2: Graph database structure

SQL
SELECT * FROM professionals p JOIN emails e ON p.professionals_id = e.emails_recipient_id WHERE p.professionals_id = '0079e89bf1544926b98310e81315b9f1';
Cypher
MATCH (p:professionals{professionals_id: '0079e89bf1544926b98310e81315b9f1'})- [:GOT_EMAIL]->(e:emails) RETURN e

- Recursive data

This set of queries explores relationships between entities and is used to query hierarchical data, such as an organizational structure, bill-of-materials, and document hierarchy. Recursive queries are made to call themselves many times in a row until they reach some exit or termination condition. Some examples for questions:

Q4: Looking for the questions with answers recursively many times?

SQL
WITH RECURSIVE answer_replies AS(SELECT answers_id, answers_author_id, answers_question_id, answers_date_added, answers_body FROM answers WHERE answers_question_id IS not null UNION all SELECT a.answers_id, a.answers_author_id, a.answers_question_id, a.answers_date_added, a.answers_body FROM answers a INNER JOIN answer_replies ar ON ar.answers_id = a.answers_question_id) SELECT * FROM answer_replies ar LEFT JOIN questions q ON ar.answers_question_id = q.questions_id;
Cypher
MATCH (q:questions)<-[:IS_REPLY_TO*1..]-(a:answer) RETURN q,a

Q5: Looking for questions with answers recursively twice?

SQL

```
WITH RECURSIVE answer_replies AS(
 SELECT 1 as level,answers_id, answers_author_id,
   answers_question_id, answers_date_added,
   answers_body
 FROM answers WHERE answers_question_id
 IS not null UNION all
 SELECT level+1, a.answers_id,
   a.answers_author_id, a.answers_question_id,
   a.answers_date_added, a.answers_body
 FROM answers a
 INNER JOIN answer_replies ar ON
   ar.answers_id = a.answers_question_id
 WHERE level <=2)
 SELECT * FROM answer_replies ar
 LEFT JOIN questions q ON
   ar.answers_question_id = q.questions_id
```

Cypher

```
MATCH (q:questions)<-[:IS_REPLY_TO*1..2]-
 (a:Answers)
RETURN q,a
```

Q6: Looking for questions with answers recursively 3 times?

SQL

```
WITH RECURSIVE answer_replies AS(
 SELECT 1 as level,answers_id, answers_author_id,
   answers_question_id, answers_date_added,
   answers_body
 FROM answers WHERE answers_question_id
 IS not null UNION all
 SELECT level+1, a.answers_id,
   a.answers_author_id, a.answers_question_id,
   a.answers_date_added, a.answers_body
 FROM answers a
 INNER JOIN answer_replies ar ON
   ar.answers_id = a.answers_question_id
 WHERE level <=3)
 SELECT * FROM answer_replies ar
 LEFT JOIN questions q ON
   ar.answers_question_id = q.questions_id
```

Cypher

```
MATCH (q:questions)<-[:IS_REPLY_TO*1..3]-
 (a:Answers)
RETURN q,a
```

- Aggregation

This group of questions is used to find results from aggregated information from the database. Some examples of questions:

Q7: Count the number of professionals who answered the questions.

SQL

```
SELECT count(professionals_id)
 FROM professionals p JOIN answers a ON
   p.professionals_id = a.answers_question_id;
```

Cypher

```
MATCH (p:professionals)-[]->(a:Answers)
RETURN count(p)
```

Q8: Count the number of professionals of a specific tag.

SQL

```
SELECT count(*) FROM(
 SELECT DISTINCT p.* FROM professionals p
 JOIN tag_users tu ON p.professionals_id =
   tu.tag_users_user_id
 JOIN tags t ON tu.tag_users_tag_id = t.tags_tag_id
 WHERE t.tags_tag_name = 'college') AS temp;
```

Cypher

```
MATCH (p:professionals)-[:HAS_TAG]->(t:tags)
WHERE t.tags_tag_name='college'
RETURN count(p)
```

Q9: Which tag has the most professionals?

SQL

```
SELECT tags.tags_tag_id, tags_tag_name,
 COUNT(p.professionals_id)
 AS number_of_professionals FROM tags
 JOIN tag_users tu
   ON tags.tags_tag_id = tu.tag_users_tag_id
 JOIN professionals p ON p.professionals_id =
   tu.tag_users_user_id
 GROUP BY tags.tags_tag_id, tags_tag_name
 ORDER BY COUNT(p.professionals_id)
 DESC LIMIT 1;
```

Cypher

```
MATCH (p:professionals)-[:HAS_TAG]->(t:tags)
RETURN t.tags_tag_name AS TagName,
 COUNT(p) ORDER BY COUNT(p) DESC
 LIMIT 1
```

- Pattern matching

Pattern matching allows for finding patterns in the data. Pattern matching is based on how entities are related to each other. The query type is often used in cases like recommendation engines, fraud detection, or intrusion detection. Some questions might include the following:

Q10: Looking for the question answered in tags?

SQL
SELECT q.questions_id, t.tags_tag_id, a.answers_id FROM tags t JOIN tag_questions tq ON t.tags_tag_id = tq.tag_questions_tag_id JOIN questions q ON tq.tag_questions_question_id = q.questions_id JOIN answers a ON a.answers_question_id = questions_id;
Cypher
MATCH (a:Answers)-[]->(q:questions)-[]->(t:tags) RETURN a,q,t

Q11: Looking for students and professionals with the same group?

SQL
SELECT g.groups_id, professionals_id, students_id FROM groups g JOIN group_memberships gm ON g.groups_id = gm.group_memberships_group_id JOIN (SELECT group_memberships_group_id AS group_id, professionals_id FROM professionals p JOIN group_memberships gm1 ON gm1.group_memberships_user_id = p.professionals_id) pg ON pg.group_id = gm.group_memberships_group_id JOIN (SELECT group_memberships_group_id AS group_id, students_id FROM students s JOIN group_memberships gm2 ON s.students_id = gm2.group_memberships_user_id) sg ON sg.group_id= gm.group_memberships_group_id;
Cypher
MATCH (p:professionals)-[]->(g:groups)<-[]-(s:students) RETURN p, g, s

Q12: Looking for patterns that students and experts in the same tag?

SQL
SELECT pt.tags_id, st.students_id, pt.professionals_id FROM tags t JOIN tag_users tu ON t.tags_tag_id = tu.tag_users_tag_id JOIN (SELECT u.tag_users_tag_id AS tags_id, professionals_id FROM professionals p JOIN tag_users u ON p.professionals_id = u.tag_users_user_id) pt ON pt.tags_id= t.tags_tag_id JOIN (SELECT u.tag_users_tag_id AS tags_id, students_id FROM students s JOIN tag_users u ON s.students_id = u.tag_users_user_id) st ON st.tags_id = t.tags_tag_id LIMIT 100000;
Cypher
MATCH (p:professionals)-[]->(t:tags)<-[]-(s:students) RETURN p, t, s LIMIT 100000

4 EVALUATION RESULTS

4.1 Test Settings

The tests were executed in a Ubuntu Server with the following specifications:

- OS: Ubuntu 19.10, 64-bit
- Memory: 32GB
- Processor: Intel Core i7-8700 CPU 3.20GHz x 12
- Graphics: GeForce RTX2080 Ti/PCIe/SSE2
- Disk: 2.0 TB

MySQL versions 8.0.30 and Neo4j community edition version 4.4.5 were installed on this computer.

4.2 Test Results

Table 2: The query performance results on MySQL and Neo4J for the type of queries

Type of queries	Queries	Neo4j	MySQL
Selection/search	Q1	7 ms	8 ms
	Q2	9 ms	270 ms
	Q3	13 ms	39 ms
Related or recursive data	Q4	2 ms	292 ms
	Q5	3 ms	312 ms
	Q6	3 ms	314 ms
Aggregation	Q7	31 ms	118 ms
	Q8	15 ms	22 ms
	Q9	86 ms	429 ms
Pattern matching	Q10	4 ms	4 ms
	Q11	5 ms	49 ms
	Q12	1 ms	8 ms

Table 2 presents our study results. It illustrated that Neo4J outperformed the MySQL relational database system depending on the complexity of the data query. The higher the query complexity, the more Neo4j shows superiority over MySQL.

The questions of Aggregation type get aggregate information from the data, MySQL still does a great job with this type of question, but Neo4J's execution time is 1-5 times faster. The Pattern matching query group gives Neo4J results 1-10 times faster. The Select/Search type has Neo4J's query time of 1-30 times faster than MySQL. The Recursive data type has Neo4J query performance up to 146 times faster. For questions that require rich relationships in the data and use complex joins across many different tables/objects, Neo4J delivers superior query execution times compared to MySQL. Also, statements written in Cypher are often more concise than those written in SQL.

5 CONCLUSIONS

This paper presented an overview of queries categorized into four groups: selection/search, recursion, aggregation, and pattern matching. Second, we conducted results on data queries for Neo4j (a representative of the graph database) and MySQL (a representative of the relational database). Our results showed that in terms of execution time, the graph database outperforms the relational database up to 146 times when querying with complex data and big data. In the near future, we plan to add more tests with other datasets in different sectors, such as banking, the stock market, and ERP. We will compare the results on other aspects of system performance, such as memory usage, power consumption, and implementation complexities.

REFERENCES

[1] Zahid Abul-Basher, Nikolay Yakovets, Parke Godfrey, Shadi Ghajar-Khosravi, and Mark H Chignell. 2017. Tasweet: optimizing disjunctive regular path queries in graph databases. In *EDBT/ICDT 2017 joint conference 20th international conference on extending database technology. https://doi. org/10.5441/002/edbt.*

[2] Shalini Batra and Charu Tyagi. 2012. Comparative analysis of relational and graph databases. *International Journal of Soft Computing and Engineering (IJSCE)* 2, 2 (2012), 509–512.

[3] Dave Bechberger and Josh Perryman. 2020. *Graph Databases in Action.* Manning Publications.

[4] Wenfei Fan. 2012. Graph pattern matching revised for social network analysis. In *Proceedings of the 15th International Conference on Database Theory*. 8–21.

[5] Cornelia Gyorödi, Robert Gyorödi, and Roxana Sotoc. 2015. A comparative study of relational and non-relational database models in a Web-based application. *International Journal of Advanced Computer Science and Applications* 6, 11 (2015), 78–83.

[6] Jürgen Hölsch, Tobias Schmidt, and Michael Grossniklaus. 2017. On the performance of analytical and pattern matching graph queries in neo4j and a relational database. In *EDBT/ICDT 2017 Joint Conference: 6th International Workshop on Querying Graph Structured Data (GraphQ)*.

[7] Florian Holzschuher and René Peinl. 2013. Performance of graph query languages: comparison of cypher, gremlin and native access in neo4j. In *Proceedings of the Joint EDBT/ICDT 2013 Workshops*. 195–204.

[8] Yun-Wu Huang, Ning Jing, and Elke A Rundensteiner. 1996. Effective graph clustering for path queries in digital map databases. In *Proceedings of the fifth international conference on Information and knowledge management*. 215–222.

[9] Louis Jachiet, Pierre Genevès, Nils Gesbert, and Nabil Layaïda. 2020. On the optimization of recursive relational queries: Application to graph queries. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 681–697.

[10] Wisal Khan, Waqas Ahmad, Bin Luo, and Ejaz Ahmed. 2019. SQL Database with physical database tuning technique and NoSQL graph database comparisons. In *2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*. IEEE, 110–116.

[11] Wisal Khan, Waseem Shahzad, et al. 2017. Predictive performance comparison analysis of relational & nosql graph databases. *International Journal of Advanced Computer Science and Applications* 8, 5 (2017).

[12] Petri Kotiranta, Marko Junkkari, and Jyrki Nummenmaa. 2022. Performance of Graph and Relational Databases in Complex Queries. *Applied Sciences* 12, 13 (2022), 6490.

[13] Surajit Medhi and Hemanta K Baruah. 2017. Relational database and graph database: A comparative analysis. *Journal of process management and new technologies* 5, 2 (2017), 1–9.

[14] Van-Quyet Nguyen and Kyungbaek Kim. 2017. Estimating the evaluation cost of regular path queries on large graphs. In *Proceedings of the Eighth International Symposium on Information and Communication Technology*. 92–99.

[15] Van-Quyet Nguyen, Van-Hau Nguyen, et al. 2020. An efficient graph modeling approach for storing and analyzing heterogeneous IoT data. *UTEHY Journal of Science and Technology* 27 (2020), 21–27.

[16] Rahmatian Jayanty Sholichah, Mahmud Imrona, and Andry Alamsyah. 2020. Performance Analysis of Neo4j and MySQL Databases using Public Policies Decision Making Data. In *2020 7th International Conference on Information Technology, Computer, and Electrical Engineering (ICITACEE)*. IEEE, 152–157.

[17] Victor Winberg and Jan Zubac. 2019. A comparison of relational and graph databases for crm systems. *LU-CS-EX 2019-09* (2019).

[18] Xiaohan Zou. 2020. A survey on application of knowledge graph. In *Journal of Physics: Conference Series*, Vol. 1487. IOP Publishing, 012016.