Performance Comparison of SQL and Graph Databases

Subjects: Computer Science, Information Systems

Contributor: Petri Kotiranta, Marko Junkkari, Jyrki Nummenmaa

In developing NoSQL databases, a major motivation is to achieve better efficient query performance compared with relational databases. The graph database is a NoSQL paradigm where navigation is based on links instead of joining tables. Links can be implemented as pointers, and following a pointer is a constant time operation, whereas joining tables is more complicated and slower, even in the presence of foreign keys. Therefore, link-based navigation has been seen as a more efficient query approach than using join operations on tables. Existing studies strongly support this assumption. Query complexity has received less attention. For example, in enterprise information systems, queries are usually complex so data need to be collected from several tables or by traversing paths of graph nodes of different types.

Keywords: graph database ; relational database

1. Introduction

Performance is one of the motivations to use NoSQL databases instead of traditional SQL databases. With data and queries suitable for the data model, NoSQL databases might offer significant performance benefits. Researchers compared database systems of the traditional relational model and of the NoSQL graph model. In the graph model, which is one of the four major NoSQL types, the data consist of nodes and edges, and it has its own benefits when handling relationship rich data. While in SQL databases multiple tables may need to be joined for a relational query, in graph databases relational information can be queried by navigating through the graph.

Previous studies [1][2][3][4][5] where the performance of graph databases, especially Neo4j, was compared with the traditional SQL databases, indicate that graph databases possess better performance than relational databases. However, those studies mainly focused on quite simple queries. In contrast to the earlier studies, researchers investigated the performance of database systems in situations where the query complexity increased. In a complex query, the necessary data must be collected from several tables in an SQL database, or by traversing a path of different types of nodes, potentially using recursion, in a graph database. Using a complex query, an aggregated value (e.g., a count or an average) from a large data set can be calculated.

MariaDB and two versions of MySQL were selected as relational database systems. MariaDB was selected because it is a modern database system and, to the best of the knowledge, it has not been compared to graph database systems before. MySQL 8 was included in the present investigation in order to compare how the query performance of complex queries differs between MySQL and MariaDB. Old MySQL 5.1 was included to observe the efficiency development of relational database systems. MariaDB and MySQL 8 were initially based on MySQL 5.1. Thus, they belong to the same database system family, and their comparison is an indication of how relational database query efficiency has developed during the last decade. In addition to using complex queries, researchers also paid attention to factors related to efficiency. Indexing is a traditional method to improve performance and it can be applied with both relational and graph databases. Additionally, for the selected graph database system (Neo4J), a more efficient query execution type (call-function) was developed. Recursive queries can be optimized in modern versions of Neo4J. Researchers considered all these optimizations in the efficiency evaluations.

In order to benchmark the database systems using complex queries, researchers designed and implemented a new test bench that also supports complex queries. The test bench was designed for testing queries using MariaDB, MySQL, and Neo4J. The test database relates to enterprise information systems, but it is worth noting that the query types are general, and processing of the data is similar in many other domains. The test bench is called Invoicing Database Test Bench and its source code is available in GitHub [6]. The program generated a selected amount of data for the test invoicing database schema and performed various query tests. The dataset is public. The source code for generating the data is available in GitHub, and, thus, it is possible for anyone to repeat these tests by installing the same test bench and generating the same data.

2. Previous Performance Related Study

An older MySQL version was included in the comparison to make the research compatible with earlier studies. From this perspective, MariaDB is a natural choice for a modern database, because MariaDB was initially a descendant of MySQL. Based on the popularity of databases, the DB-Engines site ranks MariaDB as 8th out of 138 of relational databases [7]. Neo4j ranks 1st out of 32 graph databases on the same sites. DB-Engines ranks the databases according to current popularity. Popularity is measured using six parameters. The first parameter is the number of mentions on the websites Google and Bing. Second is the general interest which is measured by frequency in Google Trends. Third is the frequency of technical questions in Stack Overflow or DBA Stack Exchange. Fourth is the number of job offers in Indeed and Simply Hired. Fifth is the number of profiles in LinkedIn in which the system is mentioned. Sixth is the relevance in social networks which is counted by the number of tweets on Twitter, in which the system is mentioned. As all of the databases researchers studied are quite popular and are often candidates for use in enterprises. One of the goals of this study was to identify differences in what use case the databases should be used.

SQL databases and Neo4j have been compared in several studies [1][2][3][4][5]. Khan et al. compared tuned Oracle 11g and Neo4j 3.03 Community Edition [1]. They used healthcare data, including data of patients, medication, and medical staff. Performance of the databases was evaluated using ten different count(*) queries. Many of the queries included some table joins. A physical database tuning technique called tablespaces was used for Oracle. The same databases were compared without physical database tuning by Khan et al. [3]. The physical database tuning technique decreased the overall average query time of Oracle from 4.34 to 2.78 s. However, the overall average query time for Neo4j in query tests was only 0.67 s. Thus, Neo4j outperformed Oracle.

Holzschuher et al. tested Neo4j version 1.8 performance with different backend solutions [2]. Neo4j was benchmarked as embedded with native object access, as a dedicated server through RESTful Web Services, with embedded Cypher queries, with Cypher optimized for remote execution with REST, and with Gremlin queries through REST. MySQL version 5.5.27 was also included with Java Persistence API based backend. Queries were written using Cypher, Gremlin and SQL query languages. The test data consisted of data of persons and their relationships. Tests included such queries as friends of friends. As the size of the database increased, the advantages of Neo4j over MySQL became more evident. Neo4j performance stayed nearly constant when MySQL performance dropped by factors of 5 and 7–9. Queries in Neo4j query languages Gremlin and Cypher executed faster than queries using MySQL with JPA.

Vicknair et al. compared MySQL Community Server version 5.1.42 and Neo4j version 1.0-b11 in 2010 [4]. The graph database was transferred into a relational database as nodes and edges. Three types of structural and three types of data queries were made. The first structural query found all orphan nodes and the two other structural queries traversed the graph at depths of 4 and 128. The data queries were count(*) queries counting nodes with certain payloads. Neo4j performed better in structural queries. However, in data queries, MySQL was more efficient, partly due to the use of Lucene indexing in the tested Neo4j. The data contained integers, and Lucene treated the data as text by default, so conversions were necessary and thus impacted the performance. The work [4] by Vicknair et al. has been referenced in [1][2][3].

Batra et al. compared MySQL version 5.1.41 and Neo4j Community version 1.6 in 2012 [5]. They used a schema with tables user, friends, fav_movies, and actors for testing, and they tested the databases with three queries: “Find all friends of Esha”, “Find all favourite movies of Esha’s friends” and “Find the lead actors of Esha’s friends’ favourite movies”. Queries were executed on 100 and 500 objects. Neo4j had 2–5 times faster query execution times with a 100-objects data set and 15–30 times faster query execution with a 500-objects data set. The work by Batra et al. [5] was similar to that of the present study as the data were stored in an SQL database with a relational schema unlike in the work by Vicknair et al. [4]. The work [5] by Batra et al. is referenced in [2].

There also exist previous performance studies where MariaDB is involved. Tongkaw et al. compared the performance of MariaDB 10.0.21 and MySQL 5.6 [8]. They used the Sysbench and OLTP [9] software systems with OLTP-Simple and OLTP-Seats workloads. Both databases consumed the same number of resources. However, when increasing the number of threads in OLTP-Simple and the number of workers in OLTP-Seats, MySQL became clearly more efficient and outperformed MariaDB. Shalygina et al. studied the Common Table Expression capabilities of MariaDB by comparing it to Postgres [10]. The study showed that Postgres had better results, when only a few steps of recursion were needed. However, MariaDB was a better choice for longer-executing recursive queries on huge amounts of data.

Stanescu [11] compared the performance of SQL Server 2009 and Neo4j 4.0. Four datasets were used consisting of 350,000, 700,000, 1,400,000 and 2,100,000 entries. A schema with multiple relations between entities was used. Five

different query tests were performed with the different datasets. All the queries addressed relations between entities, so joins or matches between relationships were used. The results show that as query complexity and dataset complexity grew, Neo4j performed faster than the SQL Server.

Sholichah et al. [12] compared MySQL and Neo4j. Four queries were used. Three of the queries were tested with datasets containing 10, 100, 500, 1000 and 10,000 records. The fourth query was used to infer the databases' ability to handle unstructured data. In these tests, MySQL was in general faster and used less memory as the query complexity and number of records increased. Both databases were able to handle unstructured data.

Cheng et al. [13] compared RocksDB 5.8, Hbase 2.2, Cassandra 3.11, Neo4j 3.4.6 and MySQL 5.7. Four relational datasets from TPC-H benchmark were used as well as four real graph datasets. Different types of query workloads were tested, including atomic relational queries such as projection, aggregation, join and order by, TPC-H workloads, and different graph query algorithms. The conclusion of the tests was that relational databases outperformed graph databases with workloads that mainly consisted of group by, sort, aggregation operations and their combinations. Graph databases outperformed relational databases with workloads that mainly consisted of multiple table joins, pattern matching, path identification and their combinations.

3. Test Settings

The relational database has 10 tables. The basic tables customer, invoice, target, work, worktype and item represent entities of the application domain. These tables contain the customer information, customer's invoices, the target (or project) where the work is performed, a listing of each work, and a listing of different worktypes with different prices and information about the items used for each work. Relationships between the entities are stored in relationship tables of worktarget, workinvoice, useditem and workhours. These represent many-to-many relationships between entities. Below image shows the database structure as a relational database schema. Arrows illustrate how the tables are associated with each other. For example, the arrow from the invoice to the customer means that customer_id in the invoice table refers to an id in the customer table.

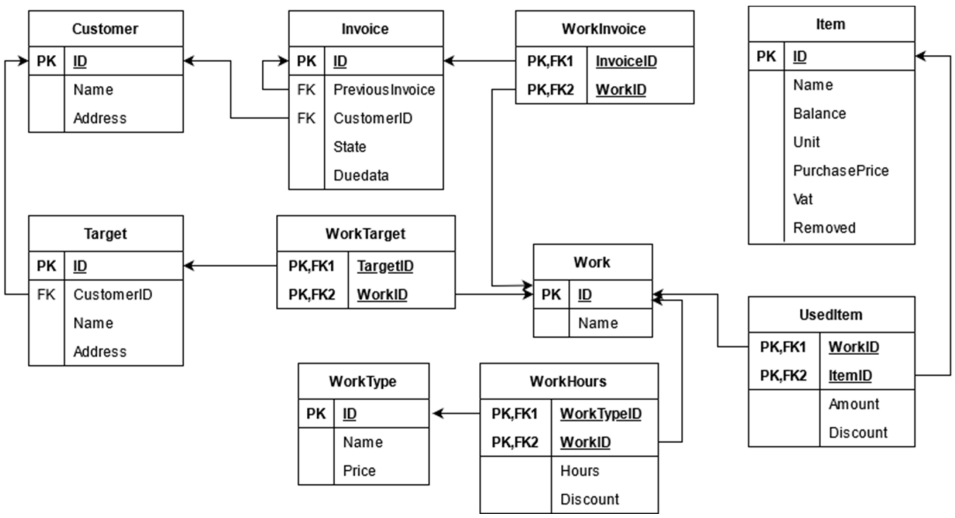

Figure 1. Database structure in relational format.

In the graph database schema, entities are represented as nodes, and relationships as directed edges. Two edges are used to represent many-to-many relationships. Customer, invoice, target, work, and worktype entities are represented as nodes. Relationships PAYS from the customer to invoice, and CUSTOMER_TARGET from the customer to the target, and PREVIOUS_INVOICE from an invoice to another are represented by directed edges, the last being a recursive relationship. WORK_TARGET, WORK_INVOICE, WORKHOURS and USED_ITEM are each represented by two edges. Below image represents the database structure in a graph format. The attributes of nodes and edges are not illustrated.

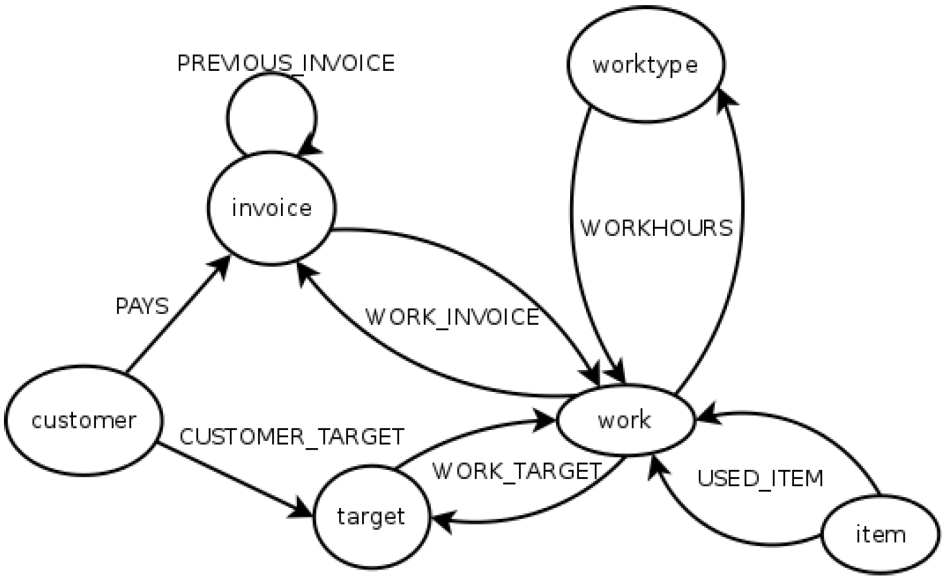

Figure 2. Graph database structure.

The test data follows the schemata of the databases given in previous figures. The used dataset was generated using the test program [6]. Table below shows the numbers of rows/objects generated for the dataset. For each row in the relationship tables of useditem, workhours, workinvoice and worktarget, two respective edges were generated for the Neo4j graph database, as a many-to-many relationship was expressed as a bidirectional relationship, i.e., two edges. The size of relational databases is 214 Mt and the size of Neo4J is 1,12 Gt.

Table 1. The numbers of the generated rows/objects in SQL and Neo4j.

Table/Object	Rows in SQL	Object in Neo4J
Customer	**10,000**	**10,000 nodes**
Invoice	**100,000**	**100,000 nodes**
Item	**100,000**	**100,000 nodes**
Target	**100,000**	**100,000 nodes**
Work	**10,000**	**10,000 nodes**
Workhours	**100,000**	**200,000 edges**
Workinvoice	**1,000,000**	**2000,000 edges**
Worktarget	**1,000,000**	**2,000,000 edges**
Worktype	**100,000**	**100,000 nodes**
UsedItem	**100,000**	**200,000 edges**
Pays	**-**	**100,000 edges**
Customertarget	**-**	**100,000 edges**
Previousinvoice	**100/1000**	**100/1000 edges**

4. Query Tests

The query tests contain queries with different complexities. A query task represents an information need to be fulfilled using a query to the database, and it is implemented in SQL and Cypher queries. Each task involves the following two Cypher queries: basic form and optimized/CALL forms. The query tasks are ordered from simple to complex starting from the work price and the work price with items ending in the invoice prices, and invoice prices for a given customer. Finally, recursive queries combine all the related invoices.

The tasks were chosen as they represent typical information needs that would be executed in the chosen test databases. Finding and calculating invoice related information the primary use for a database, and this is what all the test queries demonstrated. Querying all the information required for invoices leads to complex queries. Simpler queries were included

in order to see how databases perform with different complexities of queries.

Calculating the invoice prices is one of the most important query tasks. The schema does not store invoice prices explicitly. The price must be calculated based on the amount of workhours and the items used. The “price of work” and the “price of work with items” are the subqueries for calculating this price. The queries calculating invoice prices for a given customer add customer information into this task. The recursive queries find all the recursively related invoices given the top-level invoice.

The results of tests are given in tables below. Each query result contains an average time for the query in milliseconds. First table contains the results for the queries related to Tasks 1, 2, 3 and 4. Second table contains the result of recursive queries for Task 5. Second table does not contain results for MySQL 5.1 because MySQL 5.1 does not support those queries. The results are illustrated and further analyzed in the following subsections. Indexed (ind) is the same query on an indexed database. Notably, the performance ranking of different systems varied for different tasks and settings, with the exception that MySQL was always slower than MariaDB.

Table 2. Query performance of the MySQL, MariaDB and Neo4J.

	MySQL 5	MySQL 8	MariaDB	Neo4J	Neo4J CALL
Task 1 (Short Query)					
Avg	**576**	**464**	**486**	**162**	**149**
Avg, ind	**453**	**459**	**472**	**173**	**149**
Task 2 (Long Query)					
Avg	**6550**	**5337**	**5549**	**1868**	**1776**
Avg, ind	**5190**	**5257**	**5293**	**1968**	**1831**
Task 3 (Aggregate Query)					
Avg	**276,935**	**7674**	**7242**	**212,171**	**209,816**
Avg, ind	**251,138**	**7615**	**7117**	**215,053**	**205,198**
Task 4 (Aggregate Query with defined key)					
Avg	**3,938,500**	**5212**	**59**	**33**	**57**
Avg, ind	**3,891,082**	**5227**	**57**	**26**	**55**

Table 3. Query performance in recursive queries.

	MySQL 8	MariaDB	Neo4J	Neo4J Optimized
Recursive Query, 100 entities				
Avg	**7850**	**9152**	**73**	**42**
Avg, ind	**1**	**1**	**72**	**42**
Recursive Query, 1000 entities				
Avg	**79,037**	**92,917**	**331,338**	**2146**
Avg, ind	**2**	**4**	**208,573**	**2127**

5. Conclusions

The present study compared relational database systems (MariaDB and two versions of MySQL) and a graph database system (Neo4j) efficiency using queries with different com- plexities. The results support earlier studies where graph database systems outperformed relational database systems with structurally simple datasets and simple queries. However, with more complex queries new relational database systems outperformed Neo4j.

The significantly better performance of new relational database systems compared to MySQL 5.1 is not surprising as the tested MariaDB and MySQL 8.0.29 versions are 10 years newer, and many developments have occurred during that time. Although MariaDB is based on old MySQL, it offers a different feature set and is completely open source.

One significant change after MySQL 5.1.41 is a change in the default storage engine from MyISAM to InnoDB in version 5.5. InnoDB is used as a default storage engine of MariaDB. The study indicates the extent to which relational database query performance has improved during the last one and half decade.

Neo4j outperformed modern relational database systems in most of the query tasks. Using the best settings of database systems, Neo4J was often at least three times faster than modern relational databases. However, in the task where an aggregated value was calculated for the given entity, Neo4J was 200 times faster than MySQL 8.0.29. In this task, the most essential difference between modern databases also appeared. MariaDB was over 90 times faster than MySQL 8.0.29. In the most complex query task, MariaDB was 29 times faster than Neo4j when indices were used and Neo4J query was optimized. In the same setting, MySQL 8.0.29 was 27 times faster than Neo4J. The role of optimization and indexing played an essential role in performance, especially in the long recursive query. Without indexing, basic Neo4J was the slowest, but the optimized query was the fastest. Indexing changes the situation, i.e., relational database systems outperformed Neo4J. MySQL 8.0.29 performed best. It was over 1000 times faster than the optimized Neo4J query and over 100,000 times faster than basic Neo4J.

The general conclusion is that on the basis of tests with the data set and queries, it cannot be generally concluded which of the database systems possesses the best query efficiency. In other words, the efficiency depends on the complexity of data and queries. Furthermore, query optimization and indexing may play important roles. This means that when choosing a database for an application domain, the query needs must be analyzed carefully beforehand. The results in the present study show how a relational database system is still a good alternative when it comes to performance compared with an NoSQL graph database.

References

1. Khan, W.; Ahmad, W.; Luo, B.; Ahmed, E. SQL Database with physical database tuning technique and NoSQL graph database comparisons. In Proceedings of the 2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC), Chengdu, China, 15–17 March 2019; pp. 110–116.
2. Holzschuher, F.; Peinl, R. Performance of graph query languages: Comparison of cypher, gremlin and native access in Neo4j. In Proceedings of the Joint EDBT/ICDT 2013 Workshops, Genova, Italy, 18–22 March 2013.
3. Khan, W.; Shahzad, W. Predictive performance comparison analysis of relational & NoSQL graph databases. Int. J. Adv. Comput. Sci. Appl. 2017, 8, 523–530.
4. Vicknair, C.; Macias, M.; Zhao, Z.; Nan, X.; Chen, Y.; Wilkins, D. A comparison of a graph database and a relational database: A data provenance perspective. In Proceedings of the 48th Annual Southeast Regional Conference, Oxford, MI, USA, 15–17 April 2010; pp. 1–6.
5. Batra, S.; Tyagi, C. Comparative analysis of relational and graph databases. Int. J. Soft Comput. Eng. (IJSCE) 2012, 2, 509–512.
6. GitHub. InvoicingDBTestBench Repository. Available online: https://github.com/homebeach/InvoicingDBTestBench (accessed on 13 December 2020).
7. DB-Engines. Available online: https://db-engines.com/ (accessed on 20 June 2022).
8. Tongkaw, S.; Tongkaw, A. A comparison of database performance of MariaDB and MySQL with OLTP workload. In Proceedings of the IEEE Conference on Open Systems (ICOS), Langkawi, Malaysia, 10–12 October 2016.
9. Difallah, D.E.; Pavlo, A.; Curino, C.; Cudre-Mauroux, P. Oltp-bench: An extensible testbed for benchmarking relational databases. Proc. VLDB Endow. 2013, 7, 277–288.
10. Shalygina, G.; Novikov, B. Implementing common table expressions for MariaDB. In Proceedings of the 2nd Conference on Software Engineering and Information Management (SEIM-2017), St. Petersburg, Russia, 21 April 2017.
11. Stanescu, L. A Comparison between a Relational and a Graph Database in the Context of a Recommendation System. In Proceedings of the 16th Conference on Computer Science and Intelligence Systems, Online, 2–5 September 2021.
12. Sholichah, R.; Jayanty, M.I.; Andry, A. Performance Analysis of Neo4j and MySQL Databases using Public Policies Decision Making Data. In Proceedings of the 7th International Conference on Information Technology, Computer, and Electrical Engineering (ICITACEE), Semarang, Indonesia, 24–25 September 2021.
13. Cheng, Y.; Ding, P.; Wang, T.; Lu, W.; Du, X. Which category is better: Benchmarking relational and graph database management systems. Data Sci. Eng. 2019, 4, 309–322.