Graph vs Relational: A Comparative Database Study

Project Report

Group Members:
Rajni Hiroshima
Alireza Hoseinpour
Emmanuel Agbeli
Rahubadde De Silva
Cadence Litteral
Bowling Green State University - Department of Computer Science

September 25, 2025

Abstract

This study presents a comparative analysis of Neo4j and MySQL using two representative datasets: the Northwind dataset, modeling business transactions, and the Movie Recommendation dataset, modeling highly connected user–movie interactions. We benchmarked query performance across six categories, measured lines of code (LOC), and evaluated user-friendliness. Results show that Neo4j consistently outperforms MySQL in relationship-intensive queries, particularly in multi-hop traversals and recommendation-based queries, while MySQL is more efficient for simple lookups and transactional queries. LOC analysis further highlights Neo4j's conciseness, though SQL remains easier to interpret due to its natural language–like syntax. These findings suggest that the choice between Neo4j and MySQL should depend on the application domain: Neo4j for graph-centric workloads and MySQL for structured, transaction-oriented tasks.

Keywords: Graph databases, Relational databases, Neo4j, MySQL, Performance benchmarking

1 Introduction and Problem Statement

The choice between graph and relational databases directly affects application performance, data modeling flexibility, and query expressiveness. Relational systems such as MySQL remain central to traditional applications, offering stable tools, standardized SQL interfaces, and well-structured schemas [1]. In contrast, graph databases like Neo4j are optimized for connected data, using a property graph model and the Cypher query language to efficiently capture and traverse complex relationships. Choosing the right paradigm is therefore critical when balancing scalability, efficiency, and ease of use.

To explore these trade-offs, this study compares Neo4j and MySQL using two well-established datasets: the *Northwind* dataset, which models a retail business with customers, orders, products, and suppliers, and the *Movie Recommendation* dataset, which focuses on users, movies, and their rating relationships. These datasets were chosen to represent distinct domains—structured business transactions versus relationship-heavy recommendation systems—thereby allowing us to assess database behavior under different workload characteristics. Both datasets were accessed through Neo4j Desktop and into MySQL using MySQL Workbench, ensuring consistency across environments.

Our primary objective is to benchmark query performance across six categories of queries and determine which database better suits each dataset, ultimately drawing conclusions from these findings. In addition, we evaluate the lines of code required to express queries and conduct a user-friendliness analysis to provide a more holistic comparison between Neo4j and SQL.

2 Methodology and Implementation

In this section of the report, we discuss the implementation details and method we adapted in our work.

2.1 Database Systems Selection

To analyze the differences between graph and relational database paradigms, we selected Neo4j and MySQL as representative systems for our comparative study. Neo4j represents a leading native graph database that stores data using a property graph model. The system employs index-free adjacency, where each node maintains direct references to its adjacent nodes, enabling efficient graph traversals independent of database size. Neo4j uses the Cypher query language, which provides intuitive pattern matching syntax for expressing complex graph queries. The system is optimized for relationship-heavy workloads and supports ACID transactions while maintaining high performance for connected data scenarios. In contrast, MySQL represents a mature relational database management system that organizes data in tables with predefined schemas. The system utilizes B-tree indexes and a cost-based query optimizer to efficiently execute SQL queries. MySQL excels at structured data operations, aggregations, and analytical workloads through its comprehensive SQL implementation. The system provides robust ACID compliance, extensive tooling support, and proven scalability for traditional business applications.

2.2 Experimental Environment Setup

Our experimental setup involved configuring both database systems to ensure fair comparison conditions. For Neo4j, we utilized Neo4j Desktop (version 2.0.4), which served as our primary interface for query execution and performance monitoring. The MySQL configuration involved a local installation of MySQL Server using default settings to represent typical deployment scenarios. We utilized MySQL Workbench (version 8.0.30-0ubuntu0.20.04.2) as our client interface for query development and execution, ensuring we had access to execution plan analysis and performance metrics.

2.3 Dataset Selection and Characteristics

For our comparative analysis, we selected two well-established datasets that represent different application domains and data complexity patterns.

2.3.1 Movie Recommendations Dataset

The Movie Recommendations dataset represents a social network scenario with complex many-to-many relationships between users, movies, actors, directors, and genres [4]. The statistics for the Movie Recommendations dataset are summarized in Table 3b. Table 1 shows both the Neo4j and SQL representations of this dataset.

Table 1: Recommendation Dataset: Neo4j vs. SQL Representation. Here, (N) denotes *nodes* and (R) denotes *relationships* in the Neo4j graph model.

Neo4j Representation	SQL Representation
Movie (N)	movies(id: int (PK), budget: bigint, countries: text, imdbId: text, imdbRating: double, imdbVotes: int, languages: text, movieId: int, plot: text, poster: text, released: text, revenue: bigint, runtime: int, title: text, tmdbId: int, url: text, year: int)
Person (N)	persons(id: int (PK), born: text, bornIn: text, died: text, imdbId: text, name: text, poster: text, tmdbid: text, url: text, actor: int, director: int)
User (N)	users(id: int (PK), userId: int, name: text)
Genre (N)	genre(id: int (PK), name: text)
ACTED_IN (R)	actedIn(person_id: int (PK), movie_id: int (PK))
DIRECTED (R)	directed(person_id: int (PK), movie_id: int (PK))
IN_GENRE (R)	inGenere(movie_id: int (PK), genre_id: int (PK))
RATED (R)	ratings(user_id: int (PK), movie_id: int (PK))

2.3.2 Northwind Trading Dataset

The Northwind dataset represents a traditional business scenario with customers, orders, products, and suppliers, testing both systems' capabilities in handling typical e-commerce and inventory management queries [3]. The statistics for the Northwind dataset are summarized in Table 3a. Table 2 shows both the Neo4j and SQL representations of this dataset.

Table 2: Northwind Dataset: Neo4j vs. SQL Representation. Here, (N) denotes nodes and (R) denotes relationships in the Neo4j graph model

Neo4j Representation	SQL Representation
Category (N)	categories(categoryID: int (PK), categoryName: text, description: text, picture: text)
Customer (N)	customers(customerID: text (PK), companyName: text, contactName: text, contactTitle: text, address: text, city: text, region: text, postalCode: text, country: text, phone: text, fax: text)
Order (N), PURCHASED (R)	orders(orderID: int (PK), customerID: text, employeeID: int, orderDate: date, requiredDate: date, shippedDate: date)
Product (N), SUPPLIES (R), PART_OF (R)	products(productID: int (PK), productName: text, supplierID: int, categoryID: int, quantityPerUnit: text, unitPrice: double, unitsInStock: int, unitsOnOrder: int, reorderLevel: int, discontinued: int)
Supplier (N)	suppliers(supplierID: int (PK), companyName: text, contactName: text, contactTitle: text, address: text, city: text, region: text, postalCode: text, country: text, phone: text, fax: text, homePage: text)
ORDERS (R)	order-details(orderID: int, productID: int, unitPrice: double, quantity: int, discount: int)

Entity / Relationship	Count
Customers	91
Products	77
Product Categories	8
Suppliers	29
Orders	830
Order Items (across all orders)	2,155

(a) Northwind dataset

Entity / Relationship	Count
Movies	9,125
Persons (actors and directors)	19,047
Users (for rating analysis)	671
Genres	20
ACTED_IN relationships	35,910
DIRECTED relationships	10,007
User–Movie ratings (RATED relationships)	100,004

(b) Movie Recommendations dataset

Table 3: Dataset statistics for Northwind and Movie Recommendations.

2.4 Query Categories and Implementation

To evaluate database performance and usability, we designed a set of representative queries spanning six categories: simple lookups, one-hop and two-hop relationships, aggregations, time-based analysis, and recommendations. Two of these categories (simple lookups and one-hop relationships) include a single query each due to their relative simplicity, while the remaining four categories include two queries each to capture more diverse scenarios. The full list of implemented queries is summarized in Table 4.

Table 4: Implemented Queries Across Both Datasets

ID	Category	Query Implemented
		Movie Recommendations Dataset
M1	Simple Lookup	Find person by name
M2	One-hop Relationship	Find movies by actor
M3	Two-hop Relationship	Find common movies between two actors
M4	Two-hop Relationship	Find co-actors who worked together
M5	Aggregation	Count movies by genre
M6	Aggregation	Top 10 most prolific actors
M7	Time-based Analysis	High-rated action movies from the 1990s
M8	Time-based Analysis	Genre popularity over time
M9	Recommendation	Recommend movies based on user's rated genres
M10	Recommendation	Find similar users based on common ratings
		Northwind Trading Dataset
N1	Simple Lookup	Find a customer by name
N2	One-hop Relationship	Find all products supplied by a supplier
N3	Two-hop Relationship	Find all orders made by a specific customer and the products ordered
N4	Two-hop Relationship	Find suppliers of products in a category
N5	Aggregation	Count products by category
N6	Aggregation	Top 5 customers by number of orders
N7	Time-based Analysis	Orders placed in 1997 (with customer)
N8	Time-based Analysis	Monthly order counts for 1997
N9	Recommendation	"Bought together" with a seed product ('Chai')
N10	Recommendation	Customers similar to 'Maria Anders' (shared products)

2.5 Performance Measurement Methodology

For each experiment, every query was executed five times on both Neo4j and MySQL under identical conditions. To further ensure fairness, system caches were cleared between different query categories, and all tests were performed within the same hardware and software environment.

3 Experimental Results and Analysis

We implemented 10 queries across six categories (four categories include two queries each, while two categories have one). Runtimes are averaged over repeated trials; for categories with two queries, we report the mean of their per-query averages.

3.1 Movie Recommendation Performance

Results for the Movie recommendation dataset are presented *at the category level* in Table 5 and Figure 1b. The corresponding *per-query* breakdown (M1–M10) is reported in Table 7b. Neo4j outperforms MySQL in five of the six categories. The gains are especially pronounced on relationship-heavy workloads: simple lookups improve from 23 ms to 2 ms, one-hop traversals from 4 ms to 2 ms, two-hop traversals from 12 ms to 3 ms, and recommendation queries from 439 ms to 10 ms. Aggregation is also faster on Neo4j. The sole exception is temporal analysis, where MySQL is faster. Neo4j's superior performance can be attributed to the relationship-heavy nature of the dataset, where Neo4j's graph-native model and efficient traversal capabilities provide a natural advantage over relational joins in MySQL.

Table 5: Query Performance Comparison by Category for the Recommendation dataset

Query Category	MySQL (ms)	Neo4j (ms)	Improvement
Simple Lookups	23	2	91.3% faster (Neo4j)
One-hop Relationships	4	2	50.0% faster (Neo4j)
Two-hop Relationships	12	3	75.0% faster (Neo4j)
Aggregation & Analytics	62	39	37.1% faster (Neo4j)
Temporal Analysis	28	42	33.3% faster (MySQL)
Recommendation Queries	439	10	97.7% faster (Neo4j)

3.1.1 Northwind Performance

Results for the Northwind dataset are presented *at the category level* in Table 6 and Figure 1a. The corresponding *per-query* breakdown (N1–N10) is reported in Table 7a. Since Northwind schema is more traditionally tabular, MySQL excels on light point and one-hop queries: simple lookups and one-hop relationships. It also leads on the recommendation queries. Neo4j consistently shows an advantage as query complexity increases, particularly when multiple joins are required in SQL (i.e., two-hop and temporal analysis).

Table 6: Query Performance Comparison by Category for Northwind dataset

Query Category	MySQL (ms)	Neo4j (ms)	Improvement
Simple Lookups	0.5	2	75.0% faster (MySQL)
One-hop Relationships	0.8	2	60.0% faster (MySQL)
Two-hop Relationships	3	2	33.3% faster (Neo4j)
Aggregation & Analytics	2	2	Equal performance
Temporal Analysis	28	4	85.7% faster (Neo4j)
Recommendation Queries	2	5	60.0% faster (MySQL)

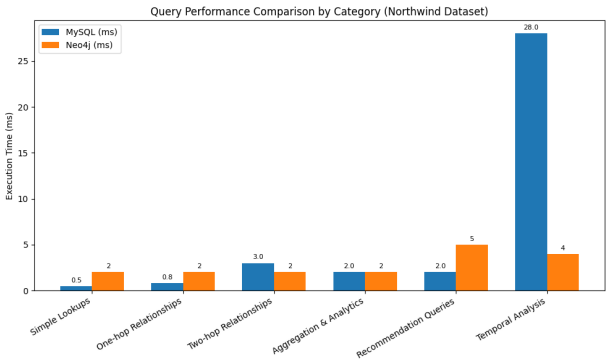
Query ID	MySQL (ms)	Neo4j (ms)	Improvement
N1	0.5	2	75.0% faster (MySQL)
N2	0.8	2	60.0% faster (MySQL)
N3	5	3	40.0% faster (Neo4j)
N4	1	2	50.0% faster (MySQL)
N5	1	2	50.0% faster (MySQL)
N6	3	3	Equal performance
N7	2	7	71.4% faster (MySQL)
N8	2	4	50.0% faster (MySQL)
N9	9	3	66.7% faster (Neo4j)
N10	47	6	87.2% faster (Neo4j)

(a) Northwind dataset

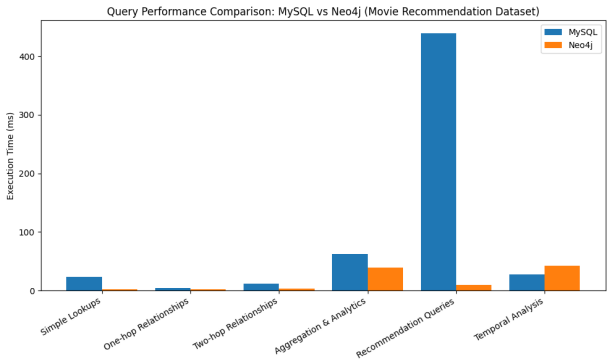
Query ID	MySQL (ms)	Neo4j (ms)	Improvement
M1	23	2	91.3% faster (Neo4j)
M2	4	2	50.0% faster (Neo4j)
M3	17	3	82.4% faster (Neo4j)
M4	7	3	57.1% faster (Neo4j)
M5	35	7	80.0% faster (Neo4j)
M6	89	71	20.2% faster (Neo4j)
M7	38	32	15.8% faster (Neo4j)
M8	53	53	Equal performance
M9	857	10	98.8% faster (Neo4j)
M10	21	10	52.4% faster (Neo4j)

(b) Recommendation dataset

Table 7: Per-query performance comparison across the two datasets.



(a) Query performance for Northwind dataset.



(b) Query performance for Recommendation dataset.

Figure 1: A side-by-side visual comparison of query performance across two datasets.

3.2 Lines of Code (LOC) Analysis and User Friendliness Implications

The LOC comparison in Table 8 highlights a consistent trend favoring Neo4j in terms of query conciseness. For the Northwind dataset, Neo4j queries generally required fewer lines than their SQL counterparts, with especially pronounced differences in Two-Hop and Recommendation queries (e.g., N3, N4, N9, N10). For the Movie Recommendation dataset, the LOC advantage of Neo4j was again evident, with especially

pronounced differences in the same categories as Northwind (Two-Hop relationships and Recommendation queries; e.g., M3, M4, M9, M10). Overall, Neo4j offered a more compact query syntax across all categories, while SQL, though more verbose, was easier to work with due to its closer resemblance to natural language.

(a) Northwind Query Line Counts

Query ID	Neo4j LOC	SQL LOC
N1	2	3
N2	2	4
N3	4	7
N4	3	6
N5	3	5
N6	4	6
N7	4	5
N8	5	5
N9	5	9
N10	5	11
Setup Count	37	61

(b) Movie Recommendation Query Line Counts

Query ID	Neo4j LOC	SQL LOC
M1	1	1
M2	2	5
M3	2	7
M4	3	9
M5	3	5
M6	4	7
M7	4	8
M8	12	15
M9	6	14
M10	4	9
Setup Count	41	80

Table 8: Lines of code (LOC) comparison for Neo4j and SQL queries across Northwind and Movie Recommendation datasets.

4 Conclusions

In this study, we compared Neo4j and MySQL across two datasets with different characteristics to evaluate their performance, usability, and query expressiveness. Our experiments showed that Neo4j consistently outperformed MySQL on relationship-heavy queries, particularly in the Movie Recommendation dataset, where traversals and recommendation queries benefited greatly from Neo4j's graph-native model and efficient traversal mechanisms. In contrast, MySQL proved more efficient for simpler, tabular operations in the Northwind dataset, reinforcing its strength in traditional business and transactional scenarios.

The lines of code (LOC) analysis further emphasized these differences. Neo4j queries were generally shorter and more concise, especially for multi-hop and recommendation queries, where SQL required significantly more code to achieve the same functionality. At the same time, SQL remained easier to interpret for many users due to its resemblance to natural language, making it a practical choice when readability and familiarity are important factors.

Taken together, our results suggest that the choice between Neo4j and MySQL should be guided by the nature of the application. Neo4j is better suited for highly connected, relationship-intensive workloads where graph traversals are frequent, while MySQL continues to excel in structured, tabular domains that rely on aggregations and straightforward analytical queries. In practice, both paradigms have clear strengths, and selecting the appropriate system requires balancing performance, ease of use, and the data model best aligned with the problem domain.

References

[1] Vicknair, C., Macias, M., Zhao, Z., Nan, X., Chen, Y., & Wilkins, D. (2010). A comparison of a graph database and a relational database: a data provenance perspective. *Proceedings of the 48th Annual Southeast Regional Conference*, 1-6.

[2] De Virgilio, R., Maccioni, A., & Torlone, R. (2013). Converting relational to graph databases. *First International Workshop on Graph Data Management Experiences and Systems*, 1-6.

[3] Northwind Datasets https://github.com/neo4j-graph-examples/northwind

[4] MovieRecommendationDatasets https://github.com/neo4j-graph-examples/recommendations

[5] MySQL Documentation https://dev.mysql.com/doc/

A Complete Code Implementations

source code to **Project**

B Peer Assessment Report

Group Member Contributions

Member	Contribution (%)	Named Contribution
Alireza Hoseinpour	20	SQL schema, Neo4j graph representation, query generation, result analysis, report, slides
Emmanuel Agbeli	20	SQL schema, Neo4j graph representation, query generation, result analysis, report, slides
Rajni Hiroshima	20	SQL schema, Neo4j graph representation, query generation, result analysis, report, slides
Rahubadde De Silva	20	SQL schema, Neo4j graph representation, query generation, result analysis, report, slides
Cadence Litteral	20	SQL schema, Neo4j graph representation, query generation, result analysis, report, slides