



Database Design for Recommender Systems

(Masters Project)

Supervisor: Diptish Dey

Student: Priyam Singh
(500926036)

Note:

*This work is accomplished in cooperation with AI Systems BV

Table of Contents

1.	Introduction	4
1.1.	Background Information.....	4
1.2.	Problem Statement	4
1.3.	Research Objectives	4
1.4.	Research Questions	5
1.5.	Research Scope and Structure.....	5
2.	Literature Review	6
2.1.	Background of Literature Study.....	6
2.2.	Literature study.....	6
2.2.1	Approach and Search terms.....	6
2.2.2.	Library Databases.....	6
2.2.3.	Paper Selection	7
3.	Introduction to Database Performance	7
3.1.	Importance of Database Performance in Recommender Systems	7
3.2.	Key Performance Metrics	7
3.3.	CRUD Operations and Memory Usage	8
3.4.	Comparative Analysis of Database Types Based on Performance Metrics	9
3.5.	Connecting Database Types with RS Performance.....	10
4.	SQL Databases in Recommender Systems	10
4.1.	Introduction to SQL Databases.....	10
4.2.	Role of SQL Databases in RS	10
4.3.	SQL Database Technologies.....	11
4.4.	Challenges with SQL Databases in RS	11
5.	NoSQL Databases in Recommender Systems	11
5.1.	Introduction to NoSQL Databases	11
5.2.	Advantages of NoSQL in RS	12
5.3.	NoSQL Database Technologies	12
5.4.	Challenges with NoSQL Databases in RS	12
5.5.	Hybrid Databases in Recommender System	13
6.	Graph Database Based RS	13
6.1.	Introduction to Graph Databases	13
6.2.	Comparative Performance Analysis	13
6.3.	Insights on Database and Algorithm Compatibility	14
6.4.	Similarity Calculation in Graph Databases.....	14
6.5.	Jaccard Similarity	14

6.6.	Rationale for Method Selection:	16
7.	Research Methodology	16
7.1.	Introduction to Research Methodology	16
7.2.	Datasets and System Specifications	16
7.3.	Database Performance Evaluation	18
7.3.1.	Overview of Database Evaluation Metrics	18
7.3.2.	PostgreSQL	18
7.3.3.	MongoDB	18
7.3.4.	Cassandra	18
7.3.5.	Redis	19
7.3.6.	Neo4j	19
7.4.	Neo4j-based Recommender System	19
7.4.1.	Introduction	19
7.4.2.	Developing a Graph-Based RS	19
7.4.3.	Neo4j Sandbox Setup and Connection	20
7.4.4.	Importing Dataset Algorithm Implementation	20
7.4.5.	Movie-Movie Similarity	20
7.4.6.	User-Based Recommendation using Jaccard Similarity	20
8.	Results, Analyses, and Tool Performance	21
8.1.	Performance Measurement of Database Operations	21
8.1.1.	Introduction	21
8.1.2.	Creation of Database Based on Dataset Features	21
8.1.3.	Inserting Records into Created Databases	21
8.1.4.	Reading Records	24
8.1.5.	Updating Records	25
8.1.6.	Deleting Records	26
8.1.7.	CRUD Insights	27
8.2.	Neo4j-based Recommender System	29
8.2.1.	Introduction	29
8.2.2.	Movie – Movie Similarity	29
8.2.3.	User-Based Recommendations using Jaccard Similarity	29
8.2.4.	Conclusion	31
9.	Insights on Database Suitability	32
9.1.	Consolidation of Research Findings	32
9.2.	Recommendations for Database Design	34
9.3.	Limitations of Research	35

9.3.1.	Limited Scope of Performance Metrics and Applications.....	35
9.3.2.	Evolving Database Technologies.....	35
9.3.3.	Sample Size and Testing Environment.....	35
9.3.4.	System Configuration Discrepancies	35
9.3.5.	Tool and Methodology Limitations.....	35
9.4.	Future Directions and Best Practices.....	36

1. Introduction

1.1. Background Information

In the modern era, the growth of online marketplaces, social networks, and collaborative platforms has created massive amounts of data, making it particularly challenging for users to filter out relevant information (Deldjoo et al., 2020; Vaidya & Khachane, 2017). This challenge has created a need for Recommender Systems (RS) to help users navigate the vast amount of information and available choices on the internet (Lex et al., 2021). Recommender Systems use various algorithms to suggest products and services to users based on previous user choices, which can be implicit (by monitoring user behavior) or explicit (like user ratings) (Bobadilla et al., 2013).

The focus of this research is to evaluate the performance of different database management systems (DBMS) for storing and processing data for RS used at AI Systems BV. Some DBMSs under consideration are PostgreSQL, MongoDB, Cassandra, Redis, and Neo4j. Relational databases have traditionally been used in RS, but due to the comparative advantage of NoSQL databases (Truică et al., 2015), their use has become more prominent. Currently, many RS use NoSQL databases or a hybrid of both types. Considering these developments, the choice of an optimal database for RS requires thorough analysis.

1.2. Problem Statement

The role of implementing Recommender Systems is pivotal in various industries (Jannach et al., 2021). AI Systems BV, an AI Based Recommender System (RS) start-up, faces a challenge in identifying the most optimal DBMS for their Recommender Systems applications. This research aims to address this challenge by providing a sophisticated analysis on DBMSs to eventually suggest a solution that ensures Recommender Systems can handle large datasets, have a fast response time, and support complex algorithms (Osadchiy et al., 2019).

Despite the advancements in RS, AI Systems BV struggles with performance issues related to their current DBMS setup. The specific problem is to determine which DBMS—SQL or NoSQL—best meets the performance requirements of their RS (Taipalus et al., 2023). This involves evaluating these databases' performance metrics such as latency, memory usage, scalability, availability, etc. (Biswas & Liu, 2022b). By addressing this problem, AI Systems BV can optimize their RS to deliver better performance and user satisfaction.

Based on the above parameters, performance evaluation of different models of RS will be done that showcases the suitability of DBMS to the current algorithmic model. Since there are many algorithms used, there can also be one database model that suits more than one algorithm or maybe more (Abramova et al., 2014) (Kipf & Welling, 2016).

1.3. Research Objectives

The primary objective of this research is to understand the impact of database architecture choices on the performance of Recommender Systems. This will be accomplished in three steps:

- **Conducting a Comprehensive Literature Review:** Recognize the various database design possibilities and understand their theoretical underpinnings.
- **Evaluating Database Performance:** Based on the literature review, identify a database that is optimized for the performance of RS.

- **Developing a Demonstrator:** Materialize these theoretical constructs into a demonstrator to test the performance of different types of databases.

Through this approach, the research aspires to provide an understanding of the association between database architecture and RS performance. In the end, actionable insights will be provided to AI Systems BV to refine their RS models for overall performance improvement (Almonte et al., 2021).

1.4. Research Questions

The study aims to investigate the performance of different database designs for RS. SQL and NoSQL databases will be evaluated and compared with each other on various parameters. The goal of the study is to find how database architecture selection for RS will impact its performance under varying algorithms and application contexts.

The research will address several sub-questions that will help to analyse RS performance with varying database architecture. The metrics can be used to compare their performance. It will also provide suggestions based on different RS models and frameworks.

Main Research Question – *“How does database architecture selection impact RS performance, considering varying algorithms and application contexts?”*

Sub-Questions:

- What types of databases are typically used in Recommender Systems and how do they compare with each other?
- Which metrics could be used to compare their performance considering different types of RS algorithms?
- How could their performance be demonstrated by developing a tool?
- Which insights could be drawn about the suitability and performance of these different combinations of databases and algorithms?

1.5. Research Scope and Structure

This research aims to determine the optimal Database Management System (DBMS) type for Recommender Systems (RS), focusing on evaluating the performance of SQL databases (e.g., PostgreSQL) and NoSQL databases (e.g., MongoDB, Cassandra, Redis, Neo4j). The evaluation will be based on key performance metrics such as latency, memory usage, scalability, and availability. The study will involve conducting performance testing using Create, Read, Update, and Delete (CRUD) operations to measure these metrics across different database types. Additionally, a Neo4j graph-based recommender system will be developed to highlight the advantages and potential of graph databases for RS applications.

By assessing the trade-offs between SQL and NoSQL databases, this research seeks to provide a thorough understanding of how database architecture impacts RS performance under varying algorithms and application contexts. The findings will offer actionable insights for AI Systems BV, helping them refine their RS models for improved overall performance and efficiency.

2. Literature Review

Chapter 2, 3, 4, 5, and 6 navigate through the literature to uncover the effects of database architecture selection on RS performance. It sets the stage for an in-depth analysis, aimed at comparing different DBMS types on distinct database types for RS.

2.1. Background of Literature Study

Relational databases like MySQL, PostgreSQL, etc have been the baseline for structured data storage and retrieval. However, the advent of Big Data enabled us to understand complex user-item interactions. These interactions are the basis for getting recommendations from an RS. Another important factor is boosting the speed of RS in an already competitive market. This challenge led to the use of NoSQL databases like MongoDB, Cassandra, Redis, Neo4j, etc. in RS as they offered greater flexibility for unstructured data. The literature study aims to discover optimal databases in context of RS by understanding their strengths and weaknesses.

2.2. Literature study

2.2.1 Approach and Search terms

A careful approach was adopted to search for relevant information in academic literature. The search terms were focused on database design considerations for RS. These are the search terms that were used to find articles -

Formulated search terms for research
Database Architecture Comparison in RS
Impact of NoSQL databases on RS efficiency
CRUD analysis on MongoDB and Cassandra
Recommender System AND Database Architecture
PostgreSQL and Redis Latency Comparison
Database Design for Recommender Systems
Comparing Latency of SQL and NoSQL Databases
Recommender System Database
Knowledge Graph-Based Recommender System
Scalability in SQL and NoSQL Databases
Graph database AND RS scalability

Table 1: Formulation of search terms

2.2.2. Library Databases

The literature search targeted databases that only offer high-quality, peer-reviewed articles. It was made sure to refer to articles from high-tier Q1-level journals. However, many articles from lower-level journals were also referred to get specific information. These databases were used for searching relevant articles:

Consensus: It is an AI-powered tool helpful in finding relevant research papers based on research questions. This enables users to view the abstract of the article without explicitly opening it. It consists of a lot of journals with their ratings. So, it becomes relatively easy to discover high-impact articles.

ScienceDirect: Used as the primary source for this research, as it provides access to many Q1 and lower-level journals. ScienceDirect is particularly strong in the fields of technology and business. Its selection is due to its extensive collection of high-impact factor articles relevant to digital innovation and business strategy.

Hogeschool van Amsterdam (HvA) Databases: Gives access to journals and papers that are not directly accessible through Science Direct or any other database. The HvA databases cover a wide range of articles, in numerous sectors including business, digital innovation, and aviation-related articles.

2.2.3. Paper Selection

The process of paper selection was carried out carefully. Initially, a pool of academic papers was chosen which were surveyed based on their quality and recency. After a thorough comparison, some of the initial papers were dropped and selected papers were eventually chosen to be included in the report. The selected papers covered the specific topic this research aims to focus on. This ensured a solid foundation for the research with a broad spectrum of perspectives.

3. Introduction to Database Performance

3.1. Importance of Database Performance in Recommender Systems

For RS, the efficiency and responsiveness of the underlying database management system (DBMS) are paramount. As RS rely heavily on large datasets to provide accurate and timely recommendations. The performance of the database directly influences the overall effectiveness of the system. Understanding and optimizing database performance is crucial for ensuring that RS can handle the demands of modern applications, which include real-time data processing and the ability to scale with increasing user interactions.

3.2. Key Performance Metrics

Evaluating the performance of databases involves considering several key metrics that impact the efficiency and reliability of RS. The following metrics are essential for a comprehensive performance assessment:

Evaluating Latency

Latency is the time taken to complete a specific operation within the database. In RS, low latency is crucial for providing real-time recommendations to users. NoSQL databases like MongoDB exhibit lower latency in certain operations compared to SQL databases (Filip & Čegan, 2020).

Memory Usage

Memory usage is the amount of hard disk or RAM space used to perform a specific operation. Efficient memory usage is vital for managing large datasets without compromising system performance. Graph databases like Neo4j, for instance, may exhibit higher memory usage due to their ability to handle complex relationships. This is followed by Document-based Storage systems like MongoDB and Cassandra (Kabakuş & Kara, 2017).

Scalability Measures

Scalability measures the database's capability to handle increasing workloads while maintaining performance. This is crucial for RS that need to accommodate growing user bases and data volumes. Evaluating scalability involves testing the database under progressively higher loads to ensure it can scale horizontally or vertically as required (Bjeladinović, 2018).

Availability Metrics

Availability assesses the database's operational performance, ensuring that the RS remains accessible even during high-load conditions or partial system failures. High availability is essential for maintaining user trust and satisfaction, as any downtime can disrupt the recommendation process (Lourenço et al., 2015; Wang et al., 2022).

Having established the importance of key performance metrics such as latency, memory usage, scalability, and availability in Recommender Systems (RS), it is crucial to delve deeper into specific operational metrics. The evaluation of CRUD operations—Create, Read, Update, and Delete—and their impact on memory usage provides a detailed understanding of database performance. This section will explore why measuring these operations is essential and how these metrics can significantly influence the efficiency and reliability of RS.

3.3. CRUD Operations and Memory Usage

For Recommender Systems (RS) to perform optimally, the underlying Database Management System (DBMS) must operate efficiently and reliably. CRUD operations, which stand for Create, Read, Update, and Delete, are fundamental to any database interaction. Evaluating these operations is crucial as they directly impact the system's responsiveness and overall performance. Memory usage, another critical performance metric, measures the amount of RAM or hard disk space utilized during these operations. Efficient memory usage ensures that the system can handle large datasets without compromising speed or reliability (Filip & Čegan, 2020).

Significance of Measuring CRUD Operations

CRUD operations form the backbone of any RS's database interactions. Measuring these operations provides insights into several performance aspects:

Create Operations: This involves inserting new data into the database. In RS, new user interactions or items must be frequently added. The efficiency of create operations affects the speed at which new data can be integrated into the system (Nakhare, 2021). Once the database is created in the Database management system, Insertion of data can take place in 2 ways -

- a) **Batch Insertion:** This method involves grouping multiple insert operations into a single batch, reducing the number of database transactions, and improving performance (Gao et al., 2017).
- b) **Bulk Insertion:** This technique is used to insert large volumes of data in a single operation, significantly reducing the overhead associated with individual inserts. Efficient batch and bulk insertions are essential for handling large datasets quickly and effectively, which is particularly important for RS applications dealing with vast amounts of user and item data (Catapang, 2018).

Read Operations: Reading data from the database is critical for generating recommendations. The latency and speed of read operations influence the system's ability to provide real-time recommendations, which is essential for maintaining user satisfaction (Truică et al., 2015).

Update Operations: Updating existing data ensures that the recommendations are based on the most current information. Efficient update operations are vital for adapting to changing user preferences and new trends (Bjeladinović, 2018).

Delete Operations: Removing outdated or irrelevant data helps in maintaining the database's efficiency and relevance. Efficient delete operations ensure that the database remains clean and performance optimized (Osadchiy et al., 2019).

By measuring the performance of these operations, we can identify potential bottlenecks and optimize the database for faster and more reliable interactions.

Significance of Memory Usage

Memory usage is a critical metric that reflects the efficiency of database operations in handling large datasets. In RS, where the volume of data is substantial, efficient memory management is essential for several reasons:

Performance Optimization: High memory usage can slow down the system and increase latency. By monitoring memory usage, we can ensure that the database operates within optimal limits, thus maintaining high performance (Lourenço et al., 2015).

Scalability: Efficient memory usage allows the system to scale effectively, handling increased loads without significant performance degradation. This is crucial for RS, which must accommodate growing user bases and data volumes (Kabakuş & Kara, 2017).

Cost Management: Memory resources are costly. By optimizing memory usage, we can reduce the operational costs associated with maintaining and scaling the database infrastructure (Wang et al., 2022).

Utilizing CRUD and Memory Metrics

The metrics obtained from measuring CRUD operations and memory usage can be used to make informed decisions about database architecture and optimization strategies. For example, if read operations are found to be slow, indexing strategies can be improved or more efficient data retrieval algorithms can be implemented. Similarly, if memory usage is high during create operations, data compression techniques or more efficient data structures might be employed (Filip & Čegan, 2020).

By systematically analyzing these metrics, we can enhance the performance and reliability of RS, ensuring that they meet the demands of modern applications. This comprehensive understanding enables AI Systems BV to optimize their database setup, thereby improving the overall effectiveness and user satisfaction of their Recommender Systems.

3.4. Comparative Analysis of Database Types Based on Performance Metrics

SQL databases, such as PostgreSQL, generally exhibit lower latency for read and write operations due to their structured nature and optimized indexing mechanisms. However, as data volumes grow, the latency can increase, particularly for complex queries involving joins (Truică et al., 2015). PostgreSQL ensures reliable transaction processing with efficient memory management, although it may consume more memory during peak loads (Nakhare, 2021). Traditionally, SQL databases scale vertically, which can limit their ability to handle massive datasets, but they ensure high availability through replication and failover mechanisms (Filip & Čegan, 2020; Lourenço et al., 2015).

NoSQL databases, such as MongoDB and Cassandra, are designed for low latency in high-throughput environments, making them suitable for real-time RS applications (Truică et al.,

2015). They offer flexible schema designs that can adapt to varying data structures and are built for horizontal scalability, allowing them to handle large-scale data effectively (Györfödi et al., 2022). High availability in NoSQL databases is achieved through distributed architectures and replication, ensuring continuous operation and reliability (Wang et al., 2022).

Graph databases like Neo4j are optimized for traversing relationships between nodes, resulting in lower latency for specific types of queries involving complex relationships (Yi et al., 2017). They often consume more memory due to their advanced querying capabilities, but they provide powerful tools for modelling and querying complex relationships (Guo et al., 2022). Graph databases can scale both horizontally and vertically, though their scalability is often limited by the complexity of the graph. High availability is ensured through replication and clustering, which is crucial for maintaining the reliability of RS (Wu et al., 2022; Zhu et al., 2019).

3.5. Connecting Database Types with RS Performance

Having established the key performance metrics and conducted a comparative analysis of SQL, NoSQL, and graph databases, it is essential to explore how each of these database management system types specifically functions within RS. The subsequent chapters will delve into the roles, advantages, challenges, and technologies associated with SQL, NoSQL, and graph databases in the context of RS. This detailed examination will provide a deeper understanding of their performance and suitability for various RS applications, setting the stage for the practical evaluation and implementation strategies discussed later in the research methodology.

4. SQL Databases in Recommender Systems

4.1. Introduction to SQL Databases

The inception of RS started with Tapestry, the first widely recognized recommender system developed in the early 1990s (Goldberg et al., 1992). Early, RS models used SQL-based databases due to relative ease of use. It was only after the introduction of NoSQL databases such as MongoDB, Cassandra, etc in the 2000s, that the use of NoSQL databases started in RS (Jannach & Jugovac, 2019).

There are 2 important concepts when it comes to SQL database use. Firstly, Normalization which is a multi-step process to organize data into smaller tables. It helps in minimizing redundancy in data while playing a crucial role in ensuring data quality and query performance. This makes SQL databases suitable to be used in RS (Nakhare, 2021).

Secondary, ACID properties of DBMS i.e. Atomicity, Consistency, Isolation, and Durability. These properties ensure that transactions are processed reliably while keeping the data integrity. This is crucial in RS, where transaction management is important to ensure user profiles and preferences (Filip & Čegan, 2020) (Quadrana et al., 2018).

4.2. Role of SQL Databases in RS

SQL databases have been used in RS since its existence. The relational nature of SQL databases has been instrumental in managing structured data for RS. There are few DBMS-based recommender systems architectures, in which a database stores the recommender system data (e.g., interaction data and the recommendation models) while consecutively generating recommendations using SQL queries (Sarwat et al., 2017). In some scenarios, DBMS-based recommender systems show lower latency than those RS that store data separately.

4.3. SQL Database Technologies

There are various SQL Database technologies like PostgreSQL, MySQL, MS SQL Server, etc. MySQL is being employed in a variety of RS because of its ability to scale vertically and its optimization of read-heavy operations (Filip & Čegan, 2020). PostgreSQL is also another popular SQL database as it supports advanced data types and allows the creation of custom functions. Its support for complex queries and transactional integrity enables RS for more dynamic recommendations (Bjeladinović, 2018b).

In this research, the focus will be on PostgreSQL in comparison with other NoSQL Databases to test their performance on CRUD operations. The latency of each of these tests will be compared with other non-relational databases. Studies suggest that PostgreSQL performs better in comparison with MySQL and other NoSQL Databases in terms of latency when performing CRUD (Truică et al., 2015). When comparing the execution time of data mining, data grouping, and restoration of the database – MySQL took more time to perform several operations with different sizes of data in terms of number of records (Solarz & Szymczyk, 2020).

4.4. Challenges with SQL Databases in RS

As the volume of data increased exponentially, the challenge of storing and processing huge volumes of complex datasets arose. Relational databases had a serious limitation in tackling this challenge, as most relational databases can be scaled vertically but not horizontally. Eventually, NoSQL databases came into existence to cater these limitations (Győrödi et al., 2022). With the creation of more diverse data features, it became vital for RS systems to use these features to increase accuracy (Lu et al., 2015). Consequently, the extraction of relevant information became more popular, which boosted the demand for RS systems. This evolution in data storage and processing led to the emergence and adoption of NoSQL databases, which offer a range of advantages over traditional relational databases.

5. NoSQL Databases in Recommender Systems

5.1. Introduction to NoSQL Databases

NoSQL Databases are based on the BASE (Basically, Available, Soft State, and Eventually Consistent) principle that sacrifices data consistency in exchange for high data availability (Abramova et al., 2014) (Chandra, 2015). One key aspect that differentiates NoSQL databases from relational ones is that tables and SQL language are not always used. Another key feature is that these databases are optimized for CREATE and READ operations and, often, they offer reduced functionality for UPDATE and DELETE queries (Truică et al., 2015).

NoSQL databases store data in a variety of ways depending on their design and purpose. MongoDB, for instance, stores data in flexible, JSON-like documents, which allows for nested structures and varying data types within the same collection, providing a high degree of flexibility and ease of use (Grolinger et al., 2013). Cassandra, on the other hand, utilizes a column-family store model, where data is stored in rows and columns but without a rigid schema, allowing for efficient handling of large volumes of data across distributed systems (Lakshman & Malik, 2010). Redis employs a key-value store approach, where data is stored as a collection of key-value pairs, enabling rapid access and manipulation of data (Carlson, 2013). Neo4j, a graph database, stores data in nodes and edges, which represent entities and their relationships, respectively. This structure is particularly effective for applications requiring complex relationship mappings and fast traversal queries (Angles & Gutiérrez, 2018).

5.2. Advantages of NoSQL in RS

NoSQL databases are built to be easily scaled across many servers. This is possible because of their ability of sharding. MongoDB supports auto-sharding where it partitions the data collections and stores the partitions across available servers (Abramova et al., 2014). It is particularly beneficial in RS which handles large, active user bases (Filip & Čegan, 2020).

Unlike SQL databases, NoSQL databases do not require a fixed schema, enabling them to adapt to diverse data types and structures. For RS, it is particularly an advantage as it can adapt to new data sources and user interactions (Bhogal & Choksi, 2015b). Usually, NoSQL databases are faster in comparison with SQL databases in terms of CRUD operations (Truică et al., 2015). It is also an important factor to be considered while using it for RS.

5.3. NoSQL Database Technologies

There are many NoSQL databases like MongoDB, Cassandra, Redis, etc. This paper will focus on MongoDB, Cassandra, and Redis as they are the most widely used and open-source DBMSs available. **MongoDB** is a leading document-oriented database. It allows for dynamic definition of schemas. It stores data in Collections which contains documents. This is especially beneficial for RS as it permits rapid evolution of database schemas without extensive downtime (Győrödi et al., 2022) (Afsar et al., 2022). It also has features like MapReduce that allows for parallel processing of queries and present combined results in the end. This is particularly beneficial for big data applications like RS (Soni & Yadav, 2015).

Cassandra is a key-value database with some features of tabular databases. Its architecture is designed for distributed deployment that inherently supports high throughput and scalability (Abramova et al., 2014). Its data model offers RS the ability to handle large volumes of write operations without significant impact on read performance (Filip & Čegan, 2020).

Redis, an in-memory data structure store while MongoDB and Cassandra are disk-based storage. It excels in handling real-time data and analytics making it ideal for updating and deletion operations in RS. It has simple and straightforward APIs compared to more complex setups of MongoDB and Cassandra (Gupta et al., 2017).

Furthermore, the **CAP theorem**, '*states the distribution system can only provide two out of the 3 guarantees - Consistency, Availability, and Partition tolerance*'. It shows a trade-off in the design and usage of NoSQL databases (Bhogal & Choksi, 2015b). RS architects must carefully evaluate these choices to best align with the specific requirements of their systems.

In conclusion, NoSQL databases discussed above bring in many opportunities and challenges for RS. MongoDB flexible schema design and powerful query capacity compared with Cassandra's robust operations and high throughput. Redis's in-memory architecture that provides rapid, real-time data processing with low latency. Each of them offers unique advantages for Recommender Systems.

5.4. Challenges with NoSQL Databases in RS

With great powers come greater responsibilities same goes with the use of NoSQL Databases in RS. One of the primary challenges with NoSQL databases is that there is a trade-off between consistency, availability, and partition tolerance, as explained by the CAP theorem (Gilbert &

Lynch, 2002). NoSQL databases have their own query language with schema-less design (Corbellini et al., 2017). As they use JSON files instead of SQL this can be particularly challenging. This can be advantageous in many scenarios. It comes with a lack of standardization in query languages across different NoSQL databases can lead to challenges in maintaining and evolving the codebase of RS (Soni & Yadav, 2015).

5.5. Hybrid Databases in Recommender System

Hybrid databases integrate features from both SQL and NoSQL databases, combining the strengths of traditional relational models and modern flexible data structures. In recommender systems, hybrid databases offer a versatile solution that can handle structured relational data and unstructured or semi-structured data, enabling efficient storage and processing of diverse datasets. For instance, a hybrid system like **Microsoft Azure Cosmos DB** supports multiple data models (document, graph, and key-value), allowing a recommender system to store user profiles in a relational format while managing item metadata and interaction logs as documents or key-value pairs (Paz, 2018). This adaptability ensures that the system can handle complex queries and scale effectively as data volume and variety increase.

A practical example is **Couchbase**, which combines the strengths of a key-value store and a document database with query capabilities like SQL (Hubail et al., 2019). In a movie recommendation system, user preferences and ratings can be stored in a key-value format for fast access, while detailed movie metadata, such as genres and descriptions, can be managed as JSON documents. Couchbase's N1QL query language allows for SQL-like queries on JSON data, facilitating complex recommendation algorithms that leverage both structured and unstructured data. This dual capability provides the flexibility to optimize for different workloads and query patterns, enhancing the overall performance and scalability of the recommender system.

Justification for Scope Limitation: Hybrid databases would introduce additional complexity due to their multifaceted nature, potentially diluting the clarity of comparisons and the specific insights this research seeks to deliver.

6. Graph Database Based RS

6.1. Introduction to Graph Databases

Graph databases like Neo4j employ nodes, and edges to represent information. Each node represents an entity. These entities can be anything ranging from a user to a specific item on an e-commerce site. For instance, Huang et al. (2002) demonstrate a graph-based RS for the digital library. It has a 2-layer model i.e. Books and customers. The nodes in both layers are connected by relationships. These relationships are based on purchase history. Along with that, there are also relationships within the Item-Item layer and customer-customer layer. In some cases, these relationships have the directional information and weight that signifies the strength and nature of linkage. The properties provide metadata about the nodes and edges (Yi et al., 2017).

6.2. Comparative Performance Analysis

The performance of graph databases in Recommender Systems (RS) is a critical area of study as it directly influences the efficiency, scalability, and accuracy of the recommendations provided to the users (Zhu et al., 2019). A comparative analysis between graph databases and other database architectures reveals distinct performance characteristics and suitability for various RS scenarios.

6.3. Insights on Database and Algorithm Compatibility

Graph-based RS offers a distinctive advantage in terms of explainability and effectively models complex connections between users and items. It leverages neighbourhood-based approaches like Jaccard similarity (Bag et al., n.d). The graph-based approach allows for the exploration of complex relationships and patterns in data. This can lead to more accurate and personalized recommendations for users (Wu et al., 2022).

6.4. Similarity Calculation in Graph Databases

Graph databases use several methods to calculate similarity between entities like movies and users:

Common Neighbours: Counts the number of shared connections (e.g., users who rated both movies or items both users rated). More shared connections indicate higher similarity.

Jaccard Similarity: Measures similarity by dividing the number of common neighbours by the total unique neighbours. It normalizes the similarity score, useful for both movies and users (Bag et al., n.d).

Cosine Similarity: Represents nodes as vectors in a multi-dimensional space and calculates the cosine of the angle between them. This method captures directional similarity (Takano et al., 2019).

Pearson Correlation: Evaluates linear correlation between interaction patterns, considering mean and standard deviation. Useful for comparing rating patterns (De Winter et al., 2016).

Node Embeddings: Uses algorithms like Node2Vec or GraphSAGE to generate low-dimensional embeddings, capturing complex graph patterns. Similarity is computed in this reduced space (Xu, 2021).

Graph databases, like Neo4j, efficiently handle and query complex relationships within these methods, supporting dynamic and scalable similarity computations.

6.5. Jaccard Similarity

The Jaccard similarity coefficient is a statistical measure used for comparing the similarity between two sets. It is defined as the size of the intersection divided by the size of the union of the sets. For users, the sets could represent items they have interacted with (e.g., watched, purchased, rated).

$$\text{Jaccard Similarity} = \frac{|A \cap B|}{|A \cup B|}$$

Equation 1: Jaccard Similarity

Below is an example scenario which explains the steps to calculate the similarity score for the users U1, U2 and U3. Each set consists of items the user has interacted with.

Example Scenario

1. **Users:** Consider three users U_1 , U_2 , and U_3 .

- U_1 has interacted with items $\{A, B, C\}$.
- U_2 has interacted with items $\{A, C, D\}$.
- U_3 has interacted with items $\{B, E, F\}$.

2. **Calculating Jaccard Similarity:**

- Similarity between U_1 and U_2 :

$$\text{Jaccard}(U_1, U_2) = \frac{|\{A, C\}|}{|\{A, B, C, D\}|} = \frac{2}{4} = 0.5$$
- Similarity between U_1 and U_3 :

$$\text{Jaccard}(U_1, U_3) = \frac{|\{B\}|}{|\{A, B, C, E, F\}|} = \frac{1}{5} = 0.2$$
- Similarity between U_2 and U_3 :

$$\text{Jaccard}(U_2, U_3) = \frac{|\{C\}|}{|\{A, C, D, E, F\}|} = \frac{1}{5} = 0.2$$

Equation 2: Jaccard Similarity Calculation Scenario

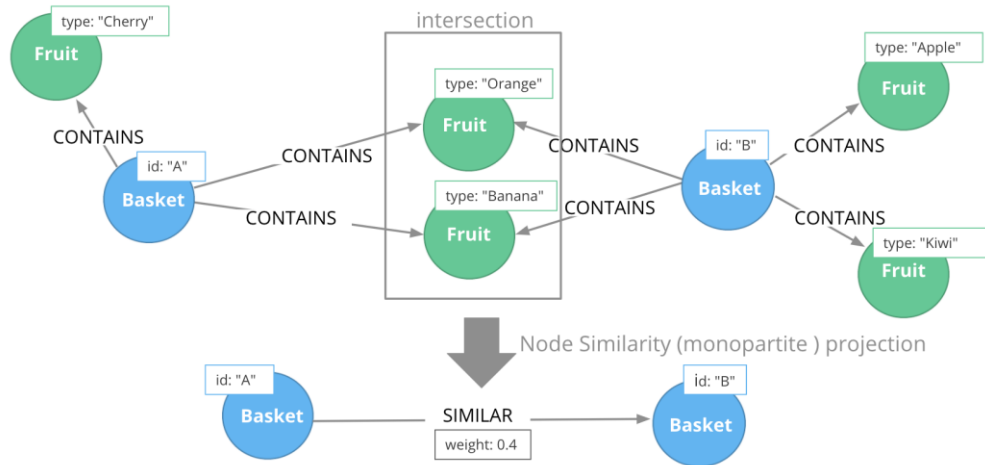


Figure 1: Jaccard Similarity (Neo4j, 2024)

For calculating the similarity score between two users. The intersection of items in both sets is placed at numerator and the union of items in both sets is placed at denominator. Then, both the numerator and denominator are converted into a number. Eventually, the Jaccard Similarity scores are obtained for 2 users that can be used further in the analysis (Bag et al., n.d).

6.6. Rationale for Method Selection:

For movie-movie similarity model, the **common user ratings (Common Neighbours)** method is chosen as it directly measures the number of users who have rated both movies. This approach provides a clear and intuitive metric for similarity based on shared audience preferences, crucial for identifying related movies based on viewing patterns and enhancing recommendation accuracy.

For Neo4j based RS model, **Jaccard similarity** is selected due to its ability to account for both shared and unique interactions, making it effective in identifying how users' interests overlap while providing a normalized score (Bag et al., n.d). This helps in capturing meaningful similarities even when users have diverse interaction histories.

7. Research Methodology

7.1. Introduction to Research Methodology

The Methodology can be broadly divided into 2 subtopics: CRUD Operations on 5 DBMSs and a Neo4j-based recommender system. For CRUD operations part, Retail Rocket and Diginetica Datasets are used. For Neo4j- based recommender system, benchmark dataset of movie lens is used.

The Latency and Memory Usage will be recorded while performing CRUD operations on SQL and NoSQL databases. For that installation of Ubuntu 22.04 will be done on the system. Then choice of the appropriate dataset will be made. The latency of PostgreSQL in comparison with other NoSQL databases (MongoDB, Cassandra, Redis, and Neo4j) will be tested. Then, a Neo4j Graph-based Recommender System will be created for demonstration (Z. Guo & Wang, 2021). In the end, insights from the analysis will be presented. Below the selection process of datasets is discussed.

7.2. Datasets and System Specifications

7.2.1. Selection of Datasets

For the evaluation of CRUD operations, this research employs the Diginetica and Retail Rocket datasets, while the Movielens dataset is used for the graph-based recommender system. Apart from these datasets, other dataset that were considered are **Amazon Reviews Dataset, Yelp Dataset, Netflix Prize Dataset, IMDB Dataset, etc.** The **choice of these datasets is guided by their relevance, diversity, and representation of real-world scenarios** in the domain of Recommender Systems (RS).

Diginetica Dataset: Diginetica is a well-known dataset in the RS community, containing a variety of user interactions with products in an e-commerce environment. It includes features such as session IDs, user IDs, item IDs, category IDs, and timestamps. The relevant CSV file used in this research is “**train-clicks.csv**”, which provide comprehensive data for evaluating CRUD operations due to their structured format and the inclusion of numerous user and item interactions, critical for analyzing the performance of various DBMS (Battle et al., 2020).

Retail Rocket Dataset: Retail Rocket provides data from an online retail environment, including user sessions, product views, and purchases. The dataset includes features such as session IDs, timestamps, item IDs, and event types (view, add to cart, transaction). The

relevant CSV file is “**events.csv**”. This dataset is selected for its rich interaction data, which allows for testing the scalability and efficiency of CRUD operations under high-load conditions typical of online retail platforms.

Movielens Dataset: Movielens is a popular dataset that encompasses user ratings of movies, widely used for testing and benchmarking recommender algorithms. It contains user IDs, movie IDs, ratings, and timestamps. The relevant DAT files used are “**users.dat, ratings.dat and movies.dat**”. This dataset is particularly suited for graph-based RS because it includes explicit ratings that can be modelled as relationships in a graph database. Since, users and movies can be represented as nodes while ratings can be used to connect them. Consequently, facilitating the analysis of graph-based algorithms (Harper & Konstan, 2015)

7.2.2. System Specifications

Ubuntu 22.04 LTS was chosen as the operating system for this research due to its renowned stability and performance. It is a dependable choice for conducting rigorous database performance tests, ensuring consistent management of system resources, which is crucial for benchmarking CRUD operations and graph-based algorithms. Additionally, Ubuntu supports a wide array of development tools, database systems, and libraries, simplifying the setup and execution of various database technologies such as PostgreSQL, MongoDB, Cassandra, Redis, and Neo4j. The extensive community and support available for Ubuntu also provide valuable resources and troubleshooting assistance, ensuring swift resolution of any potential issues, thereby minimizing downtime, and enhancing research productivity (Awan, 2022).

While other operating systems were considered, they were not chosen for the following reasons:

- **Windows 11:** Higher resource overhead and less predictable performance for server-based database tasks. Lacks native support for many open-source database tools (Goyal et al., 2018).
- **macOS Ventura:** Less common in server environments, with a focus on development rather than large-scale database deployment. Higher costs and hardware limitations make it less practical.
- **Fedora 38:** Offers cutting-edge features but has a rapid release cycle, leading to potential instability. Ubuntu's LTS version provides more consistent stability (Lagoze et al., 2005).
- **CentOS Stream:** The rolling-release model can cause stability issues, whereas Ubuntu's regular updates and LTS support ensure a more reliable research environment (Tenaya et al., 2022).

The system is configured with the following specifications: a partitioning scheme that includes **55 GB for the operating system, 77 GB for the hard drive, and 15 GB for other uses**. The machine is equipped with **16 GB of RAM (15.8 GB usable)** and operates on a **64-bit architecture**. This robust setup is designed to Open handle the intensive computational requirements of database performance evaluations, ensuring that the system can effectively manage large datasets and perform complex operations without significant latency or performance degradation.

Having established the system specifications and dataset selection, the subsequent section will delve into the CRUD operations. These operations are fundamental for assessing database performance, and their evaluation will provide insights into the efficiency and scalability of different DBMS configurations within Recommender Systems. The analysis will focus on the performance metrics derived from these operations, offering a comprehensive understanding of how various databases handle essential tasks in real-world scenarios.

7.3. Database Performance Evaluation

7.3.1. Overview of Database Evaluation Metrics

RS depends heavily on user data to generate personalized suggestions. Performance testing of databases in RS based on CRUD operation is crucial to measure its efficiency. Building user profiles, item catalogues, and store user interactions is done by the “Create” operation. For retrieving this data to understand user preferences and behaviours “Read” operation is used. Consequently, “Update” and “Delete” operations are used to modify existing information and remove obsolete data (Li & Manoharan, 2013).

This section delves into a comparative analysis of SQL and NoSQL databases in terms of CRUD operations performance.

7.3.2. PostgreSQL

The connection to PostgreSQL is established using ‘psycopg2’ library in python. A Cursor is created to interact with the database inside ‘create_postgres’ function. The database connections parameter is passed to establish the connection. This cursor is eventually used to create tables and interact with the data using SQL. For measuring the CPU memory usage over time ‘psutil’ library. A function named ‘monitor_resources’ is used to measure the CPU resource utilization.

PostgreSQL scales well vertically with increased hardware resources. It also supports horizontal scaling through partitioning and sharding, making it suitable for handling large datasets typical in RS. High availability in PostgreSQL is achieved through replication and failover mechanisms. Tools like Patroni or native replication features ensure minimal downtime, enhancing reliability for critical applications.

7.3.3. MongoDB

The connection to MongoDB is established using ‘pymongo’ library. A client is created to connect to the locally hosted using default connection string i.e. ‘mongodb://localhost:27017/’. It is passed through MongoClient function. Database in MongoDB is stored in collections. MongoDB Query Language (MQL) is used to query data. Like PostgreSQL, ‘psutil’ library is used to measure CPU memory usage.

MongoDB excels in horizontal scalability with its sharding capabilities, distributing data across multiple servers. This is ideal for large-scale RS implementations. MongoDB ensures high availability through replica sets, providing automatic failover and data redundancy, making it a robust choice for applications requiring reliable performance and minimal downtime.

7.3.4. Cassandra

For Cassandra, ‘cassandra-driver’ library is used to interact with Cassandra in VS code. The default loopback address i.e. ‘127.0.0.1’ for Cassandra is passed in Cluster to start the

‘session’. Once ‘session’ is established, Queries can be passed to perform operations on the database. The Cassandra Query Language (CQL) is used for performing CRUD operations on the database.

Cassandra offers excellent horizontal scalability, capable of handling massive amounts of data with ease. Its peer-to-peer architecture allows seamless addition of nodes, making it highly adaptable for growing datasets. High availability is a key feature of Cassandra, with its replication strategy ensuring no single point of failure, providing reliable performance even under significant load.

7.3.5. Redis

Redis establish connection using the ‘redis’ library. Port number ‘6379’ and db = 0 are the default parameters to connect to the localhost in redis. The data is stored in a key-value pair. Redis Command Line Interface (CLI) is used for interacting with the data. Memory usage is measured using ‘psutil’ library. The default port for redis is ‘6379’.

Redis supports horizontal scaling through clustering, allowing it to handle large datasets by distributing them across multiple nodes. This ensures high performance and quick data access. Redis ensures high availability with its built-in replication and automatic failover mechanisms. Sentinel provides additional high availability and monitoring, making Redis a reliable choice for real-time applications.

7.3.6. Neo4j

For Neo4j connections are established using ‘neo4j’ library. The default connection URI for neo4j is ‘neo4j://localhost:7687’, when using locally hosted database. Graph database driver is used to authenticate the credentials. Once the connection is established. Cypher Qwery Language is used to perform CRUD operations.

Neo4j scales well vertically and is increasingly supporting horizontal scaling with its Causal Clustering, making it suitable for complex RS applications involving large graphs. Neo4j provides high availability through its clustering capabilities, ensuring that the database remains operational even if some nodes fail, thus offering reliability for critical graph data applications.

7.4. Neo4j-based Recommender System

7.4.1. Introduction

Using Neo4j, 2 models were created – The first aims to discover similar movies by counting the number of common users that have given a particular rating to both movies. The second model uses an aggregate of Jaccard Similarity and movies rated to recommend movies to a user.

7.4.2. Developing a Graph-Based RS

Graph databases like Neo4j offer a natural fit for RS due to their structure, which is inherently designed to map and query relationships. The RS leverages this capability to offer recommendations based on a variety of factors, including user behaviour, item attributes, and the connections between them (Huang et al., 2002). To demonstrate the use of graph databases for recommender systems a popular benchmark dataset like movie lens dataset

can be used. A similarity method can be used to get user-user similarity metrics like Jaccard Similarity (Yi et al., 2017). It is a similarity between two sets that can be applied to node neighbourhoods (Bag et al., n.d.).

7.4.3. Neo4j Sandbox Setup and Connection

Neo4j Sandbox is a cloud-based instance of Neo4j that is used to host the RS model. The code uses Graph data science library to connect to the cloud storage of Neo4j. Once the connection is established the code print the version of graph data science library. For the model, the movie lens dataset was considered because it has a clear distinction of files that is be used to create nodes and relationships. Moreover, it is a benchmark dataset where user and movies file can be used to create nodes and ratings is used to create relationships between them.

7.4.4. Importing Dataset Algorithm Implementation

Importing the dataset is done firstly for the user's file that created 6040 customer nodes. Then similarly 3883 movie nodes were created. Since, rating file was large, so it was split into 20 equal batches. The relationships were created using these batches. A total of 1000209 relationships were created.

7.4.5. Movie-Movie Similarity

The calculation of similar movies is by counting the number of common users between 2 movies via rating. The vectors/ relationship is the rating information. The code shows a list of 5 most similar movies to 'Toy Story (1995)' based on common user ratings. The threshold set is that the users should have rated both movies '5'.

7.4.6. User-Based Recommendation using Jaccard Similarity

A Graph Projection named 'MyGraph' is used to create a projection. The projection is a sort of virtual subgraph. Jaccard Similarity is used as a measure to compute similarity between 2 user nodes. The graph collects the neighbours of a user node into a set, two user nodes are considered similar if their neighbour sets are similar. Similarity range from 0 to 1, where 1 indicates two sets are exactly same, 0 means that two sets are completely different.

Similarity score:

$$r_{m,avg} = \frac{1}{\sum s_{user_2}} \sum s_{user_2} r_m$$

where sum runs over all paths
 $(user_1) - [s_{user_2}] - (user_2) - [r_m] \rightarrow (movie)$

Figure 2: Average Rating Weighted by Similarity Score

Based on Jaccard's similarity a new relationship between users is created. The similarity score between them is added as an attribute. In order to recommend movies the user1, first similar users and then the movies they have rated are found. For each movie, an average rating weighted by similarity score is calculated. In implementation, a log of number of paths is added to boost movies that are connected to user1 through multiple such paths. Finally, a score is generated for each movie and the movies with highest scores are recommended.

8. Results, Analyses, and Tool Performance

8.1. Performance Measurement of Database Operations

8.1.1. Introduction

Analysing the output generated from the model is crucial. It helps understand changes in efficiency and resource utilization of each database system. This is particularly important for RS when handling large-scale data.

The Retail Rocket dataset contains events data. It includes customer behaviour data like clicks, add-to-carts, transactions, etc. It is particularly useful for creating RS models. The second dataset used is Diginetica, for the analysis train clicks file is particularly interesting with regards to latency analysis.

8.1.2. Creation of Database Based on Dataset Features

For Creation, Redis and MongoDB are most efficient in terms of time taken and CPU usage for database creation. This can be explained as both doesn't require indexing. Cassandra was least efficient, with high latency and memory usage. Neo4j showed variable performance with high memory usage for Retail Rocket but moderate for Diginetica. The performance of all the databases is relevant as most of them took more time for Retail Rocket since it had more features.

Database	Dataset	Time(s)	CPU Time (s)	Memory Usage (GB)
PostgreSQL	Retail Rocket	0.0126	0.0386	7.5012
	Diginetica	0.0039	0.0296	8.8888
MongoDB	Retail Rocket	0.0015	0.0308	7.5102
	Diginetica	0.0024	0.0354	8.8914
Cassandra	Retail Rocket	0.6377	0.4721	60.1256
	Diginetica	0.5541	0.2058	62.1402
Redis	Retail Rocket	0.0006	0.0156	7.5099
	Diginetica	0.0011	0.0864	8.8981
Neo4j	Retail Rocket	0.1567	0.0717	15.0236
	Diginetica	0.0639	0.0354	8.8774

Table 2: Database Creation Parameters

8.1.3. Inserting Records into Created Databases

Bulk Insertion

For Retail Rocket dataset, MongoDB demonstrated the best performance by taking only 0.000058 seconds per record for bulk insertion. This was comparable with PostgreSQL's 0.000337 seconds per record and Neo4j's 0.260486 seconds per record. At 1,000,000 records, MongoDB still led with a speed of 0.000145 seconds per record. For Diginetica

dataset, MongoDB had the fastest insertion time of 0.000009 seconds per record for 100000 records.

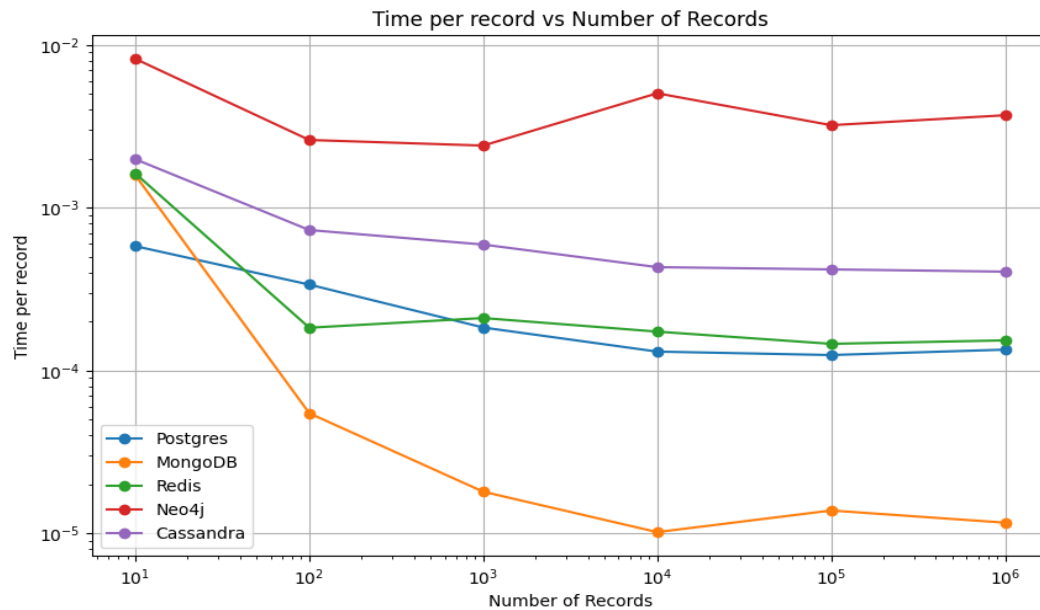


Figure 3: Retail Rocket Bulk Insertion Latency

In terms of memory usage for Retail Rocket Dataset, Neo4j used the most resources. It used approximately 4.88e+8 MB of memory for 1 million records. Cassandra used 5e+7 MB. For Diginetica dataset, Neo4j also showed high memory usage. This suggests that while Neo4j is good at handling complex relationships well, but it requires significantly more memory. In conclusion, MongoDB is most efficient in terms of memory consumption with PostgreSQL and Redis being in the top three choices.

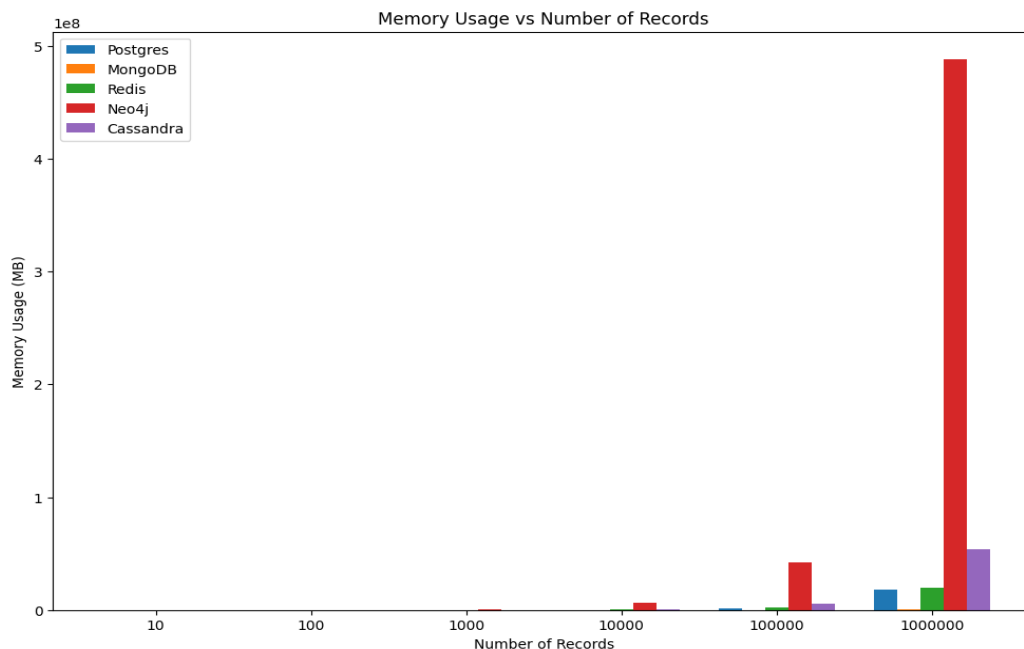


Figure 4: Retail Rocket Bulk Insertion Memory Usage

Batch Insertion

For Retail Rocket dataset, Initially PostgreSQL showed fast speed for batch insertion up until 50 records (Choina & Skublewska-Paszkowska, 2022). Neo4j was the slowest for batch insertions. Overall, MongoDB was most efficient in inserting data in batches. For Diginetica, Similar results can be seen. MongoDB has superior performance in handling batch insertions, making it an optimal choice for applications requiring rapid data insertion.

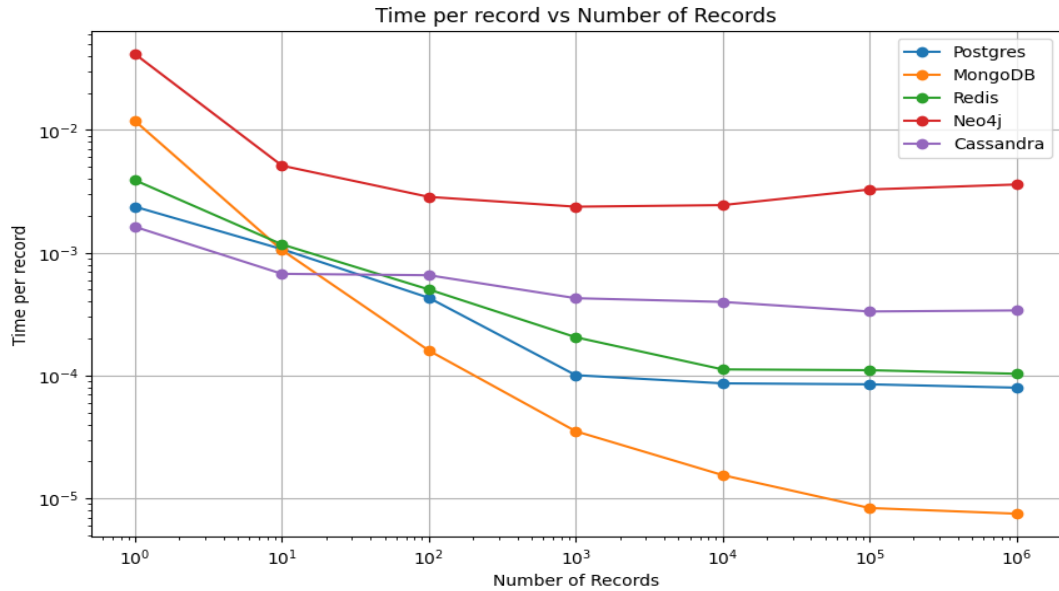


Figure 5: Diginetica Batch Insertion Latency

In terms of memory usage, for Retail Rocket dataset, Neo4j used the most space. MongoDB followed by Redis and PostgreSQL have the least space requirements (Choina & Skublewska-Paszkowska, 2022). For Diginetica Dataset, Neo4j used the most resources. This suggest that while Neo4j is good at handling complex relationships well, but it requires significantly more memory. In conclusion, MongoDB is most efficient in terms of memory consumption with PostgreSQL and Redis being in top three choices.

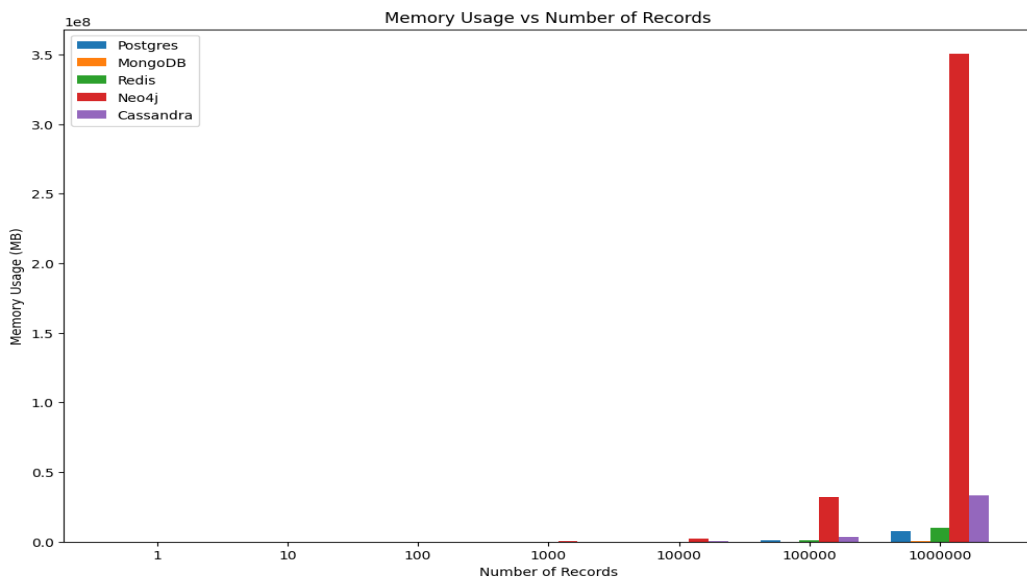


Figure 6: Diginetica Batch Insertion Memory Usage

8.1.4. Reading Records

For Retail Rocket, Postgres demonstrated the fastest performance from the start as it took only 1 second to read 1 million records. MongoDB was the second-best performing DBMS at approximately 7 seconds. Neo4j and Cassandra were the slowest. For Diginetica, PostgreSQL was the fastest with time less than a second to read 1 million records. MongoDB and Cassandra are the second and third runners-up for read operations.

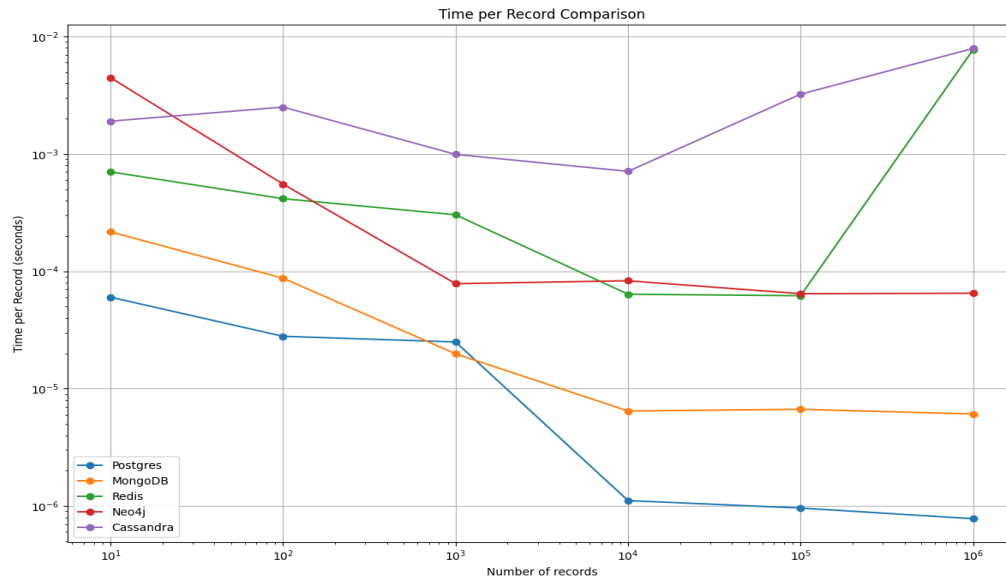


Figure 7: Retail Rocket Reading Latency

For Retail Rocket Dataset, the memory usage of Cassandra was maximum. Redis and Neo4j were in top 3 databases in terms of memory usage. For Diginetica, Redis used the most space with Neo4j and Cassandra being in the top three in terms of memory usage in read operation.

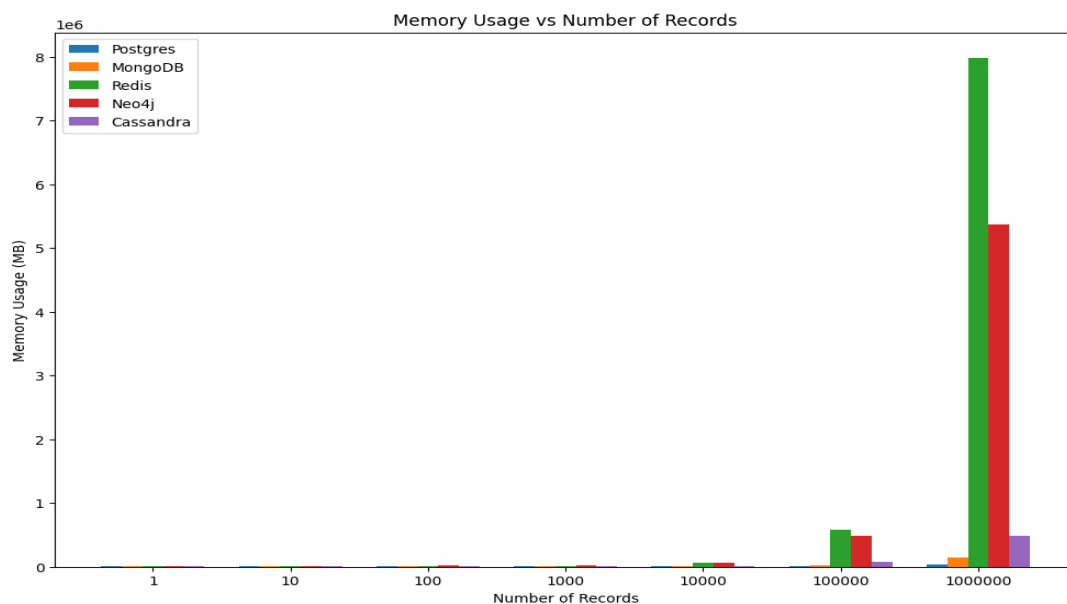


Figure 8: Diginetica Reading Memory Usage

8.1.5. Updating Records

In Cassandra, a primary key feature is fixed so update operations were performed on these features. For Retail Rocket dataset, visitorid, itemid, and transactionid features were incremented, events string was set to 'updated'. In Diginetica Dataset, timeframe and itemid features were updated by concatenating '_updated' in each value.

For Retail Rocket, Redis showed the fastest update time with a minimum time of 0.0038 seconds for 10 records and 51.24 seconds for 1,000,000 records. PostgreSQL showed a similar performance to Redis. In contrast, Neo4j and Cassandra were the slowest. For Diginetica Dataset, PostgreSQL showed the fastest performance by taking only 4.0614 seconds to update 1 million records. Neo4j and Redis were second and third fastest in updating the operations.

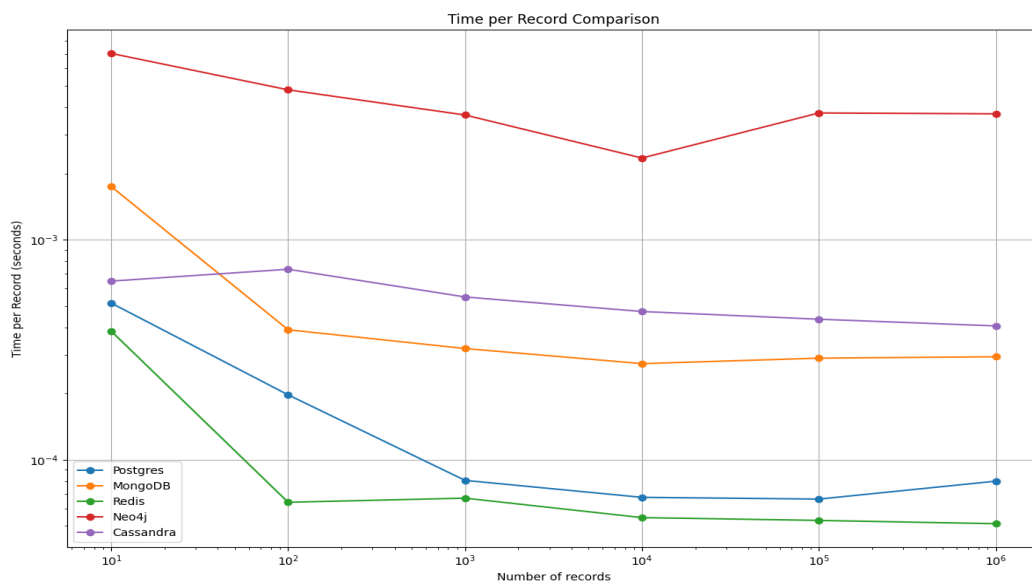


Figure 9: Retail Rocket Updating Latency

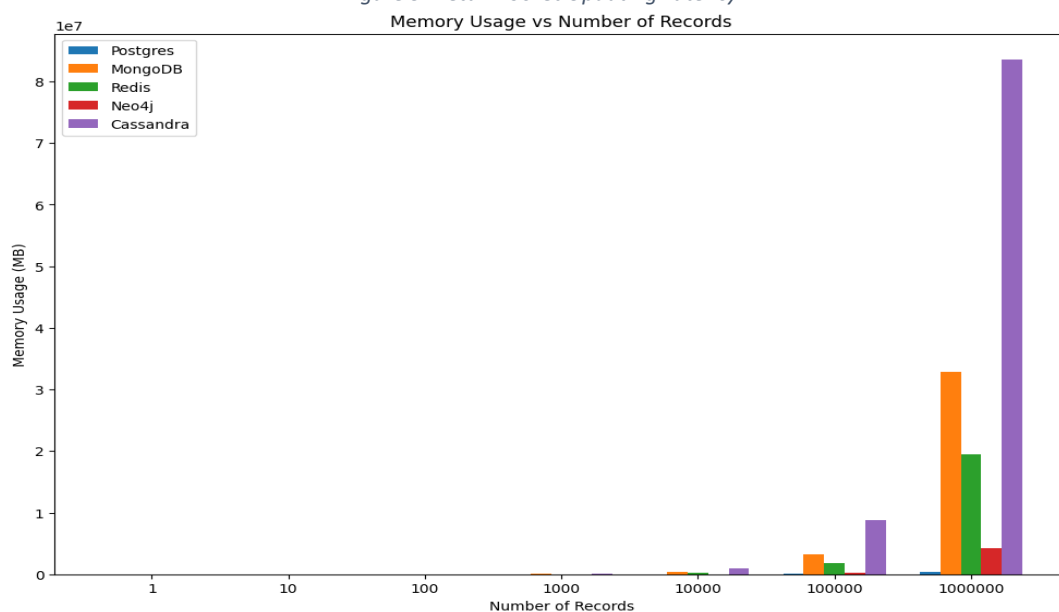


Figure 10: Diginetica Updating Memory Usage

In terms of Memory usage, For Retail Rocket, Neo4j utilized the most memory. Redis and PostgreSQL used the least space. For Diginetica, Cassandra used the most memory. The least memory space was used by PostgreSQL and Neo4j (Choina & Skublewska-Paszkowska, 2022).

8.1.6. Deleting Records

For Retail Rocket, Redis was the fastest for deletion operation at 37.083 seconds. PostgreSQL was second fastest with 37.08 seconds. Neo4j and Cassandra were the slowest. For Diginetica, PostgreSQL was the fastest for deletion operation at less than a second for deletion of 1 million records. Redis is the second fastest at approximately 2 seconds.

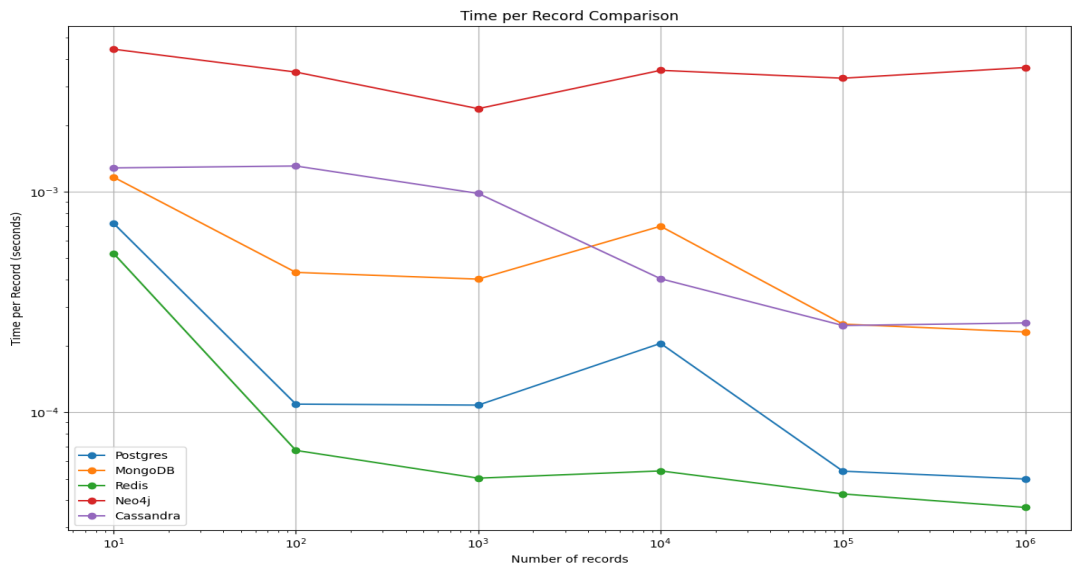


Figure 11: Retail Rocket Deleting Latency

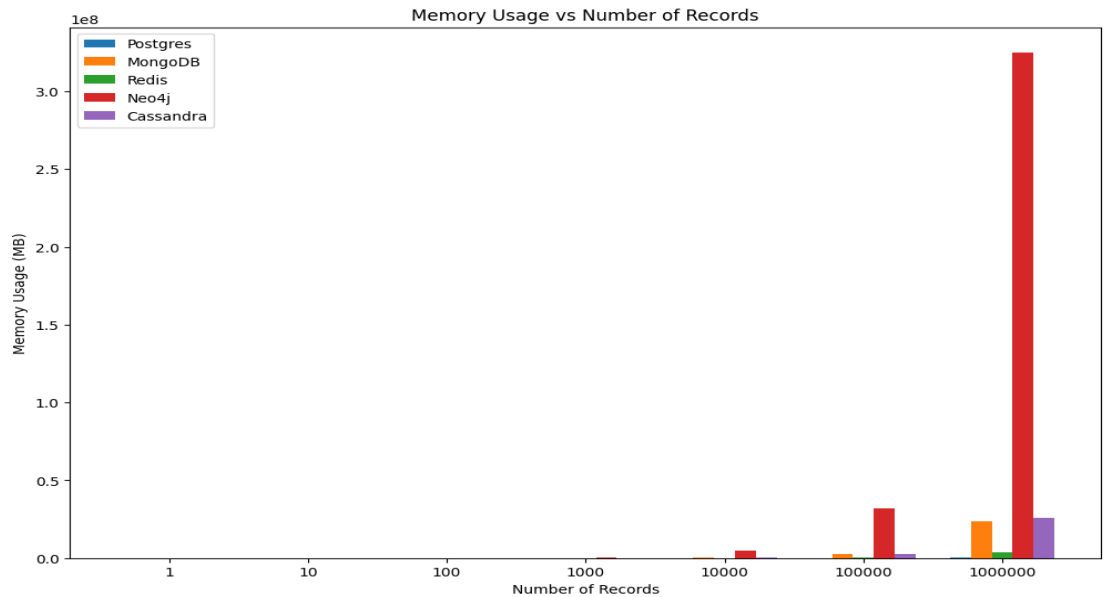


Figure 12: Diginetica Deleting Memory Usage

Regarding memory usage, the Retail Rocket dataset showed that Neo4j used the most memory. PostgreSQL and MongoDB used the least memory. For Diginetica Dataset, similar

results were obtained with Neo4j using the maximum memory. The least memory was used by PostgreSQL and Redis respectively.

8.1.7. CRUD Insights

This section provides an in-depth analysis of the performance of CRUD operations on Retail Rocket and Diginetica Datasets. Understanding the efficiency and resource usage of these operations is crucial for optimizing database performance in RS. The analysis includes both a tabular representation of key parameters and graphical insights to highlight the comparative performance across different database management systems.

The Retail Rocket dataset is used to evaluate the performance of various customer behaviour events such as clicks, add-to-carts, and transactions. Key parameters assessed include creation time, bulk and batch insertion times, reading latency, updating time, deleting time, and memory usage.

Database	Creation	Bulk Insertion	Batch Insertion	Reading	Updating	Deleting	Memory Usage
PostgreSQL	0.0125515	134.21906	142.22567	0.780382	79.957343	49.883506	Low
MongoDB	0.001479	11.609656	21.756386	6.087698	293.84253	231.62558	Low
Cassandra	0.6377148	404.39918	407.11136	7921.88	405.77281	254.48584	High
Redis	0.000598	153.00538	140.58597	7745.732	51.241771	37.083317	Low
Neo4j	0.1566786	3695.8162	4728.1458	65.02913	3732.3179	3663.4662	Extremely High

Table 3: Retail Rocket Parameters

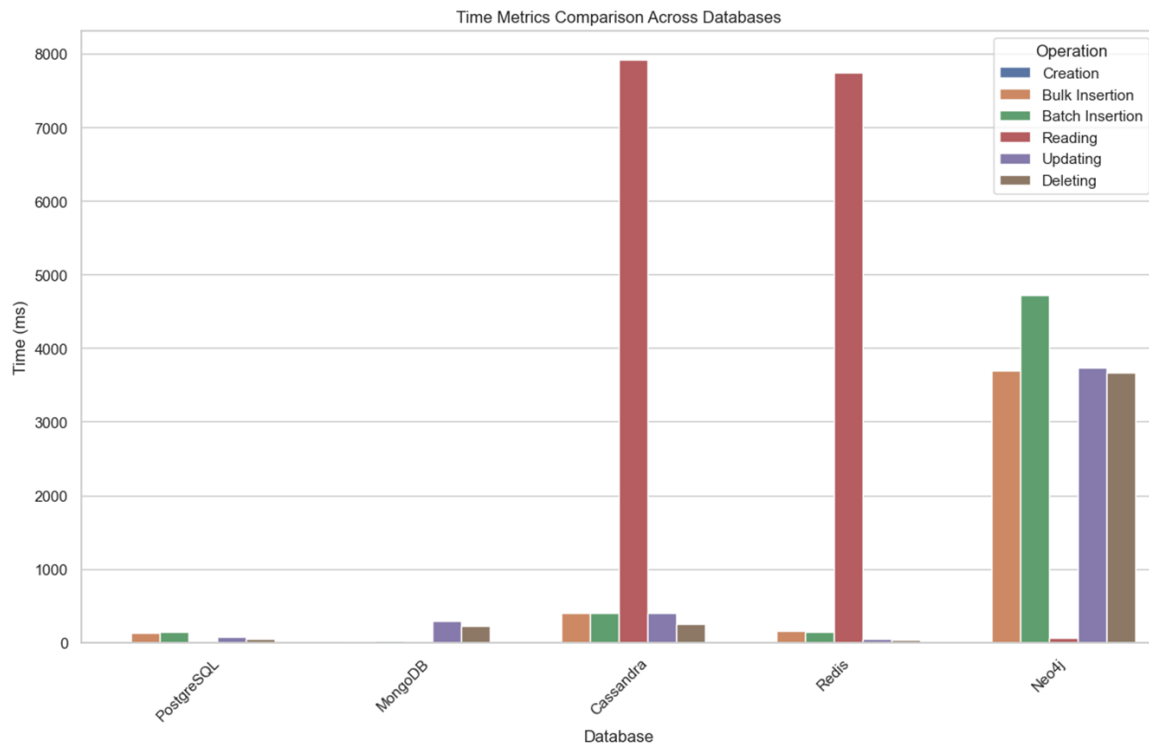


Figure 13: Retail Rocket Graph

The Retail Rocket CRUD graph in Figure 13 provides a visual representation of the efficiency of various database systems across different operations. MongoDB demonstrates superior

performance in creation and bulk insertion times, making it an excellent choice for handling large datasets efficiently (Truică et al., 2015). PostgreSQL excels in reading and updating operations, offering quick access and modifications to data, which is vital for real-time recommendation systems. On the other hand, Redis stands out with its fast update and deletion times, ensuring that data remains current and relevant. However, Neo4j exhibits high memory usage across several operations, indicating a trade-off between performance and resource consumption (Yi et al., 2017).

The Diginetica dataset is another valuable resource for analyzing clickstream data, essential for understanding user interactions and preferences. Similar to the Retail Rocket dataset, the CRUD performance across different databases is measured based on the same set of parameters.

Database	Creation	Bulk Insertion	Batch Insertion	Reading	Updating	Deleting	Memory Usage
PostgreSQL	0.0039057	96.433320	79.756680	0.5059597	4.0614347	2.1787874	Low
MongoDB	0.0023846	8.519765	7.5038914	4.1767740	296.36906	229.18407	Low
Cassandra	0.5540585	370.73268	339.251	4.4117560	636.16798	263.91741	High
Redis	0.0011107	103.76462	103.49634	74.313283	182.59218	37.491430	Low
Neo4j	0.0639102	3459.4401	3601.1960	62.813655	33.763957	3355.3404	Extremely High

Table 4: Diginetica Parameters

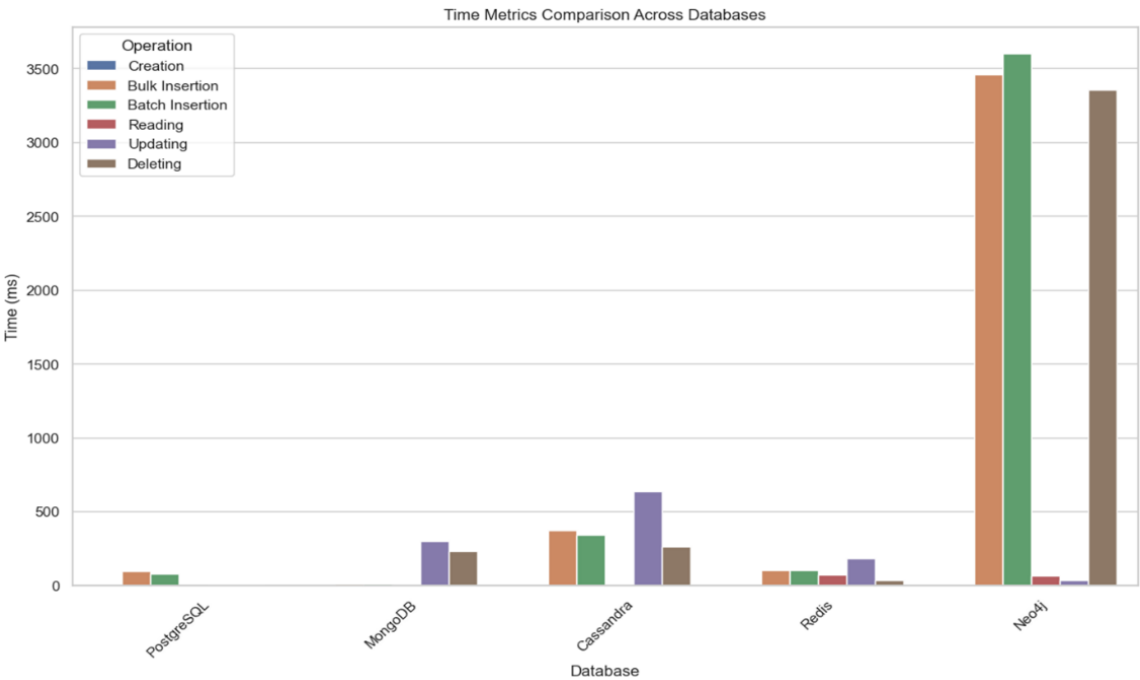


Figure 14: Diginetica Graph

The CRUD performance graph for Diginetica in Figure 14 illustrates the efficiency of various database systems across different operations. MongoDB continues to excel with efficient creation and bulk insertion times, particularly for large datasets. PostgreSQL provides quick

read and update times, which are crucial for maintaining the accuracy and relevance of recommendations. Redis, although efficient in deletion times, shows high memory usage during read operations (Gupta et al., 2017). Neo4j and Cassandra exhibit high memory usage, impacting their overall performance, especially during bulk data operations (Nakhare, 2021).

The performance analysis of CRUD operations on the Retail Rocket and Diginetica datasets underscores MongoDB's superior efficiency in bulk and batch insertions, and PostgreSQL's strength in reading, updating, and deleting operations. Neo4j, while excellent at handling complex relationships, exhibits high memory usage, and Redis is noted for its fast update and deletion times but high memory consumption during reads (Györödi et al., 2022). These insights enable database administrators and system architects to choose the most suitable database systems for specific applications, optimizing both performance and resource utilization.

The next section will delve into the specifics of utilizing Neo4j to build a recommender system. This section will explore how Neo4j's graph database capabilities can be leveraged to enhance the performance and accuracy of recommendation algorithms, despite the challenges highlighted in the CRUD analysis.

8.2. Neo4j-based Recommender System

8.2.1. Introduction

It leverages the graph database capabilities of Neo4j to store the movie lens database. The relationship between movies and users is created and analysed. The system can efficiently compute recommendations based on interconnected nodes (Guo et al., 2022).

8.2.2. Movie – Movie Similarity

This model suggests similar movies based on the number of common users between the two movies. Since, Positive rating by both users is necessary to find similar movies. The code only **counts those unique users who rated movie1 and movie2 as 5**. The movie1 is set as 'Toy Story (1995)' and the code recommends those movies that have the highest count of positive user ratings by common users. In Table 3, results show that 'Star Wars: Episode IV - A New Hope (1977)' is the first recommended movie with highest number of positive user ratings by common user.

Index	m2.Title	m2.Genres	common_users
1	Star Wars: Episode IV - A New Hope (1977)	Action Adventure Fantasy Sci-Fi	401
2	Toy Story 2 (1999)	Animation Children's Comedy	385
3	Raiders of the Lost Ark (1981)	Action Adventure	373
4	Star Wars: Episode V - The Empire Strikes Back...	Action Adventure Drama Sci-Fi War	346
5	Shawshank Redemption, The (1994)	Drama	327

Table 5: Movie-Movie Similarity

8.2.3. User-Based Recommendations using Jaccard Similarity

The Process of recommending a movie to a user using Jaccard Similarity is done in two steps. The calculation of Jaccard Similarity is done first followed by using weighted score of ratings to recommend movies to the users.

Step 1: Jaccard Similarity Score

Firstly, Jaccard Similarity for all the UserID is calculated and added as a feature in the relationship between nodes. The Figure below shows some of the similarity indexes as a feature in node relationships of the database for UserID: 4725. The Similarity scores can be referred in Table 4. **For detailed explanation of Jaccard Similarity calculation refer section 6.5.**

Index	UserID1	UserID2	Similarity
0	4725	4808	0.755415
1	4808	4725	0.755415
2	1122	2126	0.632000
3	2126	1122	0.632000
4	1272	2837	0.601852

Table 6: Jaccard Similarity for all User Nodes

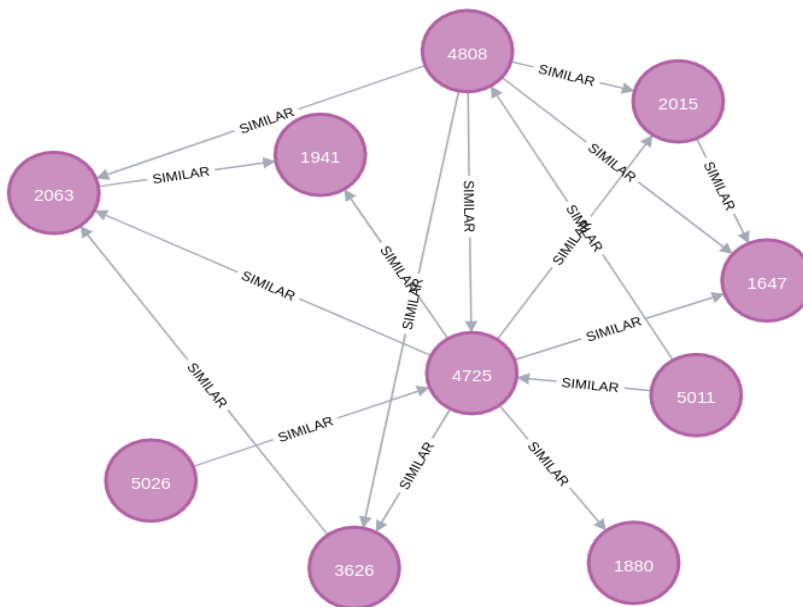


Figure 15: Similarity Relationships for UserID: 4725

Step 2: Weighted Score based on Rating

Once, Similar users are added. The average rating weighted by similarity score is calculated for each UserID. The figure below shows a sample of movies connected to UserID: 4725 via a similar user. **For detailed explanation of calculating weighted score refer section 7.4.6.** These movies are eventually weighted based on similarity scores by taking averages. The

final movie recommendations can be referred in Table 5.

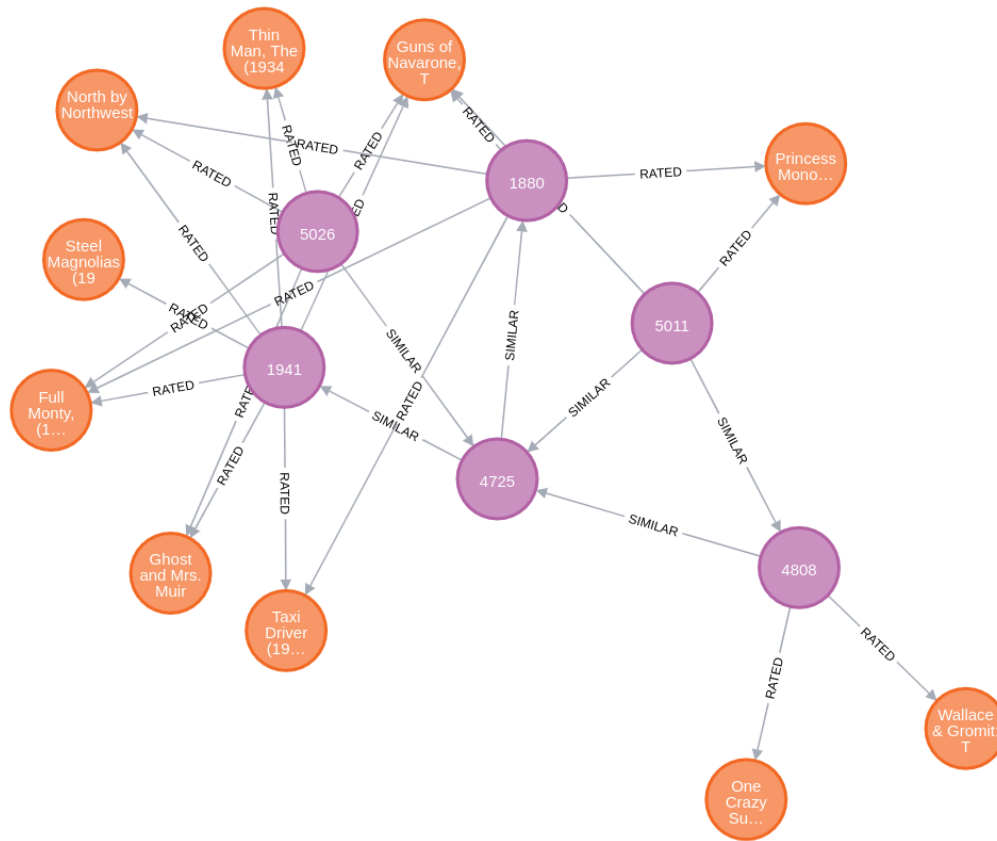


Figure 16: Movie Connections for UserID:4725

Index	m.Title	m.Genres	score
0	Schindler's List (1993)	Drama War	6.864803
1	Toy Story (1995)	Animation Children's Comedy	6.497486
2	October Sky (1999)	Drama	6.390129
3	Almost Famous (2000)	Comedy Drama	6.389066
4	Bulworth (1998)	Comedy	6.283263
5	Boys Don't Cry (1999)	Drama	6.259083
6	Apollo 13 (1995)	Drama	6.209804
7	Boogie Nights (1997)	Drama	6.169459
8	GoodFellas (1990)	Crime Drama	6.156693
9	Simple Plan, A (1998)	Crime Thriller	6.120458

Table 7: Movie Recommendations for UserID:4725

Based on the recommendations from the model, the UserID:4725 is recommended movies falling into genre of 'Drama', 'Comedy' and 'Crime' such as 'Schindler's List (1993)' and 'October Sky (1999)'.

8.2.4. Conclusion

In Summary, Neo4j-based RS effectively utilized graph database capabilities to provide personalized movie recommendations. The limitation of this model is the data need to have enough features to be segmented into nodes and relationship features (Dueñas-Lerín et al., 2023). These findings demonstrate the system's potential for real-world applications.

9. Insights on Database Suitability

9.1. Consolidation of Research Findings

The selection of the appropriate database architecture is crucial in optimizing the performance of Recommender Systems (RS). This research aims to elucidate the impact of different database architectures on RS performance by considering various algorithms and application contexts. Through an in-depth analysis of SQL, NoSQL, and graph databases, key strengths and weaknesses have been identified that inform the suitability of each database type for specific RS tasks. Below, the research question and critical insights gained from the study are discussed.

Main Research Question: *“How does database architecture selection impact RS performance, considering varying algorithms and application contexts?”*

Key Findings: The comprehensive analysis of SQL, NoSQL, and graph Databases has yielded several insights on their usefulness for RS:

9.1.1. SQL Databases (PostgreSQL):

PostgreSQL excelled in handling structured data and the use of complex queries. For Read and Delete operations, PostgreSQL demonstrated superior performance. This showed its efficiency in managing large volumes of structured data. However, PostgreSQL’s limitation of vertical scalability is a drawback for modern RS at AI Systems BV (Nakhare, 2021) (Filip & Čegan, 2020).

9.1.2. NoSQL Databases (MongoDB, Cassandra, Redis):

NoSQL databases have an advantage of more flexibility and availability. MongoDB excelled in bulk and batch insertion. It also showed fast performance in creation operations due to dynamic schema and auto-sharding capabilities. This makes MongoDB an ideal choice for handling large-scale unstructured data in RS (Győrödi et al. 2022). Cassandra’s strengths lie in update operations. It has a distributed architecture that is beneficial for RS with large active user bases (Abramova et al. 2014). Redis provides low latency in update and delete operations. It is essential for real-time recommendations (Gupta et al., 2017).

9.1.3. Graph Databases (Neo4j):

Neo4j is efficient in modelling and handling complex relationships between entities. It demonstrated robust performance in operations involving complex queries and relationship traversals. However, it exhibited high memory usage during bulk and batch insertions. This is a major drawback in that indicates a trade-off between performance and resource utilization (Yi et al. 2017) (Wu et al. 2022).

Sub-Questions:

1) What types of databases are typically used in Recommender Systems and how do they compare with each other?

Recommender Systems typically employ SQL, NoSQL, and graph databases. SQL databases, such as PostgreSQL, are the traditional choice and excel in structured data management. NoSQL databases, including MongoDB, Cassandra, and Redis, are favoured for their high flexibility and scalability in managing unstructured data. Graph databases like Neo4j are optimal for complex relationship modelling and traversal.

2) *Which metrics could be used to compare their performance considering different types of RS algorithms?*

Key metrics used to evaluate performance include latency, memory usage, scalability, and availability. In comparison, PostgreSQL is optimal for read and delete operations, MongoDB excels in bulk and batch insertions, Cassandra is superior for update operations, Redis is ideal for real-time recommendations, and Neo4j outperforms in complex relationship queries.

3) *How could their performance be demonstrated by developing a tool?*

The performance of these databases was demonstrated by developing a tool to measure CRUD operations and memory usage. For Neo4j, the tool assessed its efficiency in handling complex relationship queries and traversals. Specific metrics such as query response time, memory consumption during data import, and traversal speed were analysed. The Neo4j-based recommender system included setting up a Neo4j sandbox, importing datasets, implementing algorithms for movie-movie similarity, and user-based recommendations using Jaccard similarity. This thorough approach provided clear insights into Neo4j's performance and suitability for RS applications.

4) *Which insights could be drawn about the suitability and performance of these different combinations of databases and algorithms?*

Insights indicate that each database type offers unique advantages depending on the RS application context. PostgreSQL is well-suited for applications requiring strong data integrity and complex queries. MongoDB is ideal for managing large-scale unstructured data.

Cassandra excels in distributed environments with extensive active user bases. Redis is optimal for real-time applications, and Neo4j is best for managing complex relationships.

More information about the research findings can be found in the figure below. It presents the merits and demerits of each database based on their suitability for RS.






Database	Latency	Memory Usage	Scalability	Availability
PostgreSQL 	Efficient for read and delete operations , but latency can increase with complex queries (Truić et al., 2015)	Efficient memory management, but may consume more during peak loads (Nakhare, 2021)	Traditionally scales vertically , which can limit handling of massive datasets (Filip & Čegan, 2020)	High availability through replication and failover mechanisms (Lourenço et al., 2015)
MongoDB 	Low latency for create and insert operations , in high-throughput environments (Truić et al., 2015)	Efficient in memory consumption (Filip & Čegan, 2020)	Built for horizontal scalability (Gvörödi et al., 2022)	High availability through distributed architecture and replication (Wang et al., 2022)
Redis 	Very low latency , ideal for real-time recommendations (Gupta et al., 2017)	Efficient memory usage (Gupta et al., 2017)	Scales horizontally (Gvörödi et al., 2022)	High availability through replication (Gvörödi et al., 2022)
Cassandra 	Superior for update operations (Truić et al., 2015)	Higher memory usage compared to some others (Kabakuş & Kara, 2017)	Excellent horizontal scalability (Bjeladinović, 2018)	High availability through distributed architecture (Gvörödi et al., 2022)
Neo4j 	Low latency for complex relationship queries (Yi et al., 2017). Database should be representable in the form of nodes and relationship.	Extremely High memory consumption, especially during bulk insertions (Yi et al., 2017)	Can scale both horizontally and vertically, but limited by graph complexity (Wu et al., 2022)	High availability through replication and clustering (Wu et al., 2022)

Figure 17: Database Insights

Having consolidated the research findings and addressed the main and sub-questions, it is now essential to translate these insights into actionable recommendations. The following section will outline strategic recommendations for database design and implementation in RS, leveraging the strengths of various database architectures to optimize performance and scalability.

9.2. Recommendations for Database Design

Based on research findings, the following recommendations can be made –

Integrate MongoDB for Large-Scale Data Management

To address the increasing demands of larger clients, it is recommended that AI Systems BV **integrates MongoDB alongside PostgreSQL**. PostgreSQL's robust ACID compliance and superior performance in managing structured data and executing complex queries make it indispensable for transactional applications. However, MongoDB should be incorporated to handle large-scale unstructured data due to its dynamic schema and horizontal scalability. The results from CRUD operations clearly demonstrated MongoDB's excellence in managing large volumes of data with faster read and write operations, making it crucial for applications requiring high availability and scalability (Bjeladinović 2018).

Implement Performance Optimization Techniques

To maintain high performance and low latency, which are vital for user satisfaction, AI Systems BV should adopt several optimization strategies. This includes optimizing database indexing strategies to expedite query execution and **utilizing in-memory databases like Redis for caching** frequently accessed data to reduce load times (Gupta et al., 2017). Regular monitoring and tuning of database performance metrics such as query response times and memory usage are also essential. Redis, known for its in-memory data storage, provides extremely low latency and high throughput, making it ideal for caching and significantly improving overall system performance.

Enhance Scalability with NoSQL Databases

As AI Systems BV continues to grow, scalability becomes paramount. Deploying NoSQL databases like **MongoDB or Cassandra can manage expanding data volumes efficiently**. These databases offer inherent horizontal scalability, distributing data across multiple servers to ensure consistent performance even during peak loads. Cassandra, in particular, is effective for write-intensive operations, handling high write loads with minimal latency due to its distributed architecture, thus complementing the capabilities of both MongoDB and PostgreSQL (Truică et al., 2015) (Györödi et al., 2022).

Utilize Graph Databases for User-Item Interactions

For enhancing the personalization capabilities of recommender systems, the implementation of graph databases like Neo4j is recommended. Graph databases efficiently manage and query complex relationships between users and items, which is pivotal for providing accurate and personalized recommendations. By incorporating Neo4j, AI Systems BV can leverage advanced recommendation algorithms that traverse user-item interactions with higher precision, thus improving user satisfaction and engagement (Afoudi et al., 2023).

By implementing these recommendations, AI Systems BV can substantially enhance the performance, scalability, and reliability of their recommender systems. These improvements will not only boost user satisfaction and engagement but also ensure that AI Systems BV remains competitive in the ever-evolving landscape of recommender system technology.

9.3. Limitations of Research

Despite the comprehensive analysis and thorough evaluation presented in this study, several limitations must be acknowledged. These limitations provide context for the findings and offer directions for future research.

9.3.1. Limited Scope of Performance Metrics and Applications

The research primarily focused on a specific set of performance metrics, namely latency, memory usage, scalability, and availability. While these are critical factors in evaluating database performance, other important metrics such as **energy efficiency, long-term maintenance costs, and security aspects** were not explored in detail. Other important applications of RS such as **cold-start problem and prioritizing customers** were overlooked. Future studies should include these additional metrics and applications to provide a more holistic evaluation of database systems.

9.3.2. Evolving Database Technologies

Database technologies are continually evolving, with new advancements and features being introduced regularly. This research is based on the capabilities and performance of the databases at the time of the study. Future advancements, such as the development and adoption of **NewSQL databases**, could significantly alter the performance characteristics of these databases, rendering some of the findings obsolete. Ongoing research and periodic re-evaluation of database performance are necessary to stay current with technological advancements.

9.3.3. Sample Size and Testing Environment

The performance evaluations conducted in this study were based on a dataset comprising 1,000,000 records, tested within a controlled environment. The system specifications used for testing included a server with 16GB RAM, an Intel i7 processor, and SSD storage. The **sample size and the controlled nature of the environment** might not fully capture the variability and challenges encountered in real-world applications. Expanding the sample size and conducting tests in more varied and dynamic environments could yield more robust and applicable insights.

9.3.4. System Configuration Discrepancies

The tests and evaluations were performed on system configurations that may differ from those used at AI Systems BV. This discrepancy could affect the applicability of the findings to AI Systems BV's **specific environment and operational setup**. Future research should aim to replicate the tests using the actual system configurations employed by AI Systems BV to ensure the results are directly applicable and relevant.

9.3.5. Tool and Methodology Limitations

The tools and methodologies used for evaluating database performance, while comprehensive, have their limitations. The choice of tools, the configuration settings, and the specific methodologies employed could introduce biases or constraints that affect the outcomes. Alternative methodologies, such as using different benchmarking tools or varying the types of data operations beyond CRUD operations like **complex analytical queries or**

multi-user concurrency tests, could provide a more comprehensive assessment of database performance.

9.4. Future Directions and Best Practices

Future research can explore several avenues to enhance the effectiveness of the developed tool and its practical application in various database scenarios:

Integration of Advanced Algorithms: Further research should investigate the integration of advanced algorithms within AI Systems BV across various database types. This would involve comparing how different algorithms interact with SQL, NoSQL, and Hybrid databases, leading to more concrete insights into the impact of database choice on the performance of recommendation systems (Wang, 2016) (Zhang et al. 2019).

Real-Time Data Processing: Investigate real-time data processing capabilities of different Database Management Systems (DBMSs). Understanding the relationship between real-time data processing and various recommendation system (RS) models can provide significant insights into optimizing system performance (Lourenço et al. 2015).

Cold Start Solutions: Develop strategies to address the cold start problem, where the system has limited initial data. This could include initial data collection methods, user input prompts, or transfer learning techniques that can enhance the system's recommendation accuracy from the beginning.

Customer Prioritization: Explore methods for prioritizing different customer segments, especially 5-star customers. Tailoring database and recommendation system solutions to meet the unique needs of high-value users can improve overall satisfaction and system utility. This prioritization can include customized database optimizations, personalized data handling, and focused algorithm adjustments.

Efficient Data Partitioning and Indexing: Research the impact of efficient data partitioning and indexing techniques on RS models. Effective data partitioning and indexing can enhance performance and scalability, providing faster and more accurate recommendations (Sarwat et al. 2017).

Cross-Domain Applicability: Assess the applicability of the developed tool in diverse operational environments. Understanding how different domains and dataset characteristics affect tool performance can broaden its usability and effectiveness, ensuring adaptability to various real-world scenarios.

By addressing these areas, future research can contribute to more robust, adaptable, and efficient recommendation systems that are better aligned with both current technological capabilities and user needs.

References:

- Abramova, V., Bernardino, J., & Furtado, P. (2014). Which NoSQL database? A performance overview. *Open Journal of Databases (OJDB)* Volume 1, Issue 2, 2014, 1(2), 17–24. <https://estudogeral.sib.uc.pt/bitstream/10316/27748/1/Which%20NoSQL%20Database.pdf>
- Afoudi, Y., Lazaar, M., & Hmaidi, S. (2023c). An enhanced recommender system based on heterogeneous graph link prediction. *Engineering Applications of Artificial Intelligence*, 124, 106553. <https://doi.org/10.1016/j.engappai.2023.106553>
- Afsar, M M., Crump, R T., & Far, B H. (2022, December 15). Reinforcement Learning based Recommender Systems: A Survey. *Association for Computing Machinery*, 55(7), 1-38. <https://doi.org/10.1145/3543846>
- Almonte, L., Guerra, E., Cantador, I., & De Lara, J. (2021). Recommender systems in model-driven engineering. *Software and Systems Modeling*, 21(1), 249–280. <https://doi.org/10.1007/s10270-021-00905-x>
- Angles, R., & Gutiérrez, C. (2018, January 1). *An Introduction to Graph Data Management*. Springer International Publishing, 1-32. https://doi.org/10.1007/978-3-319-96193-4_1
- Awan, M. T. (2022). Linux vs. Windows: A Comparison of Two Widely Used Platforms. *Journal of Computer Science and Technology Studies*, 4(1), 41–53. <https://doi.org/10.32996/jcsts.2022.4.1.4>
- Bag, S., Kumar, S K., & Tiwari, M K. (n.d). An efficient recommendation generation using relevant Jaccard similarity
- Battle, L., Eichmann, P., Angelini, M., Catarci, T., Santucci, G., Zheng, Y., Binnig, C., Fekete, J., & Moritz, D. (2020, May 31). Database Benchmarking for Supporting Real-Time Interactive Querying of Large Data. <https://doi.org/10.1145/3318464.3389732>
- Bhogal, J., & Choksi, I. (2015b). Handling Big Data Using NoSQL. Faculty of Computing, Engineering and the Built Environment School of Computing, Telecommunication and Networks Birmingham City University, UK. <https://doi.org/10.1109/waina.2015.19>
- Biswas, P., & Liu, S. (2022b). A hybrid recommender system for recommending smartphones to prospective customers. *Expert Systems With Applications*, 208, 118058. <https://doi.org/10.1016/j.eswa.2022.118058>
- Bjeladinović, S. (2018b). A fresh approach for hybrid SQL/NoSQL database design based on data structuredness. *Enterprise Information Systems*, 12(8–9), 1202–1220. <https://doi.org/10.1080/17517575.2018.1446102>
- Bobadilla, J., Ortega, F., Hernando, A., & Gutiérrez, A. (2013). Recommender systems survey. *Knowledge-based Systems*, 46, 109–132. <https://doi.org/10.1016/j.knosys.2013.03.012>
- Catapang, J K. (2018, January 1). A collection of database industrial techniques and optimization approaches of database operations. Cornell University. <https://doi.org/10.48550/arXiv.1809>.
- Chandra, D. G. (2015). BASE analysis of NoSQL database. *Future Generation Computer Systems*, 52, 13–21. <https://doi.org/10.1016/j.future.2015.05.003>
- Carlson, J L. (2013, June 28). Redis in Action. <https://dl.acm.org/citation.cfm?id=2505464>
- Choina, M., & Skublewska-Paszkowska, M. (2022, January 1). Performance analysis of relational databases MySQL, PostgreSQL and Oracle using Doctrine libraries
- Corbellini, A., Mateos, C., Zunino, A., Godoy, D., & Schiaffino, S. (2017). Persisting big-data: The NoSQL landscape. *Information Systems*, 63, 1–23. <https://doi.org/10.1016/j.is.2016.07.009>

- Deldjoo, Y., Schedl, M., Cremonesi, P., & Pasi, G. (2020). Recommender Systems Leveraging Multimedia Content. *ACM Computing Surveys (CSUR)*, 53, 1 - 38. <https://doi.org/10.1145/3407190>.
- De Winter, J. C. F., Gosling, S. D., & Potter, J. (2016). Comparing the Pearson and Spearman correlation coefficients across distributions and sample sizes: A tutorial using simulations and empirical data. *Psychological Methods*, 21(3), 273–290. <https://doi.org/10.1037/met0000079>
- Dueñas-Lerín, J., Lara-Cabrera, R., Ortega, F., & Bobadilla, J. (2023, January 1). Deep Neural Aggregation for Recommending Items to Group of Users. Cornell University. <https://arxiv.org/abs/2307.09447>
- Filip, P., & Čegan, L. (2020). Comparison of MySQL and MongoDB with focus on performance. *International Conference on Informatics, Multimedia, Cyber and Information System (ICIMCIS)*. <https://doi.org/10.1109/icimcis51567.2020.9354307>
- Gao, J., Liu, W., Du, H., & Zhang, X. (2017, November 1). Batch insertion strategy in a distribution database. <https://doi.org/10.1109/icsess.2017.8342991>
- Goldberg, D., Nichols, D., Oki, B M., & 7~rry, D. (1992, December 1). Tapestry: An Experimental Mail System for Filtering Information
- Goyal, K., Ranawat, K R S., & Nayak, N. (2018, December 10). Operational Distinctions Between Linux and Windows. *Technoscience Academy*, 251-254. <https://doi.org/10.32628/ijrst18401152>
- Grolinger, K., Higashino, W A., Tiwari, A., & Capretz, M A M. (2013, December 1). Data management in cloud environments: NoSQL and NewSQL data stores. *Springer Nature*, 2(1). <https://doi.org/10.1186/2192-113x-2-22>
- Guo, Q., Zhuang, F., Qin, C., Zhu, H., Xie, X., Xiong, H., & He, Q. (2022). A survey on Knowledge Graph-Based Recommender Systems. *IEEE Transactions on Knowledge and Data Engineering*, 34(8), 3549–3568. <https://doi.org/10.1109/tkde.2020.3028705>
- Guo, Z., & Wang, H. (2021). A deep graph neural Network-Based mechanism for social recommendations. *IEEE Transactions on Industrial Informatics*, 17(4), 2776–2783. <https://doi.org/10.1109/tii.2020.2986316>
- Gupta, A., Tyagi, S., Panwar, N., Sachdeva, S., & Saxena, U. (2017, October 1). NoSQL databases: Critical analysis and comparison. <https://doi.org/10.1109/ic3tsn.2017.8284494>
- Győrödi, C., Dumșe-Burescu, D. V., Zmaranda, D., & Győrödi, R. (2022). A comparative study of MongoDB and Document-Based MySQL for big data application data management. *Big Data and Cognitive Computing*, 6(2), 49. <https://doi.org/10.3390/bdcc6020049>
- Harper, F., Konstan, J., & A., J. (2016). The MovieLens Datasets: History and Context. *ACM Trans. Interact. Intell. Syst.*, 5, 19:1-19:19. <https://doi.org/10.1145/2827872>.
- Huang, Z., Chung, W., Ong, T., & Chen, H. (2002). A graph-based recommender system for digital library. University of Arizona. <https://doi.org/10.1145/544220.544231>
- Hubail, M. A., Alsuliman, A., Blow, M., Carey, M., Lychagin, D., Maxon, I., & Westmann, T. (2019). Couchbase analytics. *Proceedings of the VLDB Endowment*, 12(12), 2275–2286. <https://doi.org/10.14778/3352063.3352143>
- Jannach, D., Manzoor, A., Cai, W., & Chen, L. (2020). A Survey on Conversational Recommender Systems. *ACM Computing Surveys (CSUR)*, 54, 1 - 36. <https://doi.org/10.1145/3453154>.
- Jannach, D., & Jugovac, M. (2019). Measuring the business value of recommender systems. *ACM Transactions on Management Information Systems*, 10(4), 1–23. <https://doi.org/10.1145/3370082>

- Kabakuş, A T., & Kara, R. (2017, October 1). A performance evaluation of in-memory databases. Elsevier BV, 29(4), 520-525. [https://doi.org/10.1016/j.jksuci.2016.06.007*\(Q2\)](https://doi.org/10.1016/j.jksuci.2016.06.007*(Q2))
- Kipf, T., & Welling, M. (2016). Semi-Supervised Classification with Graph Convolutional Networks. arXiv (Cornell University). <http://export.arxiv.org/pdf/1609.02907>
- Lagoze, C., Payette, S., Shin, E., & Wilper, C. (2005). Fedora: an architecture for complex objects and their relationships. *International Journal on Digital Libraries*, 6(2), 124–138. <https://doi.org/10.1007/s00799-005-0130-3>
- Lakshman, A., & Malik, P. (2010, April 14). Cassandra. Association for Computing Machinery, 44(2), 35-40. <https://doi.org/10.1145/1773912.1773922>
- Lex, E., Kowald, D., Seitlinger, P., Tran, T N T., Felfernig, A., & Schedl, M. (2021, January 1). Psychology-informed Recommender Systems. *Now Publishers*, 15(2), 134-242. <https://doi.org/10.1561/15000000090>
- Li, Y., & Manoharan, S. (2013, August 1). A performance comparison of SQL and NoSQL databases. <https://doi.org/10.1109/pacrim.2013.6625441>
- Lourenço, J. R., Cabral, B., Bernardino, J., & Vieira, M. (2015). Comparing NoSQL Databases with a Relational Database: Performance and Space. *International Journal of Big Data (Print)*, 2(1), 1–14. <https://doi.org/10.29268/stbd.2015.2.1.1>
- Lu, J., Wu, D., Mao, M., Wang, W., & Zhang, G. (2015, June 1). Recommender system application developments: A survey. Elsevier BV, 74, 12-32. <https://doi.org/10.1016/j.dss.2015.03.008>
- Nakhare, D. (2021). A Comparative study of SQL Databases and NoSQL Databases for E-Commerce. *International Journal for Research in Applied Science and Engineering Technology*, 9(12), 409–412. <https://doi.org/10.22214/ijraset.2021.39263>
- Osadchiy, T., Poliakov, I., Olivier, P., Rowland, M., & Foster, E. (2019). Recommender system based on pairwise association rules. *Expert Systems With Applications*, 115, 535–542. <https://doi.org/10.1016/j.eswa.2018.07.077>
- Paz, J. R. G. (2018). Learning Azure Cosmos DB Concepts. In *Apress eBooks* (pp. 25–59). https://doi.org/10.1007/978-1-4842-3351-1_2
- Quadrana, M., Cremonesi, P., & Jannach, D. (2018). Sequence-Aware recommender systems. *ACM Computing Surveys*, 51(4), 1–36. <https://doi.org/10.1145/3190616>
- Sarwat, M. (n.d.). ReCDB in Action: Recommendation made easy in relational databases. University of Minnesota, Twin Cities Minneapolis.
- Sarwat, M., Moraffah, R., Mokbel, M. F., & Avery, J. (2017). Database System Support for Personalized Recommendation Applications. University of Minnesota. <https://doi.org/10.1109/icde.2017.174>
- Solarz, A., & Szymczyk, T. (n.d). Comparison of database systems Oracle 19c, SQL Server 2019, PostgreSQL 12 and MySQL 8 database systems comparison
- Soni, P., & Yadav, N. S. (2015). Quantitative analysis of document stored databases. *International Journal of Computer Applications*, 118(20), 37–41. <https://doi.org/10.5120/20865-3587>
- Taipalus, T. (2023, January 3). Database management system performance comparisons: A systematic survey. <https://export.arxiv.org/pdf/2301.01095v1.pdf>
- Takano, Y., Iijima, Y., Kobayashi, K., Sakuta, H., Sakaji, H., Kohana, M., & Kobayashi, A. (2019). Improving document similarity calculation using Cosine-Similarity graphs. In *Advances in intelligent systems and computing* (pp. 512–522). https://doi.org/10.1007/978-3-030-15032-7_43

- Tenaya, G. a. P., Putra, I. D. P. G. W., Ekayana, A. a. G., Desnanjaya, I. G. M. N., & Ariana, A. a. G. B. (2022). Analisis Performansi Dua Sistem Operasi Server CentOS 8 dan Oracle Linux 8 Menggunakan Metode Levene Dengan SysBench. *Informal*, 7(1), 31. <https://doi.org/10.19184/isj.v7i1.30172>
- Truică, C., Rădulescu, F., Boicea, A., & Bucur, I. (2015). Performance evaluation for CRUD operations in asynchronously replicated document oriented database. Department of Computer Science, Faculty of Automatic Control and Computers, University Politehnica of Bucharest, Bucharest, Romania. <https://doi.org/10.1109/cscs.2015.32>
- Vaidya, N., & Khachane, A R. (2017, July 1). Recommender systems-the need of the ecommerce ERA. <https://doi.org/10.1109/iccmc.2017.8282616>
- Wang, Q. (2016, August 1). Design and implementation of recommender system based on Hadoop. <https://doi.org/10.1109/icss.2016.7883070>
- Wang, S., Zhang, X., Wang, Y., Liu, H., & Ricci, F. (2022, January 1). Trustworthy Recommender Systems. Cornell University. <https://doi.org/10.48550/arxiv.2208.06265>
- Wu, S., Sun, F., Zhang, W., Xie, X H., & Cui, B. (2022, December 3). Graph Neural Networks in Recommender Systems: A Survey. *Association for Computing Machinery*, 55(5), 1-37. <https://doi.org/10.1145/3535101>
- Xu, M. (2021). Understanding graph embedding methods and their applications. *SIAM Review*, 63(4), 825–853. <https://doi.org/10.1137/20m1386062>
- Yi, N., Li, C., Feng, X., & Shi, M. (2017). Design and implementation of Movie Recommender system based on Graph Database. University of Beijing. <https://doi.org/10.1109/wisa.2017.34>
- Zhang, S., Yao, L., Sun, A., & Tay, Y. (2019). Deep Learning based Recommender System: A survey and new perspectives. *arXiv (Cornell University)*, 52(1), 5. <http://arxiv.org/abs/1707.07435>
- Zhu, Z., Zhou, X., & Shao, K. (2019, April 1). A novel approach based on Neo4j for multi-constrained flexible job shop scheduling problem. *Elsevier BV*, 130, 671-686. <https://doi.org/10.1016/j.cie.2019.03.022>