Healthcare Data Records using Neo4j Graph Database Model

A Thesis submitted to the Faculty of Engineering & Technology,

Jadavpur University in partial fulfilment of the requirements for the

degree of

Master of Technology

in

Computer Technology

Submitted By

Bhaswati Barma

Examination Roll No.: M6TCT23004

Class Roll No.: 002010504023

Registration No.: 154188 of 2020-2021

Under the guidance of

Dr. Nandini Mukherjee

Professor, Department of Computer Science & Engineering, Jadavpur University

Department of Computer Science & Engineering

Faculty of Engineering & Technology

Jadavpur University

Kolkata-700032

September 2023

FACULTY OF ENGINEERING AND TECHNOLOGY JADAVPUR UNIVERSITY

CERTIFICATE OF RECOMMENDATION

This is to certify that the dissertation titled **Healthcare Data Records using Neo4j Graph Database Model** was completed by Bhaswati Barma, University RollNo: 002010504023, Examination Roll Number: M6TCT23004, University Registration No: 154188 of 2020-2021, under the guidance and supervision of Dr. Nandini Mukherjee, Professor, Department of Computer Science and Technology, Jadavpur University. The findings of the research detailed in the thesis have not been incorporated into any other work submitted to earn a degree at any other academic institution.

_____

Dr. Nandini Mukherjee

Professor, Department of Computer Science & Engineering

Jadavpur University

COUNTERSIGNED BY

_____

Dr. Nandini Mukherjee

Head of the Department

Department of Computer Science And Engineering

Jadavpur University

COUNTERSIGNED BY

_____

Prof. Saswati Mazumdar

Dean, FET

Faculty of Engineering and Technology

Jadavpur University

FACULTY OF ENGINEERING AND TECHNOLOGY
JADAVPUR UNIVERSITY

CERTIFICATE OF APPROVAL

This is to certify that the thesis entitled **Healthcare Data Records using Neo4j Graph Database Model** is a bonafide record of work carried out by BHASWATI BARMA in partial fulfilment of the requirements for the award of the degree Master of Technology in the Department of Computer Science and Engineering, Jadavpur University during the period of August 2022 to September 2023 (5^{th} & 6^{th} Semester). It is understood that by this approval the undersigned do not necessarily endorse or approve any statement made, opinion expressed or conclusion drawn there in but approve the thesis only for the purpose for which it has been submitted.

_____

Signature of Examiner
Date:

_____

Signature of Supervisor
Date:

DECLARATION

I certify that,

a. The work contained **Healthcare Data Records using Neo4j Graph Database Model** in this report has been done by me under the guidance of my supervisor.
b. The work has not been submitted to any other Institute for any degree or diploma.
c. I have conformed to the norms and guidelines given in the Ethical Code of Conduct of the Institute.
d. Whenever I have used materials (data, theoretical analysis, figures, and text) from other sources, I have given due credit to them by citing them inthe text of the thesis and giving their details in the references. Further, I havetaken permission from the copyright owners of the sources, whenever necessary.

Bhaswati Barma

Master of Technology

Roll No: 002010504023

Exam Roll No: M6TCT23004

Registration No: 154188 of 2020-2021

Department of Computer Science & Engineering

Jadavpur University, Kolkata-700032

ACKNOWLEDGEMENT

First and foremost, I want to express my gratitude to God Almighty for providing me with the strength, wisdom, and capability to go on this amazing adventure and to continue and successfully finish the embodied research work. I'd like to thank Dr. Nandini Mukherjee, Professor of the Department of Computer Science and Engineering at Jadavpur University for her excellent assistance, consistentsupport, and inspiration during my dissertation. I owe Jadavpur University a great debt of gratitude for providing me with the chance and facilities to complete our thesis.

I am grateful to every one of the teaching and non-teaching personnel whose assistance has made our trip during my research time much easier. I would like to thank my project guide Himadri Sekhar Roy for providing me with regular encouragement and mental support throughoutour effort.

Last, but not the least, my family deserves great recognition. There are no words to express my gratitude to my mother and father for all of the sacrifices you've made on my behalf. Your prayers for me have kept me going thus far.

Bhaswati Barma

Master of Technology

Roll No: 002010504023

Exam Roll No: M6TCT23004

Registration No: 154188 of 2020-2021

Department of Computer Science & Engineering

Jadavpur University, Kolkata-700032

Table of Content

Abstract……..……8

Chapter 1:

Introduction…….……..……..……11

1.1 Overview...…….……11

1.2 History and development of graph databases ……13

1.3 Importance of Graph Database ……..……...15

1.4 Some types of Graph Databases.....……...17

1.5 Case Study using Neo4j Graph Database...……...22

1.6 Objectives.……23

1.7 Outline of the thesis……...25

Chapter 2: Literature Survey ……26

Chapter 3: Proposed Work and Methodology...……....32

3.1 Introduction.…..……...32

3.2 Data Model.…….33

3.3 Synthetic Data Generation.……38

Chapter 4: Results and Discussion……...49

Chapter 5: Conclusion……...54

5.1 Future Scope……..56

References……...60

List of Figures

1.1 Graph representation of data……12

3.1 ER diagram for the Healthcare Database……37

3.2 GDB model of a small database of employees……38

3.3 The employee nodes whose food habit is Chinese food……..39

3.4 The employee nodes whose food habit is Chinese food and occupation is Engineer……40

4.1 Counts of every entity by the size of the dataset……..51

4.2 The File Size of every Entity by the Size of the Dataset……..52

4.3 The Execution time of a query by the size of the dataset……53

List of Table

1.1 Comparative Table for some Graph Databases……...19

2.1 An example of Execution time for MySQL vs. Neo4j…..…….…28

3.1 Table of Entities and Attributes of Healthcare Database……...33

Abstract

Graph databases are based on graph theory, just as SQL and RDBMS are based on set theory and logic. The relationships that Graph databases model are not data. Relational database management systems can now be replaced with graph databases. Applications that can be represented in a much more natural form include those in chemistry, biology, the semantic web, social networking, and recommendation engines.

Relational database management systems (Oracle, MySQL) and graph databases will be compared, with an emphasis on elements such as data structures, data model features, and query capabilities. The inherent and modern limits of some of the existing comparisons and contrasts between implementations of graph and relational databases will also be discussed. Graph problems look for the lowest collection of nodes that cover a graph, the shortest path among a set of paths in a transportation network, and so forth. Although there is no ANSI/ISO standard language for graphs, the most well-known ones include Neo4j, Orient DB, and Amazon Neptune. They frequently base their syntax on SQL or math.

In this paper, a healthcare database using NodeJS and connected to neo4j community edition as a type of graph database, with a maximum of three lakh datasets containing information about people, doctors, patients, histories, allergy histories, complaints, treatment episodes, and visits with their IDs, details, and other attributes. Utilizing NodeJS and the data saved in the provided Excel sheet, the data counts and data sizes for each graph

are computed at various intervals of the dataset. CQL is utilized in this case to analyze how long various queries take to execute.

A healthcare database using NodeJS and connected to neo4j community edition as a type of graph database, with a maximum of three lakh datasets containing information about people, doctors, patients, histories, allergy histories, complaints, treatment episodes, and visits with their IDs, details, and other attributes. Utilizing NodeJS and the data saved in the provided Excel sheet, the data counts and data sizes for each graph are computed at various intervals of the dataset. CQL is utilized in this case to analyze how long various queries take to execute.

LIST OF ABBREVIATIONS

SQL: STRUCTURED QUERY LANGUAGE

CQL: CYPHER QUERY LANGUAGE

MySQL: MY STRUCTURED QUERY LANGUAGE

GDB: GRAPH DATABASE

RDBMS: RELATIONAL DATABASE MANAGEMENT SYSTEM

OODBs: OBJECT-ORIENTED DATABASES

RDF: RESOURCE DESCRIPTION FRAMEWORK

PK: PRIMARY KEY

FK: FOREIGN KEY

ER: ENTITY-RELATIONSHIP

Chapter 1

Introduction

1.1 Overview

Since the latter half of the 1960s, relational databases have been used. Persistence, concurrency control, and integration techniques are all reliably provided. Tables defined by groups of rows and columns are kept in relational databases. While columns would be the attributes or properties of that item, a row may be thought of as the object itself. Its low capacity to explicitly represent requirement semantics is one of the relational model's limitations. The sciences have seen an increase in the prevalence of big data issues requiring intricately related information. The use of conventional RDBMS techniques makes storing, accessing, and managing such complicated data difficult. By definition, schema-based data models establish restrictions on how information will be kept. The schema needs to be redesigned manually to accommodate additional data. Graph databases like Neo4j are optimized for densely connected data, whereas the RDBMS is optimized for aggregated data. A graph is a type of data structure that consists of vertices and edges. When the focus of a data model is on the relationships between things, graph database technology is a useful tool for data modeling. Almost anything may be represented in a

matching graph by modeling objects and the connections between them. The property graph is a popular graph type that is supported by most systems. Property graphs are directed, attributed multi-graphs with labels. The property graph in Figure illustrates how interactions between people and objects are represented. Because every other form of graph is composed of subsets of the property graph implementation, the multi-graph has the advantage of being the most sophisticated implementation. This means that a property graph can accurately represent every other form of graph. The graph database is designed to process dense, connected datasets quickly and effectively. This layout enables the development of predictive models as well as the identification of connections and patterns.

Fast traversals along the edges between vertices are possible thanks to the highly dynamic data model, in which all nodes are linked together by relations. The fact that traversals are localized and do not need to consider sets of unrelated data is a special advantage. a SQL issue that is built-in.

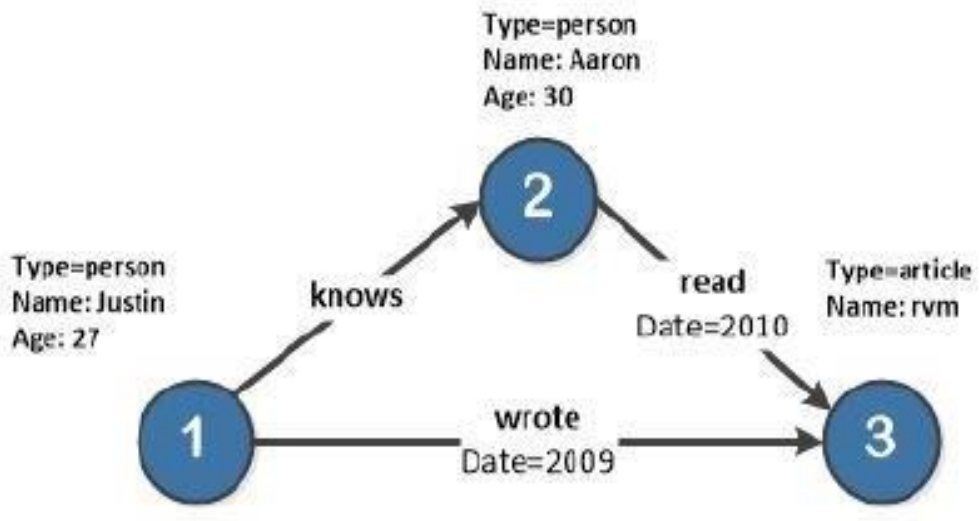

Figure 1.1: Graph representation of data

1.2 History and Development of Graph Databases

Graph databases have a long, illustrious history that goes back to the infancy of computer science. To effectively store and retrieve data with complicated relationships, they have developed over time. Here is a synopsis of the development and background of graph databases:

Hierarchical Databases (1960s):

The idea of arranging data hierarchically dates back to the 1960s, when systems like the IBM Information Management System were developed. These databases, which are regarded as an early type of hierarchical graph databases, represented data using a tree-like structure.

Network Databases (1960s-1970s):

In the 1960s and 1970s, network databases such as the Integrated Data Store and the Data Manipulation Language were developed. These databases popularised the notion of data representation as a network of connected nodes and edges, which is a key idea in graph databases.

Relational Databases (1970s-1980s):

The development of relational databases in the 1970s and 1980s, together with IBM's introduction of SQL, caused a shift in emphasis away from graph-like data representations and towards tabular structures. Because of

their ease of use and widespread adoption, relational databases have come to dominate the field of maintaining structured data.

Object-Oriented Databases (1980s-1990s):

Object-oriented databases grew in popularity in the 1980s and 1990s. Using object-oriented notions, OODBs enable developers to model complicated relationship-based data. Despite being more adaptable than relational databases, these databases still did not fully support the graph data model.

Emergence of Graph Databases (Early 2000s):

Graph databases, which are specifically made to handle data with complicated relationships, first became popular in the early 2000s. During this time, there were several significant developments including:

In 2007, Neo4j, one of the first and most significant graph databases, was made available. Neo4j made working with graph data simpler by introducing the property graph model and the Cypher querying language.

Particularly in the context of the Semantic Web, RDF, and triple stores also gained popularity as a means to encode and query graph-like data.

The Modern Era of Graph Databases (2010s–Present):

Graph databases have developed and become more common in a variety of fields. The current state of graph databases is shown by:

a. Integration with well-known programming languages and frameworks facilitates developers' use of graph data.

b. Adoption across a range of sectors, like knowledge graph applications, social networks, recommendation engines, and fraud detection.

c. The creation of several graph database systems, such as OrientDB, ArangoDB, JanusGraph, and Amazon Neptune, each with unique features and strengths, in addition to Neo4j.

d. Large-scale graph processing in distributed computing systems is made possible by improvements in scalability and distributed graph databases.

Since their origin, graph databases have come a long way, and as businesses see the benefits of modelling and querying data with complex relationships, their popularity is only increasing. They are now necessary instruments for resolving a variety of data-intensive problems in the current era.

1.3 Importance of Graph Databases

Data occupies a prominent role and is regarded as one of the most valuable assets for any organization in the modern digital age. The focus is now more on how the data may be handled to uncover hidden insights and information, rather than where to keep the data. These insights pave the way for growth, reduce obstacles in the future, and assist firms in making wise decisions. All of this resulted in the development of databases, which have continued to advance and change through time. There are several ways to organize and discipline this data. There are three types of databases: Graph, NoSQL, and SQL. The emergence of big data, the expansion of gigantic data, and the increase of data over the past ten years have been the forces that have propelled organizations and data scientists

to use these vast amounts of data to extract useful information. Data-rich organizations are seeing the value of extracting insight from their huge data, which is a trend in machine learning, an Artificial Intelligence branch that is always increasing. All of this has emphasized how significant Graph Databases are.

Modelling Complex Relationships:

Complex relationship data can be handled using graph databases, which are built expressly for this purpose. Data is interrelated in many real-world circumstances in complex ways that are difficult to express in conventional relational databases. For applications like social networks, recommendation engines, fraud detection, and supply chain management, graph databases are the best choice since they are excellent at representing and querying these relationships.

Flexibility:

The rigidity of the relational database model is one of its biggest flaws. Currently, in the year 2018, company needs are constantly evolving and changing, and relational databases were not designed to keep up with these scenarios. The problem can be solved by using nullable columns that are afterward editable, however, this isn't the best approach. Because they rely on relationships and nodes that are simple to generate and change based on the variables taken into account, graph databases do not have this issue.

Performance:

A graph database does indeed require additional storage capacity due to the need to store relationships. This indicates that the storage size increases proportionally to the graph's size. The relationships can be stored, though, which speeds up traversing. In this approach, without the requirement for JOIN procedures, the database engine may traverse the graph while looking at these relationships. Relationships are referred to as "first-class entities in graph databases" for another reason.

Scalability:

Scalability is a key consideration in the design of many graph databases. To manage large-scale graphs, they can divide data across several servers or clusters. Applications that need to expand as user bases or data volumes grow require this scalability.

1.4 Some types of Graph Databases

A database called a "graph database" prioritizes the links between the data as highly as the individual pieces of information. It represents and stores data using graph structures (node and edge). In graph databases, the record, object, or entity is represented by a node, and the link between nodes is represented by an edge.

Since graph databases are made to manage and store data in a graph style, they are effective at handling data with intricate relationships. There are

numerous varieties of graph databases, each with unique advantages and applications.

There are several graph databases and some of them:

Neo4j:

It's one of the best graph databases available, and strangely, it was created using Java. It also includes a language of its own, called Cypher, which is like declarative SQL but tailored to match graphs. Along with Java, it also supports several other well-known languages, including Python, .NET, JavaScript, and others. Neo4j is perfect for tasks like managing data centres and detecting fraud.

OrientDB:

A Java-based NoSQL database management system called OrientDB is used. It is a multi-model database that supports the object, document, graph, and key/value models. Graph databases use direct links between records to manage relationships. To manage the source code, contributors, and versioning, GitHub is used by the open-source community that OrientDB LTD leads. Users from all over the world can get free assistance via Google Group and Stack Overflow.

Amazon Neptune:

With the help of the fully managed graph database service Amazon Neptune, you may create and use programs that operate on datasets with a

lot of connections. A purpose-built, high-performance graph database engine that is optimized for storing trillions of relationships and performing rapid queries on the graph serves as the backbone of Neptune. It also supports the corresponding query languages for graph models like Property Graph and W3C's RDF, Apache Tinker Pop Gremlin, and SPARQL. For use cases like fraud detection and network security, Neptune is advised.

The data models, scalability, query languages, and use cases of these graph databases vary. The exact application needs you have as well as the type of data you have to work with should determine the graph database you use.

Name	**Neo4j**	**OrientDB**	**Amazon Neptune**
Description	Scalable, ACID-compliant graph database designed witha high-performance distributed cluster architecture, available in self-hosted and cloud offerings	Multi-model DBMS (Document, Graph, Key/Value)	Fast, reliable graph database built for the cloud
Server operating systems	Linux OS X Solaris Windows	All OS with a Java JDK (>= JDK 6)	Hosted

Replication methods	Causal Clustering using the Raft protocol	Multi-source replication	Multi-availability zones high availability, asynchronous replication for up to 15read replicas
Transaction concepts	ACID	ACID	ACID
User concepts	Users, roles, and permissions. Pluggable authentication with supported standards (LDAP, Active Directory, Kerberos)	Access rights for users and roles; record level security configurable	Access rights for users and roles can be defined via the AWS Identity and Access Management (IAM)
License	Open Source	Open Source	Commercial
Cloud-based only	No	No	Yes
Partitioning methods	Yes, using Neo4j Fabric	Sharding	None

Table 1.1: Comparative table for three Graph Databases

There are a few more graph databases:

ArangoDB:

Multi-Model Database (supports documents, graphs, and key-value). ArangoDB is a multi-model database that supports the key-value, document, and graph models of data. It enables you to manage several data kinds within a single database instance while working with related data. Use Cases: A range of applications, such as those for fraud detection, recommendation systems, and content management, make use of ArangoDB.

JanusGraph:

A database is a distributed Graph. An open-source distributed graph database that can scale across numerous servers is called JanusGraph. Both the RDF and property graph data models are supported, and it is intended for large-scale graph processing. Applications needing distributed graph processing, like social networks, geographic analysis, and recommendation systems, employ JanusGraph.

Stardog:

It is an RDF database type. An RDF database with a focus on handling knowledge graphs and semantic data, Stardog. It offers features for reasoning, ontology modelling, and sophisticated SPARQL query capabilities. Use Cases: The semantic web, data integration, and knowledge management applications all make use of Stardog.

RedisGraph:

It is an In-Memory Graph Database type. An in-memory graph database based on Redis is called RedisGraph. Real-time analytics and low-latency applications can benefit from its high-performance graph processing capabilities. RedisGraph is used in applications that require quick graph traversals, such as fraud detection and recommendation systems.

1.5 Case Study using Neo4j Graph Database

A well-known graph database is Neo4j. Oracle NoSQL Database, OrientDB, HypherGraphDB, GraphBase, InfiniteGraph, and AllegroGraph are other graph databases. Relationships in Neo4j should also be directed. Neo4j will generate an error stating that "Relationships should be directional" if we attempt to construct relationships without direction. All of the information in the Neo4j Graph Database is stored in Nodes and Relationships. To store Neo4j database data, we don't require either an additional RRBMS database or a SQL database. It keeps its data in its native format, which is Graphs. Neo4j works with its native graph storage format using the Native GPE (Graph Processing Engine). The three primary components of the Graph DB Data Model are nodes, relationships, and properties. Here, we've used circles to symbolize nodes. Arrows are a visual representation of relationships. The direction of relationships is clear. Data from Nodes can be represented as Properties (key-value pairs). Within the Node's Circle in this example, we have depicted each Node's ID property. To work with Neo4j, there is a built-in browse application. Neo4j can be

accessed at http://localhost:7474.The components of the Neo4j Graph Database include the following: Nodes, Properties, Relationships, Labels, and Data Browser are all included.

1.6 Objectives

For the goal of storing and navigating relationships, graph databases were created. In graph databases, relationships are treated as first-class citizens, and it is from these ties that most of their value is obtained. Nodes are used to store data entities in graph databases, and edges are used to store relationships between entities. An edge can indicate parent-child relationships, actions, ownership, and other things. An edge always has a start node, end node, type, and direction. The quantity and variety of connections that a node can have been both unlimited.

In a graph database, a graph can be visited along particular edge types or all across the graph. Because the associations between nodes are stored in the database rather than being calculated at query times, traversing joins or relationships in graph databases is highly quick. When you need to build linkages between data and quickly query these associations, use cases like social networking, recommendation engines, and fraud detection benefit from the use of graph databases.

When using a graph database, numerous query opportunities may be taken advantage of. The fundamental cause of this is the lack of a standard query language. Unlike conventional models, graph databases rely on specialized

algorithms to carry out their primary task of accelerating and streamlining challenging data searches.

The depth-first search and the breadth-first search are two of the most significant algorithms: In each scenario, the breadth-first search travels from layer to layer, but the depth-first search looks for the next node below. The techniques allow for the discovery of both direct and indirect nearby nodes as well as graph patterns. Using additional techniques, it is possible to find cliques (subsets of nodes), hotspots (information that is very heavily interconnected), and the shortest path between two nodes. The fact that relationships are contained in the database itself rather than having to be calculated in the query is one of the advantages of the graph database. As a result, even for complex queries, there is a good performance speed.

This Thesis aims to propose a graph data model that is suitable for healthcare database management and this is done by using the Neo4j community edition. Specific contributions made by this thesis are as follows:

- To learn about Graph Database Model which is different from RDBMS.

- To propose an efficient data model in Neo4j for a Healthcare Management system.

- To create a Neo4j Graph Database Model with the proposed data modelby connecting with NodeJS.

- To store the data for the numbers of each entity and file size of each entity for the different sizes of datasets and analyze the execution times for a query from different sizes of datasets of the data model.

- To draw some conclusions about the execution time of querying for the data model inNeo4j and discuss, how it is different in result for the case of RDBMS.

1.7 Outline of the Thesis

- In this thesis, first, a Literature survey about the Graph Database Model and Neo4j in general is in chapter 2.

- Chapter 3 shows the justification of the data model and synthetic data generation for healthcare database management with query execution time analysis.

- In Chapter 4, the results are discussed for the graph database model of healthcare database management.

- Finally in chapter 5, a few conclusions have been drawn and future scopes have been discussed about the graph database model and Neo4j.

Chapter 2

Literature Survey

Assimilation of raw facts or statistical data gathered for analysis or reference is referred to as data. It is the atomic object that needs to still be processed to yield some useful information. This information helps us enhance reality into digitally storable bits and bytes by frequently being used to depict the ecosystem around us in various ways. Relational DBMS came first, then object-oriented DBMS, XML databases, and finally NoSQL databases more recently. Relational DBMS, or RDBMS from here on, and NoSQL databases, particularly Graph databases, have been covered in this work. RDBMS stores data in the form of tables, i.e., in a 2D matrix where each cell or field is used to store the data and is identifiable by a row and a column.

There are numerous RDBMS programs, including MySQL1, MS SQL2, Oracle3, PostgreSQL4, etc. Variety, Volume, and Velocity are the three Vs that define big data. Variety refers to the different data formats of the data being generated and stored, Volume denotes the quantity of data, and Velocity denotes the rate at which this data is produced. All of this resulted

in the development of unstructured data, which are data that have little relationship among themselves. Because standard RDBMS were unable to adequately store this unstructured data, unstructured data-supporting databases were called upon, which led to the development of NoSQL databases.

There are numerous types of NoSQL databases, including graph databases, document-oriented databases, columnar databases, and key-value pair data stores. Only graph databases have been explored in this study. There are numerous well-known graph databases, including Neo4j, OrientDB, and Amazon Neptune. Relational databases retrieve and manage their database's data using SQL (Structured Query Language). Relational algebra is the foundation of the declarative language SQL. Graph databases make use of query languages for graph traversal. The strength of the graph depends on how well and quickly a query language can traverse it. Some of the query languages for graph data are Cypher for Neo4j, Gremlin, etc. Numerous applications for the semantic web, blogs, social networking, etc. should be implemented in graph databases, but this Employee table database is being studied and used. The employment of various tools and technologies during implementation is discussed in advance.

Neo Technology's Neo4j is an open-source, non-relational graph database. With its straightforward graph traversal operation queries, it is faster than relational databases when doing join queries. The implementation has been done using Neo4j Community-Edition-3.1.1.

The majority of data today comes in the form of relationships between various objects, and most of the time, these relationships are more useful than the individual pieces of data. Relational databases can be used to store highly structured data since they contain several records of the same type of data, but they do not store the relationships between the data. Graph databases hold connections and relationships as first-class entities, in contrast to conventional databases.

Theoretically, graph databases should traverse graphs much more quickly than relational databases. Find all of a user's friend's friends in a social network, for instance. For friends of friends of friends, even more so. The results of this query, which Alexa and Siri built in both MySQL and Neo4j using a database of 1,000,000 persons, are startling. Execution Time for 1,000 users is measured in seconds. Neo4j outperforms MySQL by 60% for the straightforward friends-of-friends query. Neo is 180 times faster for friends of friends of friends. Additionally, Neo4j is 1,135 times faster for depth four searches. Additionally, MySQL simply fails the depth 5 query. The outcomes are striking. The following figure has examples of some records for the execution time for MySQL and the execution time for Neo4j of different depths of data.

Depth	**Execution time-MySQL**	**Execution time-Neo4j**
2	0.016	0.010

3	30.267	0.168
4	1,543.505	1.359
5	Not Finished in 1 Hour	2.132

Table 2.1: An example of Execution time for MySQL vs. Neo4j

Neo4j is a NoSQL graph database that has been rapidly taking over the worlds of social networks and high-concurrency web applications. The academic community has been interested in the traits of supporting technology transactions and high scalability. This literature analysis therefore concentrates on analysing four study publications that were produced in the USA, Spain, India, and Germany. The authors display the outcomes of comparing Neo4j to other models, such as relational databases. The purpose of this paper is to provide examples of Neo4j's attributes, architecture, and benefits. Additionally, the goal is to determine whether Neo4j is a trustworthy replacement for the RDBMS (Relational Database Management System) and provide advice on how to do better trials.

College students are among the Internet's most ardent users, and social networking has become its most popular pastime. Unfortunately, the stuff that college students frequently publish on social networks puts them at risk. According to a study, students frequently share information on social networking sites that could harm their chances of finding employment. Although students are aware of the risks they are committing, many nevertheless submit inappropriate material.

Relational databases are still widely used in the data industry and are effective at managing transactional data storage. But with the rapid expansion of Relationship-rich applications (such as social graphs are the chosen option for networks as the model for storing and representing this data. A novel kind of data. Current platforms need to address enormous volumes of data and the expansion of information that is related (Lourenço et al., 2015a). Recently, a new database family called NoSQL, has grown significantly in popularity, especially due to the requirement to handle massive amounts of data exchanged among them, store, and recover them successfully.

A sort of non-relational database called a "graph database" offers an efficient and effective method for storing information in the current environment, where data are tightly interrelated. Graph databases are characterized as databases that replace relational databases by using graph topologies with nodes, edges, and properties to store data. In reality, however, they are an effective option when dealing with massive amounts of interrelated data. This essay's main objective is to examine the characteristics of graph databases, discuss their benefits, and assess Neo4j, the most well-known graph database.

Due to its use in fields like the Semantic Web and the Analysis of Social Networks, graph models have gained popularity in recent years. The key benefit is having instant access to complicated data, which is based, for instance, on social networks, search engines, data mining operations, and network systems.

Relational database management systems (RDBMSs) have long been the preferred option for data administration. Recent developments in large data have given rise to graph database management systems (GDBMSs), which are a crucial addition to RDBMSs. Both RDBMSs and GDBMSs are capable of managing relational and graph data, as noted in the literature now in existence; however, the distinctions between the two still need to be clarified. Due to this, we first expand a single benchmark for RDBMSs and GDBMSs using the same datasets, the same query workload, and the same metrics in this research.

We then carry out rigorous tests to assess them, and we come to the following conclusions: (1) Under workloads that primarily consist of group by, sort, and aggregation operations, as well as their combinations, RDBMSs perform significantly better than GDMBSs; (2) GDMBSs demonstrate their superiority under workloads that primarily consist of multi-table join, pattern match, path identification, and their combinations.

Chapter 3

PROPOSED WORK AND METHODOLOGY

3.1 Introduction

In this paper, we have worked on a healthcare database management system using the Neo4j, a Graph database model rather than the RDBMS model. Comparing the execution time of some relational queries from this database. The storage, management, and retrieval of data and information about healthcare are the functions of a healthcare database management system (DBMS), a specialized software system. To manage patient information, medical data, administrative chores, and more effectively, healthcare organizations must have these systems.

It takes careful planning, attention to rules, and continual maintenance to preserve data accuracy, privacy, and security while implementing and maintaining a healthcare database management system. The system should also be flexible enough to accommodate new medical procedures and

technological advancements. Using Neo4j, a graph database model, to create a healthcare database management system (DBMS) can be an effective strategy for handling complicated healthcare data and relationships between different entities in the healthcare ecosystem. Neo4j demands proficiency in database architecture, graph modeling, and healthcare domain knowledge to build a healthcare DBMS. To protect patient data privacy and security, it's also critical to collaborate closely with healthcare providers and follow any rules.

3.2 Data Model

First, discuss the data model for healthcare data record management in general. For that, we have to first know about the various entities present in the system and their attributes as well.

SL No.	Name of Entity	Their Attributes
1.	Person	Id name occupation socioEconomicStatus religion addressType addressLine1 addressLine2 cityTownVillagePoliceStation district state

		pinCode countryCode contact email gender dateOfBirthage bloodGroup foodHabbit.
2.	Doctor	drId personId organizationId specialization
3.	Patient	ptId Id alternatePtIds ecpId ecpRelation cPAHomeId cPAHomeRelation
4.	History	historyId ptId familyHistoryDetails pastHistoryDetails presentHistoryDetails personalHistoryDetails
5.	Allergy History	allergyHId ptId allergyFrom

		allergySevrity allergyStatus
6.	Complaint	complaintId ptId ComplaintDetails ComplaintStatus
7.	Treatment Episode	trtEpId ptId compId startTime endTime status
8.	Visit	vId ptId complained trtEpId visitType visitNumber timestamp reason observation summery diagnosisId sysBP disBP pulse temp tempSource

		respirationRate height weight investigationId treatmentPlanId

Table 3.1: Table of Entities and Attributes of Healthcare Database

The healthcare database management model's eight entities and their characteristics are defined in the table above, and each attribute is recorded in a database record for the healthcare management system. There is also an ER diagram of the entire database model, along with some attribute's primary key and foreign key. We may express the same information in various ways when using graph databases (Neo4j). Making each entity a Node and linking them all together via relationships, or just adding the entities to the Node's attribute list.

Some queries are used for the experiment:

1. Find all doctors' names and their details with the specialization string 'cardiologist'

2. Find all information of patients like demographic info, religion, food habits, etc. with the patient's name 'XYZ'

3. Find all ECG reports of a particular patient with the name 'x'

4. Find the history of 'Diabetes' of a particular patient with the name 'x'

5. Find sensor observed data of a patient for one hour

6. A patient treatment case history with the name 'B'

7. Find all the doctor's names who are attached to the organization whose id is "ABC-XYZ"

8. Find all the complaint details of the Patient with the name ‘Terry Medhurst’

9. Number of babies, whose ages between 0 to 5, suffering from malnutrition.

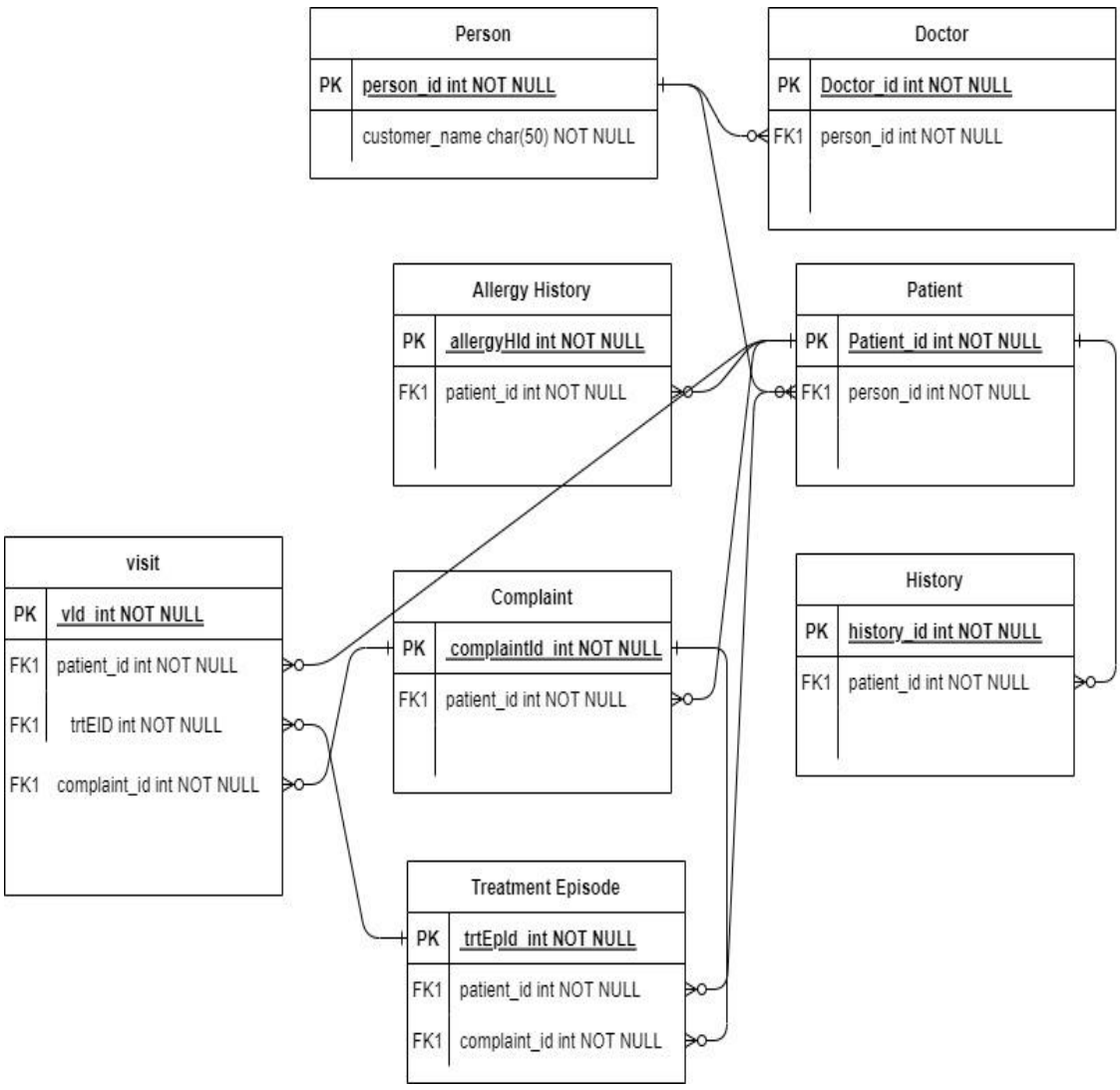

Fig. 3.1: ER diagram for the Healthcare Database

3.3 Synthetic Data Generation

Firstly, at the start of my final thesis work, I was working on a small database of Employees. an illustration of an employee table showing how the queries were written to relate to the entities. We launch the Neo4j community edition first, and then we launch the graph database for the employee table by visiting http://localhost:7474/browser/.

There are few person entity nodes and their attributes as nodes. The following figure represents the example of the employee small graph database model. There are the edges that represent the relationship between each entity. The two queries have run for this graph database model.

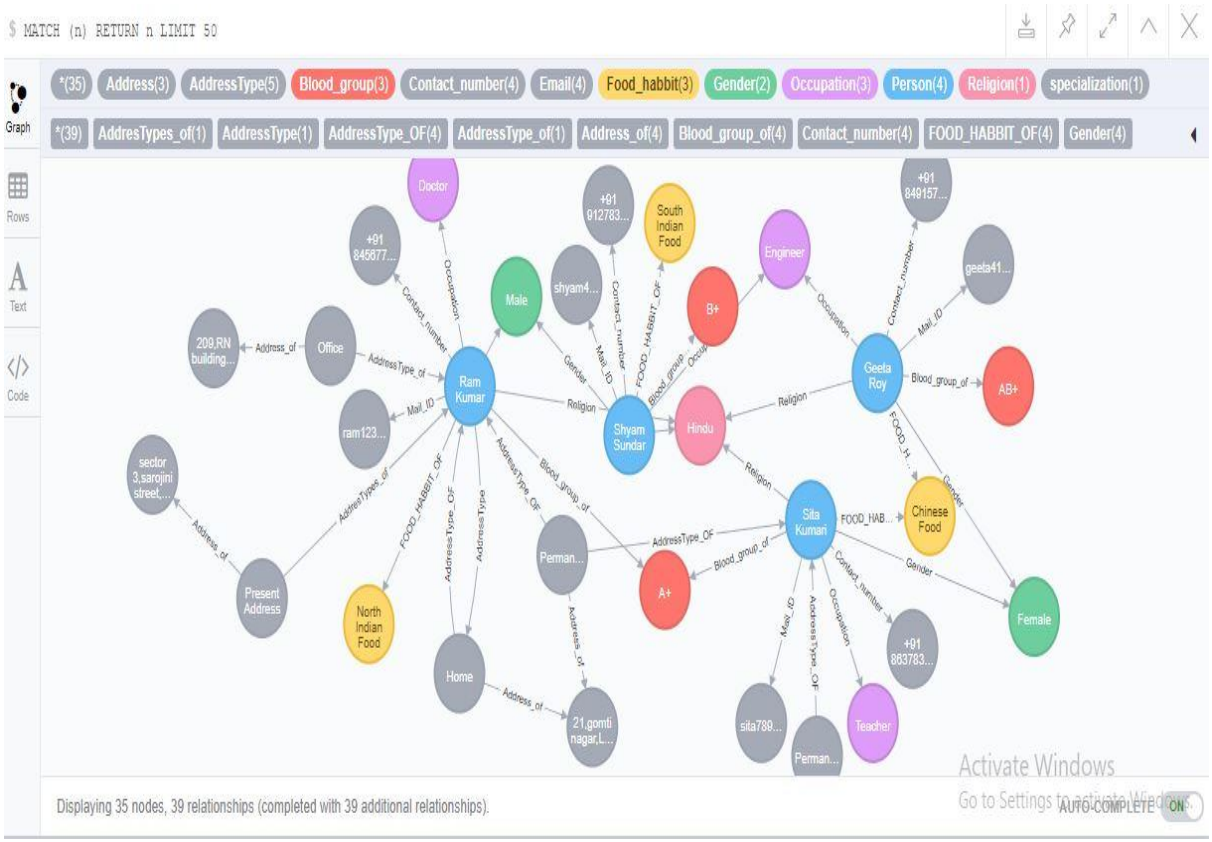

Fig.3.2: GDB model of a small database of employees

1. Show the name of the Employee whose food habit is Chinese food.

Sol. MATCH(a:Person)-[r:FOOD_HABIT_OF]→(b:Food_habit)
WHERE b.Name = “Chinese Food”
RETURN a

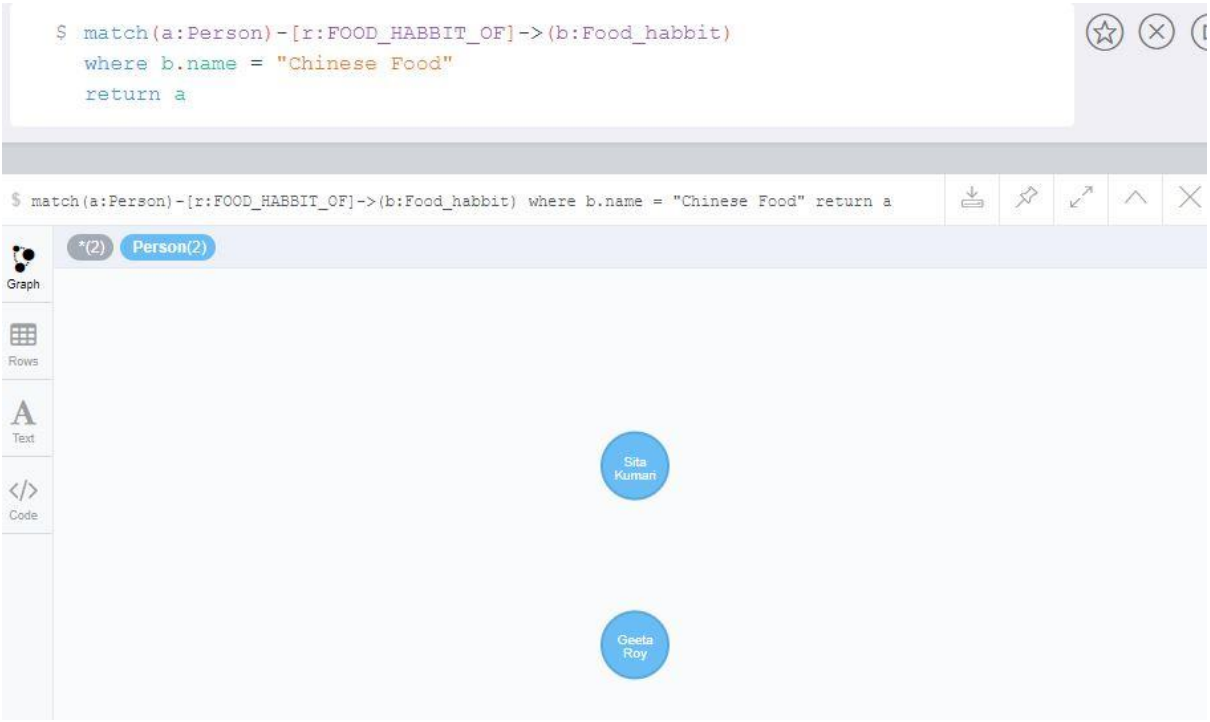

Fig. 3.3: the employee nodes whose food habit is Chinese food

2. Show the name of the Employee whose food habit is Chinese food and whose occupation is Engineer.

Sol. MATCH (b: Food_habit{name: “Chinese Food”})←[:FOOD_HABIT_OF]-(a: Person)-[:Occupation]→(c:Occupation {name: “Engineer”})
RETURN a

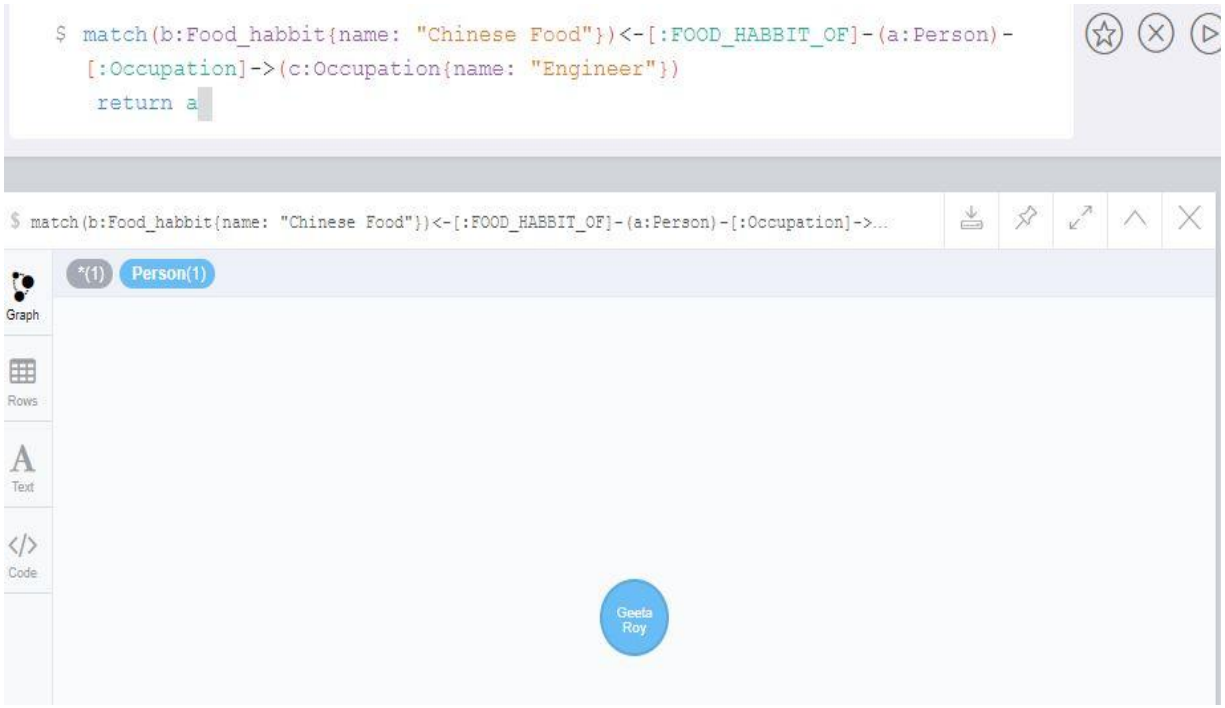

Fig. 3.4: the employee nodes whose food habit is Chinese food and whose occupation is Engineer

In this thesis work, we have worked on eight entities of healthcare database whose data model is already shown in the previous section and after creating the whole database model, the execution times of querying are recorded and stored. All queries ran on nodejs(v20) javascript runtime using the latest available official neo4j driver package namely neo4j-driver (V5.12.0).

Basically, the execution times of the query to find all the complaint details of the Patient with the name ‘Terry Medhurst’ are stored and shown in this thesis paper after connecting with neo4j and up to three lakhs of the dataset. A small part of the codes for the test query with five trials of an experiment to find the execution time:

```
const neo4j = require('neo4j-driver');

const testQuery = async () => {

const uri = 'bolt://127.0.0.1:7687';

const driver = neo4j.driver(uri, neo4j.auth. basic('neo4j', 'password'));

const session = driver.session();

const trial = 5;

let totalTime = 0;

for (let i = 1; i <= trial; i++) {

const time1 = Date.now();

ait session.run(Match (p:Person{name:"Terry Medhurst"}) <-- (d:Patient)
<-- (c:Complaint) return c;);

const timeTaken = Date.now() - time1;

totalTime += timeTaken;}

return totalTime / trial;};
```

The relational query trial is iterated 5 times and the average execution time of the data record is stored. Here, is the used software named Visual Studio Code 1.78.0 and Neo4j Community Edition 3.1.1.

All the nine queries have been run and executed. Query no. 8 in particular was recorded and the data Was stored in an Excel file. Overall, the other queries have yielded similar execution time compared with query no. 8

The query (CQL) to find all the complaint details of the Patient with the name ‘Terry Medhurst’:

Match (p: Person {name: "Terry Medhurst”}) <-- (d: Patient) <-- (c: Complaint) RETURN c;

The remaining queries are:

1. Find all doctors' names and their details with the specialization string 'cardiologist'

sol: MATCH (d: Doctor {specialization: 'cardiologist'}) RETURN d.name, d.details;

In this Cypher query:

MATCH (d: Doctor {specialization: 'cardiologist'}) searches for nodes labeled as "Doctor" with the property "specialization" equal to 'cardiologist'.

RETURN d.name AS DoctorName, d.details AS DoctorDetails specifies the properties you want to retrieve for each matching doctor node. You can adjust this list to include any additional details you want to fetch.

When you run this query in a Neo4j database (assuming your data model includes nodes labeled as "Doctor" with the specified properties), it will return the names and details of all doctors specializing in cardiology.

2. Find all information of patients like demographic info, religion, food habits, etc. with the patient's name 'XYZ'

sol: MATCH (p:Patient {name: 'XYZ'}) RETURN p;

In this Cypher query:

MATCH (p: Patient {name: 'XYZ'}) searches for nodes labeled as "Patient" with the property "name" equal to 'XYZ'. You should adjust the label and property name to match your data model.

RETURN p; specifies that you want to return all properties and relationships associated with the patient node 'XYZ'.

Running this query in Neo4j will return all information related to the patient with the name 'XYZ', including demographic info, religion, food habits, and any other properties associated with that patient node.

3. Find all ECG reports of a particular patient with the name 'x'

sol: MATCH (patient:Patient {name: 'x'})-[:HAS_ECG_REPORT]->(ecgReport:ECGReport) RETURN ecgReport;

In this Cypher query:

MATCH (patient: Patient {name: 'x'}) finds the patient node with the name 'x'. You should adjust the label and property name to match your data model.

[: HAS_ECG_REPORT]-> represents the relationship between the patient and their ECG reports. Modify this part to match the relationship type you've defined in your graph.

(ecgReport: ECGReport) specifies that you want to find nodes labeled as ECGReport that are connected to the patient through the HAS_ECG_REPORT relationship.

RETURN ecgReport; specifies that you want to return all ECG report nodes associated with the patient 'x'.

Running this query in Neo4j will return all the ECG reports of the patient with the name 'x'.

4. Find the history of 'Diabetes' of a particular patient with the name 'x'

sol: MATCH (patient:Patient {name: 'x'})-[:HAS_CONDITION]->(condition:MedicalCondition {name: 'Diabetes'}) RETURN patient, condition;

In this Cypher query:

MATCH (patient: Patient {name: 'x'}) finds the patient node with the name 'x'. You should adjust the label and property name to match your data model.

[: HAS_CONDITION]-> represents the relationship between the patient and their medical conditions. Modify this part to match the relationship type you've defined in your graph.

(condition:MedicalCondition {name: 'Diabetes'}) specifies that you want to find nodes labeled as MedicalCondition with the name 'Diabetes' that are connected to the patient through the HAS_CONDITION relationship.

RETURN patient, condition; specify that you want to return both the patient node and the medical condition node(s) representing the history of 'Diabetes' for the patient 'x'.

Running this query in Neo4j will return the patient and the 'Diabetes' medical condition nodes connected to that patient, representing the history of 'Diabetes' for the specified patient.

5. Find sensor observed data of a patient for one hour

sol: MATCH (patient:Patient {name: 'x'})-[:HAS_SENSOR_OBSERVATION]->(sensor:SensorObservation) WHERE sensor.timestamp >= timestamp() - duration('PT1H') AND sensor.timestamp <= timestamp() RETURN patient, sensor;

In this Cypher query:

MATCH (patient: Patient {name: 'x'}) finds the patient node with the name 'x'. You should adjust the label and property name to match your data model.

[: HAS_SENSOR_OBSERVATION]-> represents the relationship between the patient and their sensor observations. Modify this part to match the relationship type you've defined in your graph.

WHERE observation. timestamp >= timestamp () - duration('PT1H') AND observation. timestamp <= timestamp () filters the sensor observations based on their timestamps. It selects observations made within the last hour.

RETURN patient, observation; specifies that you want to return both the patient node and the sensor observation node(s) that occurred within the last hour.

This query assumes that sensor observations are linked to patients through relationships like HAS_SENSOR_OBSERVATION and that each observation has a timestamp property that records the time of the

observation. Adjust the relationship type and property names as needed to match your data model.

6. A patient treatment case history with the name 'B'

sol: MATCH (patient:Patient {name: 'B'})-[:HAS_TREATMENT_CASE]->(case:TreatmentCase) RETURN patient, case;

In this Cypher query:

MATCH (patient: Patient {name: 'B'}) finds the patient node with the name 'B'. You should adjust the label and property name to match your data model.

[: HAS_TREATMENT_CASE]-> represents the relationship between the patient and their treatment cases. Modify this part to match the relationship type you've defined in your graph.

(case: TreatmentCase) specifies that you want to find nodes labeled as TreatmentCase that are connected to the patient through the HAS_TREATMENT_CASE relationship.

RETURN patient, case; specifies that you want to return both the patient node and the treatment case node(s) associated with patient 'B'.

This query assumes that patient nodes are labelled as 'Patient', treatment case nodes are labelled as 'TreatmentCase', and there is a relationship 'HAS_TREATMENT_CASE' connecting patients to their treatment cases. Adjust the labels and relationship type as needed to match your data model.

7. Find all the doctor's names who are attached to the organization whose id is "ABC-XYZ"

sol: MATCH (organization:Organization {id: 'ABC-XYZ'})<-[:ATTACHED_TO]-(d:Doctor) RETURN d.name;

MATCH (organization: Organization {id: 'ABC-XYZ'}) finds the organization node with the specified id 'ABC-XYZ'. You should adjust the label and property name to match your data model.

<-[: ATTACHED_TO]-(doctor: Doctor) specifies that you want to traverse relationships of type 'ATTACHED_TO' in a reverse direction (from organization to doctor) and find doctor nodes connected to the organization.

RETURN doctor. name AS DoctorName specifies that you want to return the 'name' property of the doctor nodes as 'DoctorName'.

This query assumes that you have labelled your organization nodes as 'Organization', your doctor nodes as 'Doctor', and that there is a relationship 'ATTACHED_TO' connecting doctors to organizations. Adjust the labels and relationship type as needed to match your data model.

8. Number of babies, ages between 0 to 5, suffering from malnutrition

sol: MATCH (baby: Baby) WHERE baby.age >= 0 AND baby.age <= 5 AND baby.condition = 'malnutrition' RETURN COUNT(baby) AS NumberOfMalnourishedBabies;

In this Cypher query:

MATCH (baby: Baby) matches all nodes labelled as 'Baby'. You should adjust the label to match your data model.

WHERE baby.age >= 0 AND baby.age <= 5 filters the baby nodes based on their age, selecting only those with an age between 0 and 5 years.

AND baby. condition = 'malnutrition' further filters the babies to include only those suffering from malnutrition. You should adjust the property name ('condition') and value ('malnutrition') to match your data model.

RETURN COUNT (baby) AS NumberOfMalnourishedBabies counts the matching baby nodes and returns the count as 'NumberOfMalnourishedBabies'.

This query assumes that your data model includes baby nodes labelled as 'Baby', each with an 'age' property and a 'condition' property to indicate their health condition. Adjust the labels and property names/values to match your specific data model.

Chapter 4

Results and Discussions

For the various datasets, there exist records on the total number of each entity. We have observed that there are the same number of users across all dataset ranges. Ten percent of the users are doctors, leaving the rest users to be patients. The records are saved along with the counts of other entities that are produced by mapping and algorithms. The quantity of each object determines the size of each file. The final figure displays the execution timings, which are collected and maintained for the query to discover all the complaint information for the patient with the name "Terry Medhurst" from various dataset sizes. Thus, all of an entity's data are generated and saved in this manner. We observed that the query execution times throughout the range of dataset sizes are quite similar. It implies that the database system is operating at a high level of scalability and query optimization. This constancy can be attributed to a number of things. In conclusion, a mix of factors, including specialized query languages,

indexing, caching, optimization, scalability, and efficient data structures, can be blamed for the consistent execution time of queries for various dataset sizes in a graph database model. Graph databases are highly suited for situations where relationships between entities are important, regardless of the scale of the information, because they are built to excel at organizing and accessing graph data. Specialized query languages for rapidly traversing and querying graph topologies are frequently present in graph databases. These languages are designed for graph-based operations, such as Cypher in the case of Neo4j. Regardless of the size of the graph, the query planner and optimizer in the graph database engine can analyze the query and identify the most effective way to traverse the graph. This improvement supports consistent query performance. To hasten the retrieval of nodes and relationships, graph databases frequently employ effective indexing techniques, such as B-trees or customized graph-based indexes. These indices allow for easy identification of particular nodes and relationships. Even as the dataset size grows, a graph database's structure makes it possible to traverse node relationships quickly. Graph-based operations are well suited to this structure. Large graph databases occasionally use data partitioning or sharding techniques to spread the dataset across a number of servers or clusters. As the dataset expands, this distribution can help maintain constant query times and balance the query load. Caching techniques are commonly used in graph databases to temporarily store in-memory data and query results that are frequently retrieved. Particularly for regularly conducted queries or those containing common graph patterns, this caching can dramatically reduce query latency. The query execution plans generated by the graph database engine describe how the query will be carried out. The most effective data access paths are chosen in these strategies to optimize query performance. The engine may modify its execution strategy as the dataset size grows to ensure

effective query processing, but due to optimization efforts, the overall execution time may stay constant. Because they can scale horizontally, graph databases can accommodate growing data volumes and query demands by adding additional servers or nodes. The system can properly spread the burden as the dataset expands, maintaining query performance. The hardware and architecture that power the graph database may be sufficiently provisioned and well-configured to handle larger datasets without noticeably degrading performance.

	A	B	C	D	E	F	G	H	I
1	Size of the dataset	No. of Persons	No. of Doctors	No. of Patients	No. of Histories	No. of Allergy Histories	No. of Complaints	No. of Treatment Episodes	No. of Visits
2	1000	1000	14	986	7841	62800	2959	8744	26459
3	3000	3000	39	2961	23658	188795	9020	26873	81002
4	4000	4000	50	3950	31655	252359	11972	35672	107440
5	5000	5000	64	4936	39476	314901	14934	44537	133824
6	8000	8000	99	7901	63319	505427	23842	71177	213870
7	10000	10000	130	9870	79117	632332	29796	88988	267236
8	12000	12000	156	11844	94535	756071	35651	106519	319608
9	15000	15000	183	14817	118421	946428	44609	133588	401006
10	16000	16000	199	15801	126450	1010840	47608	142569	428082
11	20000	20000	259	19741	157625	1260496	59544	178479	535721
12	25000	25000	322	24678	197185	1577018	74520	223137	669976
13	30000	30000	388	29612	236763	1893788	89351	267648	803871
14	40000	40000	524	39476	315268	2521044	118955	356106	1069083
15	50000	50000	653	49347	393884	3149842	148366	444366	1334147
16	60000	60000	807	59193	472181	3776478	178075	533602	1600852
17	70000	70000	930	69070	551922	4415401	207655	622822	1868418
18	80000	80000	1046	78954	631276	5049905	237169	710366	2134111
19	90000	90000	1169	88831	709633	5677556	266658	798856	2398904
20	100000	100000	1291	98709	788237	6306928	296428	887389	2667768
21	110000	110000	1392	108608	867560	6940686	326262	977263	2937843
22	120000	120000	1520	118480	946504	7574124	355949	1066050	3204271
23	125000	125000	1591	123409	985989	7891086	370507	1109703	3334849
24	140000	140000	1775	138225	1104928	8840680	414892	1243230	3735434
25	150000	150000	1908	148092	1183731	9471807	444685	1331789	4003077
26	160000	160000	2052	157948	1263316	10108300	474295	1420763	4269768
27	175000	175000	2274	172726	1381256	11050834	518774	1553290	4671338
28	180000	180000	2343	177657	1420792	11366739	533462	1597155	4803061
29	200000	200000	2625	197375	1579490	12636859	592641	1774754	5334623
30	225000	225000	2985	222015	1776288	14212449	666553	1995988	5998640
31	300000	300000	3917	296083	2367833	18946654	889038	2661251	7999116

Fig. 4.1: The count of every entity by the size of the dataset

	A	J	K	L	M	N	O	P	Q
1	Size of the dataset	Size of Persons (KB)	Size of Doctors (KB)	Size of Patients (KB)	Size of Histories (KB)	Size of Allergy Histories (KB)	Size of Complaints (KB)	Size of Treatment Episodes (KB)	Size of Visits (KB)
2	1000	222.61	0.43	69.63	13509.26	2707.49	1329.67	870.68	37681.96
3	3000	668.07	1.68	255.56	41215.96	12323.64	4266.50	3562.00	118807.59
4	4000	891.23	2.25	348.43	55277.29	17173.13	5682.85	4867.74	158107.48
5	5000	1113.18	2.96	441.04	69029.83	21945.40	6111.94	6182.69	197310.46
6	8000	1789.77	4.41	720.83	111964.40	36482.17	11572.48	10236.50	316392.69
7	10000	2234.46	6.95	904.21	138451.50	46195.43	14469.64	13174.24	396226.66
8	12000	2671.42	7.13	1099.98	164878.08	57208.16	17015.15	15616.04	477274.88
9	15000	3345.53	8.93	1383.17	206573.00	70856.41	21832.36	19867.61	572431.13
10	16000	3572.64	10.36	1399.13	207020.16	72487.19	21412.81	20211.95	613866.29
11	20000	4468.63	13.36	1810.05	267357.85	94051.92	27944.52	24964.72	790204.14
12	25000	5377.37	14.77	2276.91	342231.20	118442.44	34405.84	31598.89	972421.63
13	30000	6696.03	17.41	2822.15	415485.56	123232.12	42649.77	32939.68	1012648.03
14	40000	8936.81	24	3798.01	552562.12	171847.45	57115.58	46087.11	1408087.21
15	50000	10735.03	29.24	4578.36	662925.91	210462.87	68439.82	56600.81	1725951.56
16	60000	12976.36	37.19	5555.22	800435.46	258291.68	82772.05	69943.97	2121826.72
17	70000	15216.51	43.71	6531.18	940117.43	306793.74	97068.77	83185.97	2517790.16
18	80000	17457.19	49.09	7507.89	1079234.11	355286.04	111359.67	96429.45	2911867.27
19	90000	19691.71	55.71	8489.14	1216402.71	403071.34	125595.38	109572.73	3305166.34
20	100000	21932.23	61.73	9465.31	1354520.76	451962.07	140272.24	122865.15	3708267.94
21	110000	24182.81	67.03	10499.18	1493484.08	500422.04	154771.21	136313.81	4108434.14
22	120000	26435.14	73.63	11519.51	1632474.08	548440.54	169217.98	149450.93	4502671.98
23	125000	27558.48	77.41	12031.44	1701751.84	572624.83	176267.04	155928.29	4696865.54
24	140000	30935.11	87.07	13572.13	1911684.59	644015.06	187355.69	175625.89	5056239.99
25	150000	33180.18	93.61	14595.65	2050116.07	692477.67	201797.82	188909.18	5454516.77
26	160000	35431.06	101.23	15620.22	2191041.63	740904.64	216193.13	202257.31	5852877.64
27	175000	38804.13	112.27	17148.78	2397071.62	812164.91	237391.88	222236.49	6448947.33
28	180000	39928.43	115.91	17662.14	2466532.94	836269.17	244487.26	250998.67	6644656.87
29	200000	44428.22	130.07	19703.55	2741588.25	912046.82	267134.02	272882.18	7279522.06
30	225000	50038.59	148.53	22261.41	3097779.08	118861.91	301352.02	306590.05	8260342.43
31	300000	67509.07	198.75	29464.58	3857966.43	395277.81	386263.15	377096.12	10794975.06

Fig. 4.2: The File Size of every entity by the size of the dataset

	A	R
1	**Size of the dataset**	**Time Execution for Relation Query (ms)**
2	1000	2213.8
3	3000	732.4
4	4000	709.2
5	5000	1080
6	8000	1317
7	10000	694.2
8	12000	795.6
9	15000	1162.2
10	16000	735.8
11	20000	1739.6
12	25000	4220
13	30000	3610
14	40000	4633.6
15	50000	3020.2
16	60000	2399.8
17	70000	2973
18	80000	1948
19	90000	2494.2
20	100000	1756.8
21	110000	2465
22	120000	3468.4
23	125000	3510.4
24	140000	4405.4
25	150000	3002.2
26	160000	3294.8
27	175000	4345.2
28	180000	4207.4
29	200000	3445
30	225000	2501.4
31	300000	2123.4

Fig. 4.3: The Execution time of a query by the size of the dataset

Chapter 5

Conclusion

The observation that a query's execution time in a healthcare graph database model is largely unaffected by the query size can be attributed to several factors, each of which suggests further research and development areas.

Firstly, it seems the healthcare graph database model has been well-designed in the realm of effective indexing and storage techniques. The deployment of proficient indexing and storage strategies has fostered consistency in query performance across varying data volumes. This aspect invites a deeper exploration into the specific techniques employed, with a

vision to potentially enhance them further, leveraging the existing strengths to achieve even higher efficiency.

Next, focusing on the scalability of the system, the consistent execution time across different query sizes indicates a scalable system. This is a vital attribute in the healthcare industry where the database is continually burgeoned with new entries such as patient records, therapies, or research data. Given this consistency in performance, it can be inferred that the system can handle a significant increase in data without compromising on the execution time. However, it is prudent to remain open to the possibility of further optimizations, particularly in enhancing the query speed or reducing resource usage, to ensure sustained performance in the face of an ever-growing dataset.

Thus, while the current findings are promising, they pave the way for further studies aimed at not just maintaining this level of performance, but potentially exceeding it to foster a healthcare graph database model that is both robust and highly efficient. This continual pursuit of improvement is essential to keep pace with the dynamic and expansive nature of healthcare data.

In conclusion, the utilization of graph databases in healthcare management presents a wide array of opportunities, ranging from fostering personalized medicine to enhancing healthcare operations. As healthcare systems

worldwide continue to evolve, graph databases stand as a potent tool in the journey toward a more integrated, efficient, and patient-centred healthcare paradigm. Future endeavours should focus on leveraging the unique capabilities of graph databases to address the multifaceted challenges and opportunities in the dynamic landscape of healthcare management.

5.1 Future Scope

Graph databases have evolved to become a staple in the information technology landscape, facilitating numerous business operations with unprecedented efficiencies. No longer confined to the innovative labs of startups, they have transcended mainstream usage, with a promising trajectory underscored by statistical forecasts. According to a 2017 Dataversity poll, it was projected that 22.6% of data professionals would be utilizing graph databases in the foreseeable future. Fast forward to the present, and Gartner reinforces this optimistic outlook, anticipating a yearly increase in graph technology applications through 2022.

The integration of graph databases in healthcare management heralds a new frontier in healthcare delivery and administration, offering substantial prospects for optimizing various facets of the healthcare system. Below, we delve into potential avenues and objectives that illustrate the future scope of healthcare management using graph databases:

1. Personalized Medicine

1.1 Genomic Data Analysis

Objective: To facilitate the development of personalized therapeutic strategies based on individual genetic profiles.

Approach: Utilize graph databases to store and analyze complex genomic data, facilitating the identification of patterns and relationships critical for personalized medicine.

1.2 Drug Discovery

Objective: To streamline the drug discovery process by identifying potential drug targets more efficiently.

Approach: Leverage graph databases to analyze biological networks and pathways, aiding in the identification of novel drug targets.

2. Integrated Healthcare Networks

2.1 Patient-Centric Care

Objective: To offer a more coordinated and personalized approach to patient care by linking all aspects of a patient's healthcare journey.

Approach: Use graph databases to integrate data from various healthcare providers, offering a comprehensive view of a patient’s health history and facilitating coordinated care.

2.2 Fraud Detection

Objective: To enhance the security of healthcare systems by detecting fraudulent activities more effectively.

Approach: Implement graph databases to analyze patterns and relationships in healthcare claims data, aiding in the early detection of fraud.

3. Research and Development

3.1 Knowledge Graphs

Objective: To foster a rich ecosystem of interconnected data that can enhance research and development efforts.

Approach: Develop knowledge graphs using graph databases to integrate various data sources, facilitating multidisciplinary research and innovation.

3.2 Real-World Evidence (RWE)

Objective: To leverage real-world data to inform healthcare decisions and policy-making.

Approach: Utilize graph databases to integrate and analyze real-world data, providing a robust foundation for evidence-based healthcare.

4. Healthcare Operations

4.1 Resource Optimization

Objective: To optimize healthcare operations by facilitating better resource allocation and management.

Approach: Implement graph databases to analyze relationships and patterns in healthcare operations data, aiding in more informed decision-making.

4.2 Predictive Analytics

Objective: To enhance healthcare delivery through predictive analytics.

Approach: Leverage graph databases for predictive analytics, helping to forecast healthcare trends and enhance decision-making processes.

REFERENCES

[1] Safikureshi Mondal and Nandini Mukherjee (2021). Efficient NoSQL Graph Database for Storage and access data of health care.

[2] Singh, M., & Kaur, K. (2015) SQL2Neo: Moving health-care data from relational to graphdatabase. 2015 IEEE International Advance Computing Conference (IACC). doi:10.1109/iadcc.2015.715480110.1109/iadcc.2015.7154801.

[3] Arias, J. F. (2020). The Benefits of Graph Databases for the Computation of Clinical Quality Measures. 2020 IEEE 33rd International Symposium on Computer-Based Medical Systems (CBMS). doi:10.1109/cbms49503.2020.00088.

[4] Justin J. Miller, Proceedings of the Southern Association for Information Systems Conference, Atlanta, GA, USA March 23rd-24th, 2013, Graph Database Applications and Concepts with Neo4j.

[5] López, F.M.S. and De La Cruz, E.G.S., 2015. Literature review about Neo4j graph database as a feasible alternative for replacing RDBMS. Industrial Data, 18(2).

[6] Miller, R.E., 2013. The posting paradox: Facebook vs. Twitter. In Proceedings of the Southern Association for Information Systems Conference, Savannah, GA.

[7] Cheng, Y., Ding, P., Wang, T., Lu, W. and Du, X., 2019. Which category is better: benchmarking relational and graph database management systems? Data Science and Engineering.

[8] https://www.tutorialspoint.com/neo4j/neo4j.

[9] https://www.tutorialspoint.com/Graph-Databases.

[10] https://solutionsreview.com/data-management/the-best-graph-data-bases/

[11] https://6point6.co.uk/insights/use-cases-for-graph-databases/