Database Management System (DBMS) Relational and Graphs

***Abstract*—Data storage and processing have become increasingly important in recent years. A database management system is software for defining, creating, managing, and controlling database access. This paper compares and contrasts two different types of database management systems. It compares the performance of relational versus graph-based database systems in terms of structure queries, union and intersection operations. When the database size is small, the experimental results demonstrate that relational database management systems perform better for data insertion and data searching, but as the database size grows, graph database management systems become more efficient. This can assist data professionals in discovering unexpected connections between data points, making them an excellent solution for use cases involving relationships.**

***Index Terms*—Database , Database management systems, DBMS, Relational, Graph.**

I. Introduction

FOR efficient processing and decision-making, an organization needs accurate and reliable data. To that aim, the organization keeps track of the many aspects while maintaining ties between them. A database system is a collection of related data and files that includes information about how the data in those files were interpreted. A database system is essentially a computer-based record-keeping system, that is, a system whose primary goal is to record and maintain data.

A database management system (DBMS) is simply a software-based system for providing regulated and managed data access to applications. The database management system frees the application from many of the onerous aspects in the care and feeding of its data by permitting a distinct definition of the data structure. It saves information and gives tools for organizing it. Modern database technologies (relational and later) are built on the separation of the logical representation of data and its physical instantiation, allowing one to be modified without impacting the other.

Relational database management solutions have been the preferred method of data administration for decades. Graph database management systems have recently developed as an important addition to relational database management systems due to the range of features of massive data [1]. To ensure that decisions are made correctly, the decision-making process must be based on timely and relevant facts and information. Data management and processing yield information, which is carried out by information systems with the help of information technology.

This paper focuses on database management systems involving relational and graphs database management systems, comparison, and experiments on both. The remainder of this paper is structured in the following manner. Section II Phases in database design. Section III Relational database, structures, normalization, feature, and SQL. Section IV Graph database management systems, applications, features, types of graphs, and Neo4J. Section V shows a comparison of some functionalities on relational and graphs DBMS. Section VI displays experiments on both. Section VII concludes the paper.

Aliyu Bello

II. Phases in Database Design

Database development is generally a component of the larger systems development process rather than being separate from it. The stages of systems development are similar to those of the database lifecycle. While database design focuses on the technology that will hold the data, systems design also considers the processes that will affect the data.

First phase: The general goal of the database initial study is to assess the current state of the organization/system.
- Identify the issue and restrictions.
- Determine goals.
- Define the scope and limitations.

Second phase: The database model that will support organisational activities and objectives is designed in the second phase. We can distinguish six major phases of database design at this phase:
- Gathering and analysing requirements.
- Database design concept.
- Database Management System.
- Database system implementation - Data model mapping.
- Physical database design.

III. RELATIONAL DATABASE

A relational database separates data into tables that can be linked (or related) based on shared data. You can construct a new table from data in one or more existing tables with only one query. It also allows you and your firm to better understand the connections between all available data and gain new insights for better decision-making and new opportunities. The relational database (RDB), which is based on the relational concept, was developed more than 30 years ago to facilitate commercial data processing [2]. It has since evolved into the best solution for keeping data ranging from bank records to personal information and much more. Processes like data warehouses, text management, and stream processing have quite different requirements than normal corporate data processing.

Relational databases (RDBs) have a number of advantages over other types of data storage (Codd, 1970; Harrington, 2002). The employment of indices and other optimizing devices can result in significant speed improvements for searching. As a result, redundancy and storage space is reduced,

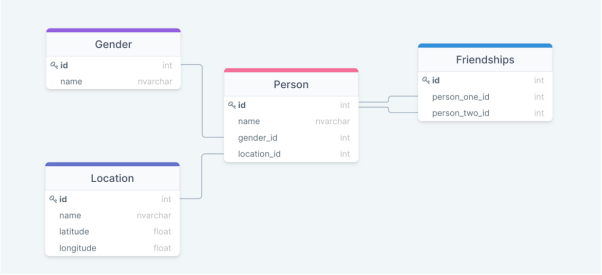
allowing for the recording of minute details that would otherwise be lost. A single computer handles data, which can be readily backed up and/or mirrored by a second computer. It is also simple to access multiple data sets while using a single, centralized database machine.

1) ACID: A transaction in database systems refers to a single logical operation on data, such as putting data into a database. A collection of qualities, including Atomicity, Consistency, Isolation, and Durability (ACID), were proposed to achieve transaction reliability.

- Atomicity: Each transaction should be atomic (all or nothing), meaning it should be considered a complete item. If any part of the transaction crashes, the actual procedure should fail without any database changes.
- Consistency: The consistency attribute ensures that a transaction moves the database from one legitimate state to another once it has been completed correctly. It is necessary to keep the database consistent regardless of whether the transactions are correct or not.
- Isolation: The isolation property specifies that each running transaction should be isolated from other concurrent transactions until it is properly completed and committed. Incomplete transactions should therefore have no effect on other transactions.
- Durability: When a transaction commits, the changes in the database should remain the same, even if the machine crashes, the power goes out, or an error occurs. This feature ensures that the execution results are permanently recorded.

A. Structure of Relational DBMS

Every program saved data in its own unique structure in the early days of databases. When developers sought to create apps that used that data, they needed to understand a lot about the data structure in order to access the information they required. These data structures were wasteful, difficult to maintain, and difficult to tune for excellent app performance. [3] The relational database model was created to address the issue of dealing with various arbitrary data structures.



Structure of RDBMS[4]

Any application could use the relational data model since it provides a common manner of representing and querying data. From the start, engineers realized that the relational database model's main strength was its use of tables, which provided an easy, fast, and flexible way to store and access structured data.

B. Normalization

The method of arranging data into tables in such a way that the outcome of utilizing the database is always clear and works as intended is known as database normalization. Relational database theory requires such standardization. It has the potential to duplicate data in the database and frequently leads to the development of new tables.

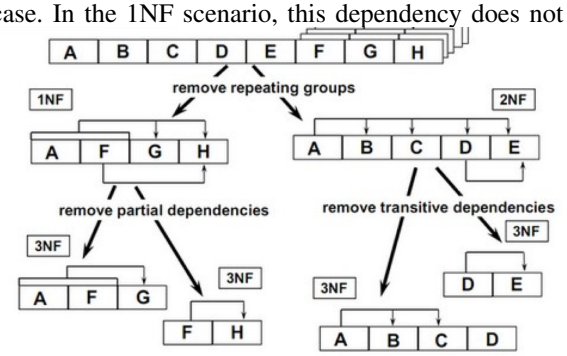
First Normal Form (1NF)

This is the "fundamental" level of database normalization, and it roughly corresponds to any database's definition, which is:

There are rows and columns in two-dimensional tables. Each column represents a subobject or property of the object that the full table represents. A single instance of that is represented by each row subobject or attribute, and it must be distinct from the others in some way (that is, no duplicate rows are possible). In any column, all entries must be of the same kind. Only customer names or numbers are allowed in the "Customer" field, for example.

Second Normal Form (2nd NF)

Each column in a table that is not a determiner of the items of another column must be a function of the other columns in the table at this level of normalization. The pricing would be a function of the customer ID (entitled to a discount) and the specific product in a database with three columns containing the customer ID, the product sold, and the price of the product when sold. The third column's data is considered to be dependent on the first and second columns' data in this case. In the 1NF scenario, this dependency does not exist.[5]



Normalization process[6]

Third Normal Form (3rd NF)

Since a change to one row in a database may affect data that references this information from another table, updates are still feasible in the second normal form. Using the previously mentioned customer table, removing an entry detailing a customer purchase (maybe due to a return) will also delete the fact that the product has a certain price. These data would be separated into two tables in the third normal form so that product prices may be tracked separately.

Boyce–Codd Normal Form (BCNF)

The Boyce–Codd Normal Form (BCNF) is built on functional dependencies that take into account all candidate keys in a relation; nevertheless, as compared to the general definition of 3NF, BCNF includes extra constraints.

C. Features OF RDBMS

Simplicity: This phrase should be self-explanatory. A single, consistent data structure is offered to the relational user. He

expresses his needs solely in terms of information substance, ignoring system-related complexity.

Data independence: This is described by C. J. Date [Zll] as ”the ability of programs to adapt to changes in storage structure and access technique.” As we’ve seen, the relational model allows us to remove storage structure and access strategy specifics from the user interface. [7]

- Simplicity: This phrase should be self-explanatory. A single, consistent data structure is offered to the relational user. He expresses his needs solely in terms of information substance, ignoring system-related complexity.
- Data independence: This is described by C. J. Date [Zll] as ”the ability of programs to adapt to changes in storage structure and access technique.” As we’ve seen, the relational model allows us to remove storage structure and access strategy specifics from the user interface.
- Symmetry: Some questions are easier to pose than others in database systems based on links between records, such as queries whose structure matches that of the database. In a hierarchical data system, for example, the easiest query to ask is one that starts at the root and progresses to the leaves, applying successive qualifications at each step. Questions that do not follow this recommended pattern may be awkwardly asked, if at all.
- Strong theoretical foundation: The relational data model is based on the first-order predicate calculus and a well-developed mathematical theory of relations. This theoretical foundation allows for the definition of relational completeness and a thorough examination of optimal database architecture (normalization)

The relational model also includes a number of secondary benefits that stem from the fundamental benefits previously mentioned. The simplicity with which high-level, nonprocedural relational languages can be defined is perhaps the most important of them.

D. Structured Language Query (SQL)

SQL is not a complete programming language; rather, it is a data sublanguage. SQL consists only of constructs for defining and processing a database. To obtain a full programming language, SQL statements must be embedded in scripting languages, such as VBScript, or in programming languages, such as Java or C. SQL statements also can be submitted interactively using a DBMS-supplied command prompt.

SQL FOR DATA MANIPULATION (DML)—SINGLE TABLE QUERIES

This section introduces the fundamental statement framework for SQL query statements. After we discuss this basic structure, you will learn how to submit SQL statements to Microsoft Access, SQL Server, Oracle Database, and MySQL/

- • The SQL SELECT clause specifies which columns are to be listed in the query results.
- • The SQL FROM clause specifies which tables are to be used in the query.
- • The SQL WHERE clause specifies which rows are to be listed in the query results.

```
SELECT    Name, id, gender_id, location_id
FROM      Person
WHERE     gender_id = 'Male'
OR        location_id = 'Dubai'
```

When SQL statements are executed, the statements transform tables. SQL query statements start with a table, process that table in some way, and then place the results in another table structure. Even if the result of the processing is just a single number, that number is considered to be a table with one row and one column. As you will learn at the end of this chapter, some SQL statements process multiple tables. Regardless of the number of input tables, though, the result of every SQL statement is a single table. SQL provides math operations and a shorthand notation for querying all of the columns of a table.

IV. GRAPHS DATABASE MANAGEMENT SYSTEMS

Graph databases are specifically intended to store and navigate relationships. In graph databases, relationships are first-class citizens, accounting for the majority of the database’s value. In graph databases, nodes hold data entities, while edges represent associations between things. A start node, an end node, a type, and a direction constitute an edge, which can be used to describe parent-child connections, actions, and ownership, among other things. A node can have an infinite number and variety of relationships. A graph is made up of nodes and their relationships. Nodes have properties and are organized by relationships that have their own attributes. A traversal looks for pathways that organize nodes in a network.

A. Applications of Graphs Databases

When working in areas where data interconnection or topology is vital, graph databases really shine. The relationships between data and the data itself are frequently at the same level in such applications. To meet the demand for graph database systems, many firms have built in-house implementations. Facebook’s Open Graph, Google’s Knowledge Graph, Twitter’s FlockDB, and others are examples. [8] The following are some instances of systems that would considerably benefit from using a graph database. RDBMS can be utilized for such systems, but only in a limited and costly manner (expensive meaning processing power caused by recursive JOINs for a friend of a friend type problems).

Structure of Graphs [9]

B. Features of Graphs Databases

Graph databases are a strong tool for graph-like queries. For example, calculating the shortest path between two nodes in a network. Other graph-like queries (such as graph diameter computations or community discovery) can be performed organically using a graph database. Graphs are adaptive, allowing the user to add new data to an existing graph without having to restart the application. The database designer does not need to draw out all of the database's future use cases in great detail.

- Storage: Graph databases have a variety of storing mechanisms. Some rely on a relational engine and use a table to store the graph data. Others store data in a key-value store or a document-oriented database, making them fundamentally NoSQL. A node is represented in the same way as any other document store, but edges that connect two nodes have unique features in their documents.
- Index-free adjacency: The performance of data lookups is determined by the access speed from one node to the next. Nodes with index-free adjacency have direct physical RAM addresses and physically point to other nodes, allowing for faster retrieval. A native graph system with index-free adjacency does not need to go through any other forms of data structures to find linkages between nodes. Directly connected nodes in the network are cached once one of the nodes in the graph is obtained, making data search even faster than the first time a user obtains a node

C. Graphs Types

- SOCIAL: Facebook, Twitter, and the concept of six degrees of separation are instances of social graphs. A connection between people
- Intent: This is about logic and motivation.
- Consumption: The consumption graph, often known as the "payment graph," is widely utilized in the retail industry. Consumption graphs are used by e-commerce corporations like Amazon, eBay, and Walmart to measure individual customers' consumption.
- Interest graphs: This highlights a person's preferences and is frequently accompanied by a social graph. It has the ability to replicate the earlier online organizing revolution by mapping the web by interest rather than indexing web pages.
- Mobile graph: This was created using cellphone data. Data from the web, applications, digital wallets, GPS, and Internet of Things (IoT) devices may all be included in future mobile data.

D. Neo4j

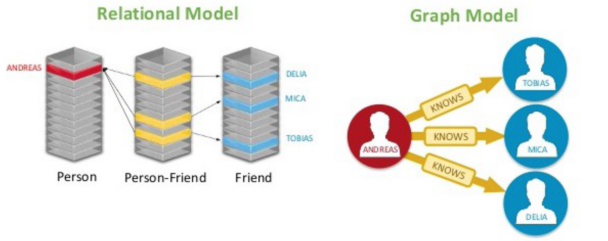
Neo4j, Inc. created a graph database management system called Neo4j. The database's developers describe it as an ACID-compliant transactional database with native graph storage and handling. Neo4j is written in Java and may be accessed via a transactional HTTP endpoint or the binary "Bolt" protocol from software written in other languages that use the Cypher query language.

- Neo4J Data Structure

Everything is saved as an edge, node, or attribute in Neo4j. There are no limitations to how many characteristics each node and edge can have. Labeling is possible for nodes and edges. Searches can be narrowed using labels. With the introduction of schemas in version 2.0, indexing became available in Cypher. Cypher previously supported indexes independently.

V. RELATIONAL DBMS AND GRAPHS DBMS COMPARISON

One of the relational model's design aims was to allow for rapid row-by-row access.Problems emerge when complex associations between stored data must be constructed. Although the relational model can be utilized to analyze relationships, it necessitates sophisticated queries that perform numerous join operations across multiple tables[10]. While working with relational models, foreign key constraints should be addressed when extracting relationships, as they create significant overhead. For associative data sets, graph databases are often faster than relational databases, and they transfer more directly to the structure of object-oriented programs. Because they don't generally require expensive join operations, they can scale to large datasets more organically. Because they don't rely on a strict structure, they're said to be superior for managing ad hoc and changing data with emerging schemas.



Relational and Graphical database management system comparison [11]

A.

- Data Model: All data is represented as mathematical relations, with a relation being a subset of the Cartesian product of N domains in relational databases based on set notions in mathematics. The data in the database is organized into relations and represented as tuples. The relation (represented by a table) has a set of Tuples (rows) that represent a sequence of attributes named column in the relation table. The type of an attribute is determined by the domain, which is a collection of values with an ordinary meaning. [12] This data model is highly detailed and ordered. A well-defined schema describes columns and their contents. The structure of the set of corresponding data stored in rows is the same. Tables, on the other hand, are not required in the graph model. Simply create nodes for things and their relationships. We

may roughly use the ER diagram in the graph database. There's no need to establish referential constraints, think about normalization and denormalization, or remember the join keys across tables. As a result, modeling complex real-world scenarios is significantly easier than with the graph model.

- Scalability Since it depends on vertical scalability, scalability in relational databases is the greatest difficulty it faces. Vertical scalability, on the other hand, is exceedingly expensive and unfeasible due to hardware limitations. The other type of scalability is horizontal (in which more commodity nodes or system units are added), but relational databases were not designed to support web applications that are distributed across multiple servers and serve millions of users, as is the case today, so it does not support horizontal scalability well. There could be an infinite number of nodes in a huge graph database. Scalability, as well as the capacity to enable use cases like compliance with data privacy requirements, is dependent on the ability to distribute the graph database across several servers.
- Big Data handling: Big data handling is a major issue in relational databases, and the solution has always been and will continue to be scalability and data distribution, which can take two forms: vertical or horizontal, in which data must be partitioned across multiple servers, which raises the issue of data joining complexity and performance. On the other hand, Graph analytics for big data is a framework for absorbing both structured and unstructured data from many sources, allowing analysts to investigate the data in an undirected manner as an alternative to the standard data warehouse strategy.
- Data warehouse: Relational databases are used for data warehousing, and, as is well known, the number of stored data grows over time, leading to the big data problem, which creates other issues such as performance degradation when doing OLAP and data mining or statistical processes.
- Complexity: The information does not fit in those tables. The database structure could be complicated, difficult, and slow to deal with if the data does not fit into those tables. The database's structure could be complicated, difficult, and time-consuming to deal with. Graph databases, on the other hand, can avoid some of the more sophisticated query complexities, such as foreign keys, nested searches, and join statements, in favor of direct or transitive relationships.
- Security: Although they face various security concerns such as SQL injection, Cross-Site Scripting, Root Kits, Weak communication protocols, and more, relational databases have implemented relatively safe techniques to provide security services. Many studies are being conducted today to investigate and solve these vulnerabilities. A graph database's data isn't the only thing that needs to be safeguarded. The structure of a graph's relationships and nodes is information in and of itself. This substantially simplifies the work of assigning permissions in identity and access control in a graph structure. The multi-database features of Neo4j 4.2 also make it easier for businesses to comply with privacy and security standards.

VI. EXPERIMENTS WITH RELATIONAL AND GRAPHS DBMS

Some data sets were tested in order to efficiently test the performance of both graph and relational database systems. Both in SQL and Neo4J, the tests reflect conducting various retrieval queries in order of difficulty. A Car dealership application is used on different datasets in which the tests were run and its structure, as seen in the illustrations below, both MS SQL Server and Neo4J database systems.

- - SQL Database System – SQL SERVER
- - Graph Database System – Neo4J Database
- Experiment 1:
  Query: Get all Cars containing "Benz"
  SQL Syntax
  SELECT DISTINCT cars FROM cars IN cars TABLE JOIN latestcars IN latestcars TABLE ON Carmodel EQUALS latestcars.Carmodel WHERE latestcars.Carmodel EQUALS "Benz"
  Neo4j Syntax
  MATCH (Cars)- [CONTAINS Model]-¿ (Model "Benz") return Distinct Cars

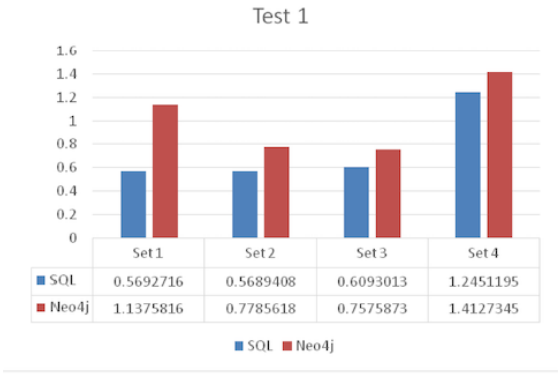


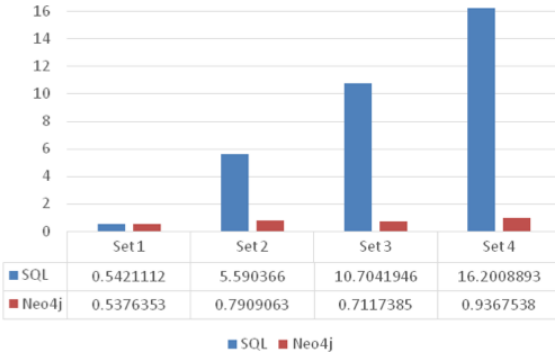
Figure 1. Query Execution Experiment 1[13]
This query involves the joining of two tables and a condition. It can be seen that for datasets, the query execution was faster in the MS SQL server than in Neo4J. The timing in seconds and milliseconds

- Experiment 2
  Query: Get all Cars similar to "Car Types" that have more than 100 likes containing "V8" from the "German" or from the "Hungary" Country
  SQL Syntax
  SELECT DISTINCT Cars FROM Cars IN Cars TABLE WHERE Cars.Likes BIGGER THAN 100 JOIN CarsSimilarity IN Cars.CarsSimilarity TABLE ON Carmodel EQUALS CarSimilarity.CarTwoId WHERE CarsSimilarity.SimilarityId EQUALS "Fast" AND CarsSimilarity.CarsOneId EQUALS "Cars Types" JOIN BenzCars IN BenzCars TABLE ON Carmodel EQUALS BenzCars.Carmodel WHERE BenzCars.CarId

EQUALS "V8" JOIN CarsCompany in CarsCompany TABLE ON Carmodel equals CarsCountry.Carmodel WHERE CarCountry.CountryId EQUALS "German" OR CarCountry.CountryId EQUALS "Hungary"

Neo4j Syntax

MATCH (Cars "Car Types")- [similarity: $IS_SIMILAR_TOSimilarity : "Fast"]- > (OtherCars), (OtherCars) - [CONTAINS_Benz]- > (Benz"V8"), (OtherCars) - [ISFROMCountry]- > (Country)whereCountryEQUALS"German"$



$Figure 2. Query Execution Experiment 2$[13]

Figure 2 shows the findings of experiment 2. The same conclusion can be drawn. The query execution time in Neo4J is substantially faster than in the relational system when there are many junctions between tables and multiple criteria, and it has very low value for all four datasets, the timing in seconds and milliseconds.

- Experiment 3:

Union and Intersection

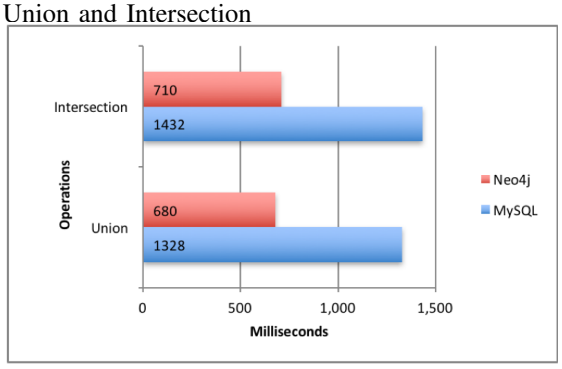


Figure 3. Union and Intersection[14]

When comparing the performance of the union and intersection operations, Neo4j outperforms MySQL marginally. According to query performance, the graph database Neo4j should be viewed as the better database system for storing and analyzing large-scale social graphs. Furthermore, when it comes to operations involving graphs, such as graph traversal, the graph database outperforms the relational database, especially when traversing the graph to a high degree. Meanwhile, Neo4j and MySQL have comparable speeds when utilizing built-in functions that are acceptable.

VII. Conclusion

In conclusion, while relational databases are powerful and reliable, there are times when a (slightly) better option for data storing exists. Graph databases, on the other hand, are an excellent solution for applications that require a huge number of data relationships. Social media programs, collaborative systems, libraries of any type, books, music, or movies are examples of such undertakings. New relationships can be introduced to graph databases without having to rewrite the schema, making them more adaptable than relational databases.

According to the experiment results, as the database size grows to a big extent, a relational database takes longer to insert or search data than a graph database. A graph database also performs significantly better when using built-in functions like union and intersection operations. When it comes to choosing which database to use, one must first conduct research, as described in the database phases, read, and, most importantly, test their applications ahead of time with huge quantities of records to forecast how they will function in the future. Users benefit from both types of databases, but the decision to utilize a graph database vs. a relational database comes down to how the database will be used. When looking at this from the future, it's interesting to contemplate how they behave in an application like this, where there are many relationships between products, as well as the suggesting engine and how it will react.

References

[1] Azhar Susanto, Meiryani. Database management systems. 2019
[2] Asuman Dogac. The Design of Relational Database . 2021
[3] Ivan Despot. Graph Database vs Relational Database. 2021
[4] Paul Saunderegger. What is a Relational Database (RDBMS)?. Databases. Oracle. 2022
[5] Margaret Rouse, Jack Vaughan. Database Normalization. Tech Target. 2019
[6] Chen, Yaowen. Comparison of Graph Databases and Relational Databases When Handling Large-Scale Social Data. 2016
[7] Donald.D. Chamberlin. Relational Database Management System. Computing Surveys. 1976
[8] Justin J. Miller . Graph Database Applications and Concepts with Neo4j. Graph Database Applications and Concepts. 2013.
[9] Naveen Singh. What are Graph databases and different types of Graph databases. 2019.
[10] Shalini Batra, Charu Tyagi. Comparative Analysis of Relational And Graph Databases. 2012
[11] Naveen Singh. RDBMS vs Graph Database. 2019
[12] E. F. Codd. A relational model of data for large shared data banks. 1970
[13] Liana Stanescu. A Comparison between a Relational and a Graph Database in the Context of a Recommendation System. 2021
[14] Chen, Yaowen. Comparison of Graph Databases and Relational Databases. 2016