Richard Lussier
Joshua Eklund
Joshua Engelkes

**Exploration of Graph Databases**

Data. In today's digital world, data has become an integral part of our lives and is one of the most important and valuable resources available today. Almost every interaction we have with technology, whether using our computers, browsing the internet, or buying groceries, produces some form of data. The ability to collect, process, analyze, and interpret data has become increasingly vital for governments, businesses, organizations, and individuals, as data analysis can help us gain insights, make informed decisions, and improve processes in almost every facet of our lives. Databases have become an essential tool for those who wish to analyze, as they provide a convenient way for storing, managing, updating, and retrieving data. There currently exist multiple different database designs that differ in how they store and represent data. In this essay, we present one such database paradigm: the Graph Database.

As stated in the introduction of this essay, there exist multiple database paradigms one can use for organizing, structuring, and manipulating data. There are relational databases (use relational tables to store data), key-value databases (where keys are unique and point to some value), document-oriented databases (each document serves as a container for key-value pairs), and many other database paradigms ("7 Database Paradigms"). Graph databases are one of these database paradigms. Specifically, graph databases are a NoSQL database that utilize graph theory to represent and store data. By NoSQL, we mean that rather than storing data as relational tables, graph databases instead store data in nodes and edges. Nodes represent entities, and edges represent the relationships between those entities. To understand why graph databases are used, we can examine the history of databases.

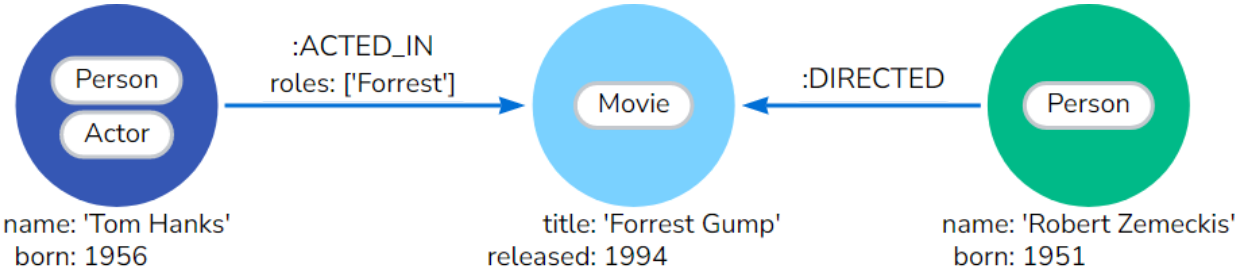
The history of databases begins in the 1960s with two early database models: the hierarchical model and the network model (CODASYL model). These two database models could be considered the grandfather of graph databases because they organize data in a tree-like structure ("A Timeline of Database History"). In the hierarchical model, data is stored as records, with each record having one parent but many children. However, it is inflexible as the one-to-many relationship cannot model complex structures ("Hierarchical Database"). The network model was similar to the hierarchical model, but instead of having only one parent, each child can have multiple parents, making it much more flexible than the hierarchical model. However, like the hierarchical model, this model also had problems, as it is difficult to implement and maintain ("Understanding the Network Database Model" ). Then, in the 1970s, Edgar F. Codd published his paper on a new database paradigm that would revolutionize the database industry: the relational database model.

The relational database model created by Edgar F. Codd dominated the database industry throughout the 70s, 80s, and 90s, becoming the predominant way to store and organize data. In a relational database model, data is stored as tables with rows and columns. Each row contains a

unique ID that allows for the establishment of relationships between tables (“What Is a Relational Database?”). The relational database model was more intuitive, easier to maintain, and easier to implement than any other previous database model. Along with the rise of the relational database model, SQL (structured query language) became the default language for creating, maintaining, and querying relational databases. However, the explosion of the internet and the sheer volume of data that was being generated led to some frustrations with relational and SQL-based databases. Relational databases at the time were difficult to scale horizontally, which was necessary for web-scale applications ("A Brief History of Databases"). This frustration with SQL databases led to the creation of NoSQL databases, which means that the database uses non-relational data structures (Smallcombe). NoSQL databases are easier to scale because they are designed to handle large volumes of unstructured or semi-structured data (“When To Use NoSQL Databases”). Although SQL databases were still the industry standard, NoSQL databases gained popularity throughout the 2000s. The rise of NoSQL databases resulted in the exploration of new database paradigms, such as the graph database.

Graph databases have become increasingly popular for their ability to store and process highly interconnected data efficiently. Graph databases are unmatched in terms of performance when working with connected data. Unlike relational databases, where the performance of join-intensive queries decreases as the dataset grows, queries in graph databases are localized to a portion of the graph. As such, the performance of the graph database remains unchanged with the expansion of the database. Graph databases are also highly agile and flexible due to their schema-less nature and the ease with which one can add new records. All of this makes graph databases particularly well-suited for social networks, recommendation engines, and fraud detection (“When To Use Graph Databases”). One specific graph database that has become popular is Neo4J. Created in 2007, Neo4J is used by multiple organizations such as eBay, Levi’s, JPMorgan, Volvo, and many others (“Who Uses Neo4j?”). To further our understanding of graph databases, we can delve further into Neo4J.

Since it is a graph database, Neo4J uses nodes and edges to establish records and the relationships between those records. In the case of Neo4J, nodes can have zero or more labels to classify what kind of nodes they are, relationships (edges) must always point in one direction, and relationships additionally must have one type to classify what kind of relationship they are. Nodes and relationships also have properties (key-value pairs) that provide additional information. These properties can hold different data types, such as strings and booleans, or even arrays of these types. To solidify this understanding, we can look at an example. The image below provides a simple example of a graph database created in Neo4J (“Graph Database Concepts”):




From the figure above, we can see that we have a database with three nodes: the node labeled Person/Actor, the node labeled Movie, and the node labeled Person. Each node has properties in the form of key-value pairs that provide additional information about the data associated with that node. For example, the node labeled "Person/Actor" has the properties "name: 'Tom Hanks', born: 1956". Additionally, the Tom Hanks node has a relationship with the Forrest Gump movie, with this relationship being that Tom Hanks acted in that movie in the role of Forrest. To create and query databases such as the example above, Neo4J uses Cypher, an SQL-inspired language. The general syntax of Cypher is as follows: nodes are denoted using parentheses, and relationships are denoted using "-[:Arrows]->". So, to create the database pictured above, the Cypher code would be the following ("Introduction to Cypher"):

```
CREATE (:Person:Actor {name: 'Tom Hanks', born: 1956})-[:ACTED_IN
{roles: ['Forrest']}]->(:Movie {title: 'Forrest
Gump'})<-[:DIRECTED]-(:Person {name: 'Robert Zemeckis', born: 1951})
```

The advantages of an appropriately used graph database exhibit the reasoning for their currently growing usage. Graph database performance in querying is potentially superior to any relational database, depending on the type of query, regardless of the size of the data (Dancuk). A graph contains an index-free adjacency property, where each node maintains only its neighbor vertices information, with no global index about node connections ("Graph Database Advantages"). The performance is constant because traversing a specific node's neighbors via edges is independent of the graph size unless more relationships are made to that specific node. This means that the graph exhibits a constant performance even as the data size grows into infinity. Unrelated data never needs to be loaded for a given query. Graph databases maintain very clear, explicit semantics for every query that can be written (Dancuk). A SQL database could contain hidden assumptions (forming cartesian products with different tables), whereas a graph query used appropriately will not. Graphical databases can perform real-time updates on big data while supporting queries ("Graph Database Advantages"). Because graph databases when used optimally will only touch relevant data, a sequential scan is not an optimization assumption. Structures used in graph databases are flexible while still maintaining ongoing queries. A relational database cannot easily adapt to this requirement, which is commonplace in the world of data management.

However, graph databases are not always the optimal choice. Graph databases work exceedingly well in cases where data is interconnected, but otherwise could be subpar. For example, one of the graph databases' biggest faults is their lack of usability when it comes to transactional databases (“What is a Graph Database and”). Transactional databases contain a lot of data, but very few relationships between the data points that a graph database could exploit. Transactional data takes up a large part of the usage of databases, so being obsolete in this field is hindering its popularity. Graph databases are also not very useful when working with data that merely contain key-value pairs (“Graph Database vs”). For example, when the sole purpose of the database is storing a user’s personal information that can be retrieved by an ID, then a graph database is not the optimal choice. Graph databases lose their quick search time when the queries span the entire database, especially when performing mass analytics queries across all relationships and nodes (Rund). There are a couple of community-based disadvantages of using a graph database as well. Since there is no standard query language, a user will need to decipher which they should use based on the platform they are using. This would mean if a user switched the graph database they are using, the syntax for querying could be entirely different, making it difficult for those without a programming language background to relearn a new querying language. Another disadvantage is that graph databases still have a small user base. This makes it difficult to participate in online discussion forums such as stack overflow to solve problems (Dancuk). Because of the potential downsides of using a graph database, it is important to know when a graph database is the most appropriate tool.

Fraud detection can be used as an example of how to deduce whether a graph database is an appropriate tool for a specific purpose. Fraud detection is used to protect enterprise accounts, information, and assets through analysis of activities by users and other defined entities (“Definition of Fraud Detection”). By uncovering patterns of behavior across multiple financial transactions, it is possible to detect anomalies indicating fraudulent activities (“Financial Fraud Detection”). The first step to detecting anomalies is building expert rules. These rules determine a baseline and general pattern for the system to search for. However, even through expert experience, it is difficult to cover all risks. Not all guidelines and risks are created equal (“Guide”). The second step is building a machine learning model to prevent and control the overall risk, and then using the expert rules to assist the model (“Financial Fraud Detection”). However, a system that uses a SQL database would run into a few problems. Since there are so many moving parts in financial transactions, there is a lack of verification and consistency among the entries in the dataset. The search time for finding samples of transactions through their entire process is difficult since a query for a SQL database would be required to search through each whole table to find potential anomalies. As well, the relations are not the primary dimension, so the secondary verification process is prolonged. Graphical databases would be a likely candidate for this type of problem. Visualized graphs could assist in recognizing fraud samples since they show relations between specific entities. Updating information would be easier and more relational to the rest of the transaction, providing more consistency in the transaction information. This would assist in the secondary verification process to ensure certain

transactions have less or more of a chance of being fraudulent (“Financial Fraud Detection”). Because of the specific purpose and implementation of fraud detection, the use of graph databases is appropriate.

With the unique way graph databases function, they’re applied in several different fields and applications. The three discussed here will be social media connections, suggestion and recommendation engines, and then some research that was conducted by a Morris faculty in 2016.

Social media is filled with great examples of graph databases. In a social media setting, you can imagine the nodes as people and the edges as the relationships between those people. An example of a directional relationship (incoming or outgoing) would be if person A follows person B on social media, but person B doesn’t follow person A back. An example of an undirected relationship (no head or tail) would be if person A follows person B and now person B also follows person A back. Instead of keeping two directional edges, you could condense the information into a single, undirected edge.

Another way graph databases are used is in suggestion/recommendation engines. Going back to our social media example, there are many (if not all) social media companies that implement a “you may also know” or “suggested contacts” feature. These features work by using *degrees of separation*, the number of nodes that separate one node from another. For example, if person A knows person B and person B knows person C, there would be 1 degree of separation between person A and person C. Suggestion engines use this concept to find people your followers follow but you don’t follow. The fewer degrees of separation, the more likely someone knows the other. Recommendation engines work in a similar way, however nodes can be more than just people. Imagine a scenario where you watch Netflix and you want movie recommendations. If you’ve watched and enjoyed movies A and B, and someone else has watched and enjoyed movies A, B, and C, then there’s a chance you might enjoy movie C as well. (Nawroth)

Going into some more technical applications, graph databases have been used in fields of research, including one of our Computer Science faculty here at Morris. Nic McPhee used graph databases in his 2016 research *Visualizing Genetic Programming Ancestries* (McPhee et al.). For the purposes of this paper, I won’t be going in depth on how evolutionary computation functions. However, I will give some info where needed. In this research, McPhee et al used the nodes of graph databases to store programs that evolved during evolutionary computation. These programs evolved similarly to biological evolution. The edges were directional, where the parent was outgoing to the child. These edges represented where a child came from during the evolutionary process. To organize the graph, you can imagine the early stages of evolution on the top and the subsequent generations are displayed below each other. The point of this research was to study how programs evolved during evolution and if there were any particular behaviors that helped evolution more. This research led to the development of newer methods of mutation and the discontinuance of using crossover in many pieces of evolutionary computation research.

In conclusion, graph databases are useful NoSQL databases that can be used in a variety of ways. Although graph databases have their pros and cons, they can be an extremely helpful tool in certain situations. With the versatile functionality of the edges and the fact nodes are unrestricted to their object value, there are many applications for graph databases, both in industry and in research.

Bibliography


“7 Database Paradigms.” *Tudip Digital*, 30 Dec. 2020, https://tudip.com/blog-post/7-database-paradigms/.

"A Brief History of Databases." *YouTube*, uploaded by CockroachDB, 12 Nov. 2019, www.youtube.com/watch?v=PA3LtpwfFwQ.

“A Timeline of Database History.” *Quickbase*, https://www.quickbase.com/articles/timeline-of-database-history. Accessed 30 Apr. 2023.

Dancuk, Milica. “What Is a Graph Database? {Definition, Use Cases & Benefits}.” *Knowledge Base by PhoenixNAP*, 1 Nov. 2022, https://phoenixnap.com/kb/graph-database.

“Definition of Fraud Detection - Gartner Information Technology Glossary.” *Gartner*, https://www.gartner.com/en/information-technology/glossary/fraud-detection.

“Financial Fraud Detection: One of the Best Practices of Knowledge Graph.” *Nebula Graph*, 12 July. 2022, https://www.nebula-graph.io/posts/financial-fraud-detection-one-of-the-best-practices-of-knowledge-graph.

“Graph Database Concepts - Getting Started.” *Neo4j* , https://neo4j.com/docs/getting-started/appendix/graphdb-concepts/. Accessed 30 Apr. 2023.

“Graph Database vs Relational Database.” *RSS*, https://memgraph.com/blog/graph-database-vs-relational-database.

“Guide to fraud detection rules & how to choose a solution.” SEON. (2022, November 24). Retrieved May 3, 2023, from https://seon.io/resources/guides/guide-to-fraud-detection-rules/#:~:text=A%20fraud%20detection%20rule%20is,%2C%20statistics%2C%20or%20logical%20comparison.

“Hierarchical Database.” *HEAVY.AI,* https://www.heavy.ai/technical-glossary/hierarchical-database. Accessed 30 Apr. 2023.

“Introduction to Cypher - Getting Started.” *Neo4j*, https://neo4j.com/docs/getting-started/cypher-intro/. Accessed 30 Apr. 2023.

McPhee, Nicholas Freitag, et al. ‘Visualizing Genetic Programming Ancestries’. *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion*, Association for Computing Machinery, 2016, pp. 1419–1426, https://doi.org10.1145/2908961.2931741. GECCO ’16 Companion.

Nawroth, Anders. “Social Networks in the Database: Using a Graph Database.” *Neo4j*, 15 Sept. 2009, https://neo4j.com/blog/social-networks-in-the-database-using-a-graph-database/.

Rund, Ben. “The Good, The Bad, and the Hype about Graph Databases for MDM.” *Transforming Data with Intelligence*, 14 Mar. 2017, https://tdwi.org/articles/2017/03/14/good-bad-and-hype-about-graph-databases-for-mdm.aspx.

Smallcombe, Mark. “SQL vs NoSQL: 5 Critical Differences.” *Integrate.Io*, https://www.integrate.io/blog/the-sql-vs-nosql-difference/. Accessed 30 Apr. 2023.

“Understanding the Network Database Model.” *MariaDB*, https://mariadb.com/kb/en/understanding-the-network-database-model/. Accessed 30 Apr. 2023.

“What Is a Relational Database?” *Oracle*, https://www.oracle.com/database/what-is-a-relational-database/. Accessed 30 Apr. 2023.

“When To Use Graph Databases: A Comprehensive Guide.” *Decipher Zone*, https://www.decipherzone.com/blog-detail/when-to-use-graph-database. Accessed 30 Apr. 2023.

“When To Use NoSQL Databases.” *MongoDB*, https://www.mongodb.com/nosql-explained/when-to-use-nosql. Accessed 30 Apr. 2023.

“Who Uses Neo4j?” *Neo4j*, https://neo4j.com/who-uses-neo4j/. Accessed 30 Apr. 2023.

Wu, Min. “What Is a Graph Database and What Are the Benefits of Graph Databases.” *NebulaGraph*, 18 Apr. 2023, https://www.nebula-graph.io/posts/what-is-a-graph-database.

Wu, Mingxi. “Graph Database Advantages: Using Graph Query Languages.” *TigerGraph*, 24 Jan. 2022,

https://www.tigergraph.com/blog/what-are-the-major-advantages-of-using-a-graph-database/.