Kyle Foss
Colin Rabe
12/17/2017

An Overview of Neo4j

Introduction

Neo4j is a graphical database developed by Neo4j Inc. It is an ACID compliant transactional database implemented in Java and Scala. It is used by companies like Ebay, Wal-Mart, Cisco, and many other big name organizations. In Neo4j, a user will have access to light U.I. that will allow them to perform queries in Neo4j's query language called Cypher. The U.I. is able to return a graphical representation of queries as connected nodes, tables, rows, and etc. So what is a graphical database? A graph database consists of vertices and edges. In Neo4j, vertices are known as nodes and are often called entities since a node can represent itself. Nodes contain labels and properties and can have relationships with other nodes. These relationships are represented by edges which can go in both directions of the nodes. Properties are represented by having a key and a value and are very similar to the properties of a JSON object, but instead properties represent the description of a node. To group nodes together, Neo4j uses labels to put nodes into sets which can be used similarly to simulate the behavior of table like in a RDMS. A good way to think about how data is represented in a graphical database is to think of an ER diagram since nodes are entities that have attributes which are properties, and relationships are like actions.

Neo4j Features

Neo4j includes many of the features one would expect from a relational database system, but also some that are unique to graphical databases. The most obvious feature that is absent from both relational and document oriented databases is the ability to visualize data and how it is interrelated by using a graphical model. However, data can also be displayed in tabular form without an accompanying graph. Neo4j has the ability to import and export data as CSV files, which can in turn be converted to and from other file types such as JSON and XLS (excel) which makes it more accessible [4]. Cypher supports many commands and clauses that users of SQL would be familiar with. Cypher statements can be combined with regular expressions which can allow commands to be more powerful and specific. Recent versions of Cypher include a query validator, which will check the syntax of a query for issues before it is run and alert a user to any problems.

Constraints, a prominent feature in relational database systems, are also possible to use in Neo4j. Multiple constraints can be applied to individual labels and relationships. One such common constraint can ensure that properties within nodes or relationships are unique. Another type can enforce that nodes or relationships must have a particular property, or they cannot be created. A third, called a node key constraint, can be made which requires that each node have a unique combination of property values and also contain the full set of properties to which the constraint was applied [4]. This can help add more structure to the database if it is necessary. Neo4j also supports user defined functions and procedures. These must be coded in java, converted into a jar file and imported into Cypher as a plugin to be used. There are also

several procedures that are built into Neo4j for administration purposes, such as examining metadata and changing a user's password.

Another cool and recent feature that Neo4j now supports is called pattern comprehension. Pattern comprehension within the context of Neo4j will examine a collection (maps, nodes or paths) and filter them through a predicate expression. Of the remaining values, another expression is mapped to them to generate a new collection to be returned. Local variables can also be used within pattern comprehension statements. The result is that more complicated actions can be performed with very compact and simple code [5].

Neo4j offers an optional schema through the use of its labels. In addition to organizing nodes into groups, labels allow indexes and constraints to be applied, which are what comprise the schema. Indexes can be based on a single property, or (recently) be based on multiple, making it a composite index. The indexes are used to help find more efficient starting points within the graph.

Similarities with MariaDB

There are a lot of differences as well as similarities in a sense that what a user can perform in MariaDB can usually be done in Neo4j just in a different way, so this part will focus on the main things the class encountered with MariaDB. The query language that Neo4j uses is called Cypher. Cypher is built from SQL in a syntactical way. Its structure for a match command is very similar to a select statement in SQL. A lot of the clauses that can be used with Cypher are almost the same as in SQL. A user is able to

utilize a where clause, order clause, set clause, limit clause, an or clause , an and clause, etc. [8] This is one of the nice features of Neo4j since users familiar with SQL are able to utilize the language a lot more effectively. Another similarity with SQL is the notion of functions. Like in mysql, Neo4j has built-in functions and allows you to create functions with Java and behave like functions in SQL. [10] Another related part of Neo4j are procedures. Procedures have the same functionality as they do in SQL. A user is able to use the same operations you can do in a procedure in MariaDB with Neo4j. They can take arguments, apply operations to the database, and return results. Another similar feature is indexes. Indexes have the same end goal of making a query faster like in SQL. [11] Another parallel the Neo4j and SQL are triggers. In Neo4j triggers are not built in, but there exist some plugins and libraries that will allow triggers to be made that will function the same as triggers in SQL. [3] Another similarity between Neo4j and mariaDB is the functionality of transactions. A transaction in Neo4j functions exactly the same as a transaction in mariadb. [14] A transaction in Neo4j has three parts to it. It can start the transaction, commit as well as roll it back if it fails. This gives Neo4j and MySql the capability to be ACID compliant.

Differences with MariaDB

Neo4j has many differences with MariaDB. One of the main differences between a relational database query language like MariaDB is how Cypher works. The goal of Cypher is to look for patterns in the data. For example, say that a node represents a person, and we want to find every friend this person has. We would use a match clause

to match the person node with all of the other person nodes that have a “Friend” relationship connecting them. Cypher will look for this pattern and return whatever it finds. [7] In MariaDB a user would have to utilize an inner left join with a foreign key to find that relationship which can lower efficiency. Another big difference between the two languages is the notion of how records, tables, and relationships are represented in SQL. Since Neo4j is schema optional; records in Neo4j utilize nodes to represent them. To keep structure like a table, Neo4j uses labels which lets users group nodes together. To link groups of nodes together, Neo4j utilizes relationships instead of foreign keys which leads to one of the advantages of Neo4j, joins. In MariaDB, users will often need to think about how to structure a join which can become very confusing when dealing with a vast amount of foreign keys. This can increase the amount of time to process a query, especially when using a filter on the results. In Neo4j the efficiency of a join in SQL can be much simpler. Instead of using a foreign key to join on, Neo4j will utilize previously defined relationships to navigate to your match. This makes the query a lot easier to read and understand and lowers the time to process an equivalent query in SQL. Another difference between the two languages are types. In Neo4j, types fall into three different broad categories: property types, structural types, and composite types. [8] Property types can be of type integer, float, string, or boolean. Structural types are nodes, relationships, and paths. Composite types are maps and lists. Structural types can only be returned from queries. Composite types can be used as parameters, Cypher literals, or be returned from queries. Another difference between Neo4j and MariaDB are how indexes work. In Neo4j you can place an index on properties. There

are two types of indexes, compound and single. Indexes can only be applied to labels with properties. One problem with using indexes in Neo4j is that certain clauses will cause them to not be used. [11] Neo4j doesn't currently support a lot of the clauses used in the match command with composite indexes, so a composite index will never be applied with something like a range clause in a match command. Another big difference between Neo4j and MariaDB is the ability to get statistics on queries. In Neo4j there is a clause called 'profile' that allows a user to see what operators in a query are being used. [12] Neo4j separates different parts of a query into operators in what is know as an execution plan. Every operator in Neo4j has its statistics that allow Neo4j to pick the effective way to perform the query. The main statistics used are: rows, estimated rows, and dbhits. Rows represent the amount of rows or nodes that the query is used on, and estimated rows are the number of rows that query uses with the cost based compiler. The dbhit statistic is an abstract measuring unit. Dbhits represent the amount of memory the query uses when retrieving or uploading data when using the database.

Similarities with MongoDB

MongoDB and Neo4j share the distinction of being NoSQL systems, so it is no surprise that they have numerous similarities. Both database platforms are open source and were released within two years of each other (Neo4j in 2007 and MongoDB in 2009). They are both the most popular database in industry today of their respective types, and they work on the same set of operating systems, which are Linux, OSX, Solaris, and Windows [3]. The key value pair form of organizing information is an

integral component to both systems. Although they are both advertised as being “schema-free”, through the use of indexes unique constraints can be enforced on both of their data units. They both allow data to be exported via CSV files. They both support a number of the same programming languages which can be used to make modifications to their internal processes. They also both support eventual and immediate consistency, which means stored data can be updated by write requests either simultaneously or with a delay.

Differences with MongoDB

Although both Neo4j and MongoDB have a lot of similarities since both are not relational databases, there are a lot of differences. The biggest difference is how data is represented. MongoDB is document based and Neo4j is a graph database. This can explain why MongoDB or Neo4j have different features from each other. MongoDB is able to support the following: sharding, mapreduce, master-slave replication, and In-memory capabilities. [3]

In contrast, Neo4j has the following features that are different from MongoDB: triggers, transactions (ACID), foreign keys, and causal clustering. [3] Triggers are absent in MongoDB as well as foreign keys. Instead of using a master-slave relationship for data replication, Neo4j uses casual clustering. For transactions, MongoDB isn’t completely ACID compliant as Neo4j is. Neo4j fully supports transactions like they are in MariaDB but MongoDB is only able to support atomic operations within a single

document. MongoDB is also only able to support data integrity after non-atomic operations. [3]

Advantages of Using Neo4j

One of the most important reasons to choose Neo4j over a relational database is that Neo4j does not require a schema. Being schema free means that it is possible for nodes to contain any kind of properties, with any type of values inside of them, even if the nodes have the same label. Likewise, relationships of the same kind are also not required to contain the same properties as each other. This makes Neo4j an appealing choice for users and companies who would benefit from or require more flexibility in how their data is handled, such as in social media applications or bioinformatics. Extra structure can always be added, however, through the use of constraints. Even though Neo4j is schema-free, it also retains its full ACID compliance which makes it very reliable because the data is more likely to be consistent and up to date. Neo4j is excellent for handling highly interrelated data, such as in many to many relationship cases. Because of its ability to create relationships between nodes of information, data that is related is effectively already joined together, which makes retrieving it in a query much more efficient. The efficiency of these join queries makes it so that creating indexes is not necessarily required, which allows other read and write operations to remain fast and for less storage space to be used. In addition, because of the relationship system, irrelevant data does not need to be examined during match queries which makes Neo4j even more efficient at retrieving specific or related information.

Disadvantages of using Neo4j

Every database technology has its drawbacks, and Neo4j is no exception. Neo4j does not support sharding, which is the practice of splitting up a database into smaller chunks across multiple servers. This means that in order to scale the size of a Neo4j system, it must be done vertically. In other words, the server Neo4j runs on must be outfitted with a better cpu, more ram, or more storage capacity, instead of being able to use multiple less expensive servers. From an industry perspective, choosing a graphical database could result in less support, both from a developer talent and customer support aspects because Neo4j is a newer technology than many other relational database systems. Neo4j is perhaps not the best choice for managing business transactions or logistical data. Companies that need to keep track of this kind of data could use another database system along with neo4j to manage this data, but it would result in a higher cost and require more personnel and maintenance. Data analysis using all data is cumbersome and limited, but can be expanded with libraries. Neo4j is also not well suited for conducting audit trails, which allows administrators to see changes to data, when they occurred, and who made the change. Related to this, there is no native versioning support, which means that Neo4j cannot show what the database looked like at a particular point in time without storing an old copy of itself. [3]

Libraries and Tools

One of the most popular libraries for Neo4j is called APOC (Awesome Procedures On Cypher). This is a library that can import helpful procedures, functions, and triggers to use inside of Neo4j. For example, it includes multiple date-time procedures which can extract specific components of dates and convert timestamps to different units. Another interesting procedure is called PageRank, which is the algorithm that google uses to find the most relevant websites to return from a search query. In a similar fashion, it is used in Neo4j to find the most important nodes related to a particular query based on the number of relationships a node has as well as by measuring the properties of relationships for arbitrary relevance. There is also a procedure for finding the soundex values of strings, which is useful for distinguishing between strings by how they are pronounced in english. These are just three examples from APOC, which contains over a hundred other possible operations.

There also exist a number of visualization libraries for Neo4j, but the most popular one is known as Gephi. Gephi allows a much more customizable visual representation of a Neo4j graph. This can be helpful for presenting models or findings from data in academic or industry research related contexts. Having a better ability to manipulate the visual representations of graphs can also potentially reveal further insights when using this tool for the purposes of data analysis. Another noteworthy library is called Spatial. As one might expect from the name, Spatial helps Neo4j import and query data that references distances between other points of data. Spatial is very applicable to the field of cartography and when representing the topology, or physical

layout, of an environment. Several queries that are supported are “Covered by”, “Intersect”, and “Within Distance”.

Neo4j Internals

Neo4j’s internal architecture is based on two general principles many graph database technologies share, which are called native graph processing and native graph storage. For a graphical database to have native graph processing, it must exhibit *index free adjacency*. This is simply the idea that there is no global index structure in the database that needs to be searched through in order to retrieve information. Instead, nodes themselves locally store the references to adjacent nodes. Native graph storage on the other hand means that the graph data is separated into different types of specialized store files. Each type of store file is responsible for a different aspect of the graph, e.g., there are different store files for nodes and relationships[1]. Records in these store files are of fixed size. Because of this, when given an id of a record, it can be looked up very quickly by simply multiplying the id by the number of fixed bytes which will result in the location of that record in the store file. Finding a record is therefore a O(1) operation. [2]

Nodes can be considered hubs that contain references to strands of information. The underlying data structure that helps make this possible is the linked list. Each node can contain a linked list of its properties and labels. In addition, the node also stores a pointer to its first relationship. A relationship is a more complicated data structure, but still essentially composed of linked lists. It is shared between two nodes because each

relationship stores a reference for the beginning node and the ending node. It also stores the references of the previous and next relationships of the beginning node, and also of the ending node, which are doubly linked lists because they can be traversed in either direction. Finally, a relationship stores a reference to a list of its properties, a reference to its type, and a way to identify if it is the first relationship in a list.

Data Model

For our data model we used a dataset from Neo4j's website authored by a user named soheli-jazayeri [6]. The dataset is about air travel and was published by the Bureau of Transportation Statistics. There are three types of nodes used in the model; they represent flights, airports, and tickets. The airport nodes are named with the abbreviation of the airport and have the label 'Airport'. Their abbreviation is also used as a value for their name property. The flight nodes have a name of 'f' with a number following that letter and the label 'flight'. The properties of flight nodes are date, duration, distance, and airline number. The ticket nodes are named as 't' followed by a number 1-3 and the name of the flight node. The properties of the the ticket nodes are class and price. In terms of relationships, flights have an origin and a destination airport and tickets are assigned to flights.

This model is capable of showing some interesting relationships within the data. We were able to experiment with making our own queries and also used some of the queries that were provided by the author. One of the queries we created uses pattern comprehension to return the distances of the flights of a particular airline to be

converted from miles to kilometers. The query works by first matching the list of all flights, filtering the flights by the airline, multiplying the distance property by a constant and then returning those new values. Another command we wrote was to add a label to all of the flights that happened on (the day before) Thanksgiving. The query worked by matching all of the flights that had a date property which contained 11/25/2015 using a regular expression. Then the label ‘Thanksgiving’ was added to the nodes with a set clause. With the new label, we made another query to grab all the ticket prices of these flights by matching the nodes that had the labels flight and Thanksgiving and returning the prices of all tickets in that set. Another neat relationship is to look at the amount of flights originating from an airport. The author on Neo4j’s dataset page made an interesting query to do this. It works by matching all airport nodes with origin relationship with flight nodes and aggregates the nodes and orders them by total number of flights, airport, and airline.

Overall, we found Neo4j to be very simple to set up and begin writing queries in. The query validator was usually a convenient feature, because (in our opinions) one of the most difficult parts of using query languages is being able to master the syntax. However, it became much less convenient when we were attempting to import a dataset through the console, in which case it would validate several thousand lines of code (which would take a few hours). In conclusion, we found Neo4j to be a very capable technology with a comprehensive set of features. Despite its radical differences from MariaDB, it still felt accessible to learn and use.

Resources

[1] Native vs. Non-Native Graph Technology (article), Joy Chao, July 18th 2016
https://neo4j.com/blog/native-vs-non-native-graph-technology/

[2] Graph Databases, Ian Robinson, Jim Webber & Emil Eifrem (see chapter 6), June 2015
https://pdfs.semanticscholar.org/f511/7084ca43e888fb3e17ab0f0e684cced0f8fd.pdf

[3] Website that Compares Database Technologies
https://db-engines.com/en/system/MongoDB;Neo4j

[4] Neo4j Developer Manual
http://neo4j.com/docs/developer-manual/current/

[5] Pattern Comprehension Section from Dev Manual
http://neo4j.com/docs/developer-manual/current/cypher/syntax/lists/#cypher-pattern-comprehension

[6] Flight data Model Source
https://neo4j.com/graphgist/flight-analyzer

[7] Cypher introduction
https://neo4j.com/docs/developer-manual/current/cypher/

[8] Cypher syntax
https://neo4j.com/docs/developer-manual/current/cypher/syntax/

[9] Query clauses
https://neo4j.com/docs/developer-manual/current/cypher/clauses/

[10] Query functions
https://neo4j.com/docs/developer-manual/current/cypher/functions/

[11] Schema
https://neo4j.com/docs/developer-manual/current/cypher/schema/

[12] Query tuning
https://neo4j.com/docs/developer-manual/current/cypher/query-tuning/

[13] Execution plans
https://neo4j.com/docs/developer-manual/current/cypher/execution-plans/

[14] Transactions
https://maxdemarzi.com/2015/03/25/triggers-in-neo4j/