International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 4 Issue 04, April-2015

Graph Database Application using Neo4j

(Railroad Planner Simulation)

Steve Ataky Tsham Mpinda, Luis Gustavo Maschietto
Departamento de Computação
Universidade Federal de São Carlos
São Carlos, Brazil

Patrick Andjasubu Bungama
Departamento de Computação
Universidade Federal do Paraná
Curitiba, Brazil

***Abstract*—Such as relational databases, most graphs databases are OLTP databases (online transaction processing) of generic use and can be used to produce a wide range of solutions. That said, they shine particularly when the solution depends, first, on our understanding of how things are connected. This is more common than one may think. And in many cases it is not only how things are connected but often one wants to know something about the different relationships in our field - their names, qualities, weight and so on. Briefly, connectivity is the key. The graphs are the best abstraction one has to model and query the connectivity; databases graphs in turn give developers and the data specialists the ability to apply this abstraction to their specific problems. For this purpose, in this paper one used this approach to simulate the route planner application, capable of querying connected data. Merely having keys and values is not enough; no more having data partially connected through joins semantically poor. We need both the connectivity and contextual richness to operate these solutions. The case study herein simulates a railway network railway stations connected with one another where each connection between two stations may have some properties. And one answers the question: how to find the optimized route (path) and know whether a station is reachable from one station or not and in which depth.**

Keywords—Graph database, Relational database, Railroad Planner, Simulation, Neo4j, Cypher

I. Introduction

A graph database is a database specifically dedicated to the storage type of graph data structures. It is therefore necessary to store only the data in the nodes and arcs. By definition, a basic graph is any storage system providing an adjacency between neighboring elements without indexation: any neighboring entity is directly accessible by a physical pointer. The types of graphs that can be stored vary, the undirected graph" single standard" to hyper-graph, including of course the property graphs. Such a database therefore meets generally the following criteria [1]: i) Optimized storage for data represented in a graph, with the option to store the nodes and arcs; ii) Optimized storage for reading and clickstream data in the graph (or Traversal), without using an index to browse relations; iii) Flexible data model for certain products: no need to explicitly create an entity for nodes or edges, unlike the rigid model tables in a relational database; iv) Integrated API to use some standard algorithms of graph theory (shortest path, Dijsktra, A *, calculating centrality ...).

A graph database is optimized for searching operator data locality, from one or more root nodes, rather than global searches.

II. CURRENT POSITION

The NoSQL movement reached its heyday in recent years, particularly as it seeks to address several issues that relational databases do not respond adequately:

- availability to handle very large volumes and Partitioning;
- flexibility scheme;
- difficulty to represent and process complex structures such as trees, graphs, or relationships in large numbers (In the databases ecosystem, graphs bases are often positioned mainly in the last two points:);
- process highly connected data;
- easily manage a complex and flexible model;
- deliver outstanding performance for local readings, for graph traversal;

III. GRAPH DATABASE, RELATIONAL DATABASE AND OTHER NOSQL

The term ‘relational hubs’ is coined in [3] to describe the differences. According to Alistair we are at a crossroads. One of the paths leads to the approach adopted by most NoSQL databases, in which the data are highly deenormalized and we rely on the application to gather with typically high latency and understanding. The other path leads to the approach adopted by the graphs databases in which we use the expressive power of the graph to build a flexible model and connected to the problem in question we then request with low latency to better understand.

Relational databases are included. As the graphs Databases, relational databases have a model centered on the queries. But this model is not as powerful as bases graphs. In particular, it does not create on the fly arbitrarily large structures, semantically rich and connected. To create any broad structure with a relational database we need to plan our knuckles in advance. To authorize changes, you end up creating a lot of columns that can be zero. Result: Tables ”dotted” fanciful joins (expensive), object-relational impedance problems even with simple applications.

Furthermore, a graph database is adapted to the use of graph-type data structures like trees or derived, especially if it is to exploit the relationships between data. The case of perfect use of a search is to start with one or more nodes and browse the graph. It is always possible to make myEntity.findAll type of readings (”find all the entities of a kind”), but in this case it is necessary to use an indexing system, which can be internal as appropriate to the graph (super-nodes for indexing) or above the graph (via Apache Lucene for example).

Conversely, relational databases are well suited to findAll queries through the internal structures of tables, especially if it is to perform aggregations of operations on all the rows in a table.

Despite their names, they are, however, less effective
on the holding relationships, which must be optimized by the index creation, including foreign key. As mentioned previously, a graph database offers the ability to browse by physical pointers relations where foreign keys offer only logical pointer.

IV. SOME GRAPH DATABASE APPLICATION

Social networks modeling have obviously in recent years become one of the most visible when using the graph databases. LinkedIn comes easily to display the degree of separation between each contact, which is ultimately only the distance between nodes in the graph representing people and their relationships. Although very interesting, this problem is not very common because of the small number of actors in this market.

Use cases that should reveal most common are [2]: i) modeling a set of knowledge about people, and a market-sector organizations or more generally ecosystem; ii) the specific business data representation such as cinema (films, actors, directors, and so on), publishing (books, authors, publisher, and so on) or the description of all the parts of an industrial machine how they are interconnected; iii) In any case, such a database will be conveniently integrated into a heterogeneous environment Persistence (thus speaks of ”polyglot persistence”) that would address the various problems the best solution, etc.

V. NEO4J

Neo4j is a graph database in Java designed to be embedded in an application or accessed in client/server via a REST API. The graph manipulation in Java Neo4j is very natural with its API: Node and Relationship are the major classes used to model a graph while adding a set of properties for each node and relationship.

Imagine a social networking application such as Facebook, Linkedin and Viadeo in which the user can bind with friends. This user wants to know what friends he has in common with other friends. With a graph, he could easily see the relationship. Here is a basic example (Figure 1):

The implementation of the below scheme in a relational database is not easy. There are several ways to do so, as the pattern Querie to make resolutions, but it is complicated to use. To solve these problems several type of graph databases exist, including the famous base Neo4j and HyperGraphDB and InfoGrid.

In a graph we can store two types of information: nodes and links, or , in other words, nodes and edges. Each node can have multiple links that point to other nodes. It is through this that relationships can be between nodes. They will thus allow us to organize them. In addition, each node can have multiple properties or attributes for stoker as key / value our data.

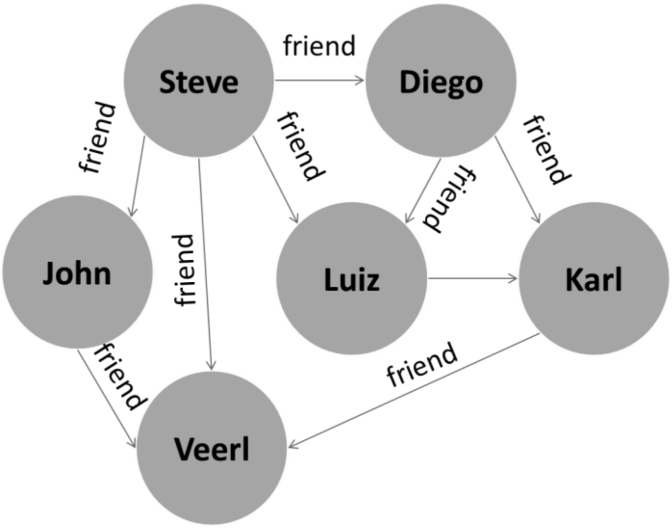


Figure 1: example of social network relationship

In brief: A graph store data in nodes that have properties. The nodes are organized by relations that have they same properties. A traversal allows navigating in the graph from a node and identifies roads or paths with as nodes ordered according options. An index is mapped by the properties of nodes and relationships.

And where is Neo4j? The database used to manage all types of objects, nodes, relationships and indexes. And through the algorithms, internal and external tools such as Apache modules Lucuene, Cypher or Gremlin recovery of our data is easier.

Developing applications on Neo4j is a breeze. These language guides help you connect to Neo4j from your preferred programming language. In this work we used Java language.

VI. CYPHER

Neo4j is generating much interest among NoSQL database users for its features, performance and scalability, and robustness. The software also provides users with a very natural and expressive graph model and ACID transactions with rollbacks [4]. However, utilizing Neo4j in a real-world project can be difficult compared to a traditional relational database. Cypher fills this gap with SQL, providing a declarative syntax and the expressiveness of pattern matching. This relatively simple but powerful language allows you to focus on your domain instead of getting lost in database access. With cypher, very complicated database queries can easily be expressed through.

VII. CASE STUDY

In our case study (figure2) we are simulating a railway network railway stations connected with one another and each connection between two stations has a property: the distance. The question is: how one can find the optimized path and know whether a station is reachable from one station or not. To do so, we need to use two approaches:

1. Breadth-first (Figure 3) search (BFS), each algorithm that begins at the root node and explores all the neighboring nodes. Then for each of those nearest nodes, it explores their unexplored neighbor nodes, and so on, until it finds the goal. In Neo4j-Cypher, one is traversing the nodes given some additional criteria only relations with a property visibility=public should be traversed, aiming at filtering nodes;
2. In addition of BFS, one is using Dijkstra algorithm (Figure 4) to calculate the shortest route from one station to another. It picks the unvisited vertex with the lowest distance, calculates the distance through it to each unvisited neighbor, and updates the neighbor's distance if smaller. Mark visited (set to red) when done with neighbors.

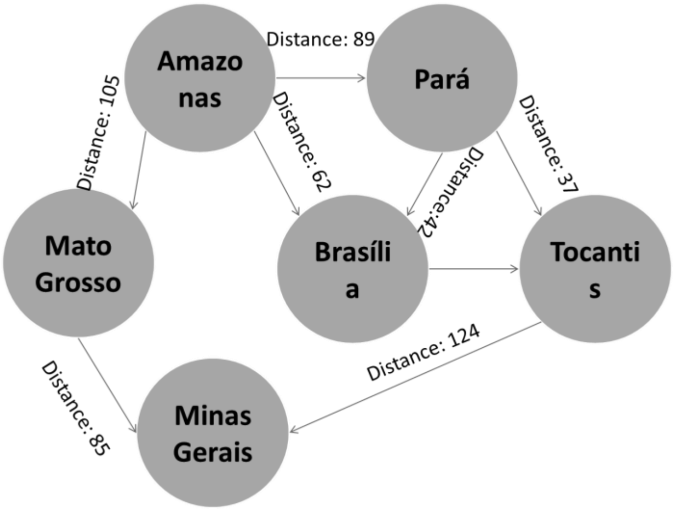
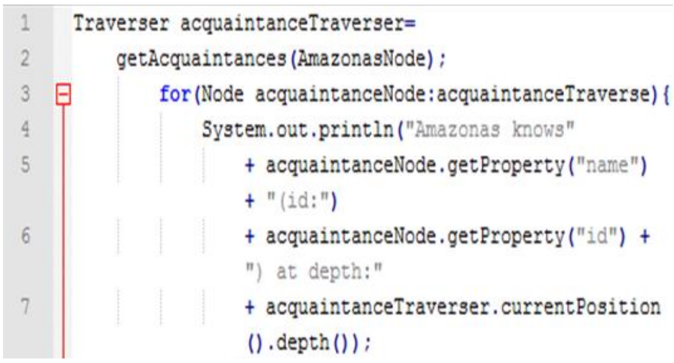
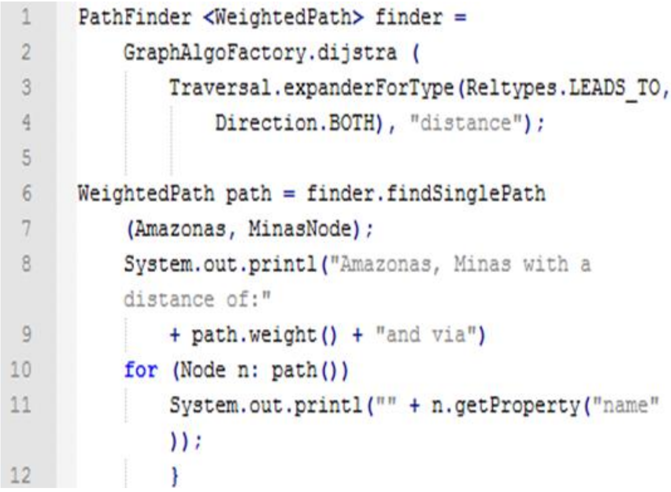


Figure 2: Railroad Simulation



```
Traverser acquaintanceTraverser=
    getAcquaintances(AmazonasNode);
        for(Node acquaintanceNode:acquaintanceTraverse){
            System.out.println("Amazonas knows"
                + acquaintanceNode.getProperty("name")
                + "(id:")
                + acquaintanceNode.getProperty("id") +
                ") at depth:"
                + acquaintanceTraverser.currentPosition
                ().depth());
```

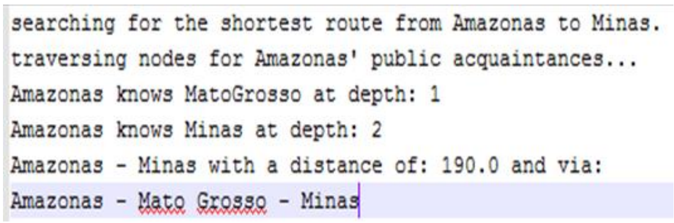
Figure 3: DFS in Cypher



```
PathFinder <WeightedPath> finder =
    GraphAlgoFactory.dijstra (
        Traversal.expanderForType(Reltypes.LEADS_TO,
            Direction.BOTH), "distance");

WeightedPath path = finder.findSinglePath
    (Amazonas, MinasNode);
    System.out.printl("Amazonas, Minas with a
    distance of:"
        + path.weight() + "and via")
    for (Node n: path())
        System.out.printl("" + n.getProperty("name"
        ));
        }
```

Figure 4: Dijkstra's algorithm in Cyphe for finding the shortest path between Amazonas and Minas considering both directions.



```
searching for the shortest route from Amazonas to Minas.
traversing nodes for Amazonas' public acquaintances...
Amazonas knows MatoGrosso at depth: 1
Amazonas knows Minas at depth: 2
Amazonas - Minas with a distance of: 190.0 and via:
Amazonas - Mato Grosso - Minas
```

Figure 5: Running the code above (Figure 3 and 4), we will get such cypher output, that shows the shortest path between Amazonas a Minas and the cost (depth and distance)

Neo4j comes with a number of built-in graph algorithms. They are performed from a start node. The traversal is controlled by the URI and the body sent with the request. These are the parameters that can be used:

algorithm

The algorithm to choose. If not set, default is shortestPath. algorithm can have one of these values:

- shortestPath : the shortestPath algorithm can find multiple paths between the same nodes. If no path algorithm is specified, a shortestPath algorithm with a max depth of 1 will be chosen.
- allSimplePaths
- allPaths
- dijkstra (optionally with cost_property and default_cost parameters)

max_depth

The maximum depth as an integer for the algorithms like shortestPath, where applicable. Default is 1.

VIII. EXPERIMENT AND RESULTS

The projects implementation using Java language with Neo4j database access can occur in two different ways: using JDBC connection driver (Java Database Connectivity) or with the use of connection class belonging to Neo4j libraries imported with build dependencies in MAVEN project specified in the pom.xml file.

For connection to Neo4j with JDBC one must import the JDBC connection driver for the project and start the Neo4j service on the local machine on the default port 7474 (http: // localhost: 7474 /) if the project is under local basis. Thus, requests for access to the database management system occur by using already established connection on the local machine, i.e., this process avoids the opening and closing the connection each request performed by the developed system.

Using the JDBC driver, it is possible to use Cypher, the specific language for data manipulation in Neo4j environment and return results using Resultset. The process occurs using established connection for both queries (MATCH) and inserts (CREATE) in the database. This process denotes a shorter duration compared to the process using Embedded Database method of GraphDatabaseFactory class.

The use of the method GraphDatabaseFactory of the Embedded Database class creates a file if it does not exist, which will store the data managed by Neo4j and opens the connection. When the database already exists the connection is established in all required process. In such cases it is always necessary to initiate a connection and at the end of the process to close it. This task causes an excessive consumption of time compared to using JDBC connection driver.

Importantly, for the use of the method Embedded-Database Neo4j service should not be started out of the application, therefore, the application tries to establish a connection that is already established, this attempt will result in a blocked connection error.

Although it seems a good option to use JDBC connection driver due to the gain in response time, some methods belonging to Neo4J library may not be used, in the case of searches for depth and using dijkstra algorithms. This fact is because the dijkstra algorithm, for example, requires the identification of an initial node to check nodes related and as the search for JDBC returns the data nodes as String and not as Node object, it would be impossible to identify and begin the search. Now with the use of Neo4j libraries one can open the connection and perform a search using the ExecutionResult class that will receive the return of the data as a list of Nodes objects.

IX. CONCLUSION AND FUTURE WORKS

The resulting model and associated queries are simply projections of questions you want to ask about your data. With Cypher, language query Neo4j, the complementarity of these projections becomes apparent: the paths used to create the structure of the graph are the same as those used for querying.

One noted that this application, even though it be possible to be implemented with relational databases, the performance would not be as good as the graph one. Moreover, it has been observed the when processing many concurrent transactions, the nature of the graph data structures helps distribute the transactional cost through the graph. Usually as the graph grows, the transactional conflict disappears. In other words, the higher the graph, the larger the flow rate is important, which is an interesting result.

As future works, it is being developed and implemented some strategies capable of replacing some meta-heuristic for railway and railroad optimization since both of scenarios can be represented as graphs. Moreover, some prediction techniques, such as estimate the planning that better fit for some period of time based on similar past planning or datas are being implemented as well.

REFERENCES

(1) Domenjoud, M. (2012). Bases de donnes graphes : un tour dhorizon. France.

(2) Figuiere, M. (2010). cember 5th, 2014 MICHAL FIGUIRE, NoSQL Europe : Bases de donnes graphe et Neo4j. http://blog.xebia.fr/2010/05/03/nosql-europebases dedonnees-graphe-et-neo4j/, accessed on December 6th, 2014.

(3) Jones,A.(2012).http://www.infoq.com/fr/articles/graphdatabases-bookreview, accessed on December 5th, 2014.

(4) PANZARINO, O. (2014). Learning Cypher, Onofrio Panzarino. Packt, 1st edition.