Section A

Summary of Data Set

This dataset is about the information about DNR Camping Parks Reservation Data in 2016. The table contains valid and meaningful information about every park with meaningful columns. The source dataset contains the park data in Canada along with few countries like the USA and Germany. The key attributes like party size, rate type, booking type, etc. give the trends of reservations. The data analysis can be done with the help of attributes like starting date and ending dates which results in the peak time of every year and help new parks to get established based on the demand of reservation. For example, if parks a, b, c belongs to the same area with the highest reservation demand, there might be scope for the establishment of new parks to serve the demand of people. After further analyzing the data set, it was noted that more than half of the parks in Canada are from Nova Scotia and this shows the hiking trend of tourism in nova scotia. The reservation data also shows that the preference of people was most likely in booking a single tent whereas the other equipment like was less in demand. Rissers Beach has the highest number of parks. On the other side, Graves island, Caribou Munroes and the Mira river have nearly the same number of parks. The conclusion is that this dataset has enough information to analyze every park and its equipment.

Section B

- A java program has been written and attached in the submission folder which gives three files as output with names file1, file2, and file3.
- Every file was refined and cleaned based on the given requirements.
- Using File3.csv, all the details of parks in Nova Scotia are filtered which can be used while loading data in the Neo4j database.

Section C

Neo4j Installation

The graph database neo4j has been installed in my windows laptop as a desktop application which can also be accessed from the web using local port 7474. However, I accessed Neo4j through the desktop version.

Section D

Graph Formation in Neo4j

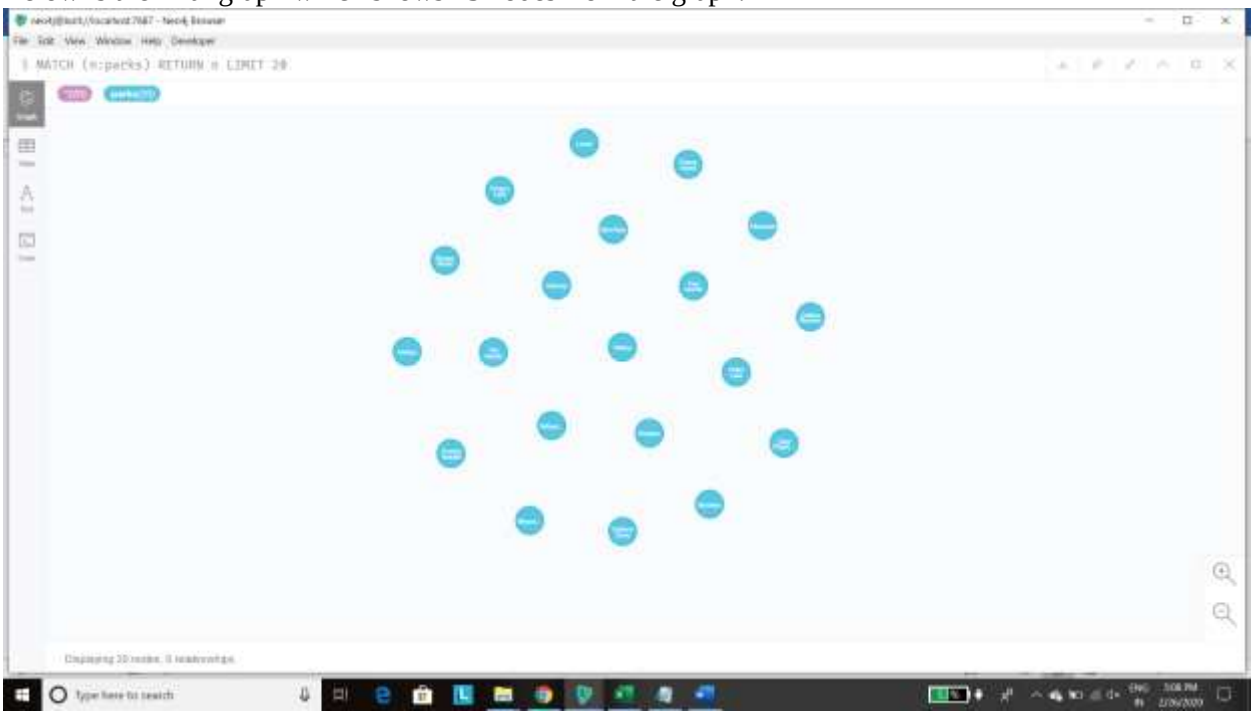
- Every park has been considered as one node in Neo4j and loaded into the database using a cypher query which is written in the below line. I have attached file4.csv which is used as a source to load all parks as nodes.
- LOAD CSV WITH HEADERS FROM "file:///file4.csv" AS row
  CREATE (p:parks {ParkName : row.ParkName, PartySize : toInteger(row.partySize),RateType : row.RateType, BookingType : row.BookingType, Equipment : row.Equipment})
- Above cypher query loads the local stored CSV file which contains all the data of parks in Nova Scotia and treats every row as a node in the graph. Every header of CSV file has been loaded with proper name.
- The relationships have been established between the nodes with given constraints. The nodes with the same rate type have been related to NeighboutByRate relation and the nodes with identical equipment have been related to NeighBourByEquipment relation. Below queries are written to establish the relationship between all nodes in the graph.

- MATCH (a:parks), (b:parks)
  WHERE NOT a=b AND a.RateType =b.RateType
  CREATE UNIQUE (a)-[r:NeghbourByRate]->(b)
- MATCH (a:parks), (b:parks)
  WHERE NOT a=b AND a.Equipment =b.Equipment
  CREATE UNIQUE (a)-[r:NeghbourByEquipment]->(b)

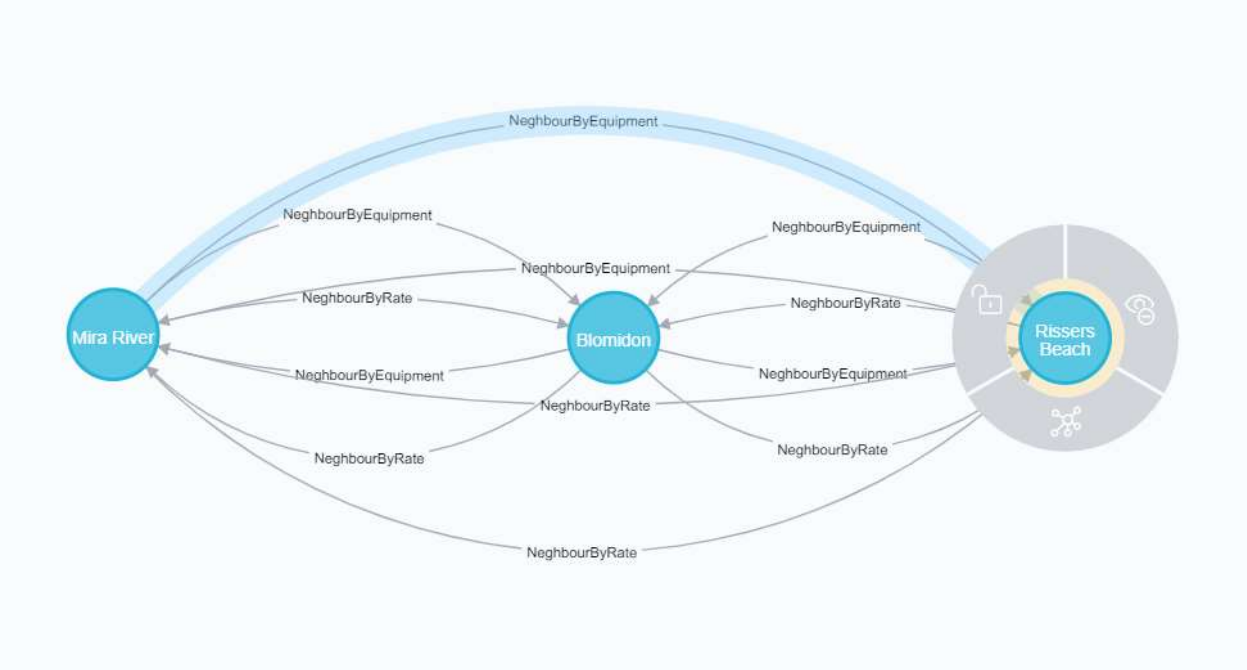
- After establishing the nodes and their relations, a cypher query has been written which displays the parks with higher size. Below is the cypher query for that.
- match (n:parks) with max(n.PartySize) as max
  match (x:parks)
  where x.PartySize = max
  return x

Screenshots of the nodes in Neo4j

- Below is the final graph which shows 25 nodes from the graph.



- The below screenshot shows the nodes (limited to three to shows the proper view of relations) which are related by relation NeighbourByRate and NeighbourByEquiment. I have attached full screenshot with all nodes in the document.

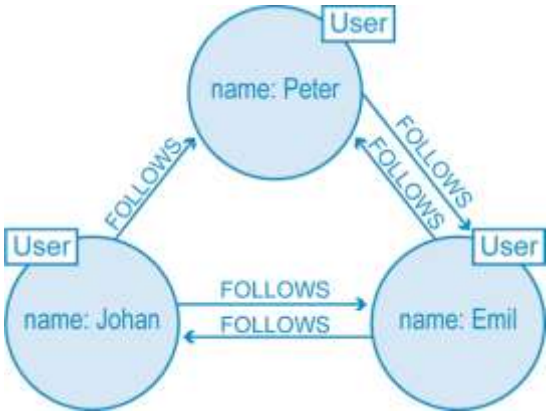


- The below screenshot is the picture of node which shows the park details with the highest number of party size.

Section E

Technical Report on Neo4J

Neo4j is a type of graph database that uses graph structure with nodes as the data and the edges as relationship between them. The graph relates the data objects in the store to the set of nodes and edges, the edges reflecting the relationship between the nodes.[1] Relationships enable the data in the store to be directly linked together and, in many cases, to be retrieved with one operation. Graph databases keep the relationship between the data using which the operations can be performed.[1]



This connection-first approach to data means that relationships and interactions are established through every phase of the data lifecycle: from the definition, logic model design, physical model implementation, query-language process, and consistency within a scalable, robust database system. [5]

Neo4j is implemented in Java and can be accessed from software written in other languages using the Cypher query language through a transactional HTTP endpoint or through a binary "bolt" protocol. It is implemented in Java and can be accessed from software written in other languages using the Cypher query language through a transactional HTTP endpoint or through a binary "bolt" protocol.[2]

Transaction processing

Graph databases are often optimized and focused on one or more of these uses. In particular, the first two uses are focused on the processing of transactions. When dealing with many concurrent transactions, the nature of the graph data structure helps the transaction overhead spread across the graph. As the graph expands, transactional disputes typically fall away, i.e. expanding the graph tends to increase the performance of the graph [8]. But not all the graphics databases are completely ACID. However, a variant based on the BASE properties often considered in the context of NoSQL databases is not suitable for graphs. In general, the distribution of graph processing involves the implementation of suitable partitioning and replication techniques to optimize the processing, i.e. to reduce the need to move data between different network nodes. For example, Neo4j uses a master-slave duplication, i.e. one computer is named a master and other a slave [8].

In Neo4j, all writings guided to any machine are passed through the master, which in turn ships the updates to the slaves when polled. If the master fails, the cluster must automatically choose a new master.

Neo4j includes a quorum to support the write load [8]. It ensures that most of the servers in the cluster need to be online for the cluster to accept write operations. Otherwise, the cluster would degrade to read-only operation until the quorum can be formed.

Significance

Many systems today manage data that is highly associative, i.e. organized as graphics (networks). The most famous example of this is social networking sites, but even labeling systems, content management systems and wikis deal with inherently hierarchical or graph-shaped data [3]. This appears to be a concern because it is difficult to deal with recursive data structures in conventional relational databases. Essentially, each traversal along a connection in a graph is a junction and the junctions are known to be very costly.[3] In addition, for user-driven content, it is difficult to pre-conceive the exact structure of the data that will be treated. Sadly, the relational model requires upfront schemas and makes it difficult to suit more complex and ad-hoc data.[3] So the relational database cannot accommodate complex relationships. Graphic structures are opaque, unmaintainable and rigid.

Limitations

Even though the graph database comes with an alternative solution with concepts like indexing, still there is a need for a better solution. Graph databases are not as useful for operational use cases because they are not efficient at processing high volumes of transactions and are not good at handling queries spanning the entire database. Because they are not designed for storing and retrieving business entities such as clients or vendors, you would need to combine a graph database with a relational or NoSQL database.[4]

Why Neo4J is better Suit?

A graph database stores the data but is also able to establish links between the things. For example, if we consider the Relational database model to analyze this data set, there is no primary key to identify every park in the data set and it needs to be dissolved into many tables by performing the normalization process. The easy alternative can be given by the graph database system where every park can be linked with the given relation. For instance, two parks with the same rate type are linked and the same park can be linked to another by relation of the same equipment type. There won't be a concept of joins to understand the property of every data row. The relationships in the data can be viewed without making a hypothesis on each tuple.[6]

Summary

Before starting a new project, there is often a temptation to use technologies that are well known or that are recent and well discussed. Consideration should be taken to see if they are really the best fit for what you need, however, or if something else might work better for you. Relational Database is often considered a safe choice, and there are several NoSQL database solutions that get a lot of online discussions these days that might be tempting to use. But if you want the best of both - the simplicity and speed of iteration that is typical in NoSQL databases, combined with the relational modeling ability of the Relational Database-then you should consider looking at the Graph Database instead, and see what it can do for you.

References

[1]"Graph database", *En.wikipedia.org*, 2020. [Online]. Available: https://en.wikipedia.org/wiki/Graph_database. [Accessed: 23- Feb- 2020].

[2]"Bolt Protocol", *Boltprotocol.org*, 2020. [Online]. Available: https://boltprotocol.org/. [Accessed: 23- Feb- 2020].

[3] E. Eifrem, "Neo4j - The Benefits of Graph Databases: O'Reilly Open Source Convention: OSCON, July 20 - 24, 2009 in San Jose, CA", *Conferences.oreilly.com*, 2020. [Online]. Available: https://conferences.oreilly.com/oscon/oscon2009/public/schedule/detail/8364. [Accessed: 23- Feb- 2020].

[4]"The Good, The Bad, and the Hype about Graph Databases for MDM | Transforming Data with Intelligence", *Transforming Data with Intelligence*, 2020. [Online]. Available: https://tdwi.org/articles/2017/03/14/good-bad-and-hype-about-graph-databases-for-mdm.aspx. [Accessed: 23- Feb- 2020].

[5]"Graph Databases for Beginners: Why Graph Technology Is the Future", *Neo4j Graph Database Platform*, 2020. [Online]. Available: https://neo4j.com/blog/why-graph-databases-are-the-future/. [Accessed: 23- Feb- 2020].

[6]"Graph Databases. What's the Big Deal?", *Medium*, 2020. [Online]. Available: https://towardsdatascience.com/graph-databases-whats-the-big-deal-ec310b1bc0ed. [Accessed: 23- Feb- 2020].

[7]"Introduction to Graph Databases", *Compose Articles*, 2020. [Online]. Available: https://www.compose.com/articles/introduction-to-graph-databases/. [Accessed: 23- Feb- 2020].

[8] *Hal.inria.fr*, 2020. [Online]. Available: https://hal.inria.fr/hal-01444505/document. [Accessed: 23- Feb- 2020].