Otto-von-Guericke-University Magdeburg

Faculty for Computer Science

CREATION OF CONVERTER TOOL FOR NEO4J

Documentation report

Vishwanath Valluri (217197)	venkata.valluri@st.ovgu.de
Harika Reddy Chandamollu (220385)	harika.chandamollu@st.ovgu.de
Gopi Krishna Burranagari (210576)	gopi.burranagari@st.ovgu.de

Supervisor: M.Sc. Marcus Pinnecke marcus.pinnecke@ovgu.de

Abstract

We have many applications which can model the data easily. The RDBMS and Graph database all come under database management software systems but its building blocks are different from each other. RDBMS comprises of tables where as Graph Database has vertices and edges. Graph database has not occurred to the place where RDBMS currently is in, it requires some more time to recognize the benefits over RDBMS like explosive volume of data, visualisation of huge data, response time, predictive analysis (Graph theory, Depth and Breadth first search algorithm, Path finding with Dijkstra algorithm), its high performance, Flexibility and many more. Some of the application areas of Graph databases Data analytics, Security, Supply chain management and the available software are TigerGraph, Neo4j, DataStax. A Graph database can change a highly complex data into understanding the insights of each and every particular layer. In this project, we have considered four real time traditional datasets and created a converter tool for some sets of data which can be directly run into Neo4j.

CONTENTS

1. INTRODUCTION

1.1 Graph database:

A graph database is a storage engine that is specialized in storing and retrieving vast networks of data. It efficiently stores nodes and relationships and allows high performance traversal of those structures. Properties can be added to nodes and relationships.

Graph databases are well suited for storing most kinds of domain models. In almost all domains, there are certain things connected to other things. In most other modelling approaches, the relationships between things are reduced to a single link without identity and attributes. Graph databases allow keeping the rich relationships that originate from the domain, equally well-represented in the database without resorting to also modelling the relationships as "things". There is very little "impedance mismatch" when putting real-life domains into a graph database [1].

1.2 Neo4j:

Neo4j is a NOSQL graph database. It is a fully transactional database (ACID) that stores data structured as graphs. A graph consists of nodes, connected by relationships. Inspired by the structure of the human mind, it allows for high query performance on complex data, while remaining intuitive and simple for the developer [1].

- Neo4j has an intuitive, rich graph-oriented model for data representation. Instead of tables, rows, and columns, you work with a graph consisting of nodes, relationships, and properties.
- It has a disk-based, native storage manager optimized for storing graph structures with maximum performance and scalability.
- It is scalable.
- Neo4j can handle graphs with many billions of nodes/relationships/properties on a single machine, but can also be scaled out across multiple machines for high availability.
- It has a powerful traversal framework and query languages for traversing the graph.
- It can be deployed as a standalone server or an embedded database with a very small distribution footprint.
- has a core Java API

In addition, Neo4j has ACID transactions, durable persistence, concurrency control, transaction recovery, high availability, and more.

Cypher is a declarative graph query language that allows for expressive and efficient querying and updating of the graph store. Cypher is a relatively simple but still very powerful language.

Very complicated database queries can easily be expressed through Cypher. This allows you to focus on your domain instead of getting lost in database access.

1.3 Basics of Neo4j:

CREATE (node1) , (node2)

CREATE (node : label1 : label2:…….labeln)

CREATE (node : label1 { key1 : value, key2 : value, …….})

Example:

CREATE (Sharma : player{name: "Rohit Sharma", YOB: 1988, POB: "Hyderabad"})

Returning the created node

CREATE (Node : Label { properties…..}) RETURN node

Creating relationships

CREATE (node1) – [:RelationshipType] → (node2)

CREATE (Sharma : Player {name : "Rohit Sharma", DOY : 1988, DOP : "Hyderabad" })

CREATE (Ind : Country { name : "India" })

Now create a relationship named BATSMAN_OF between these two nodes as

CREATE (Sharma) – [r : BATSMAN_OF] → (Ind)

2. DATASETS

We are going to deal with four datasets and discuss each of them in detail in further sections.

- TPC-H
- Google Web Graph
- Microsoft Academic Graph and
- Yahoo financial dataset

3. TPC-H BENCHMARK

TPC-H is ad-hoc decision support benchmark. This benchmark consists of a suite of business oriented ad-hoc queries and concurrent data modifications. The queries and the data populating the database have been chosen to have broad industry-wide relevance.

TPC-H benchmark is often used a method for customers to evaluate data warehouse products to make purchase decisions.

Decision support benchmark system (TPC-H) supports business and organizational decision-making activities.

Such activities can include:

- Comparison of sales figures between one week and the next
- Predicting revenue figures based on new product sales assumptions
- Evaluating the consequences of different decision alternatives from the given past experience [3].

TPC-H benchmark database consists of a schema consisting of 8 tables. The benchmark can be run using pre-determined database sizes, referred to as "scale factors". Each scale factor corresponds to the raw data size of the data warehouse.

In TPC-H benchmark, business environment is divided into two areas

- A business operation area
- A decision support area [2]

TPC-H dataset information

Domain: Retail
Size: 2GB
No of tables: 8
No of rows: 6,885,051
No of columns: 61
Data type: Real

***Create statements code snippet
```
	static String createStatementForNation(Nation_Model nation_Model){
		return
"CREATE (neo4j:nation {n_nationkey:" + nation_Model.getNNationkey() + "})\n"+
"CREATE (n_name:n_name {value: " + nation_Model.getNName() + "})\n"+
"CREATE (n_regionkey:n_regionkey {value: " + nation_Model.getNRegionkey() +
"})\n"+
"CREATE (n_comment:n_comment {value: " + nation_Model.getNComment() + "})\n";
	}
```

*** Matching with the table contents

```
 String createStmnt = Create_Statements.createStatementForNation(nationObj);
                          String neoStmt = "\nMATCH \n" +
                  " (nation: Nation) - (supplier: Supplier)\n" +
              "WHERE nation.n_nationkey = supplier.s_nationkey\n" +
                     "RETURN "+nationObj.getNNationkey()+" "
                          +nationObj.getNRegionkey()+" "
                            +nationObj.getNName()+" "
                          +nationObj.getNComment()+"\n"+
                             "MATCH (n) return n\n";
```

4. GOOGLE WEB GRAPH

Google web graph is a framework for graph compression aimed at studying web graphs. It provides simple ways to manage very large graphs. The Google web graph describes the directed links between the pages of the World Wide Web. A graph, in general, consists of several nodes, some pairs connected by edges. In a directed graph, edges are directed lines or arcs. The web graph is a directed graph, whose vertices correspond to the pages of the WWW, and a directed edge connects page X to page Y if there exists a hyperlink on page X, referring to page Y. With google web graph we can access and analyze very large web graphs. Using Web Graph is as easy as installing a few jar files and downloading a dataset. This makes studying phenomena such as PageRank, distribution of graph properties of the web graph, etc.

We can view the static Web consisting of static HTML pages together with the hyperlinks between them as a directed graph in which each web page is a node and each hyperlink a directed edge.

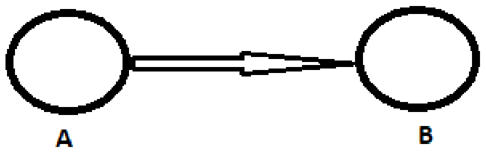

Figure 1**:** Two nodes of the web graph joined by a link.

Figure 1 shows two nodes A and B from the web graph, each corresponding to a web page, with a hyperlink from A to B. We refer to the set of all such nodes and directed edges as the web graph. Figure also shows that (as is the case with most links on web pages) there is some text surrounding the origin of the hyperlink on page A. This text is generally encapsulated in the href attribute of the <a> (for anchor) tag that encodes the hyperlink in the HTML code of page A, and is referred to as anchor text [4].

Input data: From “Google_input.txt”, considering a sample record say:

0	11342
0	824020
0	867923
0	891835

Code snippet:

For Creating and Merging the statements here is the sample of lines:

```
String createStmts =
"create(a:node_from{f_id:"+googleDataObj.getA()+"})\n"+

"create (b:node_to {to_id: "+googleDataObj.getB()+"})\n"+
and so on

String matchStmts =
"MERGE(a:node_from{f_id:"+googleDataObj.getA()+"})\n"+
"MERGE(b:node_to{to_id:"+googleDataObj.getB()+"})\n"+


Relationship statements:

String relationStmnts =
"create(a)[:directional_edge]>(b)\n"+

"create(a)[:directional_edge]>(c)\n"+
```

And the resulting output for the above input data is

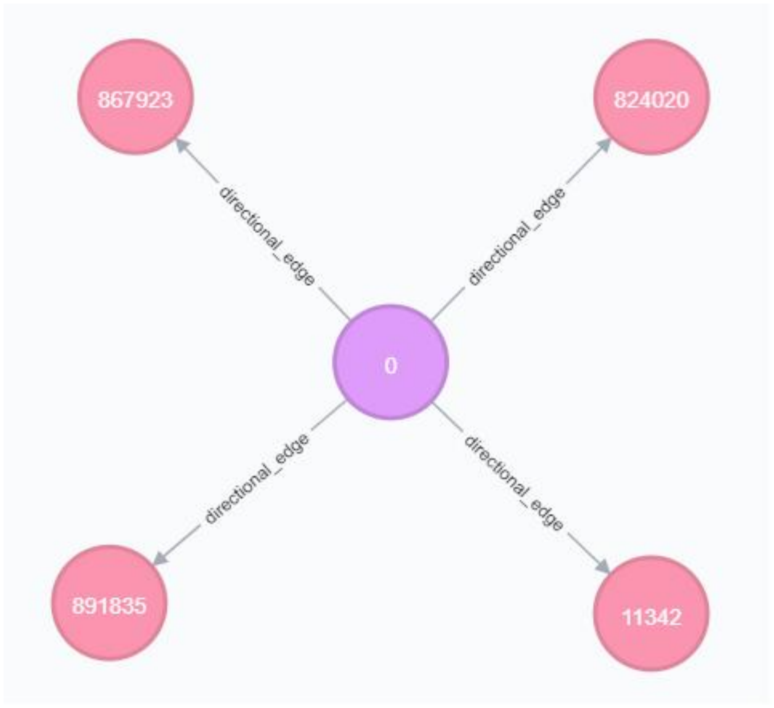

Figure 2: Google Web Graph sample output

Here, the dataset consists of two nodes **"From Node and To Node"**, A line represents the directional edge via the FROM NODE to the TO NODE. It first tries to find a node with the exact parameters specified and then returns if it found. Otherwise it creates a new node with the specified parameters and returns it. For example: **"0→11342"**, **"0→824020"** and so on.

5. <u>MICROSOFT ACADEMIC GRAPH</u>

There is an ever growing interest in datasets of scholarly publications. Research evaluation and analysis, recommender systems, search relevance ranking and literature based discovery are only some of the research areas which make use of the data. However, one of the biggest obstacles in this area is the difficulty in accessing the publications, which is caused by many factors, from the sheer number of articles being published every day and the number of publishing venues.

The Microsoft Academic Graph (MAG) is a heterogeneous graph containing scientific publication records, citation relationships between those publications, as well as authors, institutions, journals, conferences, and fields of study.

Papers	126,909,021
Authors	114,698,044
Institutions	19,843
Journals	23,404
Conferences	1,283
Conference instances	50,202
Fields of study	50,266

Table 1: Dataset size

MAG is a large heterogeneous graph comprised of more than 120 million publications and the related authors, venues, organizations, and fields of study. As of today, the MAG is the largest publicly available dataset of scholarly publications and the largest dataset of open citation data.

MAG models scholarly communication activities and which consists of six types of entities:

- publications
- authors
- institutions (affiliations)
- venues (journals and conferences)
- field of study
- Events (specific conference instances) and the relations between these entities-citations, authorship, etc.
- The dataset contains publication metadata, such as year of publication, title and DOI [5].

Input data: From "MAG_Input.txt" (considered first 50 records of data from the file), considering a sample record this result is for one record of data

Code snippet: For Creating, Merging and relationship statements here is the sample of lines:

```
String createStmnts = "";
String mergeStmnts = "";

if( magDataObj.getTitle()!=null && magDataObj.getTitle().length() >1 )
{
createStmnts=createStmnts+"create(neo4j:paper{title:"+magDataObj.getTitle()+"
})\n";
}

if( magDataObj.getLang()!=null && magDataObj.getLang().length() >1 ) {
createStmnts=createStmnts+"create(language:text{language:"+magDataObj.getLang
()+"})\n";
mergeStmnts = mergeStmnts + "MATCH (a:paper), (b:text) WHERE
a.title="+magDataObj.getTitle()+" AND b.language="+magDataObj.getLang()+"
CREATE (a)-[:language]->(b)\n";
```

`and so on`

And the resulting output for the above input data is

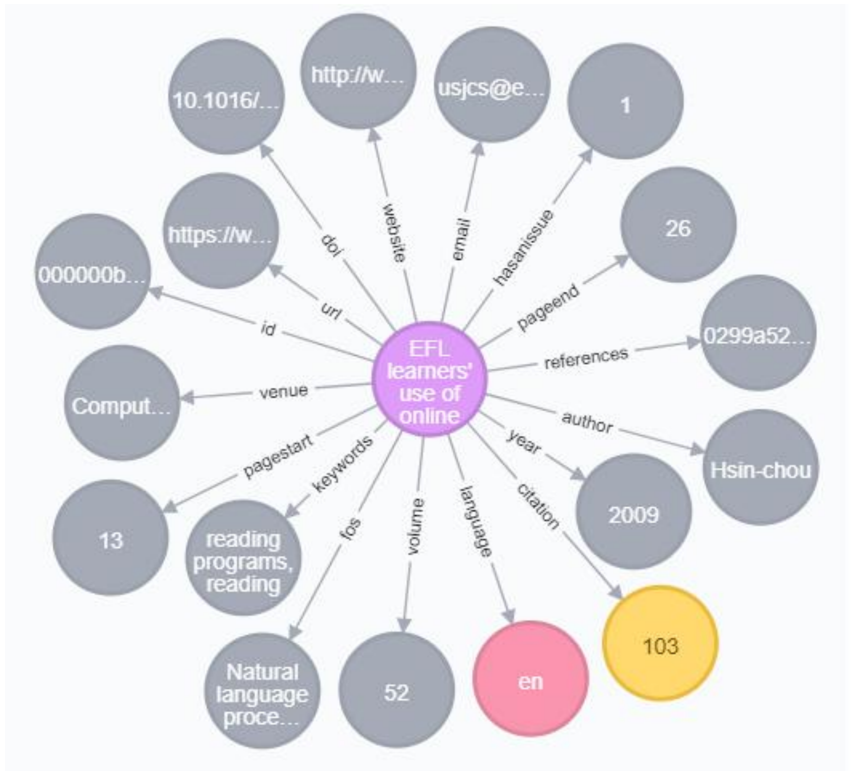

Figure 3: MAG dataset output sample

In simpler words, here, we have considered Title as a centric node because the dataset tells that each paper has a title and all the respective contents of it like abstract, authors, keywords, references, links etc, for some papers contents differ with each other. So we related each paper with their content as a Relationship tag. For example:

"PAPER→AUTHOR",
"PAPER→KEYWORDS".

6. YAHOO FINANCIAL DATASET

A financial dataset comes from a Business term. Many companies worldwide indulge in Stocks, the stock will change regularly every minute depends on the flow of the market. We are considering Yahoo financial dataset which is real time data which can be downloaded from the following source: https://finance.yahoo.com/quote/CSV/history/ from this we can download yester years to current date stocks and ETFs. We can find out the summary report of Market Capital, Previous close and Current Open values, Daily volume, average volume, dividends and splits and many more.

After downloading the dataset we have the following fields to be noted about, they are following:

- High & Low: Indicates the price range at which the stock has traded at throughout the day. In other words, these are the maximum and the minimum prices that people have paid for the stock.
- Volume: is the total number of shares traded for the day.
- Open and Close: The opening and closing price of a stock at the close of the trading day.
- Slightly complex one is the adjusted closing price that uses closing price as base and takes in factors su1ch as dividends, stock splits and new stock offerings to determine a value.
- The adjusted closing price represents a more accurate reflection of a stock's value.
- Open it: is the listing of the stock when stock market opens everyday for trading. And that's why if you see it's all starting with 0.

Input data: From "Yahoo_Input_Data.txt", considering one record of input

Date,Open,High,Low,Close,Volume,OpenInt

2010-07-21,24.333,24.333,23.946,23.946,43321,0

Code snippet: For Creating, Merging and relationship statements here is the sample of lines:

```
String createStmts =
"CREATE (neo4j:yahoo {Date: \""+yadaDataObj.getDate()+"\"})\n"+
"CREATE (Open:openstock {value: \""+yadaDataObj.getOpen()+"\"})\n"+
"CREATE (High:highstock {value: \""+yadaDataObj.getHigh()+"\"})\n"+
"CREATE (Low:lowstock {value: \""+yadaDataObj.getLow()+"\"})\n"+
"CREATE (Close:closestock {value: \""+yadaDataObj.getClose()+"\"})\n"+
"CREATE (Volume:volumestock {value: \""+yadaDataObj.getVolume()+"\"})\n"+
"CREATE (Openint:openintstock {value: \""+yadaDataObj.getOpenInt()+"\"})\n"+
"\n";

String matchStmts =
"MATCH (a:yahoo), (b:openstock) WHERE a.Date="+yadaDataObj.getDate()+" AND
b.value="+yadaDataObj.getOpen()+" CREATE (a)- [r:open_stock]->(b) RETURN
a,b\n"+
"MATCH (a:yahoo), (c:highstock) WHERE a.Date="+yadaDataObj.getDate()+" AND
c.value="+yadaDataObj.getHigh()+" CREATE (a)- [r:high_stock]->(c) RETURN
a,c\n"+
"MATCH (a:yahoo), (d:lowstock) WHERE a.Date="+yadaDataObj.getDate()+" AND
d.value="+yadaDataObj.getLow()+" CREATE (a)- [r:low_stock]->(d) RETURN
a,d\n"+
"MATCH (a:yahoo), (e:closestock) WHERE a.Date="+yadaDataObj.getDate()+" AND
e.value="+yadaDataObj.getClose()+" CREATE (a)- [r:close_stock]->(e) RETURN
a,e\n"+
"MATCH (a:yahoo), (f:volumestock) WHERE a.Date="+yadaDataObj.getDate()+" AND
f.value="+yadaDataObj.getVolume()+" CREATE (a)- [r:volume_stock]->(f) RETURN
a,f\n"+
"MATCH (a:yahoo), (g:openintstock) WHERE a.Date="+yadaDataObj.getDate()+" AND
g.value="+yadaDataObj.getOpenInt()+" CREATE (a)- [r:openint_stock]->(g)
RETURN a,g\n"+
"MATCH (n) return n\n";
.
```

And the resulting output for the above input data is

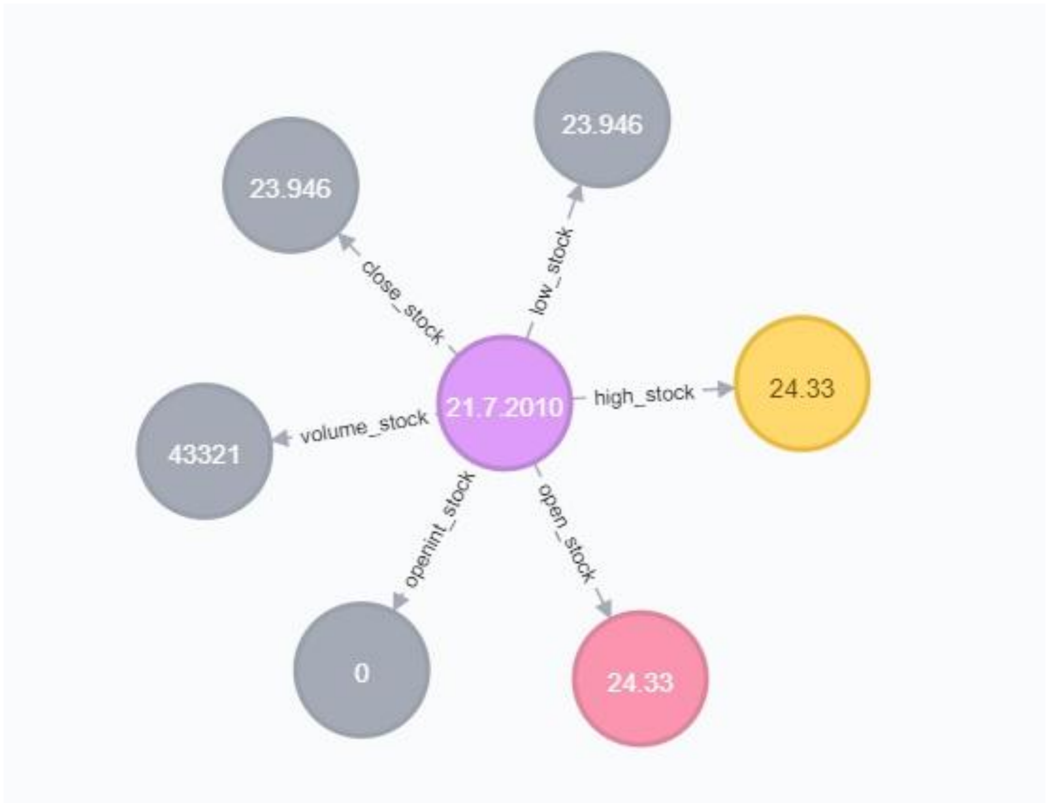

Figure 4: Yahoo dataset output sample

In simpler words, here, we have considered Date as a centric node because the dataset tells that each date has different values of stocks so it differs with each other days. So we have created relationships between:

Date→Open_stock,
Date→Close_stock,
Date→ Low_stock,
Date→high_stock,
Date→Volume_stock and
Date→Openint_stock

7. DATASET INFORMATION: The above datasets can be downloaded from the following official websites:

1. TPC-H: https://relational.fit.cvut.cz/dataset/TPCH
2. Google Web Graph: https://snap.stanford.edu/data/web-Google.html
3. MAG dataset: https://aminer.org/open-academic-graph
4. Yahoo financial dataset: https://finance.yahoo.com/quote/CSV/history?p=CSV

DATASET	**ARTIFICIAL/REAL**	**FILE FORMAT**	**SIZE**	**NODES**	**EDGES**
TPC-H	Real	CSV	2 GB	-	-
Google Web Graph	Real	Text	71.8 MB	875713	5105039
MAG dataset	Real	CSV	100 GB	384 (for a file of 48 records	336 (for a file of 48 records)
Yahoo financial dataset	Real	CSV	115 MB	9800 (for a file of 1400 records)	8400 (for a file of 1400 records

Table 2: Dataset information

8. CONCLUSION:

The above mentioned dataset information are real time data which has huge information if we dig in further. We have created only some samples of records from each dataset since the dataset is huge and each of them is not identical with other records (particularly in MAG dataset, but the tool does work for whole datasets). After achieving the results it is very good to see the visualization part for some records from each of the datasets.

9. REFERENCES:

1) https://docs.spring.io/spring-data/neo4j/docs/2.3.0.RELEASE/reference/html/neo4j.html

2) https://relational.fit.cvut.cz/dataset/TPCH

3) https://www.exasol.com/en/blog/10-questions-about-the-tpc-h-benchmark/

4) https://nlp.stanford.edu/IR-book/html/htmledition/the-web-graph-1.html

5) http://www.dlib.org/dlib/september16/herrmannova/09herrmannova.html

6) https://finance.yahoo.com/quote/CSV/history/