PRODUCT RECALL IN SUPPLY CHAIN MANAGEMENT USING NEO4J GRAPH DATABASE

A Project

Presented to the faculty of the Department of Computer Science
California State University, Sacramento

MASTER OF SCIENCE

in

Computer Science

by

Sushma Gajendra

FALL
2016
PRODUCT RECALL IN SUPPLY CHAIN MANAGEMENT USING NEO4J GRAPH DATABASE

A Project

by

Sushma Gajendra

Approved by:

__________________________________, Committee Chair
Ying Jin, Ph.D.

__________________________________, Second Reader
Xiaoyan Sun, Ph.D.

__________________________________
Date
Student: Sushma Gajendra

I certify that this student has met the requirements for format contained in the University format manual, and that this project is suitable for shelving in the Library and credit is to be awarded for the Project.

__________________________________________, Graduate Coordinator

Ying Jin, Ph.D.  

Date

Department of Computer Science
Abstract

of

PRODUCT RECALL IN SUPPLY CHAIN MANAGEMENT USING NEO4J GRAPH DATABASE

by

Sushma Gajendra

The web connected world we are living in, imparts a gamut of data that can be utilized for various purpose given the advent of web 2.0. Such data that is evolving, can be represented in a data structure like a ‘Graph’. Of the major categories of NoSQL databases, Graph database has been successful in handling the complexity of a highly inter-connected data.

Neo4j, as the most widely used graph database, is a JVM based open source graph database that allows to model, query and manipulate the data that is stored in the form of a graph. It is ideal for monitoring densely connected semi structured data, especially when it involves numerous joins.

The domain of my project to use Neo4j is ‘supply chain management’. The biggest challenge of supply chain management is its increasingly inter-connected data. With globalization at peak, more complexity is added to the supply networks. Since supply chain network involves a lot of relationships, relational databases require more table joins thereby
increasing complexity and execution time and decreasing performance. The main objective of this project is to solve one complex and commonly faced supply chain management issue, i.e. product recall. A few reasons that lead to product recall could be because the product was infected, defective, substandard or malfunctioning. This leads us to backtrack the supply chain and identify the root cause while recalling orders. Often this is achieved by a complicated query involving multi-table joins. And when dealing with large datasets, it’s better to find an efficient solution to solve such problems.

This project includes two different types of implementation that involves creating & manipulating the graph data. The first implementation is a standalone application via Neo4j server. The server can be accessed through a web browser and provides a platform to run and visualize ad-hoc queries. The second implementation is a java based web application that includes an interactive user interface and processes data upon button click. In this case, I have the database embedded within my project and use Neo4j’s java API’s to manipulate the database and provide output. The web service is hosted on my local Tomcat server. Recalls in legacy systems typically involve looking up data from various source tables and joining them to form a solution. Using graph database provides an efficient solution to the recall problem because traversals from the source to destination involves finding a path based on relationships between nodes. This eliminates long query execution time because of joining multiple tables and thereby increasing efficiency. This is demonstrated further in this report.
_______________________, Committee Chair

Ying Jin, Ph.D.

_______________________

Date
DEDICATION

To my loving husband (Pourna Chandra), parents (Jayanthi & Gajendra), sister (Suchitra), parents-in-law (Sarala & Narayana Reddy), sister-in-law(Kunkuma Varma), brother-in-law(Manjunatha Reddy) and my adorable nephew (Rishik).
CONTENTS

Page

Dedication ................................................................................................................... viii

List of Tables ................................................................................................................. xi

List of Figures ............................................................................................................... xii

Chapter

1. INTRODUCTION .................................................................................................. 1

2. BACKGROUND ....................................................................................................... 5

   2.1 Objective of Graph Databases ........................................................................... 5

   2.2 Overview of Neo4j ......................................................................................... 5

   2.3 Domains of Neo4j ......................................................................................... 7

3. CONFIGURING NEO4J ...................................................................................... 11

   3.1 Requirements ............................................................................................... 11

   3.2 Installation ................................................................................................... 11

   3.3 Neo4j Access mode ..................................................................................... 14

   3.4 Query Manipulation in Neo4j ..................................................................... 17
### 4. PRODUCT RECALL USING NEO4J

- **4.1 Food Supply Chain and Dataset exploration** ........................................ 20
- **4.2 How to achieve product recall using Neo4j** ........................................ 23
- **4.3 Neo4j Web Browser Manipulation** ................................................... 23
- **4.4 Neo4j Web Application for Product Recall** ......................................... 41
- **4.5 Challenges faced** .............................................................................. 50

### 5. CONCLUSION AND FUTURE WORK

- **APPENDIX** ............................................................................................ 53
- **BIBLIOGRAPHY** .................................................................................... 56
## LIST OF TABLES

<table>
<thead>
<tr>
<th>Tables</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1. Types of NoSQL Databases</td>
<td>2</td>
</tr>
<tr>
<td>Table 2. Basic SQL and CQL commands</td>
<td>18</td>
</tr>
<tr>
<td>Table 3. List of commonly used CQL commands</td>
<td>18</td>
</tr>
</tbody>
</table>
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figures</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1. Nodes in Neo4j</td>
<td>6</td>
</tr>
<tr>
<td>Figure 2. Relationships between nodes in Neo4j</td>
<td>6</td>
</tr>
<tr>
<td>Figure 3. Installing Neo4j CE on local machine</td>
<td>12</td>
</tr>
<tr>
<td>Figure 4. Choose a location in local machine for graph database</td>
<td>12</td>
</tr>
<tr>
<td>Figure 5. Starting the Neo4j server to activate Neo4j web browser</td>
<td>13</td>
</tr>
<tr>
<td>Figure 6. Neo4j web browser home</td>
<td>13</td>
</tr>
<tr>
<td>Figure 7. Maven plugin for eclipse</td>
<td>15</td>
</tr>
<tr>
<td>Figure 8. Installing Maven for Eclipse</td>
<td>15</td>
</tr>
<tr>
<td>Figure 9. Maven NEO4J jar dependencies</td>
<td>17</td>
</tr>
<tr>
<td>Figure 10. Understanding food supply chain management</td>
<td>21</td>
</tr>
<tr>
<td>Figure 11. Domain model of food supply chain</td>
<td>22</td>
</tr>
<tr>
<td>Figure 12. Load supplier1.csv using LOAD CSV in Cypher</td>
<td>24</td>
</tr>
<tr>
<td>Figure 13. Supplier 1 nodes represented in the graph database</td>
<td>24</td>
</tr>
<tr>
<td>Figure 14. Load supplier2.csv using LOAD CSV in Cypher</td>
<td>25</td>
</tr>
</tbody>
</table>
Figure 15. Supplier 2 nodes represented in the graph database .......................... 25

Figure 16. Load Factory.csv using LOAD CSV in Cypher ............................. 26

Figure 17. Factory nodes represented in the graph database ............................ 26

Figure 18. Load Distributor.csv using LOAD CSV in Cypher ........................... 27

Figure 19. Distributor nodes represented in the graph database ....................... 27

Figure 20. Load Retailer.csv using LOAD CSV in Cypher ................................. 28

Figure 21. Retailer nodes represented in the graph database .......................... 28

Figure 22. Creating relationships between supplier 1 and supplier 2 nodes .......... 29

Figure 23. Graphical visualization of supplier relationships in the database .......... 29

Figure 24. Creating relationships between supplier 2 and factory nodes .......... 30

Figure 25. Graphical visualization of factory relationships in the database .......... 30

Figure 26. Creating relationships between factory and distributor nodes .......... 31

Figure 27. Graphical visualization of distributor relationships in the database ....... 31

Figure 28. Creating relationships between distributor and retailer nodes .......... 32

Figure 29. Graphical visualization of retailer relationships in the database ......... 32

Figure 30. Retailer output nodes ................................................................. 33
Figure 31. Distributor output nodes ......................................................... 34
Figure 32. Distributor output path ............................................................. 34
Figure 33. Factory output nodes ................................................................. 35
Figure 34. Factory output path ................................................................. 35
Figure 35. Level 2 supplier nodes ............................................................... 36
Figure 36. Level 2 supplier output path ...................................................... 37
Figure 37. Level 1 supplier nodes ............................................................... 38
Figure 38. Total recall output path .............................................................. 38
Figure 39. Partial view of the graph database .............................................. 39
Figure 40. Instance (1) of the supply chain graph ....................................... 39
Figure 41. Instance (2) of the supply chain graph ....................................... 40
Figure 42. Closer view at the graph database ............................................. 40
Figure 43. Database Information in the web browser ................................... 41
Figure 44. Recall output in row format ...................................................... 41
Figure 45. Recall output in text format ..................................................... 42
Figure 46. Java class file ........................................................................... 43
Figure 47. Neo4j java service, path and driver setup ........................................... 43
Figure 48. Session object to create nodes & index .............................................. 44
Figure 49. Creating relationship between nodes .................................................. 44
Figure 50. Querying retailer node ...................................................................... 45
Figure 51. Shutdown the graph database .............................................................. 45
Figure 52. Deleting the graph database ................................................................. 46
Figure 53. Architecture for the graph database web application ....................... 46
Figure 54. Graph database web application ......................................................... 47
Figure 55. Retailer recall ..................................................................................... 47
Figure 56. Distributor recall ................................................................................ 48
Figure 57. Factory recall ...................................................................................... 48
Figure 58. Supplier recall ..................................................................................... 49
Figure 59. CREATE node command (1) ............................................................... 53
Figure 60. CREATE node command (2) ............................................................... 53
Figure 61. DELETE command (1) .................................................................... 54
Figure 62. DELETE command (2) .................................................................... 54
1. INTRODUCTION

With the ever evolving trend in big data, organizations have been redefining and re-evaluating existing strategies involving traditional relational database management systems which were designed to address data that was structured, smaller in volume and required fewer updates [1]. Now, with big data emerging rapidly, a growing number of companies rely on NoSql databases to handle the complexity of everyday data. As a result we now have various NoSql databases that enterprises can choose from based on their developing data needs.

NoSQL database systems abbreviation of not only SQL, means non-relational, database systems. Currently there are more than 225 NoSQL databases in use for achieving various functionalities [10]. NoSQL databases are grouped into various clusters based on the way they store data. Few of the mostly commonly used are mentioned in the table below [10].
<table>
<thead>
<tr>
<th>Type of NoSQL Database</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wide Column Store/Column Families</td>
<td>Hadoop, Cassandra, Druid etc</td>
</tr>
<tr>
<td>Document Store</td>
<td>MongoDB, IBM Cloudant, CouchDB etc</td>
</tr>
<tr>
<td>Key Value / Tuple Store</td>
<td>DynamoDB, Oracle NoSQL DB, Sophia</td>
</tr>
<tr>
<td>Graph Databases</td>
<td>Neo4j, Titan, Trinity, GraphBase etc</td>
</tr>
<tr>
<td>Multi Model Databases</td>
<td>Wonder DB, Foundation db etc</td>
</tr>
<tr>
<td>Object Databases</td>
<td>Object DB, Objectivity, StarCounter etc</td>
</tr>
<tr>
<td>Grid and Cloud Database Solutions</td>
<td>Oracle Coherence, GemFire etc</td>
</tr>
<tr>
<td>XML Databases</td>
<td>BaseX, QizX etc</td>
</tr>
<tr>
<td>Multidimensional Databases</td>
<td>Dagger DB, Globals etc</td>
</tr>
<tr>
<td>Multivalue Databases</td>
<td>U2, jBase, Reality etc</td>
</tr>
</tbody>
</table>

Table 1 Types of NoSql databases

There are many advantages to using NoSql databases. Few of them are mentioned below that applies to all NoSql databases in general.

i. Elastic Scaling

NoSQL DB’s are designed to expand horizontally which facilitates the addition of new nodes at any point of time. On the contrary, SQL DB’s can only expand vertically which in turn leads to multiple servers and increasing the cost.
ii. Bigger Data Handling Capabilities:

NoSQL DB’s allow data to be spread across numerous servers without affecting performance.

iii. Cheaper Maintenance:

NoSQL features like automatic repair, easy data distribution and simple data models which helps in keeping the DB maintenance economical. Whereas, SQL DB’s require a skilled DBA’s which can turn out more expensive than a NoSQL DB maintenance.

iv. Integrated Caching Facility:

To increase performance, advanced NoSQL techniques allows to cache data in system memory.

This also provides increased output.

v. Lesser Service Costs:

Typically NoSQL DB’s use clusters of inexpensive servers to manage the ever booming data as compared to the proprietary storage systems SQL DB uses.

So when we think about storing data per GB, NoSQL’s way works out much cheaper than SQL.
vi. No Schema:

Data can be populated without defining a schema. This means, we can change the data model any time to suit our business needs and not have to worry about existing data to be consistent.

In this project we will work to resolve one of the most common problems in supply chain management that arises when a faulty product is detected and reported which is also referred as ‘Product Recall’. In order to recall products, organizations must spend quality time and investment in ‘Traceability’. Traceability refers to the ability to verify the history and application of the product throughout its product journey from source to destination. It is a very crucial step because this information is very vital to verify quality assurance and recall products that are not up to the standards. With traceability, it becomes possible for us to rapidly detect the details about the faulty product and it also aids in finding the root cause that led to this.

We use NEO4J, the most popular graph database for product recall in this project. Graph Databases store data in the structure of a Graph. The primitive units of a graph are nodes and relationships. This enables users to fetch data and perform transactions efficiently especially when the data size is humongous. With graph databases the performance remains unaffected with the increase in the size of data.
2. BACKGROUND

Relational Database Management Systems (RDBMS) consists of Tables which contain data in the form of Rows and Columns. After its inception, RDBMS has been one of the most commonly used database. But since the advent of big data, it poses certain limitations which coerced the use of NoSQL Databases. When you have extremely large datasets and transact heavy updates (or inserts or deletes), NoSQL databases perform really well. Of the available NoSql databases. Graph databases in particular have been growing at rapid speed because of its ability to play around highly complex interconnected data. One such graph database is NEO4J.

2.1 Objective of Graph Databases

Graph Databases are useful to store interconnected data. IT may not be the most efficient way to use RDBMS to store interconnected data as the performance decreases and traversal through thick connected data becomes extremely difficult and consuming. SQL queries which contains substantial joins involving more than 40 tables can be efficiently and easily optimized using Graph Databases.

2.2 Overview of NEO4J

NEO4J stores data within its nodes and relationship. Both nodes and relationships have properties. Properties are Key-Value pairs. One or more nodes that are connected via a relationship is known as a Path. To fetch data from the graph, one needs to traverse
it. Traversing a graph is nothing but visiting nodes based on some relationship and its rules. Traversals can be either depth first or breadth first.

A graph stores data in nodes and relationships. Both nodes as well as relationships can have properties. This is sometimes referred to as the "Property Graph Model" [5].

**Nodes:** Nodes are used to represent *entities* [5].

**Relationship:** Relationships between nodes are key part of any graph database. They are responsible to enable finding related data. Relationships can also have properties [5].
Neo4j is a fully ACID transactional, java based, embedded database that stores data in the form of nodes and properties in graphs instead of rows and columns in a table [3]. The developers claim it is exceptionally scalable (more than a billion nodes on one machine – although it requires an enterprise version of Neo4j), provides easy-to-use API’s which support efficient traversals. Neo4j is integrated with Apache’s Lucene 3 for indexing and search functionality. Lucene is a text search engine which is written in Java and provides high performance.

Some of the prominent features of Neo4j are mentioned below:

- Neo4j is fully compliant with ACID (Atomicity, Consistency, Isolation and Durability) properties [11].
- Neo4j can be integrated on systems that are running on different programming languages and not just java.
- Supports vertical and horizontal scaling that is optimized for graphs in getting accurate and timely results from graphs [12].
- Neo4j is whiteboard friendly which makes it fast and productive while working [24].
- Neo4j is also available in Java Library so, instead of writing a separate server or cluster of servers for running the database you can have the database embedded in your application directly [11].
2.3 Domains of Neo4j

Everything around us is connected and can be easily represented in the form of a graph. This provides us with many use cases that can utilize the power of Neo4j. Before choosing supply chain management as my domain to illustrate the use of Neo4j, there were few other use cases discussed.

i) Electricity distribution

- Neo4j can be used to address electricity outages due to leakage (unknown issues), load (high or low, timing dependent) or a faulty transformer problems (pole breakage). In such cases, Neo4j can be utilized to provide the below solutions.
- Finding the best possible transformer that covers the smallest possible geographical area to shut off during leakage.
- Balance electricity supply during peak hours. For example, industrial areas require more electricity than residential areas during the day.

ii) Best shipment problem

- Commonly known as ‘Amazon problem’, it involves finding the best carrier to ship books.
- The best carrier price can depend on the distance in miles from the source (seller) to the destination (buyer), weight of the package to be shipped etc.

iii) Constraint satisfaction
- Constraint satisfaction problems like setting course schedules in a university by making sure certain courses don’t conflict, a professor isn’t teaching 2 courses at the same time, lectures occur during certain timeslots and so on.

- Similar constraint satisfaction problems can be modelled and solved using graphs.

iv) Real time recommendation engine
- Provide suggestions for products/people based on relationships.
- Similar implementations are available in Facebook (recommend friends), Twitter and Netflix who have implemented their own graph database systems.

v) Software metric analyzer
- Neo4j can be used as a tool to analyze Ruby or Java code which can in turn be helpful to detect ‘code smells’.
- For example: Input a java source code file and we can use APIs to implement methods that determine number of classes in the source file, number of methods in that class, where is a particular library uses and number of times it is used, find the most frequently thrown IO exception. These properties then can be useful to identify code smells like god class, dead code, unreachable code etc. This will provide an overview of the entire software at a glance without having to look at the whole code for analysis purpose.

- How to use: If we detect code smells (using Jdeodrant, Incode, PMD) we need to break the class. First we need to detect the class with the longest inherited path using Neo4j. Then break it down. This can be faster and more efficient.
vi) Full stack web development to compare performance of Neo4j & RDBMS

- While RDBMS performance is indirectly proportional to the size of the database, Neo4j’s performance doesn’t depend on the size of data. This can be shown by creating an interface that can help us access and manipulate Neo4j and RDBMS database simultaneously to demonstrate the performance and benchmark aspects of Neo4j over RDBMS.
3. CONFIGURING NEO4J

3.1 Requirements

The following are the requirements that need to be satisfied in order to create/develop this project:

Java: To install NEO4J on our system we need Java (JDK/JRE) pre-installed. I am using the embedded mode where my local machine itself holds the database.

Eclipse Mars: Integrated development environment to facilitate transactions.

Maven plugin: Build tool to configure dependencies of JAR files used in this project.


3.2 Installation

Once, we download the community edition of NEO4J from https://neo4j.com/download/ we need to run the executable and install it on our local machine. After installation is completed, we need to choose a location of our choice to store/create/manipulate the graph database.

NOTE: Initially by default, the database ‘default.graphdb’ will be created.
The server then displays a dialog box pop-up from where the localhost and port from where we can access NEO4J. This is demonstrated in the figure below.

Figure 3 Installing Neo4j CE on local machine

Figure 4 Choose a location in local machine for graph database
We need to click on “Start” button above to start the Neo4j database in server mode.

![Figure 5](image1.png)

Figure 5 Starting the Neo4j server to activate Neo4j web browser

The default URL is http://localhost:7474/. This will enable the visualization of Neo4j.

![Figure 6](image2.png)

Figure 6 Neo4j web browser home
Finally when we access Neo4j web browser, it should resemble figure 6. The cursor pointer at the top of the page is where we can start creating nodes, relationships, properties and querying them to perform transactions.

Neo4j browser is the graphical user interface provided to access and manipulate the server by running ad-hoc queries [4]. Neo4j instances can also be hosted on the cloud. A few popular Neo4j providers are GrapheneDB [13], GraphStory [13], GraphGrid [13]. All of them are available as a pay as you use pricing model that offer end to end graph hosting services for both community and enterprise editions.

3.3 Neo4j Access mode

We can access Neo4j in various modes. In this project we are accessing Neo4j by embedding it into my Java application in Eclipse. For accessing in embedded mode, Neo4j provides a list of API’s for creating, manipulating and querying the graph database. This can be set up robustly by using Maven, which is a build automation tool. I have listed the below steps that can help us embed Neo4j in our Java project and perform operations via API’s.

1. Copy the Maven plugin link from eclipse M2E website. This is mentioned in the installation section below:

   http://download.eclipse.org/technology/m2e/releases/
2. Go to eclipse -> help -> Install new software -> Paste URL in the ‘work with’ tab.

Check the ‘Name’ and follow the installation wizard to complete the installation process. In order to verify the correct installation of the plugin, we can try to right
click on project explorer -> New -> Other -> new Maven project. If we are able to do
this, then the installation of Maven plugin is successful.

3. Once the plugin is successfully installed, right click in the project explorer
   and create a ‘New Maven project’.

   This is add the default folders along with a file called ‘pom.xml’. This is where
   we need to define our dependencies. In pom.xml, add the following piece of code
   to get all the Neo4j jars and its dependencies.

   ```xml
   <dependency>
     <groupId>org.neo4j</groupId>
     <artifactId>neo4j</artifactId>
     <version>3.0.4</version>
   </dependency>
   
   If you encounter any errors related to JRE, add the following piece of code in
   pom.xml

   ```xml
   <build>
     <plugins>
       <plugin>
         <groupId>org.apache.maven.plugins</groupId>
         <artifactId>maven-compiler-plugin</artifactId>
         <version>2.5.1</version>
         <configuration>
           <source>1.7</source>
           <target>1.7</target>
         </configuration>
       </plugin>
     </plugins>
   </build>
   ```
To ensure the pom.xml has downloaded all the dependencies required by the neo4j version, you can expand the project navigation set up in eclipse and look for Maven dependencies. It should look like figure 9:

![Figure 9 Maven NEO4J jar dependencies](image)

You can see all Jar dependencies of Neo4j version 3.0.4 has been added to our project. This completes the maven project set up and you are good to start operations on Neo4j.

3.4 Query Manipulation in Neo4j

Neo4j has its own query language called CQL (cypher query language) to perform transactions on the data stored in the property graph. CQL is a declarative language that describes ‘What to search’ instead of ‘How to search’.
CQL also helps in identifying potential query problems like expensive queries and deprecated query features in real time by issuing live cypher warnings. This is a very useful feature that can help us build an efficient system from the scratch. Since we are adapted to SQL, CQL is also built similar to SQL to help users understand the language easily without any pre requisites or drastic explanations. Below is the difference between some basic SQL and CQL queries just to give the users an idea about the easiness of CQL.

<table>
<thead>
<tr>
<th>SQL</th>
<th>CQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select * from Product;</td>
<td>MATCH (p:Product)</td>
</tr>
<tr>
<td></td>
<td>RETURN p;</td>
</tr>
<tr>
<td>Select P.ProductName, P.Price</td>
<td>MATCH (p:Product)</td>
</tr>
<tr>
<td>From Product as P</td>
<td>Return p.ProductName, p.Price</td>
</tr>
<tr>
<td>Order by P.unitprice DESC</td>
<td>Order by p.unitprice DESC</td>
</tr>
<tr>
<td>Limit 10;</td>
<td>Limit 10;</td>
</tr>
</tbody>
</table>

Table 2 Basic SQL and CQL commands
Below is a list of commonly used CQL commands [25]:

<table>
<thead>
<tr>
<th></th>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CREATE*</td>
<td>To create nodes, relationships and properties</td>
</tr>
<tr>
<td>2</td>
<td>MATCH</td>
<td>To retrieve data about nodes, relationships and properties</td>
</tr>
<tr>
<td>3</td>
<td>RETURN</td>
<td>To return query results</td>
</tr>
<tr>
<td>4</td>
<td>WHERE</td>
<td>To provide conditions to filter retrieval data</td>
</tr>
<tr>
<td>5</td>
<td>DELETE*</td>
<td>To delete nodes and relationships</td>
</tr>
<tr>
<td>6</td>
<td>REMOVE*</td>
<td>To delete properties of nodes and relationships</td>
</tr>
<tr>
<td>7</td>
<td>ORDER BY</td>
<td>To sort retrieval data</td>
</tr>
<tr>
<td>8</td>
<td>SET*</td>
<td>To add or update labels</td>
</tr>
</tbody>
</table>

Table 3 List of commonly used CQL commands

*Usage demonstrated in appendices
4. PRODUCT RECALL USING NEO4J

4.1 Food Supply Chain and Dataset exploration

Food supply chain is a very complex process that is broadly divided into raw supplier, process supplier, manufacturer, distributor, retailer and consumer. To help you understand these clearly I have mentioned some real world examples via which you can map the terms easily.

i) Raw suppliers are also called as farm owners who own agricultural land and harvest crops seasonally. They supply raw materials required to prepare a product. For example: Tomatoes, Potatoes, wheat, milk, poultry, meat etc.

ii) Process suppliers are still suppliers who purchase raw materials from the raw suppliers and process them to be later used in a product. For example: Tomato puree, milk solids, cleaned and frozen meat, Sugar, refined gluten free flour etc.

iii) Manufacturers are factory owned units that produce a packaged product which is ready to be sold to consumers. For example: Morton Salt Inc., Kirkland etc.

iv) Distributors are responsible for distributing the products to retailers. They purchase products in bulk from the manufacturers and distribute it to many retailers. A distributor can distribute multiple products. For example: Costco, Unilever, ITC etc.

v) Retailers are stores that consumers are familiar with and purchase product from.
For example: Walmart, Safeway, Target, Savemart, Trader Joe’s, Whole foods etc.

Figure 10 is a sample image to understand the food supply chain briefly [7].

![Food Safety ASSURED by series of Internal & Boundary Control Points in Integrated Supply Chain](image)

**Figure 10 Understanding food supply chain management**

Since supply chain is a very complexly interconnected chain, I have modelled my project using custom dataset that consists of all the above nodes in the supply chain. Some of it were part of the collected data available online and rest were created after thoroughly studying the supply chain process related to food industry. Both these datasets have been merged carefully to form that graph database in my project.
I have created a domain model below to show the flow of input resources from farm who supply raw materials all the way through the retailers via whom the product is sold to consumers. The relationships carry information related to logistics.

The dataset [14, 15, 16, 17, 18, 19, 20, 21, 22 and 23] includes the most necessary attributes required to perform recall in the food industry. It comprises of 5 comma separated files that includes information of retailers, distributors, factories and 2 levels of suppliers. The nodes in the database are created based on the csv files and labelled accordingly. The only relationship connecting the nodes based on the properties is Delivers. This model is graphically shown in Figure 11.

Figure 11 Domain model of food supply chain
4.2 How to achieve product recall using Neo4J

Based on the availability of data sources, the scope of this merged dataset is limited to a small business short supply chain management. Product recall involves transparent traceability process where the information flow about the management of produce in each stage of the supply chain plays a crucial role in determining the root cause of the recall. The food supply chain traceability is much more complex than other product traceability as it has a higher level of width and precision.

There are 3 ways to communicate with a Neo4j database:

1. Core API in JAVA
2. Traversal Framework
3. Cypher Query Language (CQL)

I have implemented the project using the Cypher Query Language for manipulating from the Neo4j web browser and using Core Java APIs to build my web application. Both the methods are demonstrated below.

4.3 Neo4j web browser manipulation

Instructions to install and start the neo4j server and loading the web browser are mentioned in Configuration section. To load the dataset in bulk, Neo4j provides cypher commands. The most commonly used command is “LOAD CSV” command. It converts the CSV dataset file into graph format. The usage is demonstrated in figure 12.
1. Loading the supplier1.csv file.

LOAD CSV WITH HEADERS FROM "file:/sup1.csv"

AS csv

MERGE


Figure 12 Load supplier1.csv using LOAD CSV in Cypher

Once we create the nodes in the database, they can be graphically viewed in the server’s web browser. For this we need to specify the node label in the cypher command as shown in Figure 13.

Figure 13 Supplier 1 nodes represented in the graph database
2. Loading the supplier2.csv file.

```
LOAD CSV WITH HEADERS FROM "file:/sup2.csv"
AS csv
MERGE (s2:sup2)
```

Figure 14 Load supplier2.csv using LOAD CSV in Cypher

Figure 15 Supplier 2 nodes represented in the graph database

```
LOAD CSV WITH HEADERS FROM "file:/factory.csv"
AS csv
MERGE (f:Factory
{Name:csv.fName,Product:csv.fProduct,Desc:csv.fDescr,BatchCode:csv.fBatchCode,Price:csv.fPrice,Address:csv.fAddress,ZipCode:csv.fZipCode,
Item:csv.Items})
```

Figure 16 Load Factory.csv using LOAD CSV in Cypher

Figure 17 Factory nodes represented in the graph database
4. Loading the Distributor.csv file.

```plaintext
LOAD CSV WITH HEADERS FROM "file://distributor.csv"
AS csv
MERGE
```

Figure 18 Load Distributor.csv using LOAD CSV in Cypher

Figure 19 Distributor nodes represented in the graph database
5. Loading the supplier1.csv file.

```
LOAD CSV WITH HEADERS FROM "file:retailer.csv"
AS csv
MERGE
(r:Retailer{Name:csv.rName,Product:csv.rProduct,BatchCode:csv.rBatc
```

![Figure 20 Load Retailer.csv using LOAD CSV in Cypher](image)

![Figure 21 Retailer nodes represented in the graph database](image)

After loading the dataset nodes we can create ‘Relationships’ among them in the database to facilitate the traversals during transactions. The cypher query to create
relationship between nodes is mentioned in the following screenshots.

NOTE: “LIMIT” command limits the output to the required number of rows.

1. Creating relationships between supplier 1 and supplier 2 nodes

   MATCH (s1:sup1)
   WITH s1
   MATCH (s2:sup2)
   WHERE s2.RawMaterials CONTAINS s1.Resource
   CREATE (s1)-[:DELIVERS]->(s2)

   Figure 22 Creating relationships between supplier 1 and supplier 2 nodes

2. Creating relationships between supplier 2 and factory nodes

   Figure 23 Graphical visualization of supplier relationships in the database
MATCH (f:Factory)

WITH f

MATCH (s2:sup2)

WHERE f.Item CONTAINS s2.Item

CREATE (s2)-[:DELIVERS]->(f)

Figure 24 Creating relationships between supplier 2 and factory nodes

Figure 25 Graphical visualization of factory relationships in the database

3. Creating relationships between factory and distributor nodes
MATCH (d:Distributor)
WITH d
MATCH (f:Factory)
WHERE f.Product = d.Product
CREATE (f)-[:DELIVERS]->(d)

Figure 26 Creating relationships between factory and distributor nodes

Figure 27 Graphical visualization of distributor relationships in the database

4. Creating relationships between distributor and retailer nodes
MATCH (r:Retailer)

WITH r

MATCH (d:Distributor)

WHERE d.Product = r.Product

CREATE (d)-[[:DELIVERS]]->(r)

Figure 28 Creating relationships between distributor and retailer nodes

Figure 29 Graphical visualization of retailer relationships in the database

Now after we have set up the graph database, we can start querying to find out specific information needed to recall a product. We can take small steps that lead us to
form the total path of the recall. The below screenshots demonstrate a set of queries that together form the end result.

1. Find the retailer stores that are selling the recalled product.

   MATCH (r:Retailer)
   WHERE r.Product = 'pasta'
   RETURN r

![Retail output nodes](image)

Figure 30 Retail output nodes

2. Find the Distributors that distributed the products to retailer.

   MATCH (d:Distributor)-[:DELIVERS]-> (r:Retailer{Product:'pasta',BatchCode:'8000'})
   RETURN DISTINCT d
MATCH (d:Distributor)-[[:DELIVERS]]-(r:Retailer{Product:'pasta',BatchCode:'8000'})
RETURN DISTINCT d,r

Figure 31 Distributor output nodes

Figure 32 Distributor output path
3. Find the factory/manufacturer that produced the defected product

```
MATCH (f:Factory)-[:DELIVERS]-> (d:Distributor)-[:DELIVERS]-> (r:Retailer {Product: 'pasta', BatchCode: '8000'})
RETURN DISTINCT f
```

Figure 33 Factory output nodes

```
MATCH (f:Factory)-[:DELIVERS]-> (d:Distributor)-[:DELIVERS]-> (r:Retailer {Product: 'pasta', BatchCode: '8000'})
RETURN DISTINCT f,d,r
```

Figure 34 Factory output path
4. Find the sources (Level 2) the manufacturer sourced items related to the product being recalled

MATCH (s2:sup2)-[:DELIVERS]->(f:Factory)-[:DELIVERS]->(d:Distributor)-[:DELIVERS]->(r:Retailer {Product:'pasta',BatchCode:'8000'})

WHERE f.Item CONTAINS s2.Item

RETURN s2

Figure 35 Level 2 supplier output nodes

MATCH (s2:sup2)-[:DELIVERS]->(f:Factory)-[:DELIVERS]->(d:Distributor)-[:DELIVERS]->(r:Retailer {Product:'pasta',BatchCode:'8000'})

WHERE f.Item CONTAINS s2.Item

RETURN s2,f,d,r
5. Find the sources (Level 1) the Level 2 suppliers sourced items related to the product being recalled

```cypher
MATCH (r:Retailer{Product:"pasta", BatchCode: "8000"}) <- [:DELIVERS] -(d:Distributor)<- [:DELIVERS] -(f:Factory) <- [:DELIVERS]- (s2:sup2)<- [:DELIVERS]-(s1:sup1)
WHERE s1.Resource CONTAINS s2.RawMaterials
RETURN s1
```
WHERE s1.Resource CONTAINS s2.RawMaterials
RETURN s1, s2, f, d, r
After creating the required nodes, relationships & properties, we can visualize the entire graph database by using a simple CQL command. Please refer to figure 39 to view an instance of the database and the command used.

Figure 39 Partial view of the graph database

The entire graph database looks like figure 40. The nodes have been minimized to fit a part of the graph structure into the screen.

Figure 40 Instance (1) of the supply chain graph
Additionally, we can also see all the types of nodes, relationships and properties available in the database at a glance as shown in Figure 43.
Figure 43 Database Information available in the top right of the web browser

The output can also be displayed in various formats other than graph.

Figure 44 Recall output in row format
4.4 Neo4j Web Application for Product Recall:

This project also includes a standalone hosted web application that can perform product recall on the click of a button. The user interface is designed keeping in mind the complexity of supply chain management. The database is created, manipulated and accessed via the “Bolt” protocol using Cypher Query Language. All the traversals are implemented in Java using Eclipse IDE. (Configuring Neo4j with Java is previously discussed in chapter 3).

My main class file contains methods that consists of queries for different nodes present in the database.
The initial step is to create the graph database at a specified location. Neo4j server by default create a “default.graphdb” in the users/document/Neo4j folder. I will be using the same location in my project. The next important step is to create a graph database service that provides us a handle to manipulate the database in the further steps. Figure 47 provides the necessary steps.

```java
package org.neo4j.ProductRecall;

import java.io.File;

public class pr3 {
    public static List<V10> performDatabaseTransaction_Retailer(String pName, int bCode) {
        File neoDBpath = new File("C:/Users/sushma/Documents/Neo4j/default.graphdb");
        GraphDatabaseService gds;
        public static List<V10> performDatabaseTransaction_Distributor(String pName, int bCode) {
        public static List<V10> performDatabaseTransaction_Factory(String pName, int bCode) {
        public static List<V10> performDatabaseTransaction_Supplier1(String pName, int bCode) {
    }
}
```

Figure 46 Java class file

Once the database is created and connected, we need to establish a session to carry out any kind of transaction. The session once opened must be closed so it can return to the
session pool and reused later on. Using the session object we can create nodes, indexes and relationships in the database. This is demonstrated in Figure 48:

```java
void createDatabase()
{
    /* Creating a graph DB service */
    GraphDatabaseService db = new GraphDatabaseFactory().newEmbeddedDatabase( neo4jPath );
    System.out.println("Database created \n");
    /* Pass the user name and password to login to the server hosted on Local host */
    Config noSSL = Config.build().withEncryptionLevel(Config.EncryptionLevel.NONE).toConfig();
    try
    {
        Driver driver = GraphDatabase.driver( "bolt://localhost:7474", AuthTokens.basic( "neo4j", "hello" ), noSSL);
        System.out.println("Database connected \n");
        Session session = driver.session();
        System.out.println("Session established\n");
        session.run("LOAD CSV WITH HEADERS FROM 'file://suppliers.csv' " + "AS line " + "MERGE({id:line.id}) AS s " + "ON CREATE SET s.name = line.name, " + "s.address = line.address," + "s.zipCode = line.zipCode, " + "s.resource = line.resource, " + "s.price = line.price)" + "RETURN s");
        System.out.println("Unloaded Level 1 supplier file Successfully\n");
        StatementResult suppliersIndex = session.run("CREATE INDEX ON :Supplier(id)\n");
        System.out.println("Uncreated index on Level 1 supplier node\n");
    }";
}
```

**Figure 48 Session object to create nodes & index**

```java
/* ****************** CREATING RELATIONSHIPS BETWEEN NODES ***********************/

StatementResult relationships = session.run("MATCH (s1:Supplier)" + "WITH s1" + "MATCH (s2:Supplier)" + "WHERE s2.Name CONTAINS s1.Resource" + "CREATE (s1)-[:RELATIONSHIP]->(s2)\n");
System.out.println("Uncreated Relationship between Level 1 Supplier and Level 2 Supplier\n");
```

**Figure 49: Creating relationship between nodes**

The method to query the ‘Retailer’ node in order to get the recall output is provided in figure 50:
After performing transaction we need to make sure the database is shutdown to be accessible from the web browser.

```java
public static List<V10> performDatabaseTransaction_Retailer(String pName, int hCode)
{
    String productName = pName;
    int batchCode = hCode;
    List<V10> resList = new ArrayList<V10>();
    V10 res = new V10();
    Config noSSL = Config.builder().withEncryptionLevel(Config.EncryptionLevel.NONE).toConfig();
    Driver driver = GraphDatabase.driver( "bolt://localhost:7687", AuthTokens.basic( "neo4j", "hello" ), noSSL);
    Session session = driver.session();
    StatementResult query1 = session.run( "MATCH (r:Retailer) " + "WHERE r.Product CONTAINS (" + code + ") " + "RETURN DISTINCT r.Name, r.BatchCode, r.Address, r.ZipCode, r.Product, r.Price" + Values.parameters("urn", productName, "code", code, batchCode));
    for (Record r : query1.list())
    {
        res = new V10();
        String supName = r.get("r.Name").toString();
        String supAddress = r.get("r.Address").toString();
        String supZipCode = r.get("r.ZipCode").toString();
        String supProduct = r.get("r.Product").toString();
        String supPrice = r.get("r.Price").toString();
        res.set supName(supName);
        res.setAddress(supAddress);
        res.setZipCode(supZipCode);
        res.setBatchCode(supZipCode);
        res.setProduct(supProduct);
        res.setPrice(supPrice);
        resList.add(res);
    }
    return resList;
}
```

Figure 50 Querying retailer node

If the database has to be deleted, we must make sure to delete all the relationships along with the nodes.

```java
// Shutdown the database after performing the transaction - Else neo4j web browser won't be accessible
void shutdown()
{
    try
    {
        gds.shutdown();
        System.out.println("\nGraphDatabase Shutdown\n");
    }
    catch (Exception e)
    {
        //System.err.println(e.getMessage());
        System.out.println("No Graph Database Found to Shutdown!");
    }
}
```

Figure 51 Shutdown the graph database
/* Deleting all the nodes and relationships from the database */

```java
void deleteDatabase()
{
    Session session = driver.session();
    System.out.println("Deleting graph database");
    session.run("MATCH (n) DETACH DELETE n");
}
```

Figure 52 Deleting the graph database

In addition to this, the java based database application is hosted on a tomcat server that provides an easy-to-use interface. The architecture to construct the web application is mentioned in figure 53. This helps to connect the front end to the backend via servlets.

ARCHITECTURE / DESIGN

Figure 53 Architecture for the graph database web application
The web application screenshot is as mentioned in Figure 54:

Figure 54 Graph database web application

The web application require two user inputs: Name of the product that needs to be recalled and the batch code of the defected product. When the user input is provided, clicking on the button below displays respective solution for product recall.

Figure 55 Retailer recall
Figure 56 Distributor recall

Figure 57 Factory recall
Figure 58 Supplier recall
4.5 Challenges faced

Neo4j is developing rapidly and with the recent release it has come out with some amazing features like the ‘Bolt’ protocol. Since it is only past its budding stage, there are a few necessary features that will be implemented in the future releases. Few of the issues that I faced while implementing are listed in the following steps:

i. Neo4j jars are only available via Maven dependencies.

ii. Due to its rapid releases, the transaction.close() method available in Neo4j 2.0 was deprecated in Neo4j 3.0 and replaced with finish().

iii. By default, ‘LOAD CSV’ exports data as strings. For parameters like Zipcode and Price had to be explicitly converted toInt().

iv. If there is no such column while importing from the file, NEO4j creates a new column (when it doesn’t find a matching one) and returns null values in all rows of this column instead.

v. While loading CSV dataset using LOAD CSV USING HEADERS, if there is a leading empty line, this will break the export.

vi. If there are ‘null’ value cells in the dataset, there are only 2 options to handle it.
   a. Skip the row having the null cell
   b. Set default value upon creating the null cell

Some other things to keep in mind are:

i. The maven project needs to be ‘updated’ every time there is a change.

ii. We need to make sure the maven build is 100% success to make sure all the required dependencies are loaded.
iii.  Ensure to close the session object after a transaction. Leaking session object can lead to garbage data.

iv.  Neo4j is not best suited in cases of legacy applications that contain stored procedures or applications that contain only table with no relationship between them. One of the biggest disadvantage of Neo4j is that we cannot shard the database. This means, the entire graph data should be available on one single server [9].
5. CONCLUSION AND FUTURE WORK

The purpose of this project is to solve the product recall problem in the food supply chain industry by utilizing the power of Neo4j. I have provided 2 different ways to access, manipulate and view the results of product recall. The standalone web application can also be used to host a different dataset with a different purpose and utilize the graph database. To sum it up, my primary goal has been achieved with this project.

Throughout this project I have been constantly learning about the possibilities of Neo4j. It also provides a large possibility to use Neo4j for other applications in various domains like master data management, networks, web analytics, fraud detection etc.

As part of the future work, Neo4j can be used to implement any of the applications mentioned in section 2.3. This project can be used as a foundation for understanding Neo4j, how data is stored in graph databases, and how to manipulate data using cypher query language. We can extend this project by adding more enhancements to the existing project by using real time data from a small/medium size business.
Creating node with label and properties using ‘CREATE’ CQL command in Neo4j:

```
CREATE (sup1:Supplier1 {Name:"Arrowhead Farm", Address:"1289 Granite Bay"});
```

![Figure 59 CREATE node command (1)](image1)

```
CREATE (sup2:Supplier2 {Name:"Mayhem Farm", Address:"3473 Sutter St"});
```

![Figure 60 CREATE node command (2)](image2)

Creating relationships using ‘CREATE’ CQL command in Neo4j:

1. CREATE (sup1:Supplier1) – [:DELIVERS] -> (sup2: Supplier2)
2. CREATE (sup1:Supplier1) – [:DELIVERS] -> (sup2: Supplier2)
   WHERE s2.Item CONTAINS s1.Resource
Deleting a node using CQL command in Neo4j:

In order to delete a node in Neo4j, we must make sure its relationships are also deleted. Otherwise, CQL will throw an error. This is demonstrated in Figure 62.

MATCH (s1:Supplier1) DELETE s1

![Figure 61 DELETE command (1)](image1)

MATCH (s1:Supplier1) - [d:DELIVERS] -> (s2:Supplier2) DELETE s1, s2, d

![Figure 62 DELETE command (2)](image2)

Removing node properties using CQL command in Neo4j:

MATCH (s1:sup1 {Name: 'Davis Ranch'}) REMOVE s1.Address

![Image](image3)
Set properties using CQL command in Neo4j:

To add a new property to an existing node we can use SET command in CQL. It is demonstrated below:

MATCH (s1:Sup1) SET s1.newName = "Sacramento"
BIBLIOGRAPHY


https://neo4j.com/developer/graph-database/

https://neo4j.com/blog/neo4j-scalability-infographic/

https://neo4j.com/developer/guide-cloud-deployment/


[16] Supplier level 2 dataset [Online]. Available:
https://catalog.data.gov/dataset?tags=grocery

http://www.fda.gov/DataSets/Recalls/Food/Food.xml


[21] Retail store dataset [Online]. Available:
https://archive.ics.uci.edu/ml/datasets/Online+Retail
[22] Distributor dataset [Online]. Available:
https://archive.ics.uci.edu/ml/datasets/Wholesale+customers

[23] American Farm Dataset [Online]. Available:
http://www.lovelongears.com/Farm_AE.html


[25] List of CQL commands in Neo4j [Online]. Available:
https://www.tutorialspoint.com/neo4j/neo4j_cql_introduction.htm