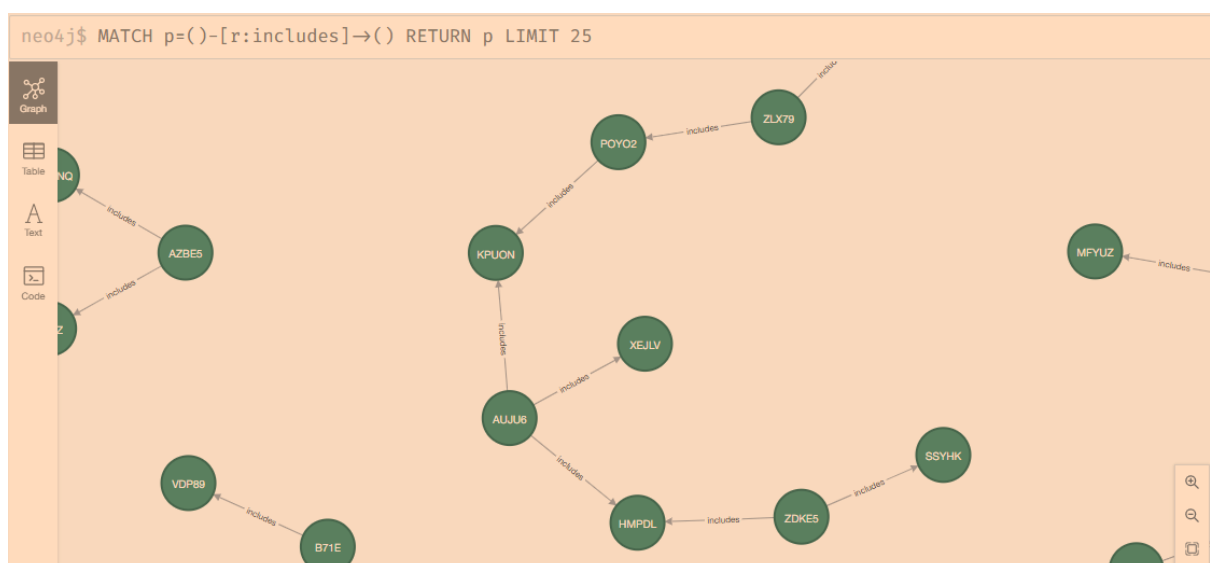


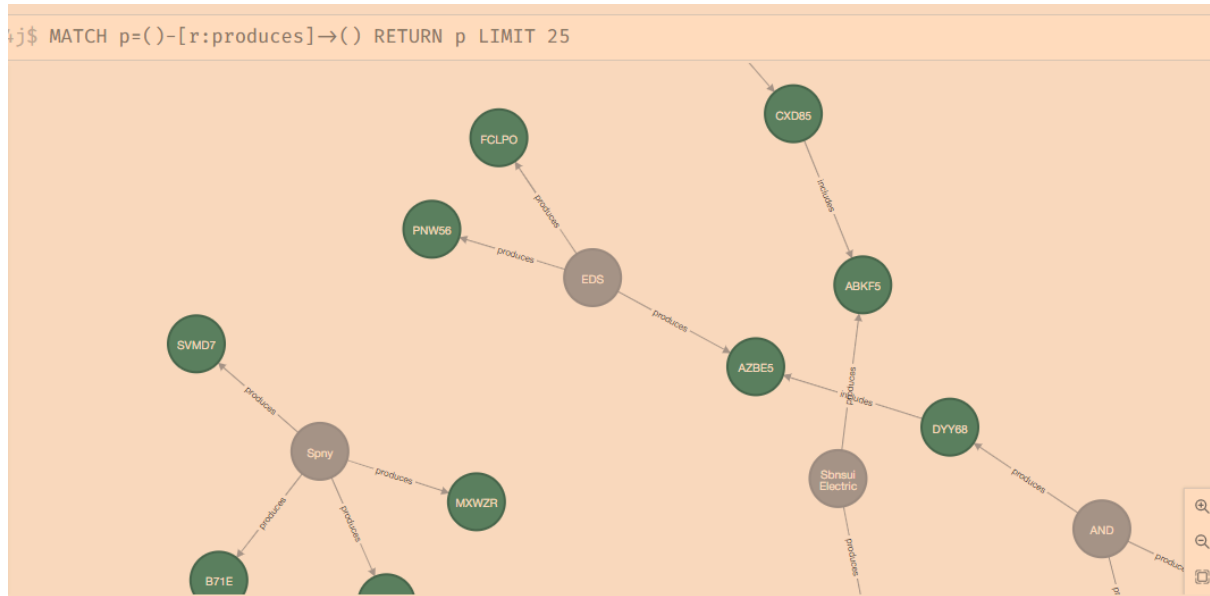
# BKMS Assignment 3

≡ 작성자 2022-24670

## A. Import data to Neo4j from CSV files

```
#import products
LOAD CSV WITH HEADERS FROM "file:///Products.csv" AS row
CREATE (p:product {pid: toInteger(row.PID), product : row.Product, company: row.Compan
Y,
unit_price : toInteger(row.Unit_price)})
#import BOM with relationships
LOAD CSV with headers from "file:///BOM.csv" as row
MATCH (p:product),(p1:product)
WHERE p.pid = toInteger(row.Pid_sub) AND p1.pid = toInteger(row.Pid_parent)
MERGE (p1)-[r:includes {quantity: toInteger(row.Quantity)}]-(p)
#import company nodes
LOAD CSV WITH HEADERS FROM "file:///Products.csv" AS row
WITH row.Company as company
unwind company as comp
with distinct comp
CREATE (c:company {company:comp});
#import produces relations
MATCH(p:product), (c:company)
WHERE p.company=c.company
MERGE (c)-[rel:produces]->(p)
return count(rel)
```



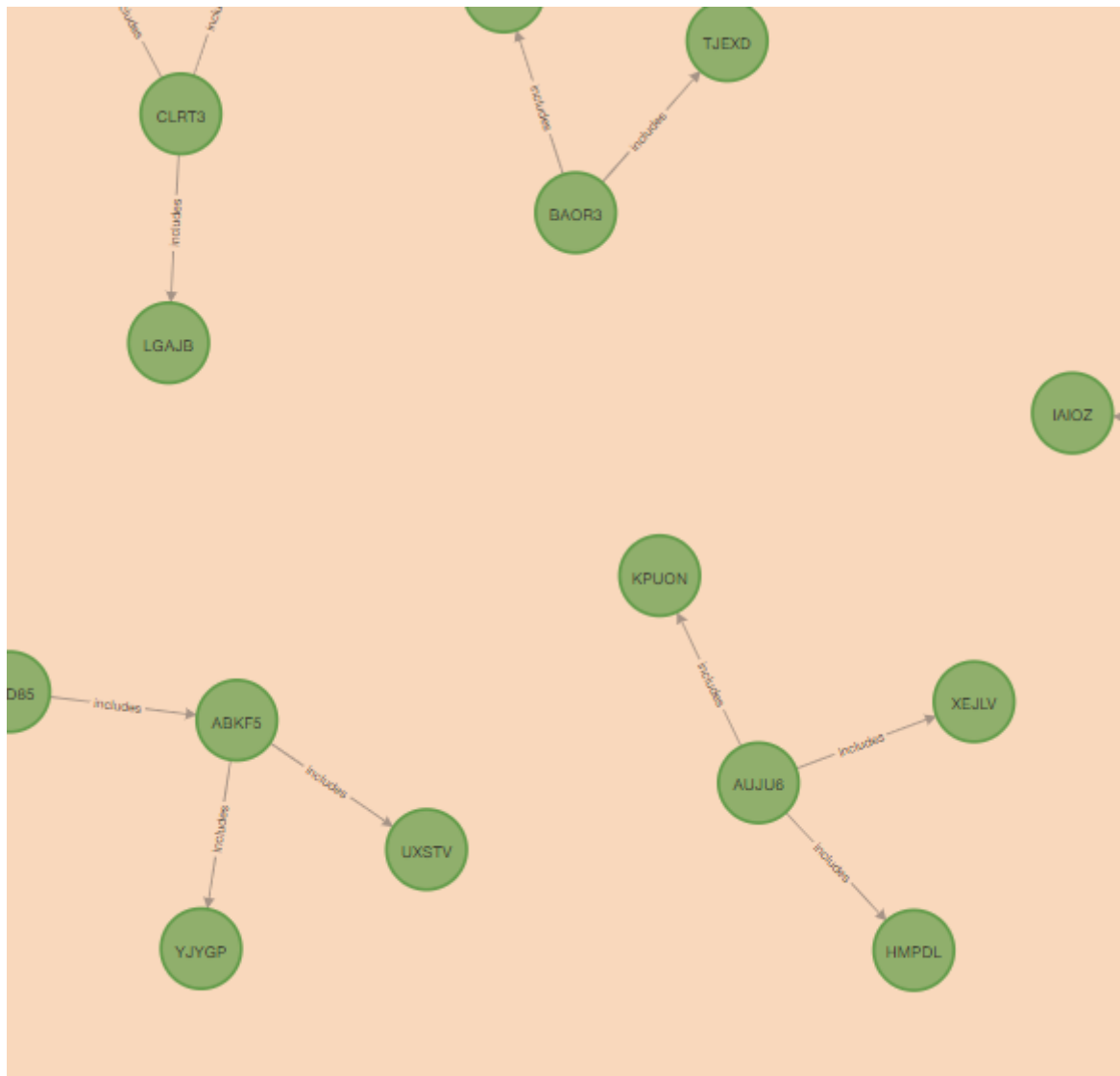


## B. Import data to Neo4j from RDB tables

```
#1
CALL apoc.load.jdbc("jdbc:postgresql://localhost:5432/postgres?user=postgres&password=***", "boms")
YIELD row CREATE (:bom {pid_sub:row.pid_sub, pid_parent:row.pid_parent, quantity:row.quantity})

#2
CALL apoc.load.jdbc("jdbc:postgresql://localhost:5432/postgres?user=postgres&password=***", "boms") YIELD row
MATCH (p:products), (p1:products)
WHERE p.pid = toInteger(row.pid_sub) AND p1.pid = toInteger(row.pid_parent)
MERGE (p1)-[r:includes {quantity: toInteger(row.quantity)}]-(p)
--
MATCH p=()-[r:includes]->() RETURN p LIMIT 25
```

Query above satisfies the graph schema of the question as below



**C. For each database created in A and B, you must perform the following three cypher queries:**

- Query 1: Find all companies that supplies sub-components to the company ‘EDS’**
- Query 2: Find all leaf components and their quantities required to produce a product ‘KQX18’**

## – Query 3: Your own query that is difficult to perform in RDBMS due to the need for many joins

For DB obtained from problem A

Query 1.

```
MATCH p=( :company{company:"EDS"})-->( :product)-[r:includes]->(psub:product)
```

```
Return psub.company;
```

Query 2.

```
MATCH p=(n:product{product:"KQX18"})-[r:includes]->(m:product) RETURN m.product, r.quantity LIMIT 25
```

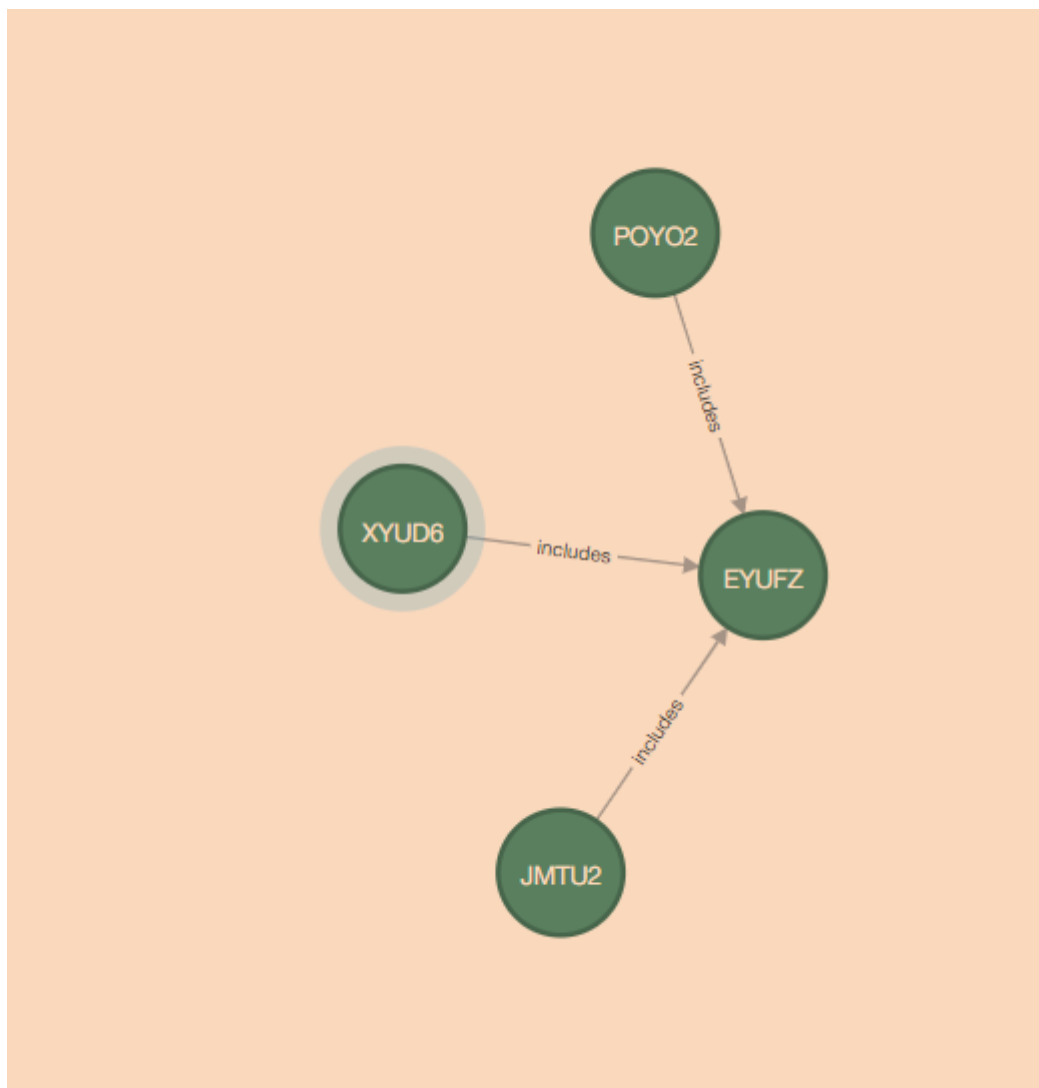
Query 3. there's a big fire to the company of slyworth. which products will have difficulty manufacturing?

```
MATCH p=(b:product)-[r:includes]->(a:product) where a.company = "Slyworth"
```

```
Return p;
```

psub.company	
1	"Slyworth"
2	"Aoalog Devices"
3	"DMink"
4	"AND"

	m.product	r.quantity
1	"AUJU6"	2
2	"ZKNE4"	1
3	"LFNB7"	1



For DB B,

```

Query 1.
MATCH p=( :products{comapny:"EDS"})-[r:includes]->(sub)
Return sub.comapny;
Query 2.

```

```
MATCH p=(n:products{product:"KQX18"})-[r:includes]->(m:products) RETURN m.product, r.quantity LIMIT 25
```

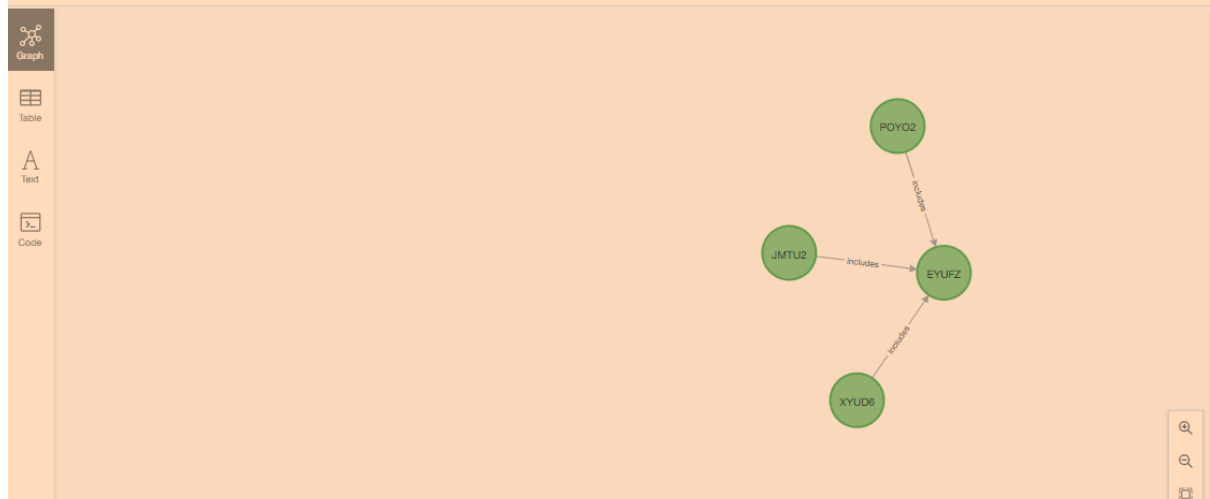
```
neo4j$ MATCH p=(:products{comapny:"EDS"})-[r:includes]->(sub) Return sub.comapny;
```

	sub.comapny
1	"Pjoneer"
2	"Njkon"
3	"EQoX"
4	"ZUE"

```
j$ MATCH p=(n:products{product:"KQX18"})-[r:includes]->(m:products) RETURN m.product, r.quantity LIMIT 25
```

	m.product	r.quantity
1	"LFNB7"	1
2	"ZKNE4"	1
3	"AUJU6"	2

```
neo4j$ MATCH p=(b:products)-[r:includes]->(a:products) where a.comapny = "Slyworth" Return p;
```



## **D. The report must include an analysis of the following:**

### **– Pros and cons of using Neo4j compared with RDBMS**

Pros:

- Neo4j is good at processing highly connected or dynamic data such as networks. It enables agility of whole database.
- Data relationships can be easily found even in a huge database. (Deep search)

Cons

- RDBMS is great for tabular, structured data.
- Neo4J is inappropriate for understanding consistent table.
- Neo\$J is not useful to find the data which is stored in key/value storage

Source: <https://neo4j.com/blog/3-advantages-neo4j-alongside-oracle-rdbms/>

## **Pros and cons of using Neo4j compared with Prolog**

PROS

- Data is guaranteed to store in the graph while prolog conceptually stores graph.
- While prolog reasons through lots of data, neo4j describes a graph pattern the user wish to match.

CONS

- Neo4j cannot solve the problem based on general knowledge.
- Neo4j requires a lot of computing load while prolog does not.

## **Considerations for graph schema design**

- Understand domain and users well
  - graph schema should be optimized for expected behavior of users. Schema should be designed for users to access information in need easily.

- Schema should mirror real graph.
  - The data and the relationships should be reflected in schema.
- Be specific with naming and datatypes.