BKMS Homework 3

2022-24790 박성우

A. Load CSV

- `Neo4j`로 CSV를 import하기 위해서는 우선 플랫폼에 데이터베이스를 생성한 뒤, `Import` 폴더에 파일들을 업로드하는 과정이 선행되어야 한다. 그 후 아래와 같은 방법을 통해 import한 csv 파일로부터 그래프 데이터를 생성할 수 있게 된다.

1. Create constraint

```
create constraint pkey1 on (n:Product) ASSERT n.PID is unique;
create constraint pkey2 on (n:Product) ASSERT n.Product is unique;
create constraint ckey on (n:Company) ASSERT n.Company is unique;
```

2. Load CSV

```
load csv with headers from "file:///Products.CSV" as row
with toInteger(row.PID) as pid, row.Product as pname, row.Company as cname, toInteger(row.Unit_price) as uprice
merge (c:Company{Company: cname})
merge (p:Product{PID: pid, Product: pname, Unit_price: uprice})
merge (c)-[r:PRODUCE]->(p);
```

```
load csv with headers from "file:///BOM.CSV" as row
with toInteger(row.Pid_sub) as pid_sub, toInteger(row.Pid_parent) as pid_parent, toInteger(row.Quantity) as quantity
match (m:Product{PID: pid_parent})
match (n:Product{PID: pid_sub})
merge (m)-[r:INCLUDE{Quantity: quantity}]->(n);
```

B. DB Integration

- 여기서는 `postgres`에 저장되어 있는 데이터를 추출한 뒤 그것을 성공적으로 `Neo4j`에 저장하는 과정이 필요하다. 이는 아래와 같은 command를 통해 수행될 수 있다.

```
call apoc.load.jdbc("jdbc:postgresql://localhost:5432/postgres?user=postgres&password=password","products")
yield row create (p:product {pid:toInteger(row.pid), product:row.product, company:row.company})
```

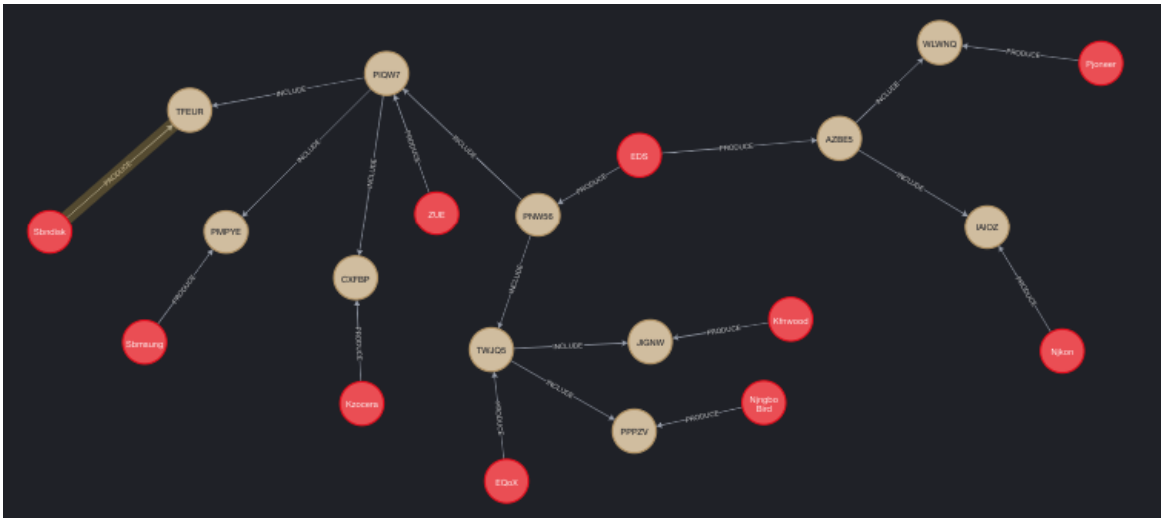
```
call apoc.load.jdbc("jdbc:postgresql://localhost:5432/postgres?user=postgres&password=암호!","bom_csv")
yield row
with toInteger(row.pid_sub) as Pid_sub, toInteger(row.pid_parent) as Pid_parent, toInteger(row.quantity) as Quantity
match (p:product{pid: Pid_parent})
match (q:product{pid: Pid_sub})
merge (p)-[r:INCLUDE{Quantity: Quantity}]->(q)
```

C. Queries

C-(a)

- 아래의 query를 통해 수행될 수 있으며, 수행 결과는 아래 그림과 같다.

```
match p=(c1:Company {Company:"EDS"})-[:PRODUCE]->(q1:Product)-[r:INCLUDE*..]->
(q2:Product)<-[:PRODUCE]-(c2:Company)
return p;
```



All companies that supplies sub-components to the company ‘EDS’

C-(b)

```
match p=(p1:Product {Product: "KQX18"})-[r:INCLUDE*..]->(p2:Product)
where not (p2)-[:INCLUDE]->()
return p2.PID,
reduce(acc=1, q IN relationships(p) | acc*q.Quantity) as required_quantity
```

	p2.PID	required_quantity
1	253	1
2	75	6
3	242	6
4	108	2
5	14	1

All leaf components and their quantities required to produce a product ‘KQX18’

C-(c)

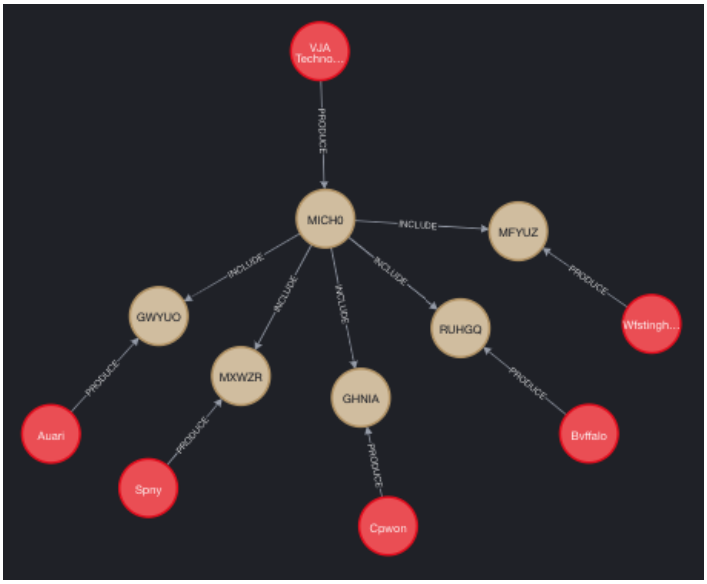
1. 부속품이 가장 많은 제품에 대해서 해당 부속품의 정보와 해당 제조사의 관계를 살펴보는 query

```
// 부속품이 가장 많은 제품 찾기 - SQL 사용 (결과: PID - 131/ Product - MICH0 / 5개)
select
  b.pid_parent, p.Product, p.Company,
  count(distinct b.pid_sub)
from bom_csv b join products p
on b.pid_parent = p.pid
group by 1,2,3
order by 4 desc
limit 1;
```

```
// 부속품이 가장 많은 제품 MICH0에 대해 모든 부속품 목록 및 해당 부속품에 관련된 정보
match p=(c1:Company)-[:PRODUCE]-(p1:Product {Product: "MICH0"})
-[:INCLUDE*..]->(p2:Product)<-[:PRODUCE]-(c2:Company)
return p, p2.PID, p2.Unit_price;
```

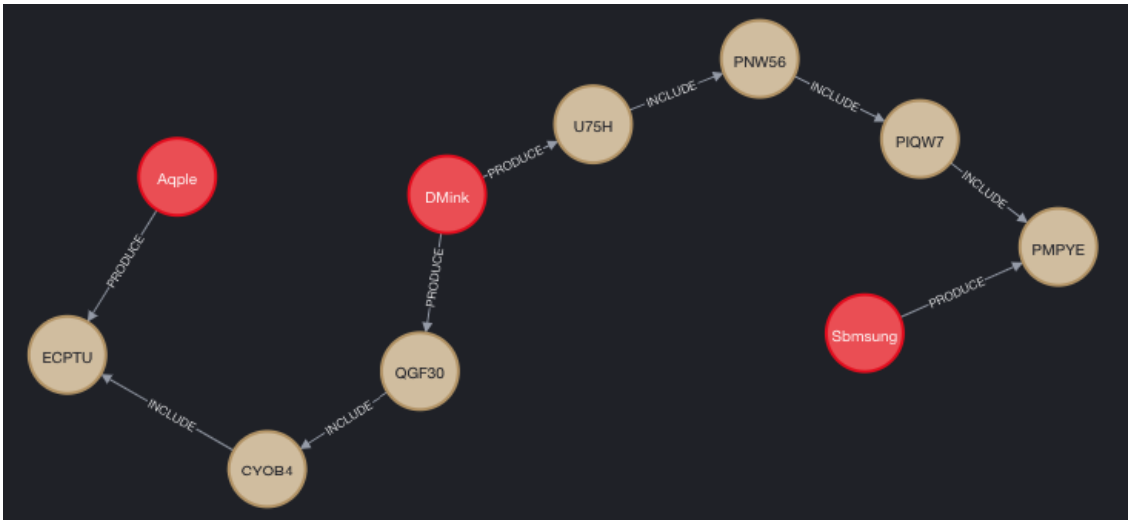
	p2.PID	p2.Unit_price
1	66	1
2	137	9
3	59	5
4	128	10
5	188	5

부속품이 가장 많은 제품에 대해서 해당 부속품에 관한 정보



부속품이 가장 많은 제품에 대해서 부속품의 제조사 및 품목 정보

```
// Aqple과 Sbmsung 사이에 존재하는 shortest path
match p=shortestPath(
    (qual:Company {Company: "Sbmsung"})-[*]-(sung:Company {Company: "Aqple"})
)
return p;
```

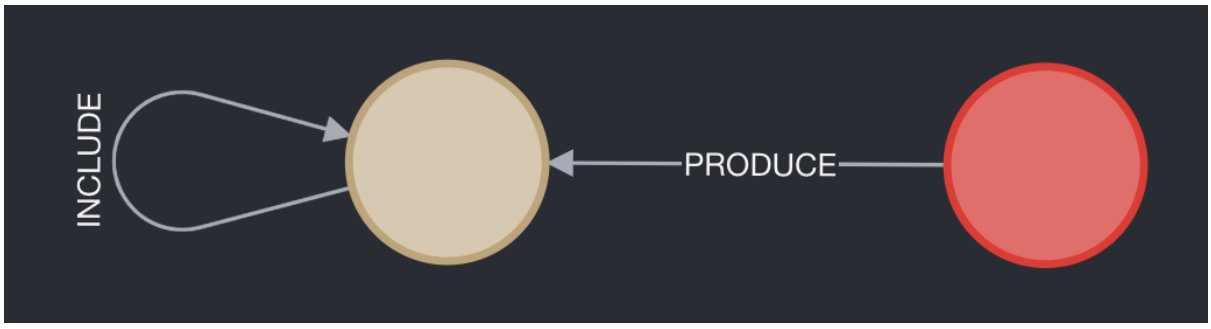


Aqple과 Sbmsung 사이에 존재하는 shortest path

- 이 경로를 통해서 두 회사 Aqple과 Sbmsung은 DMink라는 업체에 부속품을 납품하는 관계임을 알 수 있다. 즉, DMink가 제조하는 제품 QGF30과 U75H에 대해서 각각 Aqple의 ECPTU라는 제품과 Sbmsung의 PMPYE라는 제품이 부속품으로 들어가는 것을 확인할 수 있다.

D. Analysis

- 위에서 생성한 graph의 schema는 아래와 같은 형태를 띠고 있다.



- 여기서 붉은 색 node는 company를, 노란 색 node는 product를 나타낸다. Company와 product 사이에는 PRODUCE, 그리고 product와 product 사이에는 INCLUDE의 관계가 존재한다.
- 이처럼 `Neo4j`는 데이터 사이에 밀접한 관련이 있을 때, 그리고 저장해둔 데이터에 변화가 자주 발생할 때 사용하기 좋은 DBMS이다. 데이터와 데이터 사이의 직접적인 연관관계를 추출하여 보여주기 때문이다.
- 특히 `Neo4j` 공식 홈페이지에서는 마블 코믹스의 예시를 들고 있는데, 그와 같이 스토리(entity)가 순차적으로 이어지지 않아 데이터의 업데이트가 잦은 경우 Graph 기반의 DBMS가 강력하게 고려될 수 있다.

Pros and Cons of Neo4j over Postgres

- Pros

    - 데이터 사이의 연결성이 강한 데이터베이스의 경우 그 관계성을 잘 보여줄 수 있다. 특히 entity와 entity를 직접적으로 연결하기 때문에, JOIN 연산을 수행하는 데 있어 별도의 mapping table이 필요한 경우가 많은 RDBMS에 비해서 적은 storage 공간을 사용하고도 JOIN 작업을 수행할 수 있다.
    - 또한 데이터를 중심으로 관계를 설정하기 때문에 테이블 단위의 RDBMS와 달리 데이터가 저장되어 있는 구조(schema)로부터 자유로우며, 따라서 자주 바뀌는 데이터에 대해서 update 이후 재배열에 필요한 별도의 overhead가 발생하지 않는다.
- Cons
    - 개인적인 경험으로 Aggregate 및 Group by 연산을 수행하는 데에는 적절하지 않다는 생각이 들었다. 즉, 관계형 연산이 아닌 대수적 연산을 수행하는 데에는 RDBMS와 SQL을 사용하는 것이 더욱 효율적이라는 생각이 들었으며, 따라서 C번 문제의 마지막 query를 작성하는 부분에서 부속품이 가장 많은 제품을 찾는 작업에는 SQL을 사용했다.
    - 다만 관계를 잘 보여줄 수 있는 데이터 추출(retrieval) 방법에 집중함에 따라서 데이터 저장 자체에는 크게 신경쓰지 않기 때문에, 저장된 구조와 효율적인 저장이 중요한 데이터에 대해서는 사용이 적절하지 않을 수 있다.

Pros and Cons of Neo4j over Prolog

- Pros
    - `Prolog`에 비해서 시각화에 강점이 있다. 물론 Prolog를 사용하더라도 관계를 보여주는 그래프를 그릴 수 있다. 하지만 `Neo4j`를 이용하면 생성된 그래프에 대해서 노드의 위치 및 관계의 길이 재배치 등과 같은 interactive한 작업이 가능할 뿐 아니라, 보다 복잡한 그래프 형태를 그림으로써 데이터와 데이터 사이의 관계를 보다 효율적으로 보여줄 수 있다.
    - 또한 화살표 등을 사용함으로써 관계를 보여주는 코드가 직관적이다.
- Cons
    - `Prolog`는 유저에 의해 정의된 Fact, 그리고 사실들의 나열 및 논리적 관계로서 만들어지는 Rule을 통해 데이터를 처리하는 시스템이다. 따라서 특정 조건에 부합하는 데이터를 찾거나, 결과가 나오게 되는 과정, 또는 특정 entity 사이에 존재하는 경로 등을 추론하기에는 좋은 시스템이다.
    - 반면 `Neo4j`는 관계에 기반한 조건이 아닌 경우 조건에 부합하는 데이터를 찾기에는 어려우며, 관계 속성에 관한 논리 연산 및 추론과 관련된 지원이 부족하다. 또한 `Prolog`에 비해 프로그래밍 언어로서의 자유도가 높지 않고, 다른 프로그래밍 언어와 결합되었을 때 유의미한 결과를 만들어내는 경우가 많다.