

**GRAPH THEORY AND ITS APPLICATIONS**

**ASSIGNMENT NAME:**

**Flipkart Purchase Using Neo4j**

## **Flipkart Purchase USING NEO4J**

The aim of this assignment is to build a recommendation system which recommends Products to Customers based on the dataset using neo4j.

### **Introduction To The Dataset**

*Flipkart is one of India's leading e-commerce marketplaces. Flipkart has 100 million registered users and more than a million sellers on its electronic commerce platform. To ensure prompt delivery to its customers, the company has invested in setting up warehouses in 21 states.*

It has an enormous amount of data/Products that can be used to make an optimal model. We have used a raw data set of Flipkart. This data set contains 12 different features that can be used for recommending an app to a user.

The different columns in the csv file are \_id, \_labels, name, lastname, Reviews, Size, Installs, Type, Price, \_start, \_end, \_type, rating.

1. The column “\_id” provides the id of a particular row.
2. The column “\_labels” provides the label for the given name. It can be either :Product or :Customer or :Category.
3. The column “name” provides the name of either Product or Customer or Category.
4. The column “Review” provides the number of reviews for an app.
5. The column “Size” provides the size of an Product.
6. The column “Purchased” provides the total number of Purchases of a particular Product.
7. The column “Type” provides the type of app, which is either free or paid.
8. The column “Price” provides the price of the Product.
9. The column “\_start” is used for mapping of relationships, where \_start contains the id of the starting(source) node.
10. The column “\_type” provides the type of relationship, which can be either HAS\_Product or HAS\_CATEGORY
11. The column “\_end” is used for mapping of relationships, where \_end contains the id of the ending(destination) node.
12. The column “rating” is used along with a relationship where gives the rating a user has given for a particular app.

Products will be recommended to users by using both Content-Based Filtering and Collaborative Filtering .

### **Content-based filtering**

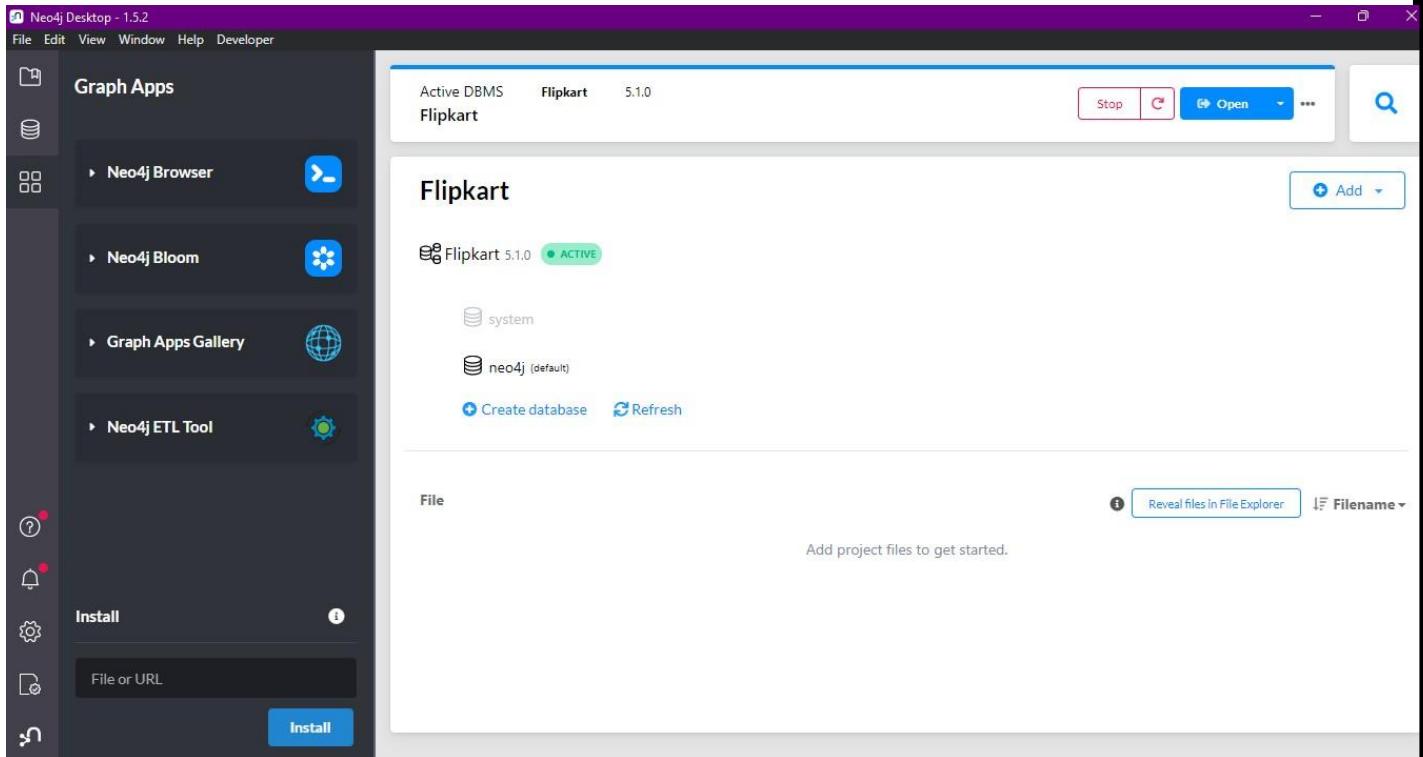
Content-based filtering uses item features to recommend other products similar to what the user likes, based on their previous actions or explicit feedback.

### **Collaborative filtering**

Collaborative filtering recommends a user the products on the basis of the preferences of the other users with similar tastes.

## Neo4j commands for constructing a graph based on the dataset Step1:

First create a project in Neo4j



Step2: Create nodes based on the dataset

### Cypher Query for Creating “Product” nodes

```
LOAD CSV FROM 'file:///Flipkart_Purchase.csv' AS row
```

```
WITH row, toInteger(row[0]) as id, row[1] as labels, row[2] as name, row[3] as Reviews, row[4] as Size, row[5] as Purchased, row[6] as Type, row[7] as Price, toInteger(row[8]) as start, toInteger(row[9]) as end, row[10] as type, toFloat(row[11]) as rating
```

```
WHERE labels=":Product"
```

```
CREATE (:Product{id:id, name: name, reviews: Reviews, size: Size, purchased: Purchased, type: Type, price: Price})
```

A screenshot of the Neo4j Browser window. The top part shows the Cypher query being typed into the editor:

```
1 LOAD CSV FROM 'file:///Flipkart_Purchase.csv' AS row
2 WITH row, toInteger(row[0]) as id, row[1] as labels, row[2] as name, row[3] as Reviews, row[4] as Size,
row[5] as Purchased, row[6] as Type, row[7] as Price, toInteger(row[8]) as start, toInteger(row[9]) as
end, row[10] as type, toFloat(row[11]) as rating
3 WHERE labels=":Product"
4 CREATE (:Product{id:id, name: name, reviews: Reviews, size: Size, purchased: Purchased, type: Type, price:
Price})
```

The bottom part shows the results of the query execution:

```
neo4j$ MATCH (n) RETURN n
Table
Completed after 26 ms.
```

The status bar at the bottom indicates the system is at 24°C, Partly sunny, and the date/time is 06-11-2022 11:05.

## Output:

```
neo4j@bolt://localhost:7687/neo4j - Neo4j Browser
File Edit View Window Help Developer
```

```
neo4j$ LOAD CSV FROM 'file:///Flipkart_Purchase.csv' AS row WITH row, toInteger(row[0]) as id, row[1] as label...
```

Added 65 labels, created 65 nodes, set 455 properties, completed after 36 ms.

```
neo4j$ MATCH (n) RETURN n
```

(no changes, no records)

```
neo4j$ LOAD CSV FROM 'file:///Flipkart_Purchase.csv' AS row WITH row, toInteger(row[0]) as id, row[1] as label...
```

Added 65 labels, created 65 nodes, set 455 properties, completed after 36 ms.

24°C Partly sunny 11:06 ENG IN 06-11-2022

## Output graph

```
neo4j@bolt://localhost:7687/neo4j - Neo4j Browser
File Edit View Window Help Developer
```

```
neo4j$
```

```
neo4j$ MATCH (n:Product) RETURN n LIMIT 100
```

Graph

Text

Code

Overview

Node labels

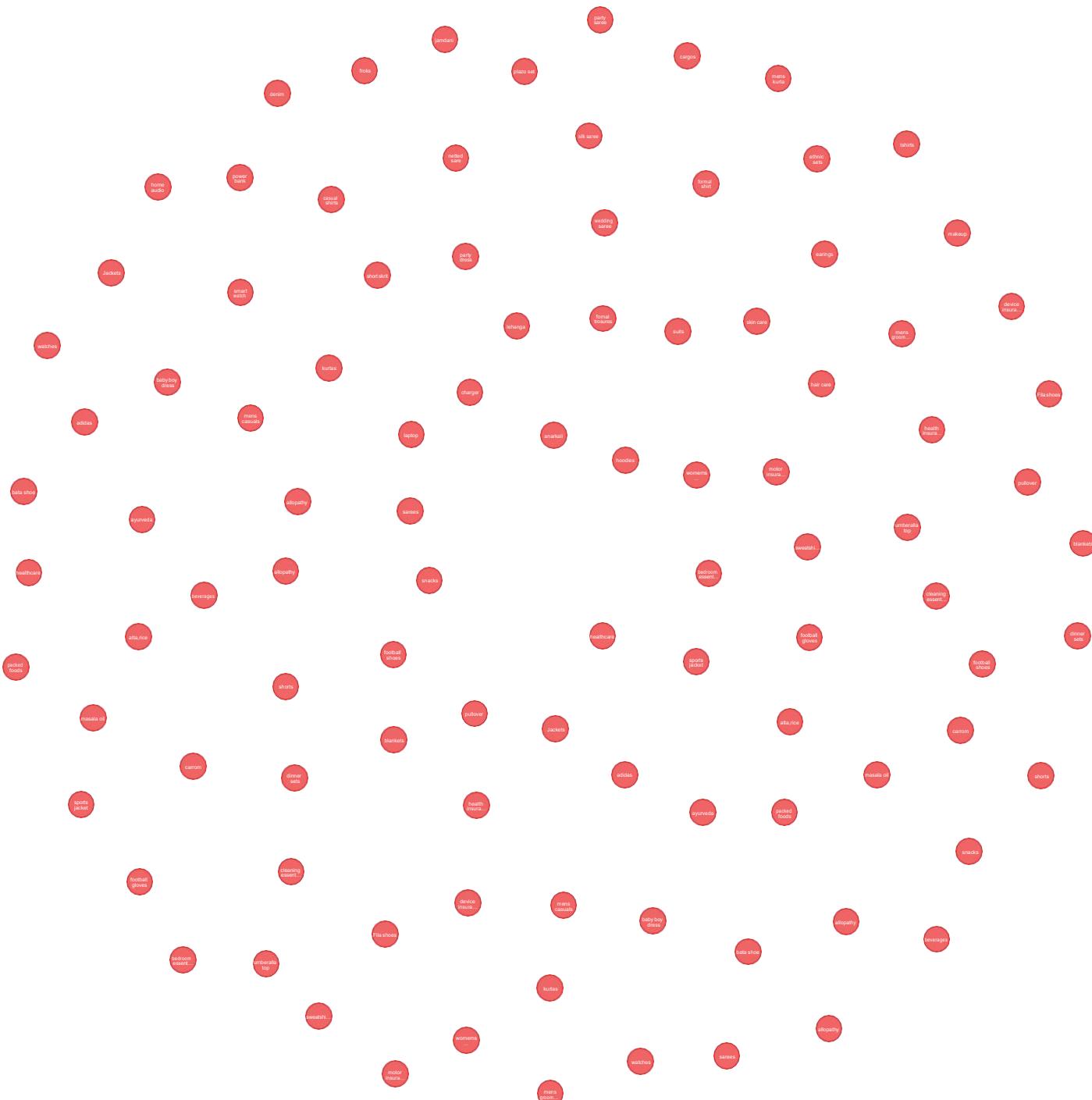
(100) Product (100)

Displaying 100 nodes, 0 relationships.

```
neo4j$ LOAD CSV FROM 'file:///Flipkart_Purchase.csv' AS row WITH row, toInteger(row[0]) as id, row[1] as label...
```

24°C Partly sunny 11:08 ENG IN 06-11-2022

## Exported graph:



# Cypher Query for Creating User Nodes

```
LOAD CSV FROM 'file:///Flipkart_Purchase.csv' AS row
WITH row, toInteger(row[0]) as id, row[1] as labels, row[2] as name, row[3] as Reviews, row[4] as Size, row[5] as Purchased, row[6] as Type, row[7] as Price, toInteger(row[8]) as start, toInteger(row[9]) as end, row[10] as type, toFloat(row[11]) as rating
WHERE labels="::Customer"
CREATE (:Customer{id:id, name: name, Reviews: Reviews})
```

The screenshot shows the Neo4j Browser interface. The top window displays the Cypher query:

```
1 LOAD CSV FROM 'file:///Flipkart_Purchase.csv' AS row
2 WITH row, toInteger(row[0]) as id, row[1] as labels, row[2] as name, row[3] as Reviews, row[4] as Size,
  row[5] as Purchased, row[6] as Type, row[7] as Price, toInteger(row[8]) as start, toInteger(row[9]) as
  end, row[10] as type, toFloat(row[11]) as rating
3 WHERE labels="::Customer"
4 CREATE (:Customer{id:id, name: name, Reviews: Reviews})
```

The bottom window shows the results of the query execution:

```
neo4j$ LOAD CSV FROM 'file:///Flipkart_Purchase.csv' AS row WITH row, toInteger(row[0]) as id, row[1] as label...
(no changes, no records)
```

A message at the bottom indicates the operation completed after 3 ms.

## OUTPUT:

The screenshot shows the Neo4j Browser interface. The top window displays the Cypher query:

```
neo4j$ LOAD CSV FROM 'file:///Flipkart_Purchase.csv' AS row WITH row, toInteger(row[0]) as id, row[1] as label...
```

The middle window shows the results of the query execution:

```
Added 40 labels, created 40 nodes, set 80 properties, completed after 17 ms.
```

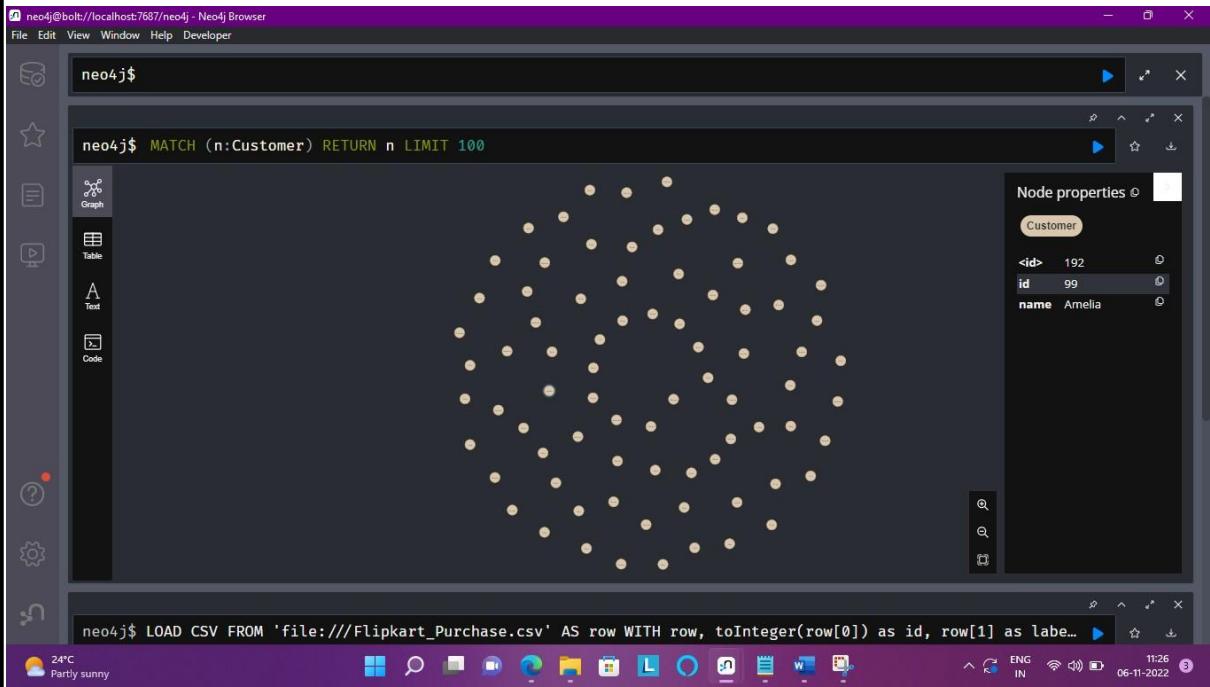
A message at the bottom indicates the operation completed after 17 ms.

The bottom window shows the results of the query execution again:

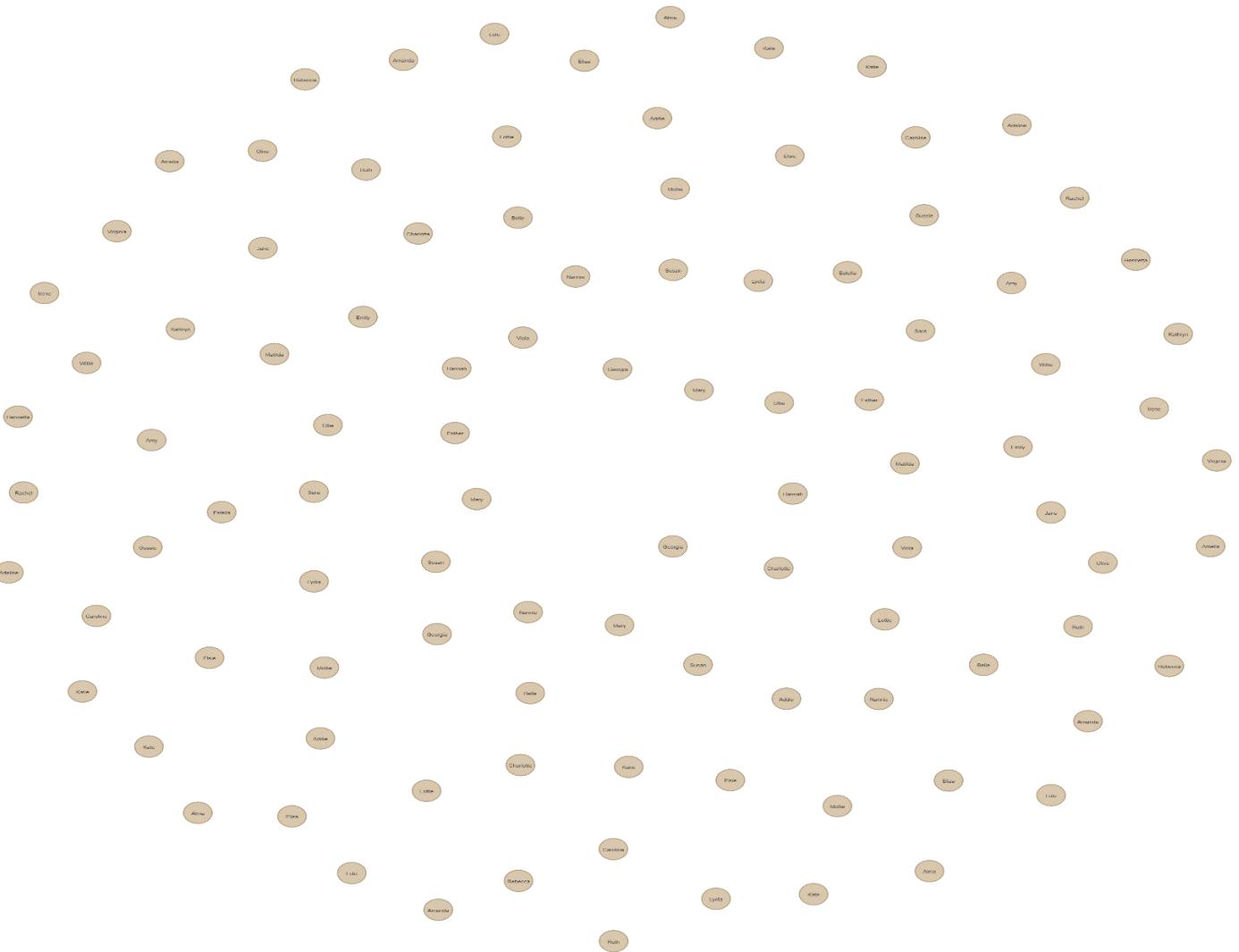
```
neo4j$ LOAD CSV FROM 'file:///Flipkart_Purchase.csv' AS row WITH row, toInteger(row[0]) as id, row[1] as label...
(no changes, no records)
```

A message at the bottom indicates the operation completed after 17 ms.

## Output graph:



## Exported graph:



## Cypher Query for Creating “Category” Nodes

```
LOAD CSV FROM 'file:///Flipkart_Purchase.csv' AS row
```

```
WITH row, toInteger(row[0]) as id, row[1] as labels, row[2] as name, row[3] as Reviews, row[4] as Size, row[5] as Purchased, row[6] as Type, row[7] as Price, toInteger(row[8]) as start, toInteger(row[9]) as end, row[10] as type, toFloat(row[11]) as rating
```

```
WHERE labels=":Category"
```

```
CREATE (:Category{id:id, name: name})
```

The screenshot shows the Neo4j Browser interface. The top panel contains the Cypher code for creating category nodes from a CSV file. The bottom panel shows the results of the query execution, indicating "(no changes, no records)" and "Completed after 3 ms." The system tray at the bottom shows a weather icon for 24°C and partly sunny, along with other system icons.

```
1 LOAD CSV FROM 'file:///Flipkart_Purchase.csv' AS row
2 WITH row, toInteger(row[0]) as id, row[1] as labels, row[2] as name, row[3] as Reviews, row[4] as Size,
row[5] as Purchased, row[6] as Type, row[7] as Price, toInteger(row[8]) as start, toInteger(row[9]) as
end, row[10] as type, toFloat(row[11]) as rating
3 WHERE labels=":Category"
4 CREATE (:Category{id:id, name: name})
5
```

```
neo4j$ LOAD CSV FROM 'file:///Flipkart_Purchase.csv' AS row WITH row, toInteger(row[0]) as id, row[1] as label...
(no changes, no records)
```

Completed after 3 ms.

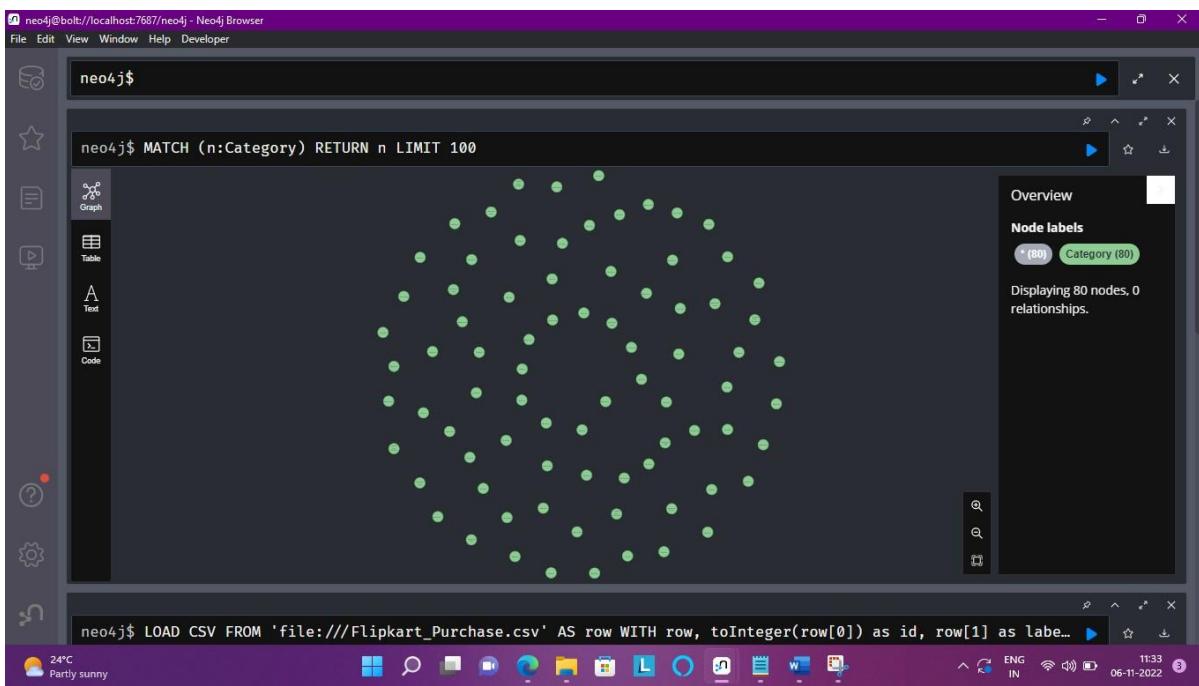
## OUTPUT:

The screenshot shows the Neo4j Browser interface. The top panel shows the executed Cypher command and its completion message: "Added 20 labels, created 20 nodes, set 40 properties, completed after 13 ms.". The bottom panel shows the results of the query execution, indicating "(no changes, no records)". The system tray at the bottom shows a weather icon for 24°C and partly sunny, along with other system icons.

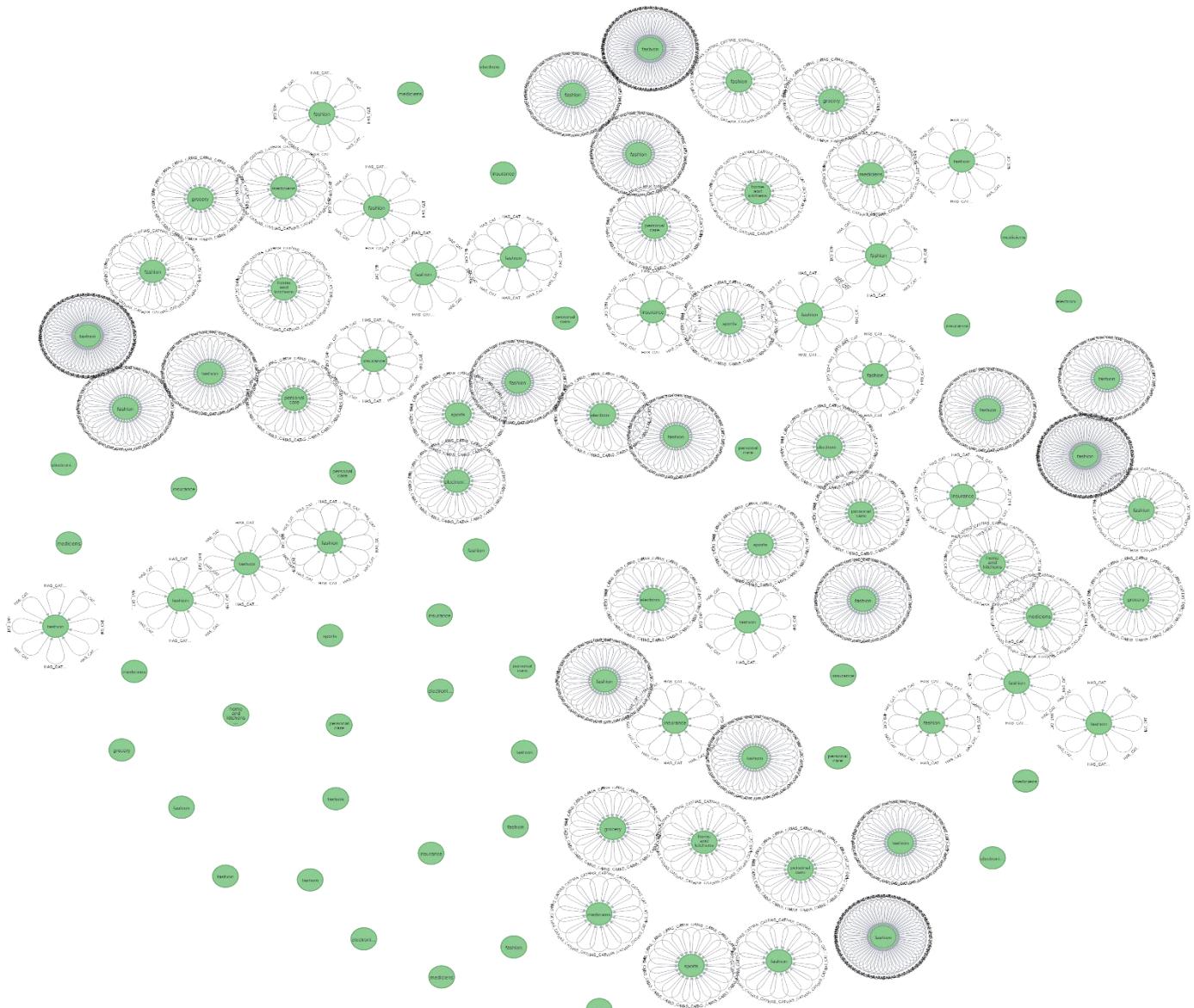
```
neo4j$ LOAD CSV FROM 'file:///Flipkart_Purchase.csv' AS row WITH row, toInteger(row[0]) as id, row[1] as label...
Added 20 labels, created 20 nodes, set 40 properties, completed after 13 ms.
```

```
neo4j$ LOAD CSV FROM 'file:///Flipkart_Purchase.csv' AS row WITH row, toInteger(row[0]) as id, row[1] as label...
(no changes, no records)
```

## OUTPUT GRAPH:



## Exported graph:



### Step3: Create realtionships based on the dataset

#### Cypher Query for Creating relationship as “HAS\_Product”

```
LOAD CSV FROM 'file:///Flipkart_Purchase.csv' AS row  
WITH row, toInteger(row[0]) as id, row[1] as labels, row[2] as name, row[3] as Reviews, row[4] as Size, row[5] as Purchased, row[6] as Type, row[7] as Price, toInteger(row[8]) as start, toInteger(row[9]) as end, row[10] as type, toFloat(row[11]) as rating  
WHERE type="HAS_Product"  
MATCH (c:Customer) WHERE c.id= start  
MATCH (p:Product) WHERE p.id= end  
CREATE (p)-[:HAS_Product{rating: rating}]->(a)
```

The screenshot shows the Neo4j Browser application window. On the left, there's a sidebar with icons for Project Files, Cypher files, and other tools. The main area has two panes: one for the Cypher query and another for the terminal output.

**Cypher Query:**

```
1 LOAD CSV FROM 'file:///Flipkart_Purchase.csv' AS row  
2 WITH row, toInteger(row[0]) as id, row[1] as labels, row[2] as name, row[3] as Reviews, row[4] as Size, row[5] as Purchased, row[6] as Type, row[7] as Price, toInteger(row[8]) as start, toInteger(row[9]) as end, row[10] as type, toFloat(row[11]) as rating  
3 WHERE type="HAS_Product"  
4 MATCH (c:Customer) WHERE c.id= start  
5 MATCH (p:Product) WHERE p.id= end  
6 CREATE (p)-[:HAS_Product{rating: rating}]->(a)  
7  
8
```

**Terminal Output:**

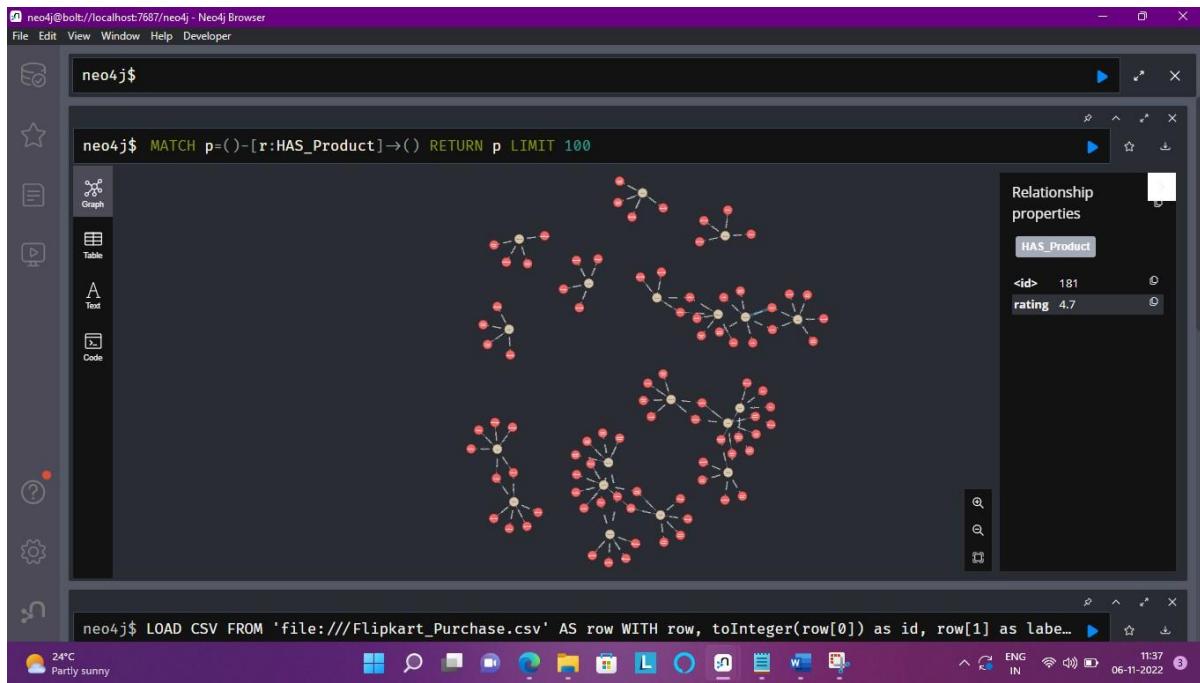
```
neo4j$ LOAD CSV FROM 'file:///Flipkart_Purchase.csv' AS row WITH row, toInteger...  
Created 384 nodes, set 384 properties, created 384 relationships, completed after 25 ms.  
Table  
Code  
Created 384 nodes, set 384 properties, created 384 relationships, completed after 25 ms.
```

The system tray at the bottom shows the date (06-11-2022), time (12:26), and battery level (ENG IN).

## OUTPUT:

```
neo4j$ LOAD CSV FROM 'file:///Flipkart_Purchase.csv' AS row WITH row, toInteger(row[0]) as id, row[1] as label...  
Set 384 properties, created 384 relationships, completed after 283 ms.  
  
neo4j$ LOAD CSV FROM 'file:///Flipkart_Purchase.csv' AS row WITH row, toInteger(row[0]) as id, row[1] as label...  
Added 20 labels, created 20 nodes, set 40 properties, completed after 6 ms.
```

## OUTPUT GRAPH:





## Cypher Query for Creating relationship as “HAS\_CATEGORY”

```
LOAD CSV FROM 'file:///Flipkart_Purchase.csv' AS row
WITH row, toInteger(row[0]) as id, row[1] as labels, row[2] as name, row[3] as Reviews, row[4] as Size, row[5] as Purchased, row[6] as Type, row[7] as Price, toInteger(row[8]) as start, toInteger(row[9]) as end, row[10] as type,
toFloat(row[11]) as rating
WHERE type="HAS_CATEGORY"
MATCH (c:Category) WHERE c.id= end
MATCH (p:Product) WHERE p.id= start
CREATE (c)-[:HAS_CATEGORY]->(c)
```

The screenshot shows the Neo4j Browser interface. On the left, there's a sidebar with icons for Project Files, Cypher files, and other developer tools. The main area has two tabs: 'Table' and 'Code'. The 'Code' tab contains the Cypher query provided above. The 'Table' tab shows the results of the query execution, which completed in 36 ms and created 656 relationships. The system tray at the bottom indicates it's 26°C and cloudy, and the date is 06-11-2022.

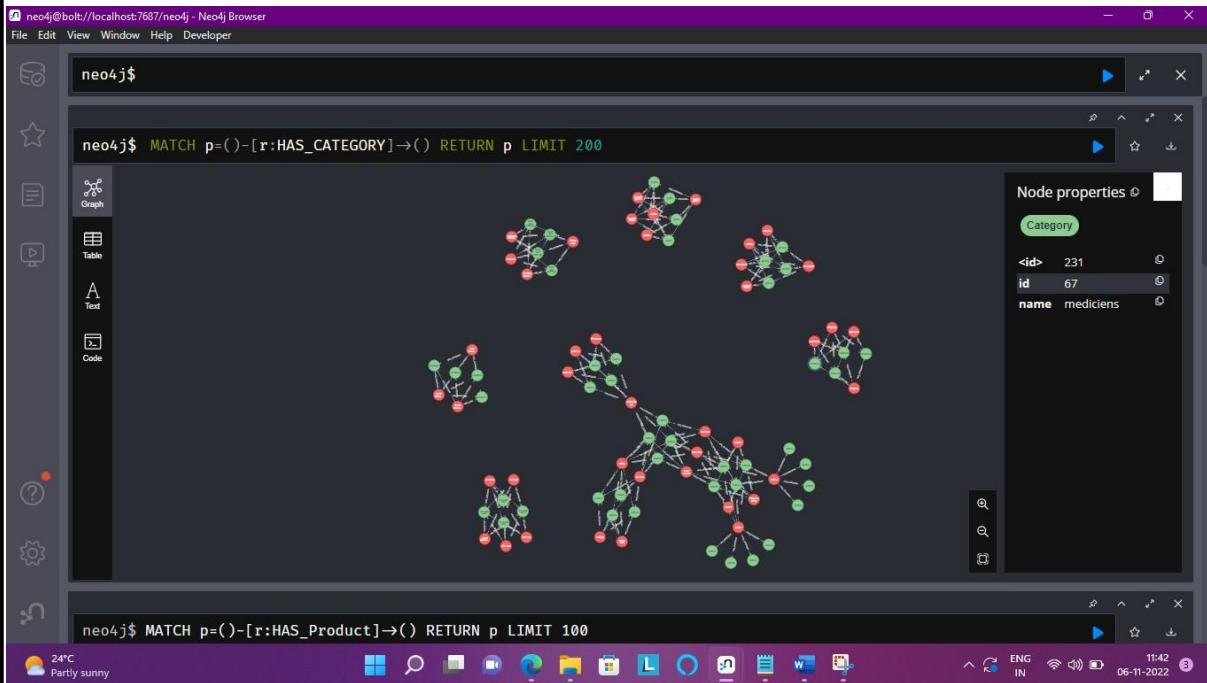
```
1 LOAD CSV FROM 'file:///Flipkart_Purchase.csv' AS row
2 WITH row, toInteger(row[0]) as id, row[1] as labels, row[2] as name, row[3] as Reviews, row[4] as Size, row[5] as Purchased, row[6] as Type, row[7] as Price, toInteger(row[8]) as start, toInteger(row[9]) as end, row[10] as type,
3 toFloat(row[11]) as rating
4 WHERE type="HAS_CATEGORY"
5 MATCH (c:Category) WHERE c.id= end
6 MATCH (p:Product) WHERE p.id= start
7 CREATE (c)-[:HAS_CATEGORY]->(c)

neo4j$ LOAD CSV FROM 'file:///Flipkart_Purchase.csv' AS row WITH row, toInteger... →
Created 656 relationships, completed after 36 ms.
```

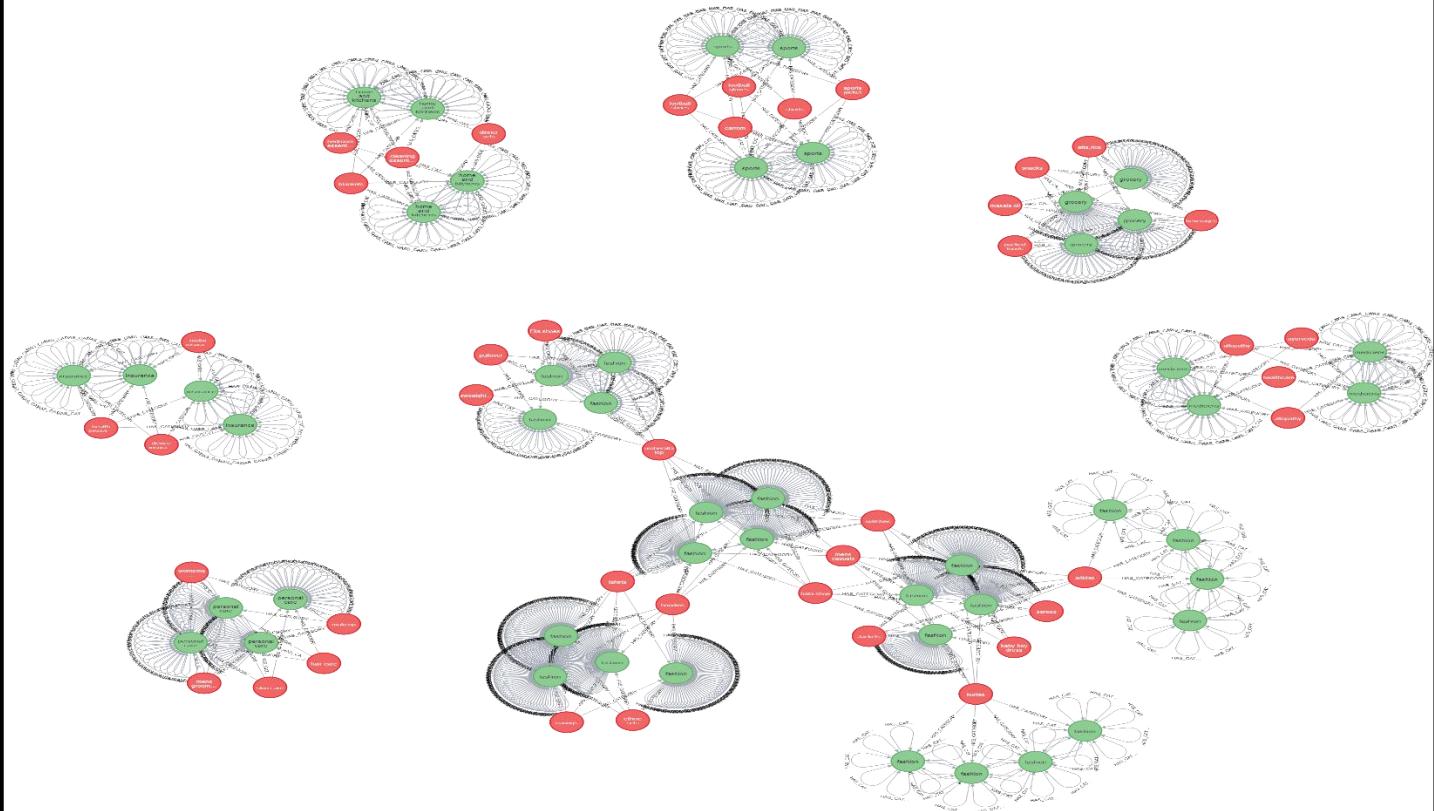
This screenshot shows another instance of the Neo4j Browser. It has a similar layout with a sidebar and tabs for 'Table' and 'Code'. The 'Code' tab contains the same Cypher query. The 'Table' tab shows the results of the query execution, which completed in 34 ms and created 656 relationships. The system tray at the bottom indicates it's 26°C and cloudy, and the date is 06-11-2022.

```
neo4j$ LOAD CSV FROM 'file:///Flipkart_Purchase.csv' AS row WITH row, toInteger(row[0]) as id, row[1] as labe... →
Created 656 relationships, completed after 34 ms.
```

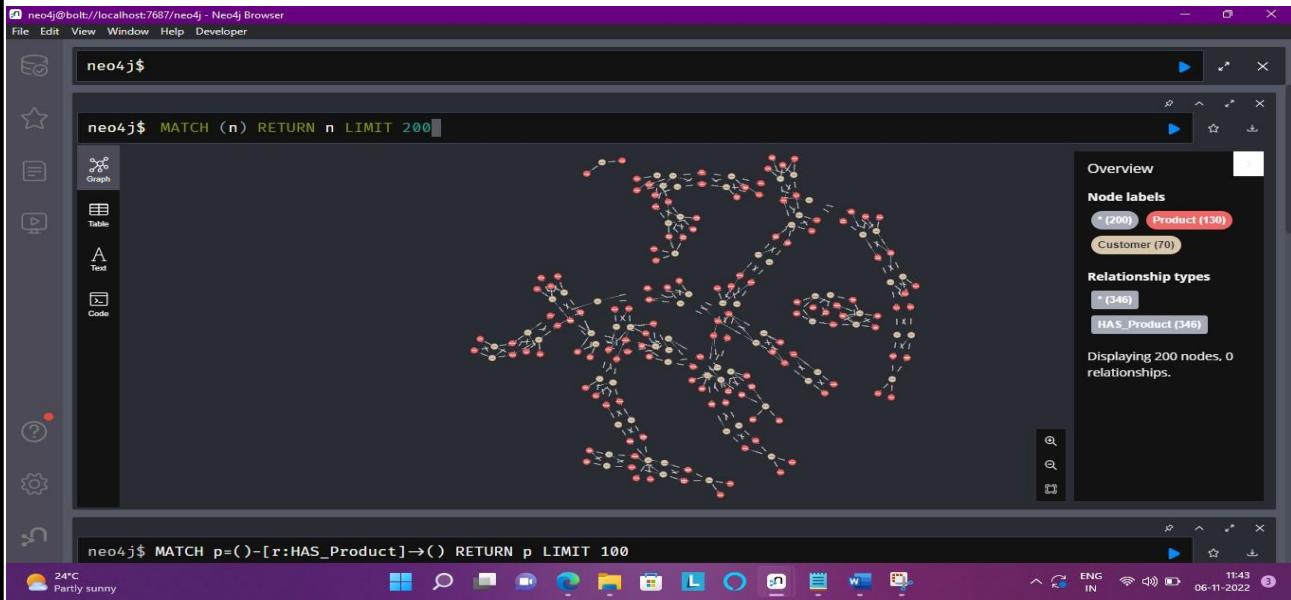
## OUTPUT GRAPH:



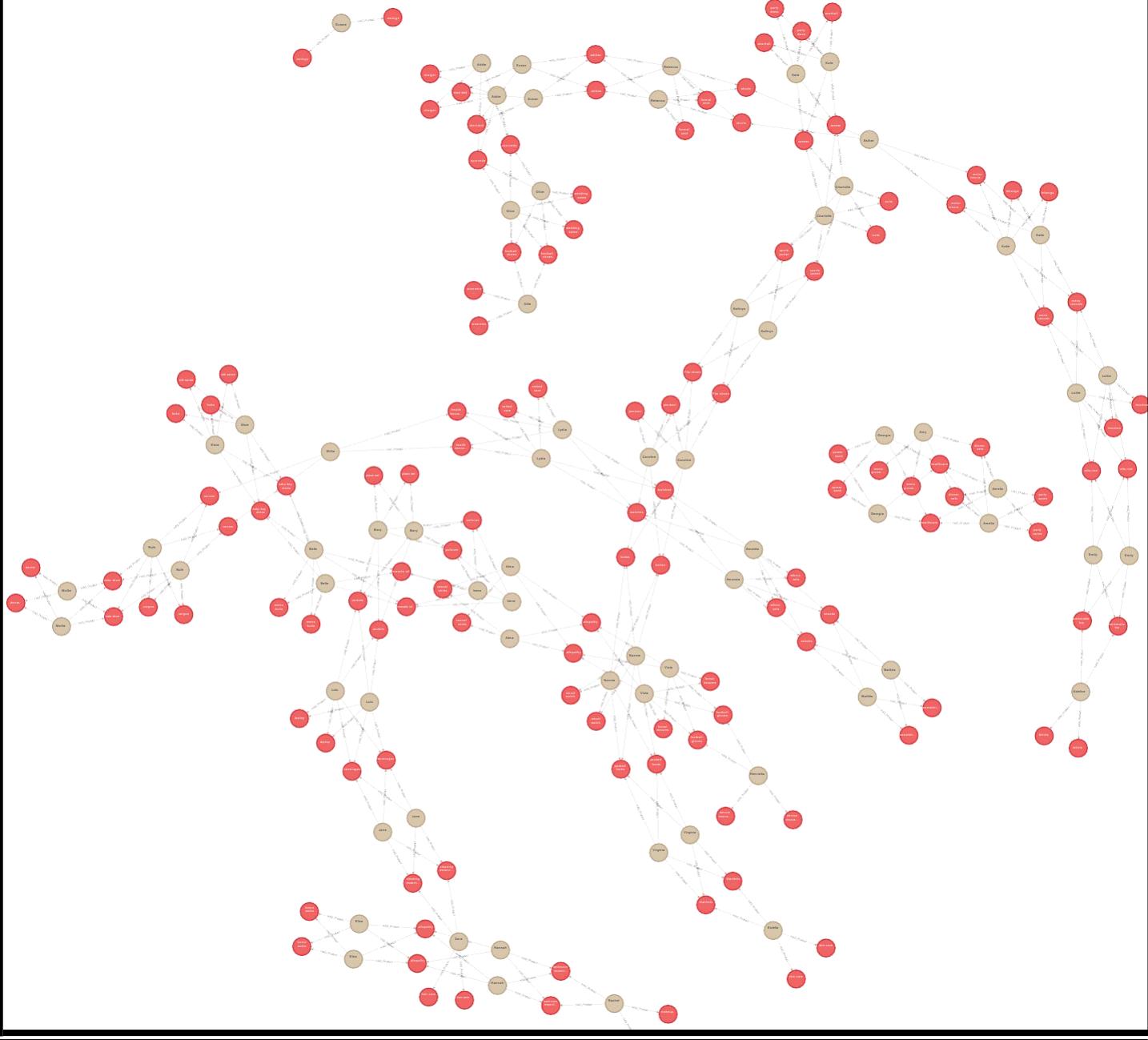
## EXPORTED GRAPH:



## Entire Graph screenshot:



Exported entire graph after constructing all nodes and relationships



## Content- Based Filtering

### Cypher query:

```
MATCH(u:Customer{name:"Belle", lastname:"Iona"})-[:HAS_Product]->(a:Product)-[:HAS_CATEGORY]->(c:Category)<-[:HAS_CATEGORY]->(z:Product)
```

```
WHERE NOT EXISTS ((u)-[:HAS_Product]->(z))
```

```
WITH a, z, COUNT(c) AS intersection
```

```
MATCH (a)-[:HAS_CATEGORY]->(ac:Category)
```

```
WITH a, z, intersection, COLLECT(ac.name) AS a1
```

```
MATCH (z)-[:HAS_CATEGORY]->(zc:Category)
```

```
WITH a, z, a1, intersection, COLLECT(zc.name) AS a2
```

```
WITH a, z, intersection, a1+[x IN a2 WHERE NOT x IN a1] AS union, a1, a2
```

```
RETURN a.name as CustomerProduct, z.name as RecommendedProduct, a1 as CustomerProductCategory, a2 as RecommendedProductCategory, ((1.0*intersection)/SIZE(union)) AS jaccard ORDER BY jaccard DESC
```

### Output in table form:

|   | CustomerProduct  | RecommendedProduct | CustomerProductCategory                      | RecommendedProductCategory   | jaccard |
|---|------------------|--------------------|--|--|---------|
| 1 | "baby boy dress" | "bata shoe"        | ["fashion", "fashion", "fashion", "fashion"] | ["fashion", "fashion", "fashion", "fashion", "fashion", "fashion", "fashion", "fashion"] | 2.0     |
| 2 | "baby boy dress" | "sarees"           | ["fashion", "fashion", "fashion", "fashion"] | ["fashion", "fashion", "fashion", "fashion"]   | 2.0     |
| 3 | "baby boy dress" | "sarees"           | ["fashion", "fashion", "fashion", "fashion"] | ["fashion", "fashion", "fashion", "fashion"]   | 2.0     |
| 4 | "baby boy dress" | "bata shoe"        | ["fashion", "fashion", "fashion", "fashion"] | ["fashion", "fashion", "fashion", "fashion", "fashion", "fashion", "fashion", "fashion"] | 2.0     |
| 5 | "baby boy dress" | "watches"          | ["fashion", "fashion", "fashion", "fashion"] | ["fashion", "fashion", "fashion", "fashion", "fashion", "fashion", "fashion", "fashion"] | 2.0     |
| 6 | "baby boy dress" | "adidas"           | ["fashion", "fashion", "fashion", "fashion"] | ["fashion", "fashion", "fashion", "fashion", "fashion", "fashion", "fashion", "fashion"] | 2.0     |

Started streaming 96 records after 25 ms and completed after 54 ms.

neo4j@bolt://localhost:7687/neo4j - Neo4j Browser

File Edit View Window Help Developer

neo4j\$ MATCH(u:Customer{name:"Belle"})-[:HAS\_Product]→(a:Product)-[:HAS\_CATEGORY]→(c:Category)←[:HAS\_CATEGORY]-(d:Category)

|    | CustomerProduct  | RecommendedProduct | CustomerProductCategory                      | RecommendedProductCategory  | jac |
|----|------------------|--------------------|--|---|-----|
| 7  | "baby boy dress" | "Jackets"          | ["fashion", "fashion", "fashion", "fashion"] | ["fashion", "fashion", "fashion", "fashion"]                                  | 2.0 |
| 8  | "baby boy dress" | "mens casuals"     | ["fashion", "fashion", "fashion", "fashion"] | ["fashion", "fashion", "fashion", "fashion", "fashion", "fashion", "fashion"] | 2.0 |
| 9  | "baby boy dress" | "adidas"           | ["fashion", "fashion", "fashion", "fashion"] | ["fashion", "fashion", "fashion", "fashion", "fashion", "fashion", "fashion"] | 2.0 |
| 10 | "baby boy dress" | "Jackets"          | ["fashion", "fashion", "fashion", "fashion"] | ["fashion", "fashion", "fashion", "fashion"]                                  | 2.0 |
| 11 | "baby boy dress" | "kurtas"           | ["fashion", "fashion", "fashion", "fashion"] | ["fashion", "fashion", "fashion", "fashion", "fashion", "fashion", "fashion"] | 2.0 |
| 12 | "baby boy dress" | "mens casuals"     | ["fashion", "fashion", "fashion", "fashion"] | ["fashion", "fashion", "fashion", "fashion", "fashion", "fashion", "fashion"] | 2.0 |

Started streaming 96 records after 25 ms and completed after 54 ms.

24°C Partly sunny 11:48 ENG IN 06-11-2022

neo4j@bolt://localhost:7687/neo4j - Neo4j Browser

File Edit View Window Help Developer

neo4j\$ MATCH(u:Customer{name:"Belle"})-[:HAS\_Product]→(a:Product)-[:HAS\_CATEGORY]→(c:Category)←[:HAS\_CATEGORY]-(d:Category)

|    | CustomerProduct  | RecommendedProduct | CustomerProductCategory                      | RecommendedProductCategory  | jac |
|----|------------------|--------------------|--|---|-----|
| 13 | "baby boy dress" | "watches"          | ["fashion", "fashion", "fashion", "fashion"] | ["fashion", "fashion", "fashion", "fashion", "fashion", "fashion", "fashion"] | 2.0 |
| 14 | "baby boy dress" | "kurtas"           | ["fashion", "fashion", "fashion", "fashion"] | ["fashion", "fashion", "fashion", "fashion", "fashion", "fashion", "fashion"] | 2.0 |
| 15 | "masala oil"     | "packed foods"     | ["grocery", "grocery", "grocery", "grocery"] | ["grocery", "grocery", "grocery", "grocery"]                                  | 2.0 |
| 16 | "masala oil"     | "snacks"           | ["grocery", "grocery", "grocery", "grocery"] | ["grocery", "grocery", "grocery", "grocery"]                                  | 2.0 |
| 17 | "masala oil"     | "packed foods"     | ["grocery", "grocery", "grocery", "grocery"] | ["grocery", "grocery", "grocery", "grocery"]                                  | 2.0 |
| 18 | "masala oil"     | "beverages"        | ["grocery", "grocery", "grocery", "grocery"] | ["grocery", "grocery", "grocery", "grocery"]                                  | 2.0 |

Started streaming 96 records after 25 ms and completed after 54 ms.

24°C Partly sunny 11:49 ENG IN 06-11-2022

neo4j@bolt://localhost:7687/neo4j - Neo4j Browser

File Edit View Window Help Developer

neo4j\$ MATCH(u:Customer{name:"Belle"})-[:HAS\_Product]→(a:Product)-[:HAS\_CATEGORY]→(c:Category)←[:HAS\_CATEGORY]→(d:Category)-[:HAS\_Product]→(e:Product)-[:HAS\_PRODUCT]→(f:Product)

|    | CustomerProduct | RecommendedProduct | CustomerProductCategory                      | RecommendedProductCategory                   | jac |
|----|-----------------|--------------------|--|--|-----|
| 19 | "masala oil"    | "snacks"           | ["grocery", "grocery", "grocery", "grocery"] | ["grocery", "grocery", "grocery", "grocery"] | 2.0 |
| 20 | "masala oil"    | "atta,rice"        | ["grocery", "grocery", "grocery", "grocery"] | ["grocery", "grocery", "grocery", "grocery"] | 2.0 |
| 21 | "masala oil"    | "beverages"        | ["grocery", "grocery", "grocery", "grocery"] | ["grocery", "grocery", "grocery", "grocery"] | 2.0 |
| 22 | "masala oil"    | "atta,rice"        | ["grocery", "grocery", "grocery", "grocery"] | ["grocery", "grocery", "grocery", "grocery"] | 2.0 |
| 23 | "mens kurta"    | "earings"          | ["fashion", "fashion", "fashion", "fashion"] | ["fashion", "fashion", "fashion", "fashion"] | 2.0 |
| 24 | "mens kurta"    | "ethnic sets"      | ["fashion", "fashion", "fashion", "fashion"] | ["fashion", "fashion", "fashion", "fashion"] | 2.0 |

Started streaming 96 records after 25 ms and completed after 54 ms.

24°C Partly sunny 11:49 ENG IN 06-11-2022

neo4j@bolt://localhost:7687/neo4j - Neo4j Browser

File Edit View Window Help Developer

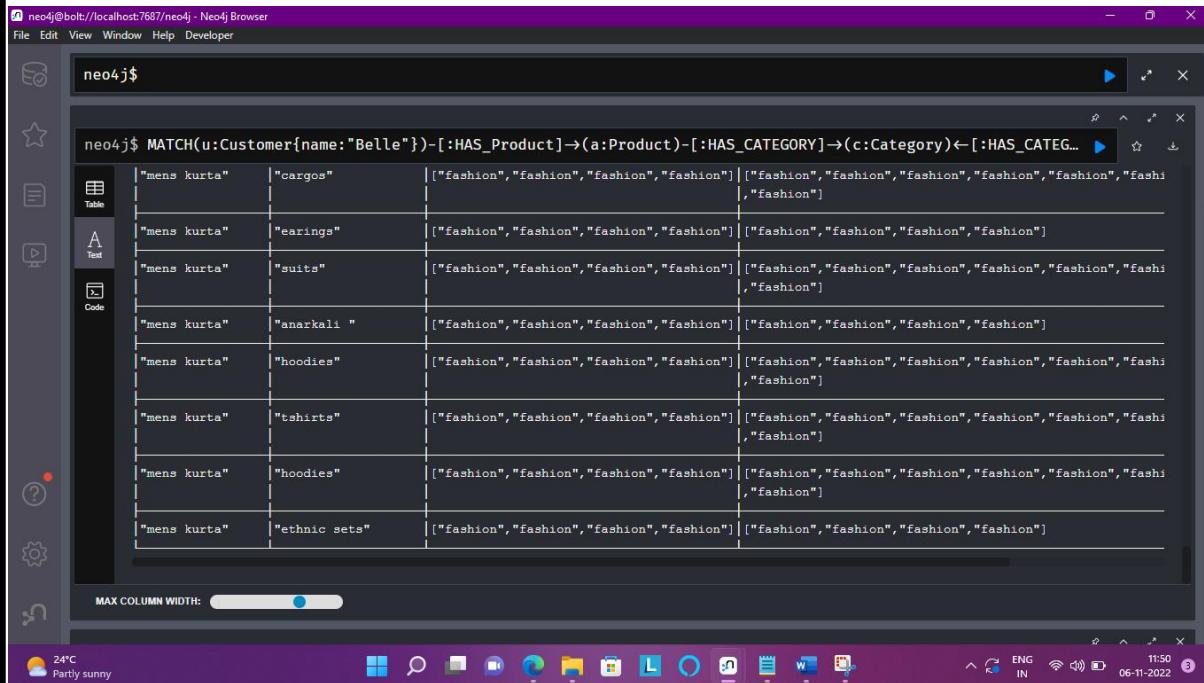
neo4j\$ MATCH(u:Customer{name:"Belle"})-[:HAS\_Product]→(a:Product)-[:HAS\_CATEGORY]→(c:Category)←[:HAS\_CATEGORY]→(d:Category)-[:HAS\_Product]→(e:Product)-[:HAS\_PRODUCT]→(f:Product)

|    | CustomerProduct | RecommendedProduct | CustomerProductCategory                      | RecommendedProductCategory  | jac |
|----|-----------------|--------------------|--|---|-----|
| 30 | "mens kurta"    | "earings"          | ["fashion", "fashion", "fashion", "fashion"] | ["fashion", "fashion", "fashion", "fashion"]                                  | 2.0 |
| 31 | "mens kurta"    | "suits"            | ["fashion", "fashion", "fashion", "fashion"] | ["fashion", "fashion", "fashion", "fashion", "fashion", "fashion", "fashion"] | 2.0 |
| 32 | "mens kurta"    | "anarkali "        | ["fashion", "fashion", "fashion", "fashion"] | ["fashion", "fashion", "fashion", "fashion"]                                  | 2.0 |
| 33 | "mens kurta"    | "hoodies"          | ["fashion", "fashion", "fashion", "fashion"] | ["fashion", "fashion", "fashion", "fashion", "fashion", "fashion", "fashion"] | 2.0 |
| 34 | "mens kurta"    | "tshirts"          | ["fashion", "fashion", "fashion", "fashion"] | ["fashion", "fashion", "fashion", "fashion", "fashion", "fashion", "fashion"] | 2.0 |
| 35 | "mens kurta"    | "hoodies"          | ["fashion", "fashion", "fashion", "fashion"] | ["fashion", "fashion", "fashion", "fashion", "fashion", "fashion", "fashion"] | 2.0 |

Started streaming 96 records after 25 ms and completed after 54 ms.

24°C Partly sunny 11:49 ENG IN 06-11-2022

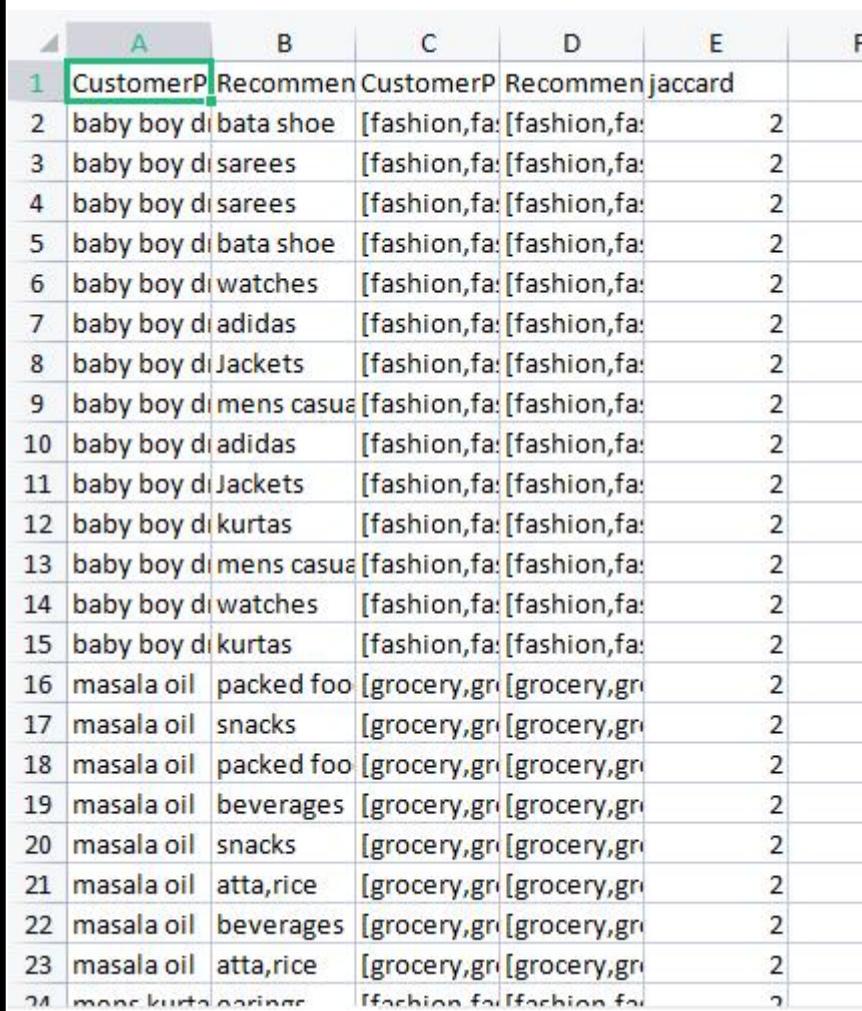
## Output represented in the form of text



The screenshot shows the Neo4j Browser interface with a query results table. The table has three columns: Product Name, Category, and Description. The products listed are "mens kurta", "earings", "suits", "anarkali ", "hoodies", "tshirts", "hoodies", and "ethnic sets". Each product is associated with multiple categories and descriptions.

| Product      | Category      | Description   |
|--------------|---------------|---|
| "mens kurta" | "cargos"      | ["fashion", "fashion", "fashion", "fashion"]   ["fashion", "fashion", "fashion", "fashion", "fashion", "fashion"] |
| "mens kurta" | "earings"     | ["fashion", "fashion", "fashion", "fashion"]   ["fashion", "fashion", "fashion", "fashion"]                       |
| "mens kurta" | "suits"       | ["fashion", "fashion", "fashion", "fashion"]   ["fashion", "fashion", "fashion", "fashion", "fashion", "fashion"] |
| "mens kurta" | "anarkali "   | ["fashion", "fashion", "fashion", "fashion"]   ["fashion", "fashion", "fashion", "fashion"]                       |
| "mens kurta" | "hoodies"     | ["fashion", "fashion", "fashion", "fashion"]   ["fashion", "fashion", "fashion", "fashion", "fashion", "fashion"] |
| "mens kurta" | "tshirts"     | ["fashion", "fashion", "fashion", "fashion"]   ["fashion", "fashion", "fashion", "fashion", "fashion", "fashion"] |
| "mens kurta" | "hoodies"     | ["fashion", "fashion", "fashion", "fashion"]   ["fashion", "fashion", "fashion", "fashion", "fashion", "fashion"] |
| "mens kurta" | "ethnic sets" | ["fashion", "fashion", "fashion", "fashion"]   ["fashion", "fashion", "fashion", "fashion"]                       |

## Exported excel sheet output for content-based filtering:



The screenshot shows an Excel spreadsheet with columns A through F. The first row contains headers: "CustomerP", "Recommen", "CustomerP", "Recommen", and "jaccard". Rows 2 through 24 show various recommendations for different customers, such as "baby boy di bata shoe", "baby boy di sarees", etc., along with their corresponding product names and jaccard similarity scores.

| CustomerP | Recommen               | CustomerP    | Recommen     | jaccard |
|-----------|------------------------|--------------|--------------|---------|
| 2         | baby boy di bata shoe  | [fashion,fa: | [fashion,fa: | 2       |
| 3         | baby boy di sarees     | [fashion,fa: | [fashion,fa: | 2       |
| 4         | baby boy di sarees     | [fashion,fa: | [fashion,fa: | 2       |
| 5         | baby boy di bata shoe  | [fashion,fa: | [fashion,fa: | 2       |
| 6         | baby boy di watches    | [fashion,fa: | [fashion,fa: | 2       |
| 7         | baby boy di adidas     | [fashion,fa: | [fashion,fa: | 2       |
| 8         | baby boy di Jackets    | [fashion,fa: | [fashion,fa: | 2       |
| 9         | baby boy di mens casua | [fashion,fa: | [fashion,fa: | 2       |
| 10        | baby boy di adidas     | [fashion,fa: | [fashion,fa: | 2       |
| 11        | baby boy di Jackets    | [fashion,fa: | [fashion,fa: | 2       |
| 12        | baby boy di kurtas     | [fashion,fa: | [fashion,fa: | 2       |
| 13        | baby boy di mens casua | [fashion,fa: | [fashion,fa: | 2       |
| 14        | baby boy di watches    | [fashion,fa: | [fashion,fa: | 2       |
| 15        | baby boy di kurtas     | [fashion,fa: | [fashion,fa: | 2       |
| 16        | masala oil packed foo  | [grocery,gr  | [grocery,gr  | 2       |
| 17        | masala oil snacks      | [grocery,gr  | [grocery,gr  | 2       |
| 18        | masala oil packed foo  | [grocery,gr  | [grocery,gr  | 2       |
| 19        | masala oil beverages   | [grocery,gr  | [grocery,gr  | 2       |
| 20        | masala oil snacks      | [grocery,gr  | [grocery,gr  | 2       |
| 21        | masala oil atta,rice   | [grocery,gr  | [grocery,gr  | 2       |
| 22        | masala oil beverages   | [grocery,gr  | [grocery,gr  | 2       |
| 23        | masala oil atta,rice   | [grocery,gr  | [grocery,gr  | 2       |
| 24        | mens kurta earings     | [fashion,fa  | [fashion,fa  | 2       |

# Collaborative Filtering

## Cypher Query:

```
MATCH(u:Customer {name: "Amanda"})-[HAS_Product]->(a:Product)<-[HAS_Product]-(x:Customer)
MATCH(x)-[:]->(z:Product) WHERE NOT exists((u)-[HAS_Product]->(z))
RETURN u.name as CustomerName, a.name as CustomerProduct, x.name as OtherCustomersWithSameProduct, z.name
as OtherCustomersProduct, z.name as RecommendedProduct
```

## Output in table format:

The screenshot shows the Neo4j Browser interface with the following details:

- Top bar: neo4j@bolt://localhost:7687/neo4j - Neo4j Browser, File, Edit, View, Window, Help, Developer.
- Toolbar: Table, Text, Code.
- Query results table:

|   | CustomerName | CustomerProduct | OtherCustomersWithSameProduct | OtherCustomersProduct | RecommendedProduct |
|---|--------------|-----------------|-------------------------------|-----------------------|--------------------|
| 1 | "Amanda"     | "watches"       | "Amanda"                      | "snacks"              | "snacks"           |
| 2 | "Amanda"     | "watches"       | "Amanda"                      | "ethnic sets"         | "ethnic sets"      |
| 3 | "Amanda"     | "watches"       | "Amanda"                      | "snacks"              | "snacks"           |
| 4 | "Amanda"     | "watches"       | "Amanda"                      | "watches"             | "watches"          |
| 5 | "Amanda"     | "watches"       | "Amanda"                      | "ethnic sets"         | "ethnic sets"      |
| 6 | "Amanda"     | "watches"       | "Lydia"                       | "health insurance"    | "health insurance" |
| 7 |              |                 |                               |                       |                    |
- Message at bottom: Started streaming 124 records after 12 ms and completed after 16 ms.
- Bottom bar: 24°C Partly sunny, system icons, ENG IN, 11:56, 06-11-2022.

The screenshot shows the Neo4j Browser interface with the following details:

- Top bar: neo4j@bolt://localhost:7687/neo4j - Neo4j Browser, File, Edit, View, Window, Help, Developer.
- Toolbar: Table, Text, Code.
- Query results table:

|     | CustomerName | CustomerProduct | OtherCustomersWithSameProduct | OtherCustomersProduct | RecommendedProduct |
|-----|--------------|-----------------|-------------------------------|-----------------------|--------------------|
| 119 | "Amanda"     | "watches"       | "Lydia"                       | "watches"             | "watches"          |
| 120 | "Amanda"     | "watches"       | "Lydia"                       | "netted sare"         | "netted sare"      |
| 121 | "Amanda"     | "watches"       | "Lydia"                       | "health insurance"    | "health insurance" |
| 122 | "Amanda"     | "watches"       | "Lydia"                       | "netted sare"         | "netted sare"      |
| 123 | "Amanda"     | "watches"       | "Lydia"                       | "health insurance"    | "health insurance" |
| 124 | "Amanda"     | "watches"       | "Lydia"                       | "watches"             | "watches"          |
|     |              |                 |                               |                       |                    |
- Message at bottom: Started streaming 124 records after 12 ms and completed after 16 ms.
- Bottom bar: 24°C Partly sunny, system icons, ENG IN, 11:56, 06-11-2022.

## Output in text format:

```
neo4j@bolt://localhost:7687/neo4j - Neo4j Browser
File Edit View Window Help Developer
neo4j$
```

neo4j\$ MATCH(u:Customer {name: "Amanda"})-[HAS\_Product]-(a:Product)-[:HAS\_Product]-(x:Customer) MATCH(x)-[:...]

| CustomerName | CustomerProduct | OtherCustomersWithSameProduct | OtherCustomersProduct | RecommendedProduct |
|--------------|-----------------|-------------------------------|-----------------------|--------------------|
| "Amanda"     | "watches"       | "Amanda"                      | "snacks"              | "snacks"           |
| "Amanda"     | "watches"       | "Amanda"                      | "ethnic sets"         | "ethnic sets"      |
| "Amanda"     | "watches"       | "Amanda"                      | "snacks"              | "snacks"           |
| "Amanda"     | "watches"       | "Amanda"                      | "watches"             | "watches"          |
| "Amanda"     | "watches"       | "Amanda"                      | "ethnic sets"         | "ethnic sets"      |
| "Amanda"     | "watches"       | "Lydia"                       | "health insurance"    | "health insurance" |
| "Amanda"     | "watches"       | "Lydia"                       | "health insurance"    | "health insurance" |
| "Amanda"     | "watches"       | "Lydia"                       | "netted sare"         | "netted sare"      |
| "Amanda"     | "watches"       | "Lydia"                       | "netted sare"         | "netted sare"      |
| "Amanda"     | "watches"       | "Lydia"                       | "watches"             | "watches"          |

MAX COLUMN WIDTH:

```
24°C Partly sunny 11:57 06-11-2022
```

```
neo4j@bolt://localhost:7687/neo4j - Neo4j Browser
File Edit View Window Help Developer
neo4j$
```

neo4j\$ MATCH(u:Customer {name: "Amanda"})-[HAS\_Product]-(a:Product)-[:HAS\_Product]-(x:Customer) MATCH(x)-[:...]

| Amanda   | watches   | Amanda  | ethnic sets        | ethnic sets        |
|----------|-----------|---------|--------------------|--------------------|
| "Amanda" | "watches" | "Lydia" | "health insurance" | "health insurance" |
| "Amanda" | "watches" | "Lydia" | "health insurance" | "health insurance" |
| "Amanda" | "watches" | "Lydia" | "netted sare"      | "netted sare"      |
| "Amanda" | "watches" | "Lydia" | "netted sare"      | "netted sare"      |
| "Amanda" | "watches" | "Lydia" | "watches"          | "watches"          |
| "Amanda" | "watches" | "Lydia" | "netted sare"      | "netted sare"      |
| "Amanda" | "watches" | "Lydia" | "health insurance" | "health insurance" |
| "Amanda" | "watches" | "Lydia" | "netted sare"      | "netted sare"      |
| "Amanda" | "watches" | "Lydia" | "health insurance" | "health insurance" |
| "Amanda" | "watches" | "Lydia" | "watches"          | "watches"          |

MAX COLUMN WIDTH:

```
24°C Partly sunny 11:58 06-11-2022
```

## Centrality Measure

In graph analytics, Centrality is a very important concept in identifying important nodes in a graph. It is used to measure the importance (or “centrality” as in how “central” a node is in the graph) of various nodes in a graph. Now, each node could be important from an angle depending on how “importance” is defined. Centrality comes in different flavors and each flavor or a metric defines importance of a node from a different perspective and further provides relevant analytical information about the graph and its nodes.

### Degree Centrality

In a **non-directed graph**, degree of a node is defined as the number of direct connections a node has with other nodes

In a **directed graph** (each edge has a direction), degree of a node is further divided into In-degree and Out-degree. In-degree refers to the number of edges/connections incident on it and Out-degree refers to the number of edges/connections from it to other nodes. Let's look at a sample Twitter graph below where nodes are individuals and edges with arrows indicate the “Follows” relationship:

### Closeness Centrality

Closeness centrality metric defines the importance of a node in a graph as being measured by how close it is to all other nodes in the graph. For a node, it is defined as the sum of the geodesic distance between that node to all other nodes in the network.

### Betweenness Centrality

Betweenness Centrality metric defines and measures the importance of a node in a network based upon how many times it occurs in the shortest path between all pairs of nodes in a graph.

### Eigen Vector Centrality

Eigen Vector Centrality metric measures the importance of a node in a graph as a function of the importance of its neighbors. If a node is connected to highly important nodes, it will have a higher EigenVector Centrality score as compared to a node which is connected to lesser important nodes.

According to our graph the nodes with highest centrality measures are “free” and “education”.

## **Analysis/ Inference**

By the constructed graph we can analyse the apps that a user like and then recommend it to them by considering the attributes

### **Analysis of the working of Content-based Filtering**

Based on the Cypher query executed previously

- In Content-based filtering the apps were recommended to a user based on the apps which the user has installed.
- This done by checking the category of the app installed and then matching that category with apps with similar category and then recommending it to the user.

### **Analysis of the working of Collaborative Filtering**

Based on the Cypher query executed previously

- In Collaborative filtering the apps were recommended to a user based on which other users have the same app as our user.
- This done by checking the app name of the app installed by our user and then matching that app name with other users app.
- Now if a user has same app, then the other apps installed by that user is recommended to our user

## **Conclusion:**

We can conclude that by using Neo4j we can do both Content-based Filtering and Collaborative Filtering easily. Then by using the above created graphs we can implement this in real-time to provide recommendation for users. We can also find the popularity of an app based on this. These methodologies can also be used in other recommendation system.