

NOSQL PROJECT
REVIEW- 2
PRODUCT RECOMMENDATION USING NEO4J

20MIA1046 Vishakhaa S

20MIA1153 Yashwanth Sarathi S P

Datasets:

Orders.csv
Employees.csv
Categories.csv
Customers.csv
Products.csv
Suppliers.csv

Creating Orders:

In this section, data from a CSV file containing order information is loaded into the Neo4j database. Each row in the CSV file corresponds to an order, and a node labeled as Order is created for each order. The orderID property is set based on the data in the CSV file.

// Create orders

```
LOAD CSV WITH HEADERS FROM file:///orders.csv' AS row
MERGE (order:Order {orderID: row.OrderID})
ON CREATE SET order.shipName = row.ShipName;
```

The screenshot shows the Neo4j Browser interface. In the top-left panel, a code editor contains the following Cypher query:

```
1 LOAD CSV WITH HEADERS FROM 'file:///orders.csv' AS row
2 MERGE (order:Order {orderID: row.OrderID})
3 | ON CREATE SET order.shipName = row.ShipName;
4
```

Below the code editor, a message indicates: "Added 830 labels, created 830 nodes, set 1660 properties, completed after 2446 ms." In the bottom panel, a command line interface shows the result of running the query: "\$:play start". The system status bar at the bottom right shows the date and time as 02-11-2023 01:32 PM.

`MATCH (o:Order) return o LIMIT 5;`

The screenshot shows the Neo4j Browser interface. In the top-left panel, a code editor contains the following Cypher query:

```
neo4j$ MATCH (o:Order) return o LIMIT 5;
```

The main workspace displays five nodes, each represented by a brown circle labeled with a number from 0 to 4. To the right, an "Overview" panel shows the results:

- Node labels:** * (5) Order (5)
- Displaying 5 nodes, 0 relationships.

The system status bar at the bottom right shows the date and time as 02-11-2023 01:24 PM.

Creating Products:

Similar to the previous section, this part loads data from a CSV file containing product information. For each product in the CSV file, a node labeled as Product is created in the Neo4j database. The product's name and unit price are set as properties of the node.

`// Create products`

```
LOAD CSV WITH HEADERS FROM file:///products.csv' AS row
```

```
MERGE (product:Product {productID: row.ProductID})
```

```
ON CREATE SET product.productName = row.ProductName, product.unitPrice =  
toFloat(row.UnitPrice);
```

The screenshot shows the Neo4j Browser interface. In the top-left panel, there is a code editor containing the following Cypher script:

```
1 LOAD CSV WITH HEADERS FROM 'file:///products.csv' AS row  
2 MERGE (product:Product {productID: row.ProductID})  
3 | ON CREATE SET product.productName = row.ProductName, product.unitPrice =  
4 toFloat(row.UnitPrice);
```

Below the code editor, a message box displays the results of the execution:

Added 77 labels, created 77 nodes, set 231 properties, completed after 588 ms.

At the bottom of the interface, another message box shows:

neo4j\$ MATCH (o:Order) return o LIMIT 5;

The system tray at the bottom of the screen indicates it's 01:25 PM on 02-11-2023, with a weather icon showing 31°C and "Light rain".

```
MATCH (p:Product) return p LIMIT 5;
```

The screenshot shows the Neo4j Browser interface. In the top-left panel, there is a code editor containing the following Cypher script:

```
neo4j$ MATCH (p:Product) return p LIMIT 5;
```

On the right side of the interface, a graph visualization shows five yellow circular nodes representing products. The nodes are labeled with their names: "Chef Anton's Season... ", "Aniseed Syrup", "Chang", "Chai", and "Chef Anton's Gumbo Mix".

A sidebar on the right provides an "Overview" of the results:

- Node labels: * (5) Product (5)
- Displaying 5 nodes, 0 relationships.

The system tray at the bottom of the screen indicates it's 01:33 PM on 02-11-2023, with a weather icon showing 31°C and "Light rain".

Creating Suppliers, Employees, and Categories:

These sections follow a similar pattern. They load data from CSV files related to suppliers, employees, and product categories. Nodes labeled as Supplier, Employee, and Category are created for each entry in the respective CSV files. The properties of these nodes, such as supplierID, employeeID, companyName, etc., are set based on the CSV data.

// Create suppliers

```
LOAD CSV WITH HEADERS FROM file:///suppliers.csv' AS row  
MERGE (supplier:Supplier {supplierID: row.SupplierID})  
ON CREATE SET supplier.companyName = row.CompanyName;
```

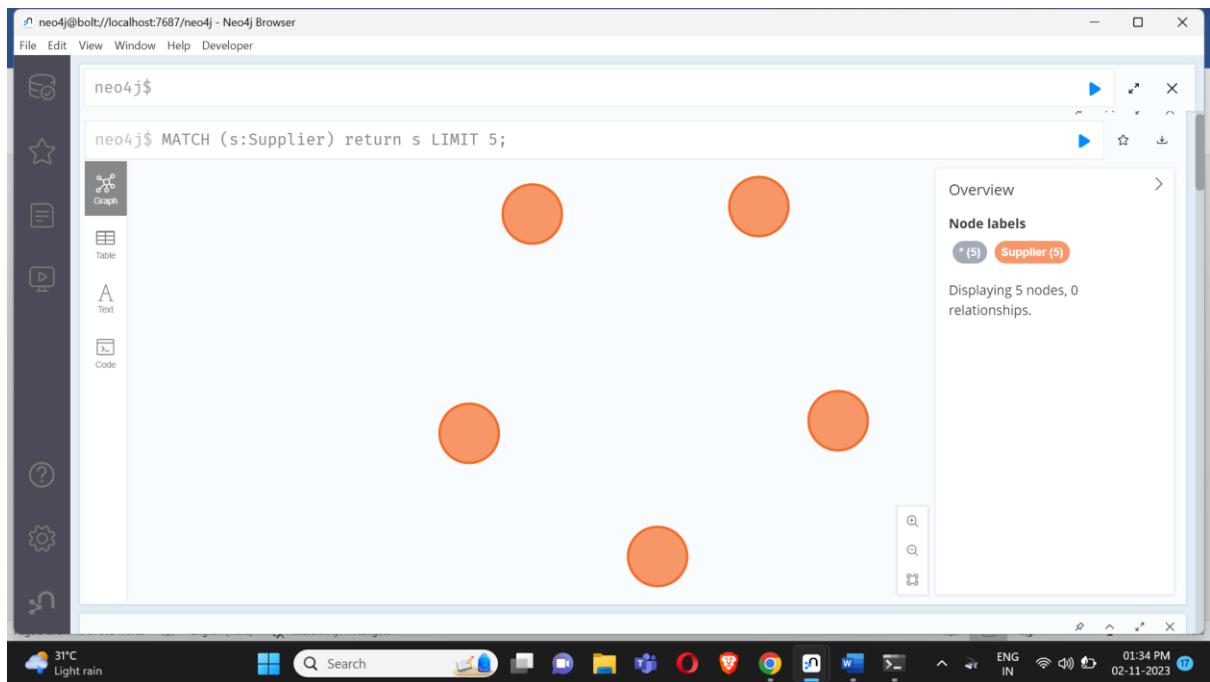
The screenshot shows the Neo4j Browser interface. The main window displays the following Cypher script:

```
1 // Create suppliers  
2 LOAD CSV WITH HEADERS FROM 'file:///suppliers.csv' AS row  
3 MERGE (supplier:Supplier {supplierID: row.SupplierID})  
4 | ON CREATE SET supplier.companyName = row.CompanyName;  
5
```

Below the script, a message indicates: "Added 29 labels, created 29 nodes, set 58 properties, completed after 584 ms." A second message below it also states: "Added 29 labels, created 29 nodes, set 58 properties, completed after 584 ms." At the bottom of the browser window, there is another Cypher query:neo4j\$ MATCH (p:Product) return p LIMIT 5;

The system tray at the bottom of the screen shows the date and time as 01:33 PM 02-11-2023.

```
MATCH (s:Supplier) return s LIMIT 5;
```

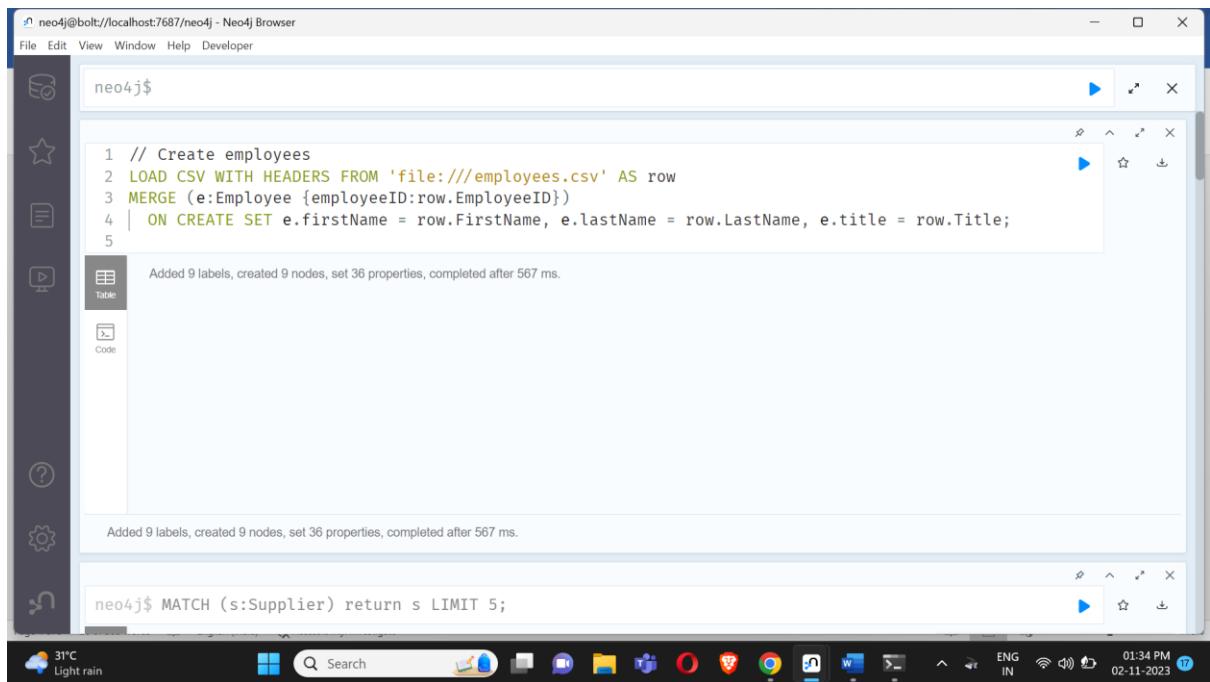


// Create employees

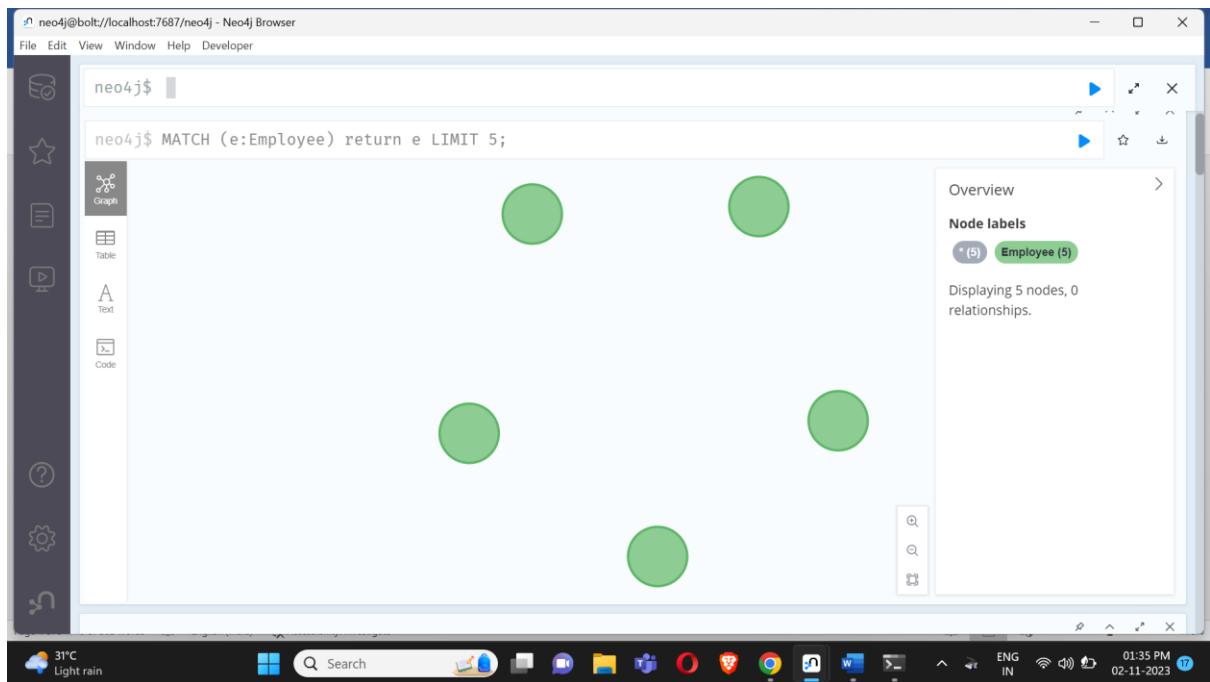
```
LOAD CSV WITH HEADERS FROM file:///employees.csv' AS row
```

```
MERGE (e:Employee {employeeID:row.EmployeeID})
```

```
ON CREATE SET e.firstName = row.FirstName, e.lastName = row.LastName, e.title =  
row.Title;
```

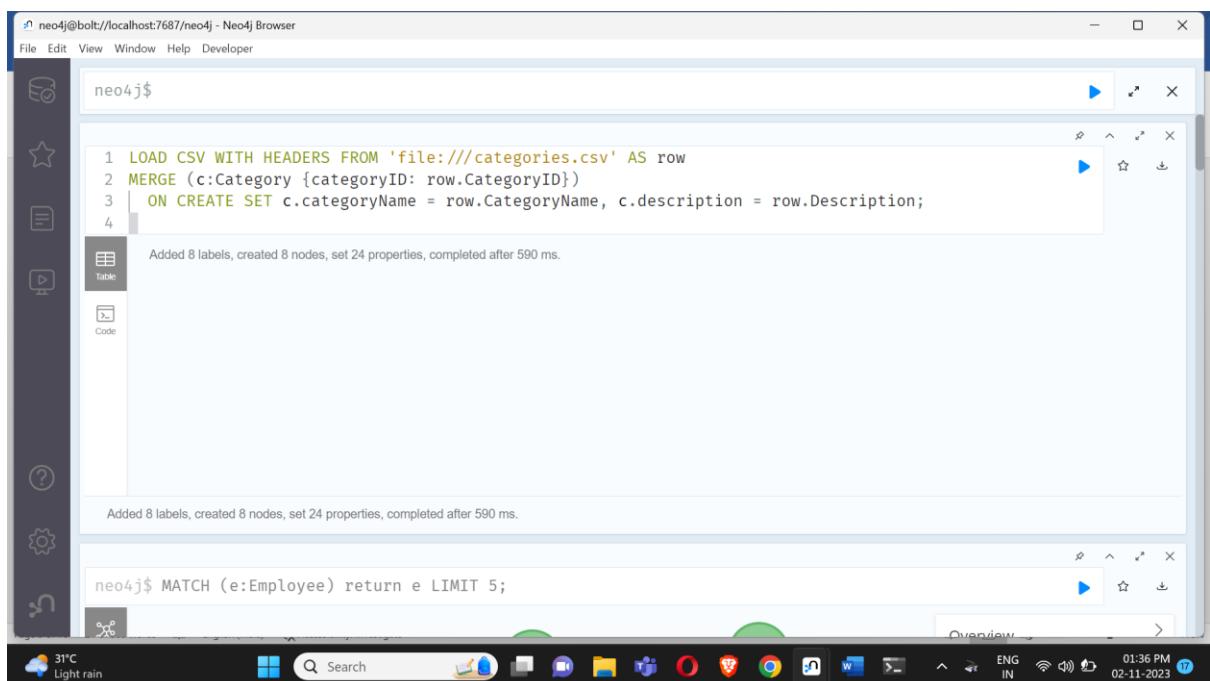


```
MATCH (e:Employee) return e LIMIT 5;
```

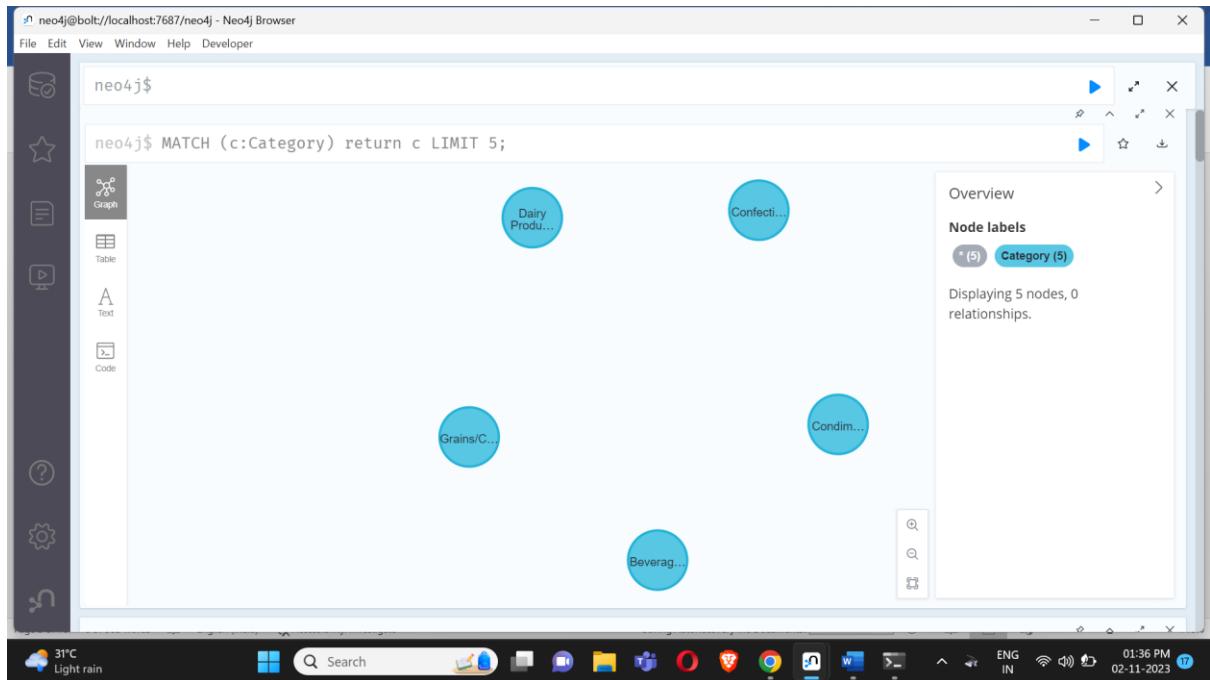


// Create categories

```
LOAD CSV WITH HEADERS FROM file:///categories.csv' AS row  
MERGE (c:Category {categoryID: row.CategoryID})  
ON CREATE SET c.categoryName = row.CategoryName, c.description = row.Description;
```



```
MATCH (c:Category) return c LIMIT 5;
```

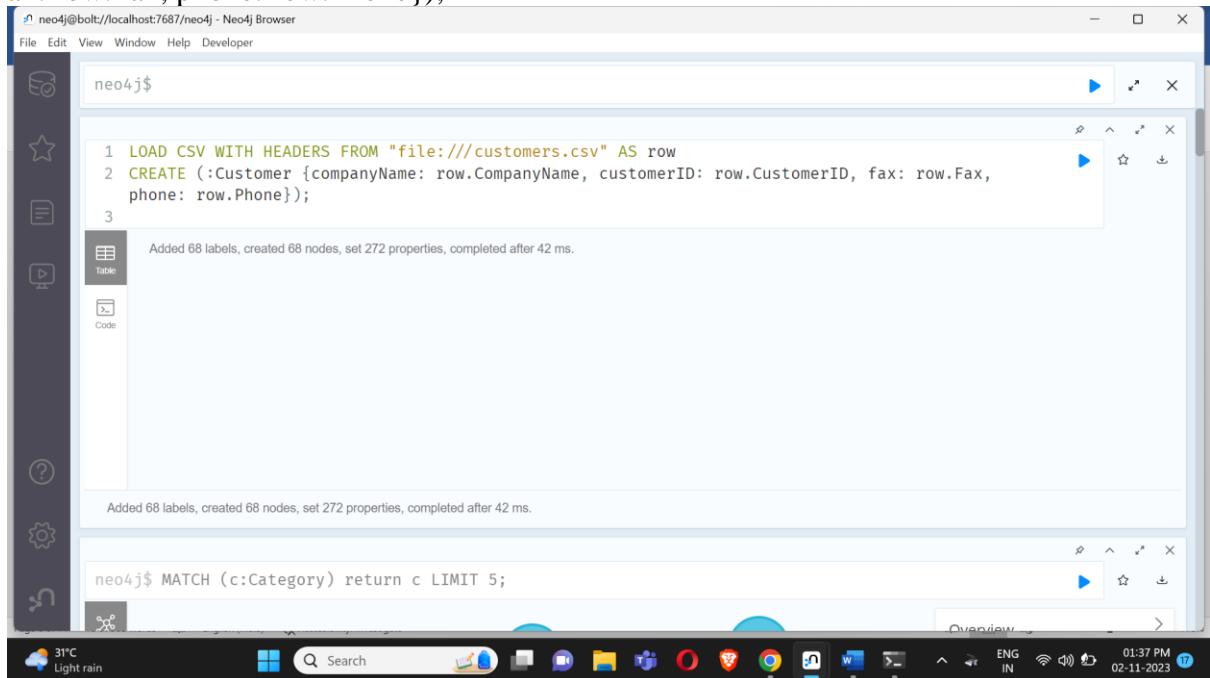


Creating Customers:

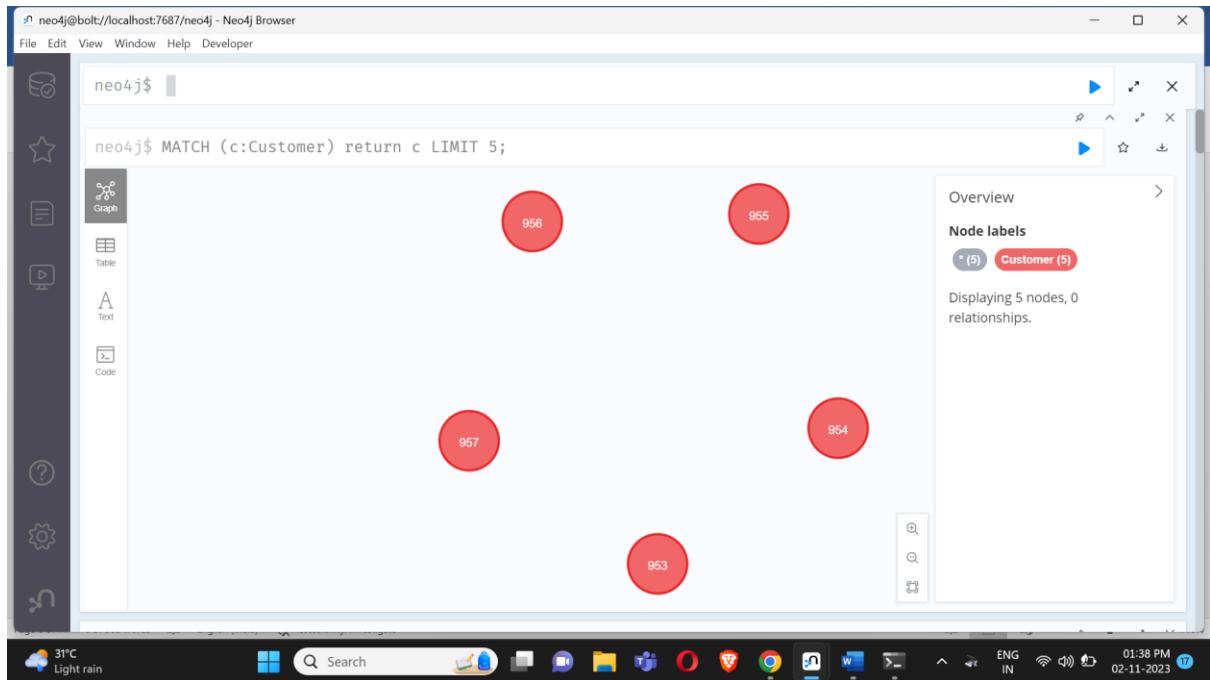
This section loads customer data from a CSV file and creates nodes labeled as Customer in the Neo4j database. The properties of these nodes, such as customerID, companyName, fax, and phone, are set based on the CSV data.

//create customers

```
LOAD CSV WITH HEADERS FROM "file:///customers.csv" AS row
CREATE (:Customer {companyName: row.CompanyName, customerID: row.CustomerID, f
ax: row.Fax, phone: row.Phone});
```



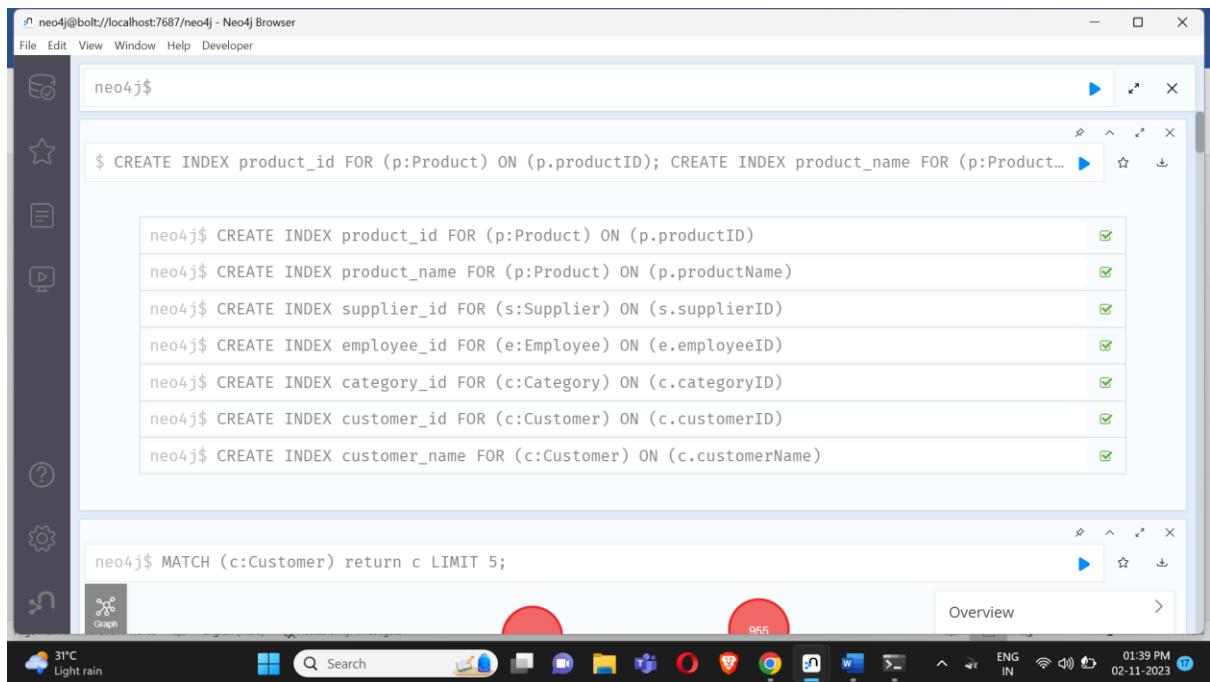
```
MATCH (c:Customer) return c LIMIT 5;
```



Creating Indexes and Constraints:

Indexes and constraints are essential for optimizing database performance and ensuring data integrity. In this section, various indexes are created on properties of nodes to speed up data retrieval. Additionally, a constraint is set on the orderID property to ensure that order IDs are unique.

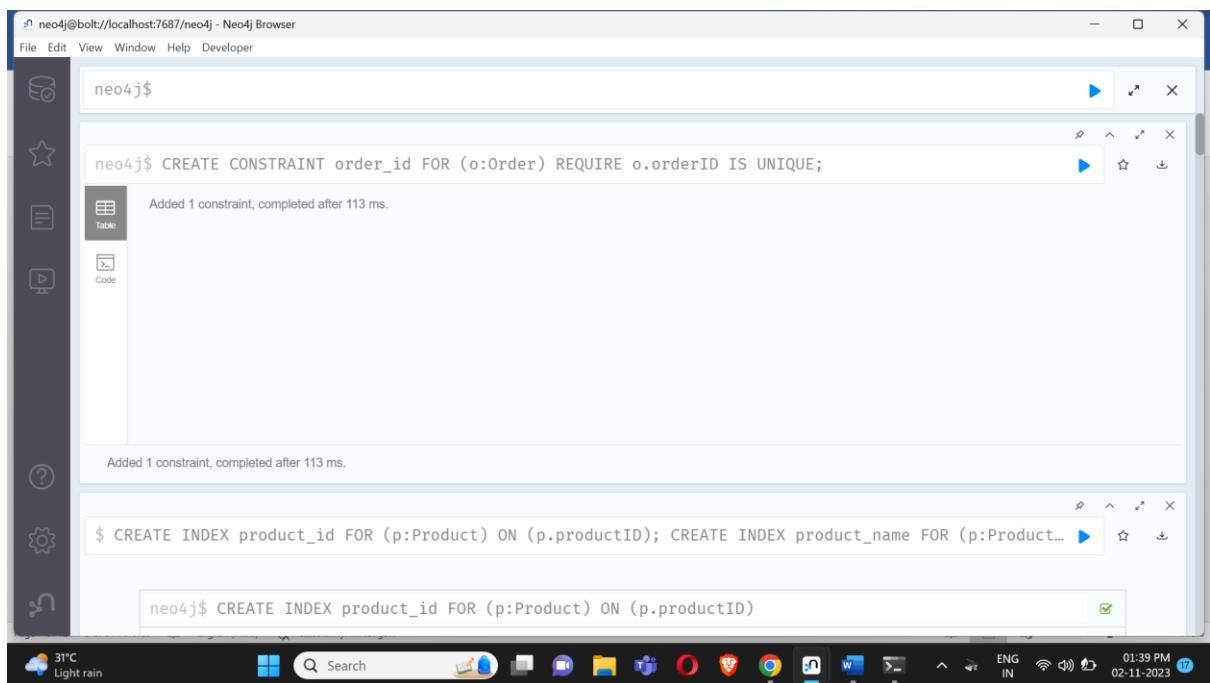
```
CREATE INDEX product_id FOR (p:Product) ON (p.productID);
CREATE INDEX product_name FOR (p:Product) ON (p.productName);
CREATE INDEX supplier_id FOR (s:Supplier) ON (s.supplierID);
CREATE INDEX employee_id FOR (e:Employee) ON (e.employeeID);
CREATE INDEX category_id FOR (c:Category) ON (c.categoryID);
CREATE INDEX customer_id FOR (c:Customer) ON (c.customerID);
CREATE INDEX customer_name FOR (c:Customer) ON (c.customerName);
```



neo4j\$ CREATE INDEX product_id FOR (p:Product) ON (p.productID); CREATE INDEX product_name FOR (p:Product...
neo4j\$ CREATE INDEX product_id FOR (p:Product) ON (p.productID)
neo4j\$ CREATE INDEX product_name FOR (p:Product) ON (p.productName)
neo4j\$ CREATE INDEX supplier_id FOR (s:Supplier) ON (s.supplierID)
neo4j\$ CREATE INDEX employee_id FOR (e:Employee) ON (e.employeeID)
neo4j\$ CREATE INDEX category_id FOR (c:Category) ON (c.categoryID)
neo4j\$ CREATE INDEX customer_id FOR (c:Customer) ON (c.customerID)
neo4j\$ CREATE INDEX customer_name FOR (c:Customer) ON (c.customerName)

neo4j\$ MATCH (c:Customer) return c LIMIT 5;

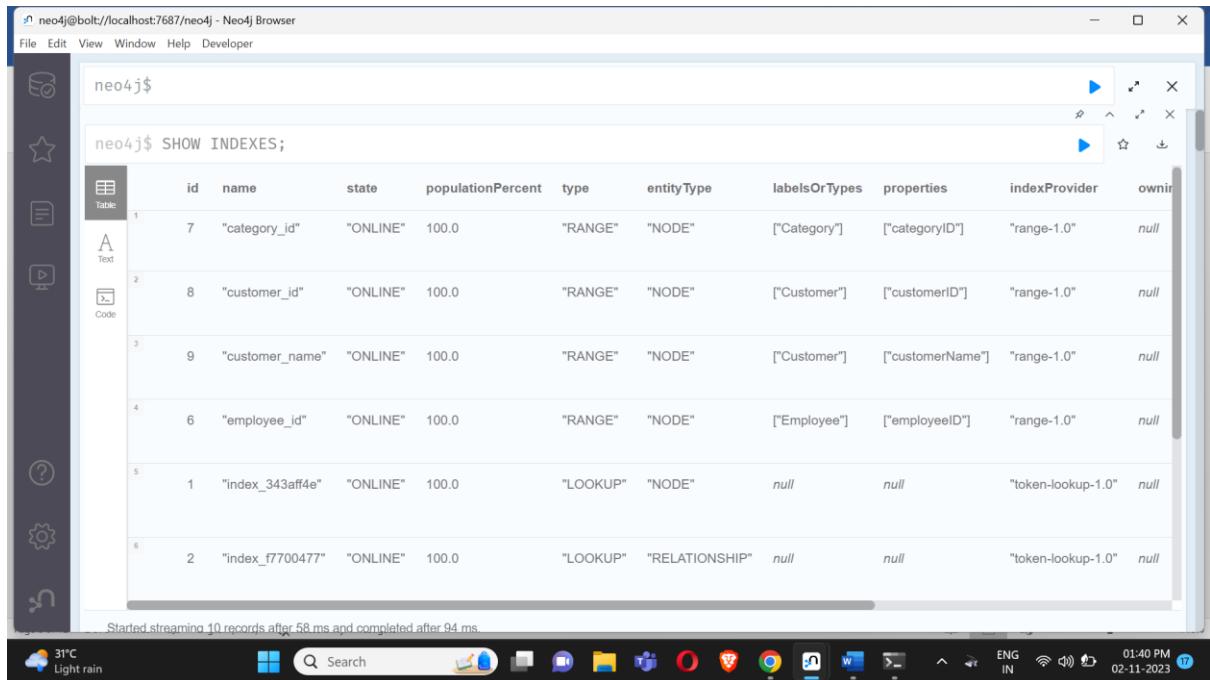
CREATE CONSTRAINT order_id FOR (o:Order) REQUIRE o.orderID IS UNIQUE;



neo4j\$ CREATE CONSTRAINT order_id FOR (o:Order) REQUIRE o.orderID IS UNIQUE;
Added 1 constraint, completed after 113 ms.

neo4j\$ CREATE INDEX product_id FOR (p:Product) ON (p.productID); CREATE INDEX product_name FOR (p:Product...
neo4j\$ CREATE INDEX product_id FOR (p:Product) ON (p.productID)

SHOW INDEXES;



Creating Relationships between Orders and Products:

This part establishes relationships between Order nodes and Product nodes. It represents that an order contains one or more products. Information about the unit price and quantity of each product within an order is also stored as properties on the relationship.

// Create relationships between orders and products

```
LOAD CSV WITH HEADERS FROM file:///orders.csv' AS row
```

```
MATCH (order:Order {orderID: row.OrderID})
```

```
MATCH (product:Product {productID: row.ProductID})
```

```
MERGE (order)-[op:CONTAINS]->(product)
```

```
ON CREATE SET op.unitPrice = toFloat(row.UnitPrice), op.quantity =
toFloat(row.Quantity);
```

```

neo4j$ 
1 // Create relationships between orders and products
2 LOAD CSV WITH HEADERS FROM 'file:///orders.csv' AS row
3 MATCH (order:Order {orderId: row.OrderID})
4 MATCH (product:Product {productID: row.ProductID})
5 MERGE (order)-[op:CONTAINS]-(product)
6 ON CREATE SET op.unitPrice = toFloat(row.UnitPrice), op.quantity = toFloat(row.Quantity);
7
8

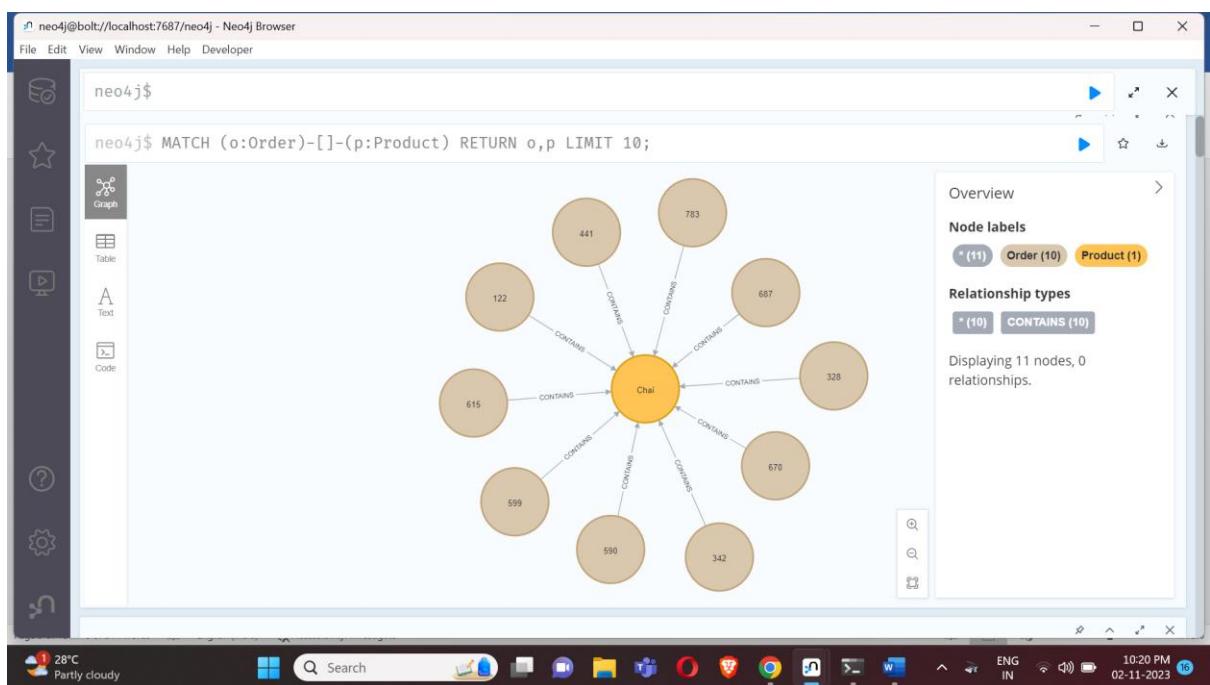
```

Set 4310 properties, created 2155 relationships, completed after 1699 ms.

Set 4310 properties, created 2155 relationships, completed after 1699 ms.

MATCH (o:Order)-[]-(p:Product)

RETURN o,p LIMIT 10;



Creating Relationships between Orders and Employees:

Relationships are created between Order nodes and Employee nodes, indicating which employee is responsible for selling a particular order. This information can be used to analyze the performance of employees in terms of sales.

// Create relationships between orders and employees

```
LOAD CSV WITH HEADERS FROM file:///orders.csv' AS row
```

```
MATCH (order:Order {orderId: row.OrderID})
```

```
MATCH (employee:Employee {employeeID: row.EmployeeID})
```

```
MERGE (employee)-[:SOLD]->(order);
```

The screenshot shows the Neo4j Browser interface. In the top-left corner, there's a terminal window with the command:

```
neo4j$ LOAD CSV WITH HEADERS FROM 'file:///orders.csv' AS row  
MATCH (order:Order {orderId: row.OrderID})  
MATCH (employee:Employee {employeeID: row.EmployeeID})  
MERGE (employee)-[:SOLD]->(order);
```

Below the command, a message indicates: "Created 830 relationships, completed after 743 ms." In the bottom-left corner of the browser window, another terminal window shows the result of the query:

```
neo4j$ MATCH (o:Order)-[]-(p:Product) RETURN o,p LIMIT 10;
```

The system status bar at the bottom right shows the date and time: 02-11-2023 10:21 PM.

```
MATCH (o:Order)-[]-(e:Employee)
```

```
RETURN o,e LIMIT 10;
```

The screenshot shows the Neo4j Browser interface with the results of the previous query. The main area displays a graph where a central green node (representing an Order) is connected to ten other brown nodes (representing Employees) by arrows labeled "SOLD". The nodes are numbered: 92, 44, 56, 552, 646, 662, 139, 728, 152, and 638. To the right of the graph, the "Overview" panel provides summary statistics:

- Node labels:** Order (10), Employee (1)
- Relationship types:** SOLD (10)

The panel also states: "Displaying 11 nodes, 0 relationships." The system status bar at the bottom right shows the date and time: 02-11-2023 10:21 PM.

Relating Customers to Orders:

This section links Customer nodes to their respective Order nodes. It signifies that a customer made a purchase by placing an order. This information is useful for understanding customer behavior and preferences.

```
// Relate customers to orders
```

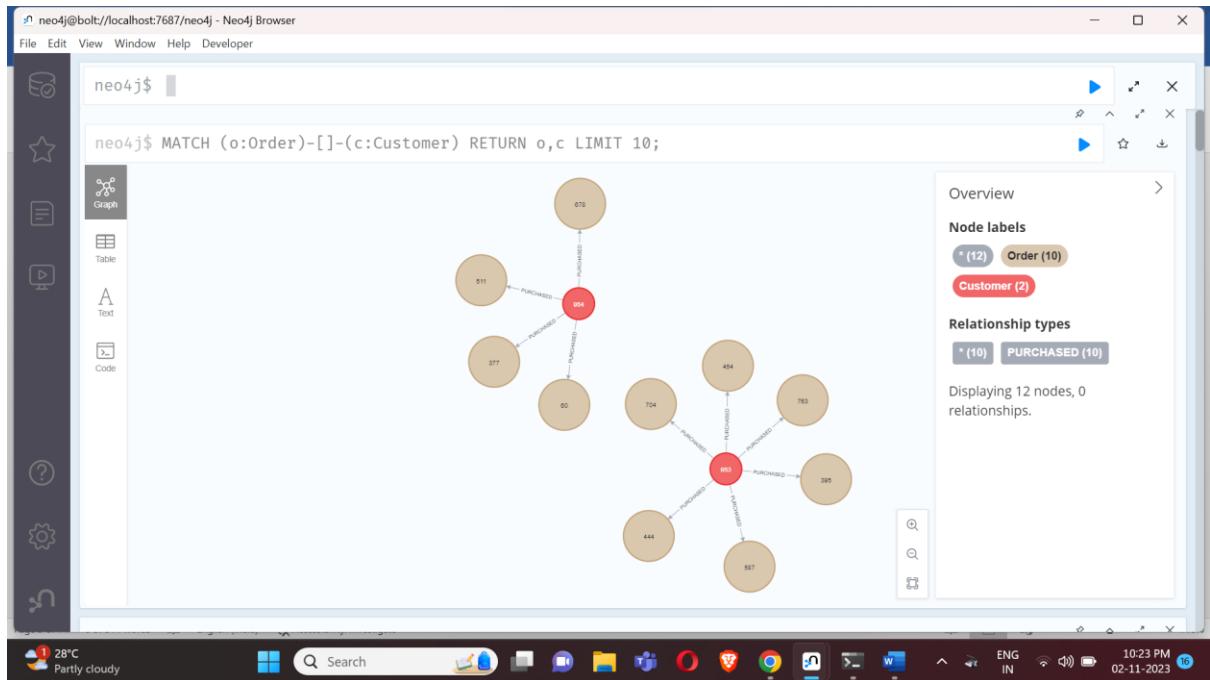
```
LOAD CSV WITH HEADERS FROM file:///orders.csv' AS row
MATCH (order:Order {orderID: row.OrderID})
MATCH (customer:Customer {customerID: row.CustomerID})
MERGE (customer)-[:PURCHASED]->(order);
```

The screenshot shows the Neo4j Browser interface. The title bar reads "neo4j@bolt://localhost:7687/neo4j - Neo4j Browser". The main window contains the following Cypher code:

```
1 // Relate customers to orders
2
3 LOAD CSV WITH HEADERS FROM 'file:///orders.csv' AS row
4 MATCH (order:Order {orderID: row.OrderID})
5 MATCH (customer:Customer {customerID: row.CustomerID})
6 MERGE (customer)-[:PURCHASED]->(order);
7
```

Below the code, a message indicates: "Created 571 relationships, completed after 684 ms." The interface includes a sidebar with icons for Table, Code, and Help, and a bottom navigation bar with various application icons and system status indicators.

```
MATCH (o:Order)-[]-(c:Customer)
RETURN o,c LIMIT 10;
```



Creating Relationships between Products and Suppliers:

Relationships are established between Product nodes and Supplier nodes, indicating which supplier supplies which products. This information is valuable for tracking the sources of products.

```
// Create relationships between products and suppliers
```

```
LOAD CSV WITH HEADERS FROM file:///products.csv
```

```
' AS row
```

```
MATCH (product:Product {productID: row.ProductID})
```

```
MATCH (supplier:Supplier {supplierID: row.SupplierID})
```

```
MERGE (supplier)-[:SUPPLIES]->(product);
```

```

1 // Create relationships between products and suppliers
2 LOAD CSV WITH HEADERS FROM 'file:///products.csv'
3 AS row
4 MATCH (product:Product {productID: row.ProductID})
5 MATCH (supplier:Supplier {supplierID: row.SupplierID})
6 MERGE (supplier)-[:SUPPLIES]-(product);
7

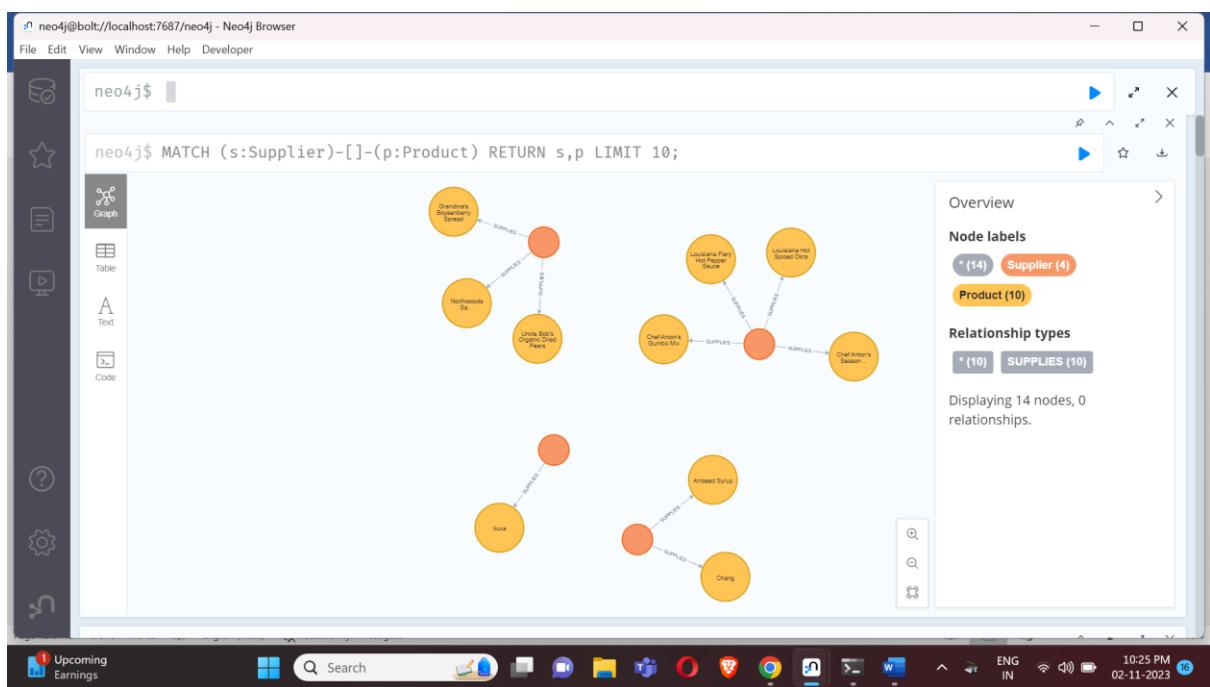
```

Created 77 relationships, completed after 721 ms.

Created 77 relationships, completed after 721 ms.

MATCH (s:Supplier)-[]-(p:Product)

RETURN s,p LIMIT 10;



Creating Relationships between Products and Categories:

This section associates Product nodes with Category nodes, representing that a product belongs to a particular category. This categorization is useful for organizing and querying products.

// Create relationships between products and categories

LOAD CSV WITH HEADERS FROM file:///products.csv

' AS row

```
MATCH (product:Product {productID: row.ProductID})
```

```
MATCH (category:Category {categoryID: row.CategoryID})
```

```
MERGE (product)-[:PART_OF]->(category);
```

The screenshot shows the Neo4j Browser interface. In the top-left corner, there's a terminal window titled "neo4j\$". Inside, the following Cypher script is run:

```
1 // Create relationships between products and categories
2 LOAD CSV WITH HEADERS FROM 'file:///products.csv'
3 ' AS row
4 MATCH (product:Product {productID: row.ProductID})
5 MATCH (category:Category {categoryID: row.CategoryID})
6 MERGE (product)-[:PART_OF]->(category);
7
```

After the script runs, a message is displayed: "Created 77 relationships, completed after 369 ms."

Below the terminal, there are tabs for "Table" and "Code". The status bar at the bottom indicates "Created 77 relationships, completed after 369 ms.".

```
MATCH (c:Category)-[]-(p:Product)
```

```
RETURN c,p LIMIT 10;
```

The screenshot shows the Neo4j Browser interface with the results of the previous query. The terminal window now displays:

```
neo4j$ MATCH (c:Category)-[]-(p:Product) RETURN c,p LIMIT 10;
```

On the right side of the interface, there's an "Overview" panel. It shows the following statistics:

- Node labels:** * (11), Category (1), Product (10)
- Relationship types:** * (10), PART_OF (10)

It also states: "Displaying 11 nodes, 0 relationships."

The main area shows a graph visualization with a central node labeled "Reverag" (Category). Various orange nodes (Products) are connected to it by arrows labeled "PART_OF". The products include "Outback Lager", "Guaraná FantaSTICA", "Lakkalakdóri", "Sasquatch Ale", "Chang", "Côte de Blaye", "Steeleye Stout", "Laughing Lumberjack Lager", "Chartreuse verte", and "Chai".

Creating Relationships between Employees (Reporting Hierarchy):

Relationships are created between Employee nodes to represent the reporting hierarchy within the organization. An Employee node is linked to their respective manager using the REPORTS_TO relationship. This hierarchy is essential for understanding the organization's structure.

// Create relationships between employees (reporting hierarchy)

```
LOAD CSV WITH HEADERS FROM file:///employees.csv' AS row
```

```
MATCH (employee:Employee {employeeID: row.EmployeeID})
```

```
MATCH (manager:Employee {employeeID: row.ReportsTo})
```

```
MERGE (employee)-[:REPORTS_TO]->(manager);
```

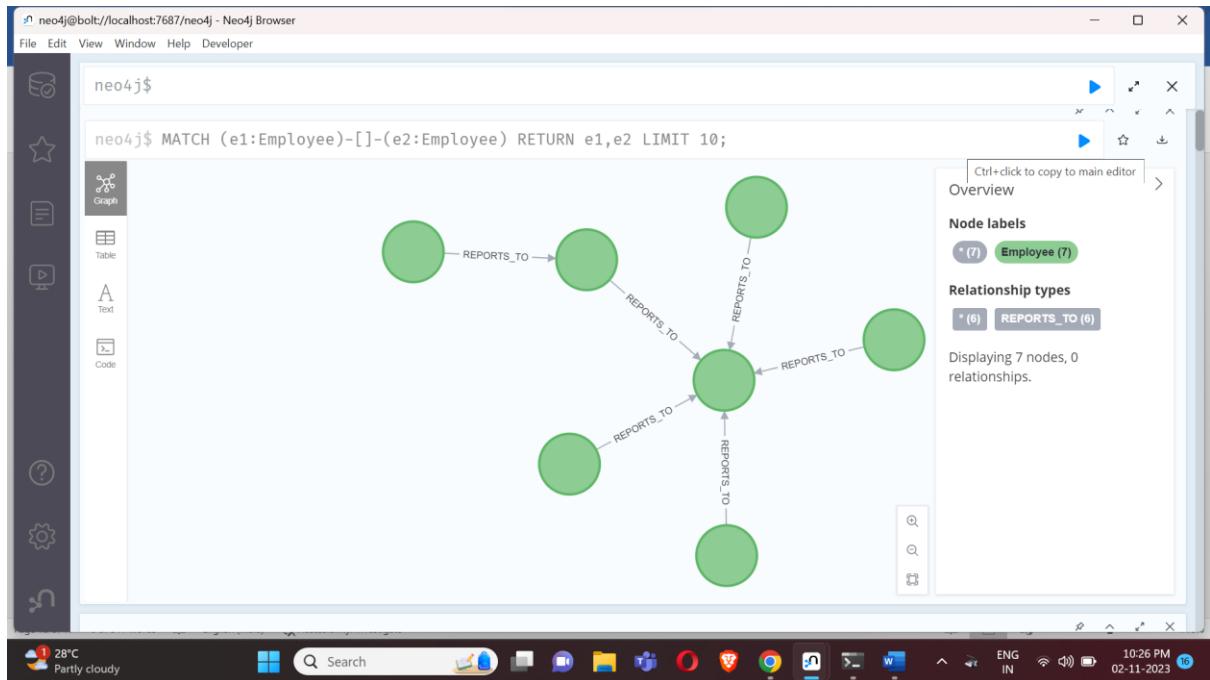
The screenshot shows the Neo4j Browser application window. The title bar reads "neo4j@bolt://localhost:7687/neo4j - Neo4j Browser". The main area contains the following Cypher code:

```
1 // Create relationships between employees (reporting hierarchy)
2 LOAD CSV WITH HEADERS FROM 'file:///employees.csv' AS row
3 MATCH (employee:Employee {employeeID: row.EmployeeID})
4 MATCH (manager:Employee {employeeID: row.ReportsTo})
5 MERGE (employee)-[:REPORTS_TO]->(manager);
6
```

Below the code, a message indicates: "Created 8 relationships, completed after 607 ms." At the bottom of the browser window, there is another Cypher query:neo4j\$ MATCH (c:Category)-[]-(p:Product) RETURN c,p LIMIT 10;

```
MATCH (e1:Employee)-[]-(e2:Employee)
```

```
RETURN e1,e2 LIMIT 10;
```

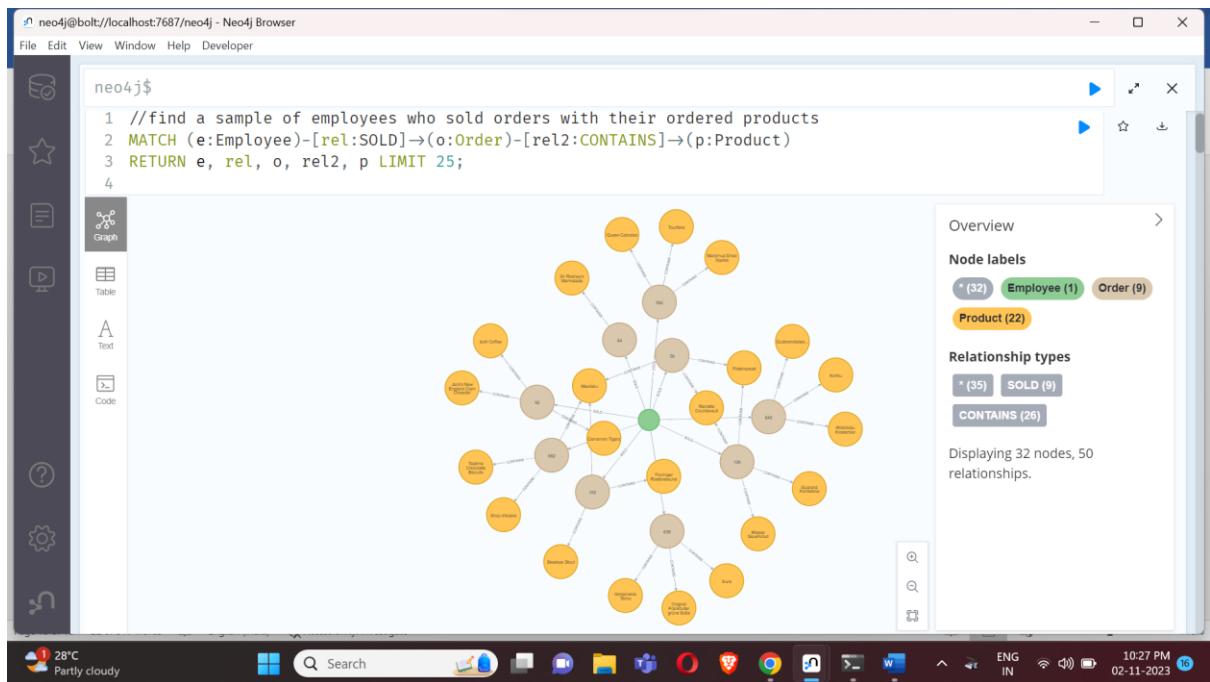


ANALYSIS:

This section contains various Cypher queries for analyzing the data in the Neo4j graph database. The queries provide insights into employee performance, product information, organizational structure, and order statistics.

//find a sample of employees who sold orders with their ordered products

```
MATCH (e:Employee)-[rel:SOLD]->(o:Order)-[rel2:CONTAINS]->(p:Product)
RETURN e, rel, o, rel2, p LIMIT 25;
```



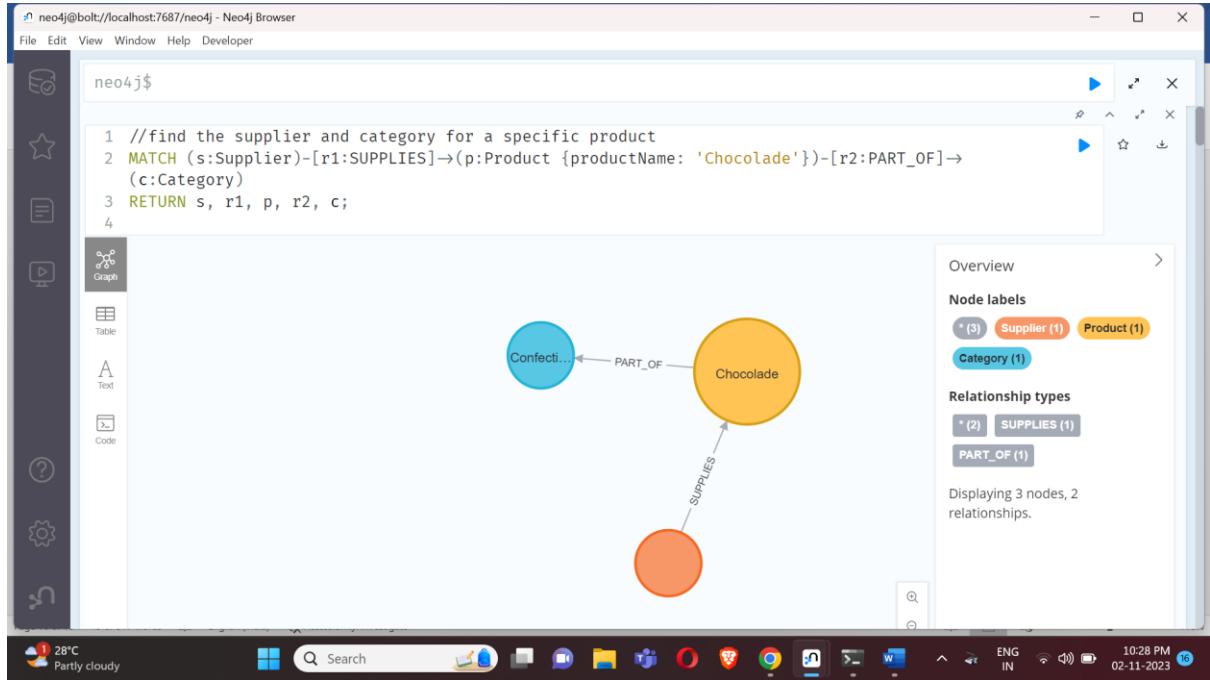
//find the supplier and category for a specific product

```

MATCH (s:Supplier)-[r1:SUPPLIES]->(p:Product {productName: 'Chocolade'})-
[r2:PART_OF]->(c:Category)

RETURN s, r1, p, r2, c;

```



// Which Employee had the Highest Cross-Selling Count of 'Chocolade' and Another Product?

```

MATCH (choc:Product {productName:'Chocolade'})<[:-CONTAINS]-(:Order)<[:-SOLD]-
(employee), (employee)-[:-SOLD]->(o2)-[:-CONTAINS]->(other:Product)

```

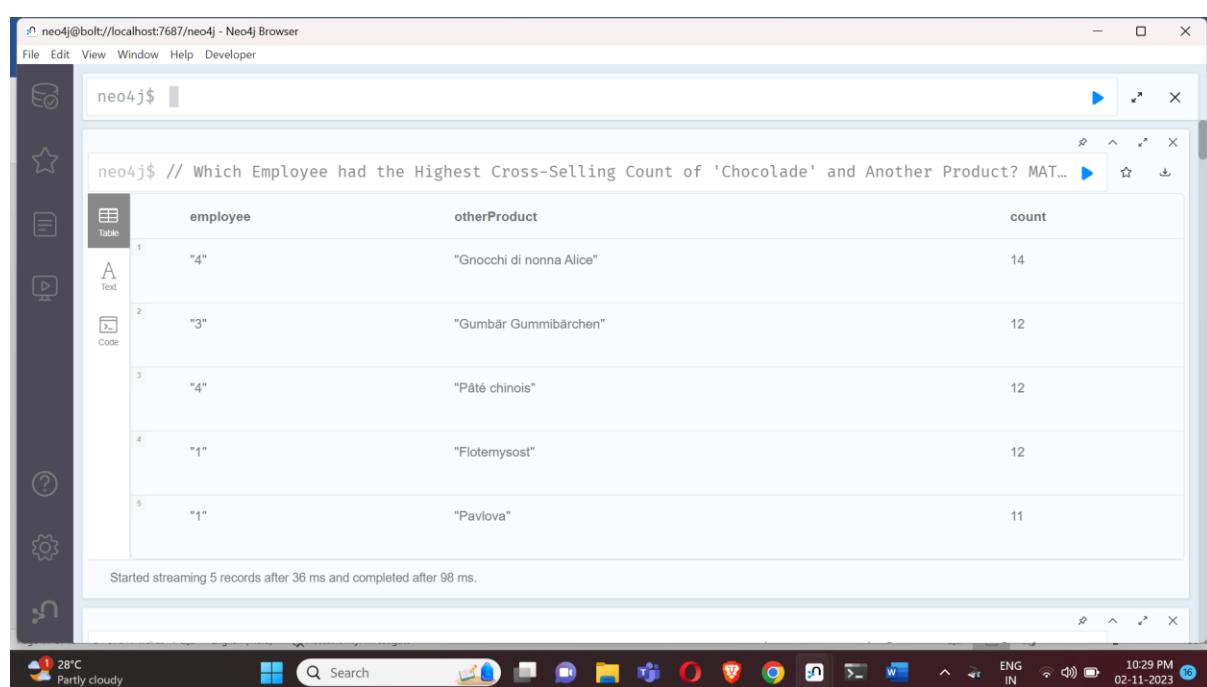
```

RETURN employee.employeeID AS employee, other.productName AS otherProduct,
count(DISTINCT o2) AS count

ORDER BY count DESC

LIMIT 5;

```



The screenshot shows the Neo4j Browser interface with a query results table. The table has three columns: employee, otherProduct, and count. The data is as follows:

	employee	otherProduct	count
1	"4"	"Gnocchi di nonna Alice"	14
2	"3"	"Gummibärchen"	12
3	"4"	"Pâté chinois"	12
4	"1"	"Flotemysost"	12
5	"1"	"Pavlova"	11

Started streaming 5 records after 36 ms and completed after 98 ms.

// How are Employees Organized? Who Reports to Whom?

```

MATCH (e:Employee)-[:REPORTS_TO]-(sub)

RETURN e.employeeID AS manager, sub.employeeID AS employee;

```

neo4j\$ // How are Employees Organized? Who Reports to Whom? MATCH (e:Employee)←[:REPORTS_TO]-(sub) ...

	manager	employee
1	"2"	"5"
2	"2"	"3"
3	"2"	"4"
4	"2"	"1"
5	"2"	"8"
6	"5"	"7"

Started streaming 8 records after 24 ms and completed after 26 ms.

// Which Employees Report to Each Other Indirectly?

```
MATCH path = (e:Employee)<-[:REPORTS_TO*]-(sub)
WITH e, sub, [person in NODES(path) | person.employeeID][1..-1] AS path
RETURN e.employeeID AS manager, path as middleManager, sub.employeeID AS employee
ORDER BY size(path);
```

neo4j\$ // Which Employees Report to Each Other Indirectly? MATCH path = (e:Employee)<-[:REPORTS_TO*]...
manager middleManager employee

	manager	middleManager	employee
1	"2"	[]	"5"
2	"2"	[]	"3"
3	"2"	[]	"4"
4	"2"	[]	"1"
5	"2"	[]	"8"
6	"5"	[]	"7"

Started streaming 11 records after 502 ms and completed after 543 ms.

// How Many Orders were Made by Each Part of the Hierarchy?

```
MATCH (e:Employee)
```

```

OPTIONAL MATCH (e)<-[:REPORTS_TO*0..]-(sub)-[:SOLD]->(order)
RETURN e.employeeID AS employee, [x IN COLLECT(DISTINCT sub.employeeID)
WHERE x <> e.employeeID] AS reportsTo, COUNT(distinct order) AS totalOrders
ORDER BY totalOrders DESC;

```



The screenshot shows the Neo4j Browser interface with a query results table. The table has three columns: 'employee', 'reportsTo', and 'totalOrders'. The data is as follows:

employee	reportsTo	totalOrders
"2"	["5", "3", "4", "1", "8", "7", "6", "9"]	830
"5"	["7", "6", "9"]	224
"4"	[]	156
"3"	[]	127
"1"	[]	123
"8"	[]	104

Started streaming 9 records after 39 ms and completed after 66 ms.

//To find the most popular products in the dataset, we can follow the path from `:Customer` to `:Product`

```

MATCH (c:Customer)-[:PURCHASED]->(o:Order)-[:CONTAINS]->(p:Product)
RETURN c.companyName, p.productName, count(o) AS orders
ORDER BY orders DESC
LIMIT 5

```

The screenshot shows the Neo4j Browser interface with a query results table. The table has three columns: c.companyName, p.productName, and orders. The results show five rows of data, all from the same company ("Ernst Handel").

c.companyName	p.productName	orders
"Ernst Handel"	"Gorgonzola Telino"	4
"Ernst Handel"	"Guaraná Fantástica"	4
"Ernst Handel"	"Alice Mutton"	4
"Ernst Handel"	"Wimmers gute Semmelknödel"	4
"Rattlesnake Canyon Grocery"	"Alice Mutton"	4

Started streaming 5 records after 29 ms and completed after 43 ms.

Content-Based Recommendations:

Content-based recommendations are based on products that a specific customer has purchased. The query finds products that the customer has bought and suggests other products within the same category that the customer hasn't purchased. This is a basic form of recommendation.

//Content Based Recommendations

```

MATCH (c:Customer)-[:PURCHASED]->(o:Order)-[:CONTAINS]->(p:Product)
<-[:CONTAINS]-(o2:Order)-[:CONTAINS]->(p2:Product)-[:PART_OF]->(:Category)<-
[:PART_OF]-(p)
WHERE c.customerID = 'ANTON' and NOT( (c)-[:PURCHASED]->(:Order)-
[:CONTAINS]->(p2) )
return c.companyName, p.productName as has_purchased, p2.productName as has_also_purchased,
count(DISTINCT o2) as occurrences
order by occurrences desc
limit 5

```

The screenshot shows the Neo4j Browser interface with a query results table. The table has four columns: c.companyName, has_purchased, has_also_purchased, and occurrences. The data is as follows:

c.companyName	has_purchased	has_also_purchased	occurrences
"Antonio Moreno Taquería"	"Raclette Courdavault"	"Fiotemysost"	4
"Antonio Moreno Taquería"	"Queso Cabrales"	"Mozzarella di Giovanni"	3
"Antonio Moreno Taquería"	"Rhönbräu Klosterbier"	"Chartreuse verte"	3
"Antonio Moreno Taquería"	"Chang"	"Outback Lager"	3
"Antonio Moreno Taquería"	"Rhönbräu Klosterbier"	"Guaraná Fantástica"	3

Started streaming 5 records after 104 ms and completed after 475 ms.

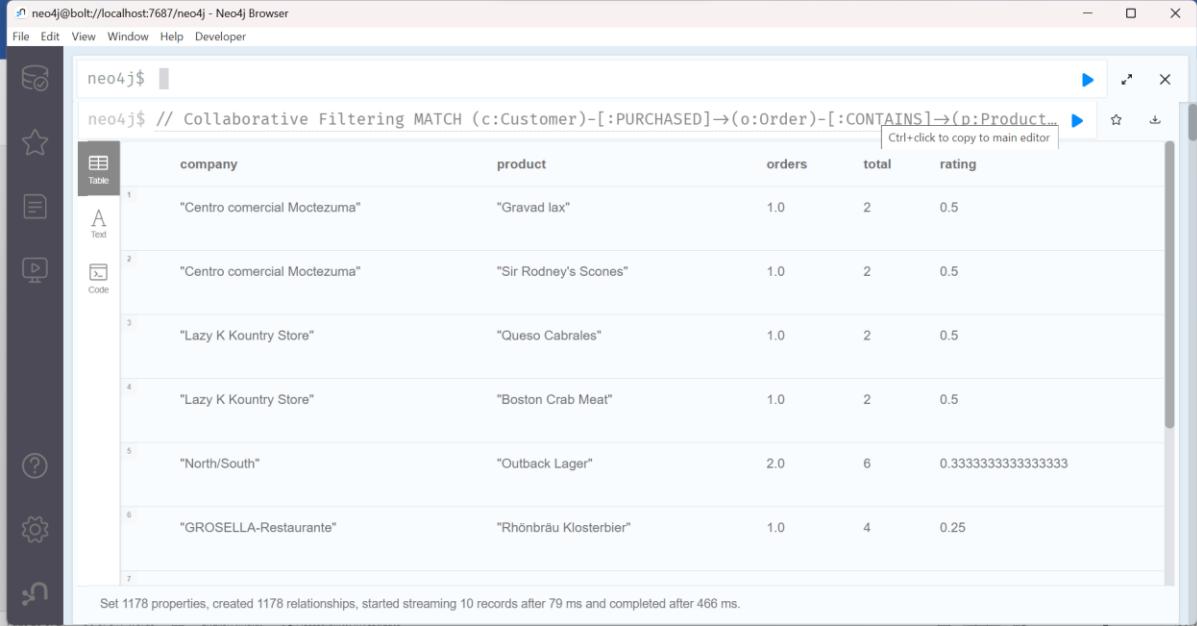
Collaborative Filtering

Collaborative filtering is a recommendation technique that identifies similarities between customers based on their product preferences. It calculates similarity scores between customers and products they have rated. These scores are used to make recommendations to customers based on the preferences of similar customers.

// Collaborative Filtering

```

MATCH (c:Customer)-[:PURCHASED]->(o:Order)-[:CONTAINS]->(p:Product)
WITH c, count(p) as total
MATCH (c)-[:PURCHASED]->(o:Order)-[:CONTAINS]->(p:Product)
WITH c, total,p, count(o)*1.0 as orders
MERGE (c)-[rated:RATED]->(p)
ON CREATE SET rated.rating = orders/total
ON MATCH SET rated.rating = orders/total
WITH c.companyName as company, p.productName as product, orders, total, rated.rating as rating
ORDER BY rating DESC
RETURN company, product, orders, total, rating LIMIT 10
    
```



```

neo4j$ // Collaborative Filtering MATCH (c:Customer)-[:PURCHASED]→(o:Order)-[:CONTAINS]→(p:Product)
    
```

Ctrl+click to copy to main editor

	company	product	orders	total	rating
1	"Centro comercial Moctezuma"	"Gravad lax"	1.0	2	0.5
2	"Centro comercial Moctezuma"	"Sir Rodney's Scones"	1.0	2	0.5
3	"Lazy K Kountry Store"	"Queso Cabrales"	1.0	2	0.5
4	"Lazy K Kountry Store"	"Boston Crab Meat"	1.0	2	0.5
5	"North/South"	"Outback Lager"	2.0	6	0.3333333333333333
6	"GROSELLA-Restaurante"	"Rhönbräu Klosterbier"	1.0	4	0.25
7					

Set 1178 properties, created 1178 relationships, started streaming 10 records after 79 ms and completed after 466 ms.



```

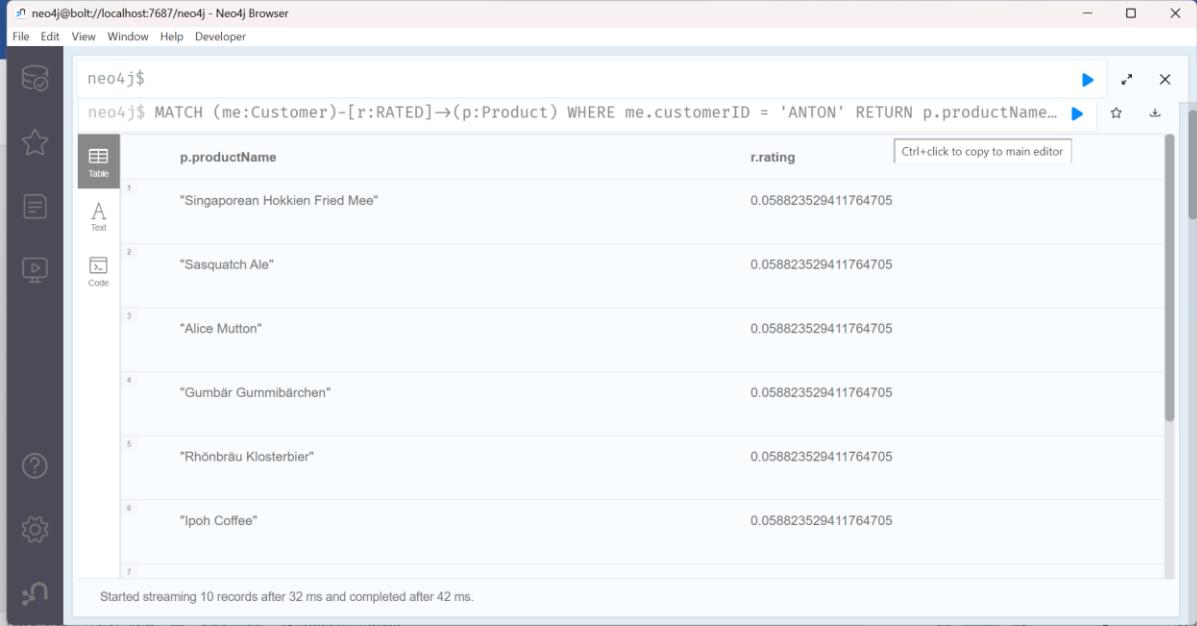
neo4j$ // Collaborative Filtering MATCH (c:Customer)-[:PURCHASED]→(o:Order)-[:CONTAINS]→(p:Product)
    
```

	company	product	orders	total	rating
5	"North/South"	"Outback Lager"	2.0	6	0.3333333333333333
6	"GROSELLA-Restaurante"	"Rhönbräu Klosterbier"	1.0	4	0.25
7	"GROSELLA-Restaurante"	"Ikura"	1.0	4	0.25
8	"GROSELLA-Restaurante"	"Mozzarella di Giovanni"	1.0	4	0.25
9	"GROSELLA-Restaurante"	"Thüringer Rostbratwurst"	1.0	4	0.25
10	"Hungry Coyote Import Store"	"Tourtière"	2.0	9	0.2222222222222222

Set 1685 properties, started streaming 10 records after 66 ms and completed after 813 ms.

```

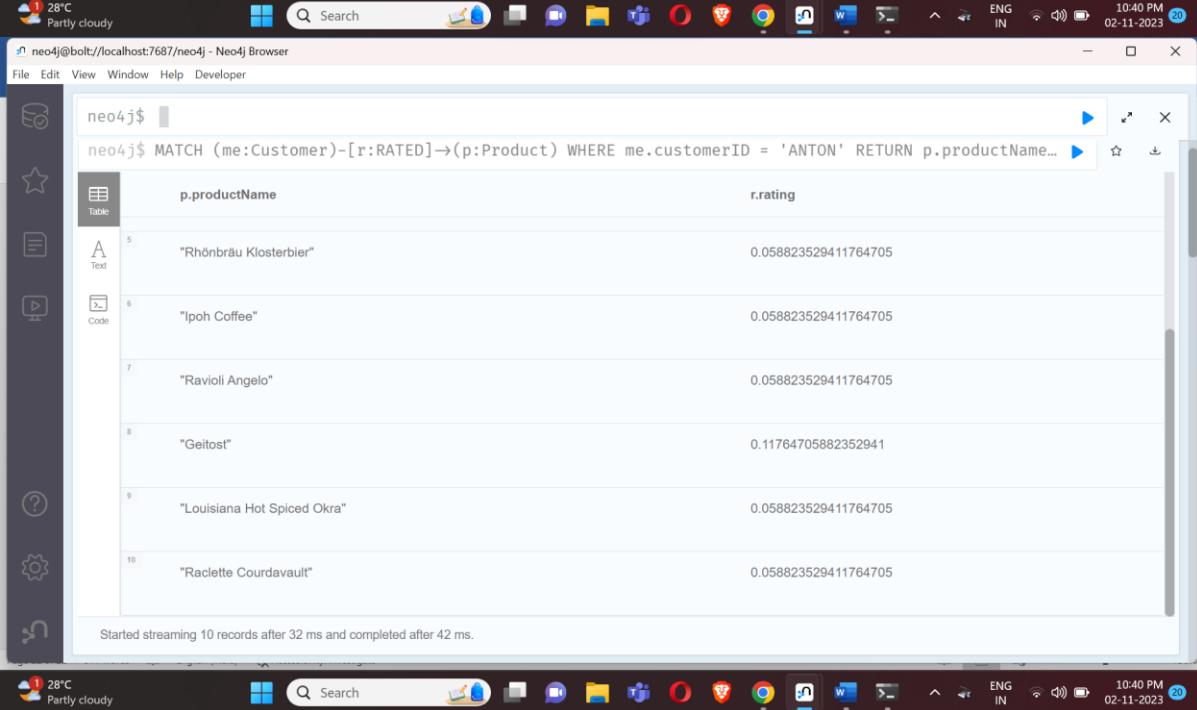
MATCH (me:Customer)-[r:RATED]->(p:Product)
WHERE me.customerID = 'ANTON'
RETURN p.productName, r.rating limit 10
    
```



The screenshot shows the Neo4j Browser interface with a query results table. The table has two columns: 'p.productName' and 'r.rating'. The results are as follows:

p.productName	r.rating
"Singaporean Hokkien Fried Mee"	0.058823529411764705
"Sasquatch Ale"	0.058823529411764705
"Alice Mutton"	0.058823529411764705
"Gumbär Gummibärchen"	0.058823529411764705
"Rhönbräu Klosterbier"	0.058823529411764705
"Ipoh Coffee"	0.058823529411764705

Started streaming 10 records after 32 ms and completed after 42 ms.



The screenshot shows the Neo4j Browser interface with a query results table. The table has two columns: 'p.productName' and 'r.rating'. The results are as follows:

p.productName	r.rating
"Rhönbräu Klosterbier"	0.058823529411764705
"Ipoh Coffee"	0.058823529411764705
"Ravioli Angelo"	0.058823529411764705
"Geitost"	0.11764705882352941
"Louisiana Hot Spiced Okra"	0.058823529411764705
"Raclette Courdavault"	0.058823529411764705

Started streaming 10 records after 32 ms and completed after 42 ms.

// See Customer's Similar Ratings to Others

```

MATCH (c1:Customer {customerID:'ANTON'})-[r1:RATED]->(p:Product)<-[r2:RATED]-
(c2:Customer)
RETURN c1.customerID, c2.customerID, p.productName, r1.rating, r2.rating,
CASE WHEN r1.rating-r2.rating < 0 THEN -(r1.rating-r2.rating) ELSE r1.rating-
r2.rating END as difference
ORDER BY difference ASC
LIMIT 15

```

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

neo4j\$

```
neo4j$ // See Customer's Similar Ratings to Others
MATCH (c1:Customer {customerID:'ANTON'})-[r1:RATES]-p
MATCH (c2:Customer {customerID:{c2}})-[r2:RATES]-p
RETURN c1.customerID AS c1.customerID, c2.customerID AS c2.customerID, p.productName AS p.productName, r1.rating AS r1.rating, r2.rating AS r2.rating, r2.rating - r1.rating AS difference
```

	c1.customerID	c2.customerID	p.productName	r1.rating	r2.rating	difference
1	"ANTON"	"WILMK"	"Raclette Courdavault"	0.058823529411764705	0.058823529411764705	0.0
2	"ANTON"	"WILMK"	"Singaporean Hokkien Fried Mee"	0.058823529411764705	0.058823529411764705	0.0
3	"ANTON"	"MAISD"	"Boston Crab Meat"	0.058823529411764705	0.058823529411764705	0.0
4	"ANTON"	"MAISD"	"Ipoh Coffee"	0.058823529411764705	0.058823529411764705	0.0
5	"ANTON"	"WILMK"	"Boston Crab Meat"	0.058823529411764705	0.058823529411764705	0.0
6	"ANTON"	"WILMK"	"Ipoh Coffee"	0.058823529411764705	0.058823529411764705	0.0
7						

Started streaming 15 records after 54 ms and completed after 65 ms.

```
28°C Partly cloudy Search ENG IN 10:41 PM 02-11-2023
```

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

neo4j\$

```
neo4j$ // See Customer's Similar Ratings to Others
MATCH (c1:Customer {customerID:'ANTON'})-[r1:RATES]-p
MATCH (c2:Customer {customerID:{c2}})-[r2:RATES]-p
RETURN c1.customerID AS c1.customerID, c2.customerID AS c2.customerID, p.productName AS p.productName, r1.rating AS r1.rating, r2.rating AS r2.rating, r2.rating - r1.rating AS difference
```

	c1.customerID	c2.customerID	p.productName	r1.rating	r2.rating	difference
7	"ANTON"	"MAISD"	"Raclette Courdavault"	0.058823529411764705	0.058823529411764705	0.0
8	"ANTON"	"LILAS"	"Gumbär Gummibärchen"	0.058823529411764705	0.058823529411764705	0.0
9	"ANTON"	"MAISD"	"Gumbär Gummibärchen"	0.058823529411764705	0.058823529411764705	0.0
10	"ANTON"	"BERGS"	"Rhönbräu Klosterbier"	0.058823529411764705	0.057692307692307696	0.0011312217194570096
11	"ANTON"	"LINOD"	"Boston Crab Meat"	0.058823529411764705	0.05714285714285714	0.001680672268907564
12	"ANTON"	"BOTTM"	"Raclette Courdavault"	0.058823529411764705	0.05714285714285714	0.001680672268907564
13						

Started streaming 15 records after 54 ms and completed after 65 ms.

```
28°C Partly cloudy Search ENG IN 10:41 PM 02-11-2023
```

The screenshot shows the Neo4j Browser interface with a table output. The table has columns: c1.customerID, c2.customerID, p.productName, r1.rating, r2.rating, and difference. The data consists of 15 rows, each representing a comparison between customer ANTON and another customer. The 'difference' column shows the absolute difference in ratings for each product.

	c1.customerID	c2.customerID	p.productName	r1.rating	r2.rating	difference
10	"ANTON"	"BERGS"	"Rhönbräu Klosterbier"	0.058823529411764705	0.057692307692307696	0.0011312217194570096
11	"ANTON"	"LINOD"	"Boston Crab Meat"	0.058823529411764705	0.05714285714285714	0.001680672268907564
12	"ANTON"	"BOTTM"	"Raclette Courdavault"	0.058823529411764705	0.05714285714285714	0.001680672268907564
13	"ANTON"	"BOTTM"	"Alice Mutton"	0.058823529411764705	0.05714285714285714	0.001680672268907564
14	"ANTON"	"RATTC"	"Alice Mutton"	0.058823529411764705	0.056338028169014086	0.0024855012427506193
15	"ANTON"	"WOLZA"	"Rhönbräu Klosterbier"	0.058823529411764705	0.0625	0.003676470588235295

Started streaming 15 records after 54 ms and completed after 65 ms.

// create a similarity score between two Customers using Cosine Similarity

```

MATCH (c1:Customer)-[r1:RATED]->(p:Product)<-[r2:RATED]-(c2:Customer)
WITH
SUM(r1.rating*r2.rating) as dot_product,SQRT( REDUCE(x=0.0, a IN COLLECT(r1.rating)
| x + a^2) ) as r1_length,
SQRT( REDUCE(y=0.0, b IN COLLECT(r2.rating) | y + b^2) ) as r2_length,
c1,c2
MERGE (c1)-[s:SIMILARITY]-(c2)
SET s.similarity = dot_product / (r1_length * r2_length)

```

The screenshot shows the Neo4j Browser interface with a table output. The table has columns: c1.customerID, c2.customerID, p.productName, r1.rating, r2.rating, and difference. The data consists of 7234 rows, representing the similarity scores for all possible pairs of customers.

	c1.customerID	c2.customerID	p.productName	r1.rating	r2.rating	difference
10	"ANTON"	"BERGS"	"Rhönbräu Klosterbier"	0.058823529411764705	0.057692307692307696	0.0011312217194570096
11	"ANTON"	"LINOD"	"Boston Crab Meat"	0.058823529411764705	0.05714285714285714	0.001680672268907564
12	"ANTON"	"BOTTM"	"Raclette Courdavault"	0.058823529411764705	0.05714285714285714	0.001680672268907564
13	"ANTON"	"BOTTM"	"Alice Mutton"	0.058823529411764705	0.05714285714285714	0.001680672268907564
14	"ANTON"	"RATTC"	"Alice Mutton"	0.058823529411764705	0.056338028169014086	0.0024855012427506193
15	"ANTON"	"WOLZA"	"Rhönbräu Klosterbier"	0.058823529411764705	0.0625	0.003676470588235295

Set 7234 properties, completed after 911 ms.

Set 7234 properties, completed after 911 ms.

neo4j\$ // See Customer's Similar Ratings to Others MATCH (c1:Customer {customerID:'ANTON'})-[r1:RATE...

```

MATCH (me:Customer)-[r:SIMILARITY]->(them)
WHERE me.customerID='ANTON'
RETURN me.companyName, them.companyName, r.similarity
ORDER BY r.similarity DESC limit 10

```

Recommendation based on Similarity Scores:

Building upon the collaborative filtering approach, this section generates recommendations for a specific customer. It identifies similar customers and suggests products that these similar customers have purchased but the specific customer has not. These recommendations are based on the calculated similarity scores.

// recommendation based on these similarity scores.

WITH 1 as neighbours

```

MATCH (me:Customer)-[:SIMILARITY]->(c:Customer)-[r:RATED]->(p:Product)
WHERE me.customerID = 'ANTON' and NOT ( (me)-[:RATED|PRODUCT|ORDER*1..2]-
>(p:Product) )
WITH p, COLLECT(r.rating)[0..neighbours] as ratings,
collect(c.companyName)[0..neighbours] as customers
WITH p, customers, REDUCE(s=0,i in ratings | s+i) / LENGTH(ratings) as recommendation
ORDER BY recommendation DESC
RETURN p.productName, customers, recommendation LIMIT 10

```

	me.companyName	them.companyName	r.similarity
1	"Antonio Moreno Taqueria"	"Rancho grande"	1.0000000000000002
2	"Antonio Moreno Taqueria"	"Bólido Comidas preparadas"	1.0000000000000002
3	"Antonio Moreno Taqueria"	"Romero y tomillo"	1.0000000000000002
4	"Antonio Moreno Taqueria"	"Maison Dewey"	1.0
5	"Antonio Moreno Taqueria"	"Old World Delicatessen"	1.0
6	"Antonio Moreno Taqueria"	"Furia Bacalhau e Frutos do Mar"	1.0

Started streaming 10 records after 31 ms and completed after 35 ms.

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

File Edit View Window Help Developer

```
neo4j$ MATCH (me:Customer)-[r:SIMILARITY]→(them) WHERE me.customerID='ANTON' RETURN me.companyName, them.companyName, r.similarity
```

me.companyName	them.companyName	r.similarity
"Antonio Moreno Taqueria"	"Old World Delicatessen"	1.0
"Antonio Moreno Taquería"	"Furia Bacalhau e Frutos do Mar"	1.0
"Antonio Moreno Taqueria"	"Alfreds Futterkiste"	1.0
"Antonio Moreno Taqueria"	"The Big Cheese"	1.0
"Antonio Moreno Taquería"	"Let's Stop N Shop"	1.0
"Antonio Moreno Taqueria"	"Du monde entier"	1.0

Started streaming 10 records after 31 ms and completed after 35 ms.

28°C Partly cloudy Search ENG IN 10:42 PM 02-11-2023

Database Information

Use database

neo4j 

Node labels

- *(1,044)
- Category
- Customer
- Employee
- Order
- Product
- Supplier

Relationship types

- *(9,279)
- CONTAINS
- PART_OF
- PURCHASED
- RATED
- REPORTS_TO
- SIMILARITY
- SOLD
- SUPPLIES

Property keys

categoryID	categoryName	
companyName	customerID	
customerName	description	
employeeID	fax	firstName
lastName	orderId	phone
productID	productName	
quantity	rating	shipName
similarity	supplierID	title
unitPrice		