CSE 5331-001

DBMS MODELS AND IMPLEMENTATION TECHNIQUES

Summer 2023

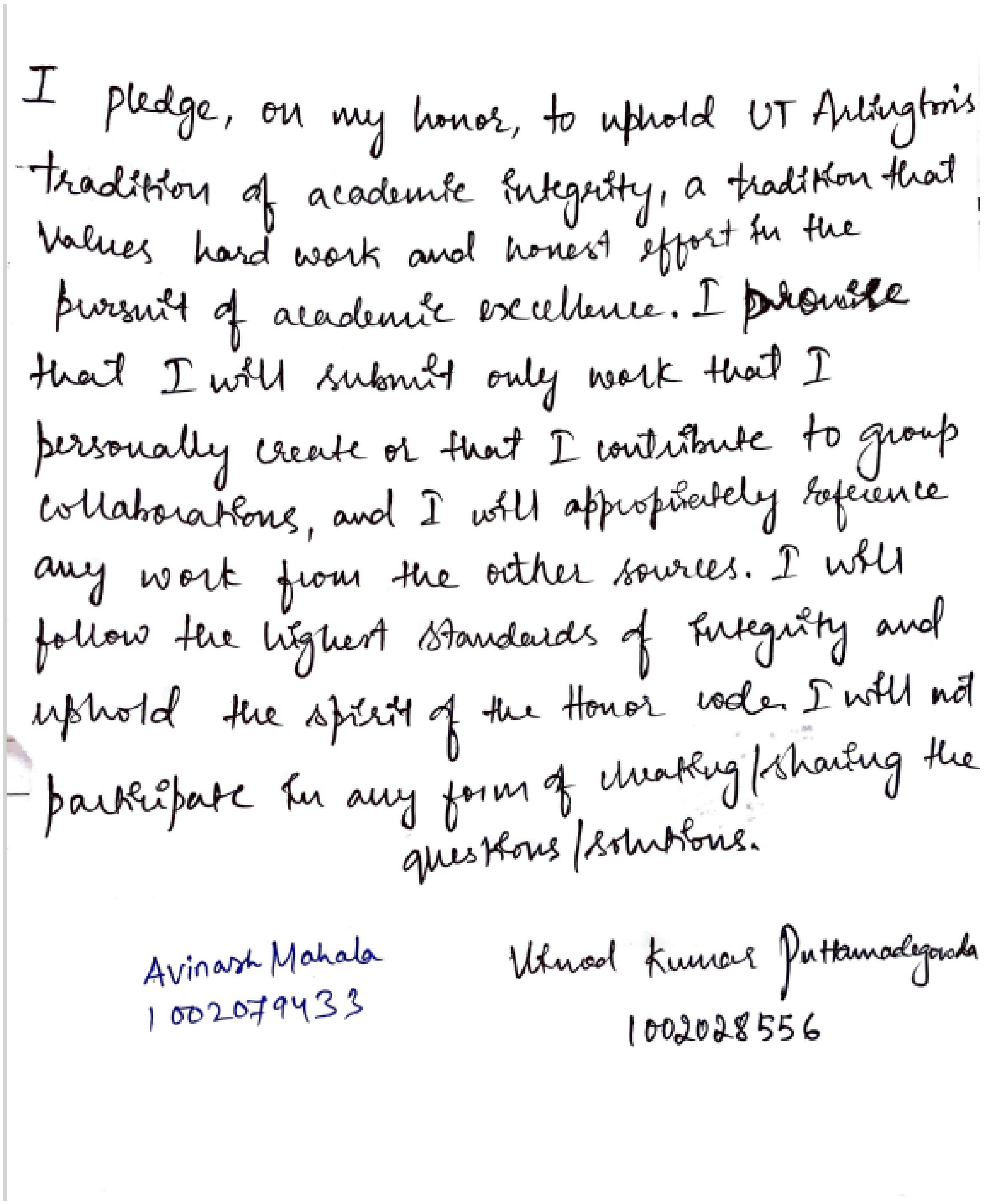
Project 2 - Arlington Sprouts Store Neo4j Project

Team Number - 01

Team Members:
Avinash Mahala (1002079433)
Vinod Kumar Puttamadegowda (1002028556)

Contents

1. Honor Code



I pledge, on my honor, to uphold UT Arlington's tradition of academic integrity, a tradition that values hard work and honest effort in the pursuit of academic excellence. I promise that I will submit only work that I personally create or that I contribute to group collaborations, and I will appropriately reference any work from the other sources. I will follow the highest standards of integrity and uphold the spirit of the Honor code. I will not participate in any form of cheating/sharing the questions/solutions.

Avinash Mahala
1002079433

Vinod Kumar Puttamadegowda
1002028556

2. Design Of The Program:-

Objective:

To seamlessly and accurately convert tabular data sourced from the CSV files of the Arlington Sprouts Store into a representative graph structure in Neo4j.

Workflow:

1. Establish Connection:
    - Purpose: Ensure that the program can communicate with the Neo4j database.
    - Steps:
        - Use the provided credentials (username, password, and URI) to initiate a connection.
        - Utilize the neo4j Python library to facilitate this connection.
2. **CSV File Parsing:**
    - **Purpose**: To interpret and transform the data contained in the CSV files to a format suitable for node creation in Neo4j.
    - **Steps**:
        - Read the File:
            - Utilize the Python CSV library to read each file in the predetermined list.
            - Access the header to determine column names which will subsequently serve as property keys for the nodes.
        - Parse the Data:
            - For each row in the CSV, interpret the data contained.
            - Using a utility function, determine the data type of each column (integer, float, or string) to ensure appropriate data type assignment in Neo4j.
3. **Node Creation:**
    - **Purpose**: To create entities in Neo4j that represent individual data points from the CSV.
    - **Steps**:
        - **Label Assignment:**
            - Use the name of the CSV file (excluding the .csv extension) as the label for the node.
            - This ensures a structured and predictable node naming convention, simplifying queries.
        - **Property Assignment:**
            - For every row in the CSV, iterate over each column.
            - Assign the column's data to the node as properties.
            - Execute a CREATE statement in Neo4j to instantiate the node with the assigned properties.

4. **Relationship Creation:**

- **Purpose**: To establish connections between nodes, creating a web of interconnected data.
- **Steps**:
    - **Relationship Definitions:**
        - Based on the project instructions, predefine the relationship types between nodes.
        - For example, a Vendor would have a "SELLS" relationship to an Item.
    - **Cypher Execution:**
        - After all nodes are created, iterate over the predefined relationships.
        - Using the MATCH clause, locate the nodes that need to be connected.
        - Establish the relationship between these nodes using the CREATE clause in Cypher.
        - Ensure the relationship direction is adhered to for accurate data representation.

3. Our Approach:-

Introduction:

For this project, our mission was to convert tabular data from CSV files (sourced from Arlington Sprouts Store) into a comprehensive graph representation using Neo4j. This documentation outlines our approach from initial setup to completion.

Approach:

1. Understanding the Dataset:

Before diving into code, we spent time understanding the structure and relationships of the data:

- What kind of data each CSV file contained.
- The relationships between different entities (like Vendor, Item, Customer, etc.).
- How data from one CSV relates to another (e.g., Vendor and Vendor_Item).

2. Setting Up the Development Environment:

- Initialized a blank sandbox in Neo4j.
- Set up a Python environment with necessary packages like neo4j and csv.

3. Designing the NeoClient Class:

The core functionality was encapsulated within a class, NeoClient. This design promoted code modularity and ease of use.

4. Establishing Connection with Neo4j:

- Using the GraphDatabase.driver function from the neo4j library, a connection was initiated.
- Connection credentials (URI, username, and password) were securely stored.

5. CSV Data Parsing and Node Creation:

- For each CSV file, we read and parsed the data.
- A utility function was developed (getTypedValue) to determine the appropriate data type for each column.
- Each row from the CSV was then converted into a node in the Neo4j graph, using the file name as the label and the row's columns as properties.

6. Defining and Creating Relationships:

- Based on the project description, relationships between nodes were outlined.
- Cypher queries were then crafted to create these relationships in the graph, linking nodes together.

7. Testing:

- After node and relationship creation, we executed MATCH queries to retrieve and validate the stored data.
- Ensured that relationships were correctly established and data was accurately represented.

8. Optimizing and Handling Errors:

- Added exception handling to gracefully manage any errors during the data insertion process.
- Checked for potential bottlenecks and optimized the code where necessary, ensuring efficient data insertion.

9. Cypher Querying for Part 3:

- Leveraged the Cypher query language to address the queries provided in Part 3 of the project.
- These queries ranged from simple data retrievals to more complex graph traversals, showcasing the power of the Neo4j graph database.

10. Documentation and Reporting:

- Throughout the project, we maintained detailed documentation to keep track of the process, challenges faced, and solutions implemented.
- Concluded by preparing a comprehensive report detailing the design, approach, and individual contributions.

4. Team Member Contributions

Avinash:

1. **Initial Setup and Data Understanding**
    - Was responsible for initially setting up the Neo4j sandbox and the Python environment.
    - Played a pivotal role in understanding the relationships and structure of the dataset.
2. **NeoClient Design and Connection Handling**
    - Designed the basic framework of the NeoClient class.
    - Implemented the connection handling methods (connectToNeo and closeConnection).
3. **Node Creation from CSVs**
    - Took the lead on parsing CSV data and the node creation process.
    - Implemented the utility function (getTypedValue) to infer data types from CSV.
4. **Testing and Validation**
    - After the nodes and relationships were created, Avinash focused on writing various MATCH queries to validate that data was entered correctly.

Vinod:

1. **Understanding Relationships**
    - Dug deep into understanding potential relationships among the entities (like how Vendor is related to Item and so on).
    - Played a crucial role in defining how these relationships would be represented in the graph.
2. **Relationship Creation in Neo4j**
    - Took the lead on implementing the createRelationships function in the NeoClient class.
    - Wrote the Cypher queries required to establish relationships based on the project's description.
3. **Cypher Querying for Part 3**
    - Vinod focused on addressing the queries provided in Part 3, leveraging the Cypher query language to extract required information.
4. **Error Handling and Optimization**
    - Worked on optimizing the code for better performance.
    - Implemented error handling to manage any issues gracefully during the data insertion process.
5. **Documentation and Reporting**
    - Played a major role in maintaining detailed documentation throughout the project.
    - Assisted in preparing the comprehensive report that detailed the design, approach, and contributions.

Collaborative Efforts:

1. **Joint Brainstorming Sessions:**
    - Vinod and Avinash initiated the project with a series of brainstorming sessions. This was a key moment where both of them chalked out the blueprint of how to approach the data transformation from tabular to graph.
    - The duo regularly convened to identify potential pitfalls, ensuring that they approached problems with a unified vision.
2. **Code Pairing:**
    - To maximize efficiency and reduce redundancy, Vinod and Avinash practiced pair programming. While one wrote the code, the other reviewed it in real-time, ensuring that the logic was sound and free of errors.
    - This collaborative technique not only led to cleaner code but also served as an excellent knowledge exchange, ensuring both were on the same page.
3. **Database Design and Integration:**
    - Vinod's deep understanding of relationships was complemented by Avinash's knack for data representation. They worked collaboratively on schema design, ensuring the graph database was reflective of the relational data's integrity.
    - Avinash took the lead in creating nodes, while Vinod focused on relationships, ensuring both tasks were handled simultaneously and seamlessly integrated.
4. **Debugging Sessions:**
    - Like any software project, the duo encountered their share of bugs. These were often addressed in tandem. While Vinod provided insights into the relational aspects and potential data mismatches, Avinash deep-dived into the code logic, ensuring bugs were squashed efficiently.
5. **Documentation and Reporting:**
    - Recognizing the importance of clear documentation, both members collaborated on this front. Vinod documented the Cypher query specifics and the logic behind them, while Avinash detailed the implementation and the overall program design.
    - They both reviewed each other's contributions, ensuring the documentation was comprehensive and free from any oversight.
6. **Feedback & Iterative Improvement:**
    - Vinod and Avinash maintained an open channel for feedback. After completing major milestones, they would sit down for a feedback session, evaluating their progress and identifying areas of improvement.
    - These sessions were invaluable, ensuring that the project was always on the right track and continually evolving.

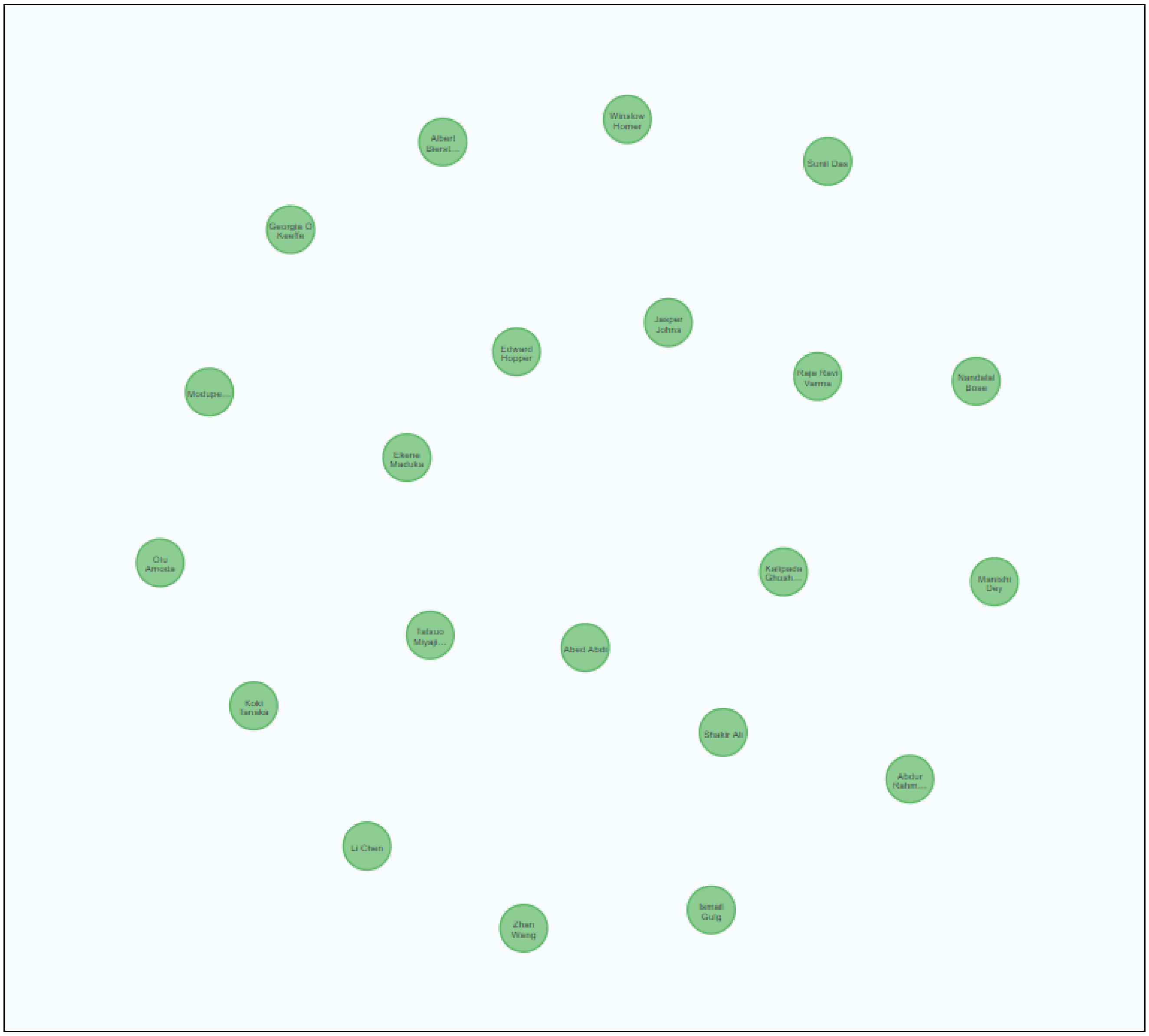
5. Answers to Part 2

NODES:-

CUSTOMER

```
// CUSTOMER node creation
CREATE (customer:CUSTOMER {
  cId: {cIdValue},
  Cname: {CnameValue},
  Street: {StreetValue},
  City: {CityValue},
  StateAb: {StateAbValue},
  Zipcode: {ZipcodeValue}
})
```

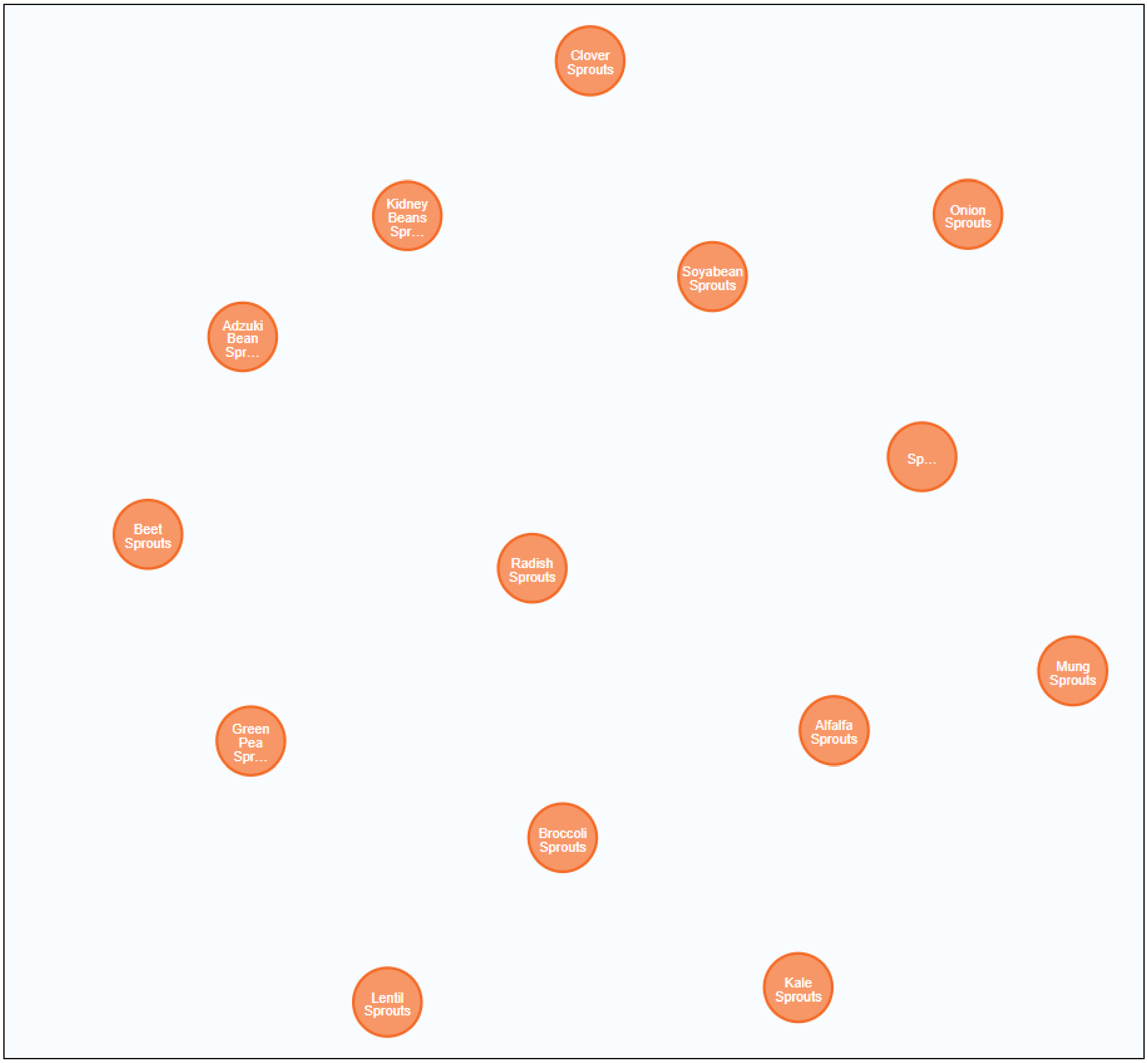
```
MATCH (n:CUSTOMER) RETURN n
```



ITEM

```
// ITEM node creation
CREATE (item:ITEM {
  iId: {iIdValue},
  Iname: {InameValue},
  Sprice: {SpriceValue}
})
```

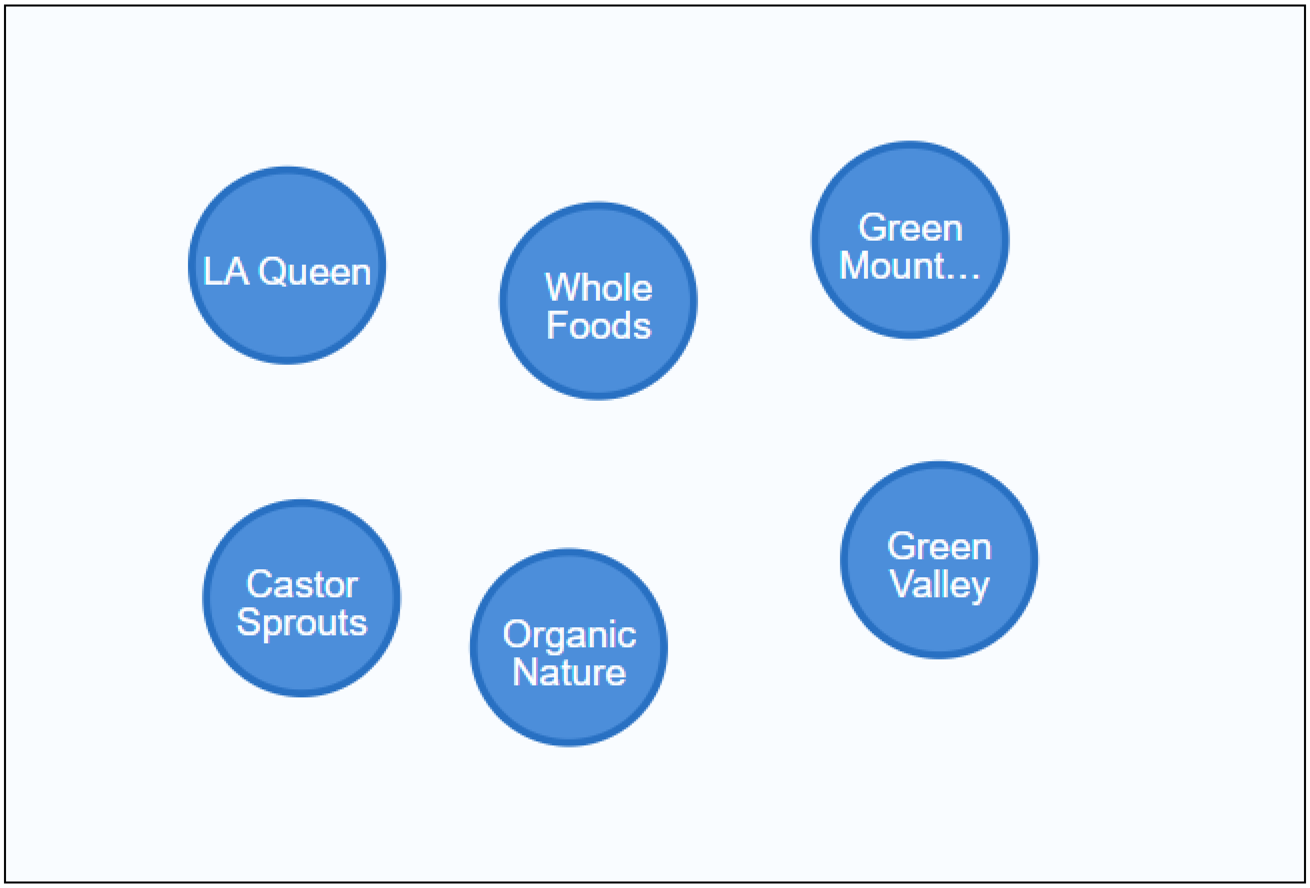
```
MATCH (n:ITEM) RETURN n
```



VENDOR

```
// VENDOR node creation
CREATE (vendor:VENDOR {
  vId: {vIdValue},
  Vname: {VnameValue},
  Street: {StreetValue},
  City: {CityValue},
  StateAb: {StateAbValue},
  ZipCode: {ZipCodeValue}
})
```

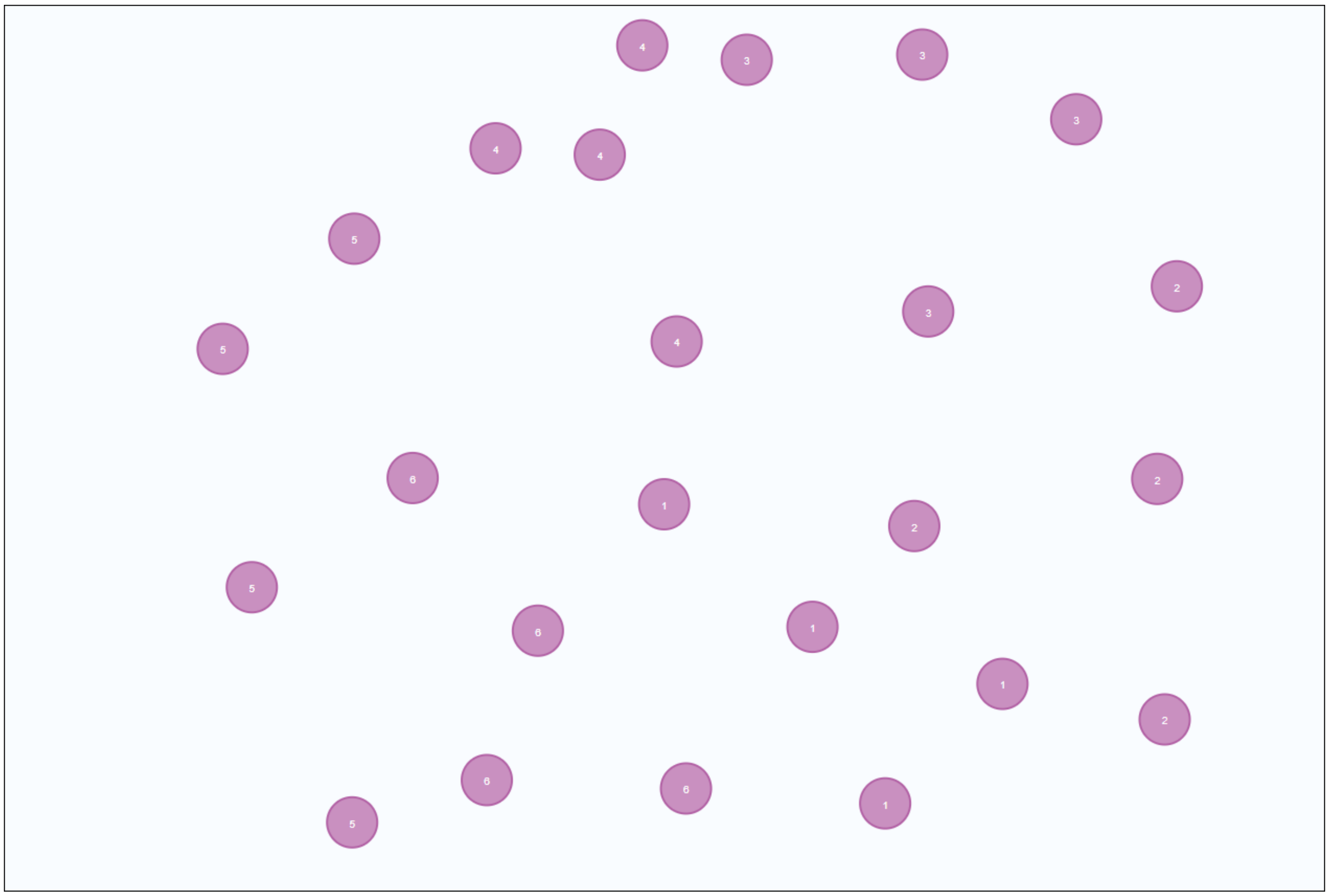
```
MATCH (n:VENDOR) RETURN n
```



VENDOR_ITEM

```
// VENDOR_ITEM node creation
CREATE (vendorItem:VENDOR_ITEM {
  vId: {vIdValue},
  iId: {iIdValue},
  Vprice: {VpriceValue}
})
```

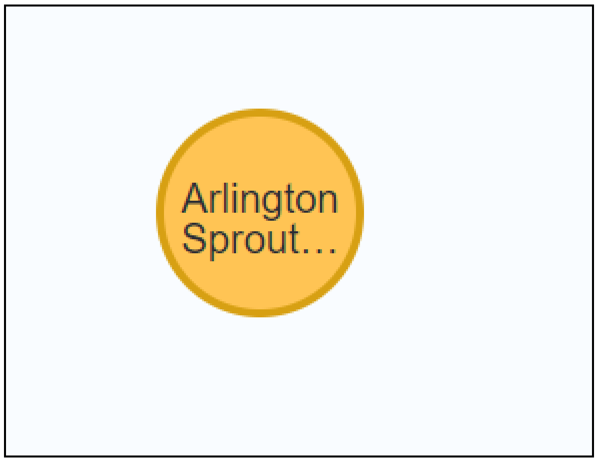
```
MATCH (n:VENDOR_ITEM) RETURN n
```



STORE

```
// STORE node creation
CREATE (store:STORE {
  sId: {sIdValue},
  Sname: {SnameValue},
  Street: {StreetValue},
  City: {CityValue},
  StateAb: {StateAbValue},
  ZipCode: {ZipCodeValue},
  Sdate: {SdateValue},
  Telno: {TelnoValue},
  URL: {URLValue}
})
```

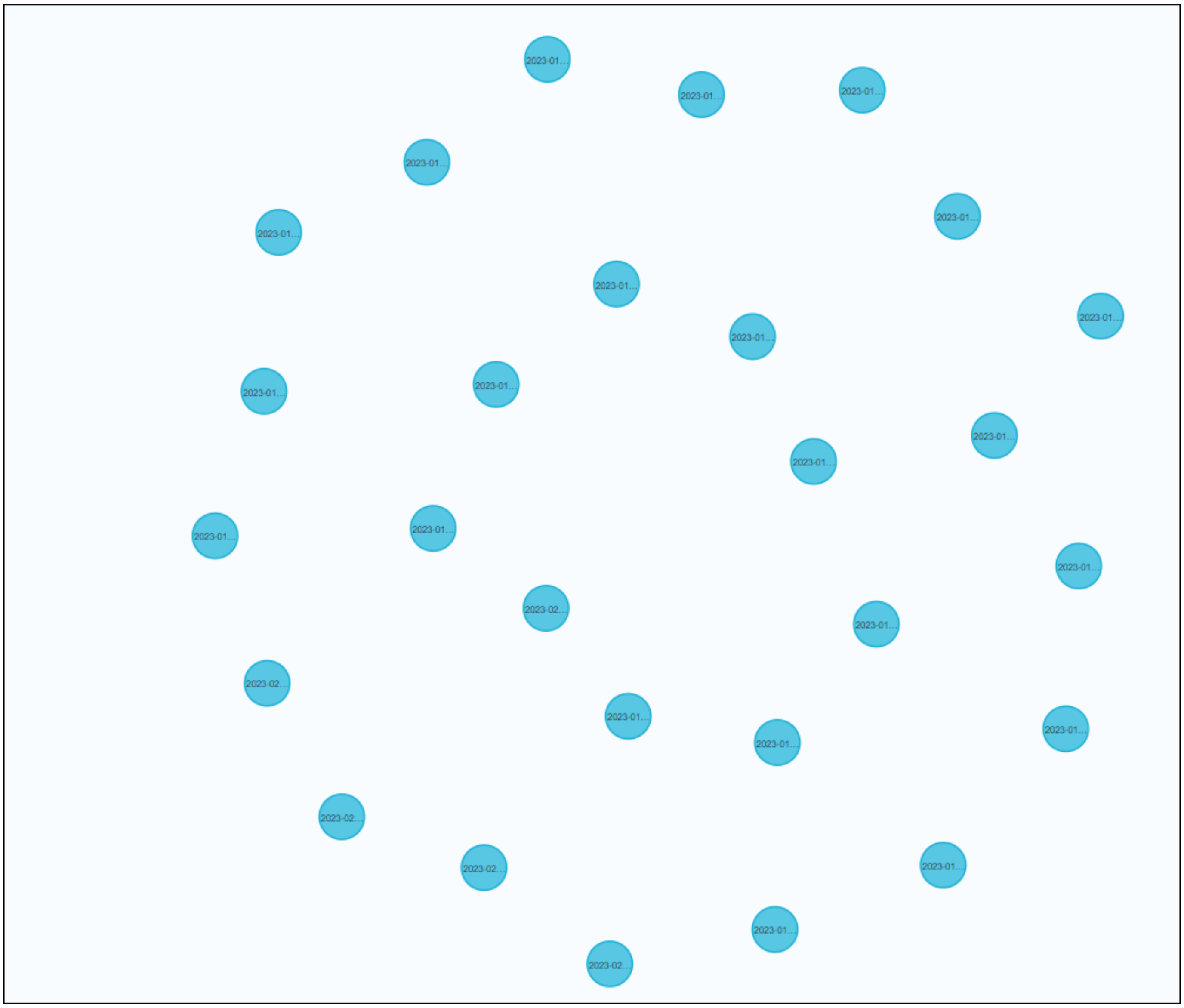
```
MATCH (n:STORE) RETURN n
```



ORDERS

```
// ORDERS node creation
CREATE (order:ORDERS {
  oId: {oIdValue},
  sId: {sIdValue},
  cId: {cIdValue},
  Odate: {OdateValue},
  Ddate: {DdateValue},
  Amount: {AmountValue}
})
```

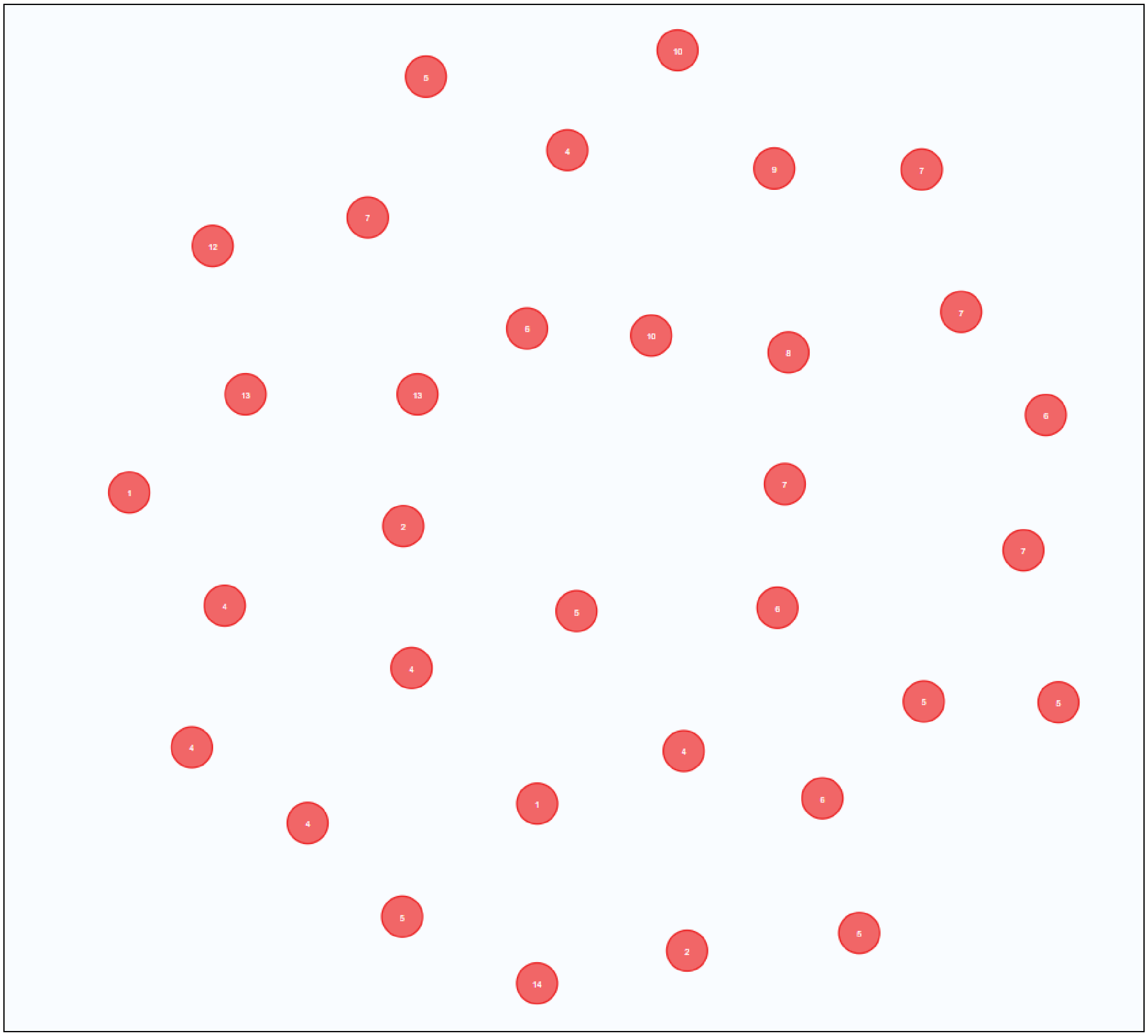
```
MATCH (n:ORDERS) RETURN n
```



ORDER_ITEM

```
// ORDER_ITEM node creation
CREATE (orderItem:ORDER_ITEM {
  oId: {oIdValue},
  iId: {iIdValue},
  Icount: {IcountValue}
})
```

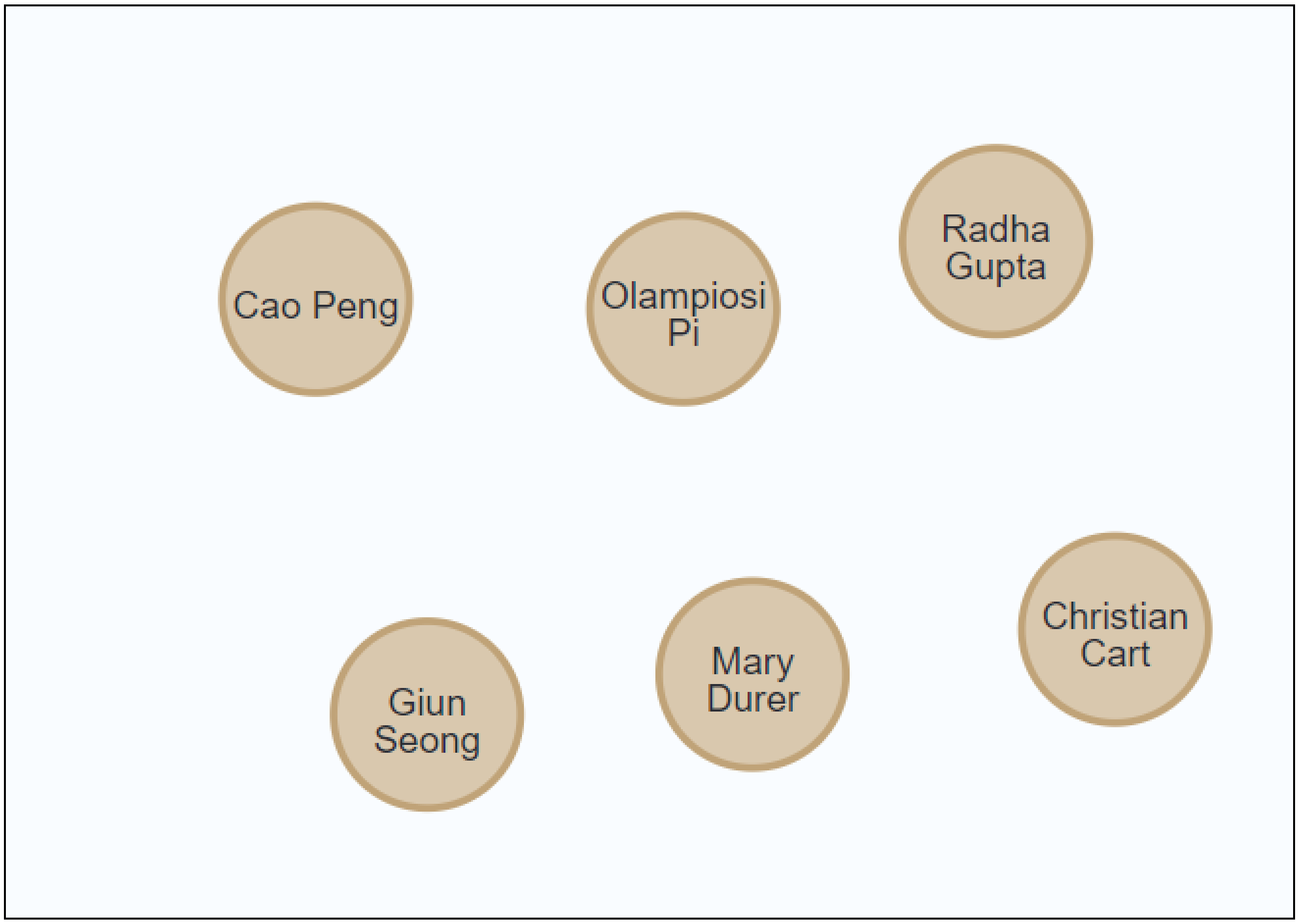
```
MATCH (n:ORDER_ITEM) RETURN n
```



EMPLOYEE

```
// EMPLOYEE node creation
CREATE (employee:EMPLOYEE {
  sId: {sIdValue},
  SSN: {SSNValue},
  Sname: {SnameValue},
  Street: {StreetValue},
  City: {CityValue},
  StateAb: {StateAbValue},
  Etype: {EtypeValue},
  Bdate: {BdateValue},
  Sdate: {SdateValue},
  Edate: {EdateValue},
  Level: {LevelValue},
  Asalary: {AsalaryValue},
  Agency: {AgencyValue},
  Hsalary: {HsalaryValue},
  Institute: {InstituteValue},
  Itype: {ItypeValue}
})
```

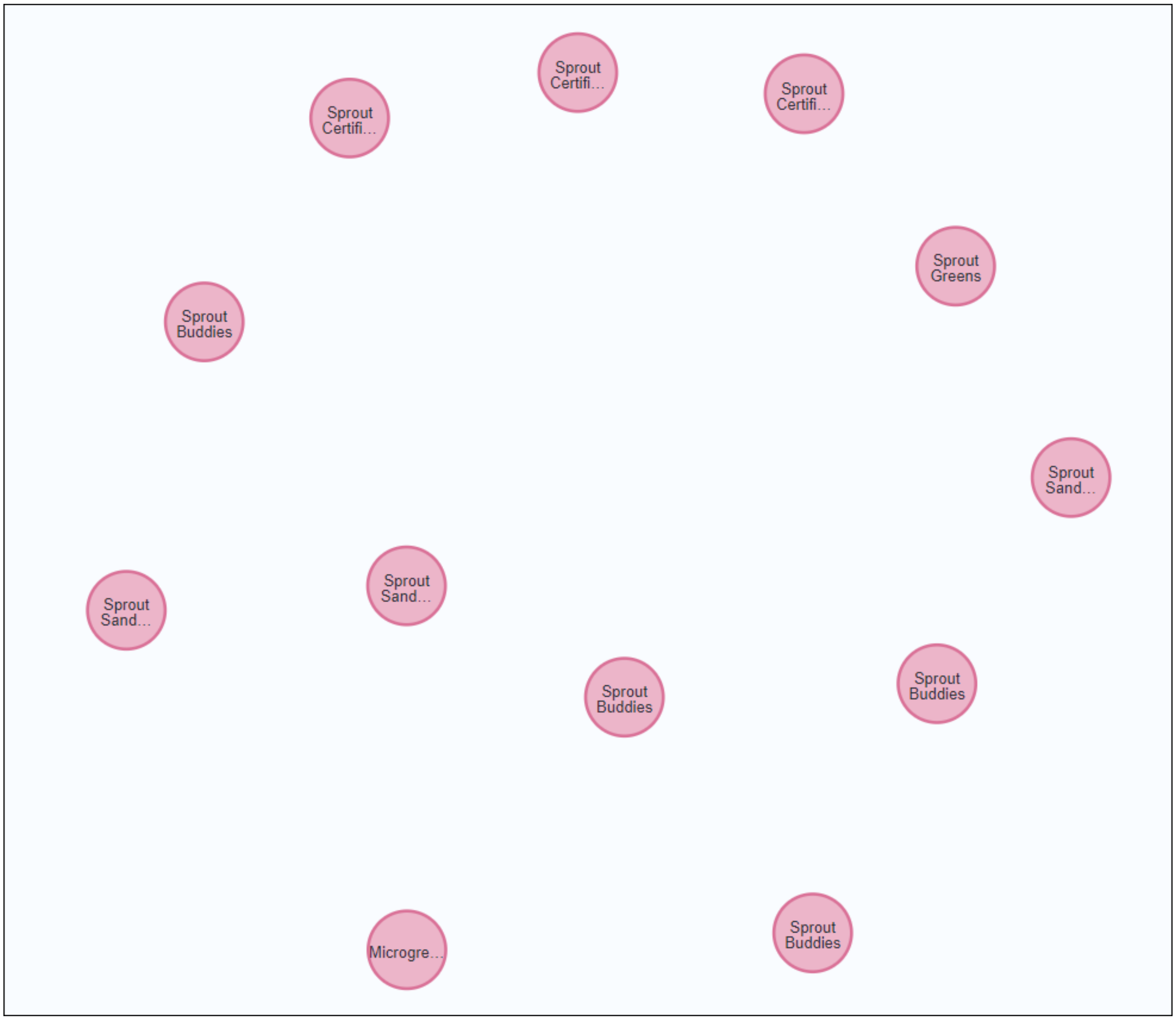
```
MATCH (n:EMPLOYEE) RETURN n
```



CONTRACT

```
// CONTRACT node creation
CREATE (contract:CONTRACT {
  vId: {vIdValue},
  ctId: {ctIdValue},
  Sdate: {SdateValue},
  Ctime: {CtimeValue},
  Cname: {CnameValue}
})
```

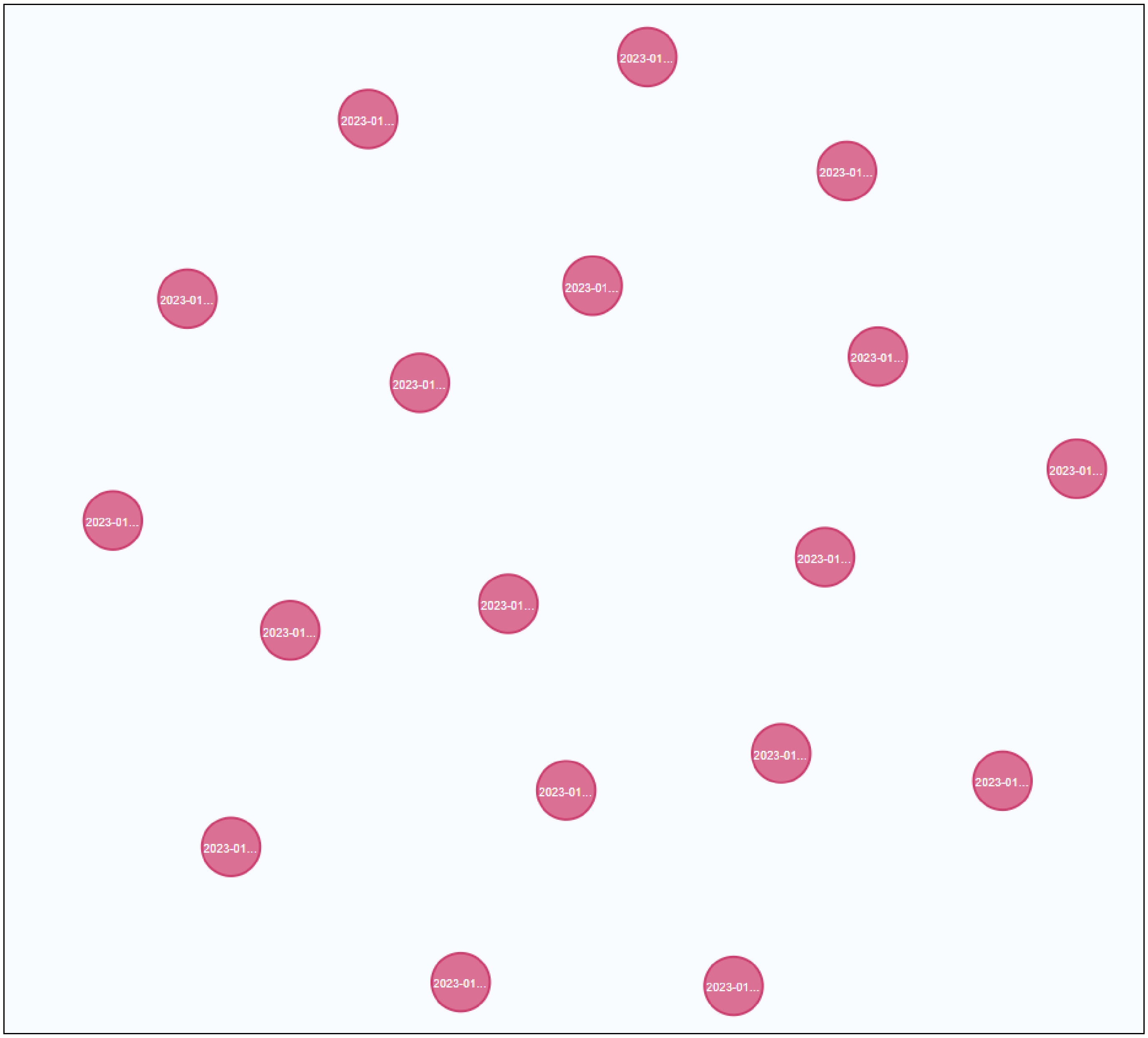
```
MATCH (n:CONTRACT) RETURN n
```



OLDPRICE

```
// OLDPRICE node creation
CREATE (oldprice:OLDPRICE {
  iId: {iIdValue},
  Sprice: {SpriceValue},
  Sdate: {SdateValue},
  Edate: {EdateValue}
})
```

```
MATCH (n:OLDPRICE) RETURN n
```

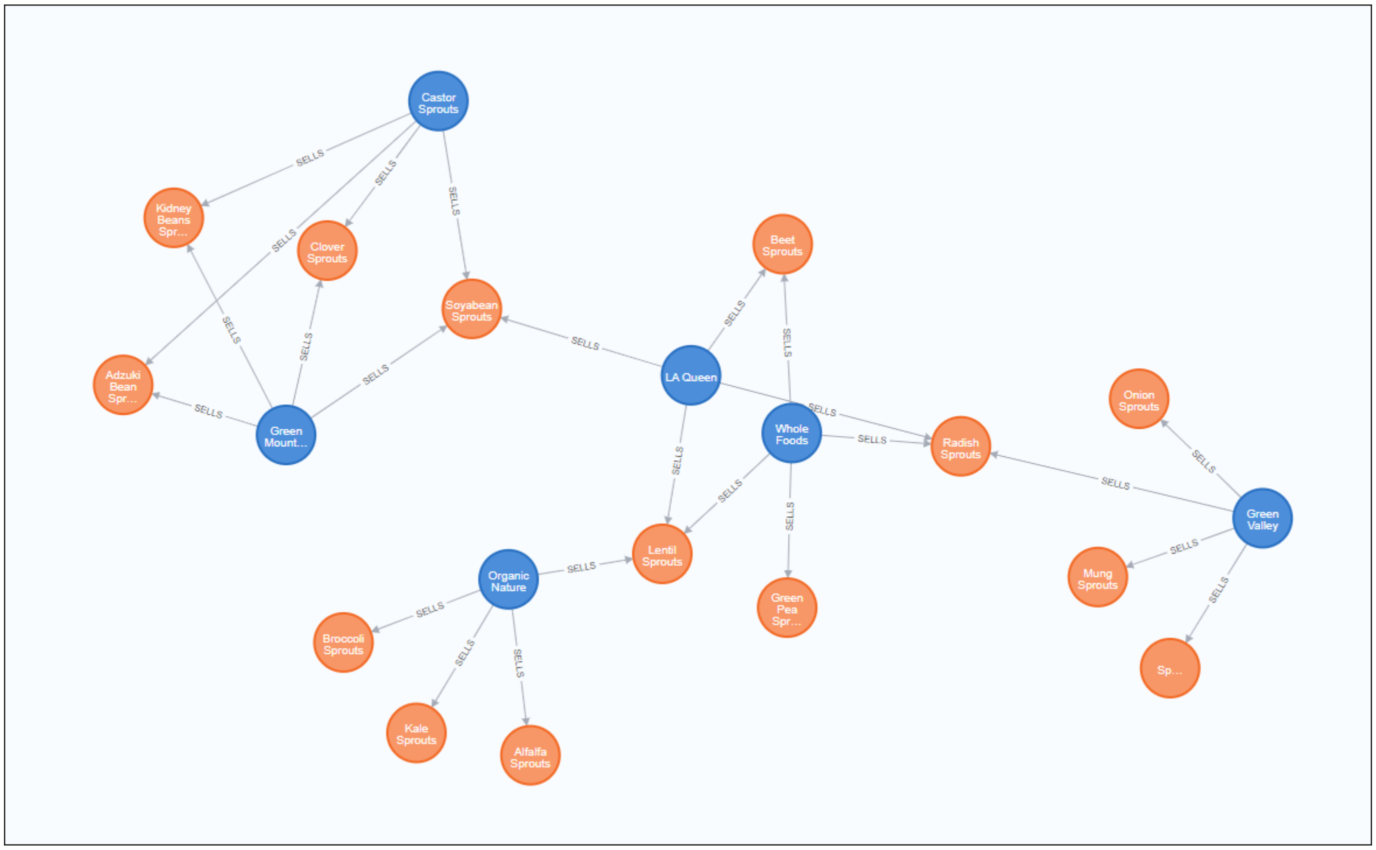


RELATIONSHIPS:

i. Vendor SELLS Item

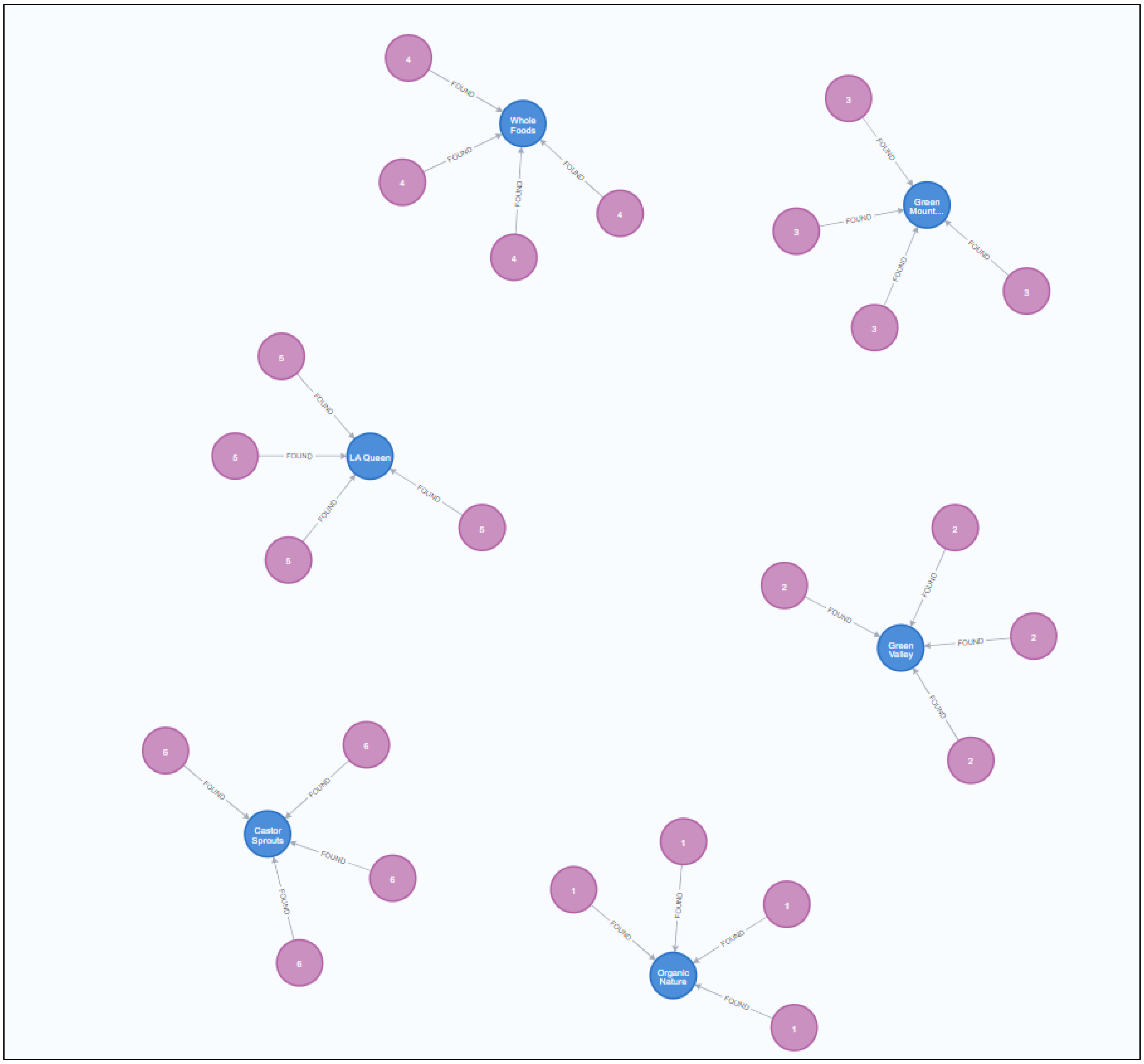
```
      MATCH (vi:VENDOR_ITEM) WITH vi.vId AS vendorId, vi.iId AS itemId MATCH
(v:VENDOR {vId: vendorId}) MATCH (i:ITEM {iId: itemId}) CREATE (v)-[:SELLS]->(i)
```

```
MATCH p=()-[r:SELLS]->() RETURN p
```



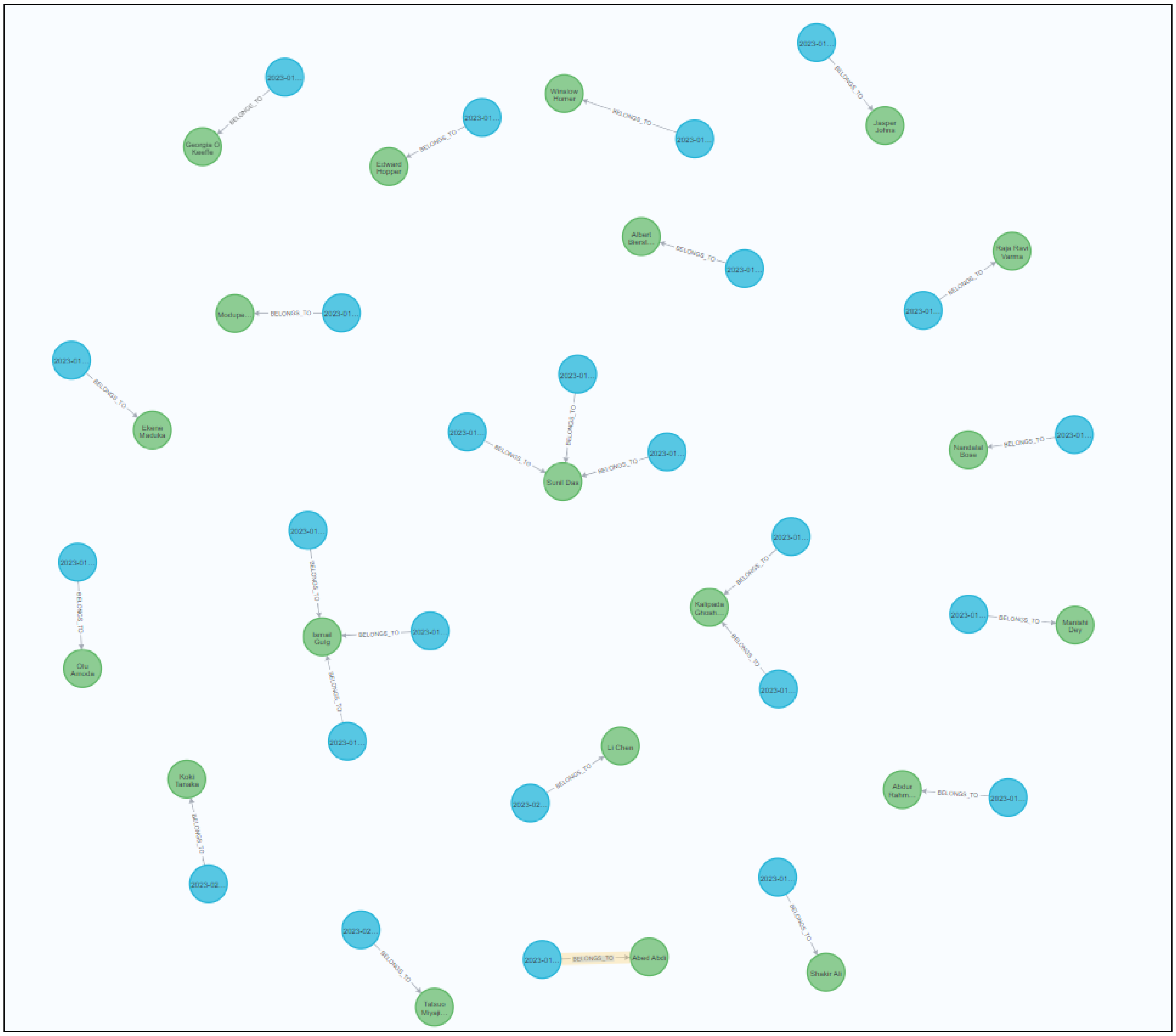
ii. Vendor_Item FOUND_IN Vendor

```
MATCH (vi:VENDOR_ITEM),(v:VENDOR) WHERE vi.vId=v.vId
CREATE(vi)-[found:FOUND]->(v)
MATCH p=()-[r:FOUND]->() RETURN p
```



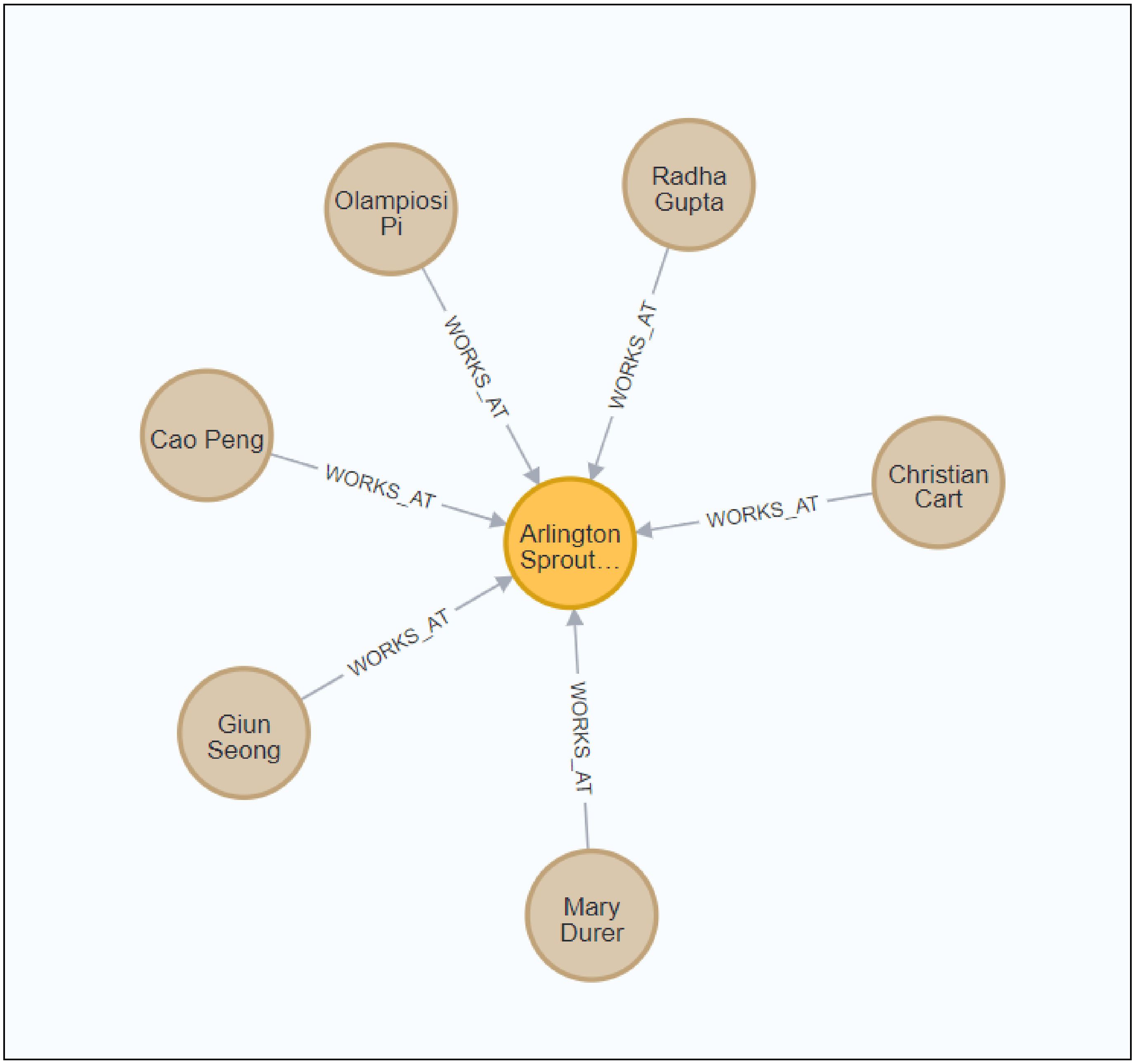
iii. Order BELONGS_TO Customer

```
MATCH (o:ORDERS),(c:CUSTOMER) WHERE o.cId=c.cId
CREATE(o)-[belongs_to:BELONGS_TO]->(c)
MATCH p=()-[r:BELONGS_TO]->() RETURN p
```



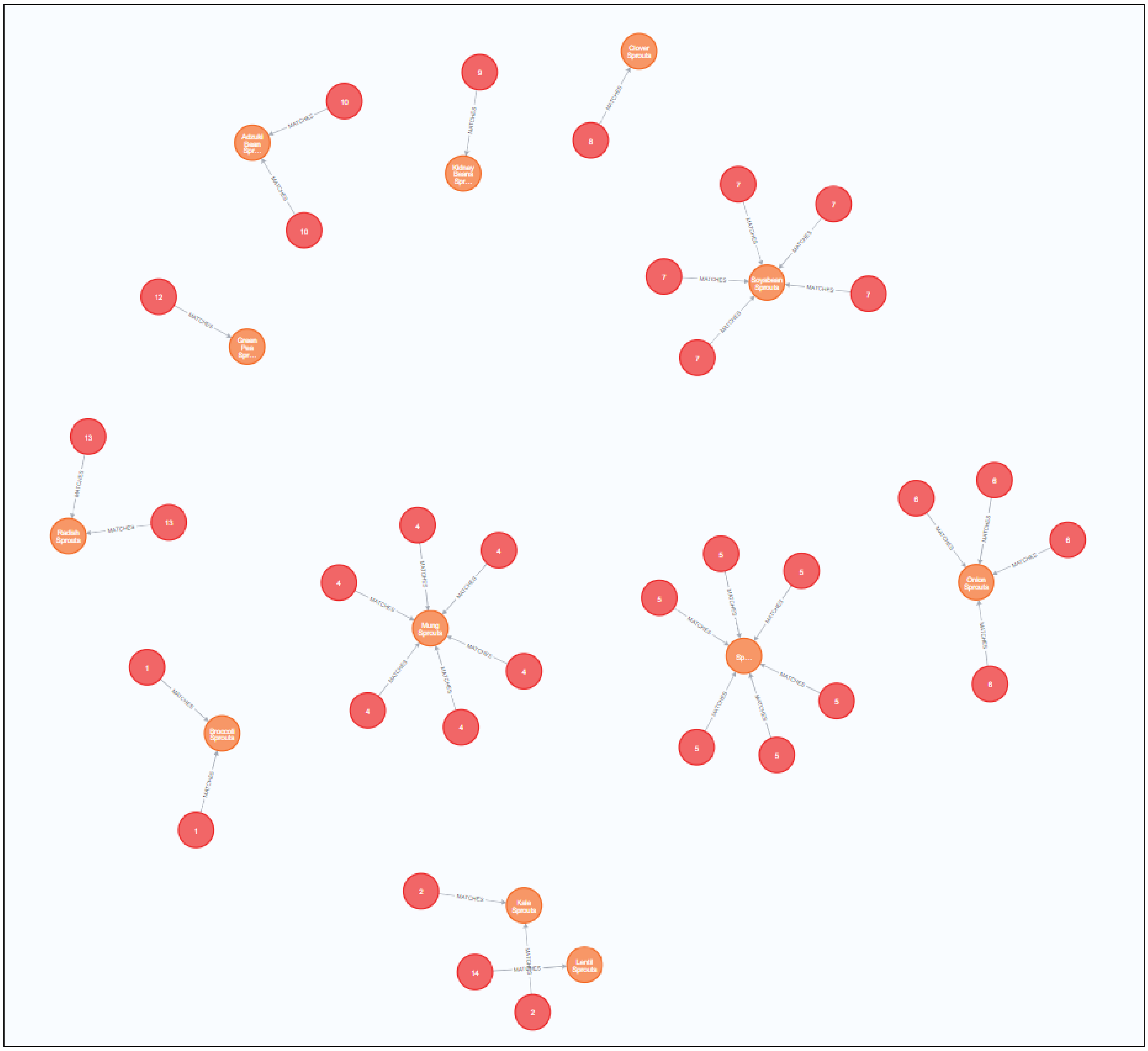
iv. Order CONTAINS Item

```
MATCH (oi:ORDER_ITEM) WITH oi.oId AS orderId, oi.iId AS itemId MATCH (o:ORDERS
{oId: orderId}) MATCH (i:ITEM {iId: itemId}) CREATE (o)-[:CONTAINS]->(i)
MATCH p=()-[r:WORKS_AT]->() RETURN p
```



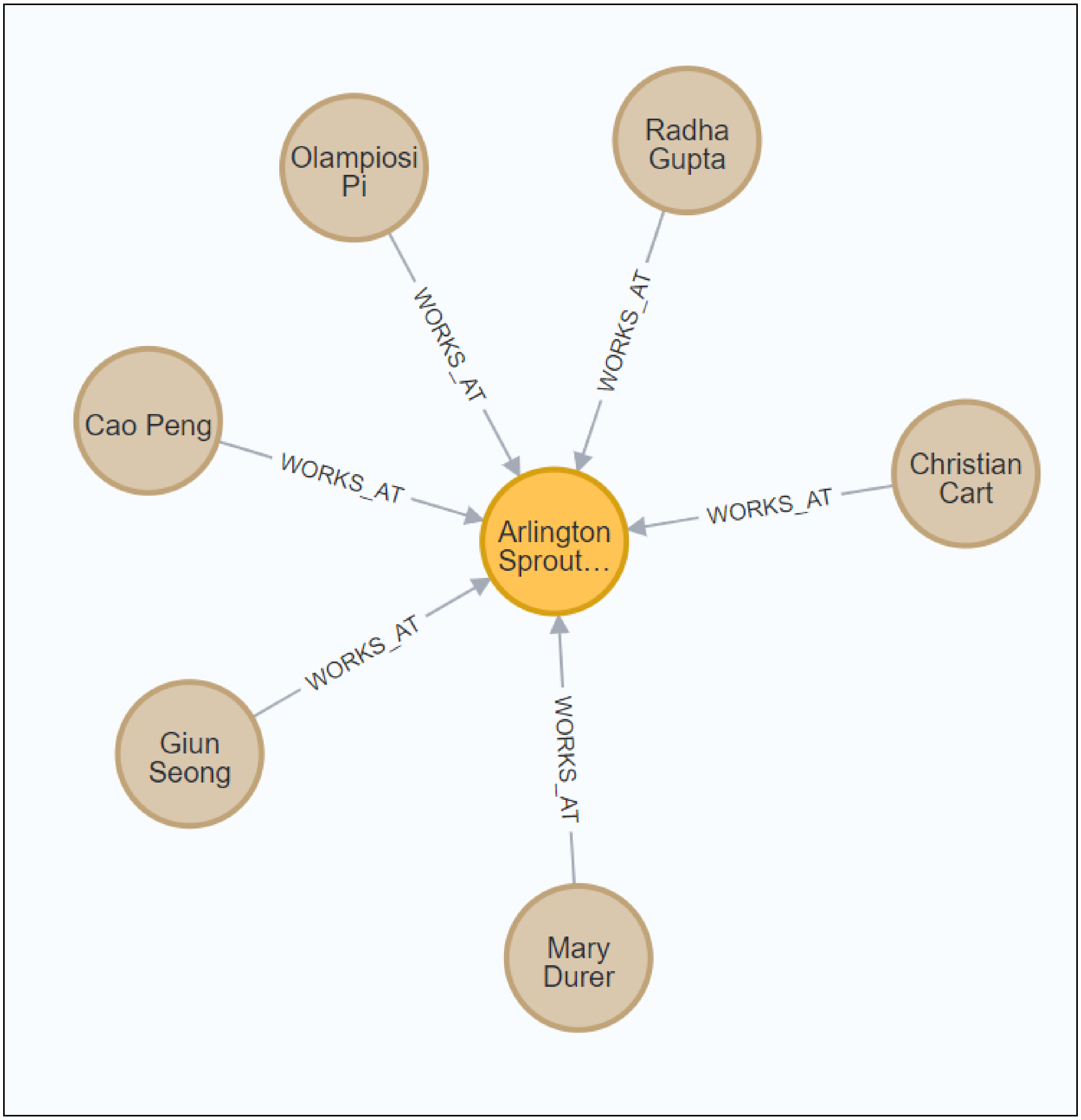
v. Order_Item MATCHES Item

```
MATCH (oi:ORDER_ITEM),(i:ITEM) WHERE oi.iId=i.iId
CREATE(oi)-[matches:MATCHES]->(i)
MATCH p=()-[r:MATCHES]->() RETURN p
```



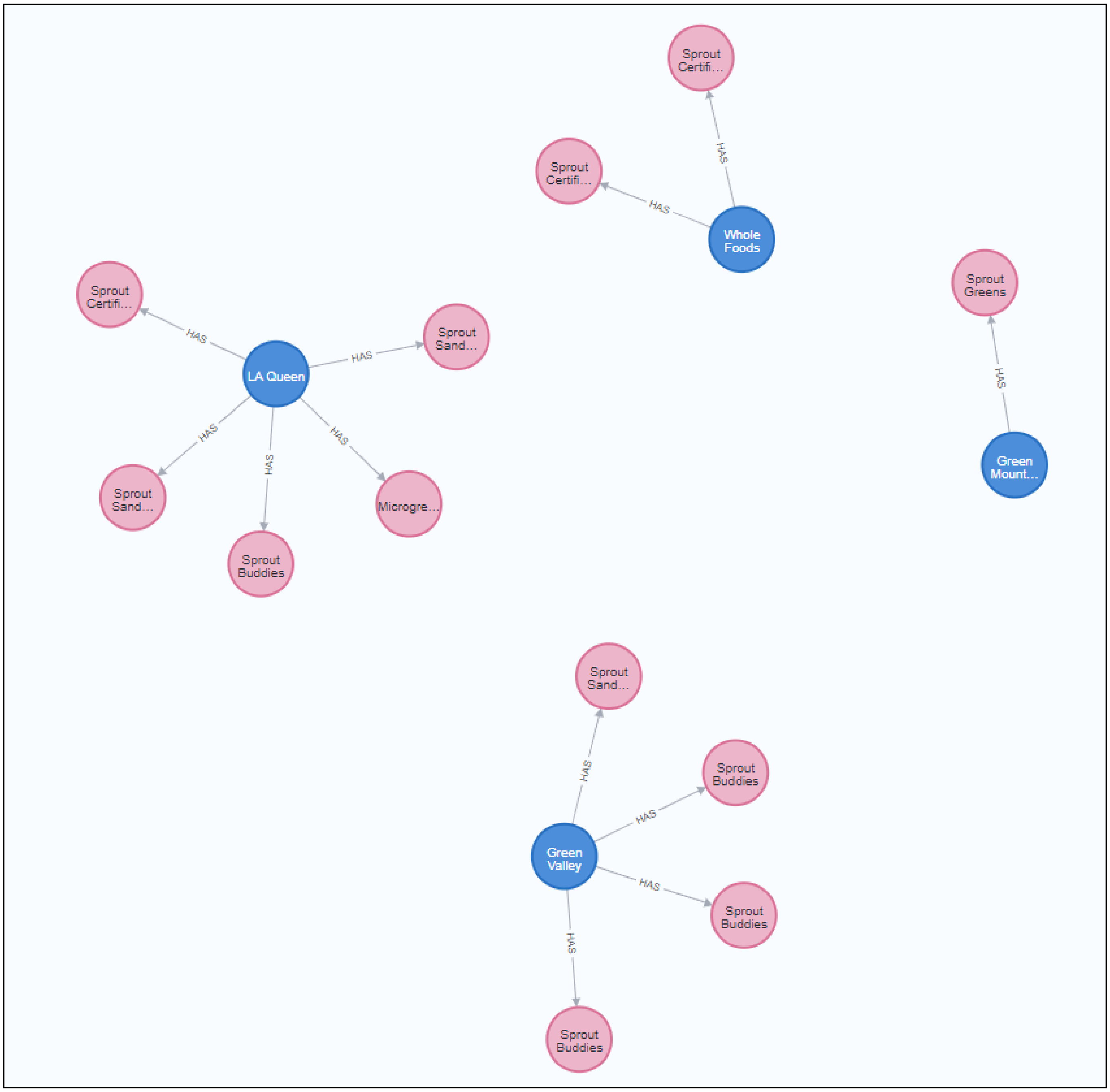
vi. Employee WORKS_AT Store

```
MATCH (e:EMPLOYEE),(s:STORE) WHERE e.sId=s.sId CREATE(e)-[works_at:WORKS_AT]->(s)
MATCH p=()-[r:WORKS_AT]->() RETURN p
```



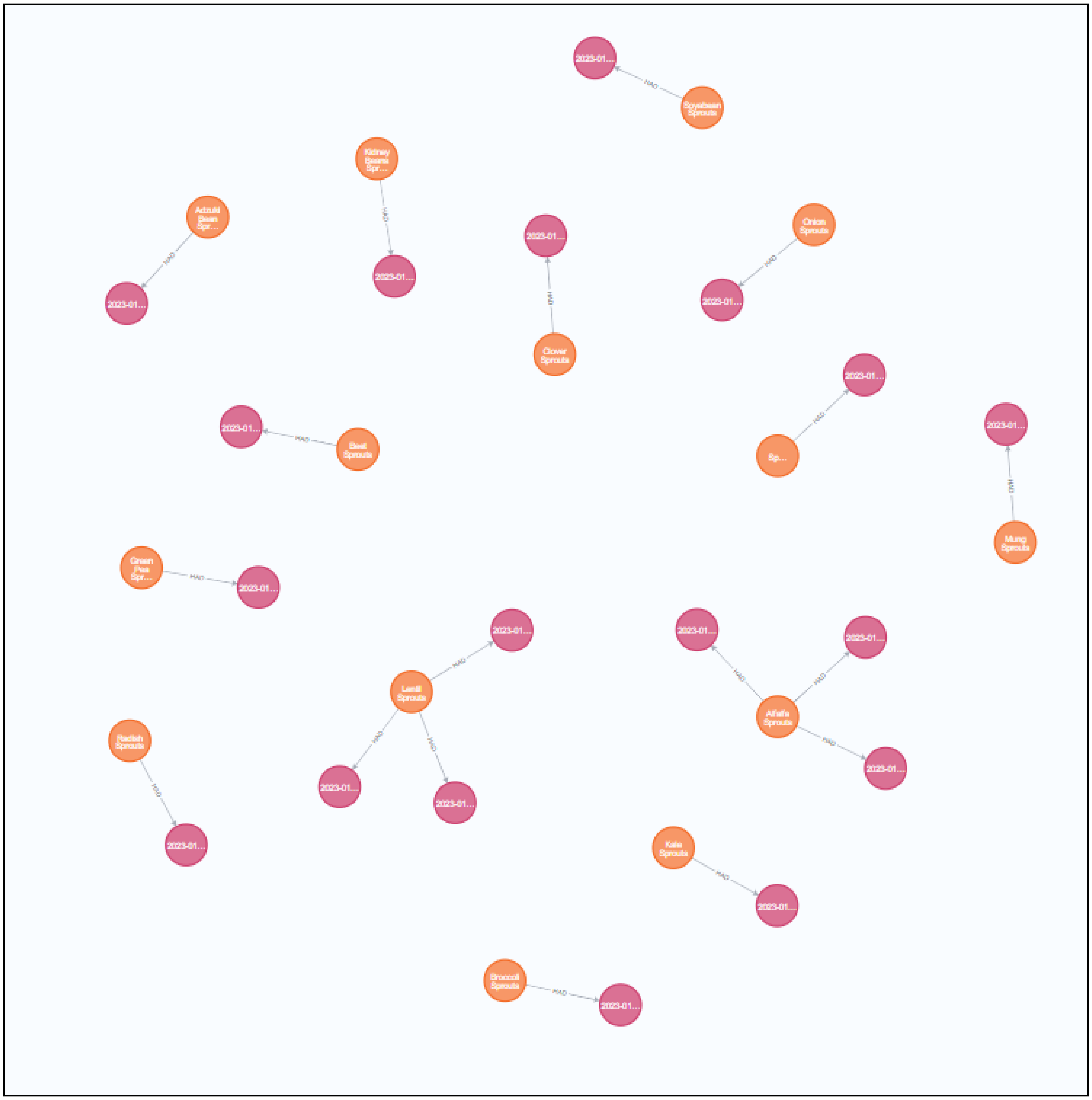
vii. Vendor HAS Contract

```
MATCH (v:VENDOR),(ct:CONTRACT) WHERE v.vId=ct.vId CREATE(v)-[has:HAS]->(ct)
MATCH p=()-[r:HAS]->() RETURN p
```



viii. Item HAD Oldprice

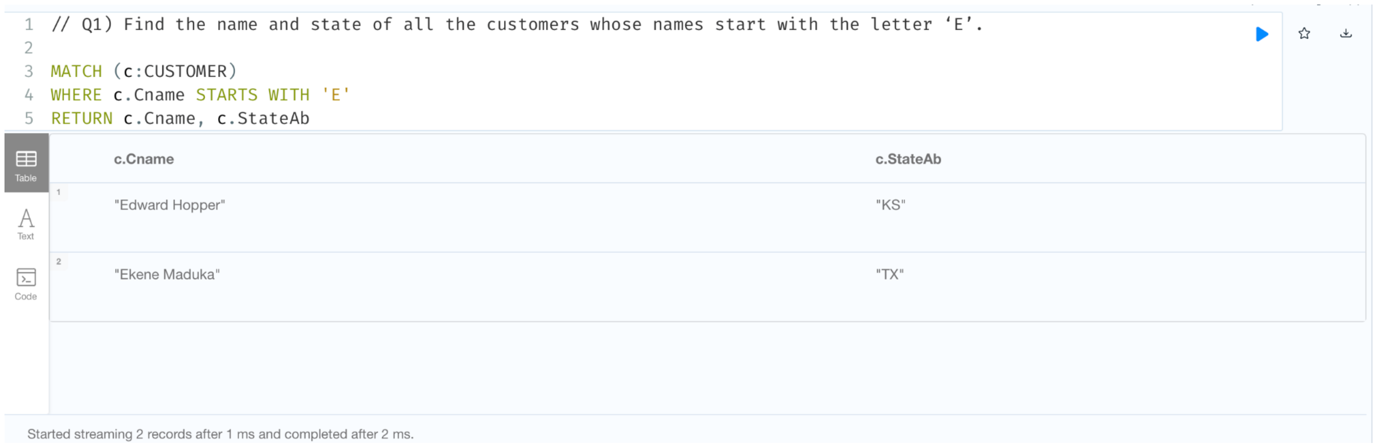
```
MATCH (i:ITEM),(op:OLDPRICE) WHERE i.iId=op.iId CREATE(i)-[had:HAD]->(op)
MATCH p=()-[r:HAD]->() RETURN p
```



6. Answers to Part 3

Q1) Find the name and state of all the customers whose names start with the letter ‘E’.

```
MATCH (c:CUSTOMER)
WHERE c.Cname STARTS WITH 'E'
RETURN c.Cname, c.StateAb
```



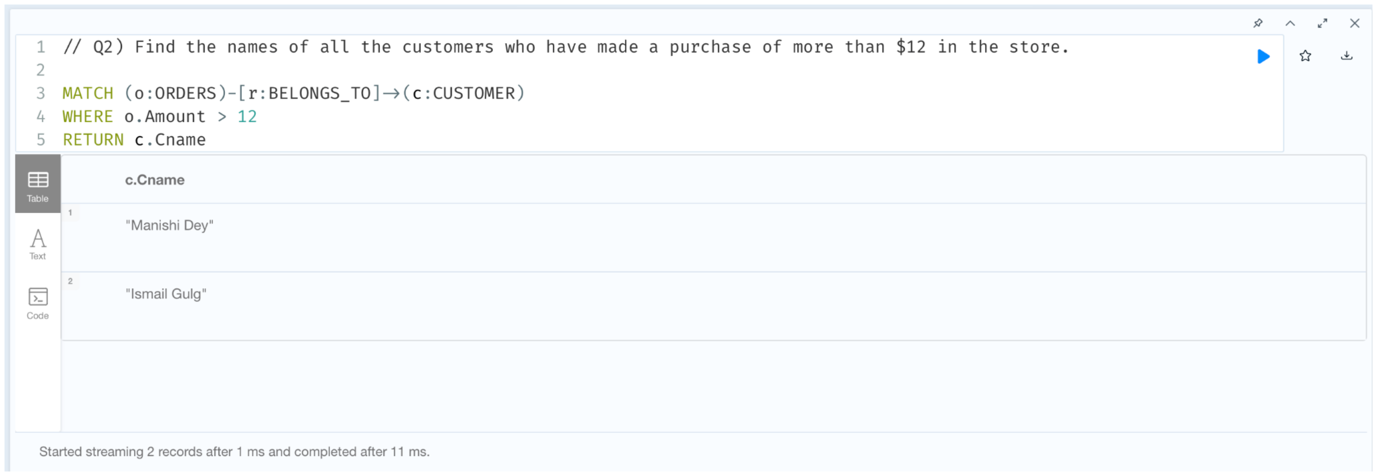
```
// Q1) Find the name and state of all the customers whose names start with the letter ‘E’.

MATCH (c:CUSTOMER)
WHERE c.Cname STARTS WITH 'E'
RETURN c.Cname, c.StateAb
```

	c.Cname	c.StateAb
1	"Edward Hopper"	"KS"
2	"Ekene Maduka"	"TX"

Started streaming 2 records after 1 ms and completed after 2 ms.

Q2) Find the names of all the customers who have made a purchase of more than $12 in the store.

```
MATCH (o:ORDERS)-[r:BELONGS_TO]->(c:CUSTOMER)
WHERE o.Amount > 12
RETURN c.Cname
```



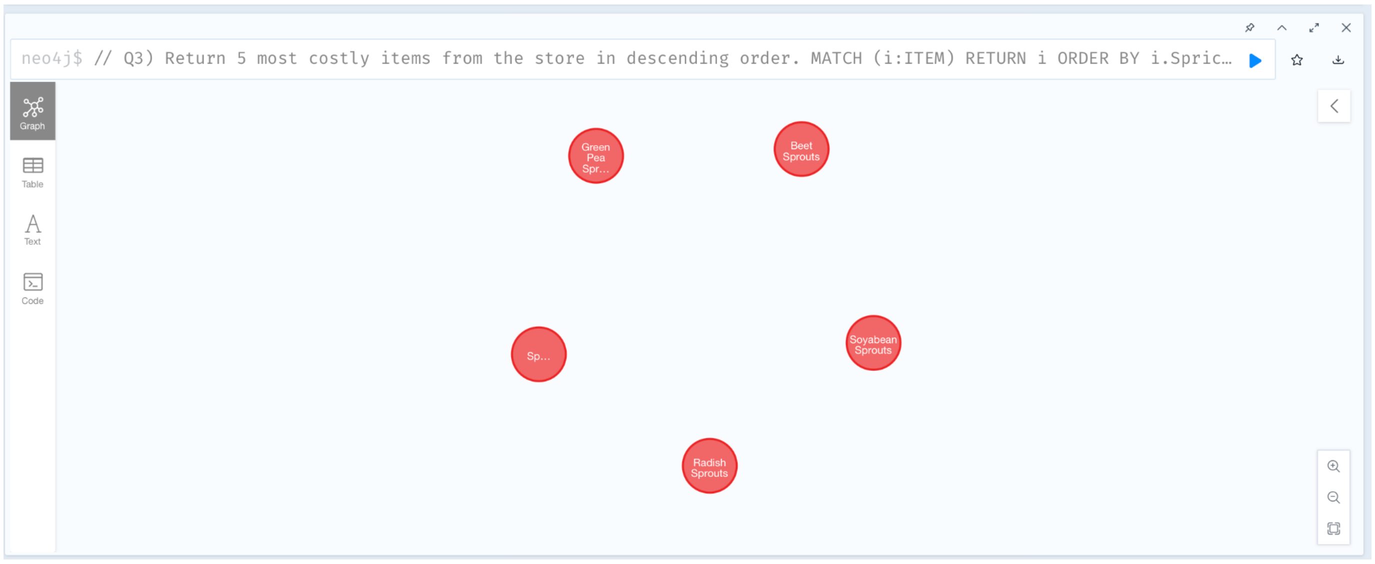
```
// Q2) Find the names of all the customers who have made a purchase of more than $12 in the store.

MATCH (o:ORDERS)-[r:BELONGS_TO]→(c:CUSTOMER)
WHERE o.Amount > 12
RETURN c.Cname
```

	c.Cname
1	"Manishi Dey"
2	"Ismail Gulg"

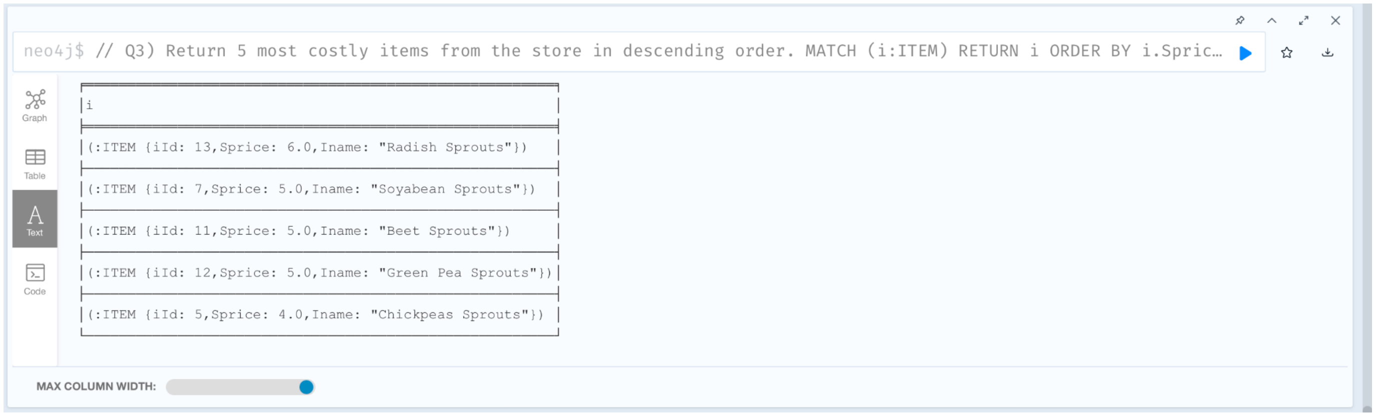
Started streaming 2 records after 1 ms and completed after 11 ms.

Q3) Return 5 most costly items from the store in descending order.

```
MATCH (i:ITEM)
RETURN i
ORDER BY i.Sprice DESC
LIMIT 5
```



```
neo4j$ // Q3) Return 5 most costly items from the store in descending order. MATCH (i:ITEM) RETURN i ORDER BY i.Spric…
```

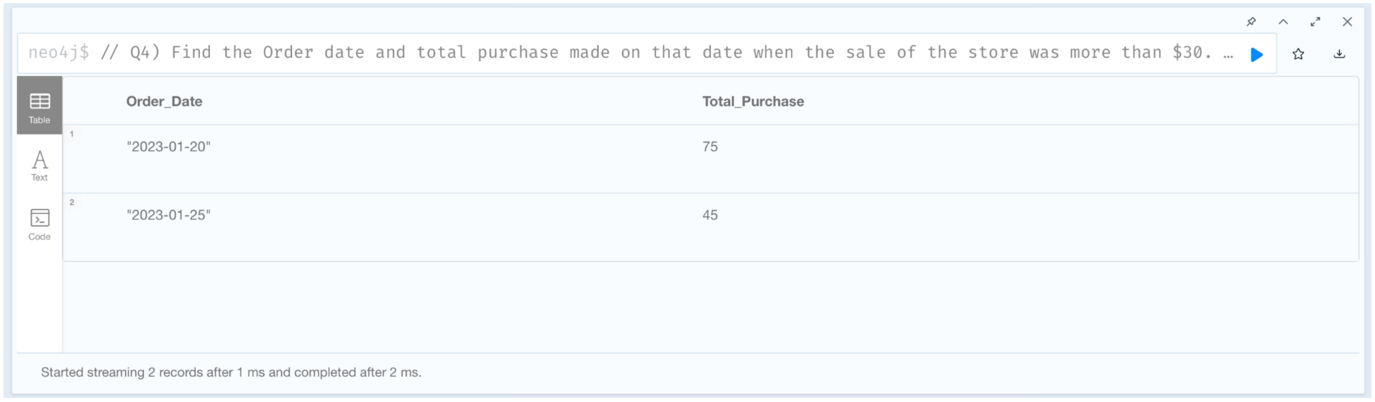


```
neo4j$ // Q3) Return 5 most costly items from the store in descending order. MATCH (i:ITEM) RETURN i ORDER BY i.Spric…
```

```
╒══════════════════════════════════════════════════════════╕
│i                                                         │
╞══════════════════════════════════════════════════════════╡
│(:ITEM {iId: 13,Sprice: 6.0,Iname: "Radish Sprouts"})     │
├──────────────────────────────────────────────────────────┤
│(:ITEM {iId: 7,Sprice: 5.0,Iname: "Soyabean Sprouts"})    │
├──────────────────────────────────────────────────────────┤
│(:ITEM {iId: 11,Sprice: 5.0,Iname: "Beet Sprouts"})       │
├──────────────────────────────────────────────────────────┤
│(:ITEM {iId: 12,Sprice: 5.0,Iname: "Green Pea Sprouts"})  │
├──────────────────────────────────────────────────────────┤
│(:ITEM {iId: 5,Sprice: 4.0,Iname: "Chickpeas Sprouts"})   │
└──────────────────────────────────────────────────────────┘
```

Q4) Find the Order date and total purchase made on that date when the sale of the store was more than $30.

```
MATCH (o:ORDERS)
WITH o.Odate as Order_Date, sum(o.Amount) as Total_Purchase
WHERE Total_Purchase > 30
RETURN Order_Date, Total_Purchase
```



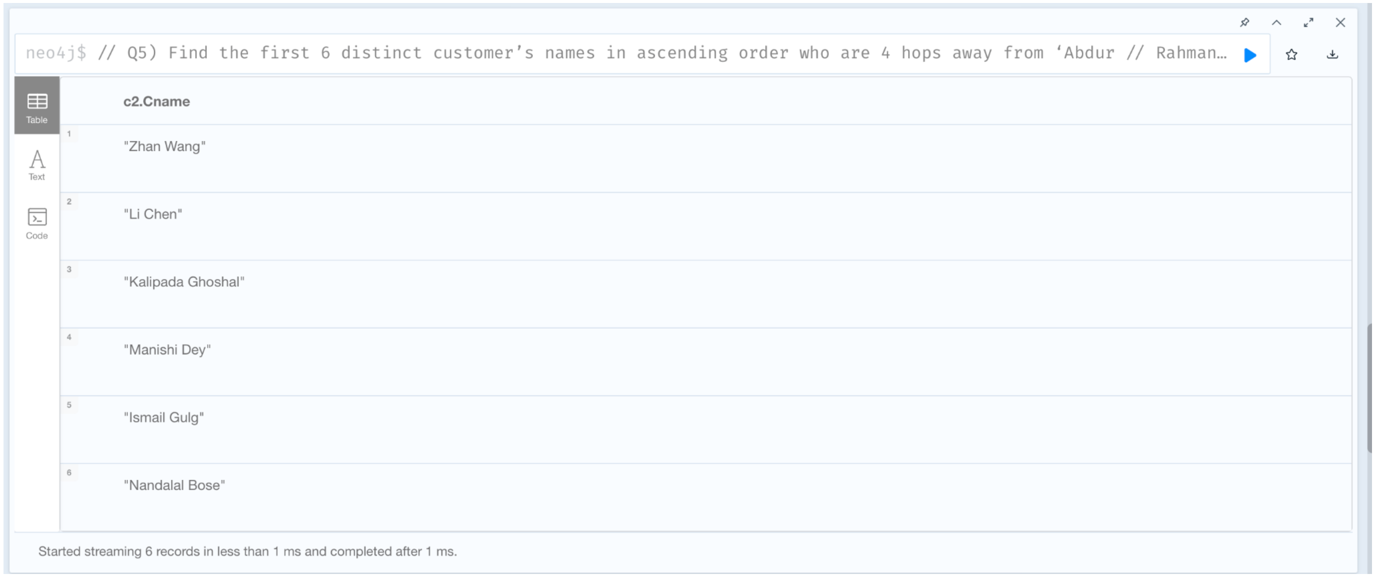
```
neo4j$ // Q4) Find the Order date and total purchase made on that date when the sale of the store was more than $30. …
```

	Order_Date	Total_Purchase
1	"2023-01-20"	75
2	"2023-01-25"	45

Started streaming 2 records after 1 ms and completed after 2 ms.

Q5) Find the first 6 distinct customer's names in ascending order who are 4 hops away from 'Abdur Rahman'.

```
MATCH path = (c1:CUSTOMER {Cname: 'Abdur Rahman'})-[*4]-(c2:CUSTOMER)
RETURN DISTINCT c2.Cname
LIMIT 6
```



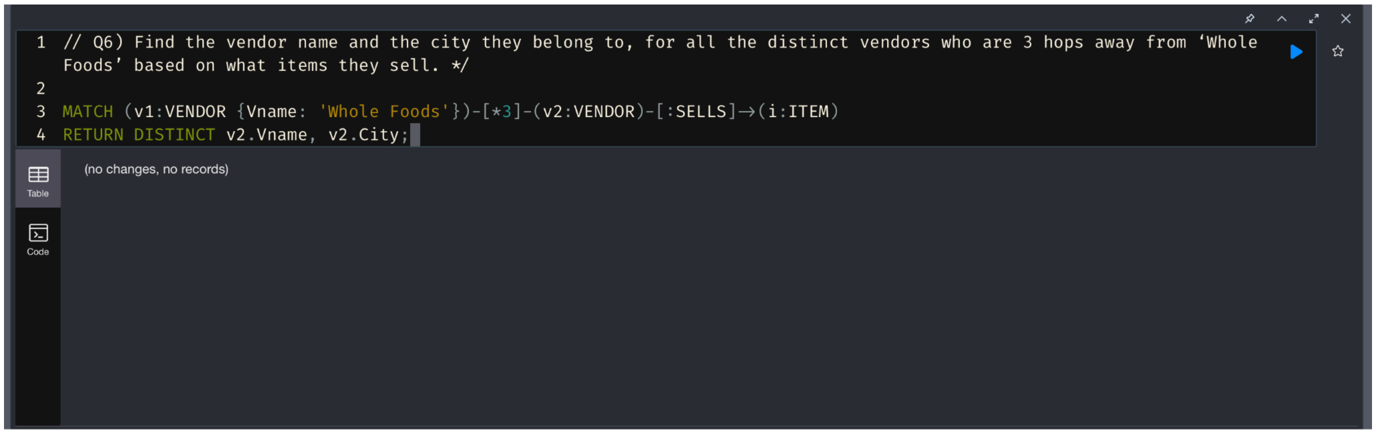
```
neo4j$ // Q5) Find the first 6 distinct customer's names in ascending order who are 4 hops away from 'Abdur // Rahman…
```

	c2.Cname
1	"Zhan Wang"
2	"Li Chen"
3	"Kalipada Ghoshal"
4	"Manishi Dey"
5	"Ismail Gulg"
6	"Nandalal Bose"

Started streaming 6 records in less than 1 ms and completed after 1 ms.

Q6) Find the vendor name and the city they belong to, for all the distinct vendors who are 3 hops away from ‘Whole Foods’ based on what items they sell.

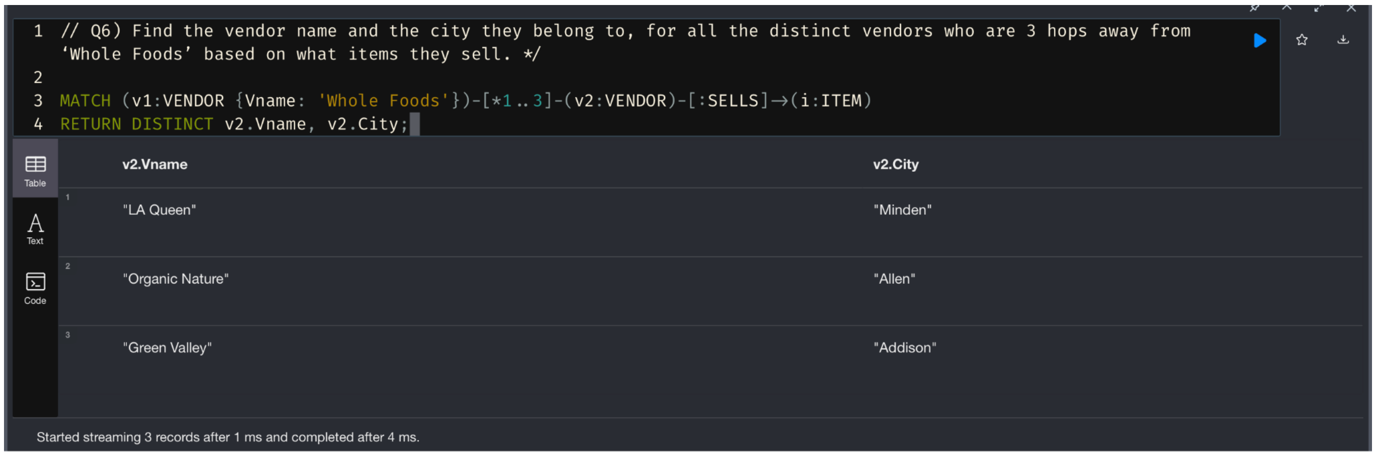
```
MATCH (v1:VENDOR {Vname: 'Whole Foods'})-[*3]-(v2:VENDOR)-[:SELLS]->(i:ITEM)

RETURN DISTINCT v2.Vname, v2.City;
```



```
// Q6) Find the vendor name and the city they belong to, for all the distinct vendors who are 3 hops away from ‘Whole Foods’ based on what items they sell. */

MATCH (v1:VENDOR {Vname: 'Whole Foods'})-[*3]-(v2:VENDOR)-[:SELLS]→(i:ITEM)
RETURN DISTINCT v2.Vname, v2.City;
```

(no changes, no records)

```
MATCH (wf:VENDOR {Vname: 'Whole Foods'})-[:SELLS]->(i:ITEM)<-[:SELLS]-(v:VENDOR)
WHERE wf <> v
RETURN DISTINCT v.Vname, v.City;
```



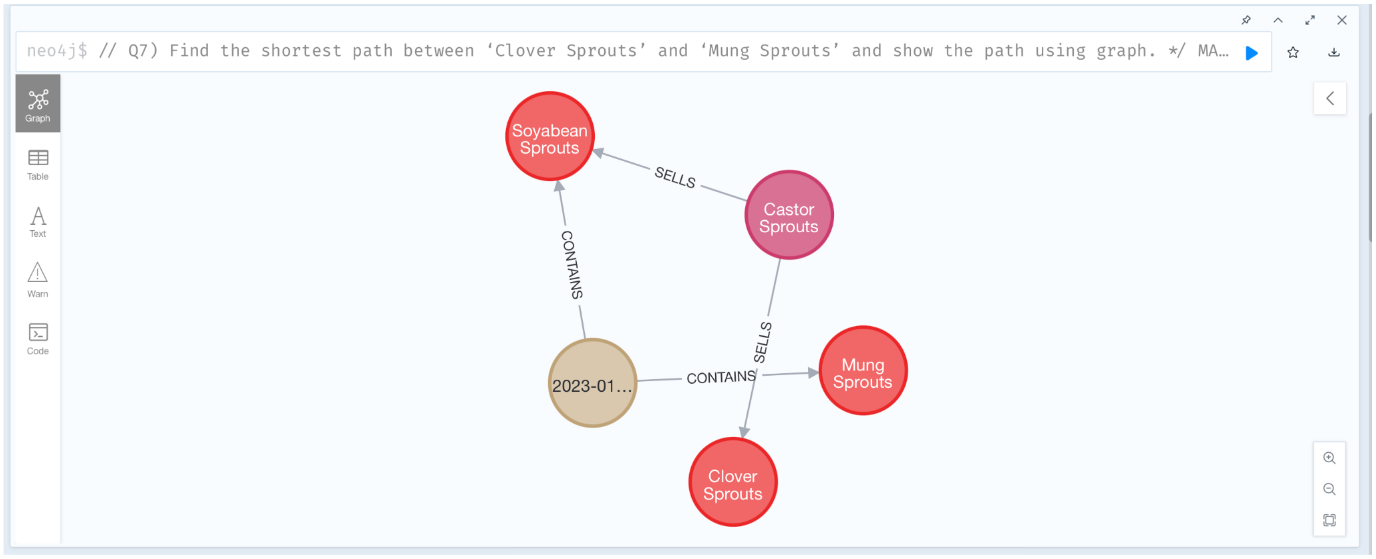
```
// Q6) Find the vendor name and the city they belong to, for all the distinct vendors who are 3 hops away from ‘Whole Foods’ based on what items they sell. */

MATCH (v1:VENDOR {Vname: 'Whole Foods'})-[*1..3]-(v2:VENDOR)-[:SELLS]→(i:ITEM)
RETURN DISTINCT v2.Vname, v2.City;
```

	v2.Vname	v2.City
1	"LA Queen"	"Minden"
2	"Organic Nature"	"Allen"
3	"Green Valley"	"Addison"

Started streaming 3 records after 1 ms and completed after 4 ms.

Q7) Find the shortest path between 'Clover Sprouts' and 'Mung Sprouts' and show the path using a graph.

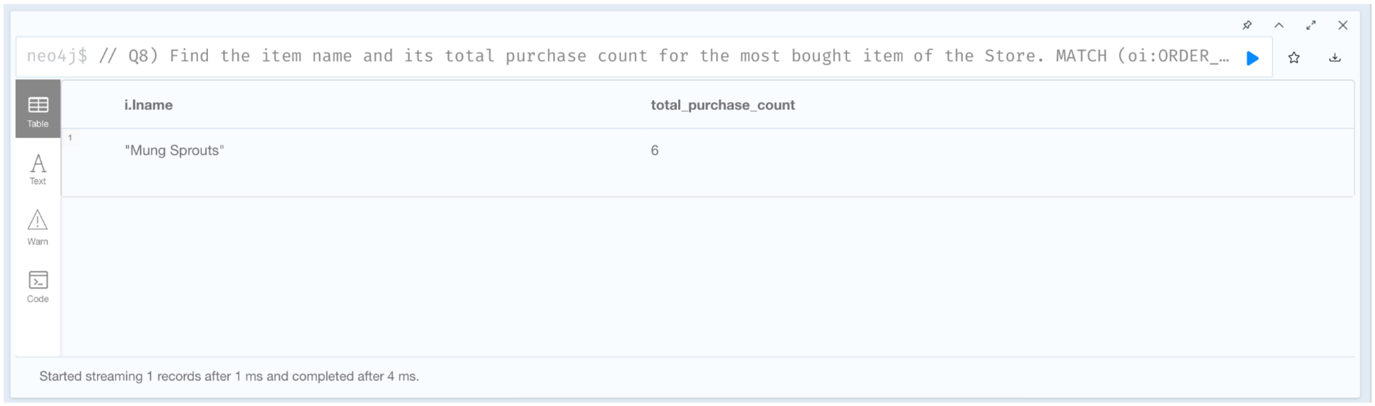
```
MATCH (start:ITEM {Iname: 'Clover Sprouts'}), (end:ITEM {Iname: 'Mung Sprouts'}),
    path = shortestPath((start)-[*]-(end))
RETURN path;
```



```
neo4j$ // Q7) Find the shortest path between 'Clover Sprouts' and 'Mung Sprouts' and show the path using graph. */ MA…
```

Q8) Find the item name and its total purchase count for the most bought item of the Store.

```
MATCH (oi:ORDER_ITEM), (i:ITEM)
WHERE oi.iId = i.iId
WITH oi.iId as Id, count(oi.Icount) as total_purchase_count, i
RETURN i.Iname, total_purchase_count
ORDER BY total_purchase_count DESC
LIMIT 1
```



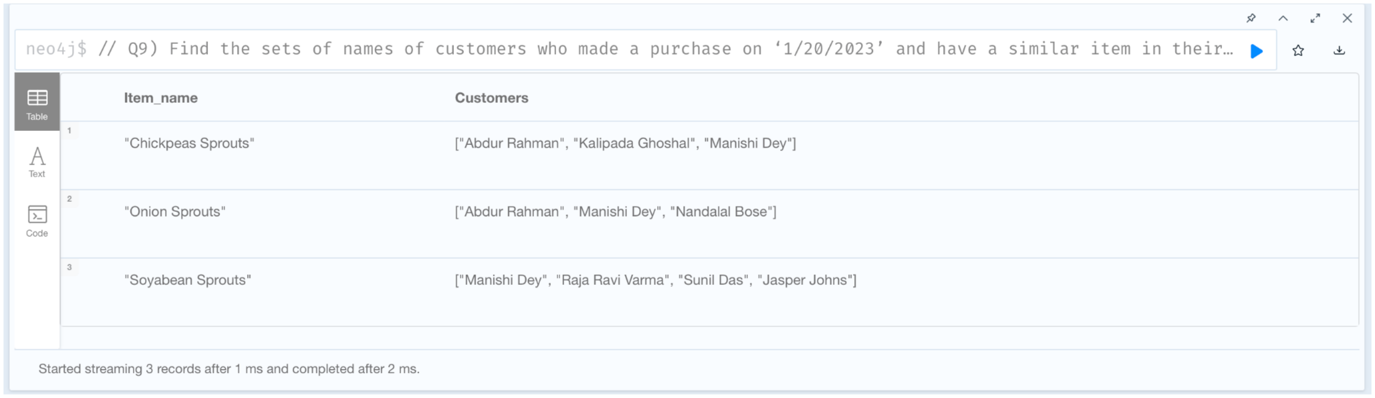
```
neo4j$ // Q8) Find the item name and its total purchase count for the most bought item of the Store. MATCH (oi:ORDER_…
```

	i.Iname	total_purchase_count
1	"Mung Sprouts"	6

Started streaming 1 records after 1 ms and completed after 4 ms.

Q9) Find the sets of names of customers who made a purchase on '1/20/2023' and have a similar item in their orders. OR
Find the sets of names of customers who bought the same items on '1/20/2023'.

```
MATCH (order:ORDERS {Odate: '2023-01-20'})-[:BELONGS_TO]->(customer:CUSTOMER),
    (order)-[:CONTAINS]->(item:ITEM)
WITH item, COLLECT(DISTINCT customer.Cname) AS Customers
WHERE SIZE(Customers) > 1
RETURN item.Iname AS Item_name, Customers
```



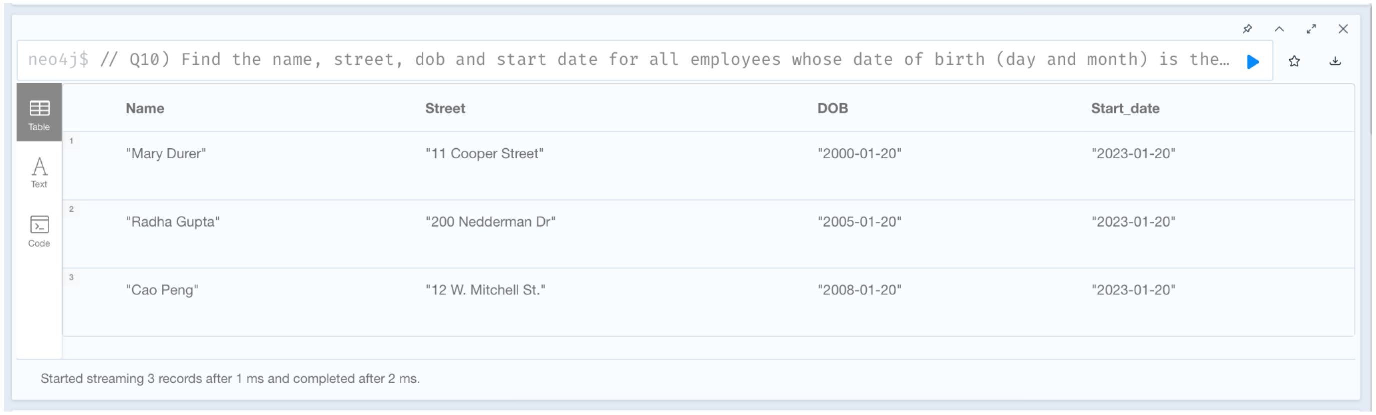
```
neo4j$ // Q9) Find the sets of names of customers who made a purchase on '1/20/2023' and have a similar item in their…
```

	Item_name	Customers
1	"Chickpeas Sprouts"	["Abdur Rahman", "Kalipada Ghoshal", "Manishi Dey"]
2	"Onion Sprouts"	["Abdur Rahman", "Manishi Dey", "Nandalal Bose"]
3	"Soyabean Sprouts"	["Manishi Dey", "Raja Ravi Varma", "Sunil Das", "Jasper Johns"]

Started streaming 3 records after 1 ms and completed after 2 ms.

Q10) Find the name, street, dob and start date for all employees whose date of birth (day and month) is the same as their start date (day and month) at the store.

```
MATCH (e:EMPLOYEE)
WHERE substring(e.Bdate, 6, 5) = substring(e.Sdate, 6, 5)
RETURN e.Sname AS Name, e.Street as Street, e.Bdate as DOB ,e.Sdate as Start_date
```



```
neo4j$ // Q10) Find the name, street, dob and start date for all employees whose date of birth (day and month) is the…
```

	Name	Street	DOB	Start_date
1	"Mary Durer"	"11 Cooper Street"	"2000-01-20"	"2023-01-20"
2	"Radha Gupta"	"200 Nedderman Dr"	"2005-01-20"	"2023-01-20"
3	"Cao Peng"	"12 W. Mitchell St."	"2008-01-20"	"2023-01-20"

Started streaming 3 records after 1 ms and completed after 2 ms.

7. References

Neo4j & Cypher:

- Neo4j's official documentation: link
- Neo4j's Cypher Query Language Portal: link
- Book: "Graph Databases" by Ian Robinson, Jim Webber, and Emil Eifrem

Graph Database Concepts:

- Book: "Graph Database: New Opportunities for Connected Data" by Ian Robinson, Jim Webber, and Emil Eifrem