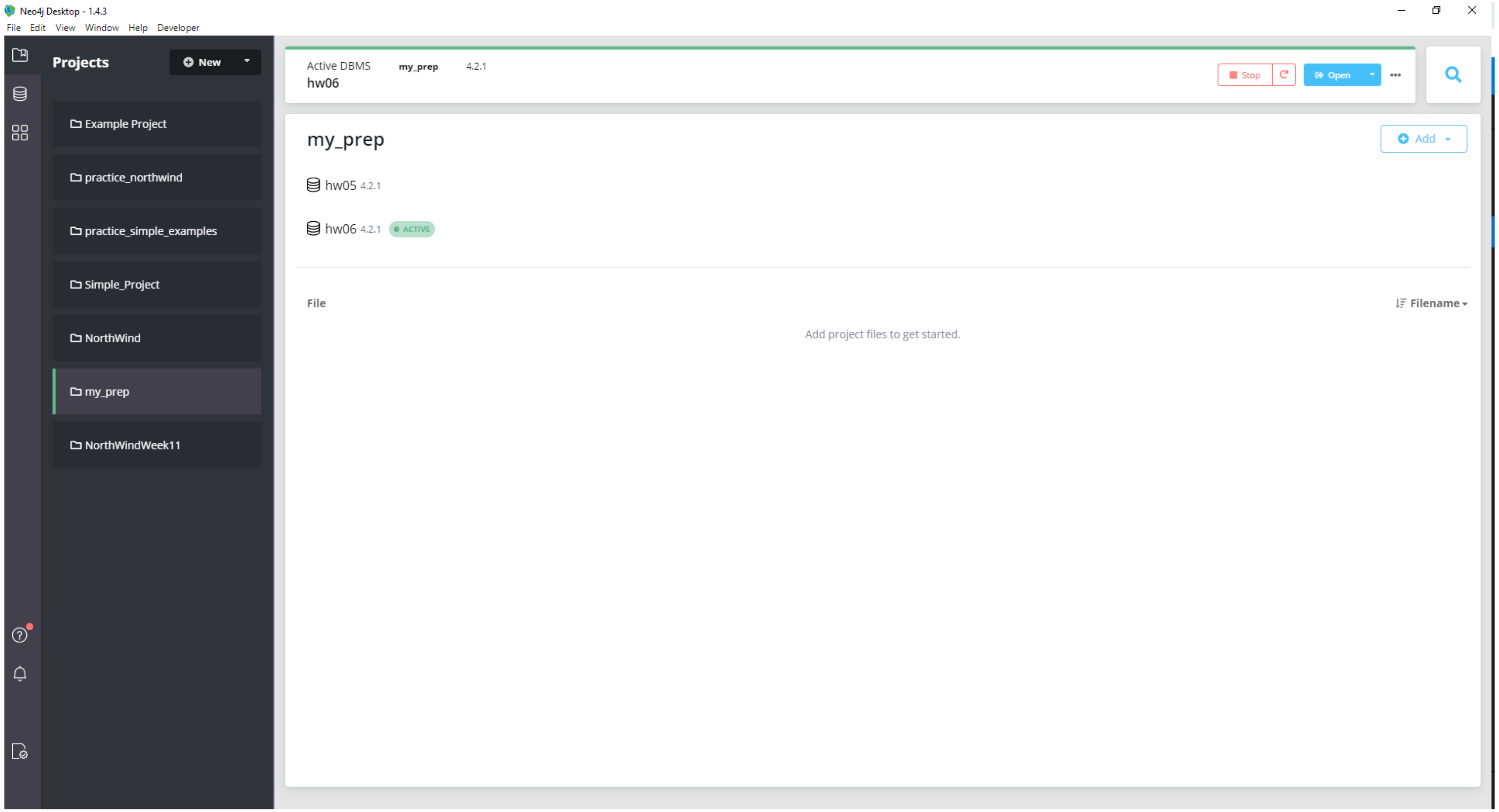
CMPINF 2110

Spring 2021

Homework 06 Solutions

Create a new local DBMS in Neo4j desktop. I chose to name mine hw06.



Active DBMS my_prep 4.2.1
hw06

- Example Project
- practice_northwind
- practice_simple_examples
- Simple_Project
- NorthWind
- my_prep
- NorthWindWeek11

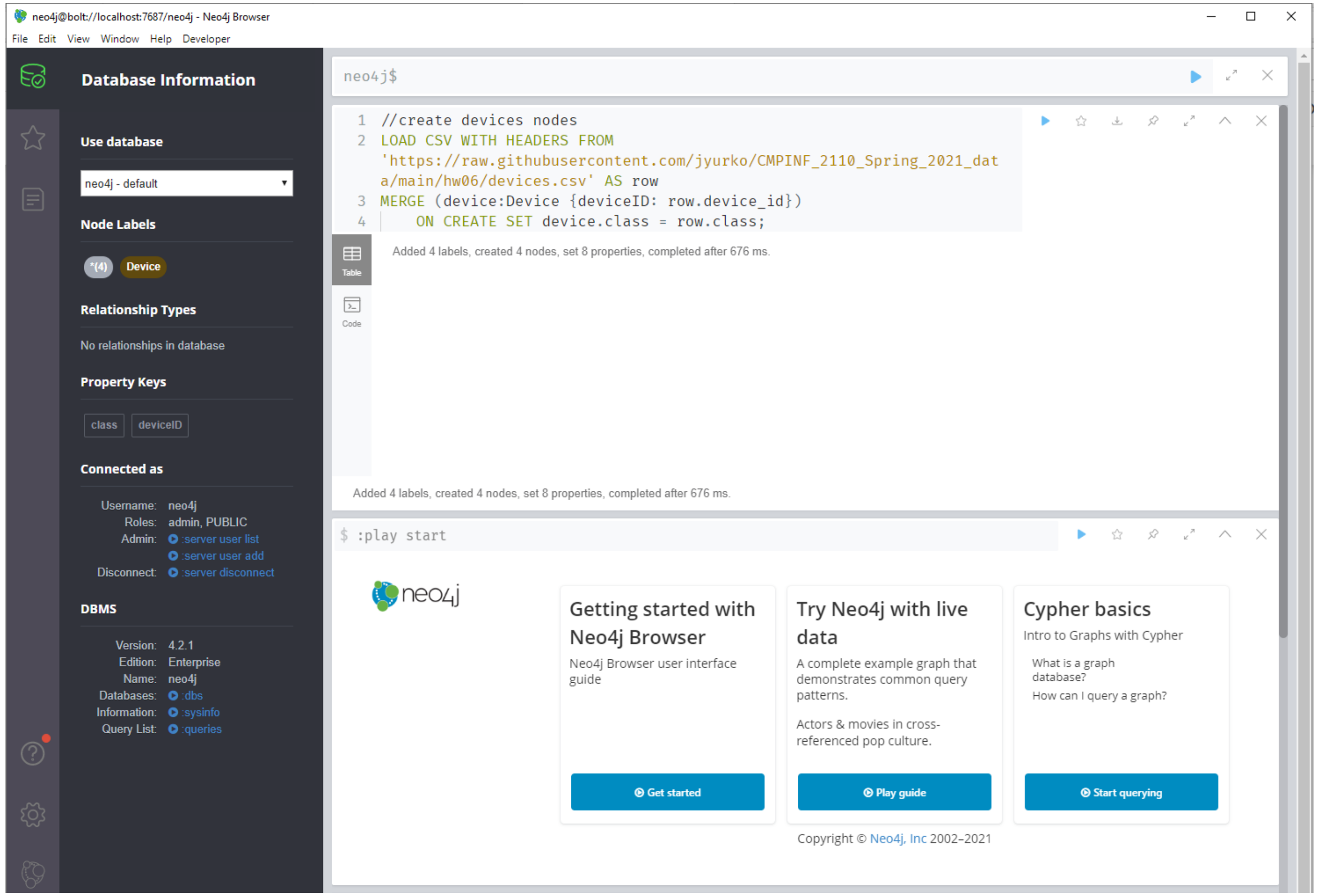
my_prep

hw05 4.2.1

hw06 4.2.1 ACTIVE

Add project files to get started.

Add Device Label and create nodes with properties set to the columns of the devices table



No relationships in database

Username: neo4j
Roles: admin, PUBLIC
Admin: :server user list
:server user add
Disconnect: :server disconnect

Version: 4.2.1
Edition: Enterprise
Name: neo4j
Databases: :dbs
Information: :sysinfo
Query List: :queries

```
//create devices nodes
LOAD CSV WITH HEADERS FROM 'https://raw.githubusercontent.com/jyurko/CMPINF_2110_Spring_2021_data/main/hw06/devices.csv' AS row
MERGE (device:Device {deviceID: row.device_id})
    ON CREATE SET device.class = row.class;
```

Added 4 labels, created 4 nodes, set 8 properties, completed after 676 ms.

Added 4 labels, created 4 nodes, set 8 properties, completed after 676 ms.

```
$ :play start
```

neo4j

Getting started with Neo4j Browser

Neo4j Browser user interface guide

Try Neo4j with live data

A complete example graph that demonstrates common query patterns.

Actors & movies in cross-referenced pop culture.

Cypher basics

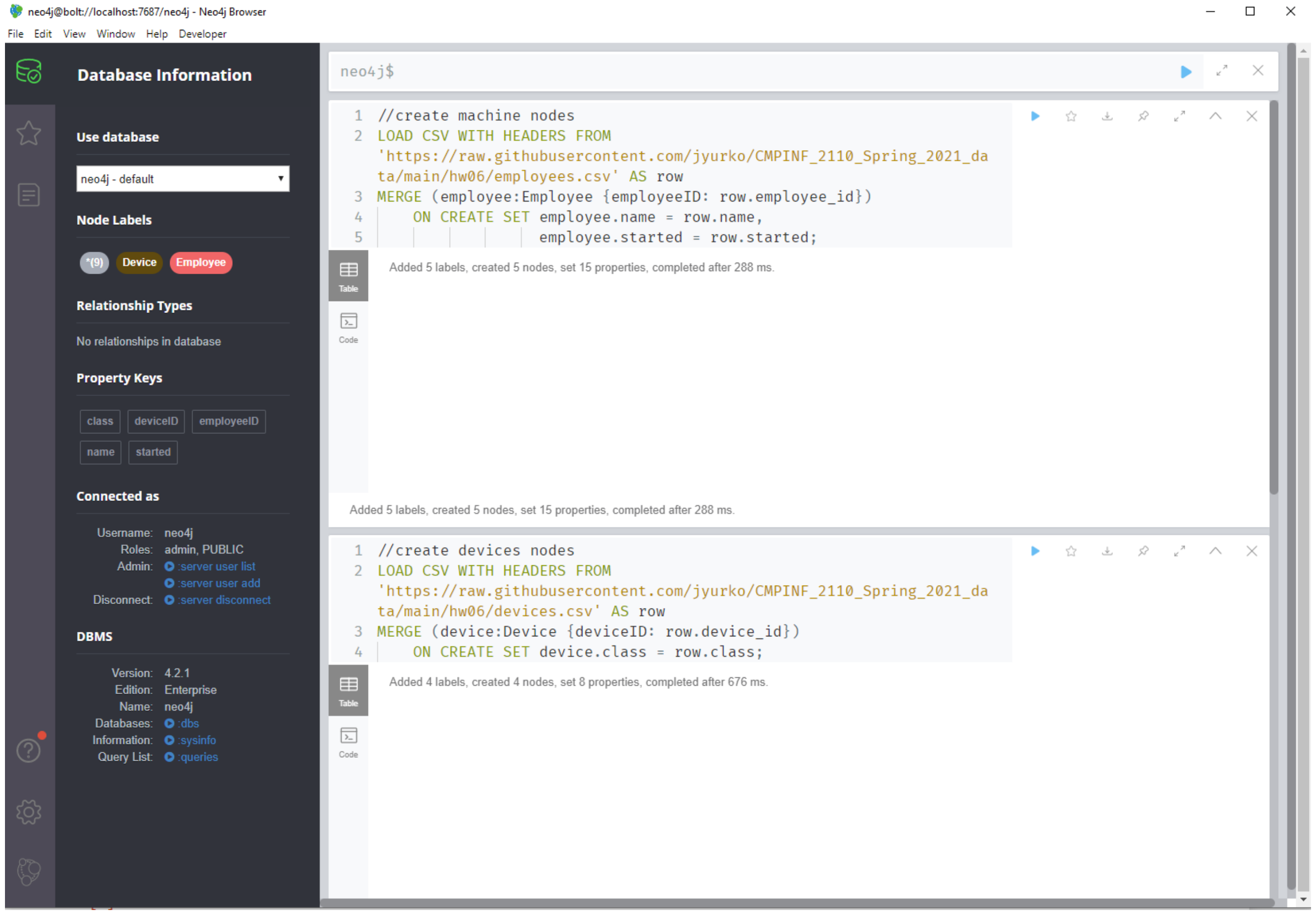
Intro to Graphs with Cypher

What is a graph database?

How can I query a graph?

Copyright © Neo4j, Inc 2002–2021

Add Employee Label and create nodes with properties set to the columns of the employees table



Database Information

Use database

neo4j - default

Node Labels

*(9) Device Employee

Relationship Types

No relationships in database

Property Keys

class deviceID employeeID name started

Connected as

Username: neo4j
Roles: admin, PUBLIC
Admin: :server user list
:server user add
Disconnect: :server disconnect

DBMS

Version: 4.2.1
Edition: Enterprise
Name: neo4j
Databases: :dbs
Information: :sysinfo
Query List: :queries

```
//create machine nodes
LOAD CSV WITH HEADERS FROM
'https://raw.githubusercontent.com/jyurko/CMPINF_2110_Spring_2021_da
ta/main/hw06/employees.csv' AS row
MERGE (employee:Employee {employeeID: row.employee_id})
    ON CREATE SET employee.name = row.name,
                  employee.started = row.started;
```

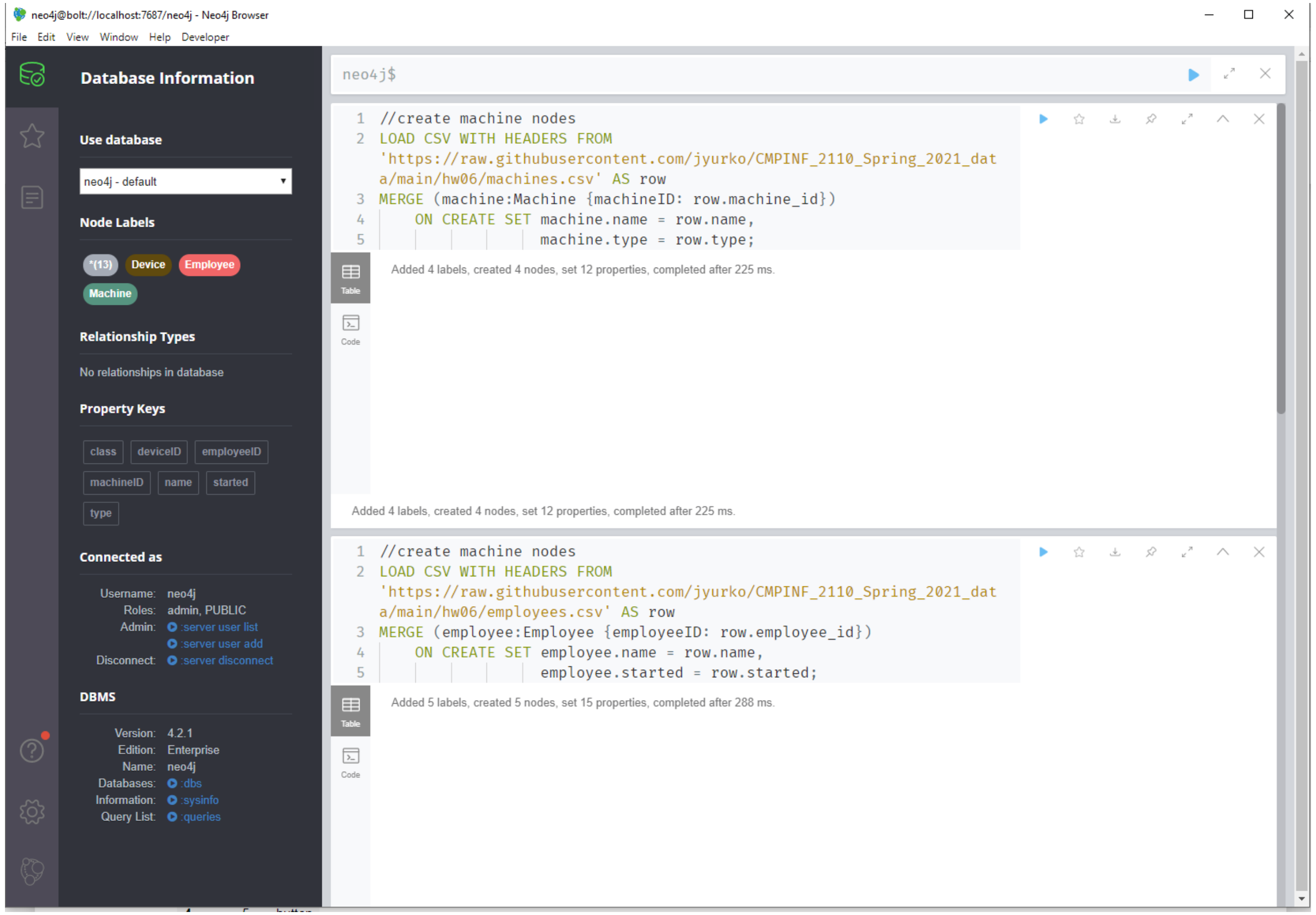
Added 5 labels, created 5 nodes, set 15 properties, completed after 288 ms.

Added 5 labels, created 5 nodes, set 15 properties, completed after 288 ms.

```
//create devices nodes
LOAD CSV WITH HEADERS FROM
'https://raw.githubusercontent.com/jyurko/CMPINF_2110_Spring_2021_da
ta/main/hw06/devices.csv' AS row
MERGE (device:Device {deviceID: row.device_id})
    ON CREATE SET device.class = row.class;
```

Added 4 labels, created 4 nodes, set 8 properties, completed after 676 ms.

Add Machine Label and create nodes with properties set to the columns of the machines table


neo4j@bolt://localhost:7687/neo4j - Neo4j Browser
File Edit View Window Help Developer
Database Information
neo4j$
Table
Code

Use database

neo4j - default

Node Labels

*(13) Device Employee Machine

Relationship Types

No relationships in database

Property Keys

class deviceID employeeID machineID name started type

Connected as

Username: neo4j
Roles: admin, PUBLIC
Admin: :server user list
:server user add
Disconnect: :server disconnect

DBMS

Version: 4.2.1
Edition: Enterprise
Name: neo4j
Databases: :dbs
Information: :sysinfo
Query List: :queries

```
//create machine nodes
LOAD CSV WITH HEADERS FROM
'https://raw.githubusercontent.com/jyurko/CMPINF_2110_Spring_2021_dat
a/main/hw06/machines.csv' AS row
MERGE (machine:Machine {machineID: row.machine_id})
    ON CREATE SET machine.name = row.name,
                  machine.type = row.type;
```

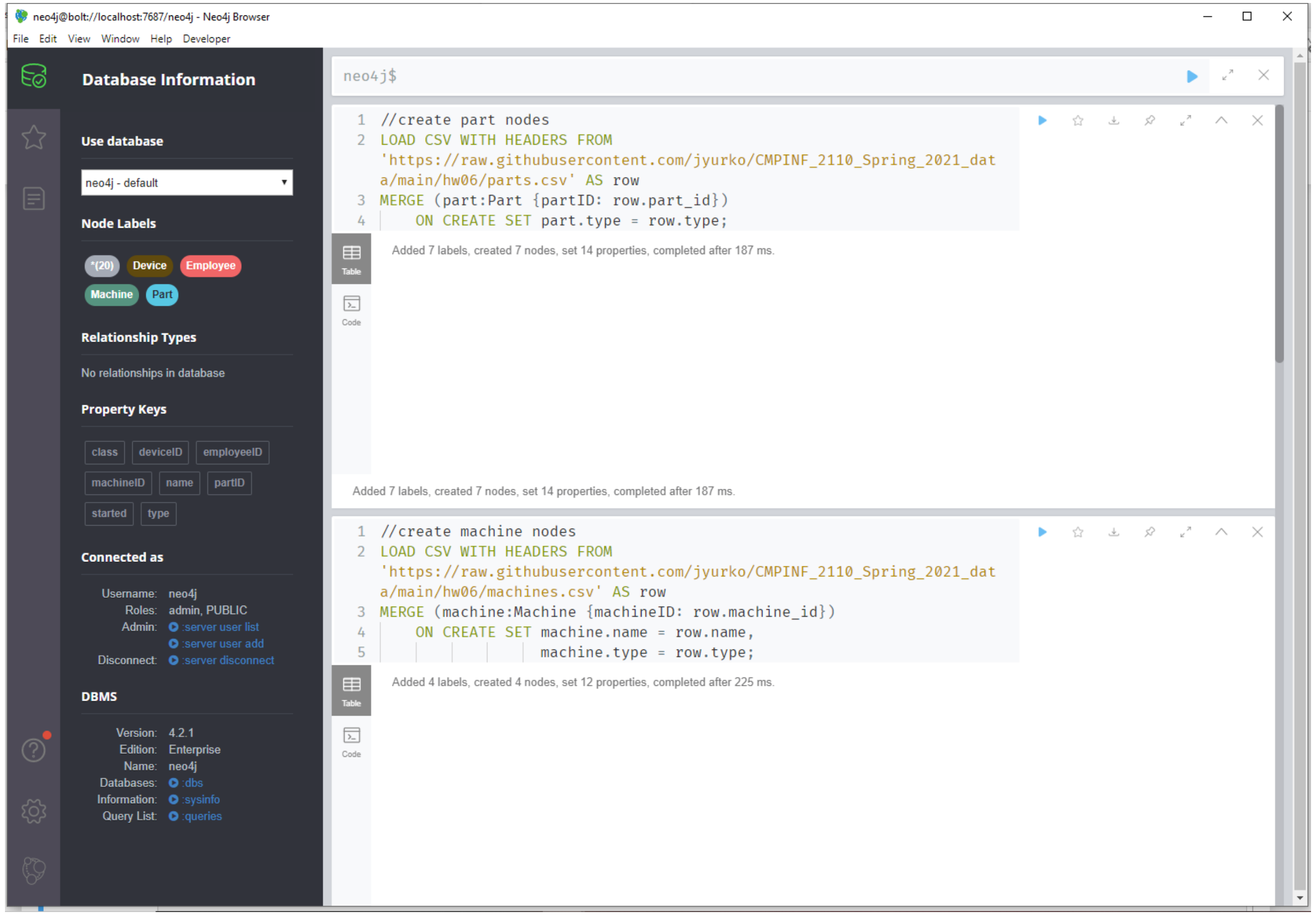
Added 4 labels, created 4 nodes, set 12 properties, completed after 225 ms.

Added 4 labels, created 4 nodes, set 12 properties, completed after 225 ms.

```
//create machine nodes
LOAD CSV WITH HEADERS FROM
'https://raw.githubusercontent.com/jyurko/CMPINF_2110_Spring_2021_dat
a/main/hw06/employees.csv' AS row
MERGE (employee:Employee {employeeID: row.employee_id})
    ON CREATE SET employee.name = row.name,
                  employee.started = row.started;
```

Added 5 labels, created 5 nodes, set 15 properties, completed after 288 ms.

Add Part Label and create nodes with properties set to the columns of the parts table



Use database: neo4j - default

No relationships in database

Username: neo4j
Roles: admin, PUBLIC
Admin: :server user list
:server user add
Disconnect: :server disconnect

Version: 4.2.1
Edition: Enterprise
Name: neo4j
Databases: :dbs
Information: :sysinfo
Query List: :queries

```
//create part nodes
LOAD CSV WITH HEADERS FROM
'https://raw.githubusercontent.com/jyurko/CMPINF_2110_Spring_2021_data/main/hw06/parts.csv' AS row
MERGE (part:Part {partID: row.part_id})
    ON CREATE SET part.type = row.type;
```

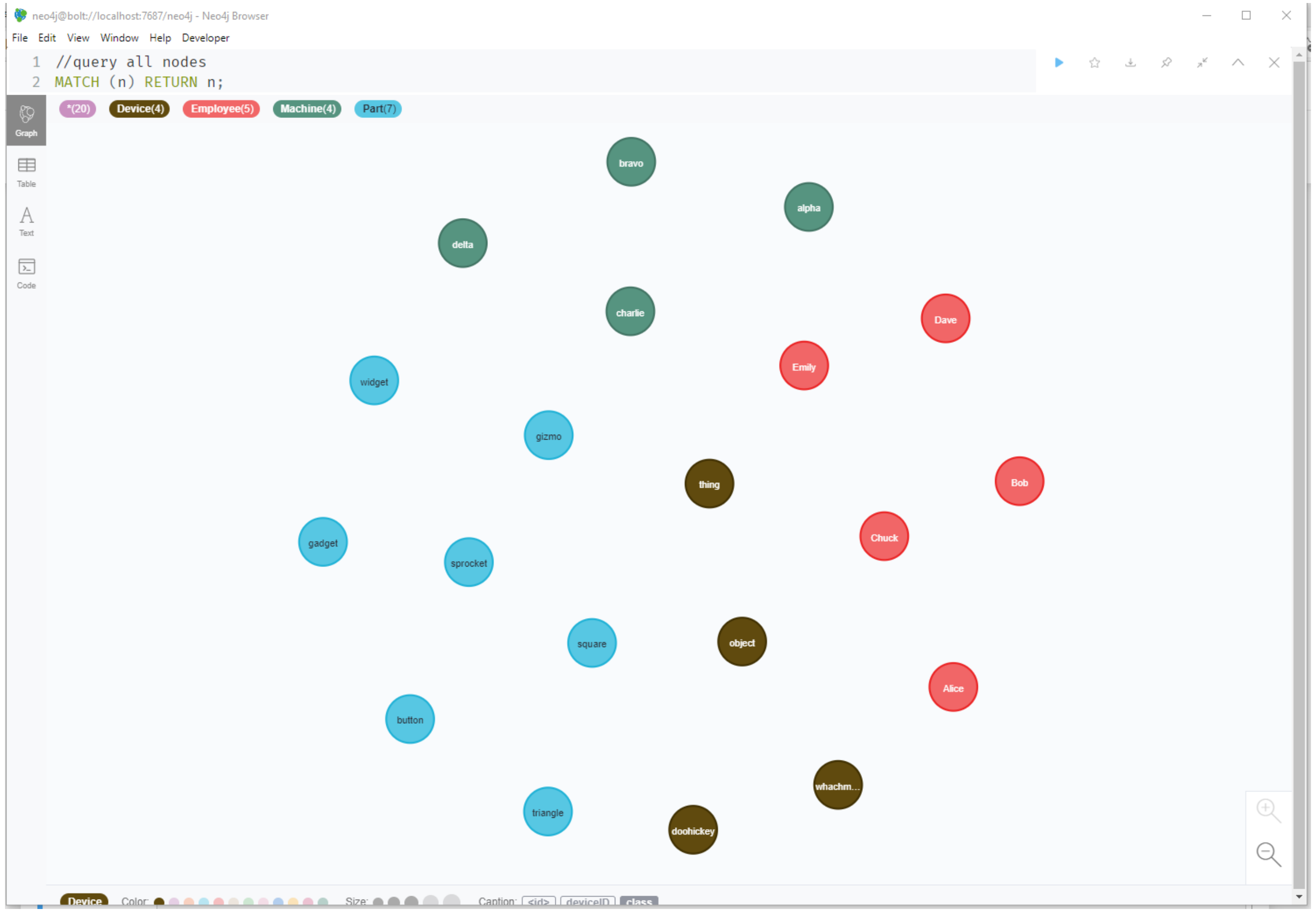
Added 7 labels, created 7 nodes, set 14 properties, completed after 187 ms.

Added 7 labels, created 7 nodes, set 14 properties, completed after 187 ms.

```
//create machine nodes
LOAD CSV WITH HEADERS FROM
'https://raw.githubusercontent.com/jyurko/CMPINF_2110_Spring_2021_data/main/hw06/machines.csv' AS row
MERGE (machine:Machine {machineID: row.machine_id})
    ON CREATE SET machine.name = row.name,
                  machine.type = row.type;
```

Added 4 labels, created 4 nodes, set 12 properties, completed after 225 ms.

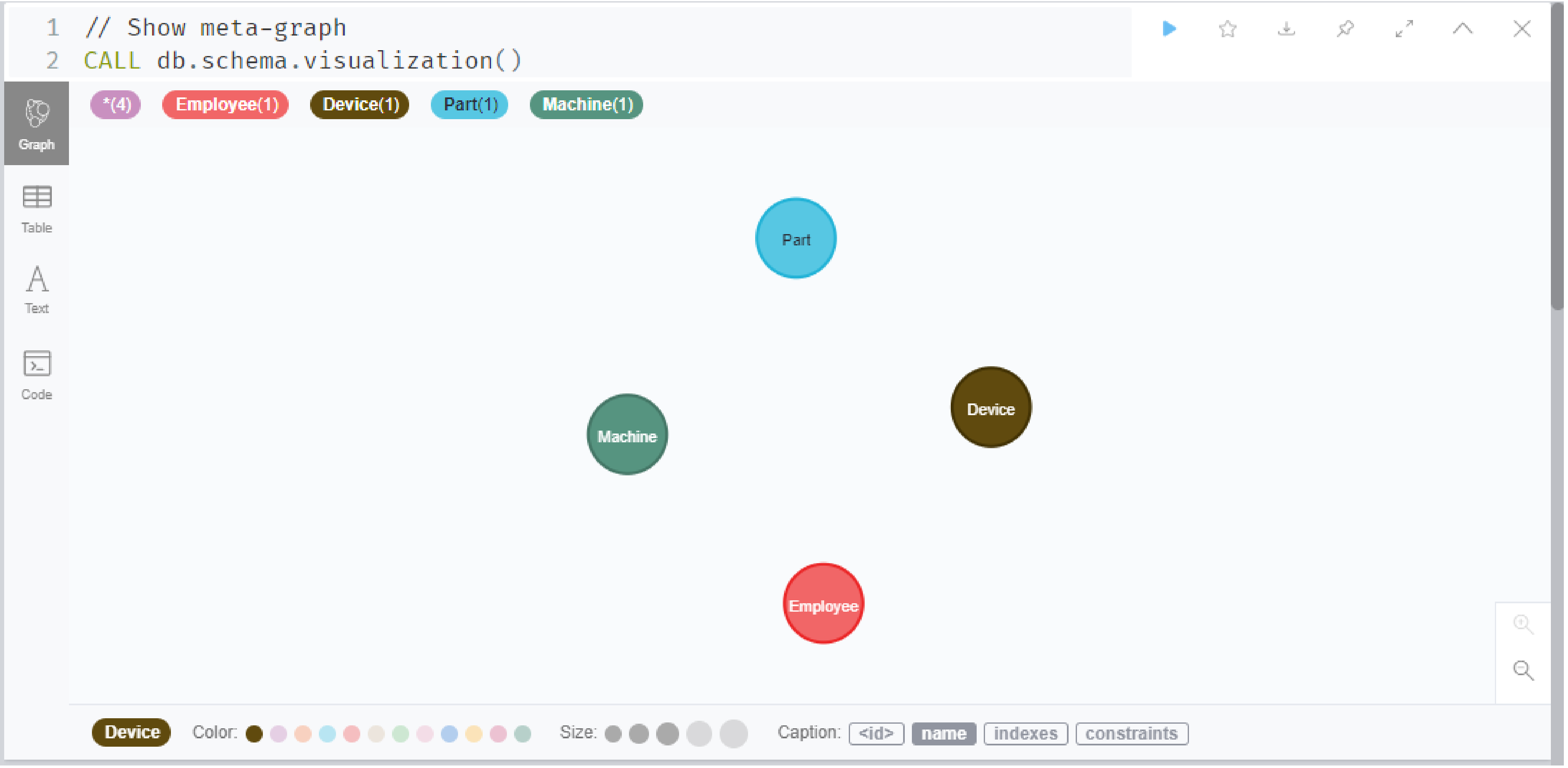
As a check, query all nodes in the graph model



```
//query all nodes
MATCH (n) RETURN n;
```

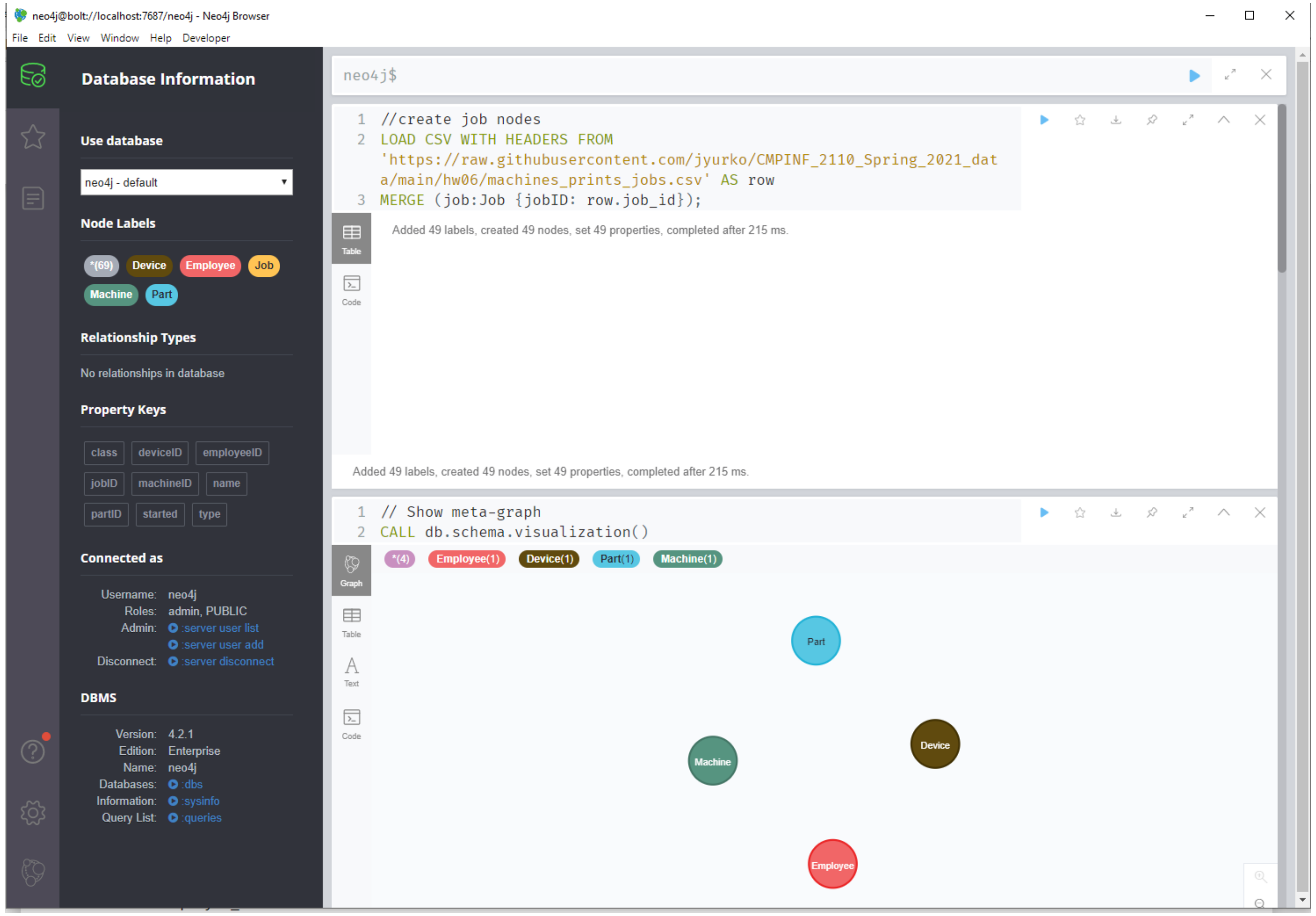
No relationships are defined as of yet!

The "meta graph" is another way to reveal no relationships are defined as of yet



```
// Show meta-graph
CALL db.schema.visualization()
```

Add Job label and create nodes with jobID property



neo4j@bolt://localhost:7687/neo4j - Neo4j Browser
File Edit View Window Help Developer
Database Information
Table
Code
Graph
Table
Text
Code
Part
Device
Machine
Employee

Use database

neo4j - default

Node Labels

*(69) Device Employee Job Machine Part

Relationship Types

No relationships in database

Property Keys

class deviceID employeeID jobID machineID name partID started type

Connected as

Username: neo4j
Roles: admin, PUBLIC
Admin: :server user list
:server user add
Disconnect: :server disconnect

DBMS

Version: 4.2.1
Edition: Enterprise
Name: neo4j
Databases: :dbs
Information: :sysinfo
Query List: :queries

```
neo4j$
```

```
//create job nodes
LOAD CSV WITH HEADERS FROM
'https://raw.githubusercontent.com/jyurko/CMPINF_2110_Spring_2021_dat
a/main/hw06/machines_prints_jobs.csv' AS row
MERGE (job:Job {jobID: row.job_id});
```

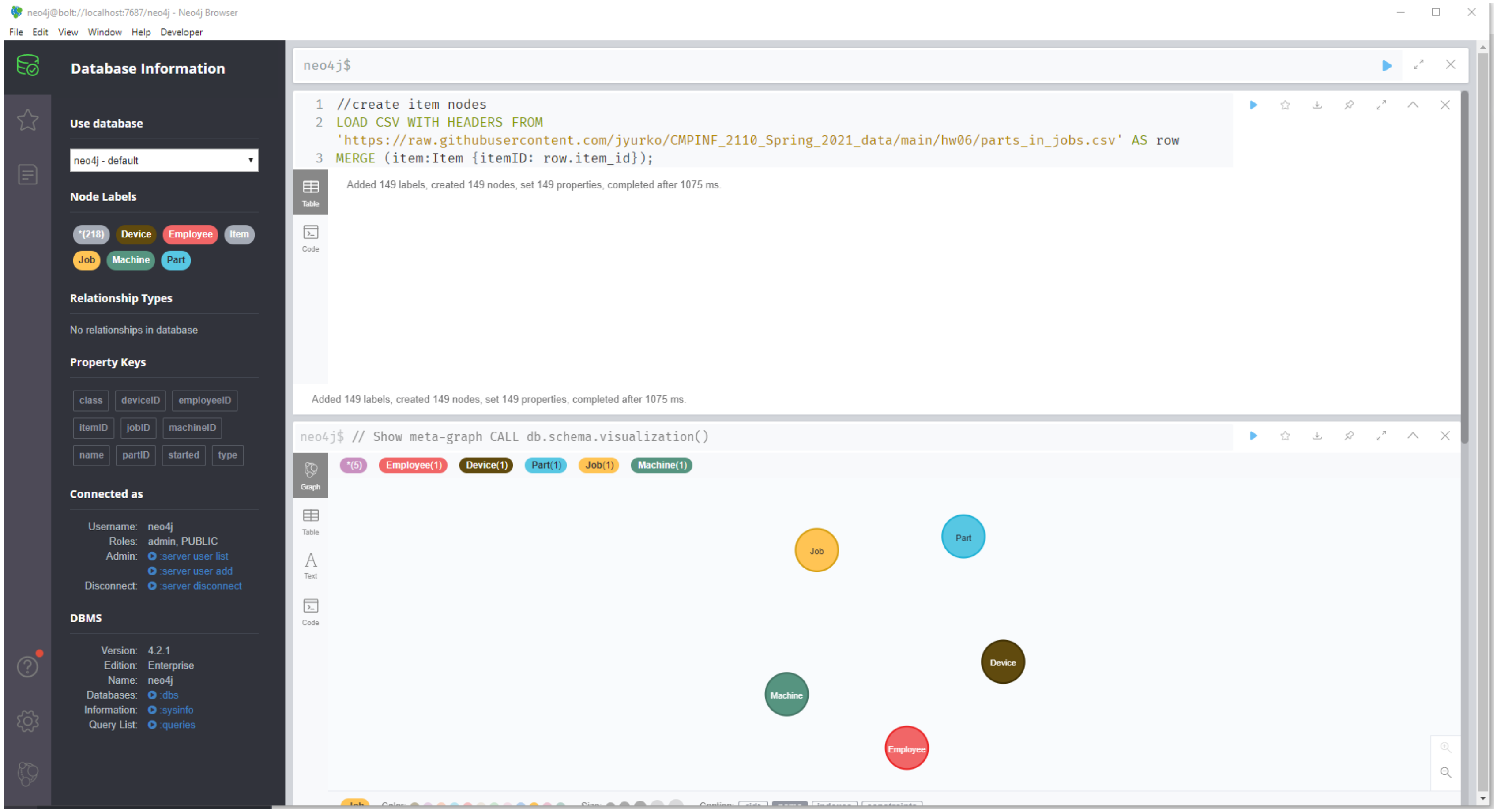
```
Added 49 labels, created 49 nodes, set 49 properties, completed after 215 ms.
```

```
Added 49 labels, created 49 nodes, set 49 properties, completed after 215 ms.
```

```
// Show meta-graph
CALL db.schema.visualization()
```

*(4) Employee(1) Device(1) Part(1) Machine(1)

Add Item label and create nodes with itemID property



neo4j@bolt://localhost:7687/neo4j - Neo4j Browser
File Edit View Window Help Developer
Database Information
Table
Code
Graph
Text
Job
Part
Device
Machine
Employee

Use database

neo4j - default

Node Labels

*(218) Device Employee Item Job Machine Part

Relationship Types

No relationships in database

Property Keys

class deviceID employeeID itemID jobID machineID name partID started type

Connected as

Username: neo4j
Roles: admin, PUBLIC
Admin: :server user list
:server user add
Disconnect: :server disconnect

DBMS

Version: 4.2.1
Edition: Enterprise
Name: neo4j
Databases: :dbs
Information: :sysinfo
Query List: :queries

```
neo4j$
```

```
//create item nodes
LOAD CSV WITH HEADERS FROM
'https://raw.githubusercontent.com/jyurko/CMPINF_2110_Spring_2021_data/main/hw06/parts_in_jobs.csv' AS row
MERGE (item:Item {itemID: row.item_id});
```

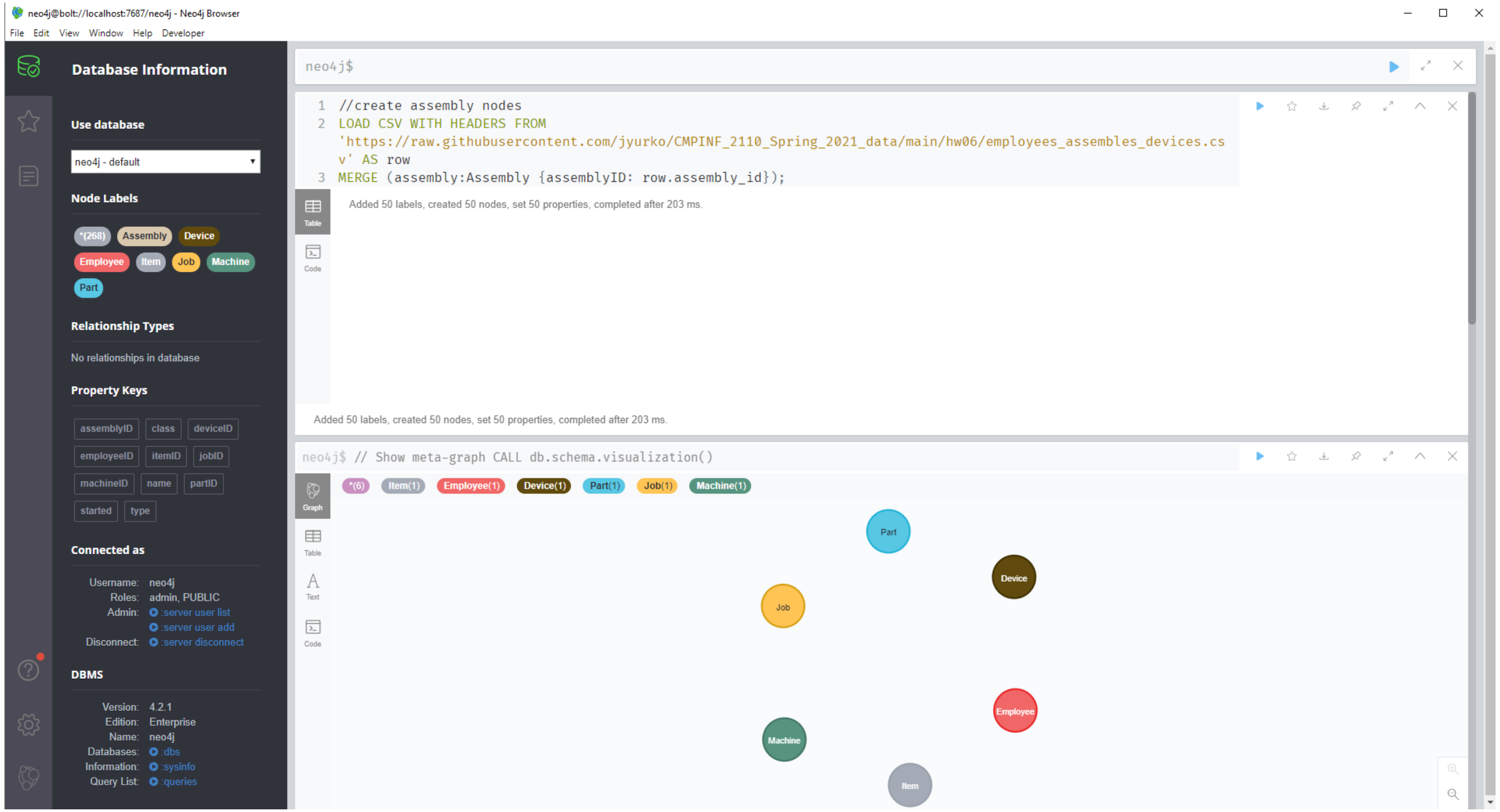
```
Added 149 labels, created 149 nodes, set 149 properties, completed after 1075 ms.
```

```
Added 149 labels, created 149 nodes, set 149 properties, completed after 1075 ms.
```

```
neo4j$ // Show meta-graph CALL db.schema.visualization()
```

*(5) Employee(1) Device(1) Part(1) Job(1) Machine(1)

Add Assembly label and create nodes with assemblyID property


neo4j@bolt://localhost:7687/neo4j - Neo4j Browser
File Edit View Window Help Developer
Database Information
Table
Code
Graph
Text
Part
Device
Job
Employee
Machine
Item

Use database

neo4j - default

Node Labels

*(268) Assembly Device Employee Item Job Machine Part

Relationship Types

No relationships in database

Property Keys

assemblyID class deviceID employeeID itemID jobID machineID name partID started type

Connected as

Username: neo4j
Roles: admin, PUBLIC
Admin: :server user list
:server user add
Disconnect: :server disconnect

DBMS

Version: 4.2.1
Edition: Enterprise
Name: neo4j
Databases: :dbs
Information: :sysinfo
Query List: :queries

```
neo4j$
```

```
//create assembly nodes
LOAD CSV WITH HEADERS FROM
'https://raw.githubusercontent.com/jyurko/CMPINF_2110_Spring_2021_data/main/hw06/employees_assembles_devices.csv' AS row
MERGE (assembly:Assembly {assemblyID: row.assembly_id});
```

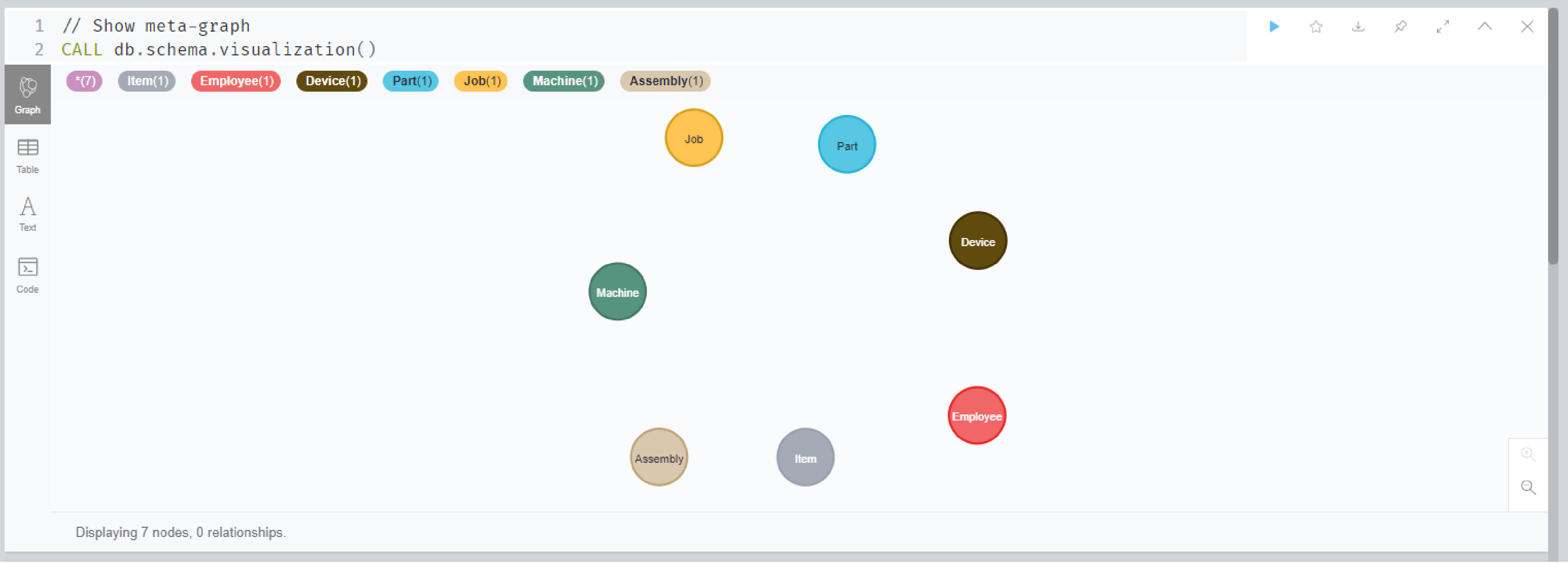
```
Added 50 labels, created 50 nodes, set 50 properties, completed after 203 ms.
```

```
Added 50 labels, created 50 nodes, set 50 properties, completed after 203 ms.
```

```
neo4j$ // Show meta-graph CALL db.schema.visualization()
```

*(6) Item(1) Employee(1) Device(1) Part(1) Job(1) Machine(1)

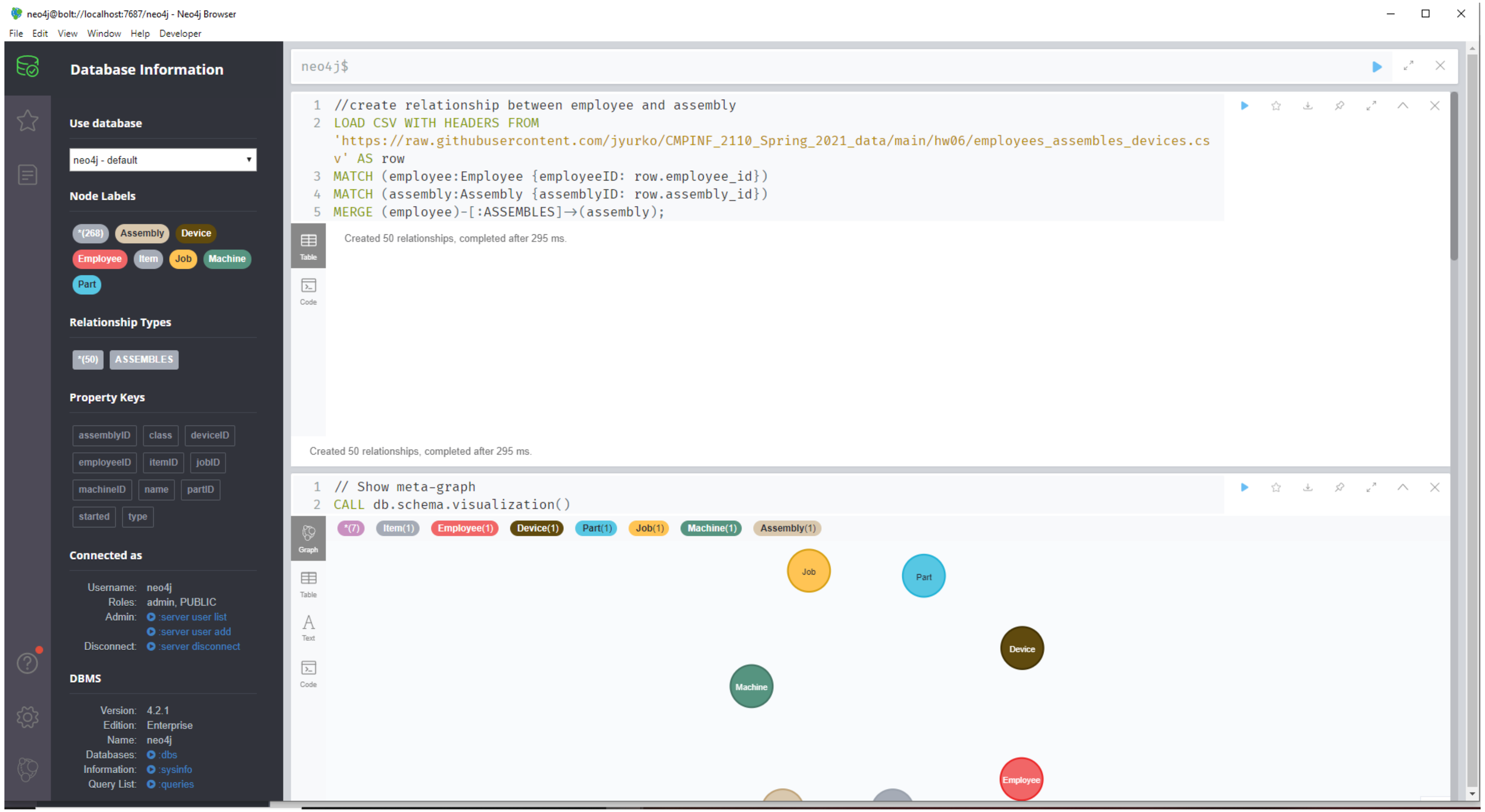
All nodes have been created. The meta-graph shows no relationships have been defined as of yet.



```
// Show meta-graph
CALL db.schema.visualization()
```

Displaying 7 nodes, 0 relationships.

Create the ASSEMBLES relationship type between Employee nodes and Assembly nodes



neo4j@bolt://localhost:7687/neo4j - Neo4j Browser
File Edit View Window Help Developer
Table
Code
Graph
Table
Text
Code
Job
Part
Device
Machine
Employee

Database Information

Use database

neo4j - default

Node Labels

*(268) Assembly Device Employee Item Job Machine Part

Relationship Types

*(50) ASSEMBLES

Property Keys

assemblyID class deviceID employeeID itemID jobID machineID name partID started type

Connected as

Username: neo4j
Roles: admin, PUBLIC
Admin: :server user list
:server user add
Disconnect: :server disconnect

DBMS

Version: 4.2.1
Edition: Enterprise
Name: neo4j
Databases: :dbs
Information: :sysinfo
Query List: :queries

```
neo4j$
```

```
//create relationship between employee and assembly
LOAD CSV WITH HEADERS FROM 'https://raw.githubusercontent.com/jyurko/CMPINF_2110_Spring_2021_data/main/hw06/employees_assembles_devices.csv' AS row
MATCH (employee:Employee {employeeID: row.employee_id})
MATCH (assembly:Assembly {assemblyID: row.assembly_id})
MERGE (employee)-[:ASSEMBLES]→(assembly);
```

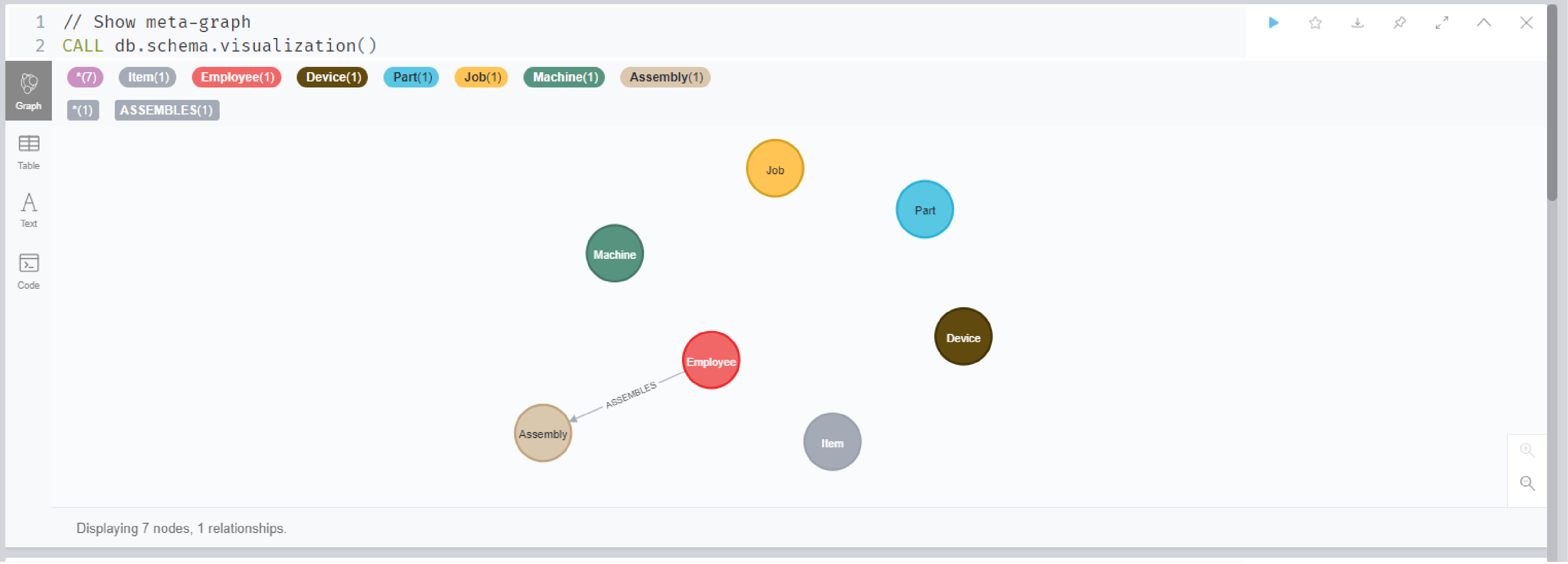
```
Created 50 relationships, completed after 295 ms.
```

```
Created 50 relationships, completed after 295 ms.
```

```
// Show meta-graph
CALL db.schema.visualization()
```

*(7) Item(1) Employee(1) Device(1) Part(1) Job(1) Machine(1) Assembly(1)

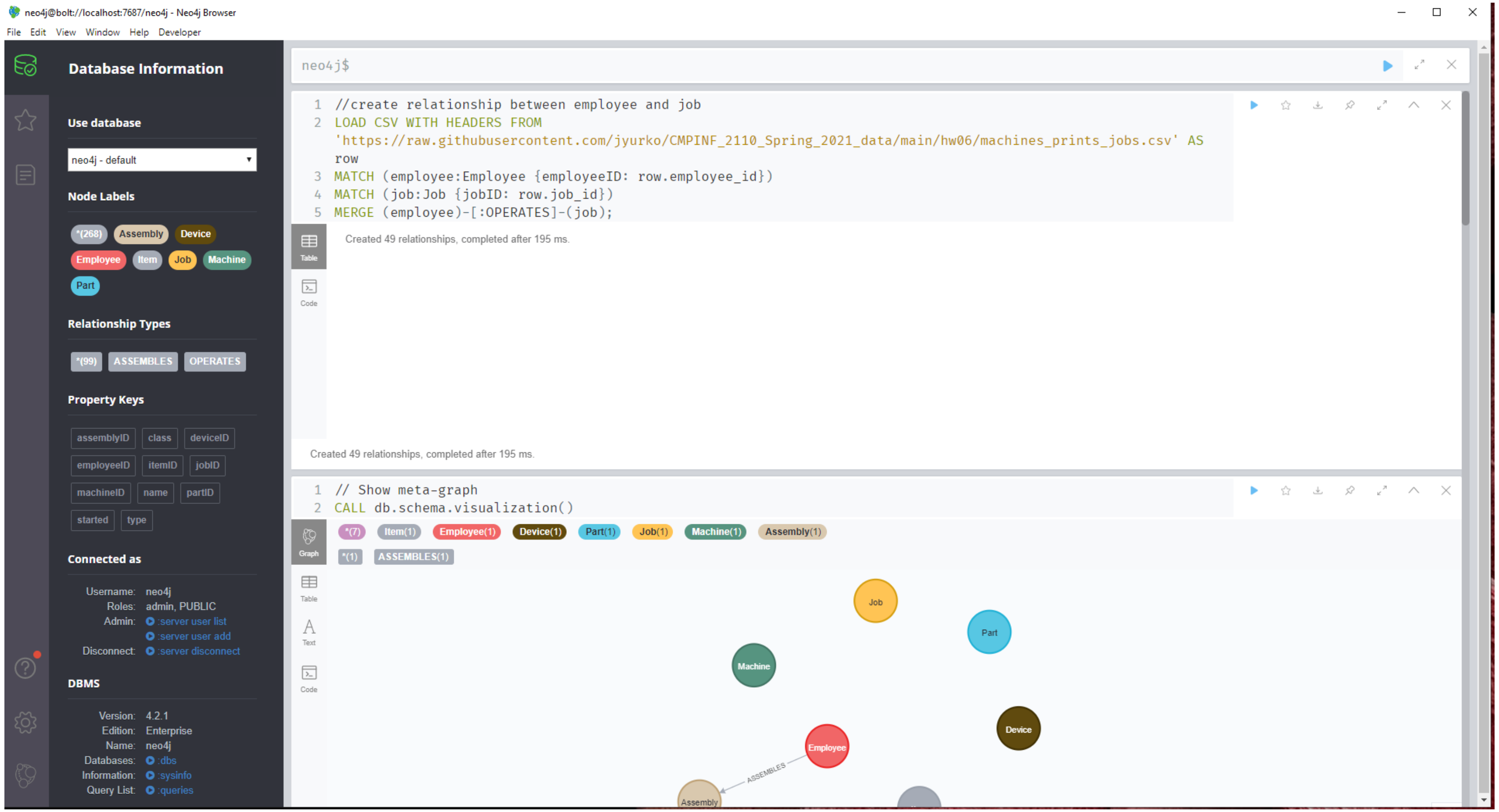
We now have a relationship!



```
// Show meta-graph
CALL db.schema.visualization()
```

Displaying 7 nodes, 1 relationships.

Create the OPERATES relationship type between Employee and Job nodes



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

Database Information

Use database: neo4j - default

Node Labels: *(268) Assembly Device Employee Item Job Machine Part

Relationship Types: *(99) ASSEMBLES OPERATES

Property Keys: assemblyID class deviceID employeeID itemID jobID machineID name partID started type

Connected as

Username: neo4j
Roles: admin, PUBLIC
Admin: :server user list
:server user add
Disconnect: :server disconnect

DBMS

Version: 4.2.1
Edition: Enterprise
Name: neo4j
Databases: :dbs
Information: :sysinfo
Query List: :queries

```
neo4j$
```

```
//create relationship between employee and job
LOAD CSV WITH HEADERS FROM
'https://raw.githubusercontent.com/jyurko/CMPINF_2110_Spring_2021_data/main/hw06/machines_prints_jobs.csv' AS
row
MATCH (employee:Employee {employeeID: row.employee_id})
MATCH (job:Job {jobID: row.job_id})
MERGE (employee)-[:OPERATES]-(job);
```

```
Created 49 relationships, completed after 195 ms.
```

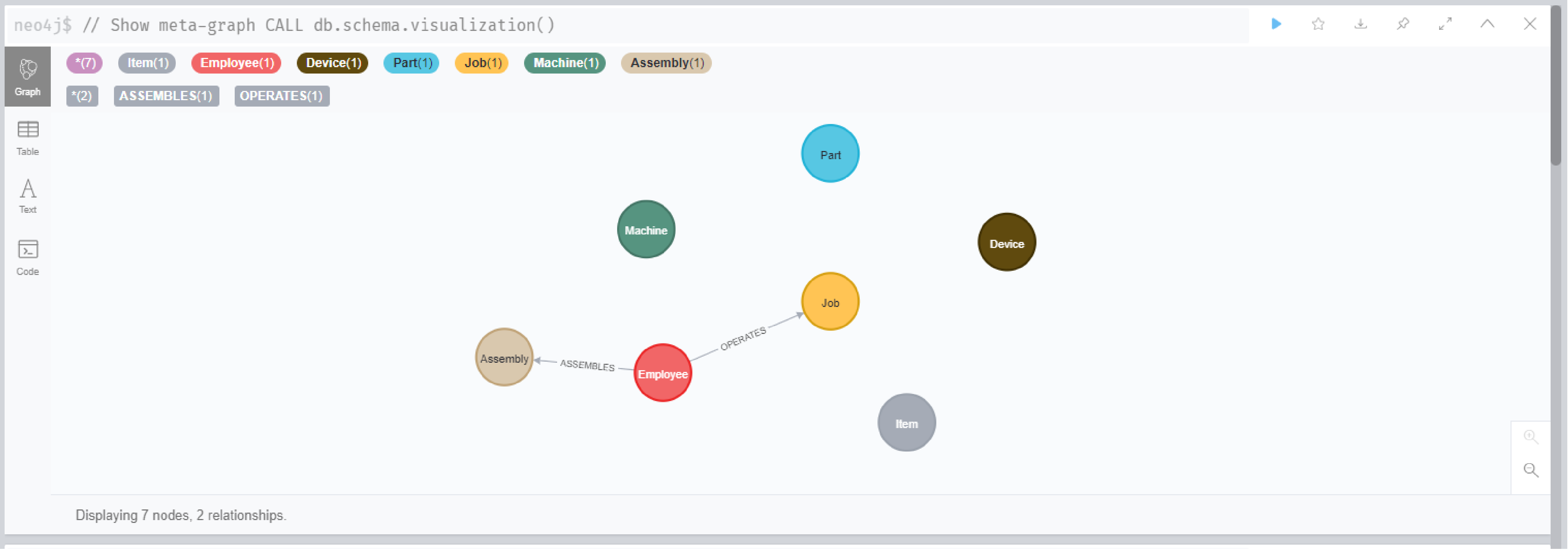
```
Created 49 relationships, completed after 195 ms.
```

```
// Show meta-graph
CALL db.schema.visualization()
```

*(7) Item(1) Employee(1) Device(1) Part(1) Job(1) Machine(1) Assembly(1)

*(1) ASSEMBLES(1)

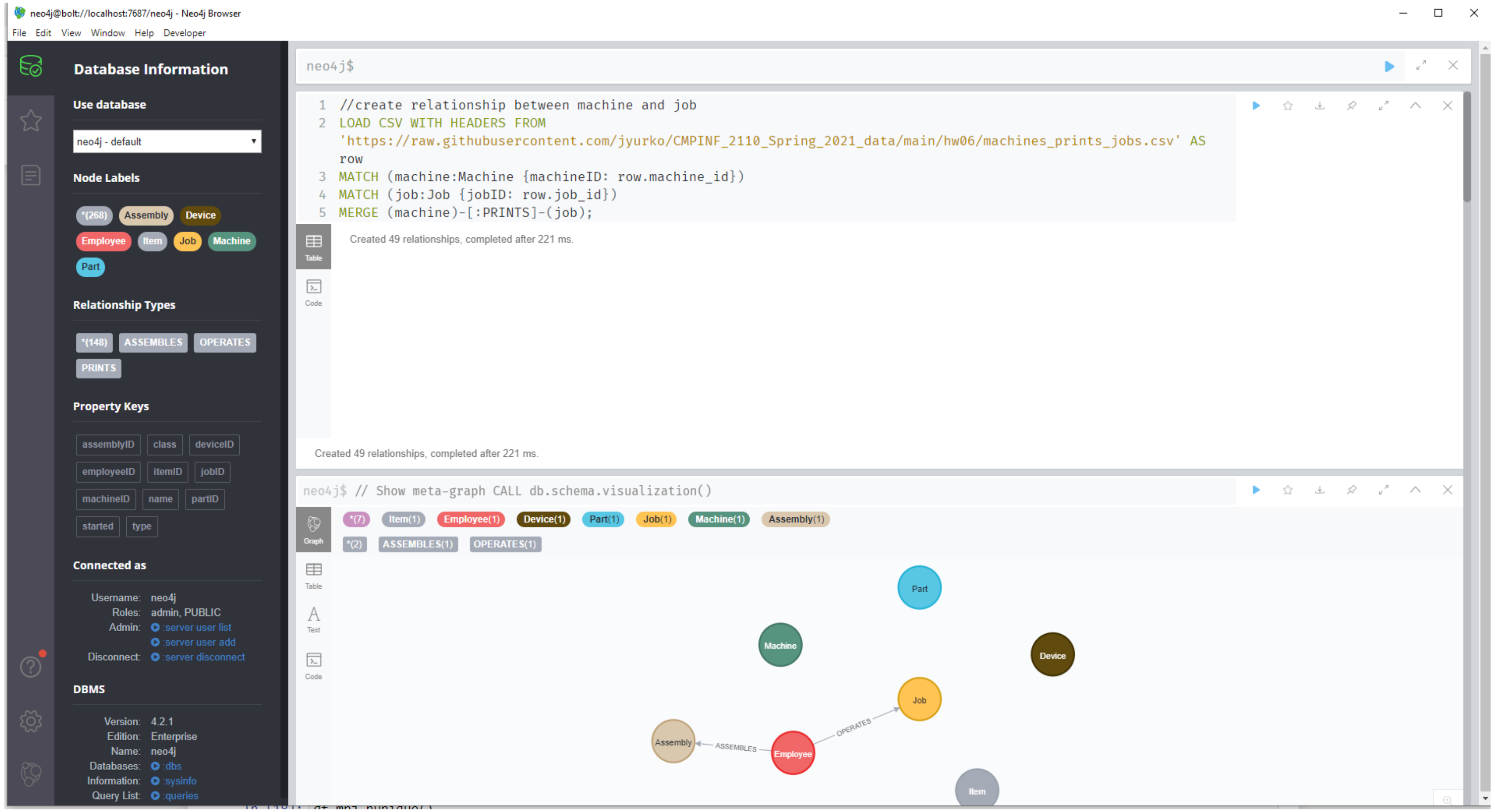
Check the meta-graph to see the two relationships



```
neo4j$ // Show meta-graph CALL db.schema.visualization()
```

```
Displaying 7 nodes, 2 relationships.
```

Create the PRINTS relationship type between Machine and Job nodes



Use database: neo4j - default

Username: neo4j
Roles: admin, PUBLIC
Admin: :server user list
:server user add
Disconnect: :server disconnect

Version: 4.2.1
Edition: Enterprise
Name: neo4j
Databases: :dbs
Information: :sysinfo
Query List: :queries

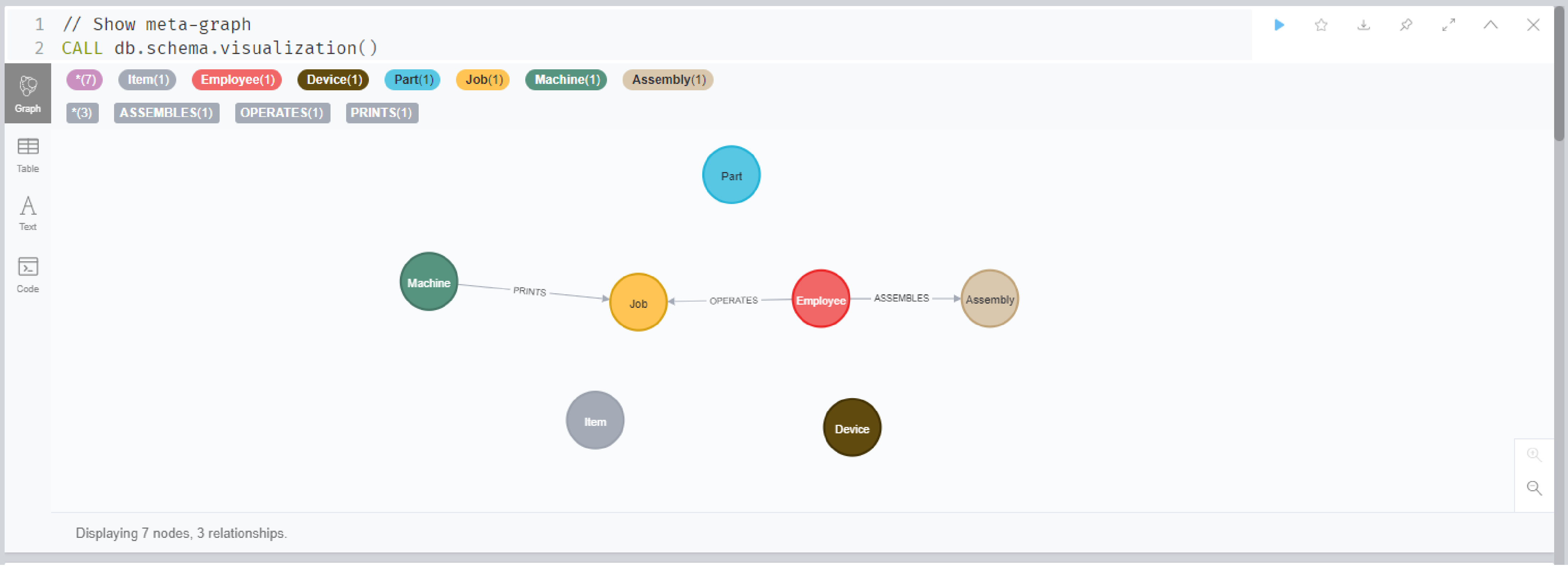
```
//create relationship between machine and job
LOAD CSV WITH HEADERS FROM 'https://raw.githubusercontent.com/jyurko/CMPINF_2110_Spring_2021_data/main/hw06/machines_prints_jobs.csv' AS row
MATCH (machine:Machine {machineID: row.machine_id})
MATCH (job:Job {jobID: row.job_id})
MERGE (machine)-[:PRINTS]-(job);
```

Created 49 relationships, completed after 221 ms.

Created 49 relationships, completed after 221 ms.

```
neo4j$ // Show meta-graph CALL db.schema.visualization()
```

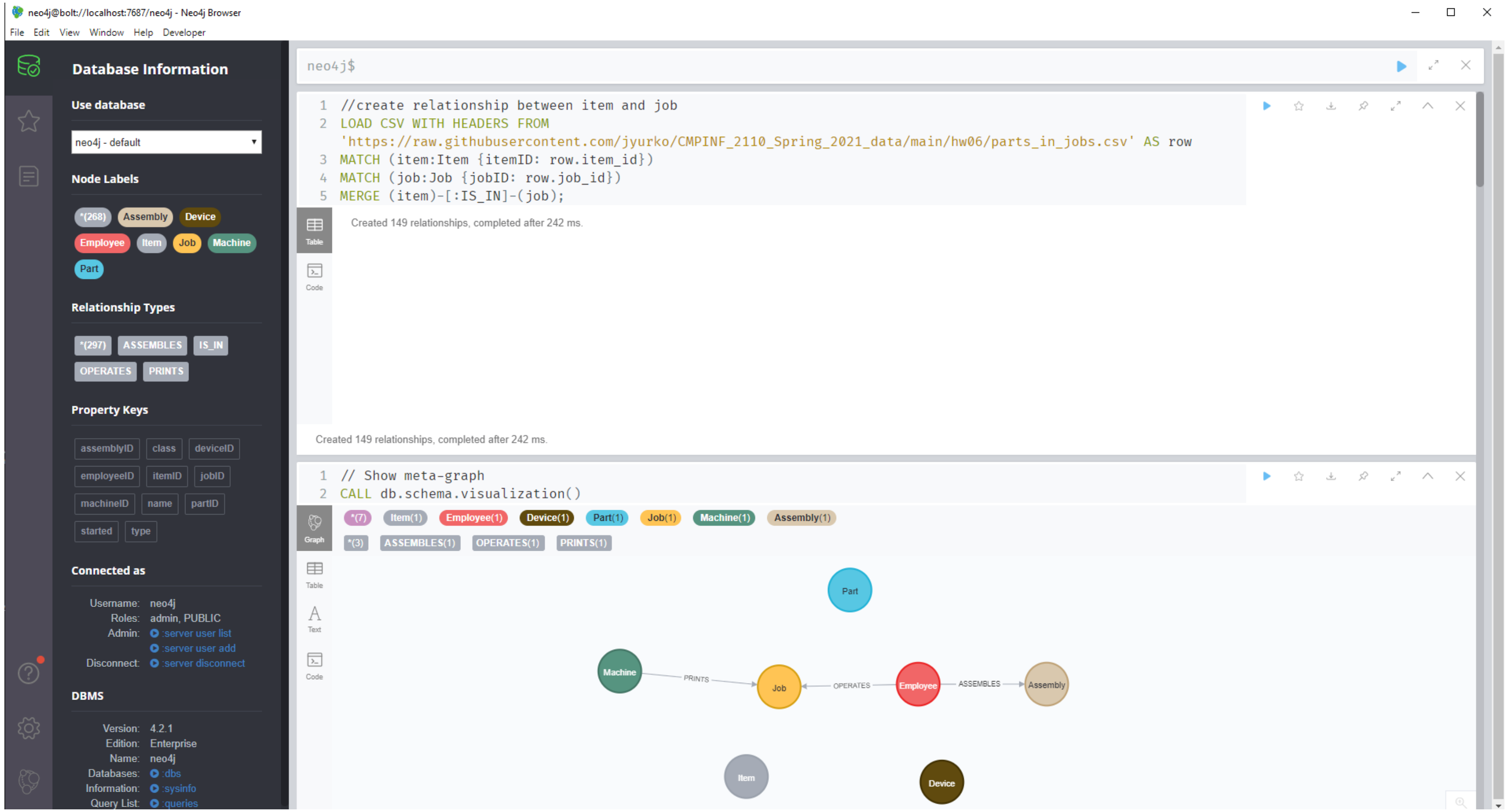
The meta-graph shows the 3 relationships currently in the graph database



```
// Show meta-graph
CALL db.schema.visualization()
```

Displaying 7 nodes, 3 relationships.

Create the IS_IN relationship type between Item and Job nodes


neo4j@bolt://localhost:7687/neo4j - Neo4j Browser
File Edit View Window Help Developer
Database Information
Use database
neo4j - default
Node Labels
*(268) Assembly Device Employee Item Job Machine Part
Relationship Types
*(297) ASSEMBLES IS_IN OPERATES PRINTS
Property Keys
assemblyID class deviceID employeeID itemID jobID machineID name partID started type
Connected as
DBMS
neo4j$
Table
Code
Graph
Text
*(7) Item(1) Employee(1) Device(1) Part(1) Job(1) Machine(1) Assembly(1)
*(3) ASSEMBLES(1) OPERATES(1) PRINTS(1)
Part
Machine
PRINTS
Job
OPERATES
Employee
ASSEMBLES
Assembly
Item
Device

Username: neo4j
Roles: admin, PUBLIC
Admin: :server user list
:server user add
Disconnect: :server disconnect

Version: 4.2.1
Edition: Enterprise
Name: neo4j
Databases: :dbs
Information: :sysinfo
Query List: :queries

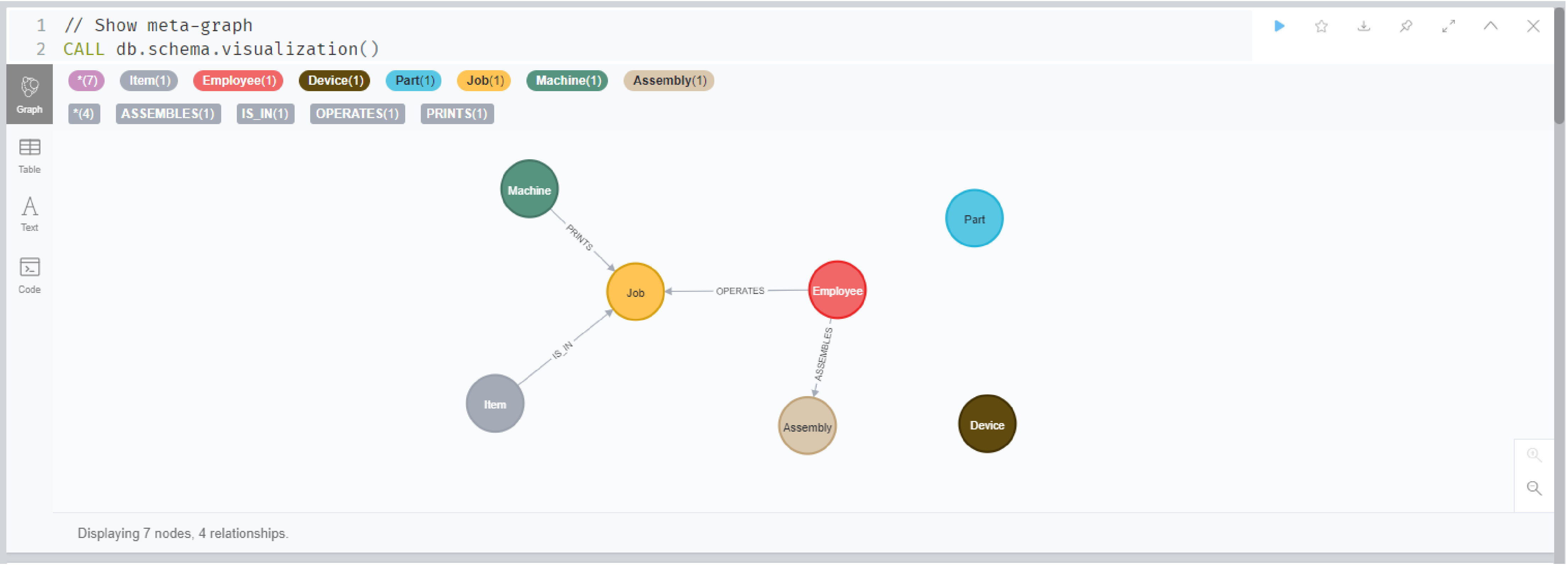
```
//create relationship between item and job
LOAD CSV WITH HEADERS FROM
'https://raw.githubusercontent.com/jyurko/CMPINF_2110_Spring_2021_data/main/hw06/parts_in_jobs.csv' AS row
MATCH (item:Item {itemID: row.item_id})
MATCH (job:Job {jobID: row.job_id})
MERGE (item)-[:IS_IN]-(job);
```

```
Created 149 relationships, completed after 242 ms.
```

```
Created 149 relationships, completed after 242 ms.
```

```
// Show meta-graph
CALL db.schema.visualization()
```

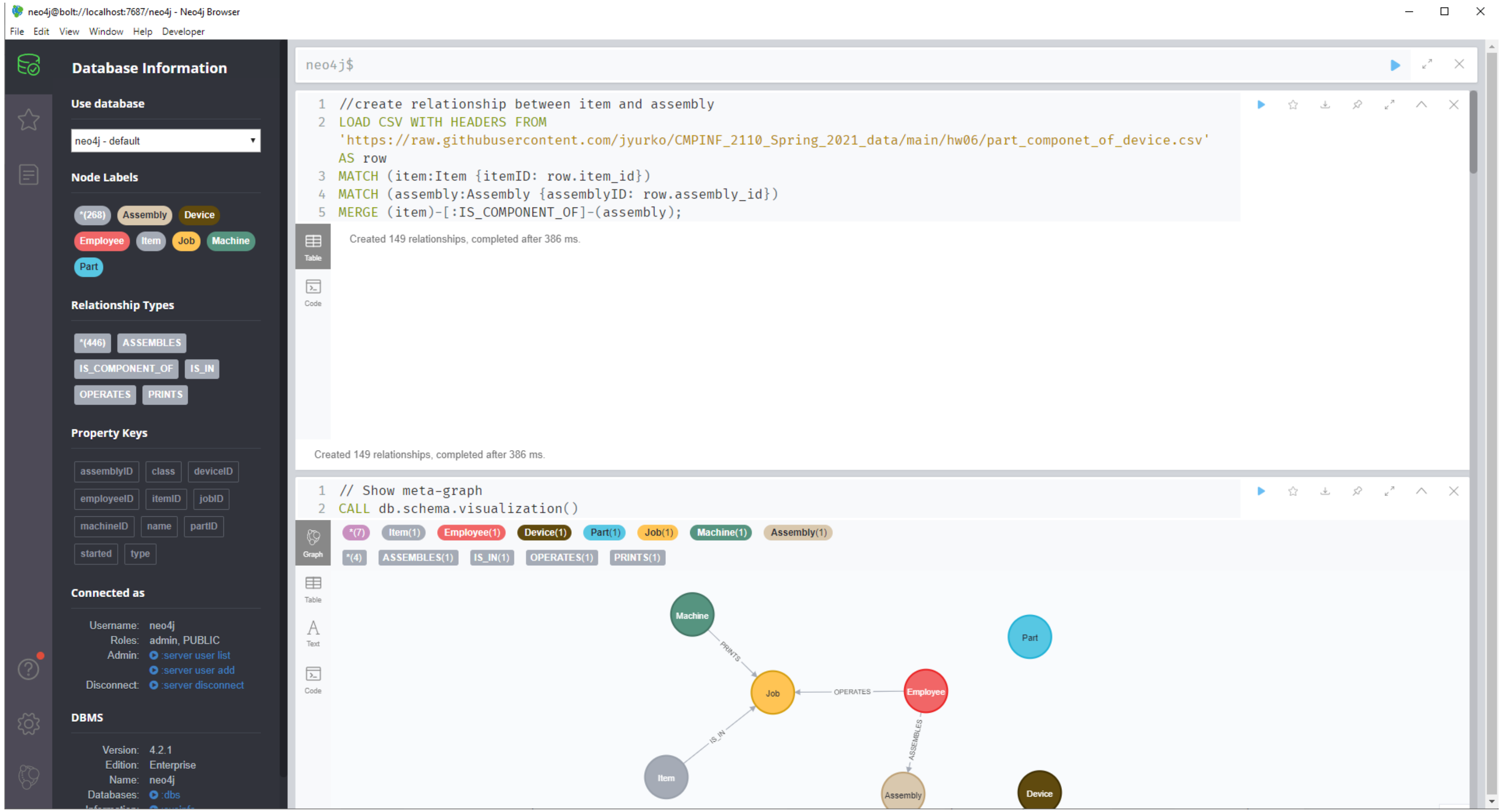
Meta-graph shows all the relationships associated with the Job Label



```
// Show meta-graph
CALL db.schema.visualization()
```

Displaying 7 nodes, 4 relationships.

Create the IS_COMPONENT_OF relationship type between Item and Assembly nodes



Username: neo4j
Roles: admin, PUBLIC
Admin: :server user list
:server user add
Disconnect: :server disconnect

Version: 4.2.1
Edition: Enterprise
Name: neo4j
Databases: :dbs

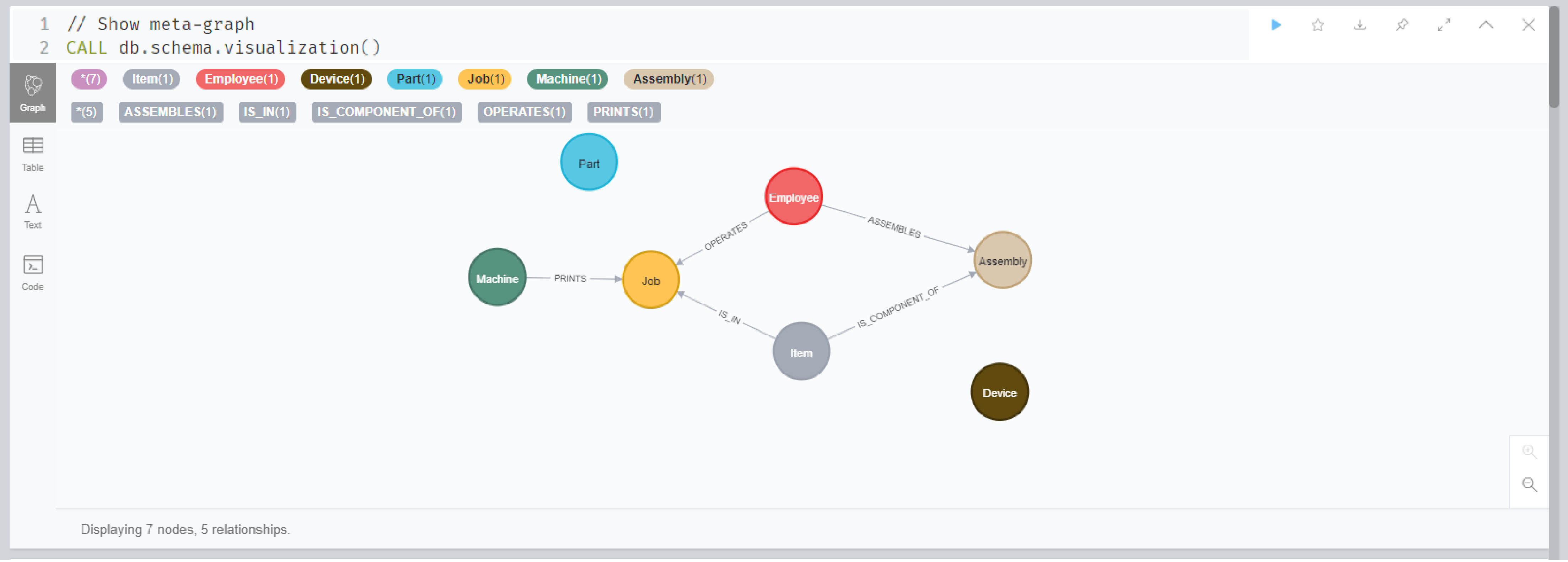
```
//create relationship between item and assembly
LOAD CSV WITH HEADERS FROM
'https://raw.githubusercontent.com/jyurko/CMPINF_2110_Spring_2021_data/main/hw06/part_componet_of_device.csv'
AS row
MATCH (item:Item {itemID: row.item_id})
MATCH (assembly:Assembly {assemblyID: row.assembly_id})
MERGE (item)-[:IS_COMPONENT_OF]-(assembly);
```

```
Created 149 relationships, completed after 386 ms.
```

```
Created 149 relationships, completed after 386 ms.
```

```
// Show meta-graph
CALL db.schema.visualization()
```

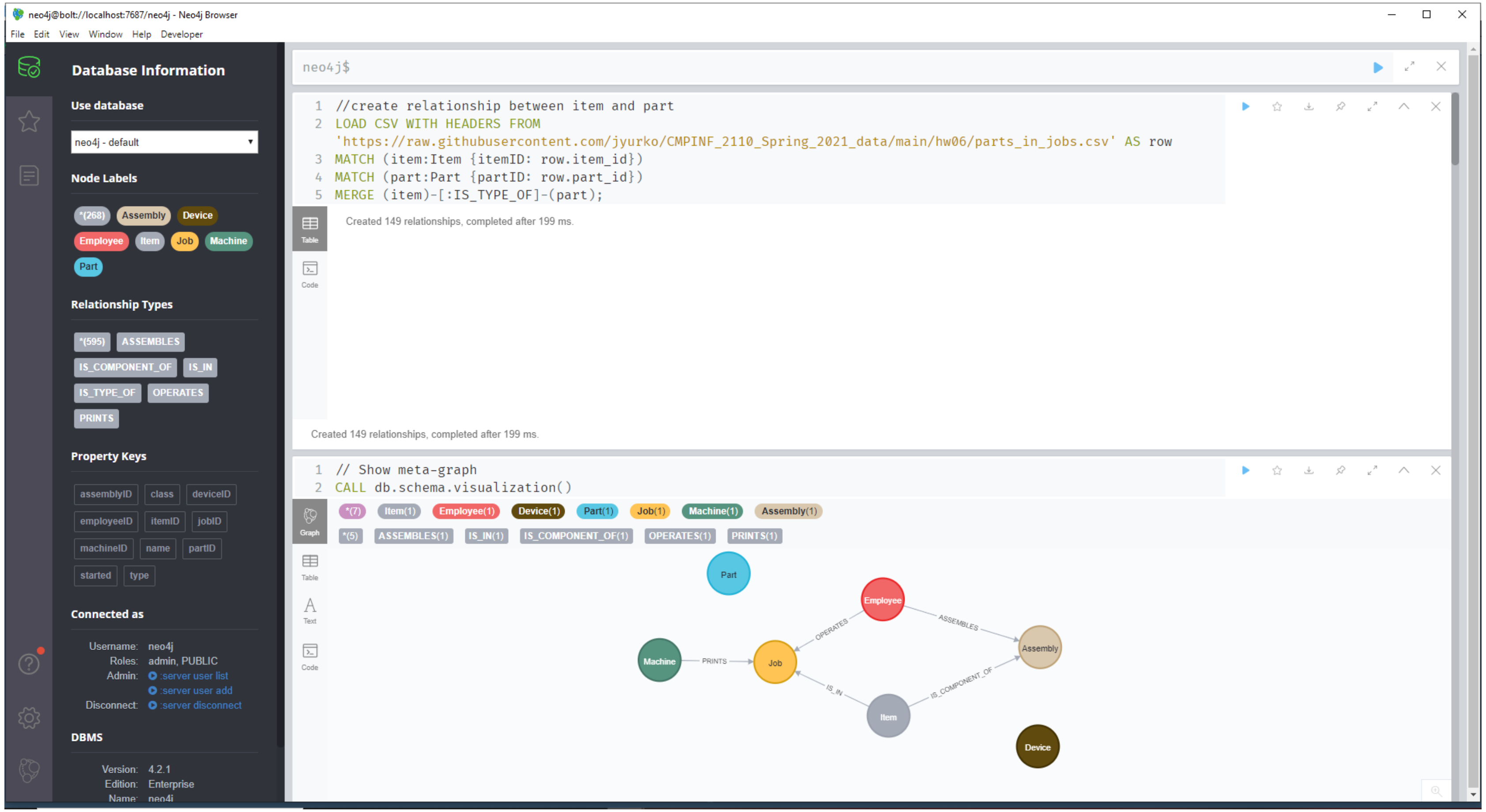
The meta-graph shows just the Part and Device labels are not connected in the network



```
// Show meta-graph
CALL db.schema.visualization()
```

Displaying 7 nodes, 5 relationships.

Create the IS_TYPE_OF relationship between the Item and Part nodes



neo4j - default

Username: neo4j
Roles: admin, PUBLIC
Admin: :server user list
:server user add
Disconnect: :server disconnect

Version: 4.2.1
Edition: Enterprise
Name: neo4j

```
neo4j$
```

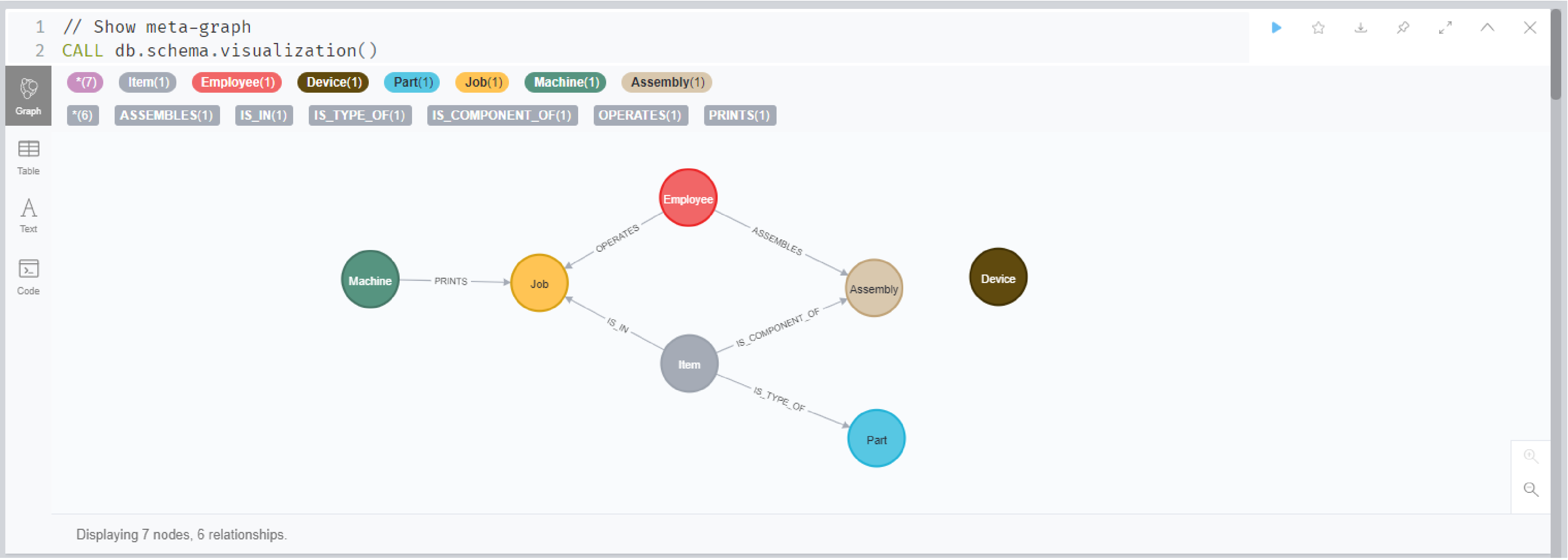
```
//create relationship between item and part
LOAD CSV WITH HEADERS FROM
'https://raw.githubusercontent.com/jyurko/CMPINF_2110_Spring_2021_data/main/hw06/parts_in_jobs.csv' AS row
MATCH (item:Item {itemID: row.item_id})
MATCH (part:Part {partID: row.part_id})
MERGE (item)-[:IS_TYPE_OF]-(part);
```

Created 149 relationships, completed after 199 ms.

Created 149 relationships, completed after 199 ms.

```
// Show meta-graph
CALL db.schema.visualization()
```

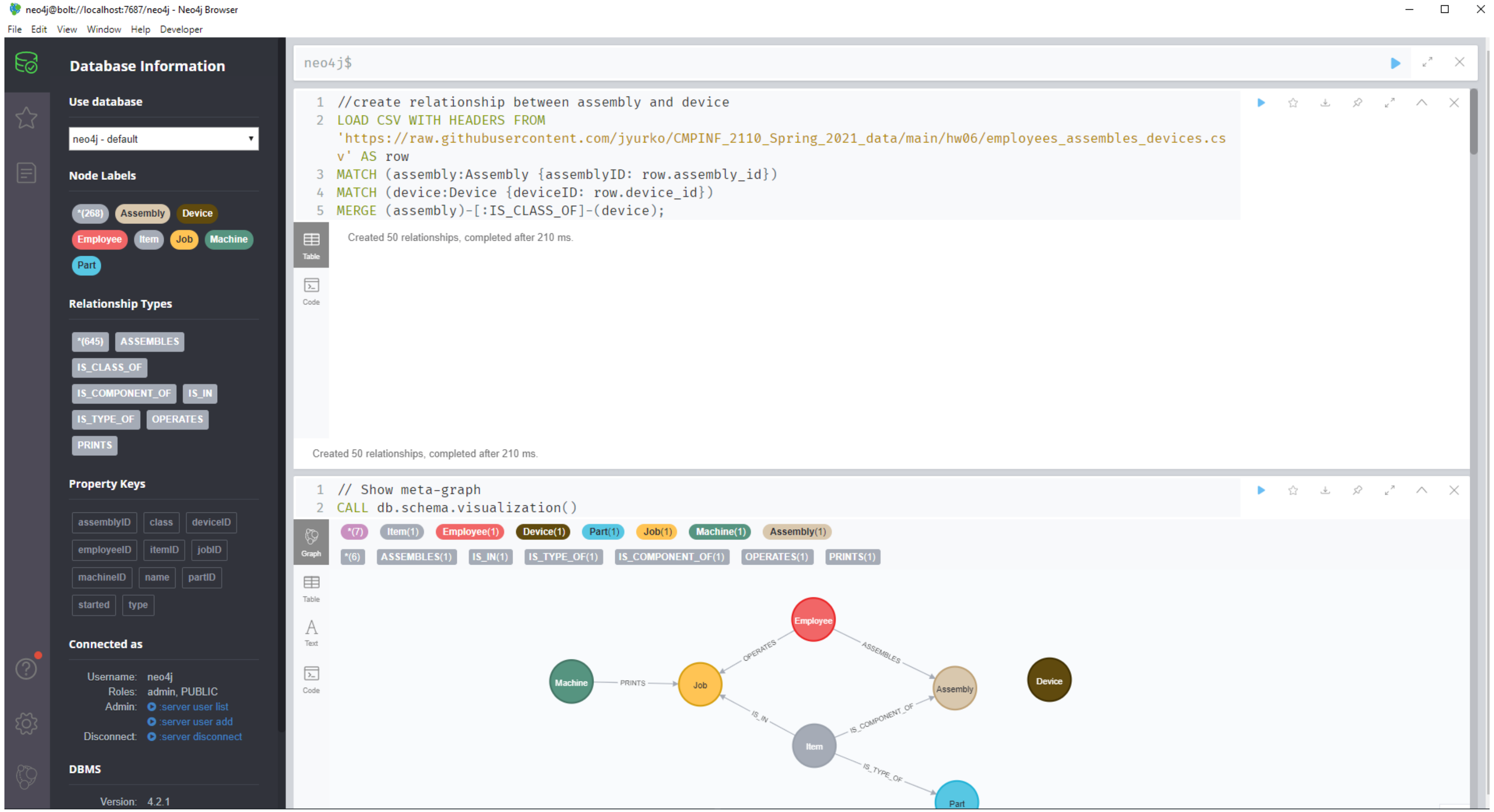
Meta-graph shows only the Device label is not connected to other Labels



```
// Show meta-graph
CALL db.schema.visualization()
```

Displaying 7 nodes, 6 relationships.

Create the IS_CLASS_OF relationship between the Assembly and Device nodes



neo4j@bolt://localhost:7687/neo4j - Neo4j Browser
File Edit View Window Help Developer
Database Information
Use database
Node Labels
*(268) Assembly Device Employee Item Job Machine Part
Relationship Types
*(645) ASSEMBLES IS_CLASS_OF IS_COMPONENT_OF IS_IN IS_TYPE_OF OPERATES PRINTS
Property Keys
assemblyID class deviceID employeeID itemID jobID machineID name partID started type
Connected as
DBMS
Table
Code
Graph
Text
*(7) Item(1) Employee(1) Device(1) Part(1) Job(1) Machine(1) Assembly(1)
*(6) ASSEMBLES(1) IS_IN(1) IS_TYPE_OF(1) IS_COMPONENT_OF(1) OPERATES(1) PRINTS(1)
Employee
OPERATES
ASSEMBLES
Machine
PRINTS
Job
Assembly
Device
IS_IN
IS_COMPONENT_OF
Item
IS_TYPE_OF
Part

Use database: neo4j - default

Username: neo4j
Roles: admin, PUBLIC
Admin: :server user list
:server user add
Disconnect: :server disconnect

Version: 4.2.1

```
neo4j$
```

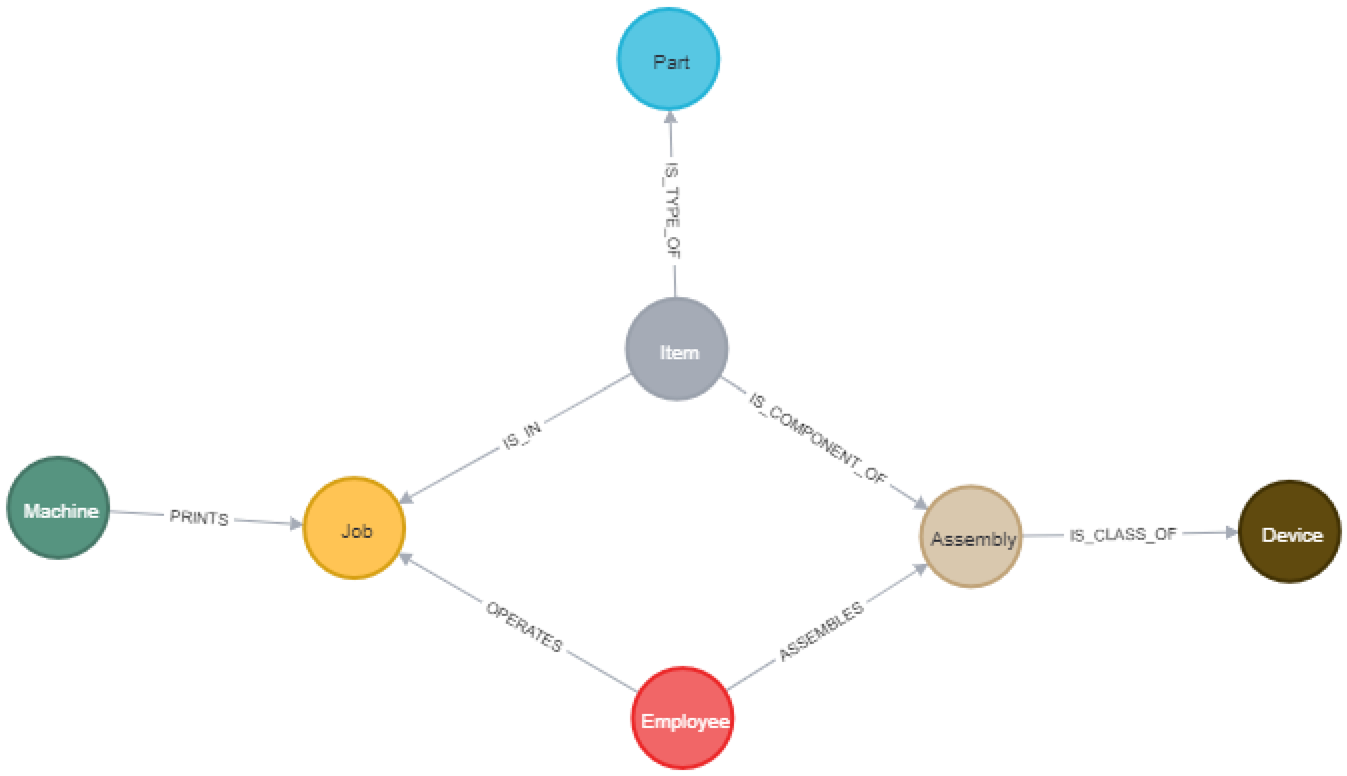
```
//create relationship between assembly and device
LOAD CSV WITH HEADERS FROM
'https://raw.githubusercontent.com/jyurko/CMPINF_2110_Spring_2021_data/main/hw06/employees_assembles_devices.cs
v' AS row
MATCH (assembly:Assembly {assemblyID: row.assembly_id})
MATCH (device:Device {deviceID: row.device_id})
MERGE (assembly)-[:IS_CLASS_OF]-(device);
```

```
Created 50 relationships, completed after 210 ms.
```

```
Created 50 relationships, completed after 210 ms.
```

```
// Show meta-graph
CALL db.schema.visualization()
```

Completed graph schema!



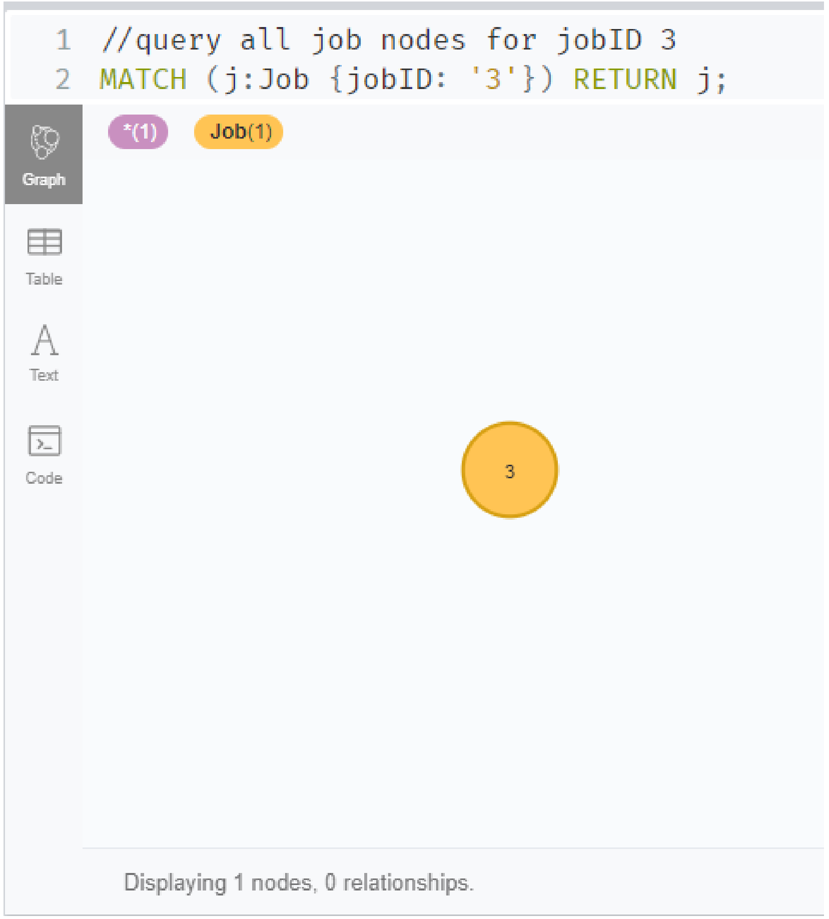
Required queries

1. Query all nodes related to the job_id = 3 Job.
2. Count all parts in job_id = 3.
3. Query all nodes related to the Employee Alice.
4. Count all devices assembled by Employee Chuck.
5. Count all parts printed by the Machine delta.

1. Query all nodes related to job_id = 3

- If we only query the Job nodes associated with jobID='3', we will only get a single node!

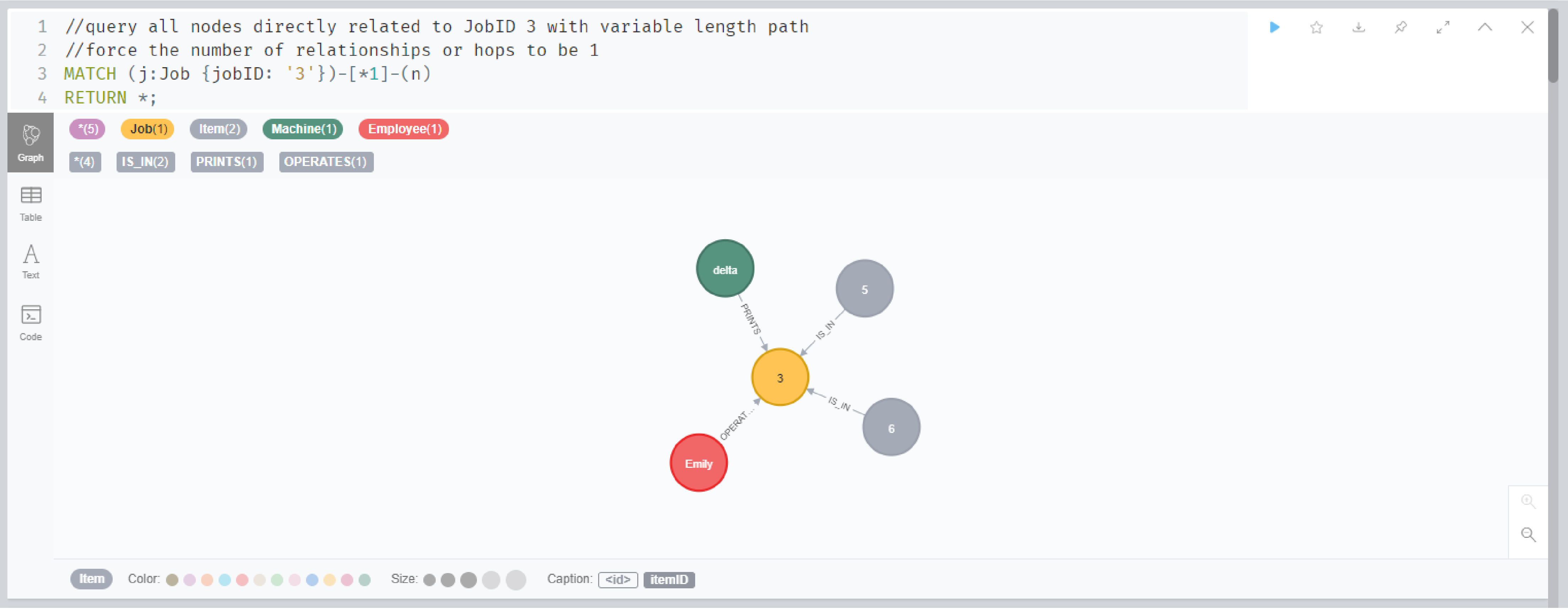
- This is NOT what this particularly query was looking for.



```
//query all job nodes for jobID 3
MATCH (j:Job {jobID: '3'}) RETURN j;
```

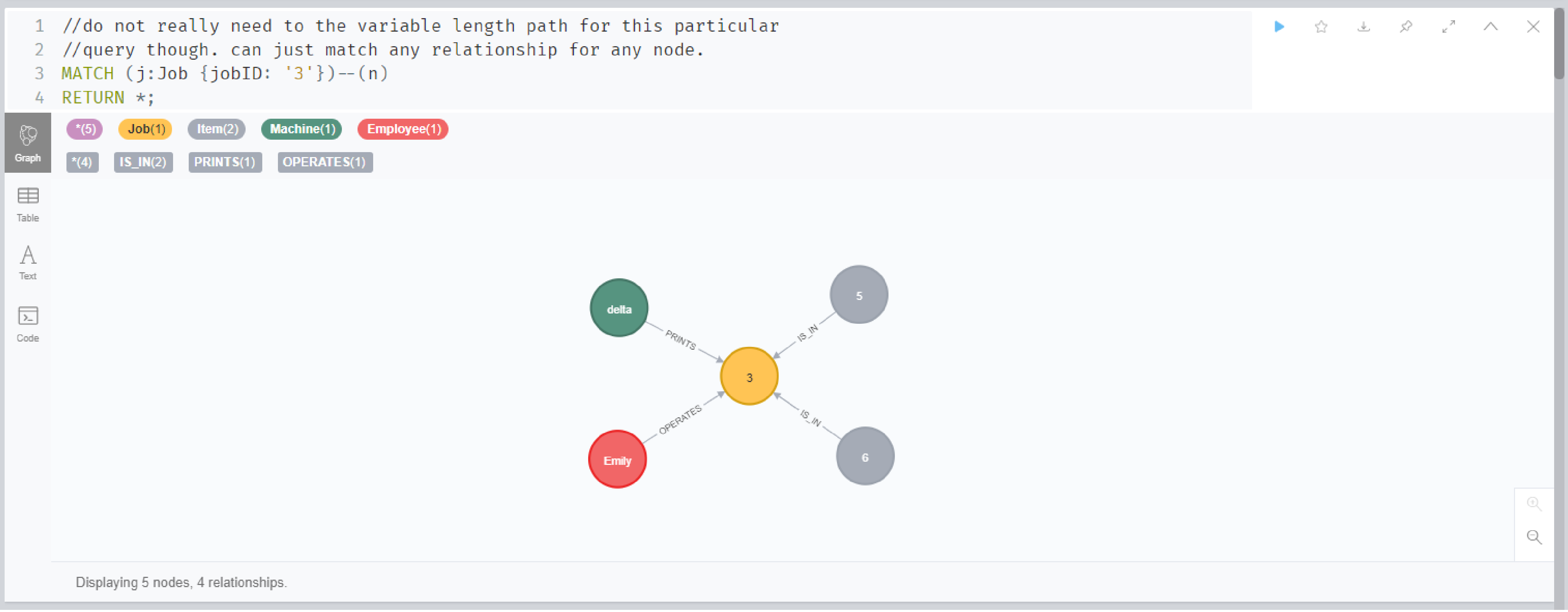
Displaying 1 nodes, 0 relationships.

A variable length path makes it easy to query any node label within 1 relationship or hop away from the Job node with jobID = ‘3’



```
//query all nodes directly related to JobID 3 with variable length path
//force the number of relationships or hops to be 1
MATCH (j:Job {jobID: '3'})-[*1]-(n)
RETURN *;
```

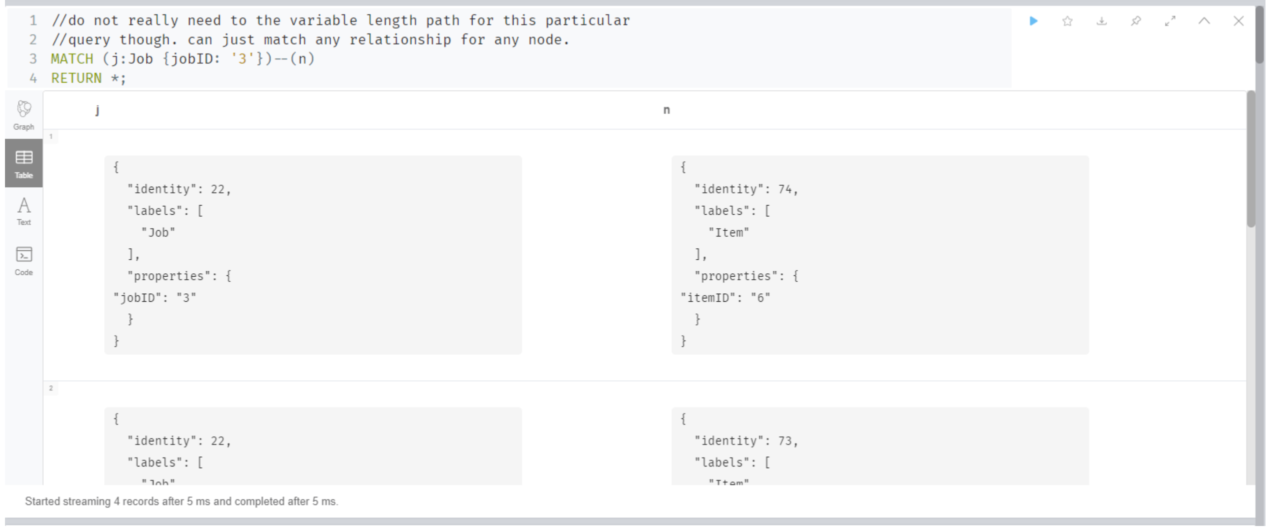
However, we do not need a variable length path for this particular query. We can simply query ANY node Label related in any way to the Job node with jobID='3'



```
//do not really need to the variable length path for this particular
//query though. can just match any relationship for any node.
MATCH (j:Job {jobID: '3'})--(n)
RETURN *;
```

Displaying 5 nodes, 4 relationships.

Table view gives the “JSON-like” representation of the query, which is saved as CSV file



```
//do not really need to the variable length path for this particular
//query though. can just match any relationship for any node.
MATCH (j:Job {jobID: '3'})--(n)
RETURN *;
```

j	n

1

```
{
  "identity": 22,
  "labels": [
    "Job"
  ],
  "properties": {
"jobID": "3"
  }
}
```

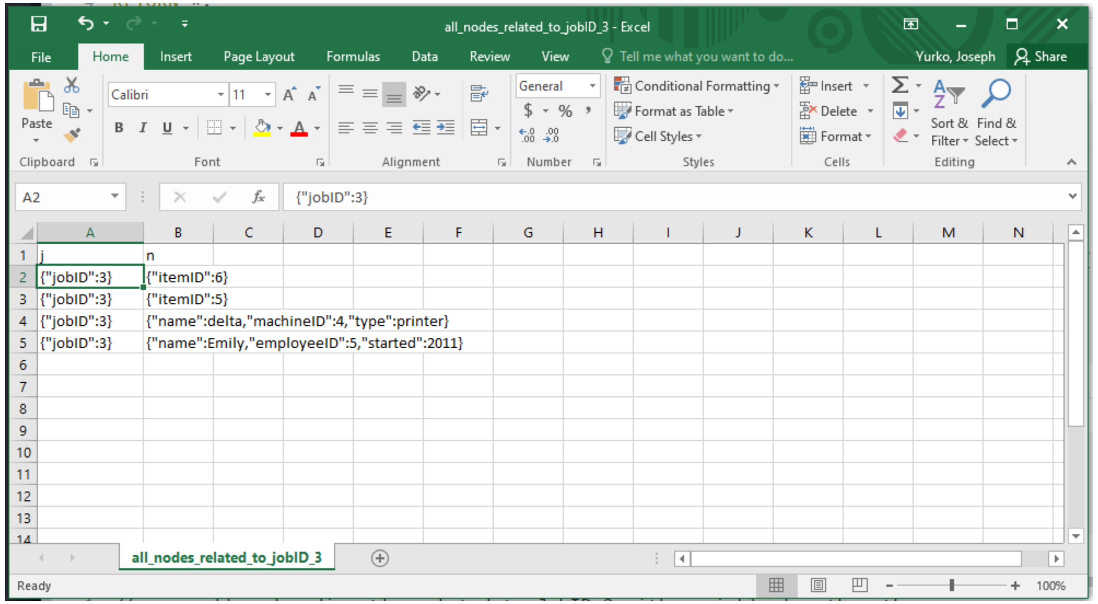
```
{
  "identity": 74,
  "labels": [
    "Item"
  ],
  "properties": {
"itemID": "6"
  }
}
```

2

```
{
  "identity": 22,
  "labels": [
    "Job"
```

```
{
  "identity": 73,
  "labels": [
    "Item"
```

Started streaming 4 records after 5 ms and completed after 5 ms.



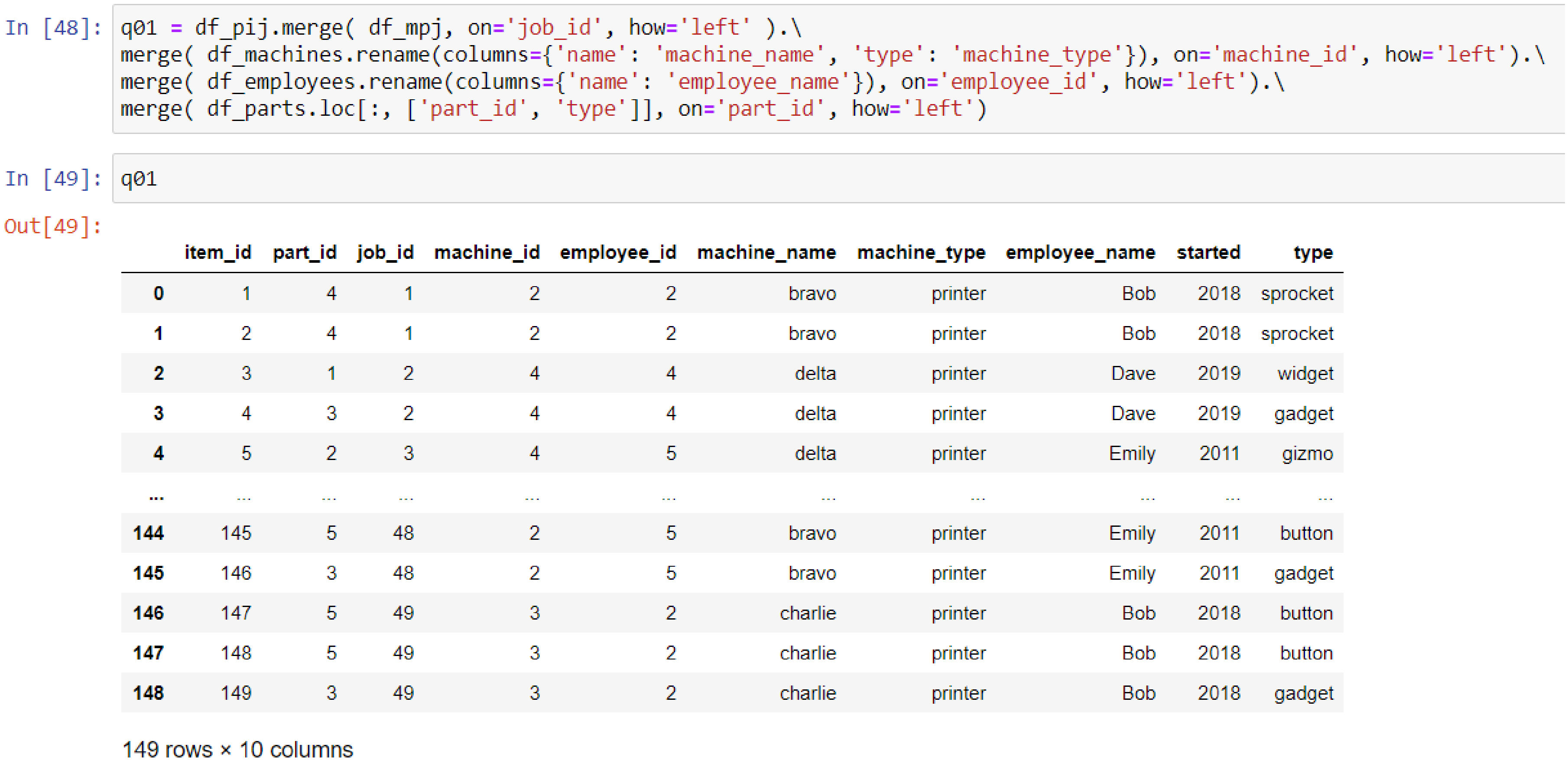
all_nodes_related_to_jobID_3 - Excel

A2 {"jobID":3}

	A	B
1	j	n
2	{"jobID":3}	{"itemID":6}
3	{"jobID":3}	{"itemID":5}
4	{"jobID":3}	{"name":delta,"machineID":4,"type":printer}
5	{"jobID":3}	{"name":Emily,"employeeID":5,"started":2011}

Let’s check our graph query with Pandas

We first need to JOIN all necessary DataFrames together



```
In [48]: q01 = df_pij.merge( df_mpj, on='job_id', how='left' ).\
merge( df_machines.rename(columns={'name': 'machine_name', 'type': 'machine_type'}), on='machine_id', how='left').\
merge( df_employees.rename(columns={'name': 'employee_name'}), on='employee_id', how='left').\
merge( df_parts.loc[:, ['part_id', 'type']], on='part_id', how='left')
```

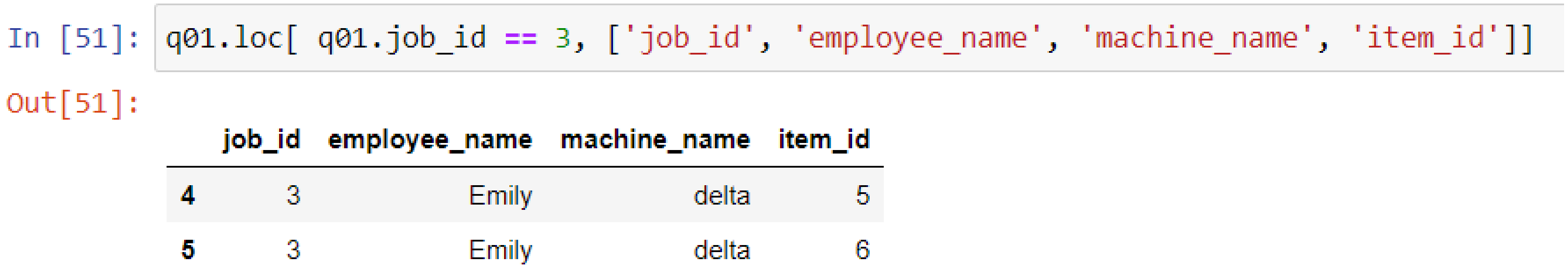
```
In [49]: q01
```

Out[49]:

	item_id	part_id	job_id	machine_id	employee_id	machine_name	machine_type	employee_name	started	type
0	1	4	1	2	2	bravo	printer	Bob	2018	sprocket
1	2	4	1	2	2	bravo	printer	Bob	2018	sprocket
2	3	1	2	4	4	delta	printer	Dave	2019	widget
3	4	3	2	4	4	delta	printer	Dave	2019	gadget
4	5	2	3	4	5	delta	printer	Emily	2011	gizmo
...	...	...	...	...	...	...	...	...	...	...
144	145	5	48	2	5	bravo	printer	Emily	2011	button
145	146	3	48	2	5	bravo	printer	Emily	2011	gadget
146	147	5	49	3	2	charlie	printer	Bob	2018	button
147	148	5	49	3	2	charlie	printer	Bob	2018	button
148	149	3	49	3	2	charlie	printer	Bob	2018	gadget

149 rows × 10 columns

Then, filter or subset to the specific Job of interest

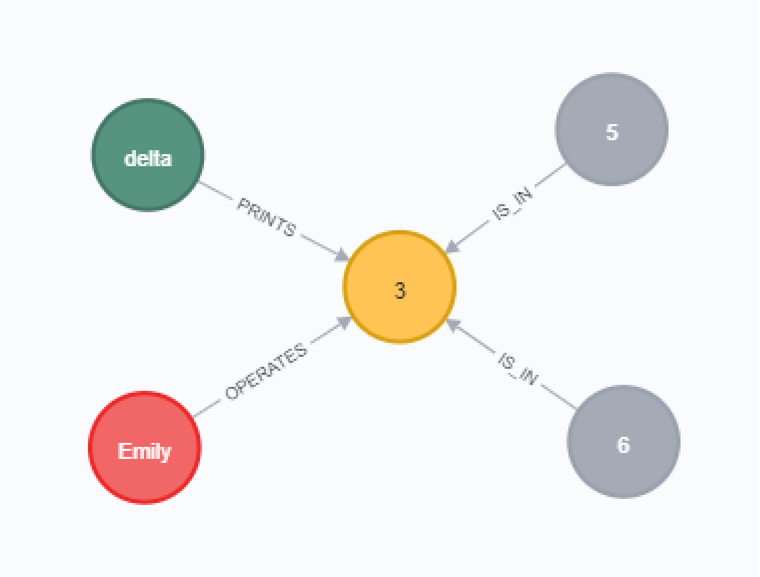


```
In [51]: q01.loc[ q01.job_id == 3, ['job_id', 'employee_name', 'machine_name', 'item_id']]
```

Out[51]:

	job_id	employee_name	machine_name	item_id
4	3	Emily	delta	5
5	3	Emily	delta	6

We get the SAME information as returned by our GRAPH!

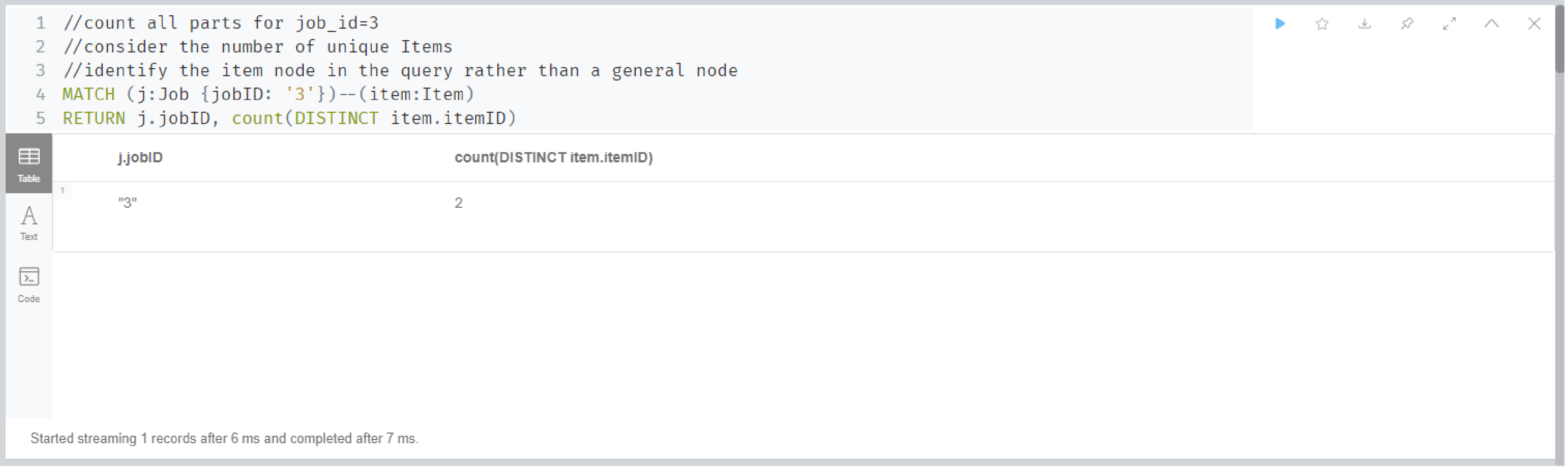


2. Count all parts in job_id = 3.

- The previous query revealed that there are 2 items printed in job_id = 3.

- We can count the number of distinct items using the DISTINCT keyword and count() functions in Cypher.

Query Job and Item nodes related in any way, return the number of unique items per job



```
//count all parts for job_id=3
//consider the number of unique Items
//identify the item node in the query rather than a general node
MATCH (j:Job {jobID: '3'})--(item:Item)
RETURN j.jobID, count(DISTINCT item.itemID)
```

	j.jobID	count(DISTINCT item.itemID)
1	"3"	2

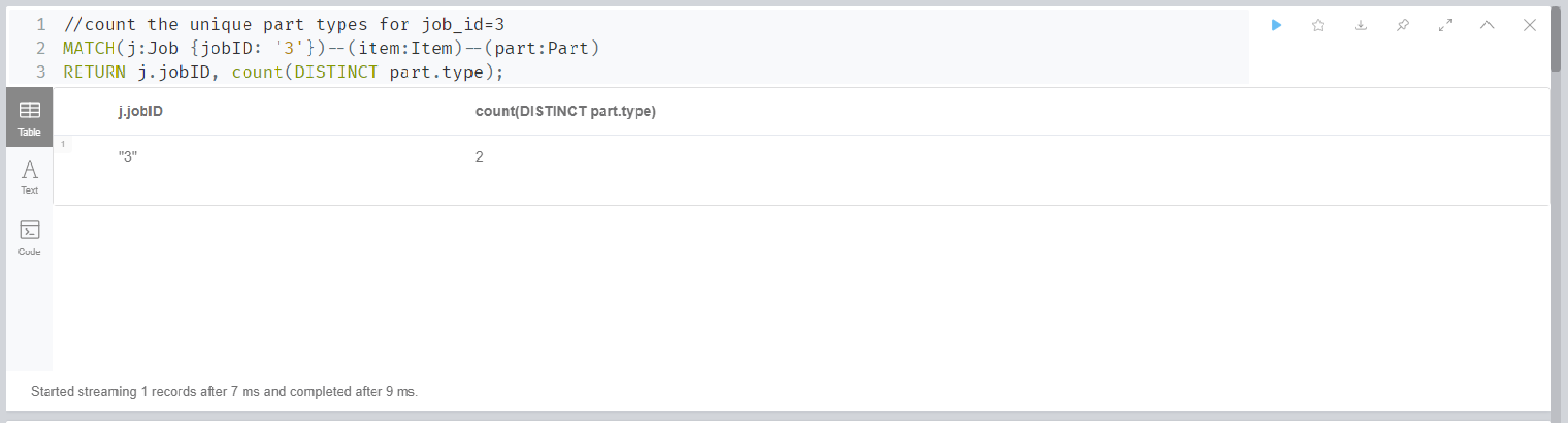
Started streaming 1 records after 6 ms and completed after 7 ms.

Alternatively, we could count the unique TYPES of parts for job_id = 3.

- Requires including the relationship between the Item node and Part node.

- But allows us to count the number of unique TYPES of parts for a given job.

We get the same result as the previous query which counted the unique number of Items



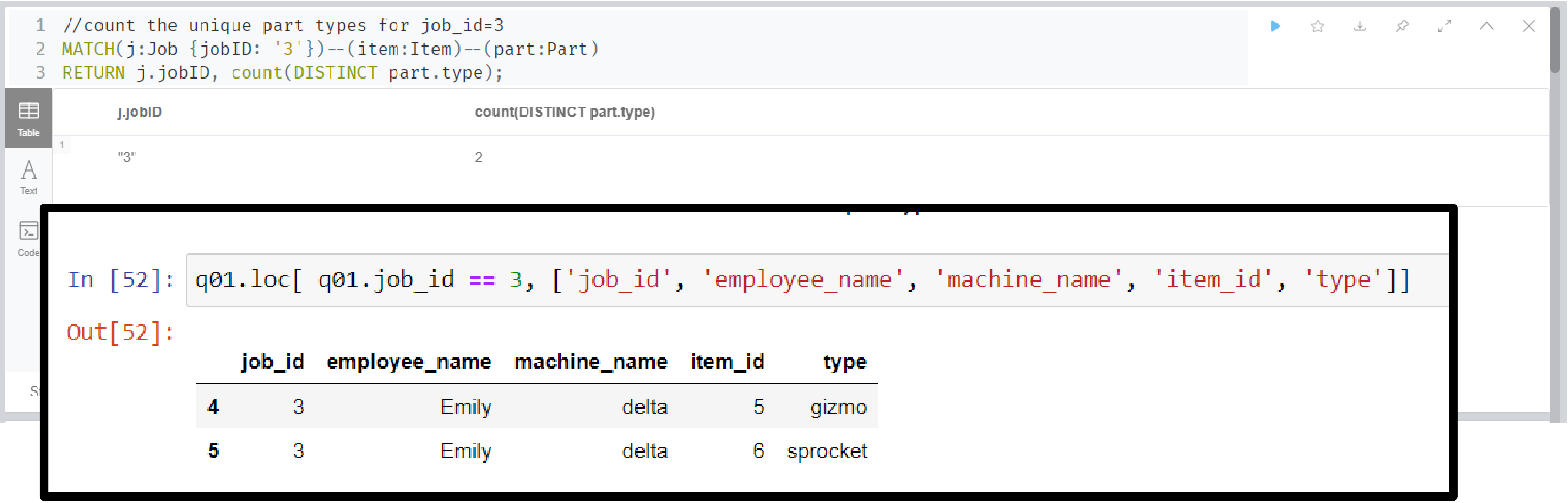
```
//count the unique part types for job_id=3
MATCH(j:Job {jobID: '3'})--(item:Item)--(part:Part)
RETURN j.jobID, count(DISTINCT part.type);
```

	j.jobID	count(DISTINCT part.type)
1	"3"	2

Started streaming 1 records after 7 ms and completed after 9 ms.

Confirm this is indeed the case with Pandas

As we see below, the two items are two different types of parts!



```
//count the unique part types for job_id=3
MATCH(j:Job {jobID: '3'})--(item:Item)--(part:Part)
RETURN j.jobID, count(DISTINCT part.type);
```

	j.jobID	count(DISTINCT part.type)
1	"3"	2

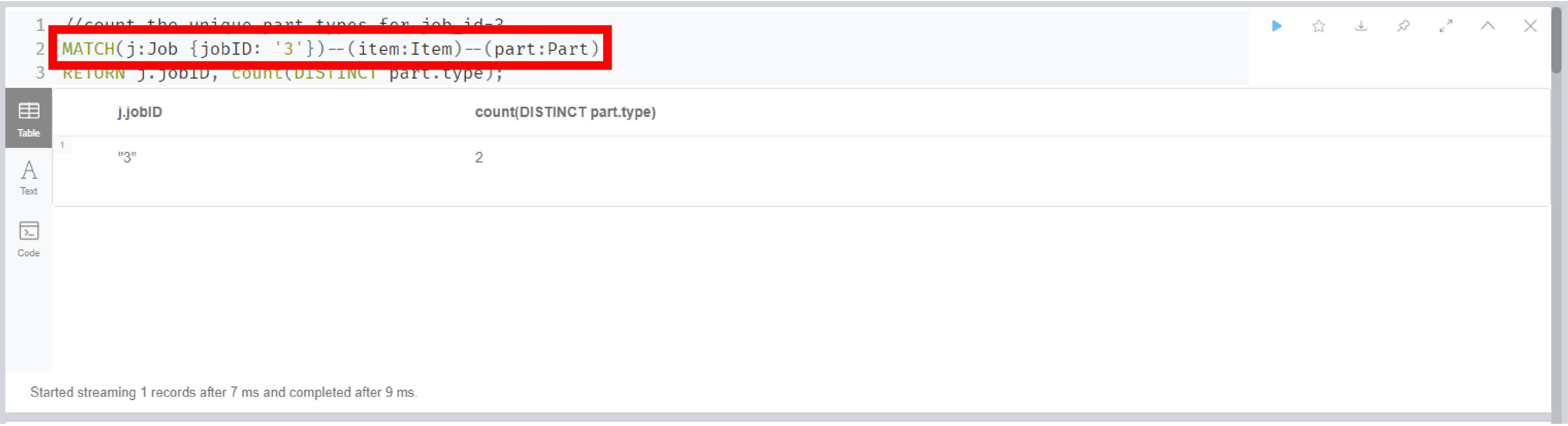
```
In [52]: q01.loc[ q01.job_id == 3, ['job_id', 'employee_name', 'machine_name', 'item_id', 'type']]
```

Out[52]:

	job_id	employee_name	machine_name	item_id	type
4	3	Emily	delta	5	gizmo
5	3	Emily	delta	6	sprocket

If we do not care about the directionality of the relationship, we do not need to specify it

No arrows are included in the pattern!



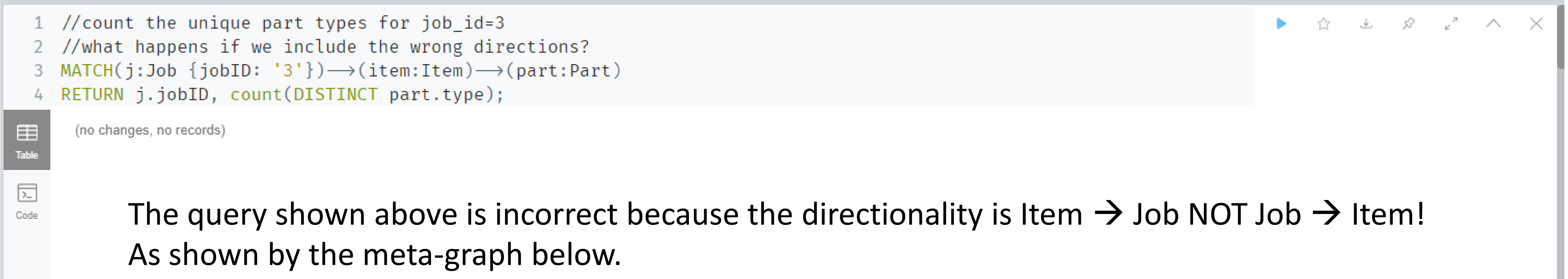
```
//count the unique part types for job_id=3
MATCH(j:Job {jobID: '3'})--(item:Item)--(part:Part)
RETURN j.jobID, count(DISTINCT part.type);
```

j.jobID	count(DISTINCT part.type)
"3"	2

Started streaming 1 records after 7 ms and completed after 9 ms.

If we want a specific directional relationship, we must be sure that is the direction embedded in the graph

For example, if we give the WRONG directional relationship no results will be returned!

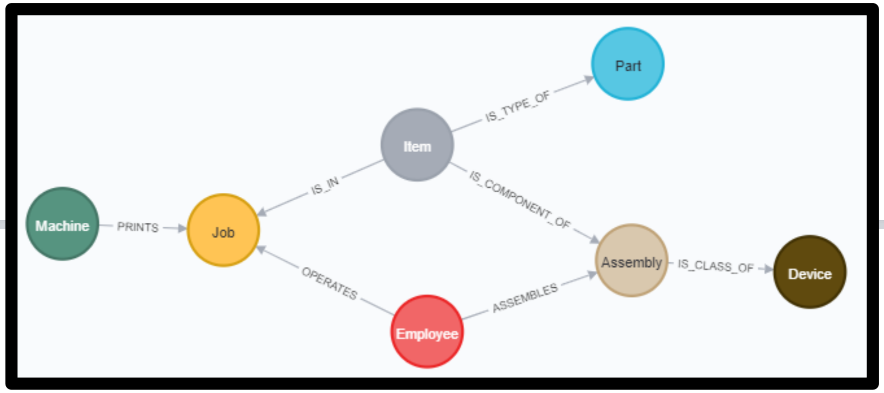


```
//count the unique part types for job_id=3
//what happens if we include the wrong directions?
MATCH(j:Job {jobID: '3'})-->(item:Item)-->(part:Part)
RETURN j.jobID, count(DISTINCT part.type);
```

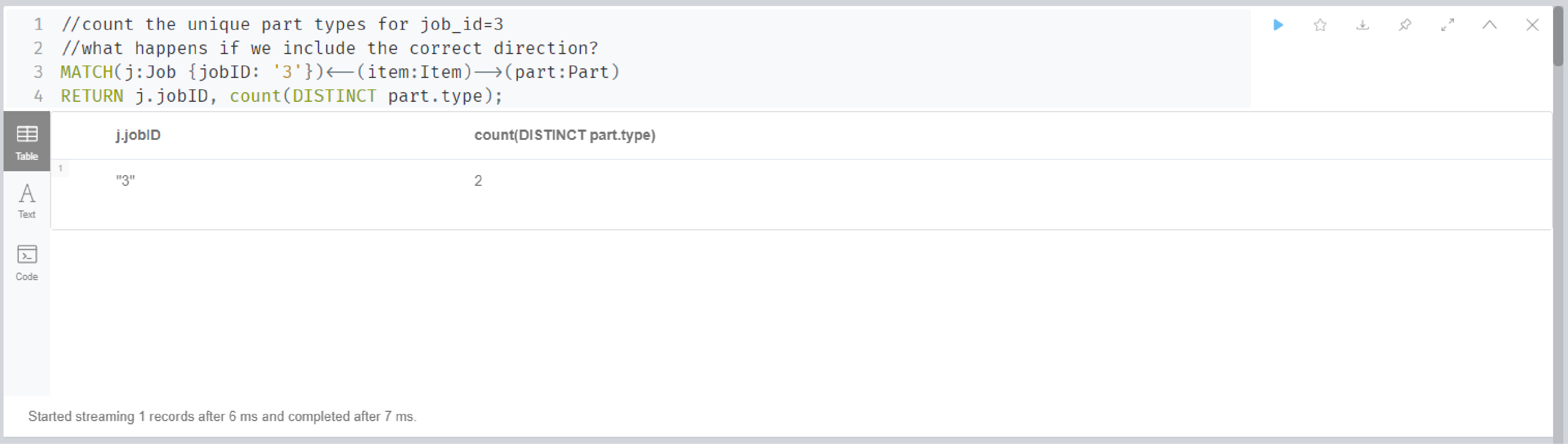
(no changes, no records)

Completed after 7 ms.

The query shown above is incorrect because the directionality is Item → Job NOT Job → Item! As shown by the meta-graph below.



The correct directional query has the arrows pointing away from the Item node



```
//count the unique part types for job_id=3
//what happens if we include the correct direction?
MATCH(j:Job {jobID: '3'})<--(item:Item)-->(part:Part)
RETURN j.jobID, count(DISTINCT part.type);
```

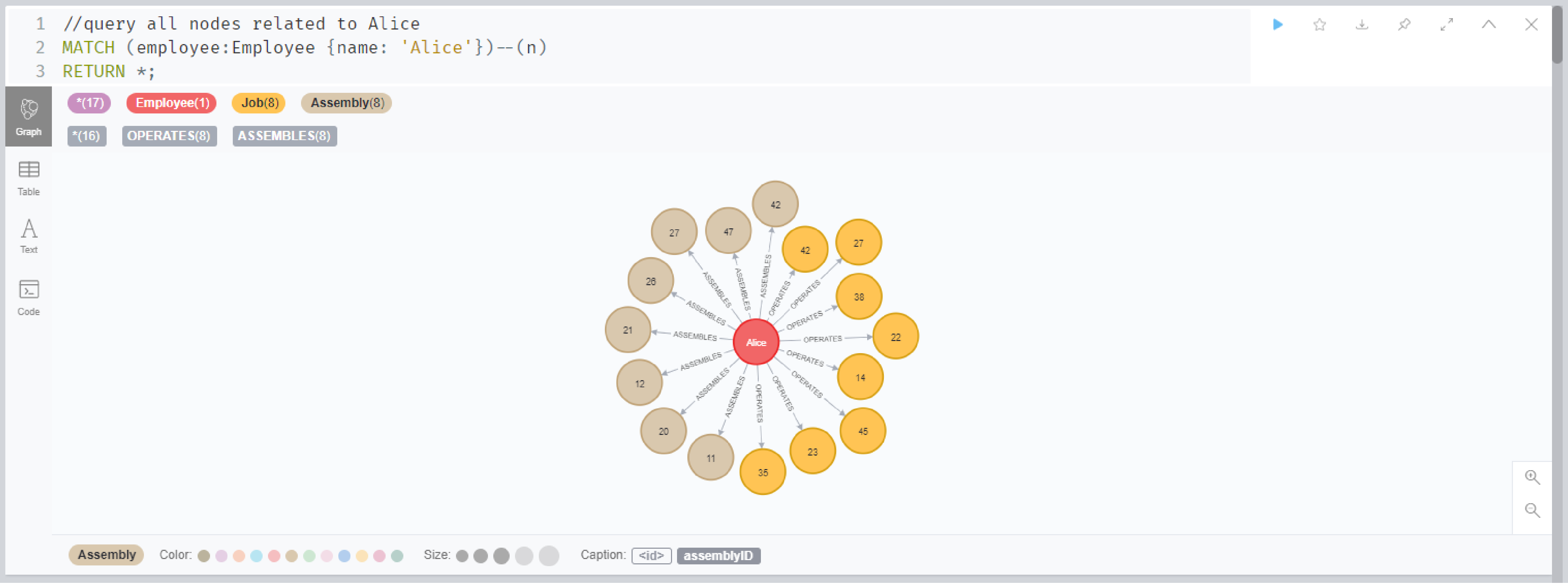
	j.jobID	count(DISTINCT part.type)
1	"3"	2

Started streaming 1 records after 6 ms and completed after 7 ms.

3. Query all nodes related to the Employee Alice.

- This query can be structured multiple ways.

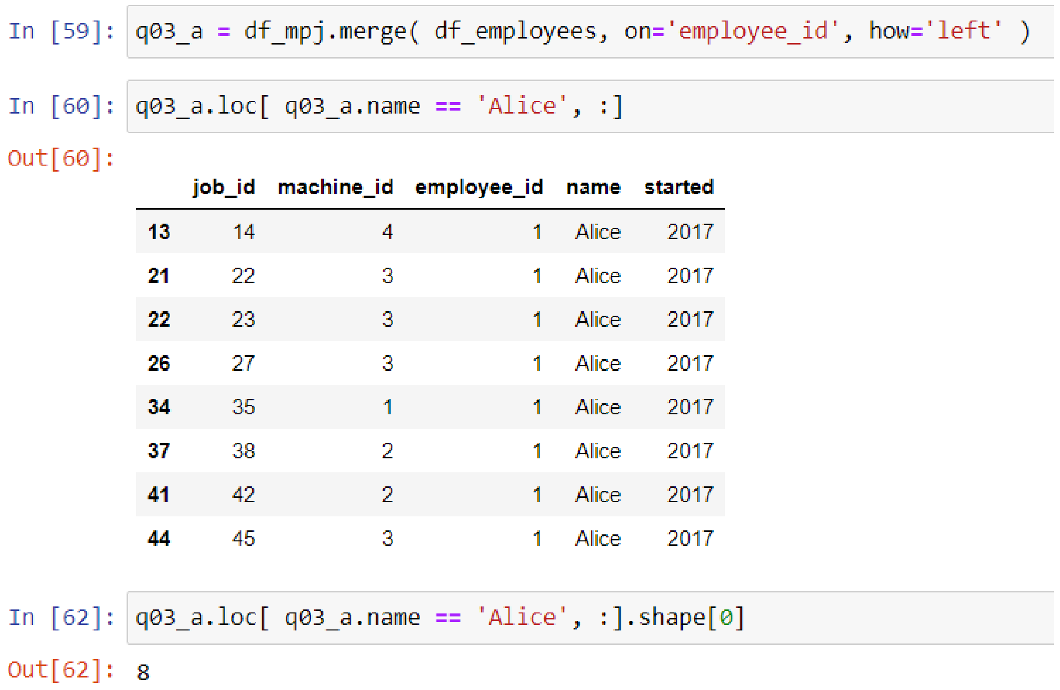
One way is to specify any node of any relationship type with the Employee node



```
//query all nodes related to Alice
MATCH (employee:Employee {name: 'Alice'})--(n)
RETURN *;
```

There are 8 Jobs and 8 Assemblies connected with Alice.

To check that this is correct, let’s use Pandas to count the number of Jobs which Alice was the machine operator.

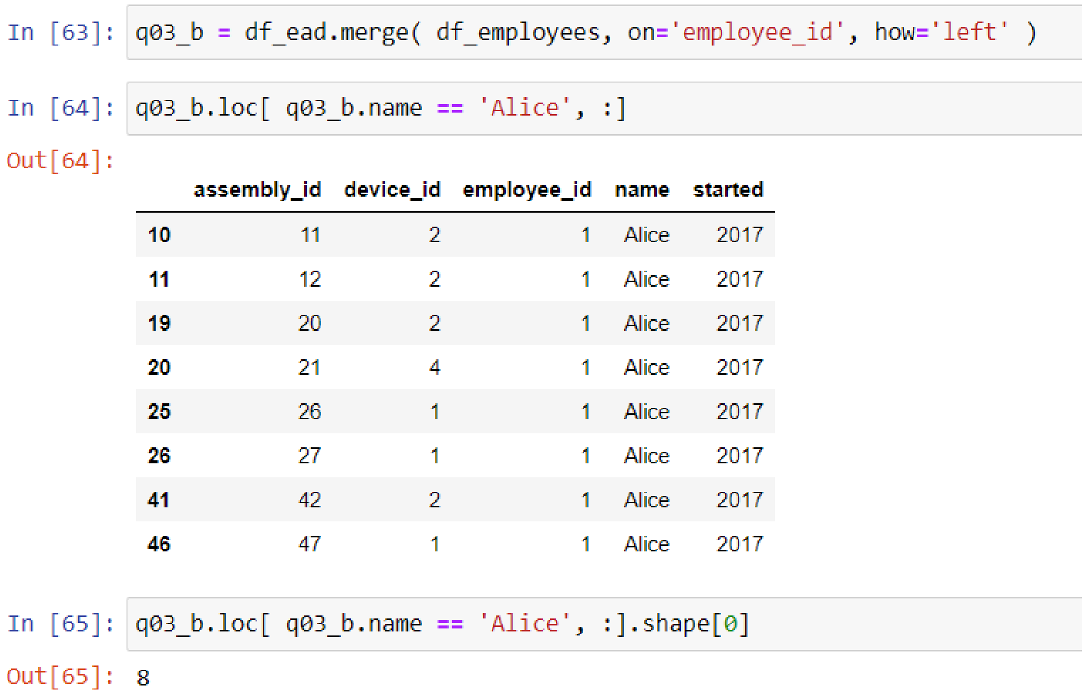


```
In [59]: q03_a = df_mpj.merge( df_employees, on='employee_id', how='left' )

In [60]: q03_a.loc[ q03_a.name == 'Alice', :]
Out[60]:
```

	job_id	machine_id	employee_id	name	started
13	14	4	1	Alice	2017
21	22	3	1	Alice	2017
22	23	3	1	Alice	2017
26	27	3	1	Alice	2017
34	35	1	1	Alice	2017
37	38	2	1	Alice	2017
41	42	2	1	Alice	2017
44	45	3	1	Alice	2017

```
In [62]: q03_a.loc[ q03_a.name == 'Alice', :].shape[0]
Out[62]: 8
```

Yes! There are 8 jobs!



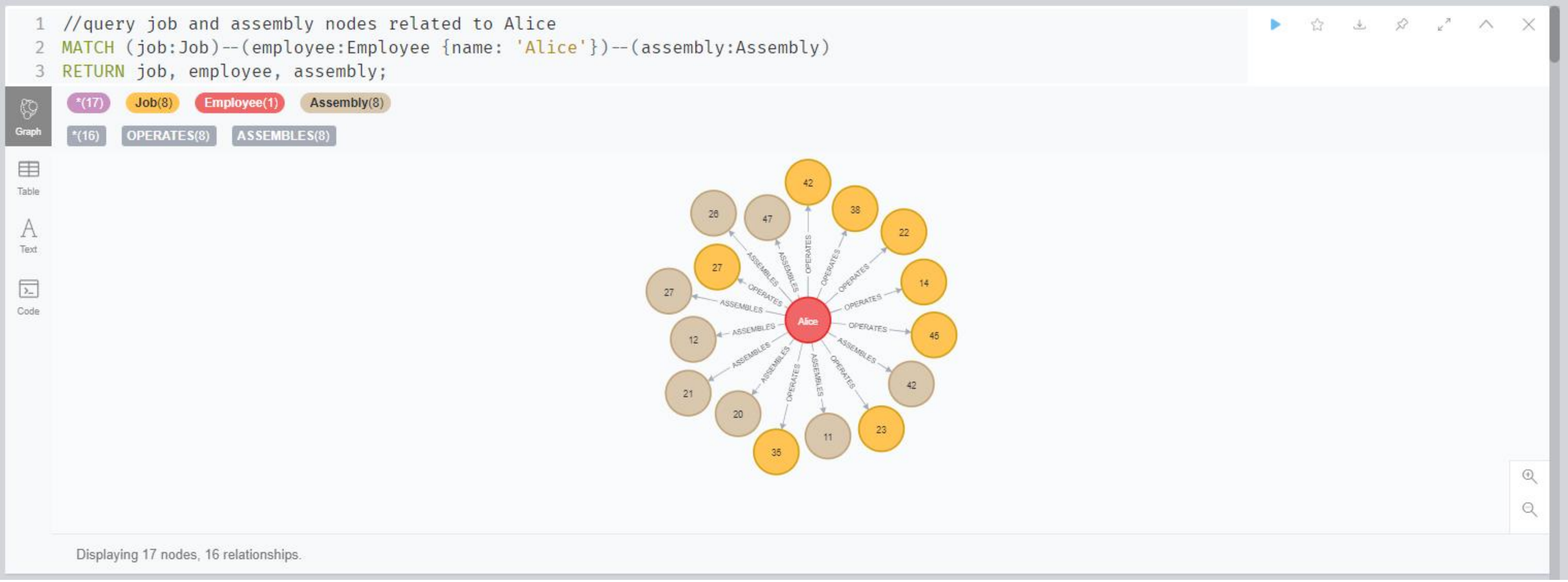
```
In [63]: q03_b = df_ead.merge( df_employees, on='employee_id', how='left' )

In [64]: q03_b.loc[ q03_b.name == 'Alice', :]
Out[64]:
```

	assembly_id	device_id	employee_id	name	started
10	11	2	1	Alice	2017
11	12	2	1	Alice	2017
19	20	2	1	Alice	2017
20	21	4	1	Alice	2017
25	26	1	1	Alice	2017
26	27	1	1	Alice	2017
41	42	2	1	Alice	2017
46	47	1	1	Alice	2017

```
In [65]: q03_b.loc[ q03_b.name == 'Alice', :].shape[0]
Out[65]: 8
```

Yes! There are 8 assemblies!

Alternatively, we can perform the same query by providing the specific node Labels we are interested in.



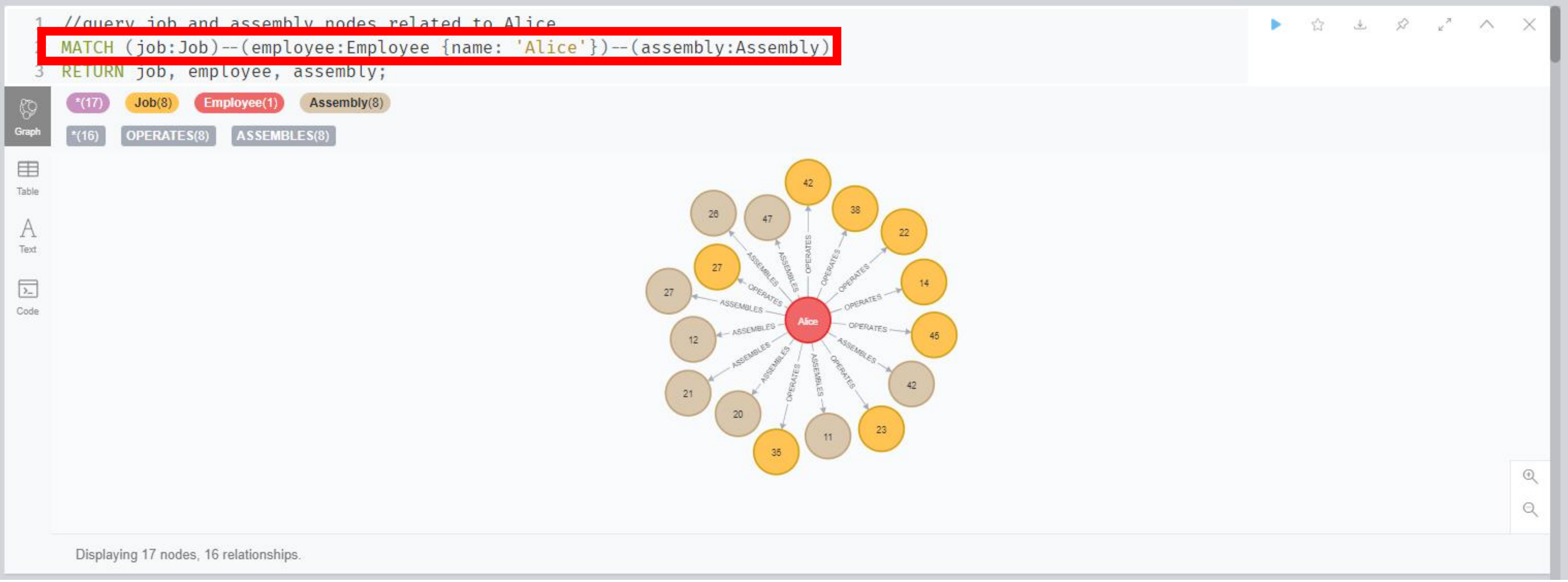
```
//query job and assembly nodes related to Alice
MATCH (job:Job)--(employee:Employee {name: 'Alice'})--(assembly:Assembly)
RETURN job, employee, assembly;
```

Displaying 17 nodes, 16 relationships.

The number of nodes and relationships returned are the same as the previous query. Though the graphical ordering of the nodes is slightly different.

Alternatively, we can perform the same query by providing the specific node Labels we are interested in.

Easier to structure this query pattern with the Employee node in the middle.



```
//query job and assembly nodes related to Alice
MATCH (job:Job)--(employee:Employee {name: 'Alice'})--(assembly:Assembly)
RETURN job, employee, assembly;
```

Displaying 17 nodes, 16 relationships.

The number of nodes and relationships returned are the same as the previous query.
Though the graphical ordering of the nodes is slightly different.

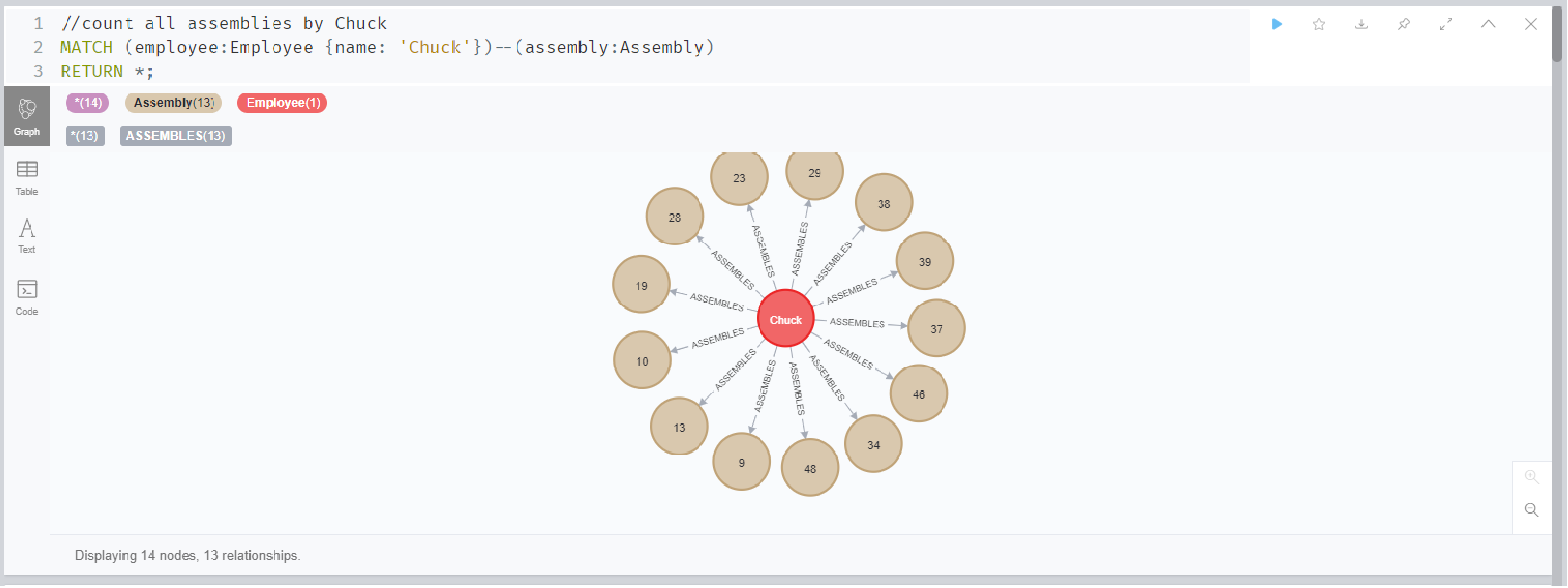
4. Count all devices assembled by Employee Chuck

- We can count the number of assemblies by querying all Assemblies related to the Employee node with name=‘Chuck’.

- If we want to also count the unique number of device classes, we will need to relate the Assembly node to the Device node.

- The question itself was open ended, you could have answered either way!

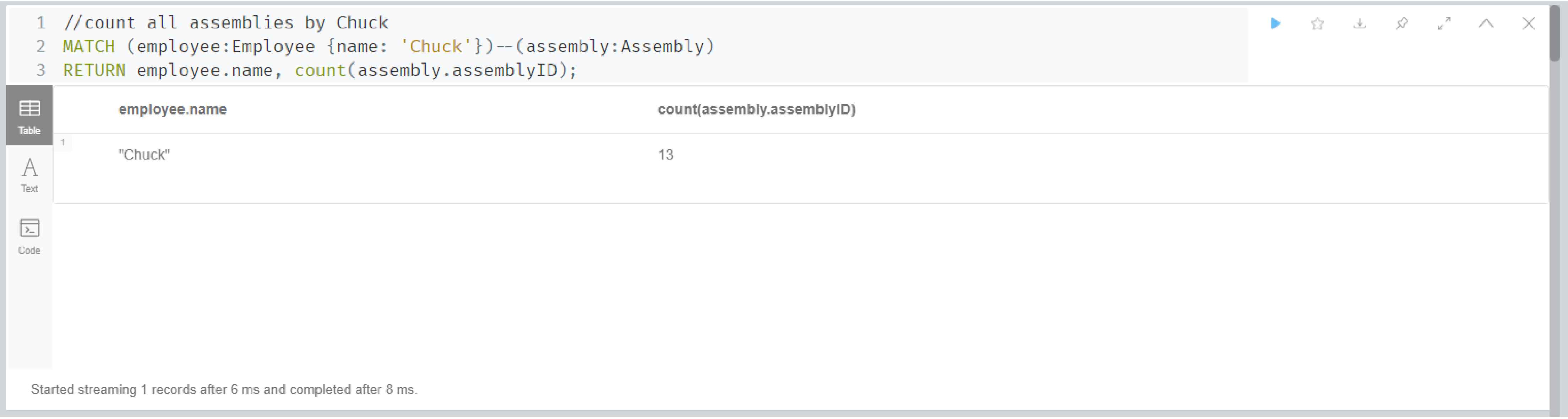
Querying all Assembly nodes related to Chuck returns 13 Assembly nodes



```
//count all assemblies by Chuck
MATCH (employee:Employee {name: 'Chuck'})--(assembly:Assembly)
RETURN *;
```

Displaying 14 nodes, 13 relationships.

Perform the grouping and aggregation operation with the count() function



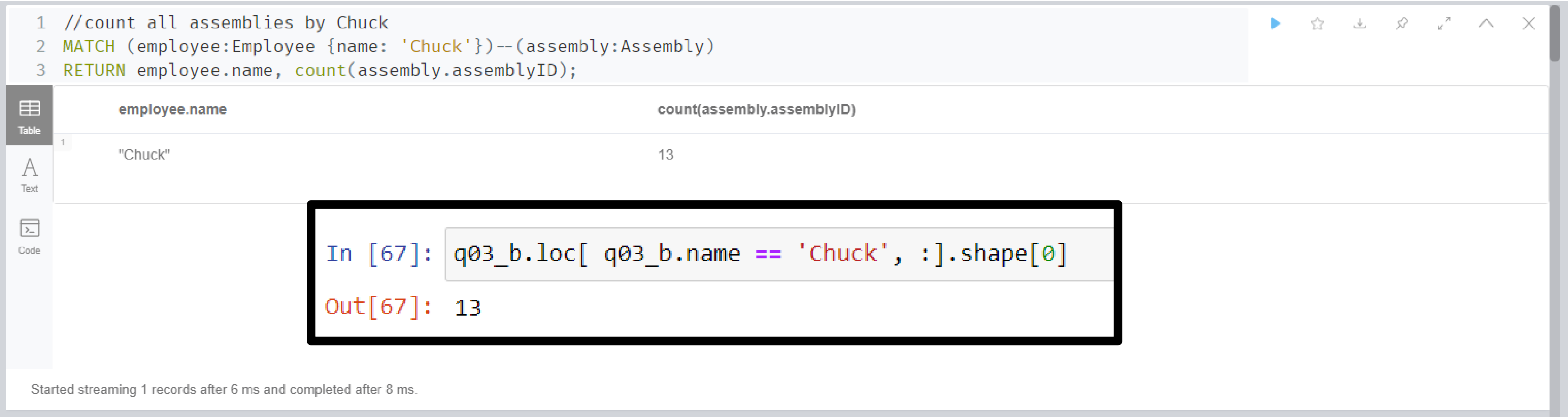
```
//count all assemblies by Chuck
MATCH (employee:Employee {name: 'Chuck'})--(assembly:Assembly)
RETURN employee.name, count(assembly.assemblyID);
```

	employee.name	count(assembly.assemblyID)
1	"Chuck"	13

Started streaming 1 records after 6 ms and completed after 8 ms.

We get the same count we could visually confirm from the previous query.

Confirm by counting the rows in Pandas



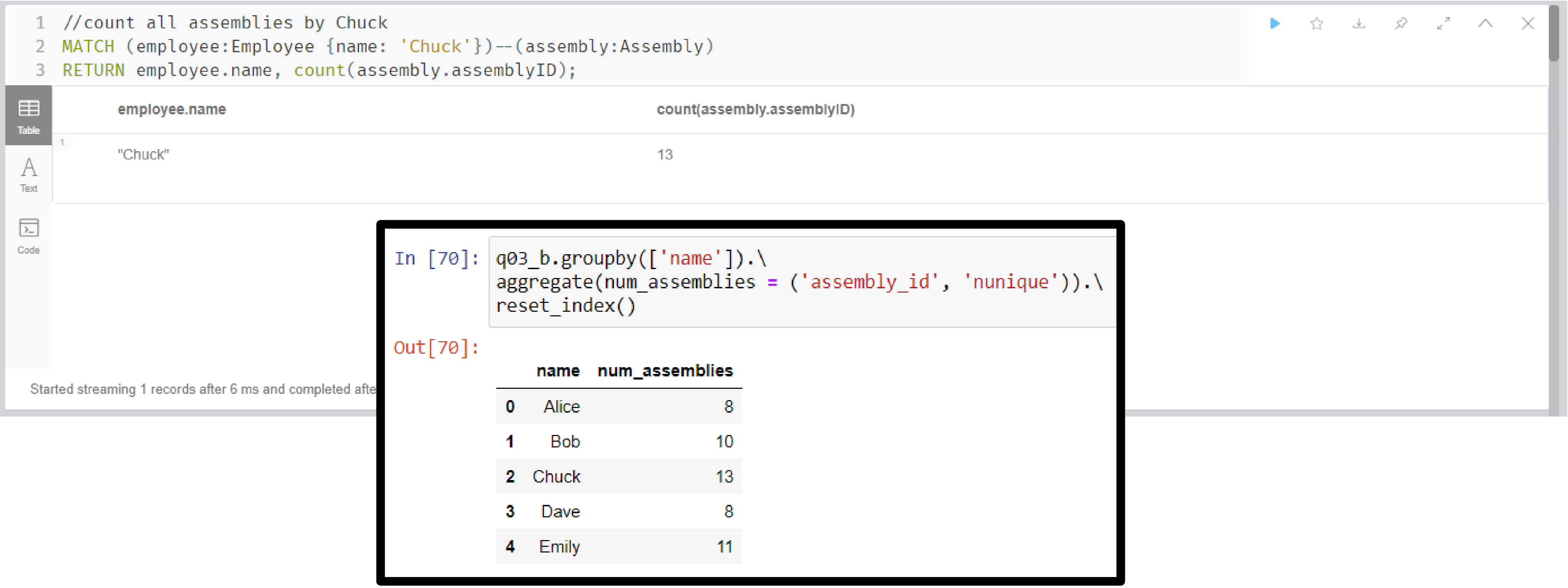
```
//count all assemblies by Chuck
MATCH (employee:Employee {name: 'Chuck'})--(assembly:Assembly)
RETURN employee.name, count(assembly.assemblyID);
```

	employee.name	count(assembly.assemblyID)
1	"Chuck"	13

Started streaming 1 records after 6 ms and completed after 8 ms.

```
In [67]: q03_b.loc[ q03_b.name == 'Chuck', :].shape[0]
Out[67]: 13
```

Or by grouping and aggregating



```
//count all assemblies by Chuck
MATCH (employee:Employee {name: 'Chuck'})--(assembly:Assembly)
RETURN employee.name, count(assembly.assemblyID);
```

	employee.name	count(assembly.assemblyID)
1	"Chuck"	13

Started streaming 1 records after 6 ms and completed afte

```
In [70]: q03_b.groupby(['name']).\
         aggregate(num_assemblies = ('assembly_id', 'nunique')).\
         reset_index()
```

Out[70]:

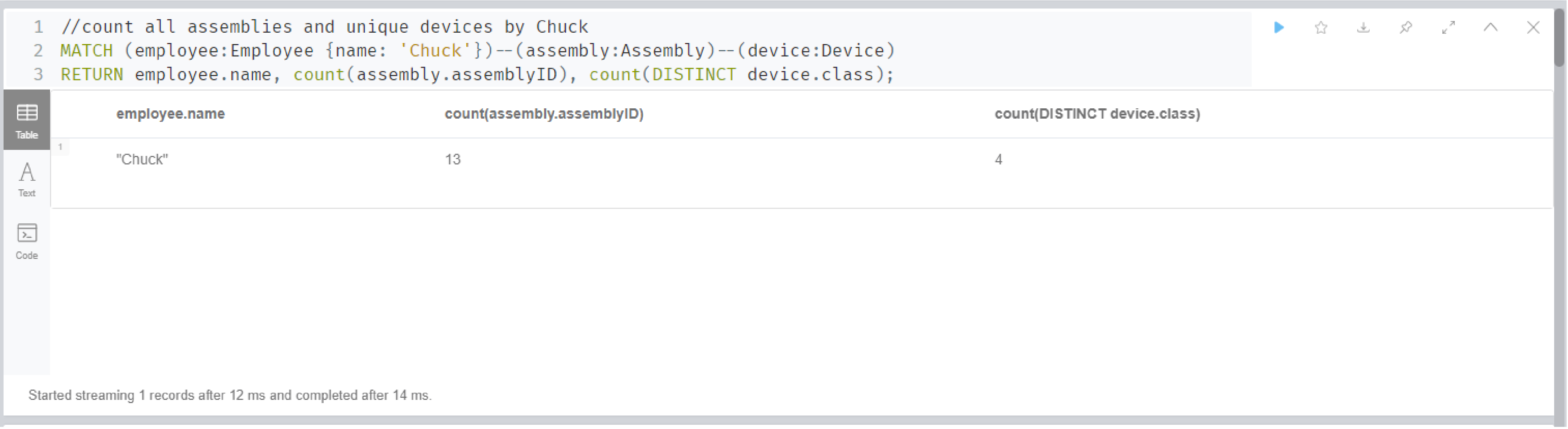
	name	num_assemblies
0	Alice	8
1	Bob	10
2	Chuck	13
3	Dave	8
4	Emily	11

To count the number of unique Device classes assembled by Chuck:

- We need to include the relationship between the Assembly node and the Device node.

- Apply the count() function to the DISTINCT device.class property.

Notice that the pattern includes 3 nodes and 2 relationships now

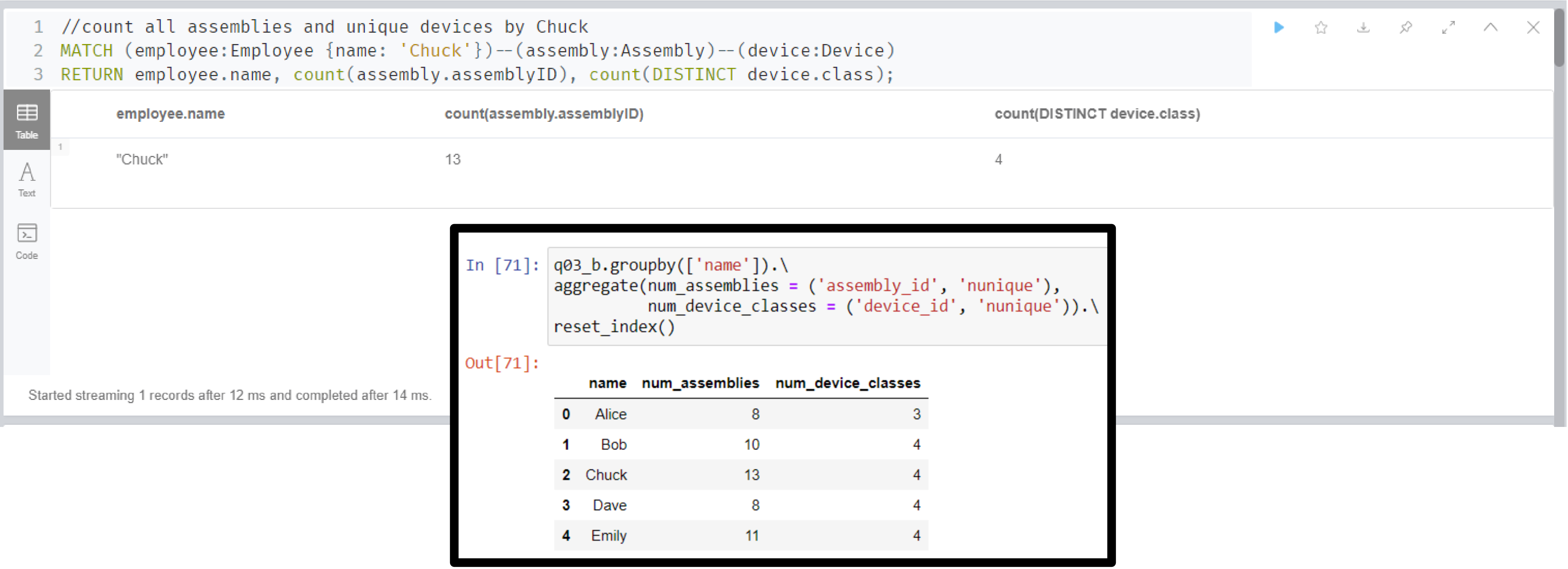


```
//count all assemblies and unique devices by Chuck
MATCH (employee:Employee {name: 'Chuck'})--(assembly:Assembly)--(device:Device)
RETURN employee.name, count(assembly.assemblyID), count(DISTINCT device.class);
```

	employee.name	count(assembly.assemblyID)	count(DISTINCT device.class)
1	"Chuck"	13	4

Started streaming 1 records after 12 ms and completed after 14 ms.

Confirm by grouping and aggregating in Pandas



```
//count all assemblies and unique devices by Chuck
MATCH (employee:Employee {name: 'Chuck'})--(assembly:Assembly)--(device:Device)
RETURN employee.name, count(assembly.assemblyID), count(DISTINCT device.class);
```

	employee.name	count(assembly.assemblyID)	count(DISTINCT device.class)
1	"Chuck"	13	4

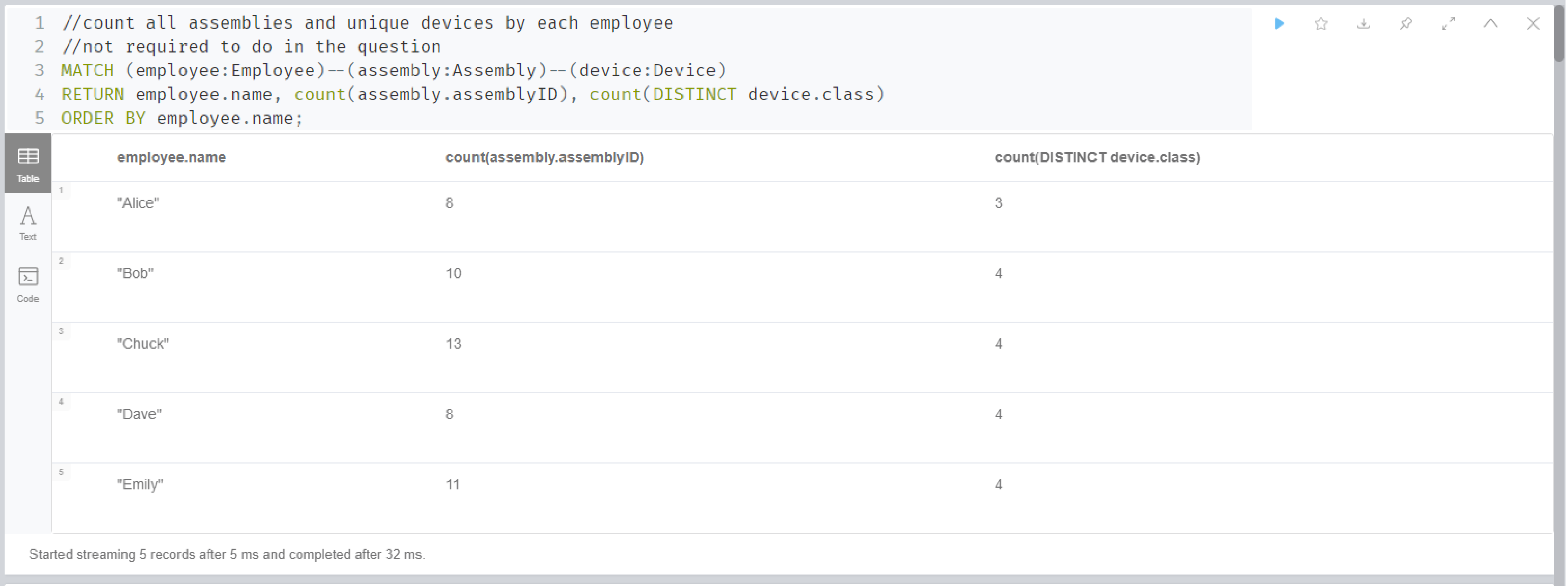
Started streaming 1 records after 12 ms and completed after 14 ms.

```
In [71]: q03_b.groupby(['name']).\
         aggregate(num_assemblies = ('assembly_id', 'nunique'),
                   num_device_classes = ('device_id', 'nunique')).\
         reset_index()
```

Out[71]:

	name	num_assemblies	num_device_classes
0	Alice	8	3
1	Bob	10	4
2	Chuck	13	4
3	Dave	8	4
4	Emily	11	4

Although not required, we could count the number of Assemblies and unique Device classes for every employee in Cypher

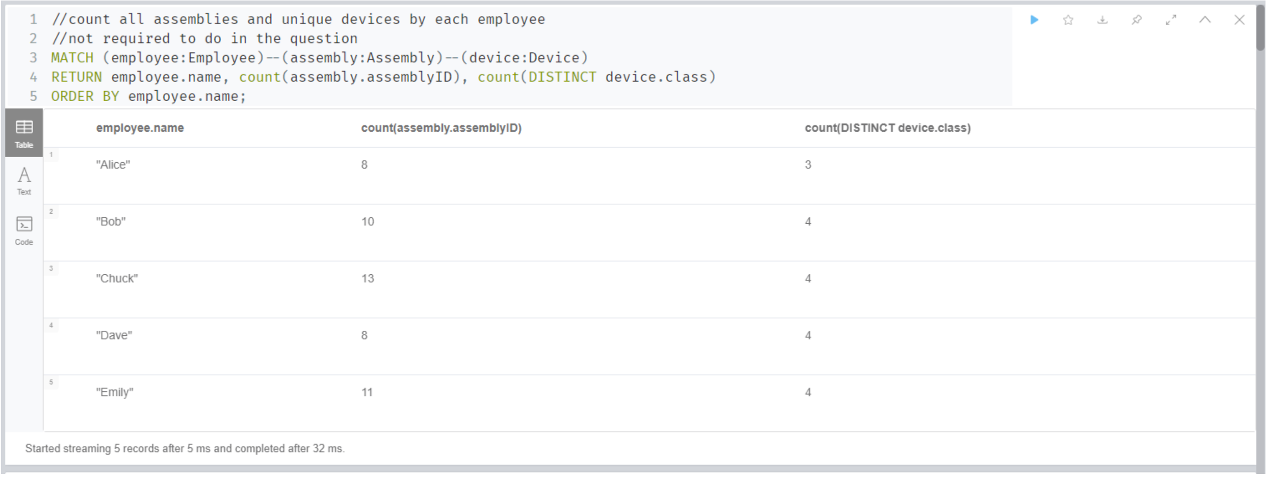


```
//count all assemblies and unique devices by each employee
//not required to do in the question
MATCH (employee:Employee)--(assembly:Assembly)--(device:Device)
RETURN employee.name, count(assembly.assemblyID), count(DISTINCT device.class)
ORDER BY employee.name;
```

employee.name	count(assembly.assemblyID)	count(DISTINCT device.class)
"Alice"	8	3
"Bob"	10	4
"Chuck"	13	4
"Dave"	8	4
"Emily"	11	4

Started streaming 5 records after 5 ms and completed after 32 ms.

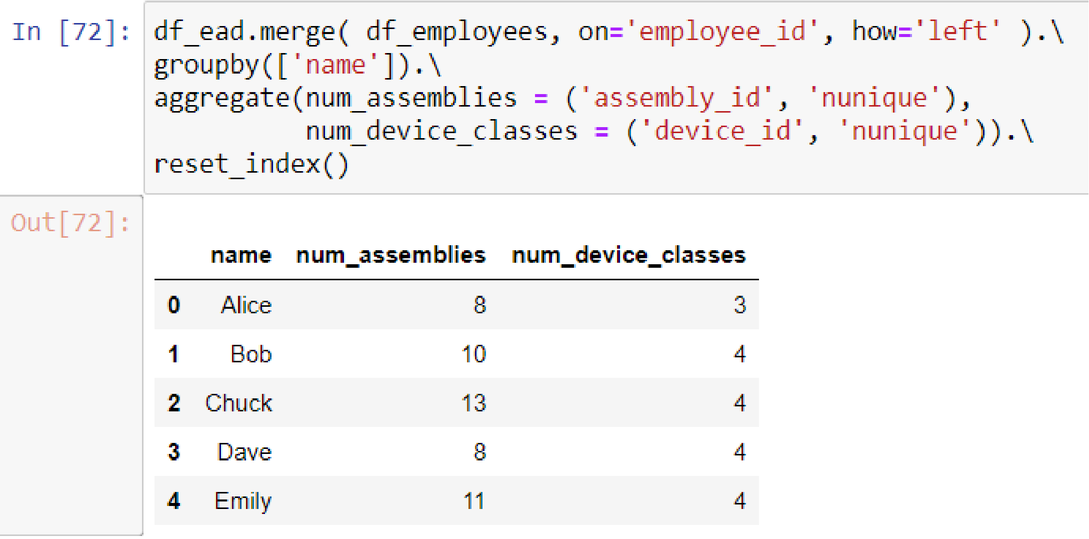
The Cypher query gives the same results as the Pandas merge(), groupby(), and aggregate() series of statements. Analogous to joining, grouping, and summarizing in MySQL.



```
//count all assemblies and unique devices by each employee
//not required to do in the question
MATCH (employee:Employee)--(assembly:Assembly)--(device:Device)
RETURN employee.name, count(assembly.assemblyID), count(DISTINCT device.class)
ORDER BY employee.name;
```

	employee.name	count(assembly.assemblyID)	count(DISTINCT device.class)
1	"Alice"	8	3
2	"Bob"	10	4
3	"Chuck"	13	4
4	"Dave"	8	4
5	"Emily"	11	4

Started streaming 5 records after 5 ms and completed after 32 ms.



```
In [72]: df_ead.merge( df_employees, on='employee_id', how='left' ).\
groupby(['name']).\
aggregate(num_assemblies = ('assembly_id', 'nunique'),
          num_device_classes = ('device_id', 'nunique')).\
reset_index()
```

Out[72]:

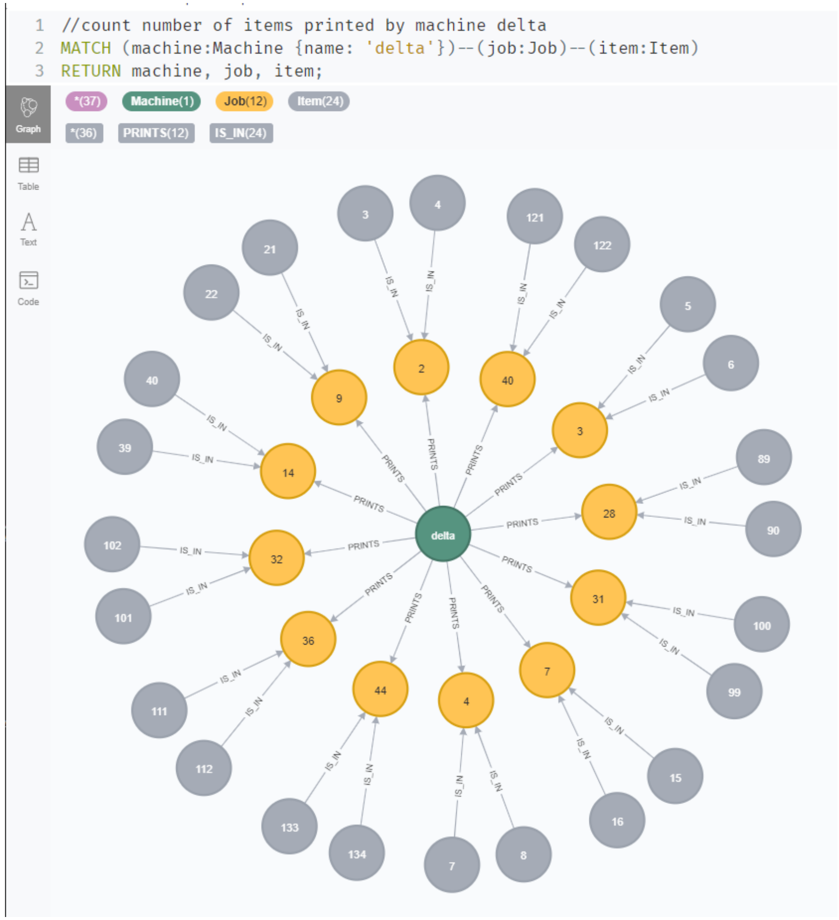
	name	num_assemblies	num_device_classes
0	Alice	8	3
1	Bob	10	4
2	Chuck	13	4
3	Dave	8	4
4	Emily	11	4

5. Count all parts printed by the Machine delta.

- Question is open ended, so could count the unique number of Items printed OR the unique number of Part types.

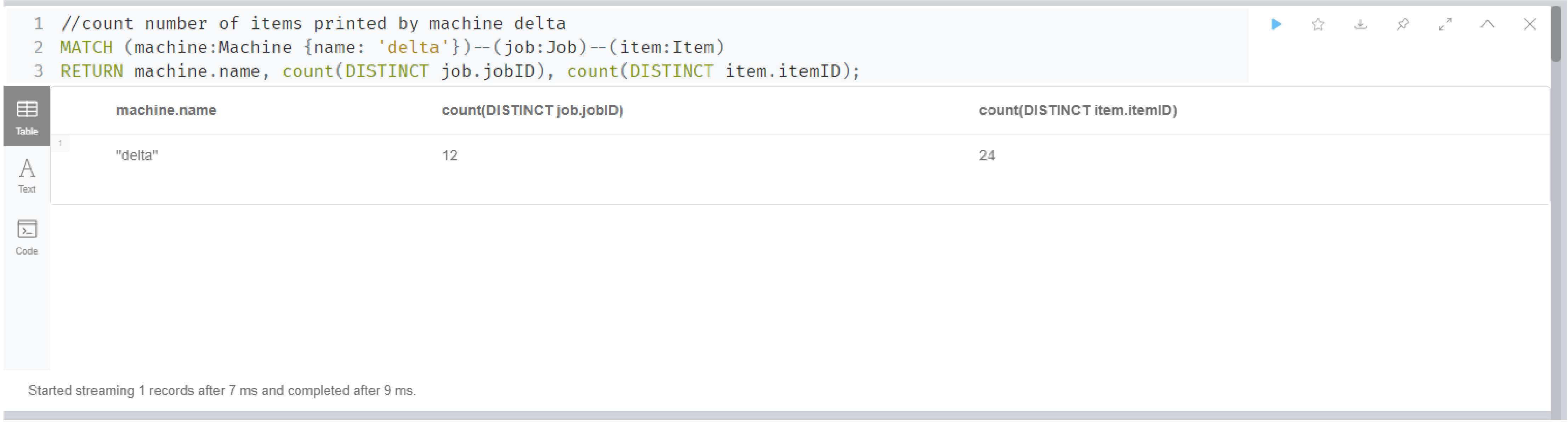
Although not required, the network for all Jobs printed by Machine delta and all items in those Jobs

The graph tells us there are 12 Jobs and 24 Items.



```
//count number of items printed by machine delta
MATCH (machine:Machine {name: 'delta'})--(job:Job)--(item:Item)
RETURN machine, job, item;
```

Query the number of Items by grouping and summarizing via the count() function



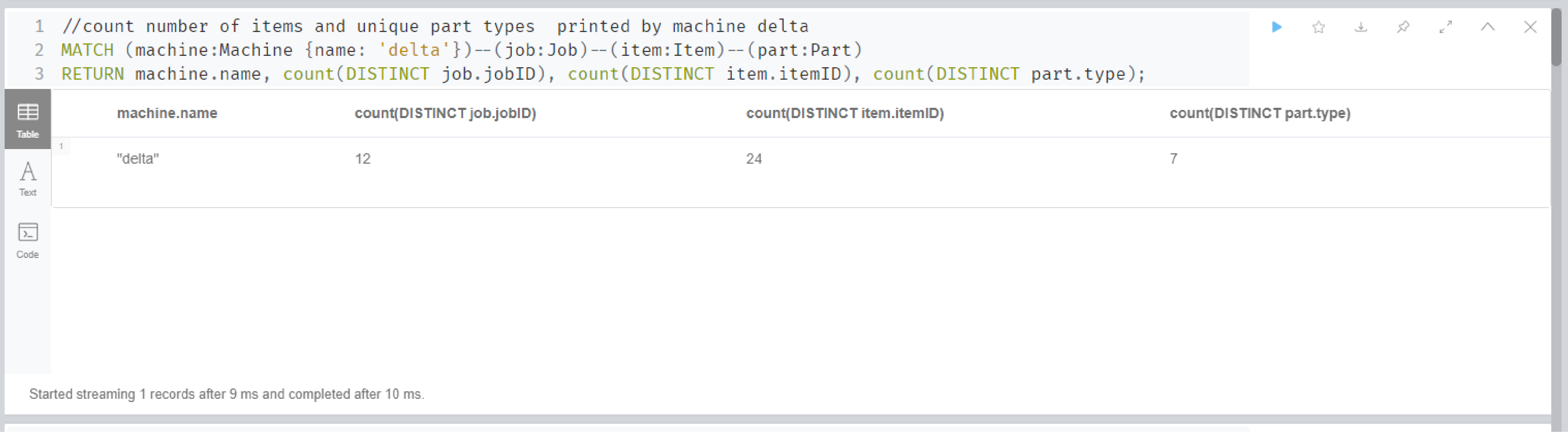
```
//count number of items printed by machine delta
MATCH (machine:Machine {name: 'delta'})--(job:Job)--(item:Item)
RETURN machine.name, count(DISTINCT job.jobID), count(DISTINCT item.itemID);
```

machine.name	count(DISTINCT job.jobID)	count(DISTINCT item.itemID)
"delta"	12	24

Started streaming 1 records after 7 ms and completed after 9 ms.

The number of unique Jobs is included, but was not required for the question.

If we also want to query the number of unique Part types printed by Machine delta, we need to include the Item to Part relationship and apply the count() function appropriately

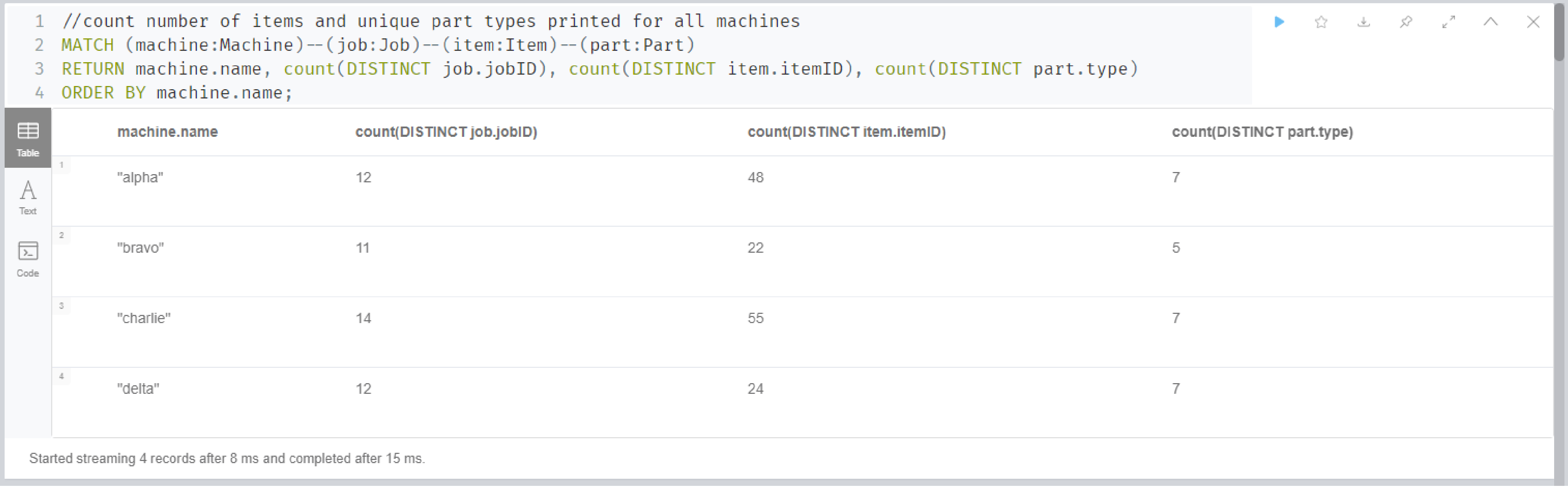


```
//count number of items and unique part types  printed by machine delta
MATCH (machine:Machine {name: 'delta'})--(job:Job)--(item:Item)--(part:Part)
RETURN machine.name, count(DISTINCT job.jobID), count(DISTINCT item.itemID), count(DISTINCT part.type);
```

machine.name	count(DISTINCT job.jobID)	count(DISTINCT item.itemID)	count(DISTINCT part.type)
"delta"	12	24	7

Started streaming 1 records after 9 ms and completed after 10 ms.

Although not required, it’s straight forward to repeat the query for all machines not just Machine delta

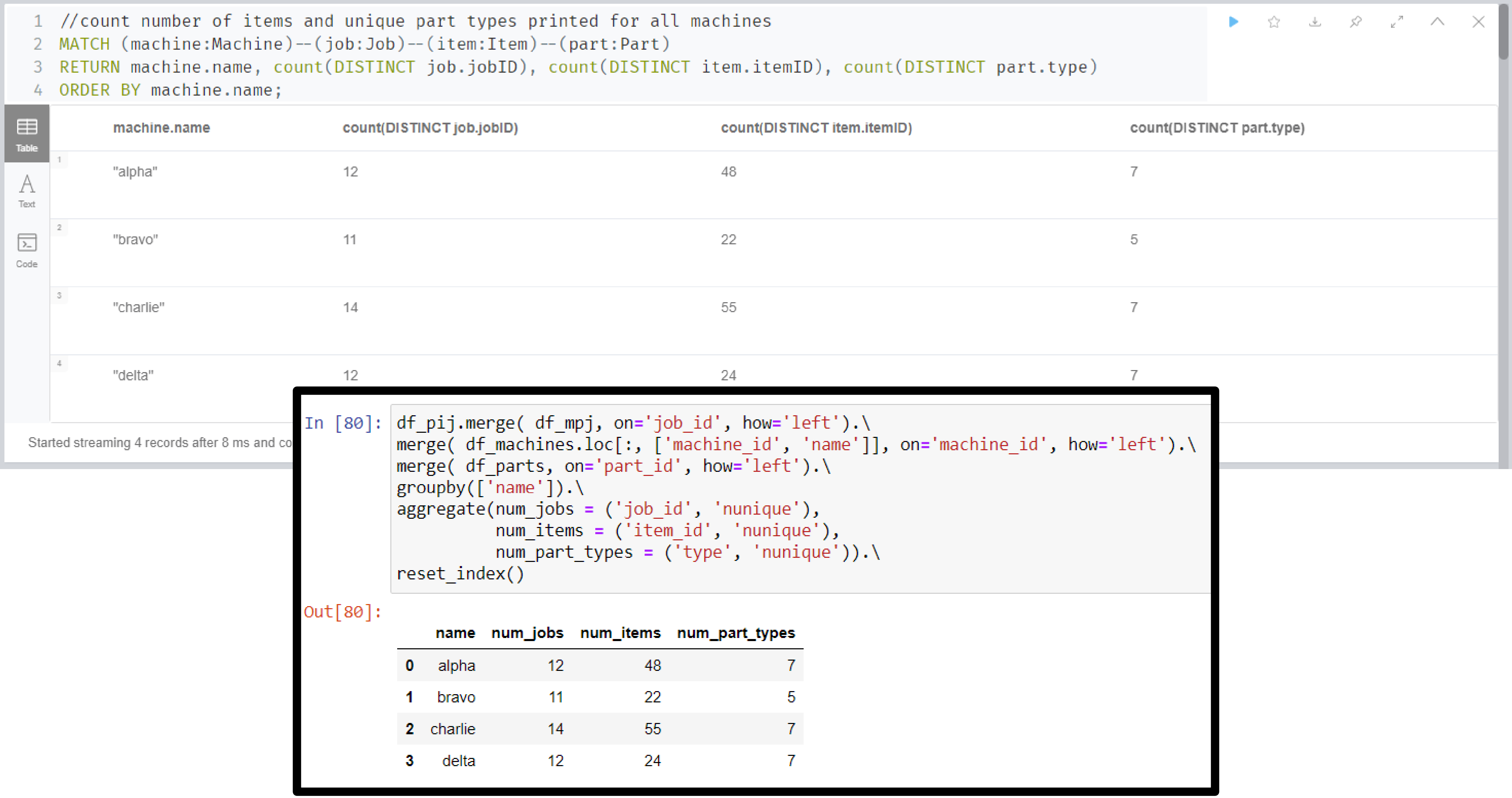


```
//count number of items and unique part types printed for all machines
MATCH (machine:Machine)--(job:Job)--(item:Item)--(part:Part)
RETURN machine.name, count(DISTINCT job.jobID), count(DISTINCT item.itemID), count(DISTINCT part.type)
ORDER BY machine.name;
```

	machine.name	count(DISTINCT job.jobID)	count(DISTINCT item.itemID)	count(DISTINCT part.type)
1	"alpha"	12	48	7
2	"bravo"	11	22	5
3	"charlie"	14	55	7
4	"delta"	12	24	7

Started streaming 4 records after 8 ms and completed after 15 ms.

Check the results with Pandas



```
//count number of items and unique part types printed for all machines
MATCH (machine:Machine)--(job:Job)--(item:Item)--(part:Part)
RETURN machine.name, count(DISTINCT job.jobID), count(DISTINCT item.itemID), count(DISTINCT part.type)
ORDER BY machine.name;
```

	machine.name	count(DISTINCT job.jobID)	count(DISTINCT item.itemID)	count(DISTINCT part.type)
1	"alpha"	12	48	7
2	"bravo"	11	22	5
3	"charlie"	14	55	7
4	"delta"	12	24	7

Started streaming 4 records after 8 ms and co

```
In [80]: df_pij.merge( df_mpj, on='job_id', how='left').\
         merge( df_machines.loc[:, ['machine_id', 'name']], on='machine_id', how='left').\
         merge( df_parts, on='part_id', how='left').\
         groupby(['name']).\
         aggregate(num_jobs = ('job_id', 'nunique'),
                   num_items = ('item_id', 'nunique'),
                   num_part_types = ('type', 'nunique')).\
         reset_index()
```

Out[80]:

	name	num_jobs	num_items	num_part_types
0	alpha	12	48	7
1	bravo	11	22	5
2	charlie	14	55	7
3	delta	12	24	7