

# Star-Wars® Hands-On Workshop Lab Exercises

Version 0.8

Mark Quinsland  
Senior Field Engineer  
February 8, 2023



# Notes on Hands-On Exercises:

The full text of the exercises and the data files are all available from the Github repo:

[https://github.com/mquinz/star\\_wars\\_demo](https://github.com/mquinz/star_wars_demo)

The exercises use the pre-release Data Importer tool that is currently available only on Neo4j Aura.  
It is assumed that you have a login & password for Neo4j Aura.

If you are unable to create an Aura account, please let me know and I will set you up with ‘Plan B’

We will also be using the browser-based Neo4j modeling tool called Arrows and is available at

<https://arrows.app/>

The Arrows app is a simple tool takes only a few minutes to learn but is invaluable for designing Neo4j data models.

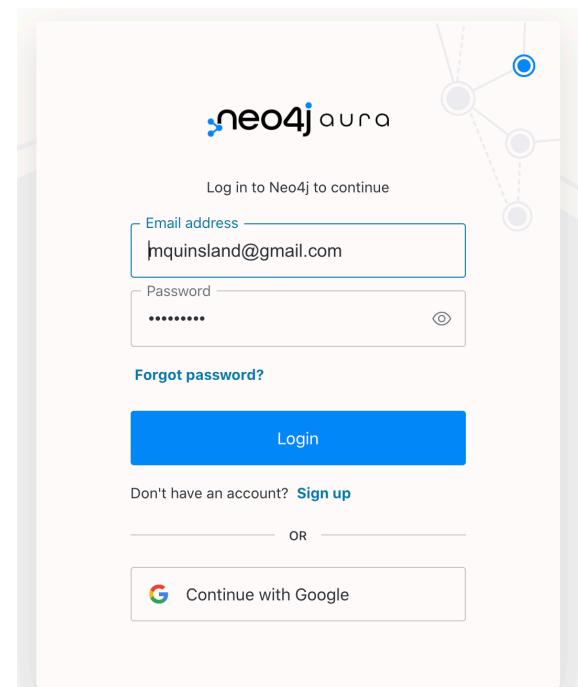
# Lab 0 - Connecting to Aura



# Lab 0 - Creating a New Database in Aura

Login - or create a free account

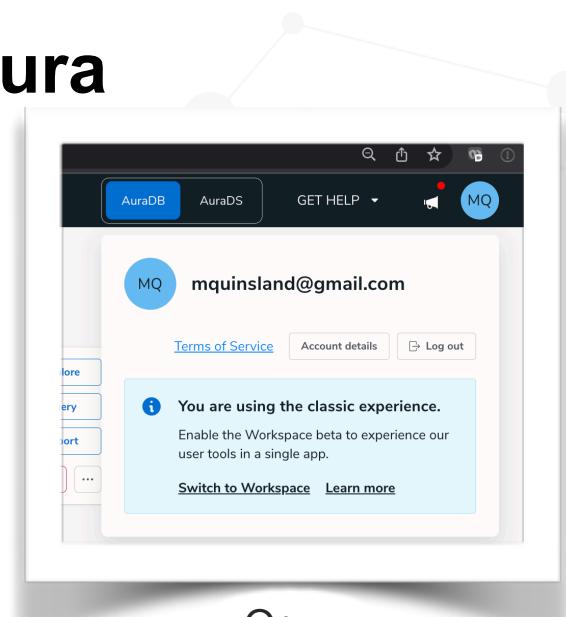
- Proceed to <https://login.neo4j.com>
- Enter username and password
- Create a new account if necessary



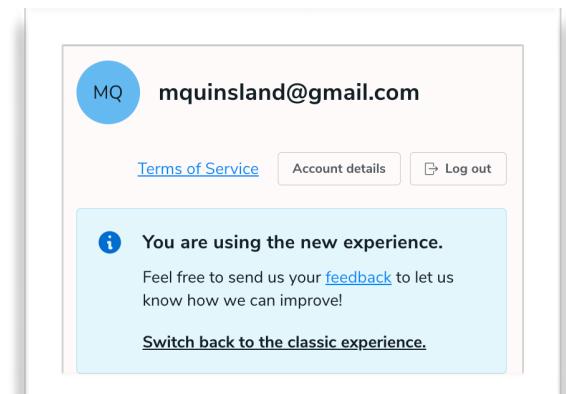
# Lab 0.1 - Creating a New Database in Aura

## Switch to New Workspace Experience

- After logging in, click on your initials in the menu bar
- You will see one of these messages to indicate whether you are in the classic version, or the new Workspace version.
- We will be using the new Workspace version. (it's in beta testing so things may get rough!)



Or



# Lab 0.2 - Creating a New Database in Aura

## Create a new Instance

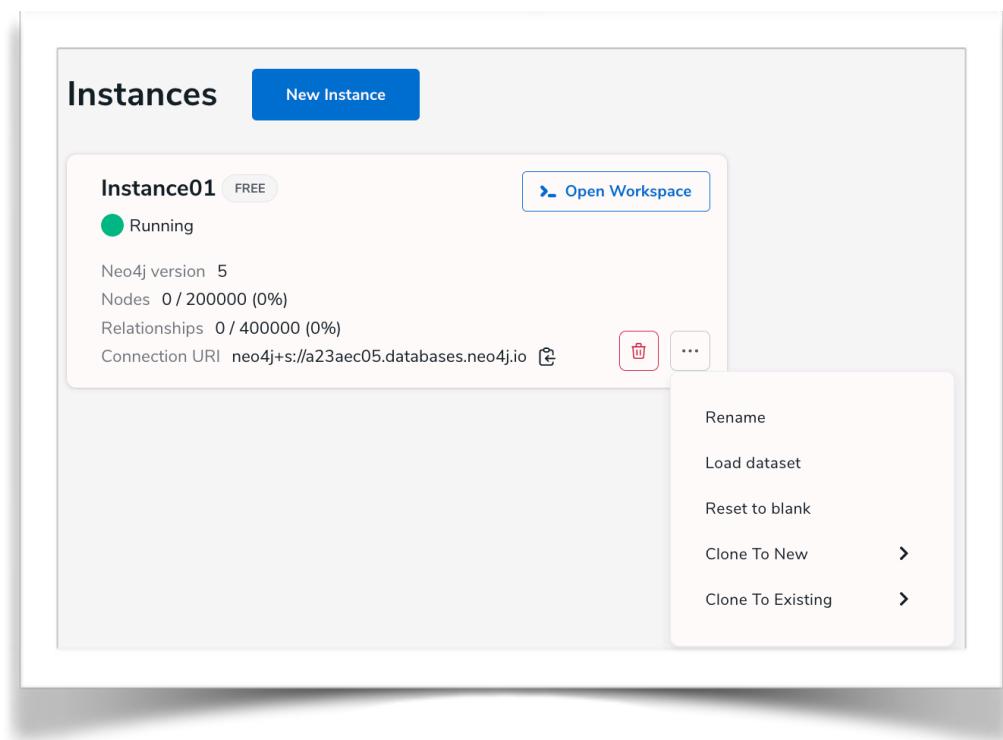
- Click on the New Instance Button
- Select the Empty instance
- Your new instance will be named Instance01 and a password will be generated. Copy the password to the clipboard and download the password file
- Please remember where you downloaded the password file!

The screenshot shows two overlapping interface panels. The top panel is titled 'Instances' and features a prominent blue 'New Instance' button. The bottom panel is titled 'Credentials for Instance01' and contains fields for 'Username' (set to 'neo4j') and 'Generated password' (containing a long string of characters). It includes a 'Download' button next to the password field. A note below advises changing the password. At the bottom, a checkbox is checked, stating: 'I confirm I have copied or downloaded the above credentials, as this password will not be available after this point'. A 'Continue' button is located at the very bottom right.

# Lab 0.3 - Creating a New Database in Aura

## Rename the new Instance

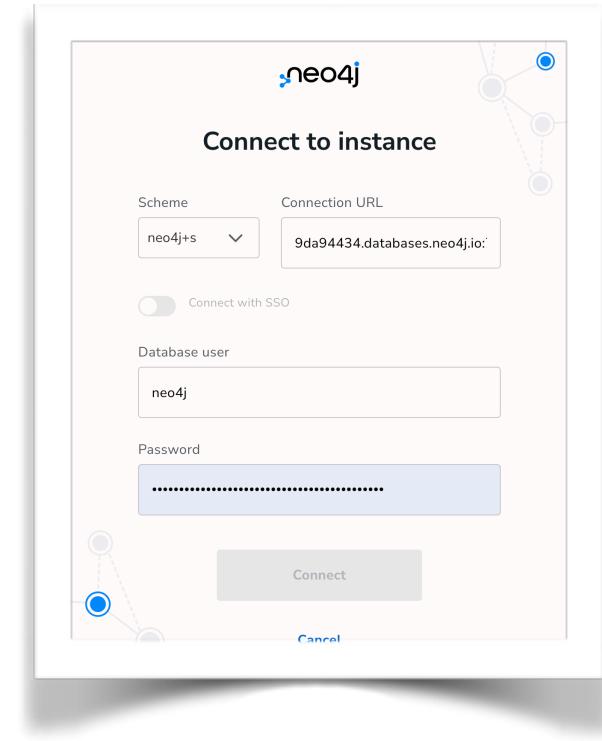
- In a moment or two, the instance will be created and the status will change to ‘Running’
- Change the name to something useful by clicking on the ... button next to the trash can.
- Some basic statistics will be shown
- The instance will be free to use for its lifetime. After periods of inactivity it will be suspended, but resuming it requires only a simple click.



# Lab 0.4 - Creating a New Database in Aura

## Testing the Connection

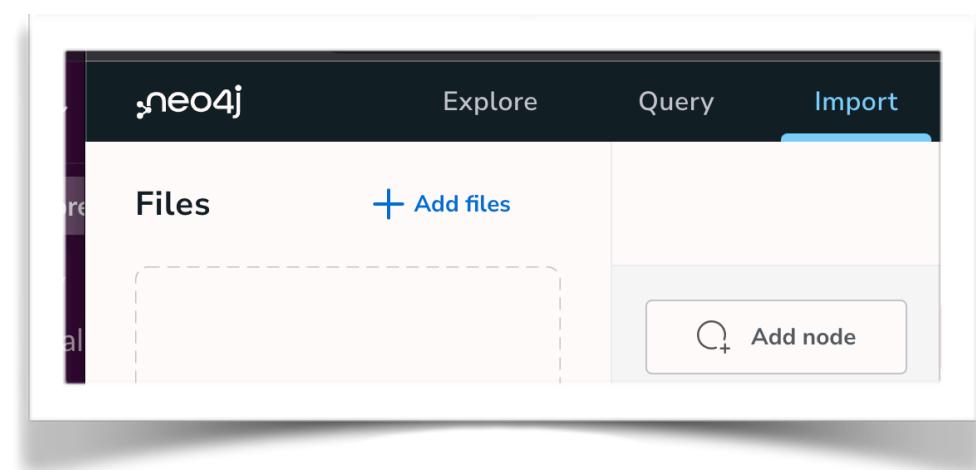
- Click on the Open Workspace button to display the Connect to Instance dialog.
- Paste your password into the appropriate field.
- If you forgot your password and failed to download it, you must delete the instance and start over.



# Lab 0.5 - Creating a New Database in Aura

## Viewing the Workspace

- The Aura Developer Workspace has 3 main tabs.
- Explore: Uses Neo4j Bloom to analyze and visualize the graph without writing code
- Query: Write and execute Cypher queries to create, read, update, and delete data.
- Import: Use the Neo4j Import tool to load CSV files without writing code
- We will start on the Query tab

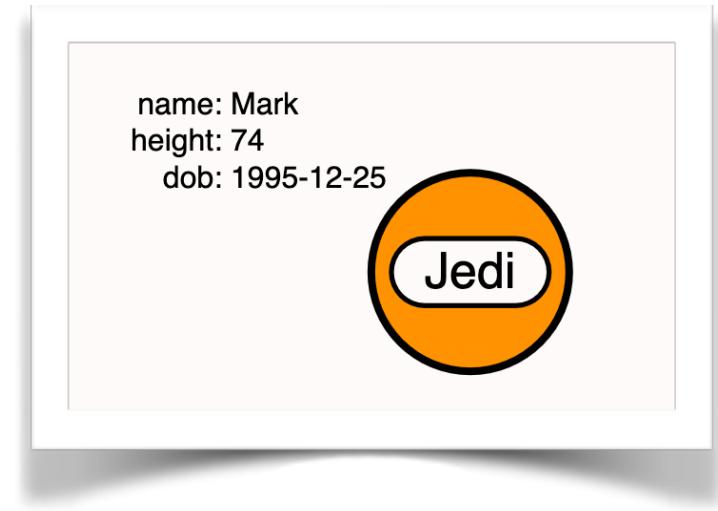


# Lab 1 - Data Modeling with Arrows

# Lab 1.1 - Data Modeling using Arrows

## Creating A New Node

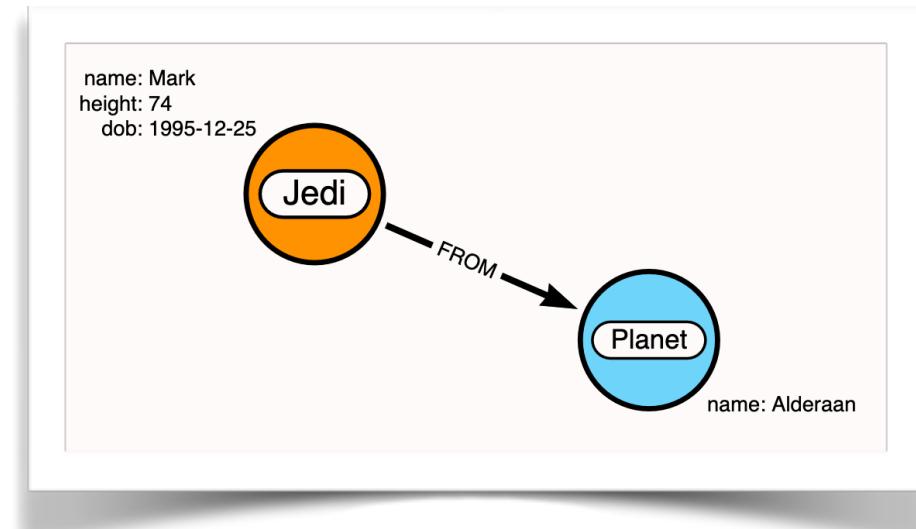
- Bring up <https://arrows.app> on a browser
- Tap on the empty node to highlight it
- On the properties tab, give the new node a label (not a caption)
- Give the node a property:value pair (name: your name)
- Give the node 2-3 additional properties
- Give the node a color



## Lab 1.2 - Data Modeling using Arrows

### Creating A Related Node

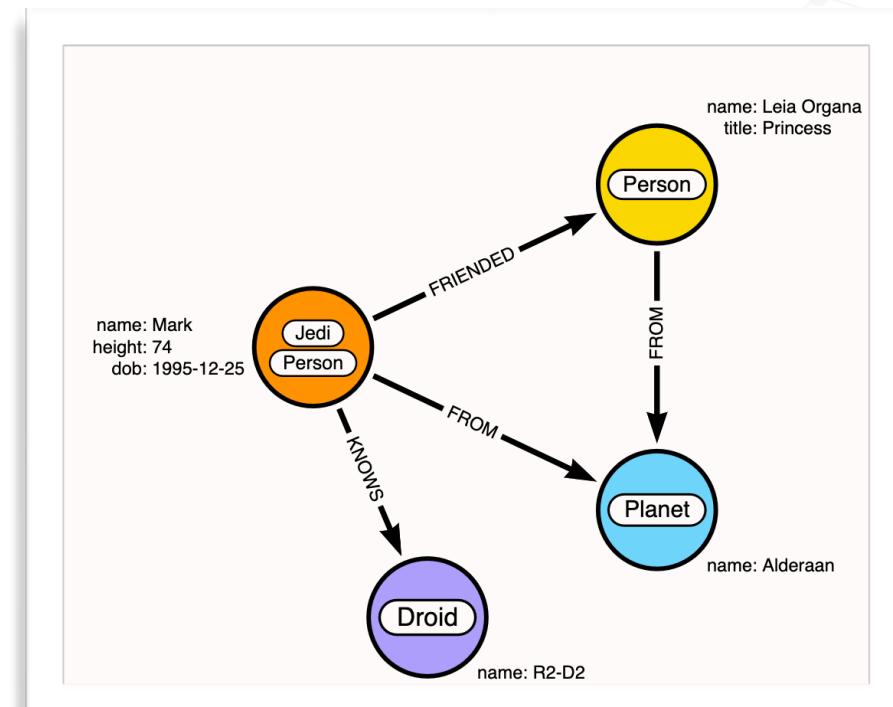
1. Hover over the border of the source node until the border shows that it is selected
2. Click on the selection border and drag it away from the node. You will see a new node appear.
3. Drop the new node away from other nodes
4. On the properties tab, give the new node a label (not a caption)
5. Give the node a property:value pair (name: your name)
6. Select the relationship and give it a type.



# Lab 1 - Data Modeling using Arrows

## Creating Related Nodes

1. Hover over the border of the source node until the border shows that it is selected
2. Click on the selection border and drag it away from the node. You will see a new node appear.
3. Drop the new node away from other nodes
4. On the properties tab, give the new node a label (not a caption)
5. Give the node a property:value pair (name: your name)
6. Select the relationship and give it a type.
7. Repeat for 2-3 more nodes



# Lab 1 - Data Modeling using Arrows

## Exporting a Model

1. Give the model a name
2. Click on the Download/Export button
3. Select the JSON tab
4. Click on the Download button
5. Select a safe place to store the JSON file.

The screenshot shows the 'Export' dialog box from the Neo4j interface. The 'JSON' tab is selected among other options like PNG, SVG, Cypher, and UR. Below the tabs is a 'Download' button with a download icon. The main area displays a portion of a JSON configuration object:

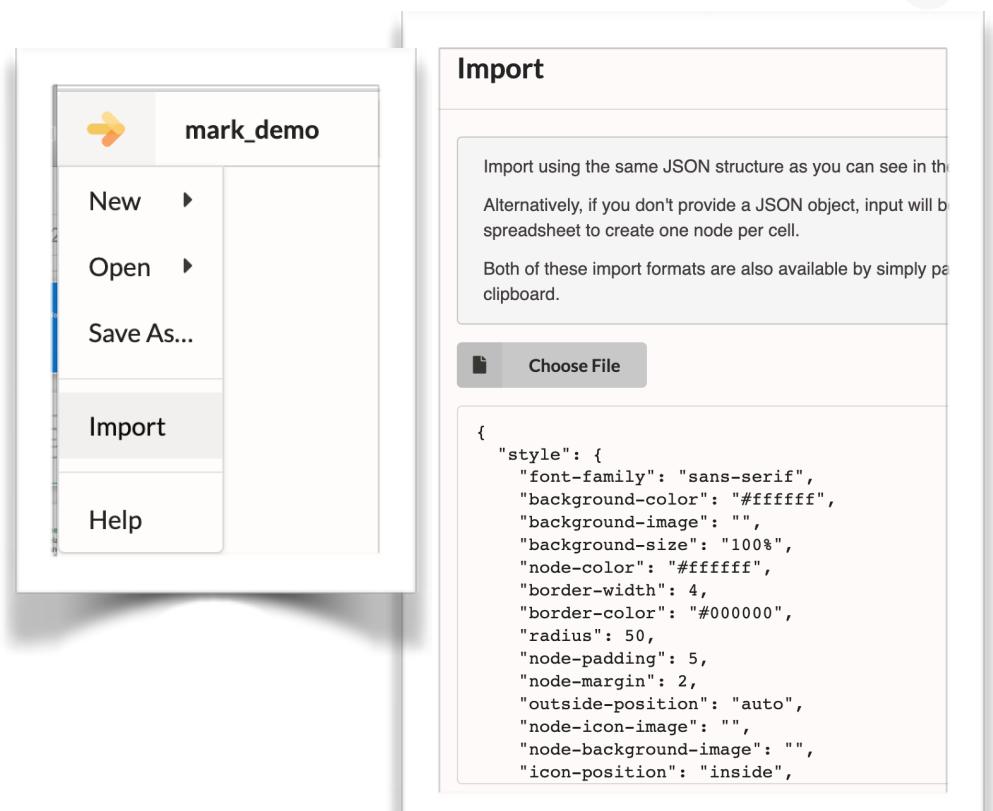
```
{  
  "style": {  
    "font-family": "sans-serif",  
    "background-color": "#ffffff",  
    "background-image": "",  
    "background-size": "100%",  
    "node-color": "#ffffff",  
    "border-width": 4,  
    "border-color": "#000000",  
    "radius": 50,  
    "node-padding": 5,  
    "node-margin": 2,  
    "outside-position": "auto",  
    "node-icon-image": "",  
    "node-background-image": "",  
    "icon-position": "inside"  
  }  
}
```

At the bottom of the dialog, there are file selection buttons for 'mark\_sw\_demo\_.json' and 'Neo4j C'.

# Lab 1 - Data Modeling using Arrows

Importing a Model that was previously exported

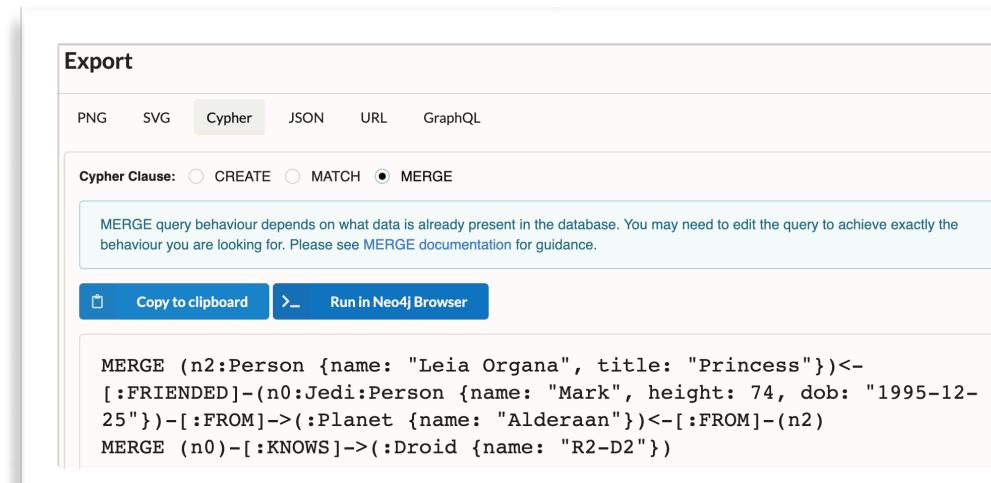
1. Click on the yellow arrow at the top left corner
2. Select the Import item
3. Choose the JSON file to be uploaded.
4. View the new nodes / relationships.



# Lab 1 - Data Modeling using Arrows

## Testing the Model

1. Click on the Download/Export button
2. Select the Cypher tab
3. Select the Create option
4. Review the Cypher code
5. Copy the code to clipboard
6. Paste it into a browser
7. Execute the Cypher to create the nodes
8. View the newly created data in the browser



The screenshot shows the Neo4j Export interface. At the top, there are tabs for PNG, SVG, Cypher, JSON, URL, and GraphQL. The Cypher tab is selected. Below the tabs, there is a section for 'Cypher Clause' with radio buttons for CREATE, MATCH, and MERGE, where MERGE is selected. A note below states: 'MERGE query behaviour depends on what data is already present in the database. You may need to edit the query to achieve exactly the behaviour you are looking for. Please see MERGE documentation for guidance.' At the bottom, there are two buttons: 'Copy to clipboard' (highlighted in blue) and 'Run in Neo4j Browser'.

```
MERGE (n2:Person {name: "Leia Organa", title: "Princess"})<-[ :FRIENDED ]-(n0:Jedi:Person {name: "Mark", height: 74, dob: "1995-12-25"})-[:FROM]->(:Planet {name: "Alderaan"})<-[:FROM]-(n2)
MERGE (n0)-[:KNOWS]->(:Droid {name: "R2-D2"})
```

# Lab 1 - Data Modeling using Arrows

## Viewing the Data in Neo4j Browser

1. Copy the Cypher text that was generated by the Export window
2. Switch to your browser and bring up the Aura Developer Workbench tab
3. Paste the code into the execution window
4. Click on the small blue arrow to execute the command

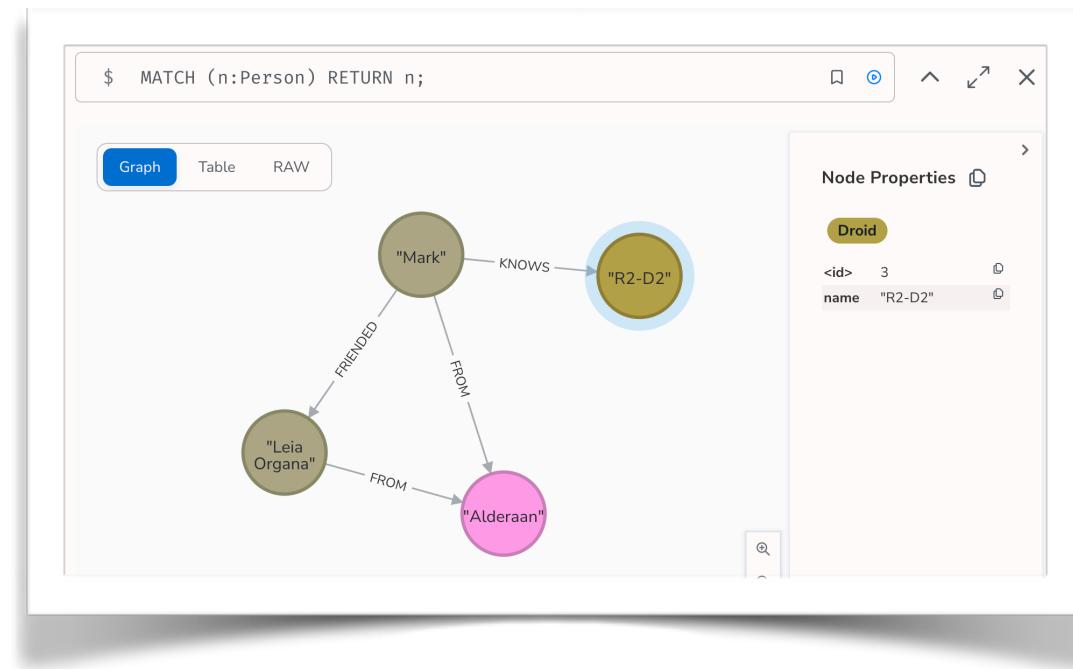
```
1 CREATE (n6:Person {name: "Leia Organa", title:  
    "Princess"})←[:FRIENDED]-(n4:Jedi:Person {name:  
    "Mark", height: 74, dob: "1995-12-25"})-[:FROM]→  
    (:Planet {name: "Alderaan"})←[:FROM]-(n6),  
2 (n4)-[:KNOWS]→(:Droid {name: "R2-D2"})
```

# Lab 1 - Data Modeling using Arrows

## Viewing the Data in Neo4j Browser

1. Run a simple Cypher command to view your data
2. Double-click on the nodes to see related nodes appear.

```
MATCH (n:Person) RETURN n;
```

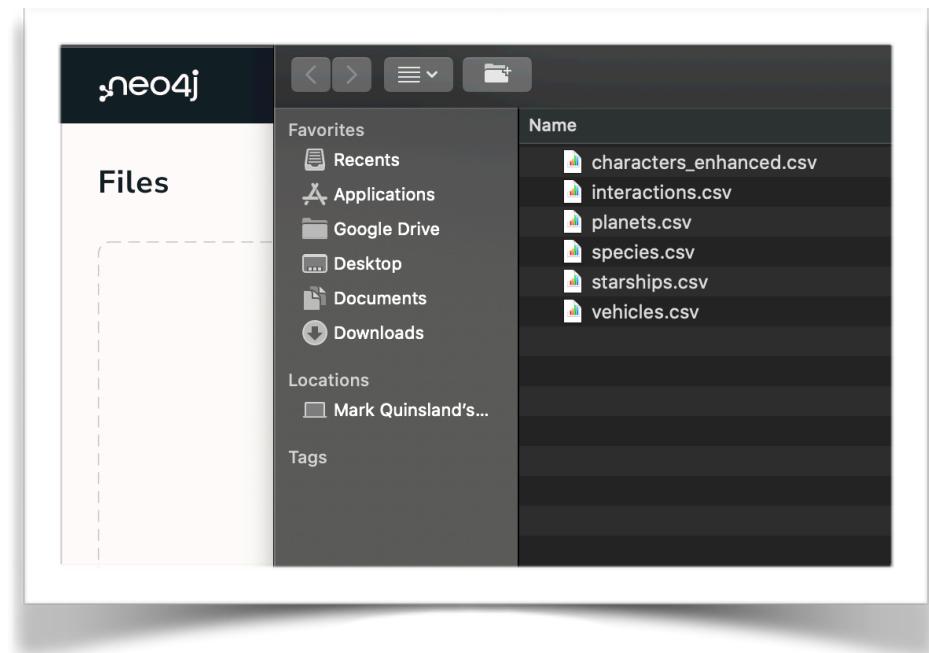


# Lab 2 - Using Workspace Importer

# Lab 2.1 - Using Workspace Importer

## Selecting CSV files to Import

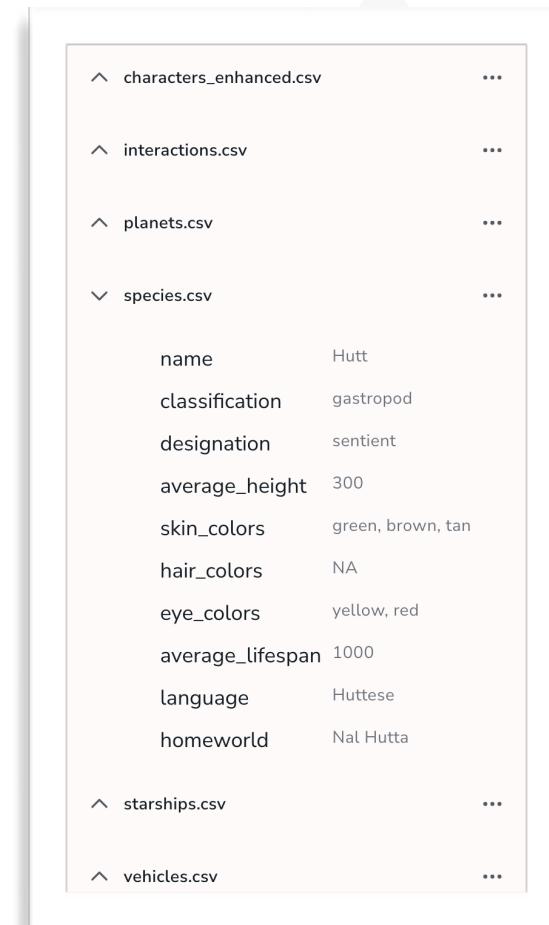
- Download the data files from
- [https://github.com/mquinz/star\\_wars\\_demo/blob/master/data/star\\_wars\\_data.zip](https://github.com/mquinz/star_wars_demo/blob/master/data/star_wars_data.zip)
- Unzip the files
- Click on the + Add files link on the Workspace Import tab or drag-and-drop the files onto the files tab.



## Lab 2.2 - Using Workspace Importer

### Previewing Data From CSV files

- After adding the 6 CSV files, expand/collapse the file names to preview the data in the files
- The importer will attempt to determine data types for the values, but you can override/correct this in the next step.



The screenshot shows a file tree interface for a workspace. The files listed are characters\_enhanced.csv, interactions.csv, planets.csv, species.csv, starships.csv, and vehicles.csv. The species.csv file is expanded, showing its contents:

name	Hutt
classification	gastropod
designation	sentient
average_height	300
skin_colors	green, brown, tan
hair_colors	NA
eye_colors	yellow, red
average_lifespan	1000
language	Huttese
homeworld	Nal Hutta

# Lab 2.3 - Using Workspace Importer

## Adding Nodes

- Click on the Add Node Button and an empty node type will appear.
- In the Mapping Details tab, enter the name of a Label e.g. Character
- Select a CSV file that contains the data to be loaded for that Label
- Click on the ‘Select from file’ link and choose the property names to be loaded.
- Edit data types or field names
- In the ID field, choose which property value should be used as the primary key of each node

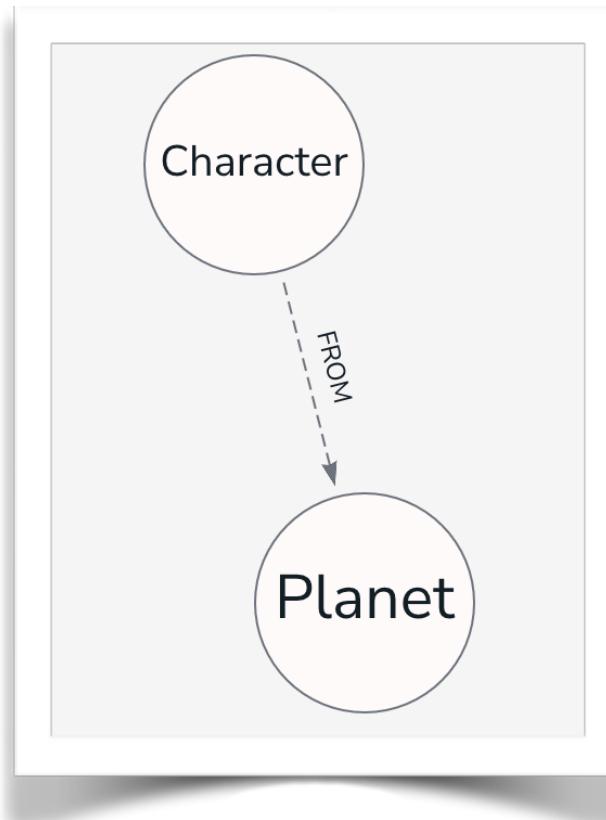
The screenshot shows the Neo4j Workspace Importer interface. On the left, there's a sidebar with 'Files' and a '+ Add files' button. Below it are three expanded CSV files: 'characters\_enhanced.csv', 'interactions.csv', and 'planets.csv'. The 'characters\_enhanced.csv' file is selected. In the center, there's a 'Select from file' dialog box with a 'Select all' checkbox. It lists properties from the file with their corresponding values for the node 'Yoda'. On the right, the 'Mapping Details' tab is active. It includes sections for 'Label' (set to 'Character'), 'File' (set to 'characters\_enhanced.csv'), 'Properties' (with a 'Select from file' button), and 'Mapping' (a table of properties and their types). The 'neo4j' logo is visible in the bottom right corner.

Property	Type	Editor	Delete
name	string		
gender	string		
birthYear	string		
hairColor	string		
eyecolor	string		
height	string		
homePlanet	string		
weight	string		
socialName	string		
species	string		
pagerank	string		
betweenness	string		
closeness	string		
louvain	string		

## Lab 2.4 - Using Workspace Importer

### Creating Relationships Between Nodes

- Select a Node type that will serve as the 'From' node
- Hover over its border until the border color changes and a + indicator appears.
- Drag the + indicator to the node that will be the 'To node.'
- Double-Click on the relationship line to select it and type in the relationship type
- The line will appear broken until the relationship info is completed

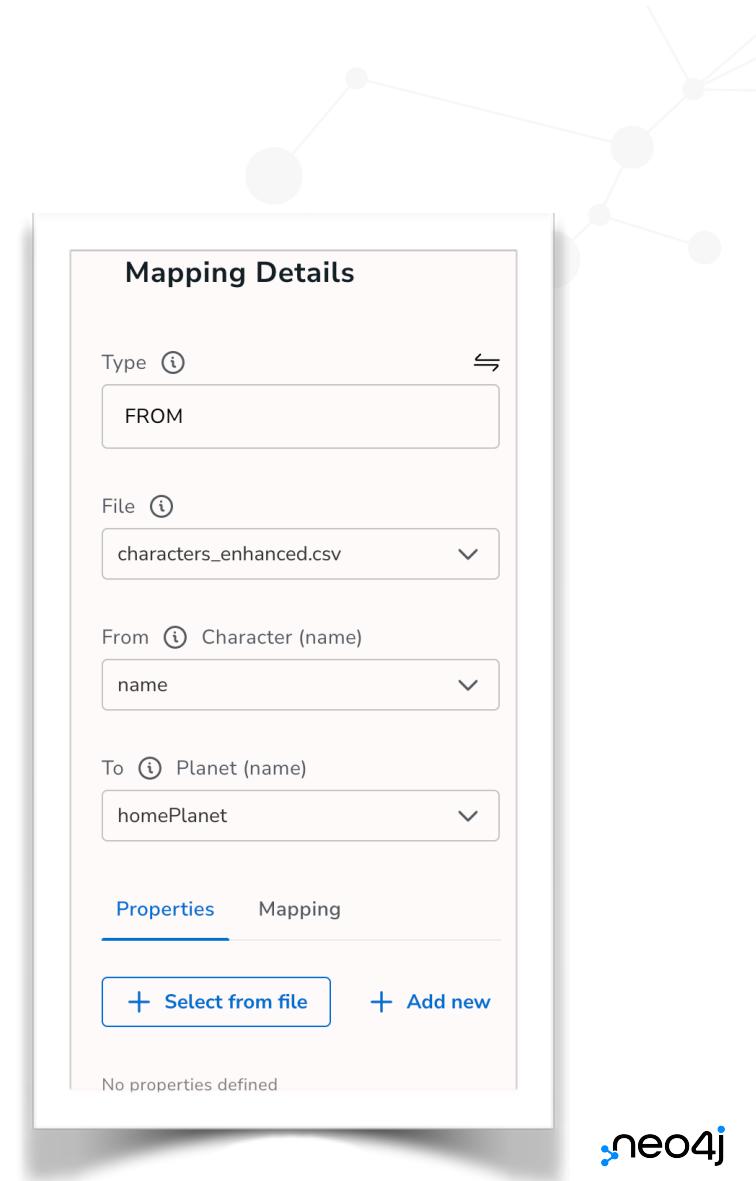


# Lab 2.5 - Using Workspace Importer

## Creating Relationships Between Nodes

- Complete the remaining fields on the Mapping Details tab.
- Type: The name of the relationship
- File: Which file has the source/target information for the relationship
- From: Which field contains the ID of the From node
- To: Which field contains the ID of the To node
- Properties: Any fields that should be used as relationship properties.

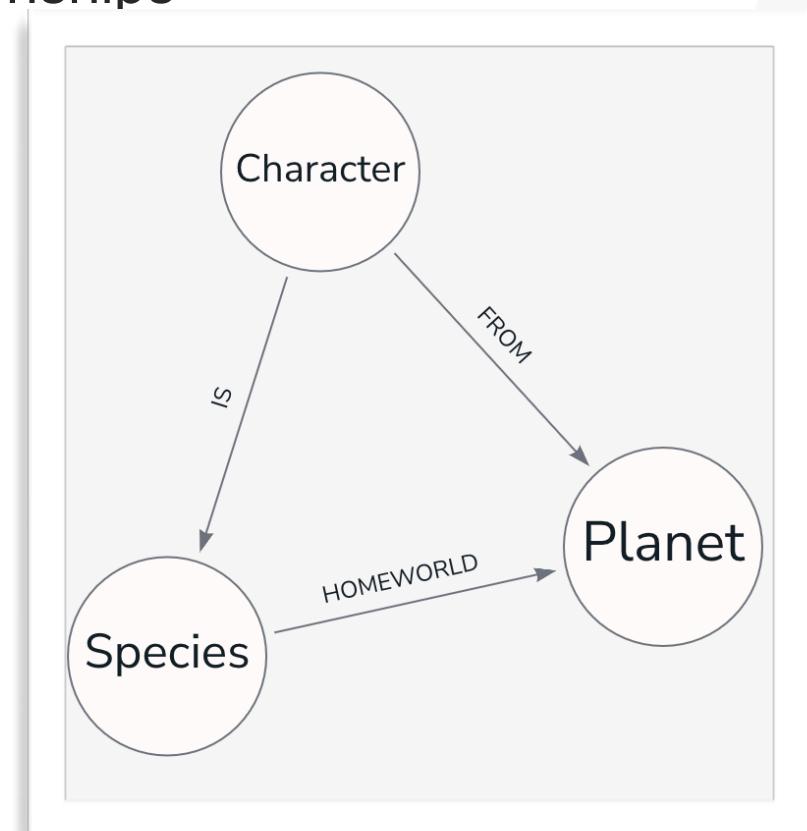
© 2022 Neo4j, Inc. All rights reserved.



# Lab 2.6 - Using Workspace Importer

## Load Species Nodes and Relationships

- Load species.csv file
- Use species name as ID field
- Create relationship from Character to Species.
- Create relationship from Character to Planet

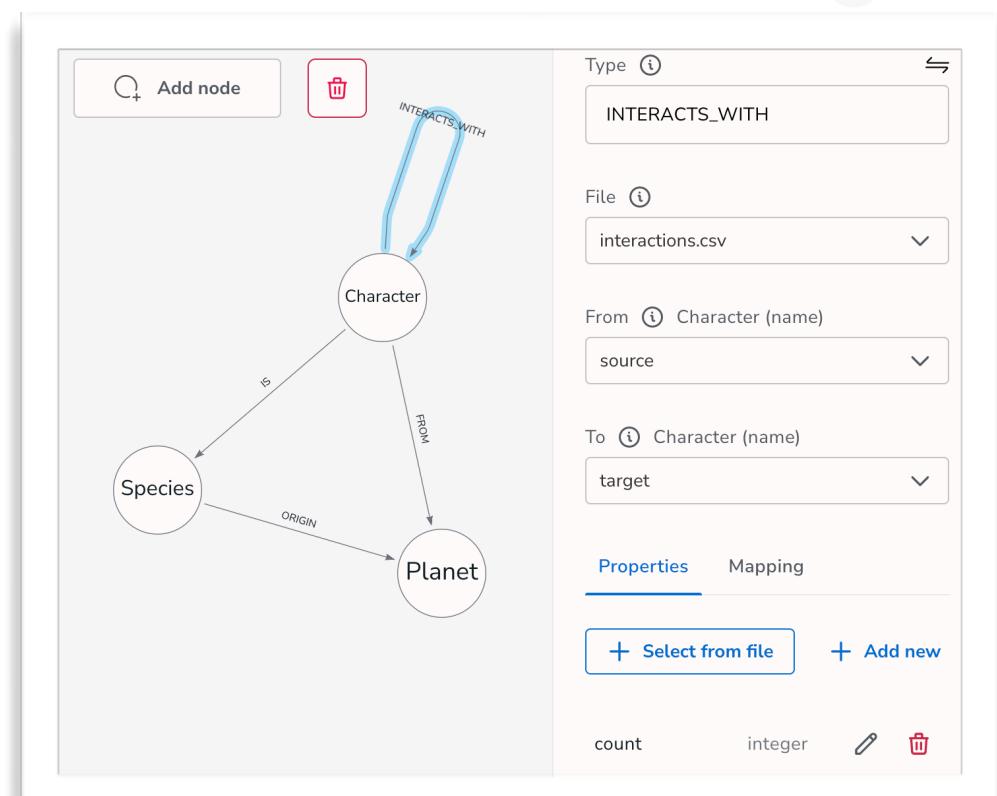


# Lab 2.7 - Using Workspace Importer

## Creating Self-Join Relationship for Characters

- Select the Character node and hover over its border until the + indicator appears
- Drag the selection + indicator away from the Character node, and then drag it back and drop it on the Character node.
- Fill out the mapping details.
- Type: INTERACTS\_WITH
- File: interactions.csv
- From: source
- To: target
- Properties: count

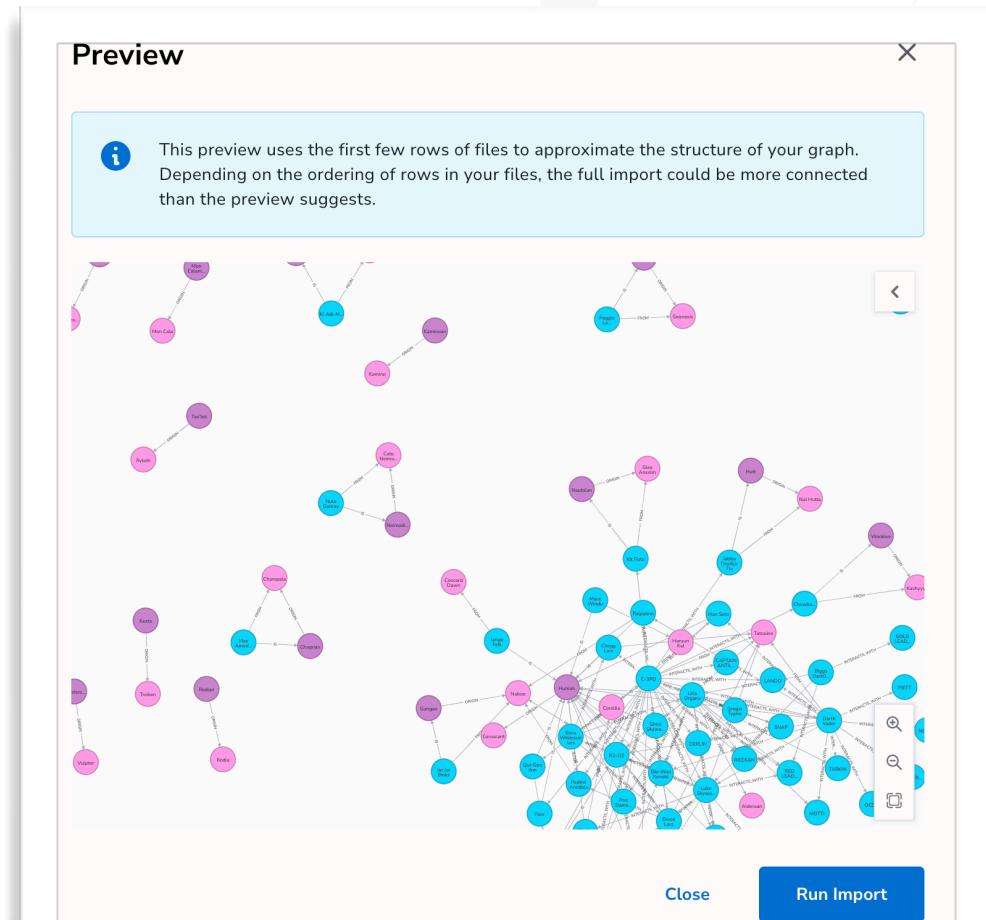
© 2022 Neo4j, Inc. All rights reserved.



# Lab 2.8 - Using Workspace Importer

## Preview the Import

- Click on the Preview button
- A sampling of records from each of the files will used to give an idea of how the nodes and relationships will look.
- If everything looks okay, click on the Run Import button.



# Lab 2.9 - Using Workspace Importer

## Run the Import

- Click on the Run Import button
- A summary of the import will be shown.
- Each Node Label and each Relationship Type will be summarized.
- If all looks good, click on the Start Exploring button.

### Import results

Import completed without errors • Total time: 00:00:04

Species	species.csv	Show cypher					
Time Taken	File Size	File Rows	Nodes Created	Properties Set	Labels Added	Query Count	Query Time
00:00:01	3.2 KiB	37	37	346	37	1	00:00:04

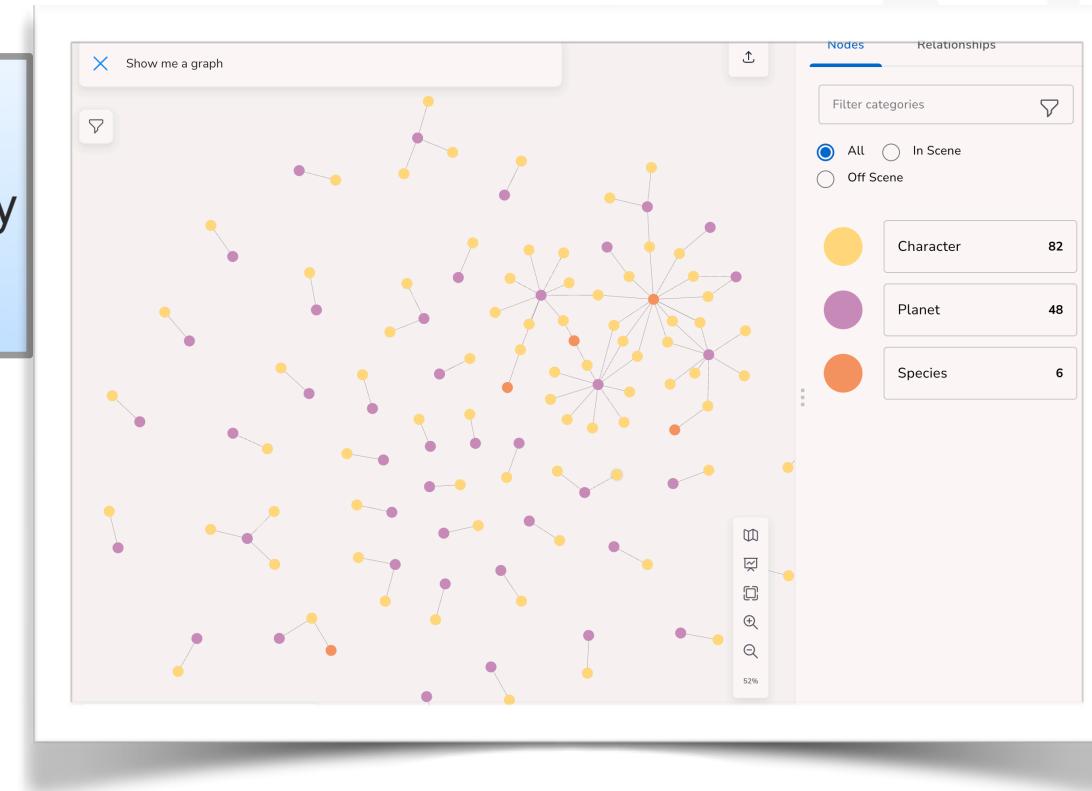
FROM	characters_enhanced.csv	Show query				
Time Taken	File Size	File Rows	Relationships Created	Properties Set	Query Count	Query Time
00:00:00	16.0 KiB	151	82	0	1	00:00:00

[Close](#) [Start Exploring](#)

# Lab 2.9.9 - Using Workspace Importer

## Viewing your Handiwork

- Click on the Start Exploring button
- This will show the Explore tab which uses Neo4j Bloom to display the newly imported data



# Lab 3 - Querying the Graph



# Lab 3.1 - Basic Queries - The Match Statement

- Most queries start with a MATCH statement which is used for pattern matching.
- It is generally the most important part of the query
- All nodes/relationships/paths that meet the filtering criteria are selected.
- Selected items are then passed to the next part of the query

```
// Return all characters  
MATCH (c:Character)  
RETURN c
```

```
// Return a specific character  
MATCH (c:Character)  
WHERE c.name = "Han Solo"  
RETURN c
```

## Lab 3.2 - Basic Queries - The Match Statement

- Most queries start with a MATCH statement
- Used for pattern matching - similar in some ways to a WHERE clause
- All nodes/relationships/paths that meet the filtering criteria are selected
- Selected items are then passed to the next part of the query
- It is generally the most important part of the query

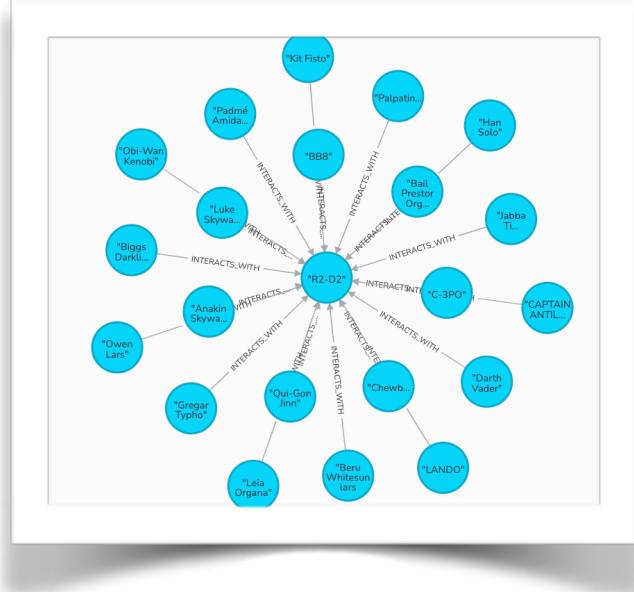
```
// Return all Characters from Alderaan
MATCH (c:Character)-[:FROM]->(p:Planet)
WHERE p.name = "Alderaan"
RETURN c
```

```
// Return a specific Character
MATCH (c:Character)
WHERE c.name = "Han Solo"
RETURN c
```

# Lab 3.4 - Basic Queries - The Match Statement

## Node Properties Used as a Filter

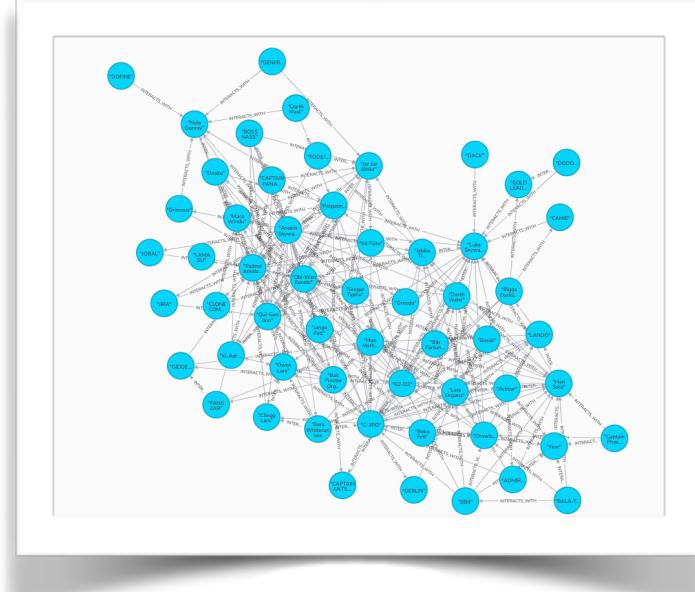
```
// Return all Characters that interacted with R2-D2
MATCH (c:Character)-[INTERACTS_WITH]-(other:Character)
WHERE c.name = "R2-D2"
RETURN other
```



## Lab 3.5- Basic Queries - The Match Statement

Unlike SQL, Variable-Length Queries are simple

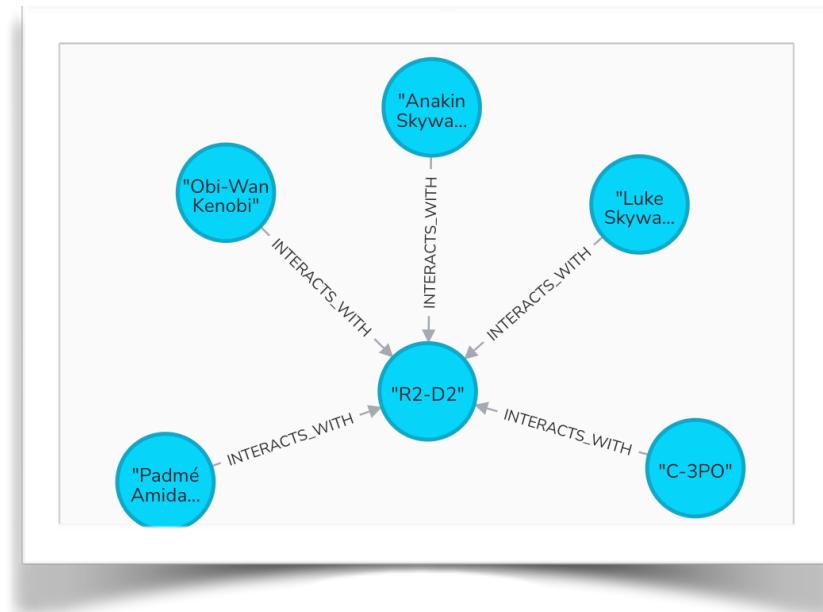
```
// Variable length query - interacts_with 3 levels down
MATCH p=(c:Character)->[INTERACTS_WITH*..3]-(other:Character)
WHERE c.name = "R2-D2"
RETURN p
```



# Lab 3.6 - Basic Queries - The Match Statement

Relationship Properties Can Also be Used

```
// Who does R2-D2 interact with the most
MATCH p=(c:Character)-[i:INTERACTS_WITH]-(other:Character)
WHERE c.name = "R2-D2"
RETURN p ORDER BY i.count DESC LIMIT 5
```



# Lab 3.7 - Basic Queries - The Match Statement

## Aggregation Values Used as a Filter

```
// Which characters interact with the most characters
MATCH (c:Character)-[INTERACTS_WITH]-(other)
WITH c.name as character, count(other) as others
WHERE others > 20
RETURN character, others
ORDER BY others desc LIMIT 5
```

character	others
"Anakin Skywalker"	44
"Obi-Wan Kenobi"	39
"C-3PO"	38
"Padm� Amidala"	36

Showing 1-5 of 5 results

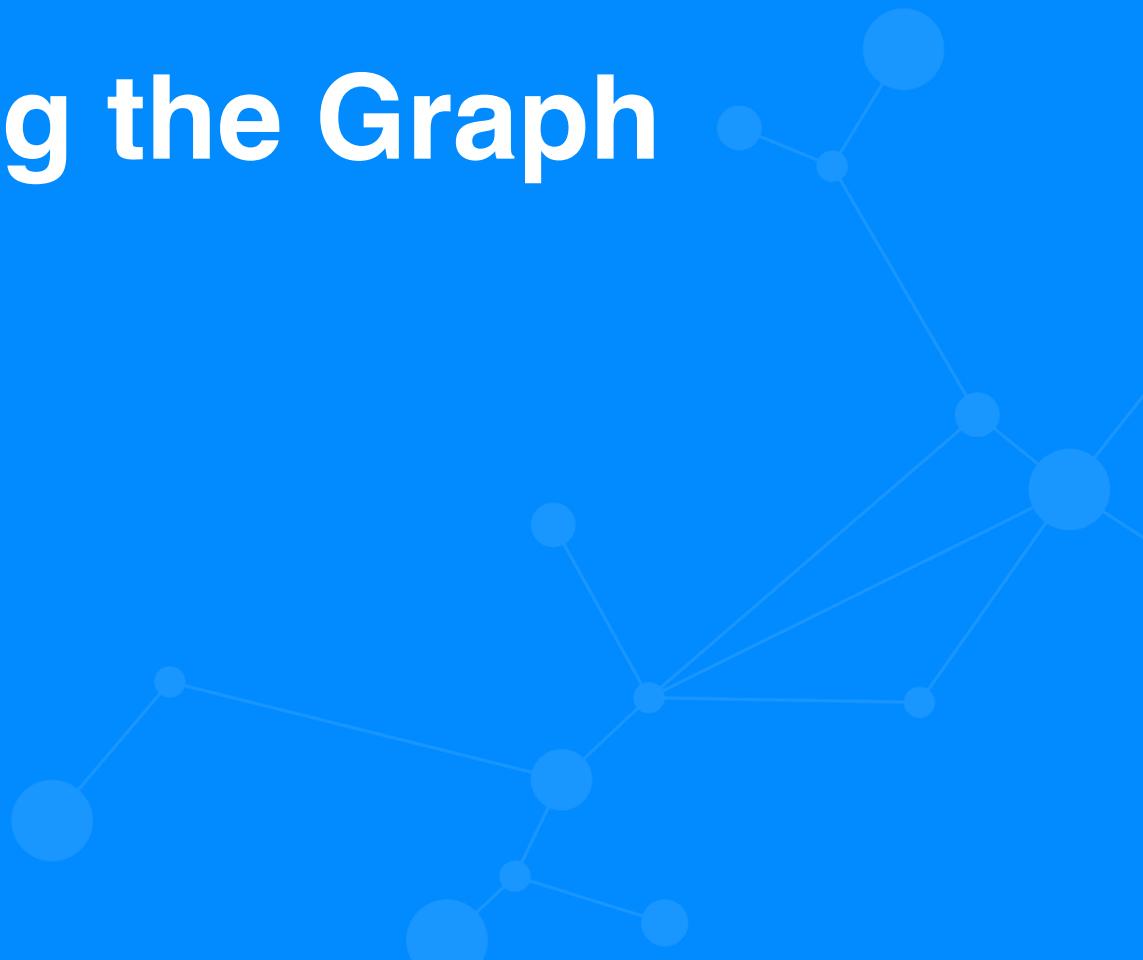
## Lab 3.8 - Basic Queries - The Match Statement

String Comparison Used as a Filter

```
// Which characters have Darth as part of their name
MATCH (c:Character)
WHERE c.name STARTS WITH "Darth"
RETURN c
```



# Lab 4 - Updating the Graph



# Lab 4.1 - Update Queries - Creating New Nodes

## Using the CREATE Statement

- New nodes can be created using the CREATE statement
- Unless uniqueness constraints are used, duplicates may result

```
// Create a new Character  
CREATE (c:Character {name:"Mark Q",  
height:74,hair_color:"Fleshtone"})  
RETURN c
```



"Mark Q"

## Lab 4.2 - Update Queries - Creating New Nodes

### Using the MERGE Statement

- New nodes can also be created using the MERGE statement
- Neo will check first to see if any nodes match before creating a new node

```
// Carefully create a new Character using MERGE
MERGE (c:Character {name:"Mark Q"})
ON CREATE SET c +=
{height:74,hair_colors:"Fleshtone"}
RETURN c
```



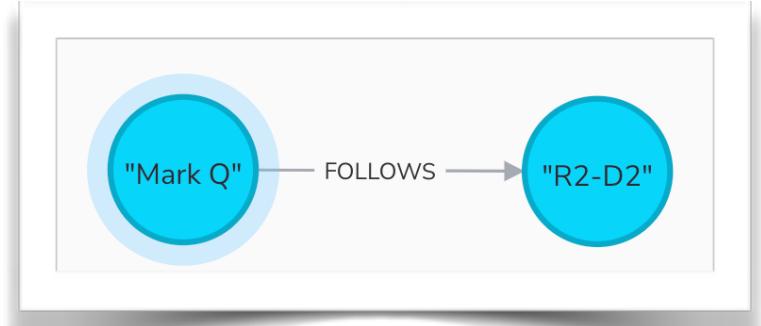
"Mark Q"

# Lab 4.3 - Update Queries - Creating New Relationships

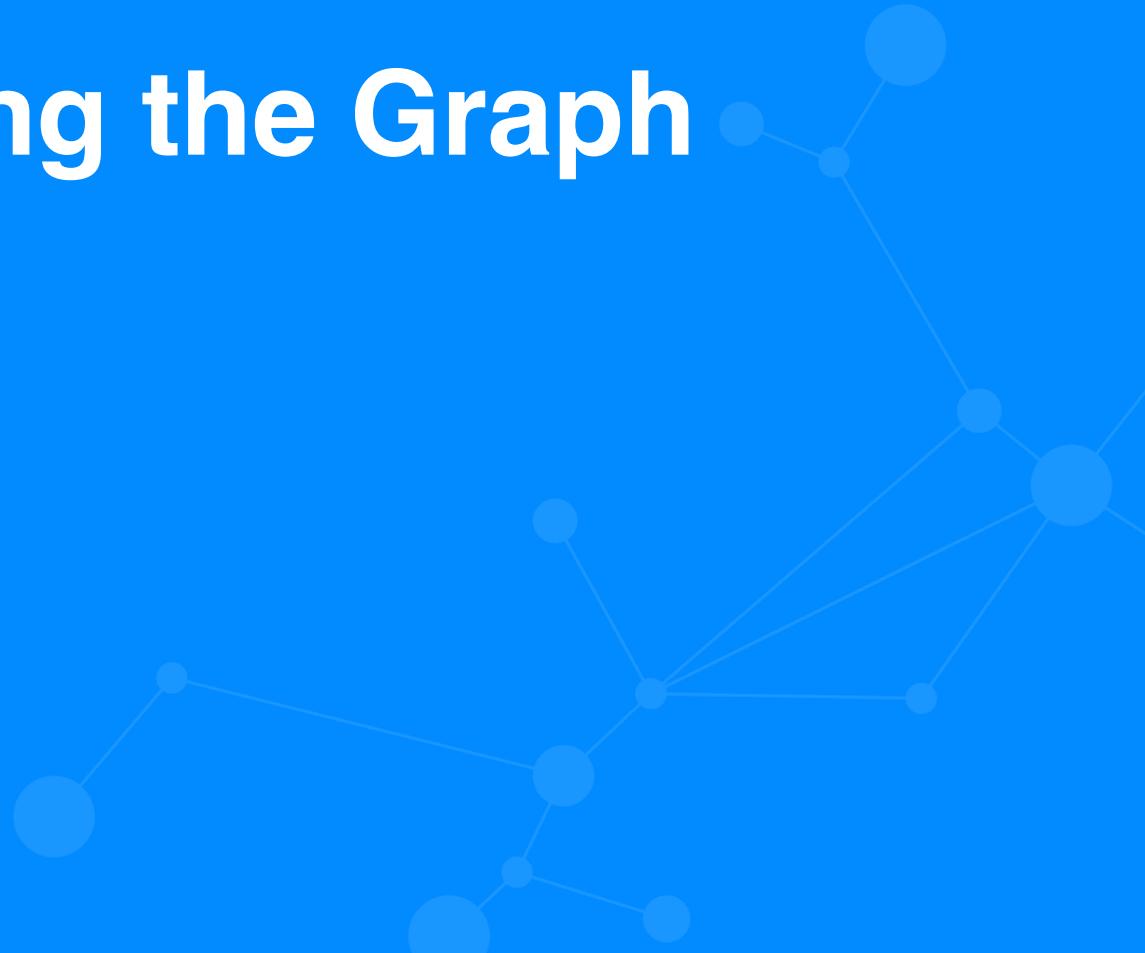
## Using the MERGE Statement

- New relationships can also be created using the MERGE statement
- Use the MATCH statement to find the “from” and “to” nodes
- Use the MERGE statement to create a new relationship (if necessary)

```
MATCH  
(r2:Character),  
(me:Character)  
WHERE r2.name = 'R2-D2' AND me.name = 'Mark Q'  
MERGE (me)-[r:FOLLOWS]->(r2)  
RETURN r2,r,me
```

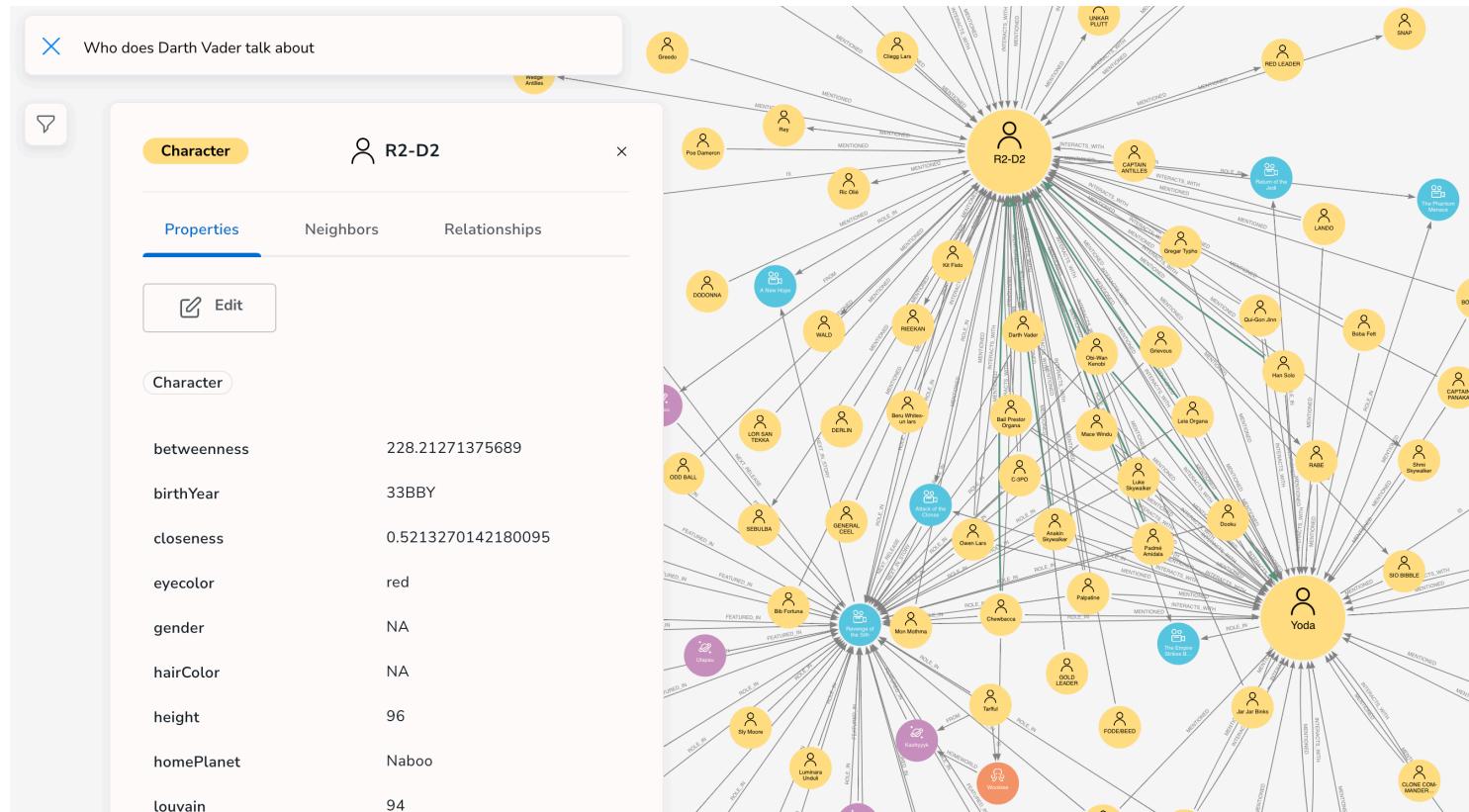


# Lab 5 - Exploring the Graph with Bloom



# Lab 5.1 - Exploring the Graph Using Bloom

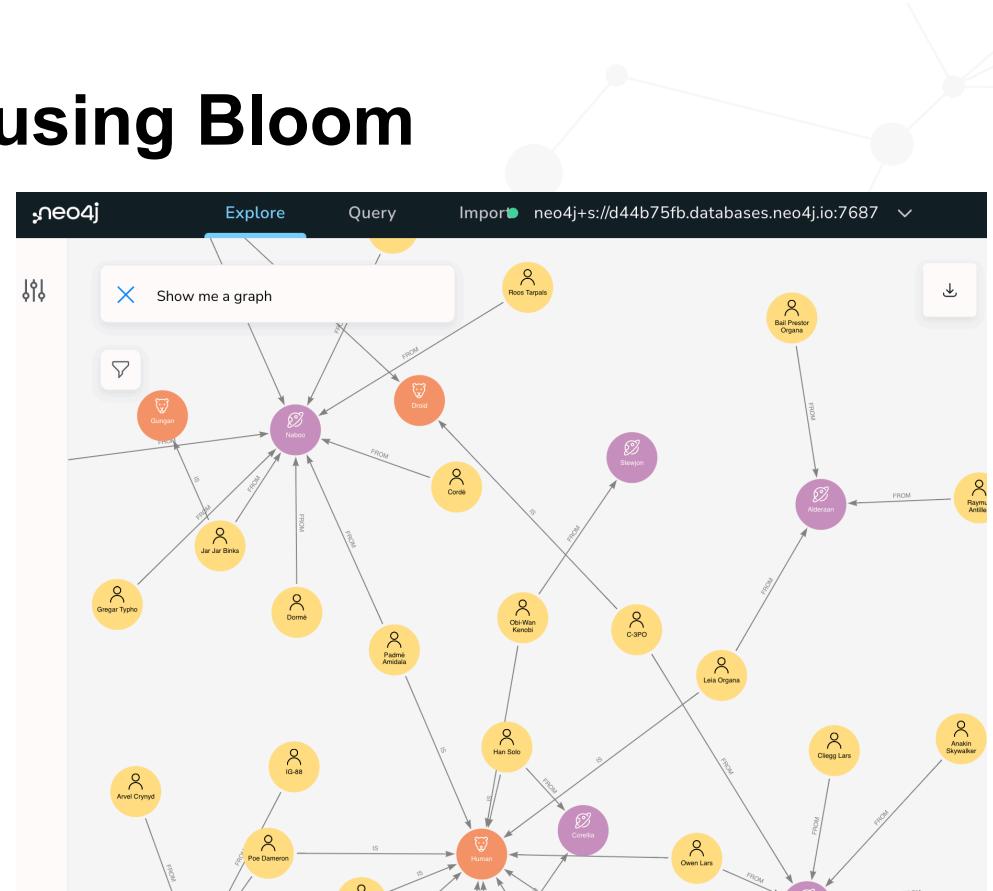
Explore the graph without writing code (unless you want to)



# Lab 5.2 - Exploring Your DB using Bloom

Switch to Explore Tab

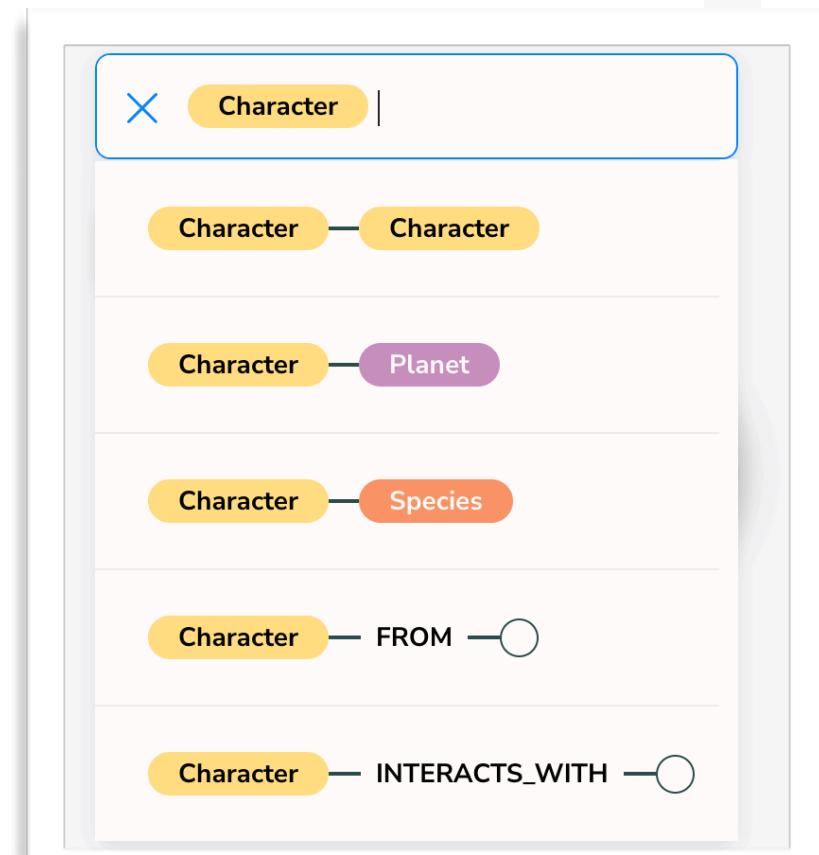
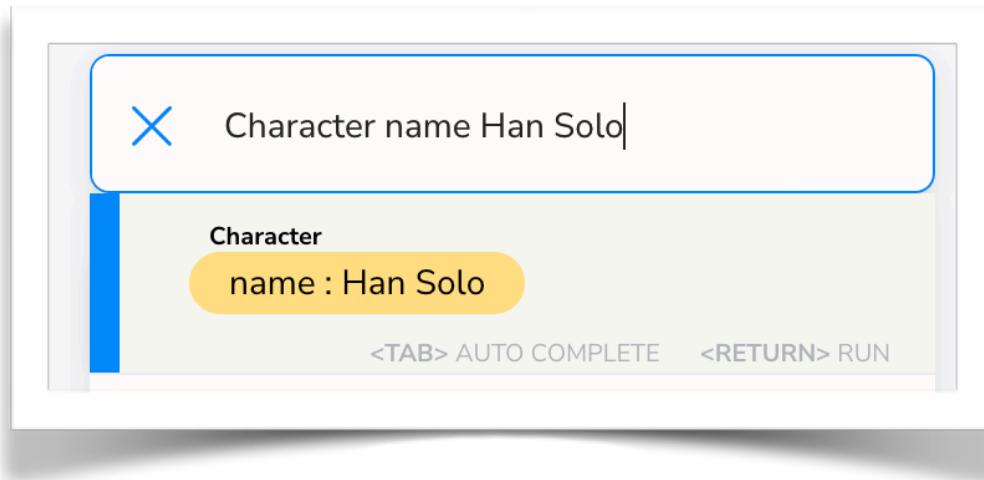
- Clicking on the Explore tab will invoke Neo4j Bloom.
- Bloom starts with a sample query to give an idea of the overall graph.



# Lab 5.3 - Exploring Your DB using Bloom

## Pattern Searching using NLP

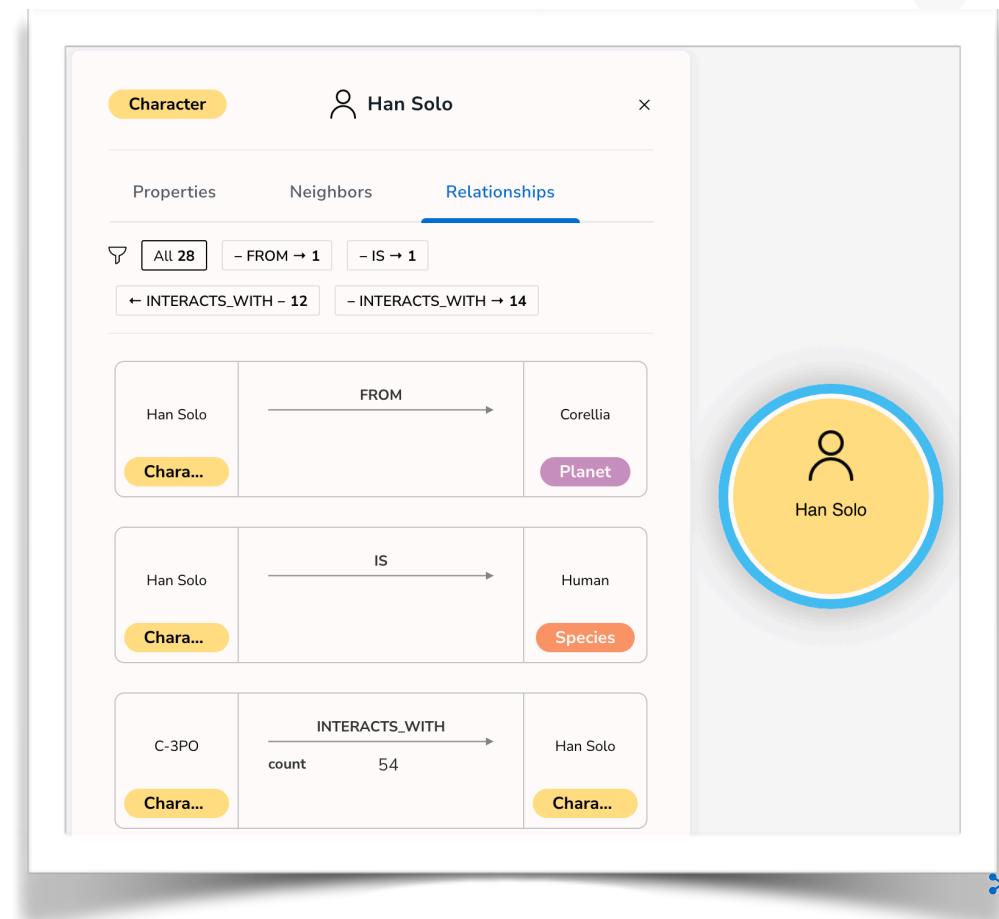
- Either type in an NLP phrase or build the pattern step by step



# Lab 5.4 - Exploring Your DB using Bloom

Inspect Nodes to see properties and relationships

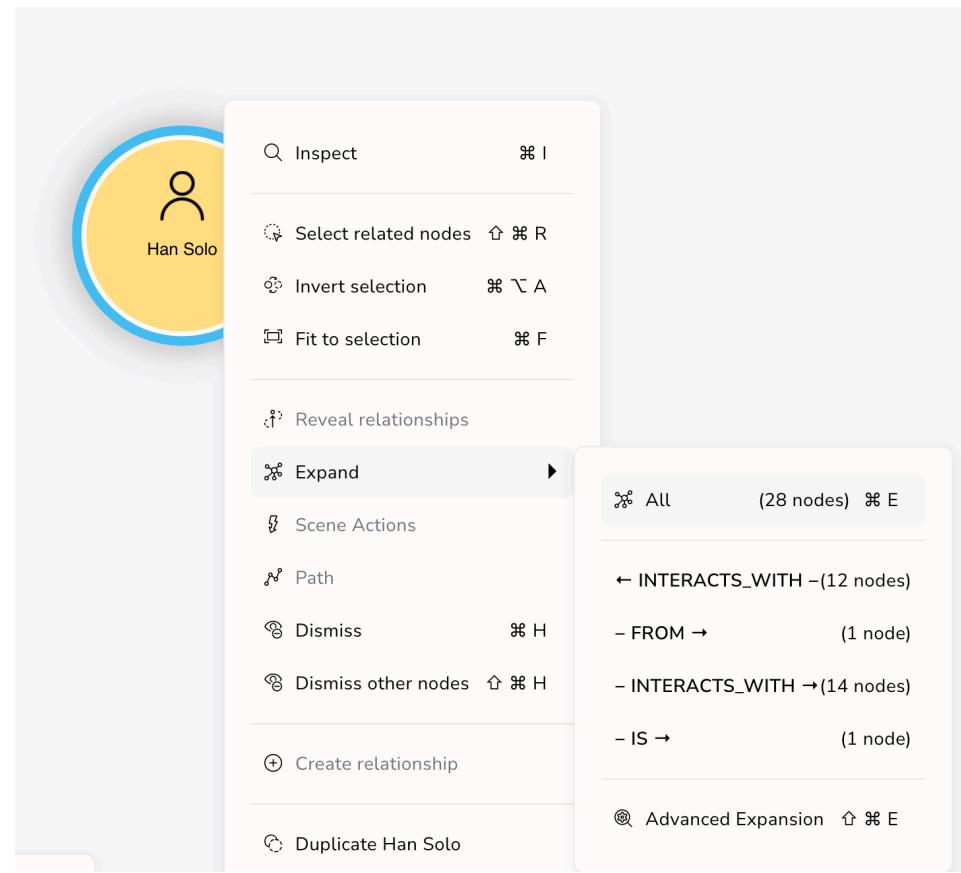
- Double-click on a node to inspect a node.
- Properties may be viewed and updated
- Related nodes can be viewed in the Neighbors tab or Relationships tab.



# Lab 5.5 - Exploring Your DB using Bloom

## Expanding Nodes to show related nodes

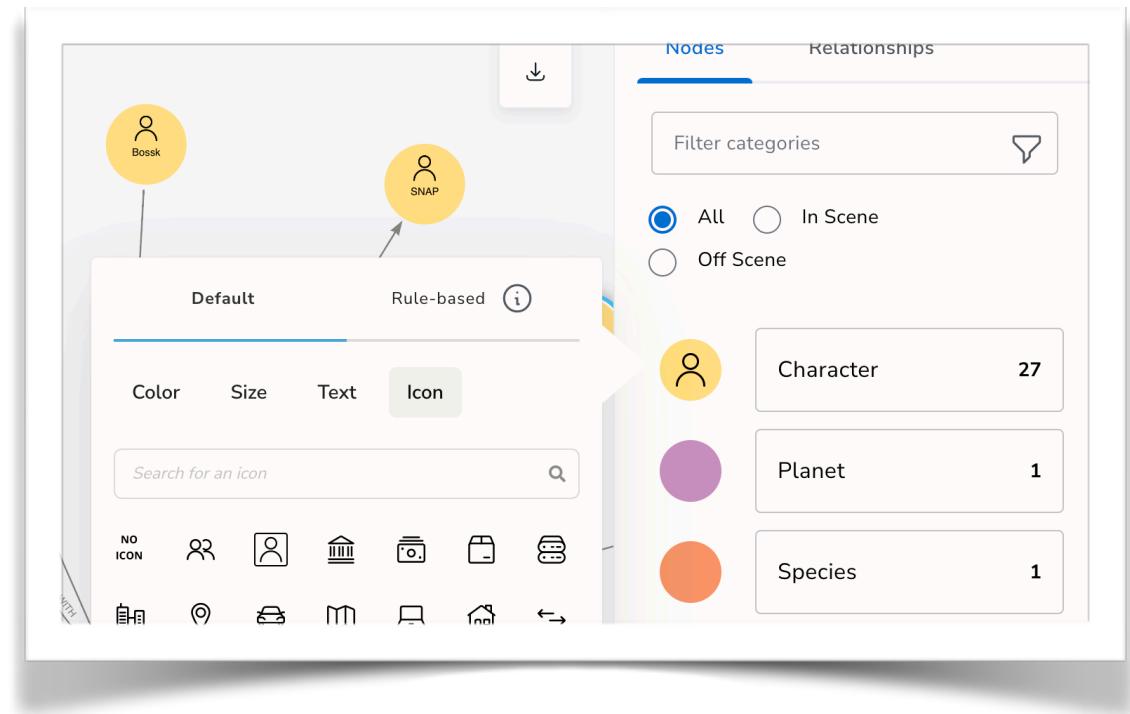
- Right click on a node to show pop-up menu for a node.
- Choose ‘Expand’ to show related nodes. Choose some or all relationships



# Lab 5.6 - Exploring Your DB using Bloom

## Customizing the Display

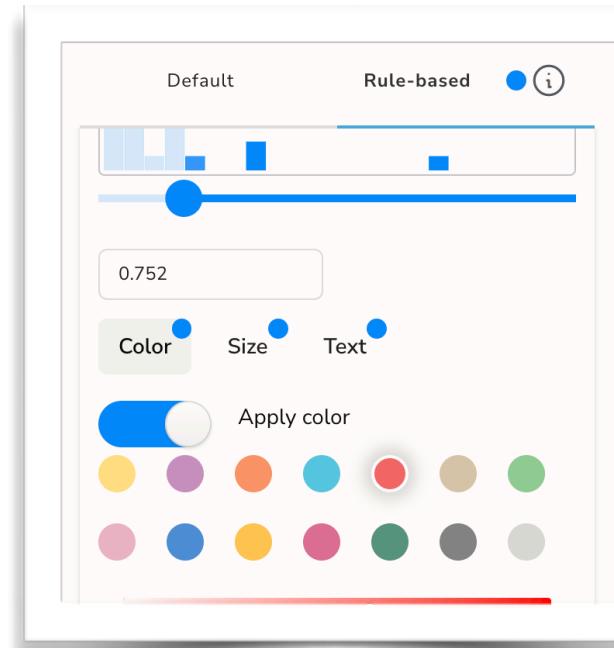
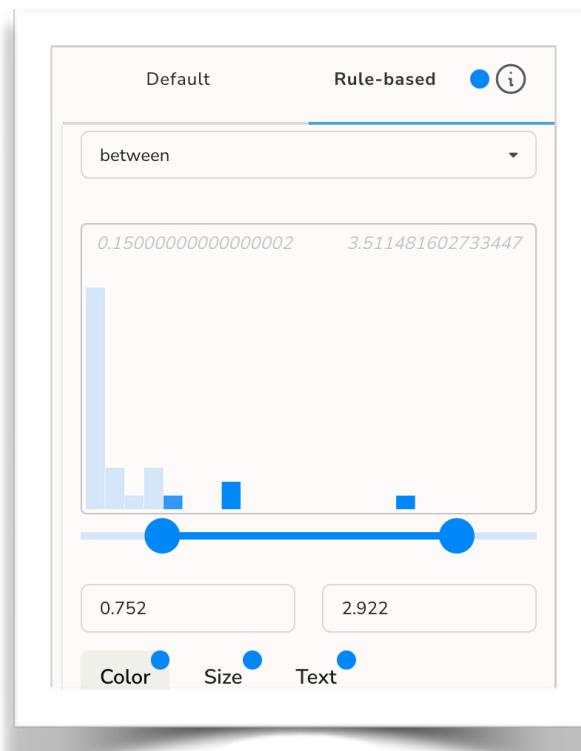
- Click on a Label to show display options.
- Color, size, text, and icons can all be selected.
- Rules can be used to highlight certain nodes



## Lab 5.7- Exploring Your DB using Bloom

Changing the size and color of Characters with high PageRank scores

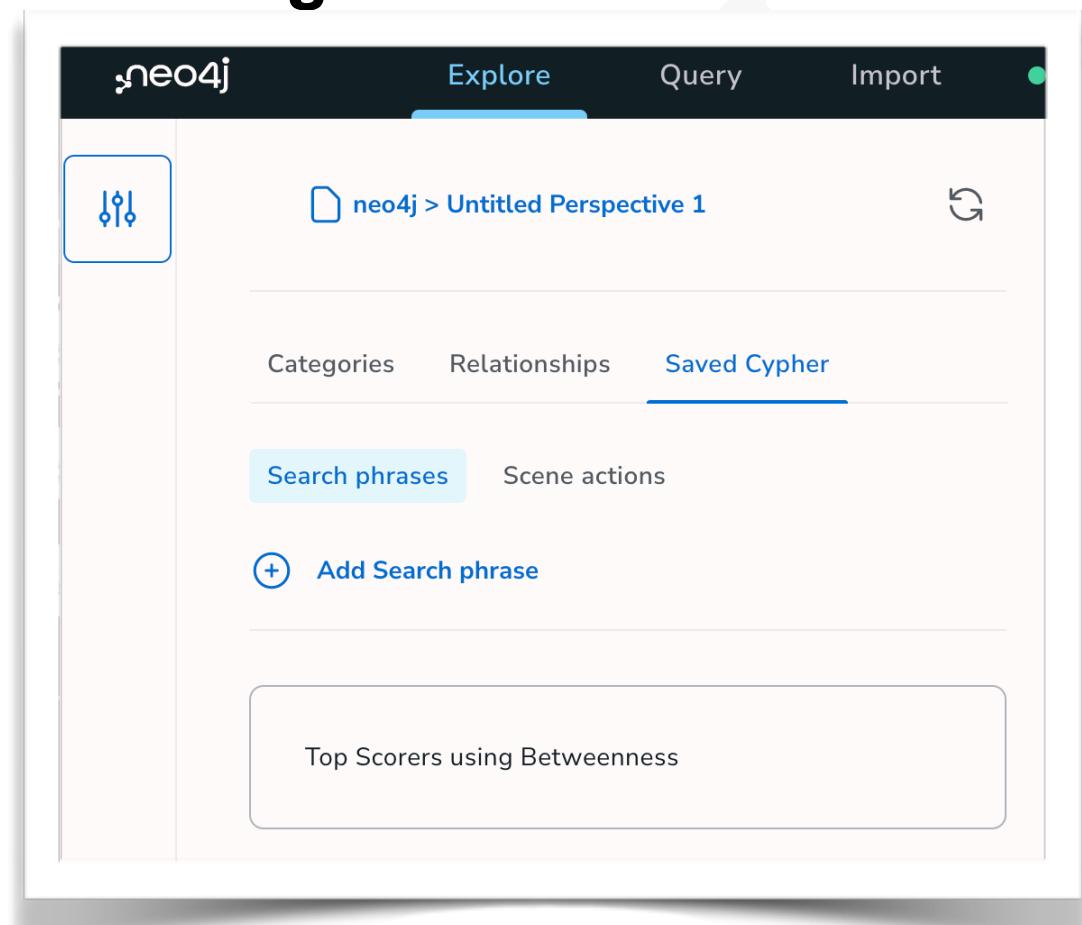
- Click on the Icon next to the Character Label
- Select Rule-based tab
- Choose the page rank property
- Choose single values and greater than or between.
- Use the slider to specify the values to use
- Choose a color and/or size for matching nodes



# Lab 5 - Exploring Your DB using Bloom

## Using Saved Cypher

- Click on the Perspectives Icon (upper left corner)
- Click on Saved Cypher / Search Phrases
- Add a search phrase



# Lab 5 - Exploring Your DB using Bloom

## Using Saved Cypher

- Give your search phrase a name and description
- Cut/paste the Cypher query that will be used
- Execute the query by typing its name into the search box

Top Scorers using Betweenness Delete

Search phrase \*

Description

Cypher query \*

```
MATCH (c:Character)
with
c ORDER BY c.pageRank
LIMIT 10
match p= (c)--(:Character)
return p
```

# Lab 5.8 - Parameterized Queries

Add a parameter to a Cypher statement

- Give your search phrase a name and description
- Add a parameter to the search phrase using \$
- Create a parameterized Cypher statement
  - cut/paste this:

```
MATCH path=(p:Planet {name: $planet})<-[FROM]-(c:Character)
RETURN path
```
- Define how Bloom should suggest parameter values

© 2022 Neo4j, Inc. All rights reserved.

The screenshot shows the Neo4j Bloom interface with the following fields filled in:

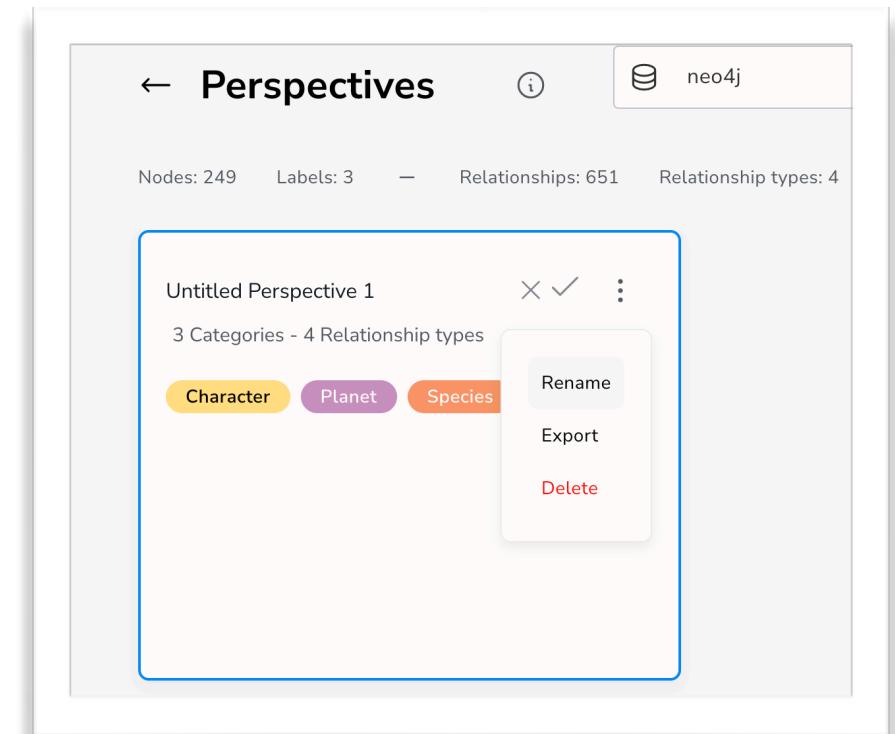
- Search phrase \***: Characters from \$planet
- Description**: Show Characters from a given planet
- Cypher query \***:

```
MATCH path=(p:Planet {name:$planet})<-[FROM]-(c:Character)
RETURN path
```
- Parameter \$planet**
  - Data type**: String
  - Search suggestions**: Label-key (with Planet selected)
  - Value**: name

## Lab 5.9- Export Perspectives

Use JSON to persist & share perspectives

- Perspectives may be exported to JSON files for persistence and versioning
- Export your perspective to a JSON file (optional)



# Lab 5.10- Import Perspectives

Import JSON to use a different perspective

- Perspectives may be imported from saved JSON files.
- Multiple JSON files may be uploaded for easily jumping from one perspective to another.
- Enterprise versions of Bloom have additional methods for sharing perspectives amongst team members
- Example JSON file is [https://github.com/mquinz/star\\_wars\\_demo/raw/master/bloom/demo\\_perspective.json](https://github.com/mquinz/star_wars_demo/raw/master/bloom/demo_perspective.json)

© 2022 Neo4j, Inc. All rights reserved.

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

- Perspectives:** The main title bar shows "Perspectives".
- Database:** neo4j
- Nodes:** 249
- Labels:** 3
- Relationships:** 651
- Relationship types:** 4
- Neo4j version:** 5.4-aura
- Import:** A blue button in the top right corner.
- Untitled Perspective 12345:** Contains 3 Categories - 4 Relationship types. Categories shown: Character (yellow), Planet (purple), Species (orange).
- Simple Example for Training:** Contains 5 Categories - 4 Relationship types. Categories shown: Character (yellow), Planet (purple), Species (orange), Starship (blue), Vehicle (green).

# Lab 6 - Using Python Notebooks



# Lab 6.1 - Neo4j Using Python Notebook

## Use official Neo4j Drivers

- Neo4j has 3 Python Drivers
- pip install neo4j

### Execute Cypher read query and export to DataFrame

In [34]:

```
# load any parameters
params = { 'name': 'Han Solo' }

query = """MATCH (p:Character {name: $name})-[i:INTERACTS_WITH]-(p2:Character)
    RETURN p2.name AS otherName, i.count as interactionCount
    ORDER BY interactionCount DESC LIMIT 10
"""

# run query to get results
result = conn.query(query, parameters=params)

# create a dataframe from Cypher Resultset
dtf_data = pd.DataFrame([dict(_) for _ in result])
dtf_data.head(3)
```

Out [34]:

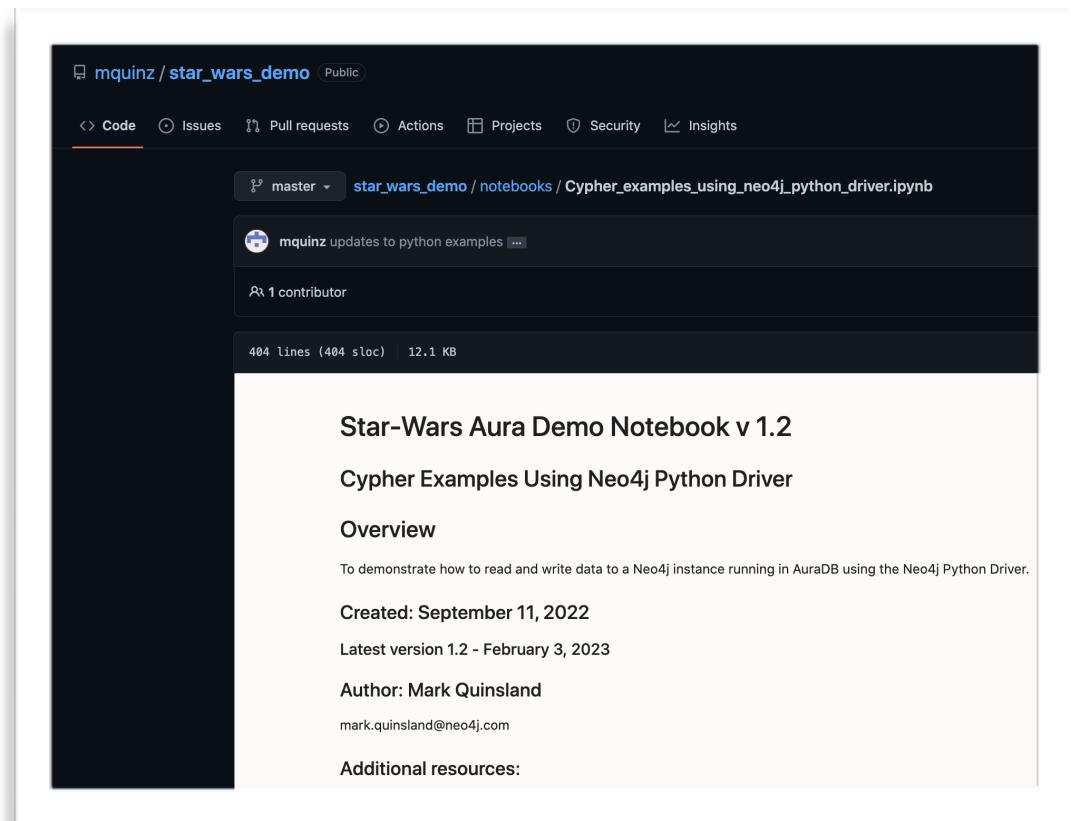
	otherName	interactionCount
0	Chewbacca	77
1	Leia Organa	69
2	C-3PO	54

[https://github.com/mquinz/star\\_wars\\_demo/blob/master/notebooks/Cypher\\_examples\\_using\\_neo4j\\_python\\_driver.ipynb](https://github.com/mquinz/star_wars_demo/blob/master/notebooks/Cypher_examples_using_neo4j_python_driver.ipynb)

# Lab 6.2 - Neo4j Using Python Notebook

## Download Example

- [https://github.com/mquinz/star\\_wars\\_demo/blob/master/notebooks/Cypher\\_examples\\_using\\_neo4j\\_python\\_driver.ipynb](https://github.com/mquinz/star_wars_demo/blob/master/notebooks/Cypher_examples_using_neo4j_python_driver.ipynb)



# Lab 6.3 - Neo4j Using Python Notebook

## Execute Query and Process Results

```
# load any parameters using a dictionary

params = {'name': 'Han Solo' }

# cypher - who does Han Solo interact with the most?
query = """MATCH (p:Character {name: $name})-[i:INTERACTS_WITH]-(p2:Character)
            RETURN p2.name AS otherName, i.count as interactionCount
            ORDER BY interactionCount DESC LIMIT 10

"""

result = conn.query(query, parameters=params)

for row in result:
    print (row['otherName'], row['interactionCount'])
```



# Lab 6.4 - Neo4j Using Python Notebook

## Load Query Results to DataFrame

```
# load any parameters

params = {'name': 'Han Solo' }

query = """MATCH (p:Character {name: $name})-[i:INTERACTS_WITH]-(p2:Character)
    RETURN p2.name AS otherName, i.count as interactionCount
    ORDER BY interactionCount DESC LIMIT 10
"""

# run query to get results
result = conn.query(query, parameters=params)

# create a dataframe from Cypher Resultset
dtf_data = pd.DataFrame([dict(_) for _ in result])
dtf_data.head(3)
```

	otherName	interactionCount
0	Chewbacca	77
1	Leia Organa	69
2	C-3PO	54

# Lab 6.5 - Neo4j Using Python Notebook

## Insert Nodes and Relationships Using Batches

```
params = { }

# create a batch of dictionaries - each item will be processed individually within the batch
bunch_of_rows = [
    {'name': 'Mark', 'gender': 'male', 'planet': 'Endor', 'jedi': 'true'},
    {'name': 'Abby', 'gender': 'female', 'planet': 'Endor', 'jedi': 'false'},
    {'name': 'Grokster', 'gender': 'male', 'planet': 'Alderaan', 'jedi': 'false'}
]

params ['rows'] = bunch_of_rows

query = """

// use UNWIND to process each item in the array
UNWIND $rows AS row
    MERGE (u:Character {name: row.name})
    SET u.gender = row.gender,
        u.jedi = toBoolean(row.jedi)

    // now create a relationship from the Character to their home planet

    WITH u, row.planet as homePlanet
        // find the planet and create a relationship to it
        MATCH (p:Planet)
            WHERE p.name = homePlanet
        MERGE (u)-[:FROM]->(p)
return count(*) as total
"""

# run query to get results
result = conn.query(query, parameters=params)
print (result)
```

