Star-Wars® Jedi Workshop

Version 1.5

Mark Quinsland
Senior Field Engineer, Neo4j
March 13, 2023

Neo4j Star-Wars® Jedi Workshop Agenda

- Quick review of Neo4j
- Creating a Jedi data model using Arrows
- Using AuraDB - Neo4j's Cloud DB
- Creating a Star-Wars® graph using Importer
- Exploring the graph visually using Bloom
- Cypher Queries to read, write, update and delete
- Loading Jedi data into Neo4j using Python
- Creating Jedi DataFrames using Jupyter notebooks
- Analyzing characters using Graph Data Science algorithms

neo4j

The World's Most Popular Property Graph DB

100m+ Neo4j Downloads
250k+ Community Members

7 **Retailers** of the Top 10

7 **Telcos** of the Top 10

8 **Automakers** of the Top 10

75+

FORTUNE 100

5 **Pharmaceuticals** of the Top 5

8 **Insurance** of the Top 10

3 **Hotels** of the Top 5

20 **Banks** of the Top 20 North American

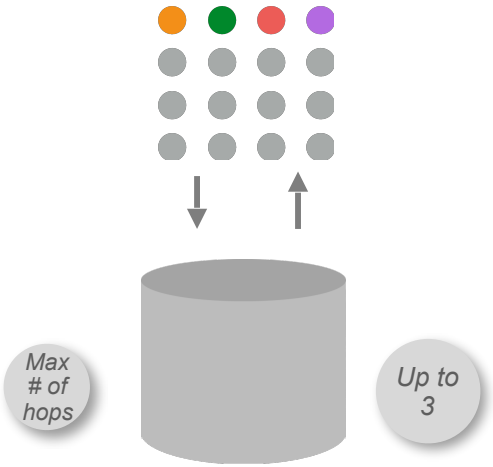
3 **Aircraft Manufacturers** of the Top 5

neo4j

Data Platform Evolution

TRADITIONAL DATABASES

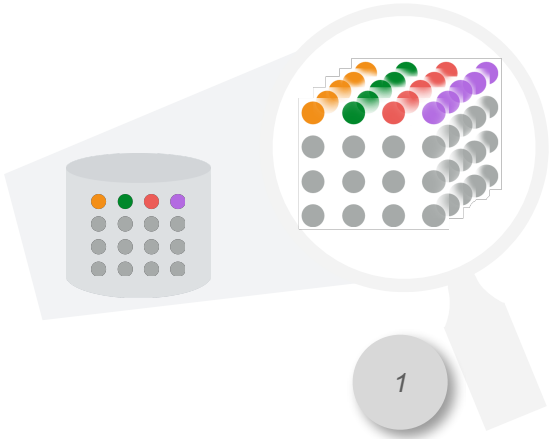
Store and retrieve data



Real time storage & retrieval for simple tabular data

BIG DATA TECHNOLOGY

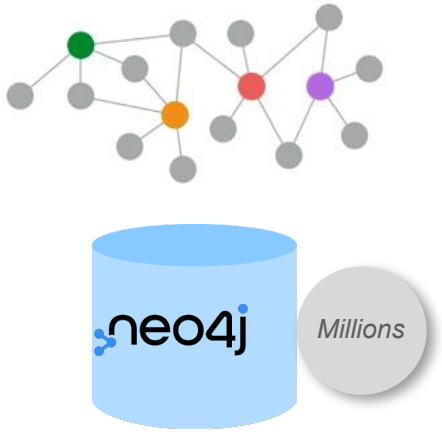
Aggregate and filter data



Long running queries, Aggregation & filtering

neo4j

Exploit connections in data



Real-time graph queries & algorithms for contextual insights

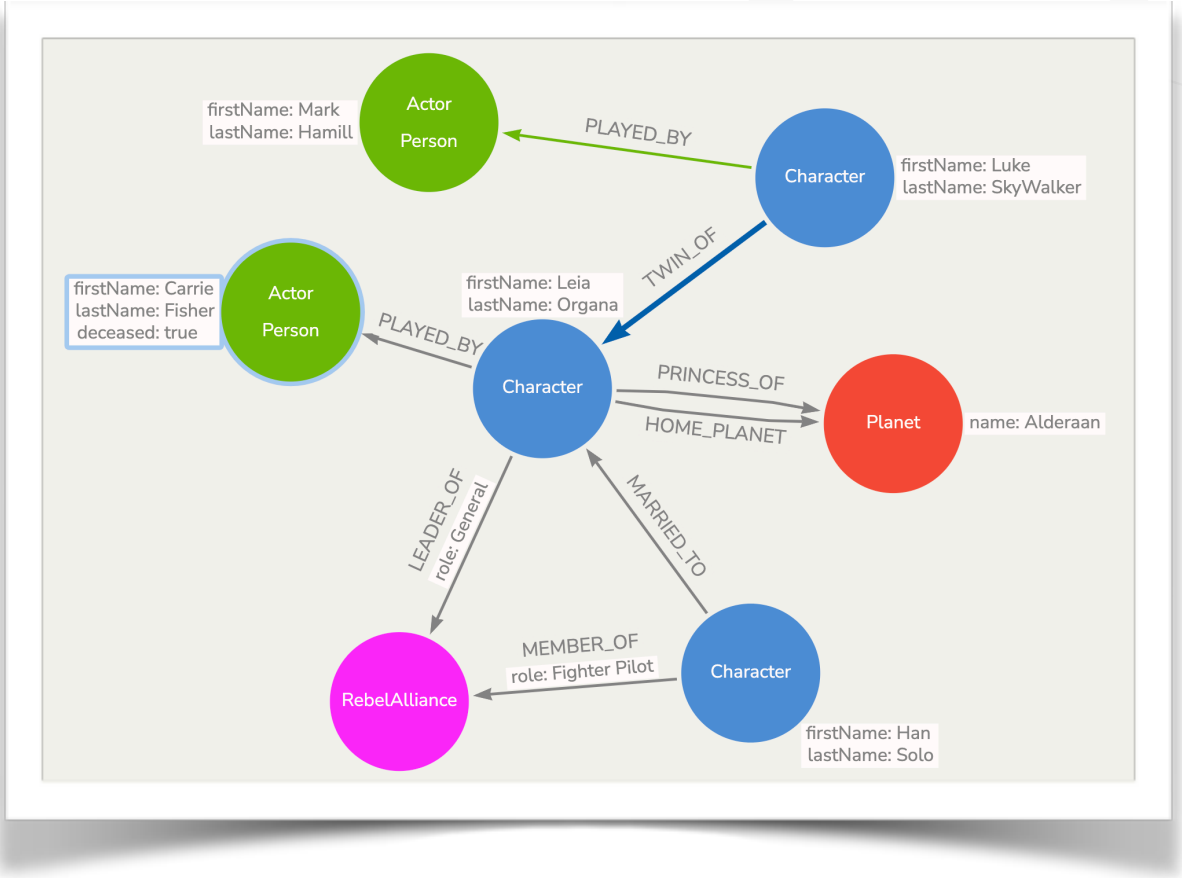
"Our **Neo4j** solution is literally **thousands of times faster** than the prior MySQL solution, with queries that require **10-100 times less code**"
Volker Pacher, Senior Developer

the Labeled Property Graph Model

Nodes
- Define the nouns/entities
- Can have multiple labels

Relationships
- Verbs that relate nodes by type and direction
- Nodes can have multiple relationships

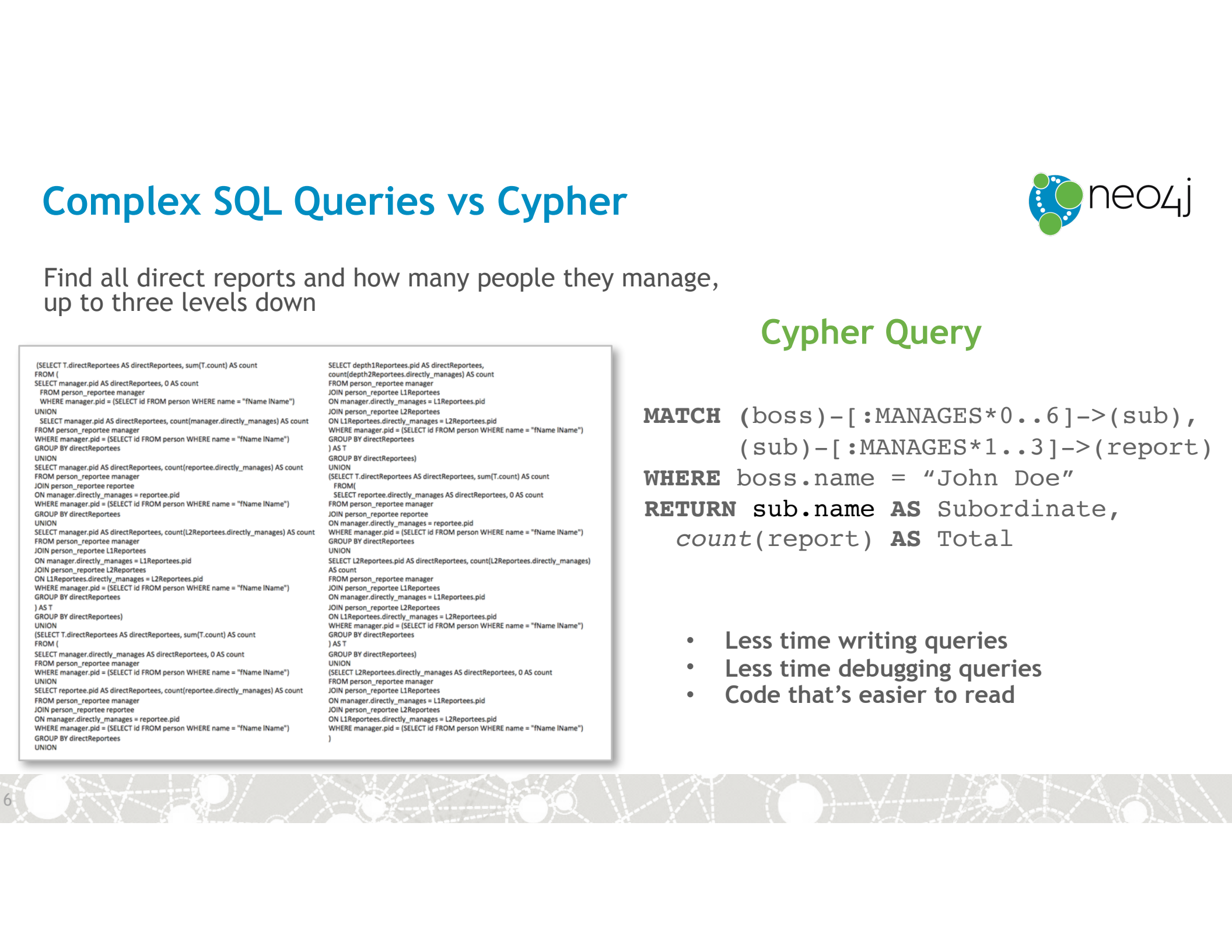
Properties
- Attributes of Nodes and Relationships
- Stored as name/value pairs



- New labels, relationship types and properties can be added w/o schema change

neo4j

Complex SQL Queries vs Cypher



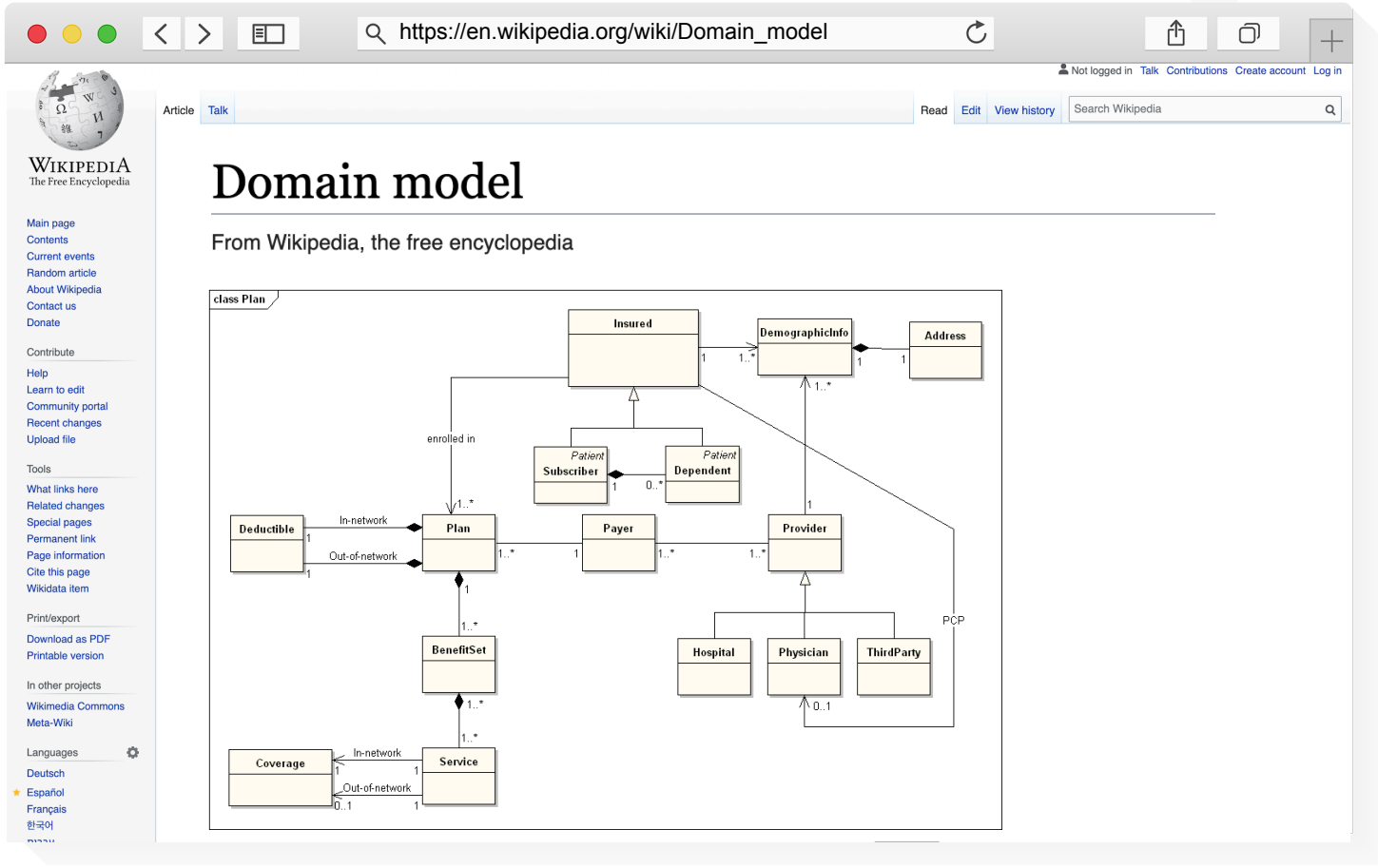
Find all direct reports and how many people they manage, up to three levels down

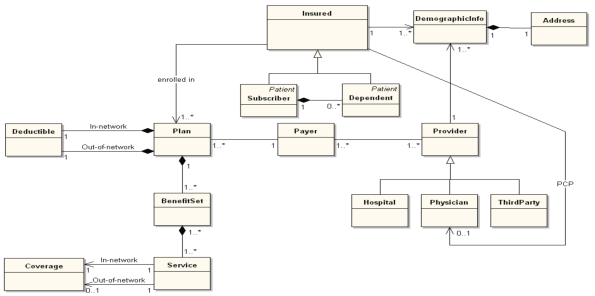
```
(SELECT T.directReportees AS directReportees, sum(T.count) AS count
FROM (
SELECT manager.pid AS directReportees, 0 AS count
  FROM person_reportee manager
  WHERE manager.pid = (SELECT id FROM person WHERE name = "fName lName")
UNION
  SELECT manager.pid AS directReportees, count(manager.directly_manages) AS count
FROM person_reportee manager
WHERE manager.pid = (SELECT id FROM person WHERE name = "fName lName")
GROUP BY directReportees
UNION
SELECT manager.pid AS directReportees, count(reportee.directly_manages) AS count
FROM person_reportee manager
JOIN person_reportee reportee
ON manager.directly_manages = reportee.pid
WHERE manager.pid = (SELECT id FROM person WHERE name = "fName lName")
GROUP BY directReportees
UNION
SELECT manager.pid AS directReportees, count(L2Reportees.directly_manages) AS count
FROM person_reportee manager
JOIN person_reportee L1Reportees
ON manager.directly_manages = L1Reportees.pid
JOIN person_reportee L2Reportees
ON L1Reportees.directly_manages = L2Reportees.pid
WHERE manager.pid = (SELECT id FROM person WHERE name = "fName lName")
GROUP BY directReportees
) AS T
GROUP BY directReportees)
UNION
(SELECT T.directReportees AS directReportees, sum(T.count) AS count
FROM (
SELECT manager.directly_manages AS directReportees, 0 AS count
FROM person_reportee manager
WHERE manager.pid = (SELECT id FROM person WHERE name = "fName lName")
UNION
SELECT reportee.pid AS directReportees, count(reportee.directly_manages) AS count
FROM person_reportee manager
JOIN person_reportee reportee
ON manager.directly_manages = reportee.pid
WHERE manager.pid = (SELECT id FROM person WHERE name = "fName lName")
GROUP BY directReportees
UNION
SELECT depth1Reportees.pid AS directReportees,
count(depth2Reportees.directly_manages) AS count
FROM person_reportee manager
JOIN person_reportee L1Reportees
ON manager.directly_manages = L1Reportees.pid
JOIN person_reportee L2Reportees
ON L1Reportees.directly_manages = L2Reportees.pid
WHERE manager.pid = (SELECT id FROM person WHERE name = "fName lName")
GROUP BY directReportees
) AS T
GROUP BY directReportees)
UNION
(SELECT T.directReportees AS directReportees, sum(T.count) AS count
  FROM(
  SELECT reportee.directly_manages AS directReportees, 0 AS count
FROM person_reportee manager
JOIN person_reportee reportee
ON manager.directly_manages = reportee.pid
WHERE manager.pid = (SELECT id FROM person WHERE name = "fName lName")
GROUP BY directReportees
UNION
SELECT L2Reportees.pid AS directReportees, count(L2Reportees.directly_manages)
AS count
FROM person_reportee manager
JOIN person_reportee L1Reportees
ON manager.directly_manages = L1Reportees.pid
JOIN person_reportee L2Reportees
ON L1Reportees.directly_manages = L2Reportees.pid
WHERE manager.pid = (SELECT id FROM person WHERE name = "fName lName")
GROUP BY directReportees
) AS T
GROUP BY directReportees)
UNION
(SELECT L2Reportees.directly_manages AS directReportees, 0 AS count
FROM person_reportee manager
JOIN person_reportee L1Reportees
ON manager.directly_manages = L1Reportees.pid
JOIN person_reportee L2Reportees
ON L1Reportees.directly_manages = L2Reportees.pid
WHERE manager.pid = (SELECT id FROM person WHERE name = "fName lName")
)
```

Cypher Query

```
MATCH (boss)-[:MANAGES*0..6]->(sub),
      (sub)-[:MANAGES*1..3]->(report)
WHERE boss.name = “John Doe”
RETURN sub.name AS Subordinate,
  count(report) AS Total
```

- Less time writing queries
- Less time debugging queries
- Code that’s easier to read





Id	Name	Address	Contact	Dependent
28430	Jane	San Mateo, CA	02349240	34324
34324	Ann	San Mateo, CA	02480240	

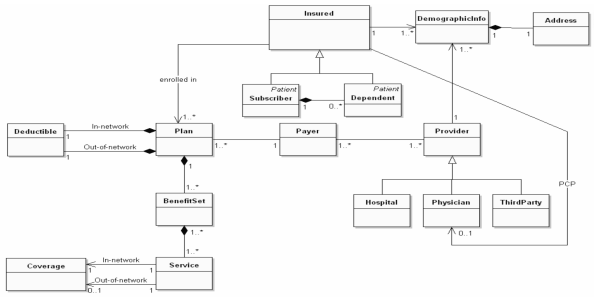
PatientId	PlanId	Enrolled
28430	32	2020-01-01

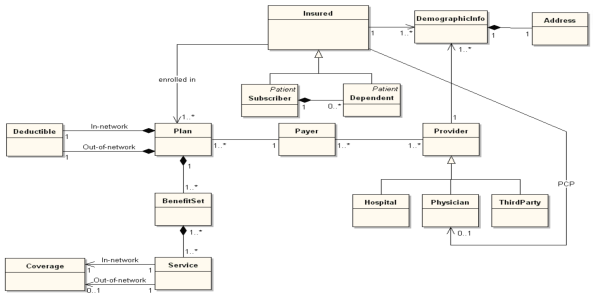
Id	Name	Provider
32	Plan X	2344

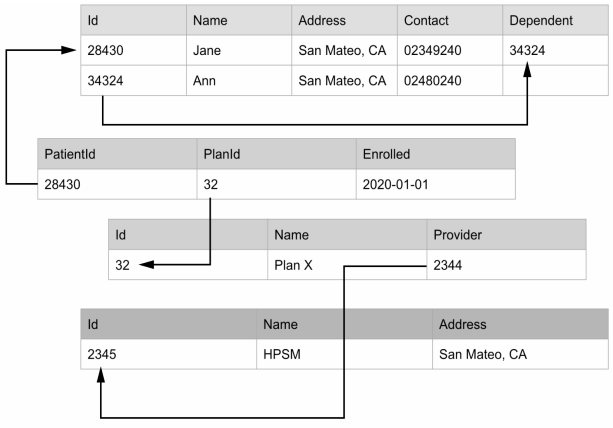
Id	Name	Address
2345	HPSM	San Mateo, CA

Relational

neo4j







Id	Name	Address	Contact	Dependent
28430	Jane	San Mateo, CA	02349240	34324
34324	Ann	San Mateo, CA	02480240	

PatientId	PlanId	Enrolled
28430	32	2020-01-01

Id	Name	Provider
32	Plan X	2344

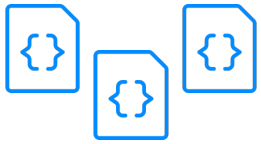
Id	Name	Address
2345	HPSM	San Mateo, CA

```
// Patient
{
  "_id": 593340651,
  "first": "Gregorio",
  "last": "Lang",
  "addr": {
      "street": "623 Flowers Rd",
      "city": "Groton",
      "state": "NH",
      "zip": 3266
  },
  "physicians": [
      10387 33456
  ],
  "procedures": [
    {"id" : "551ac", "type" : "Chest X-ray"},
    {"id" : "343fs", "type" : "Blood Test"}]
}
```

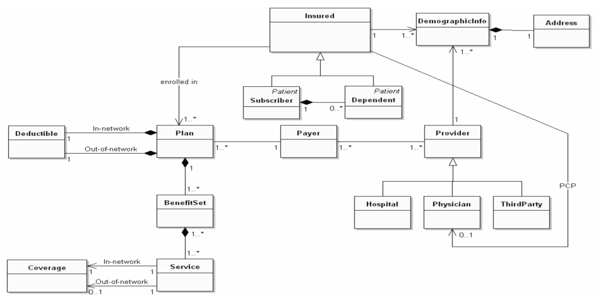
```
// Procedure
{
"_id": "551ac",
"date": "2000-04-26",
"hospital": 161,
"patient": 593340651,
"physician": 10387,
"type": "Chest
X-ray",
"records": [ "67bc6"]
}
```

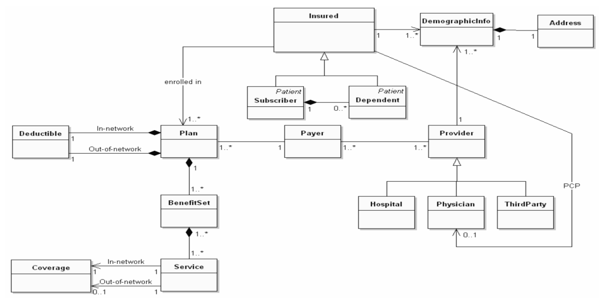
Relational

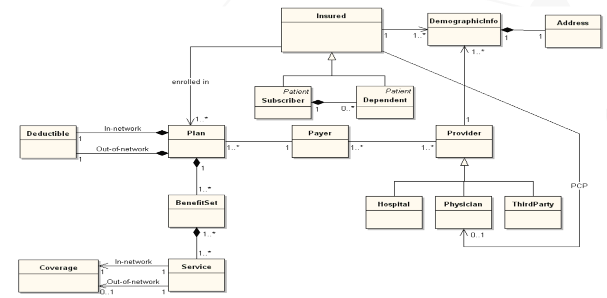
Document



neo4j







Id	Name	Address	Contact	Dependent
28430	Jane	San Mateo, CA	02349240	34324
34324	Ann	San Mateo, CA	02480240	

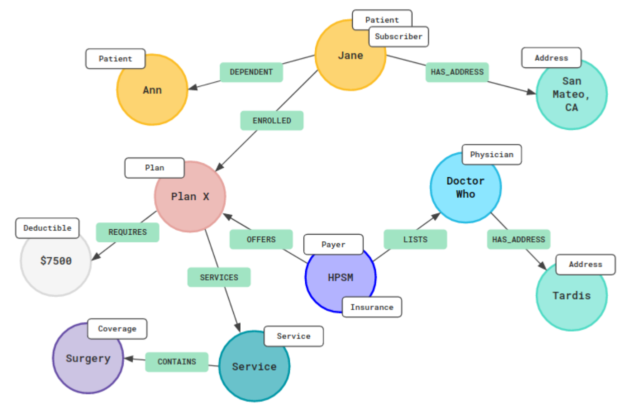
PatientId	PlanId	Enrolled
28430	32	2020-01-01

Id	Name	Provider
32	Plan X	2344

Id	Name	Address
2345	HPSM	San Mateo, CA

```
// Patient
{
  "_id": 593340651,
  "first": "Gregorio",
  "last": "Lang",
  "addr": {
      "street": "623 Flowers Rd",
      "city": "Groton",
      "state": "NH",
      "zip": 3266
  },
  "physicians": [
      10387 33456
  ],
  "procedures": [
    {"id" : "551ac", "type" : "Chest X-ray"},
    {"id" : "343fs", "type" : "Blood Test"}]
}
```

```
// Procedure
{
"_id": "551ac",
"date": "2000-04-26",
"hospital": 161,
"patient": 593340651,
"physician": 10387,
"type": "Chest
X-ray",
"records": [ "67bc6"]
}
```



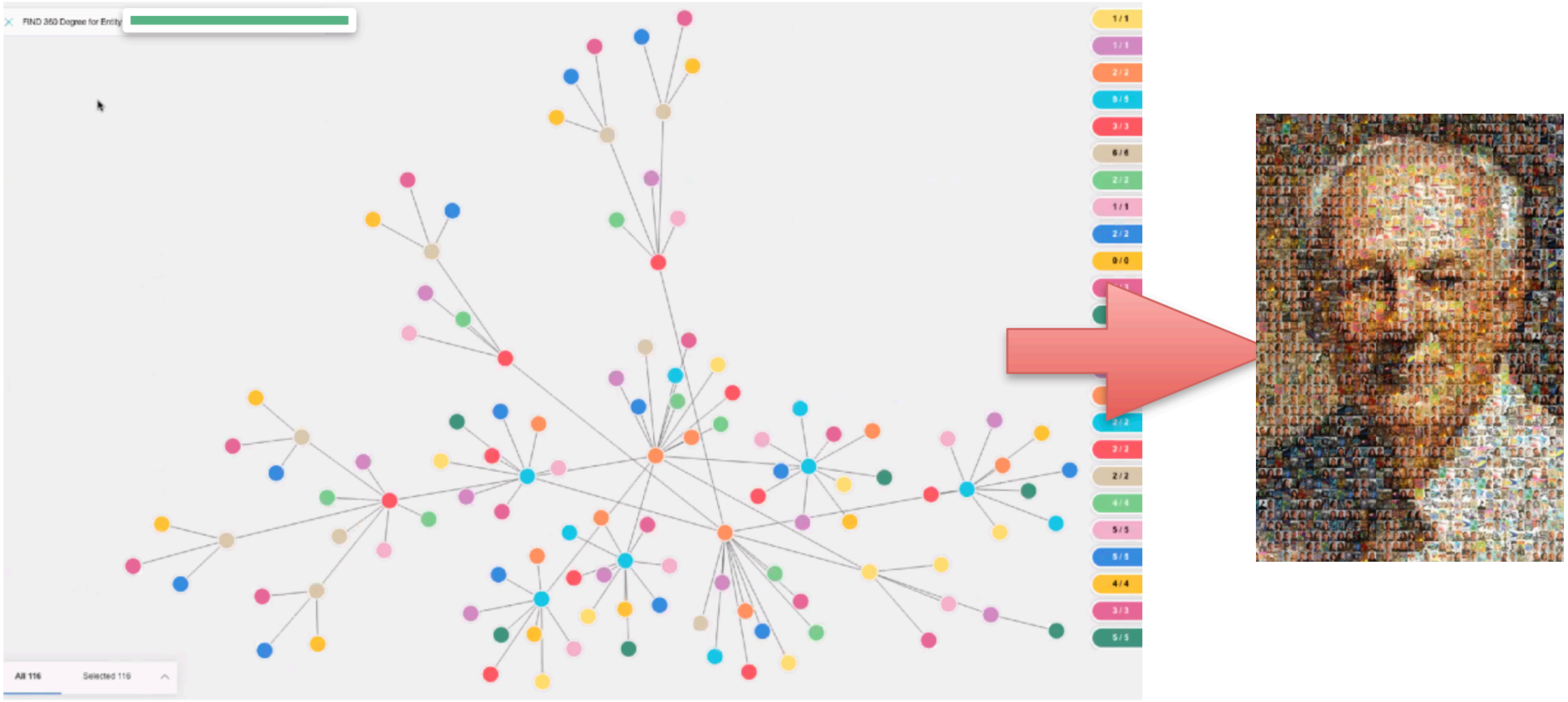
Relational

Document

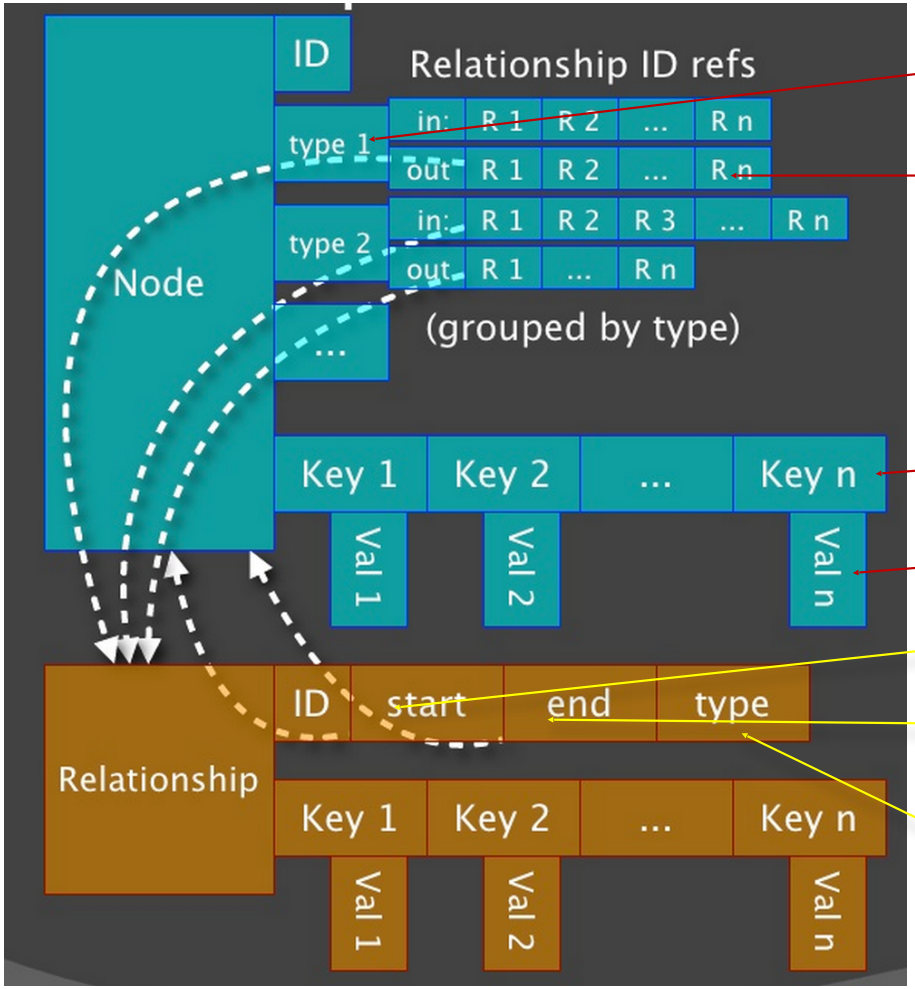
Graph

neo4j

Real-time Joins of Data From Myriad Sources to Provide 360°View



Neo4j Native Graph Data Storage - Why Neo4j is so Fast



Relationship Type (e.g. ACTED_IN)

Relationships (of that type) ….e.g. pointers to all movies acted in. Separate pointers for inbound and outbound to speed traversals

Property Names

Property Values

Starting Node (e.g. Person)

Ending Node (e.g. Movie)

Relationship Type (e.g. ACTED_IN)

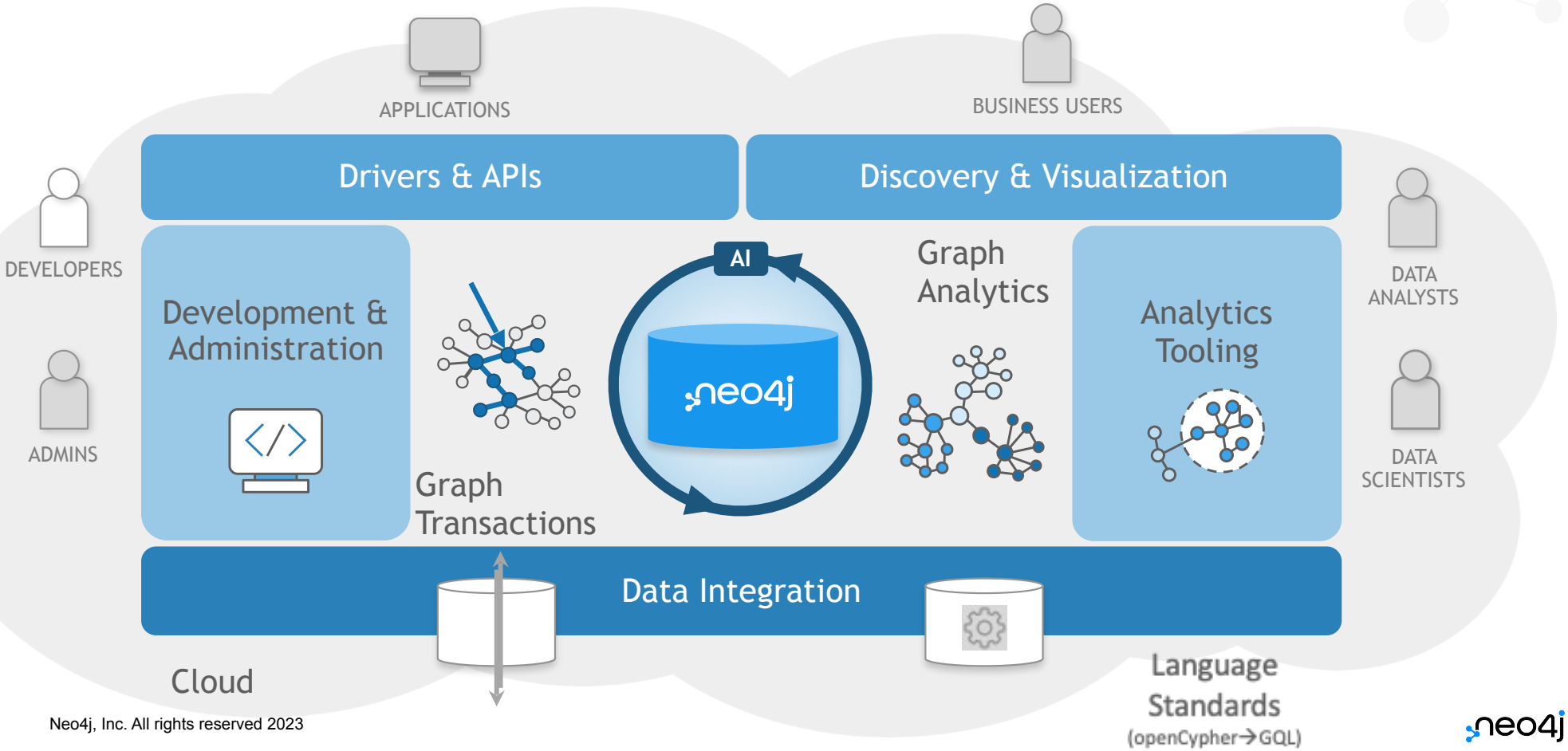
Understanding Latency

System Event	Actual Latency	Relative Latency
One CPU Cycle	0.4 ns	1 second
RAM Access (DDR-DIMM)	~100 ns	4 minutes
SSD i/O	50-150 µs	1.5 - 4 days
Rotational disk I/O	1-10 ms	1 - 10 months
Intra-AZ latency	0.5 ms - 1.0 ms	2 - 4 weeks
Inter-AZ latency	1-10 ms	1 - 10 months
SF to NYC Internet	65 ms	5 years

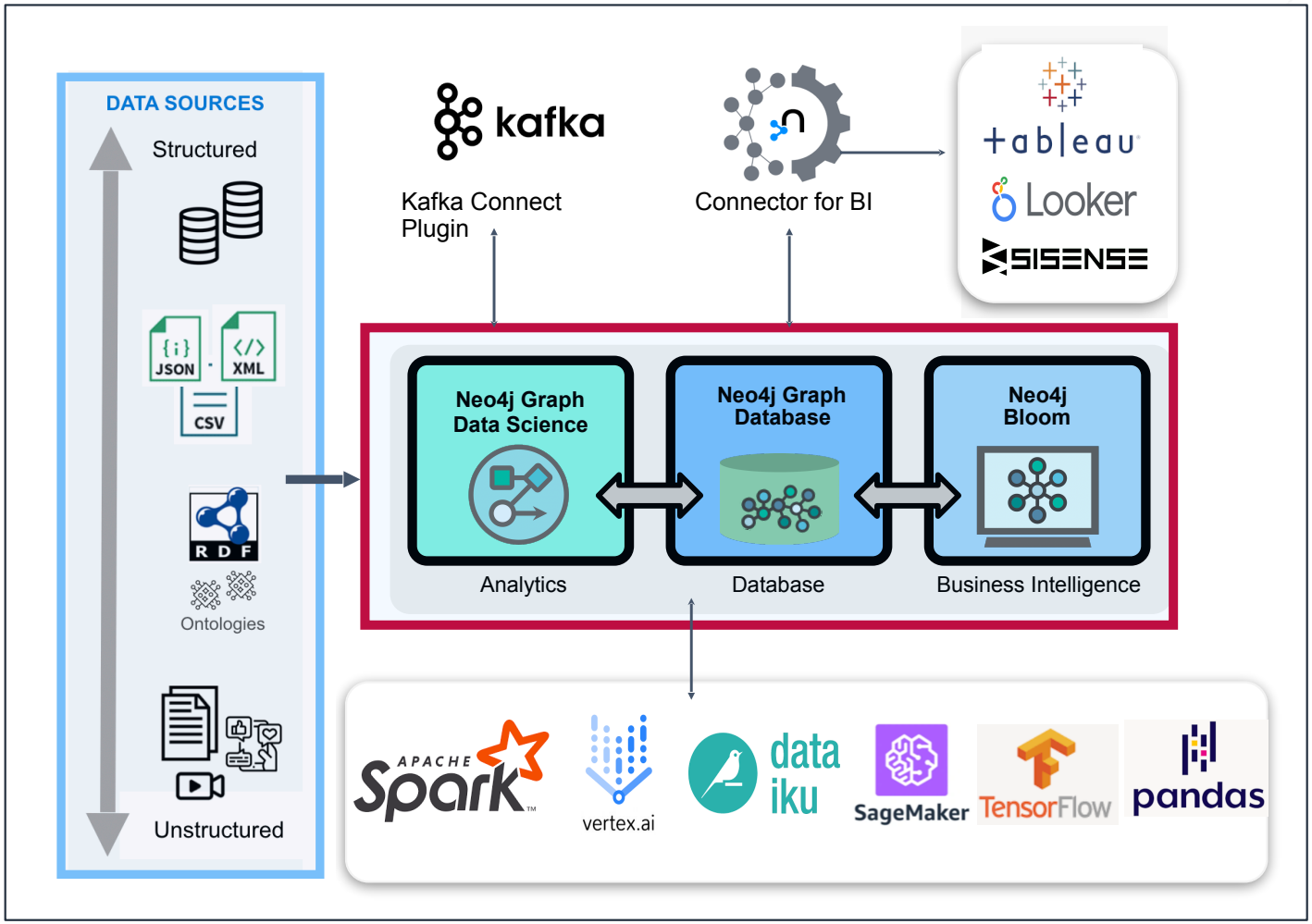
https://www.prowesscorp.com/computer-latency-at-a-human-scale/

1 Relative Latency Using One CPU Cycle = One Second

Neo4j Graph Platform



Neo4j Data Science Ecosystem



neo4j

Neo4j in the Cloud

neo4j aura

Fully-managed Cloud Service

FAST

FLEXIBLE

RELIABLE

EASY

Automated Upgrades, Maintenance

Scalable and Elastic, On-Demand

Enterprise-Grade Security

High Availability

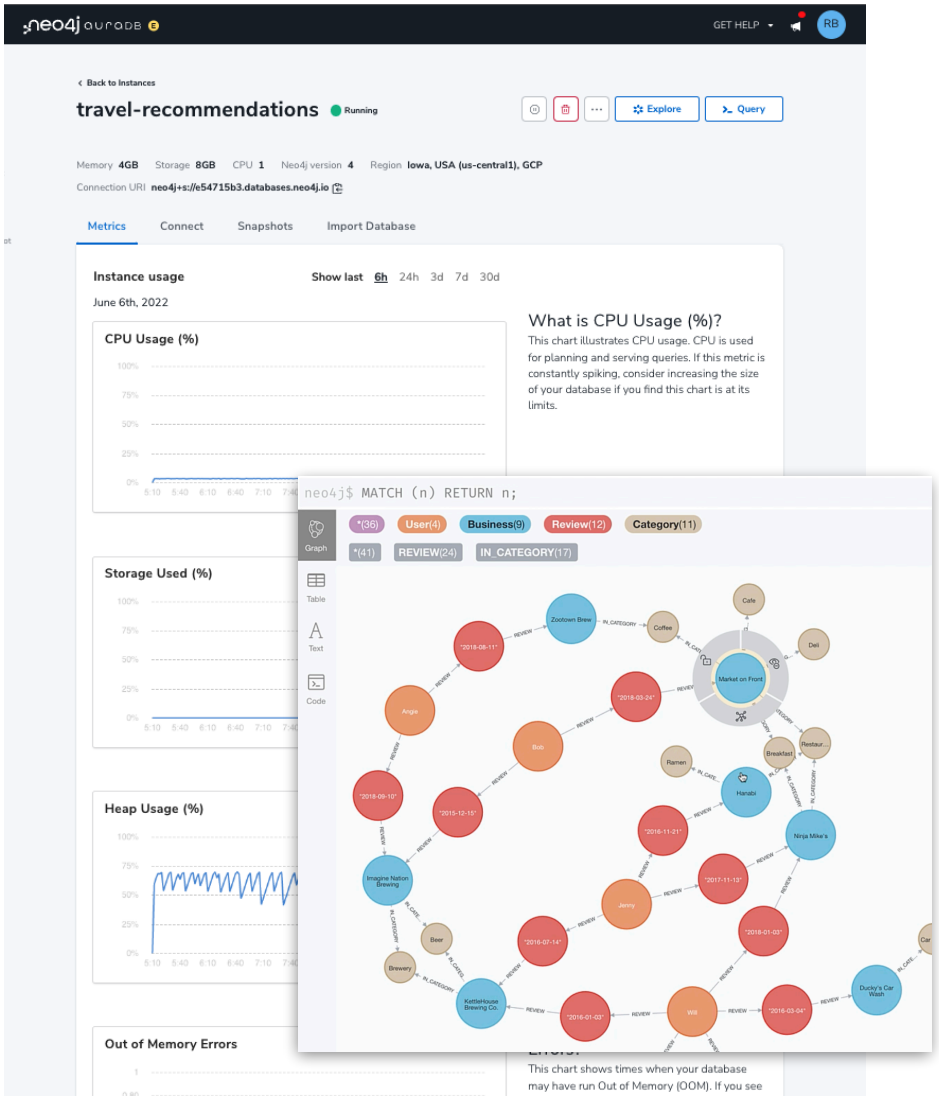
Simple Pricing, Consumption-Based

Procure Direct or via Cloud Marketplace

Google Cloud Platform

aws

Microsoft Azure



AuraDB Free

For small development projects, learning, experimentation and prototyping.

- AuraDB is Neo4j's fully managed cloud service
  - Zero-administration
  - Always-on graph database
- It's available today for free, forever
  - limited to 200k nodes and 400k relationships
  - Includes Bloom and APOC
- No credit card is required to register

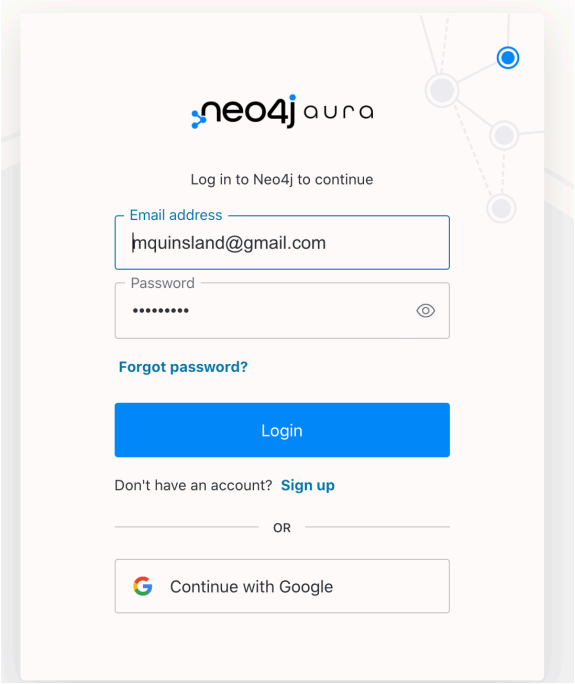
neo4j.com/cloud/aura/

Lab 0 - Connecting to Aura

Lab 0.0 - Creating a New Database in Aura

Login - or create a free account

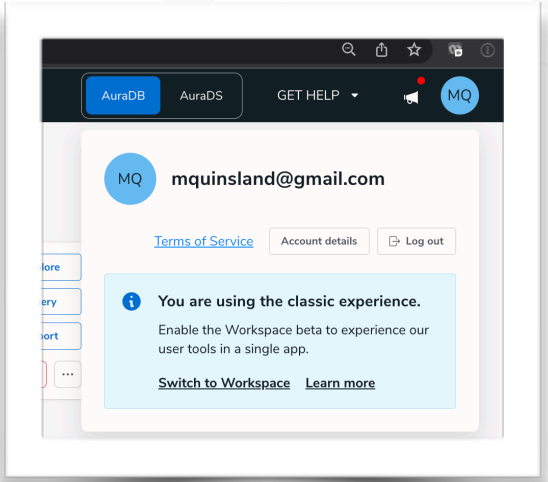
- Proceed to https://login.neo4j.com
- Enter username and password
- Create a new account if necessary



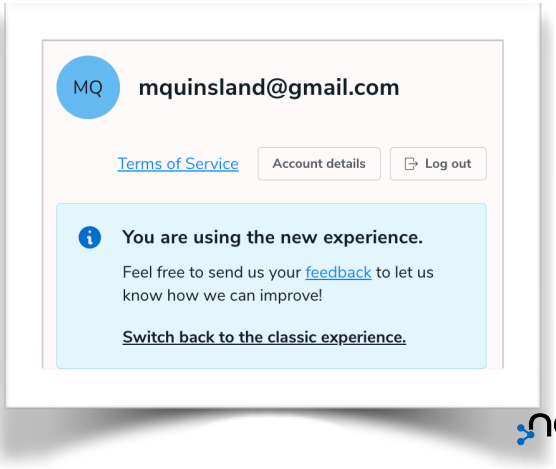
Lab 0.1 - Creating a New Database in Aura

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

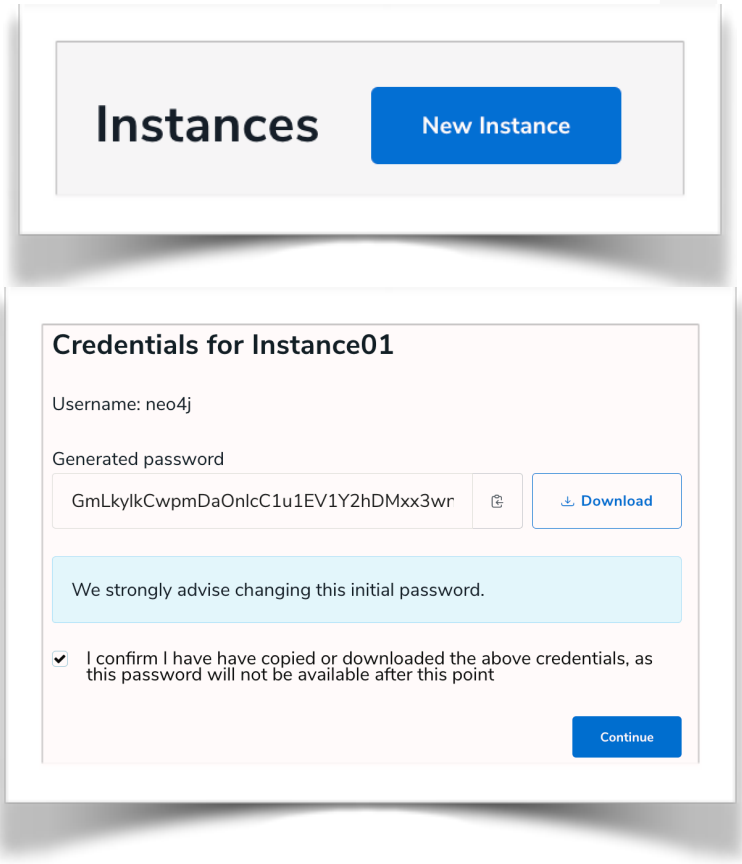


Lab 0.2 - Creating a New Database in Aura

Create a new Instance

- Click on the New Instance Button
- Select the Empty instance

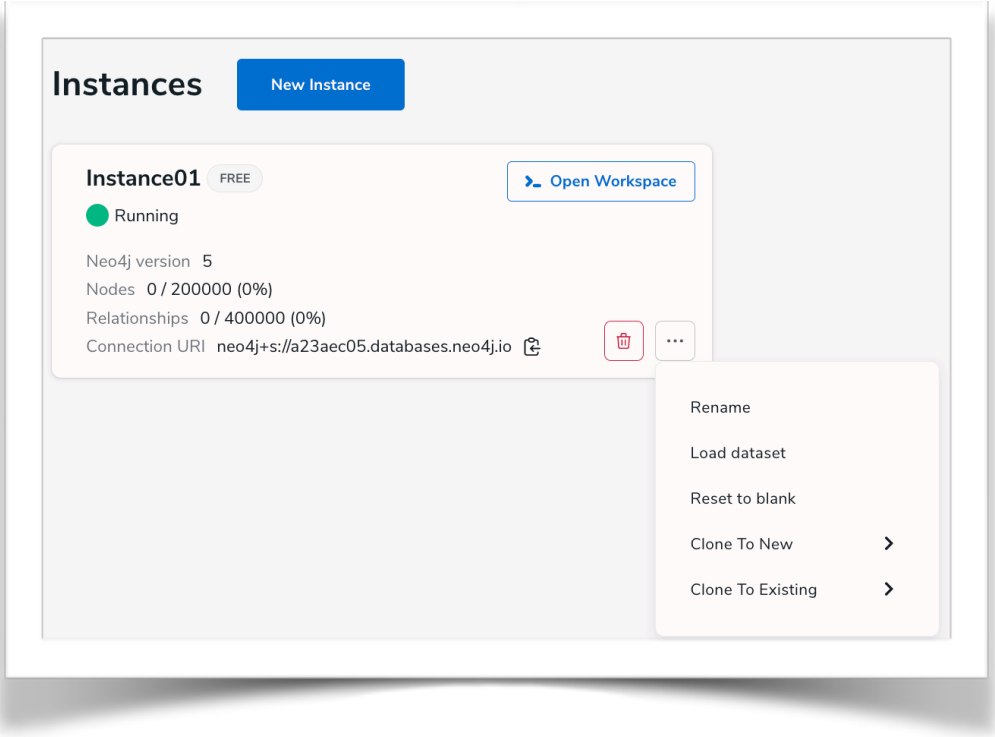
- Your new instance will be named Instance01 and a password will be generated. Copy the password to the clipboard and download the password file
- Please remember where you downloaded the password file!



Lab 0.3 - Creating a New Database in Aura

Rename the new Instance

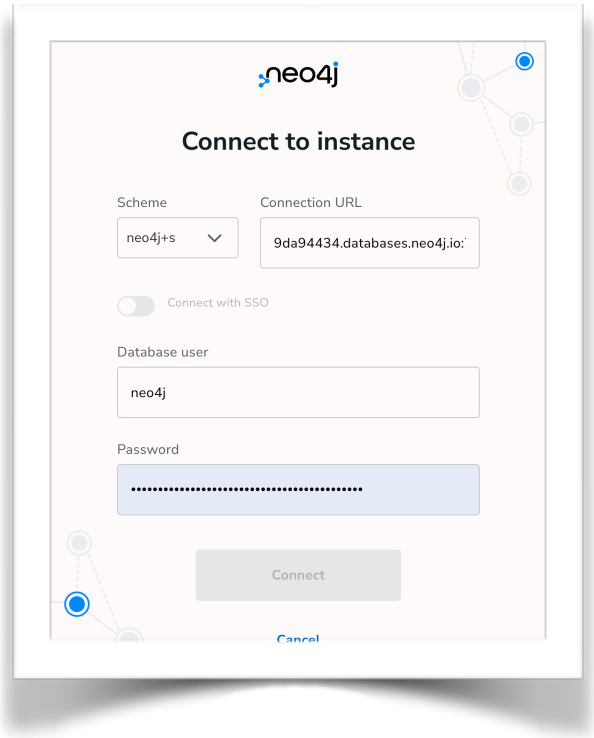
- In a moment or two, the instance will be created and the status will change to ‘Running’
- Change the name to something useful by clicking on the … button next to the trash can.
- Some basic statistics will be shown
- The instance will be free to use for its lifetime. After periods of inactivity it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



Lab 0.4 - Creating a New Database in Aura

Testing the Connection

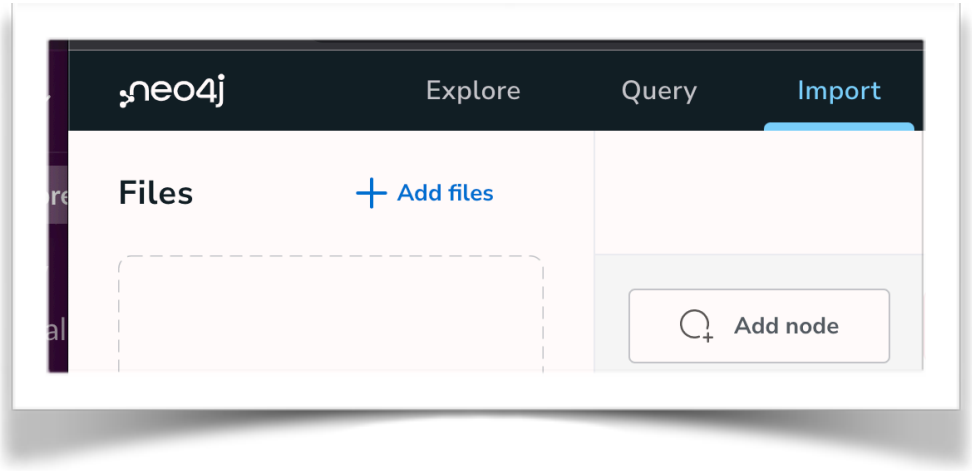
- Click on the Open Workspace button to display the Connect to Instance dialog.
- Paste your password into the appropriate field.
- If you forgot your password and failed to download it, you must delete the instance and start over.



Lab 0.5 - Creating a New Database in Aura

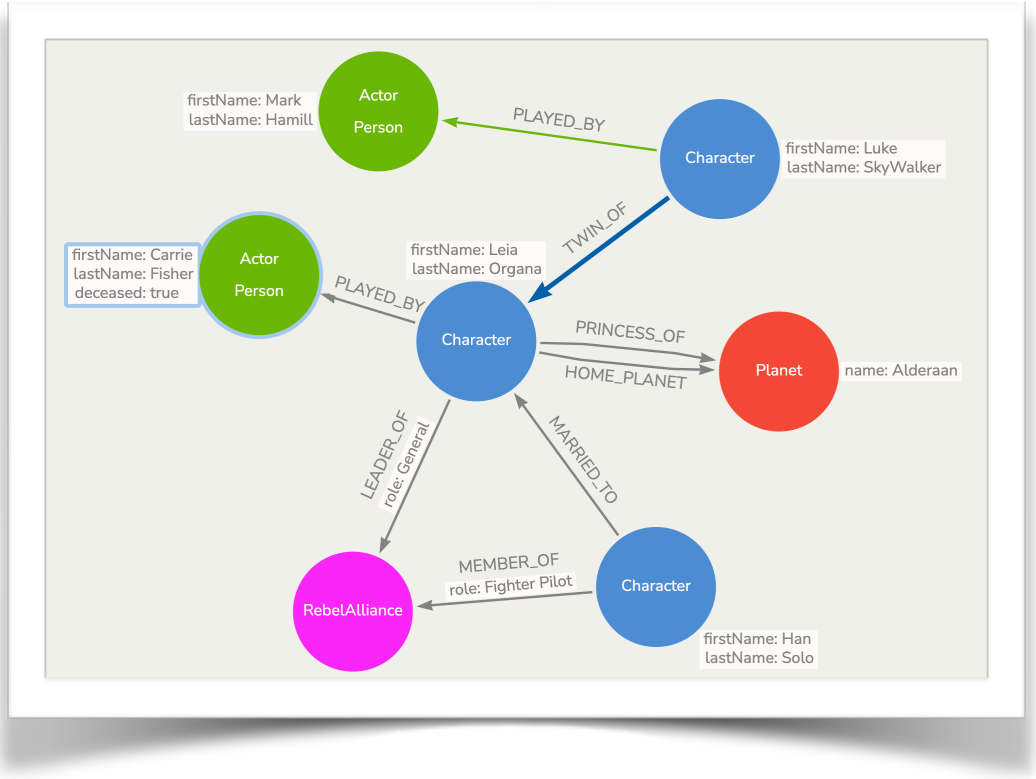
Viewing the Workspace

- 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
- But first, let's create some data and a data model!



Neo4j Data Modeling

Visually Create Data Models with Arrows

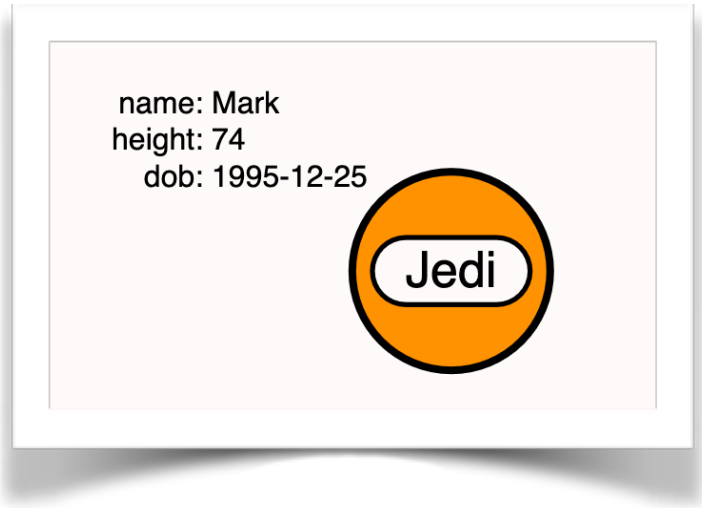


- Create Graphs visually using sample data
- Export to Cypher
- Import / Export models
- Simple to learn
- https://arrows.app

Lab 1 - Data Modeling with Arrows

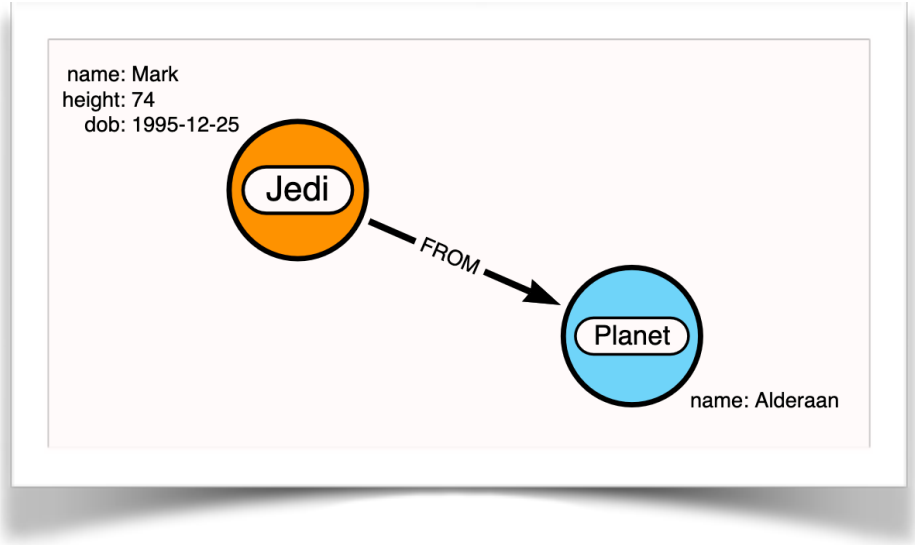
Lab 1.1 - Creating a New Node

- 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 - Creating a Related Node

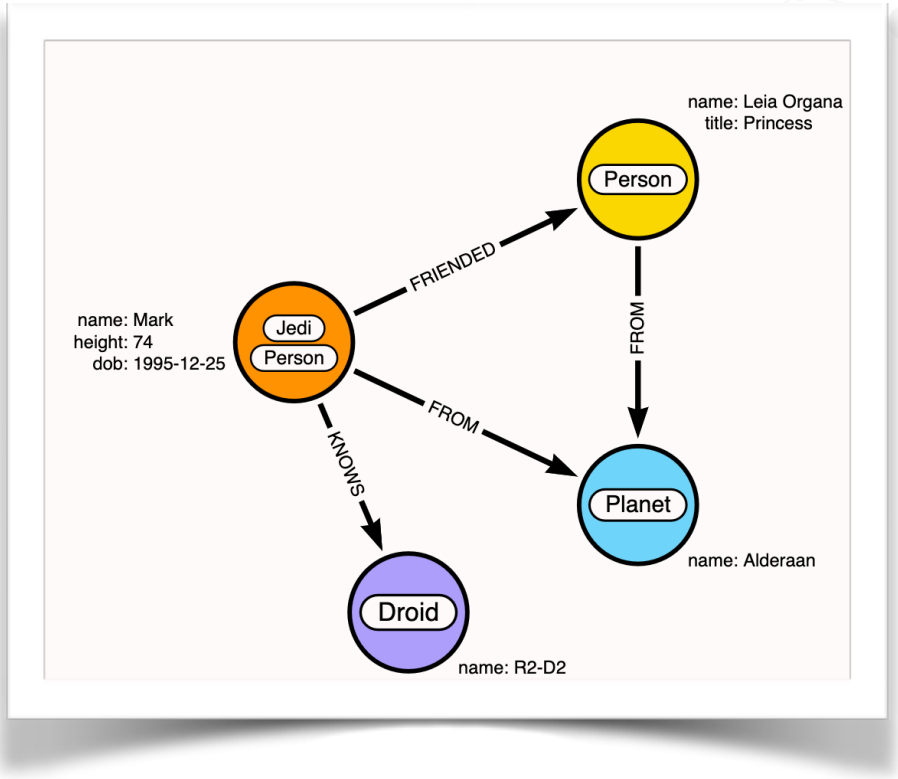
1. Hover over the border of the source node until the border shows that it is selected
2. Click on the selection border and drag it away from the node. You will see a new node appear.
3. Drop the new node away from other nodes
4. On the properties tab, give the new node a label (not a caption)
5. Give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



neo4j

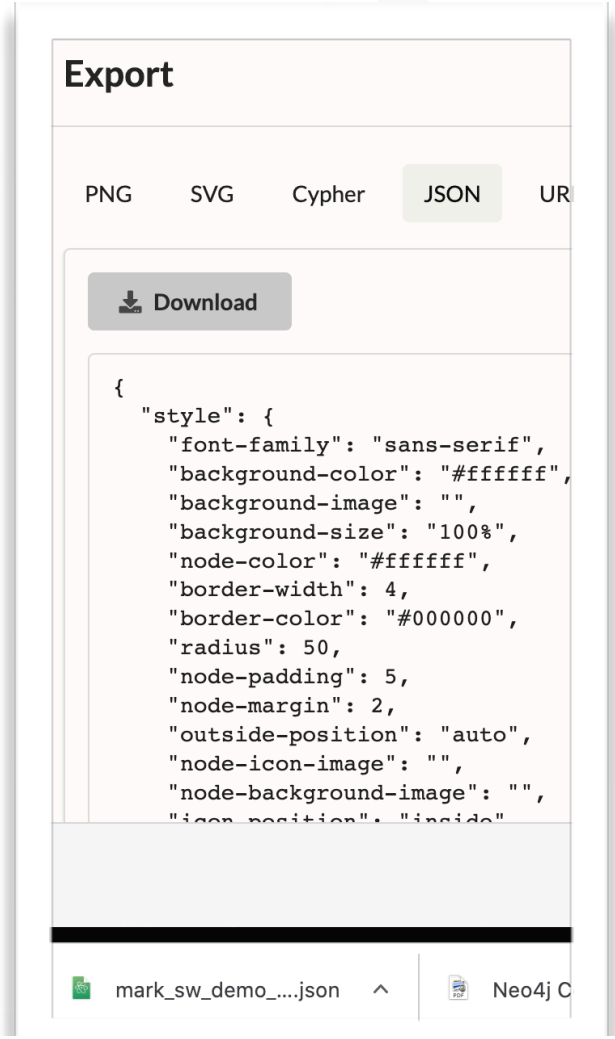
Lab 1.3 - Creating Additional Related Nodes

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



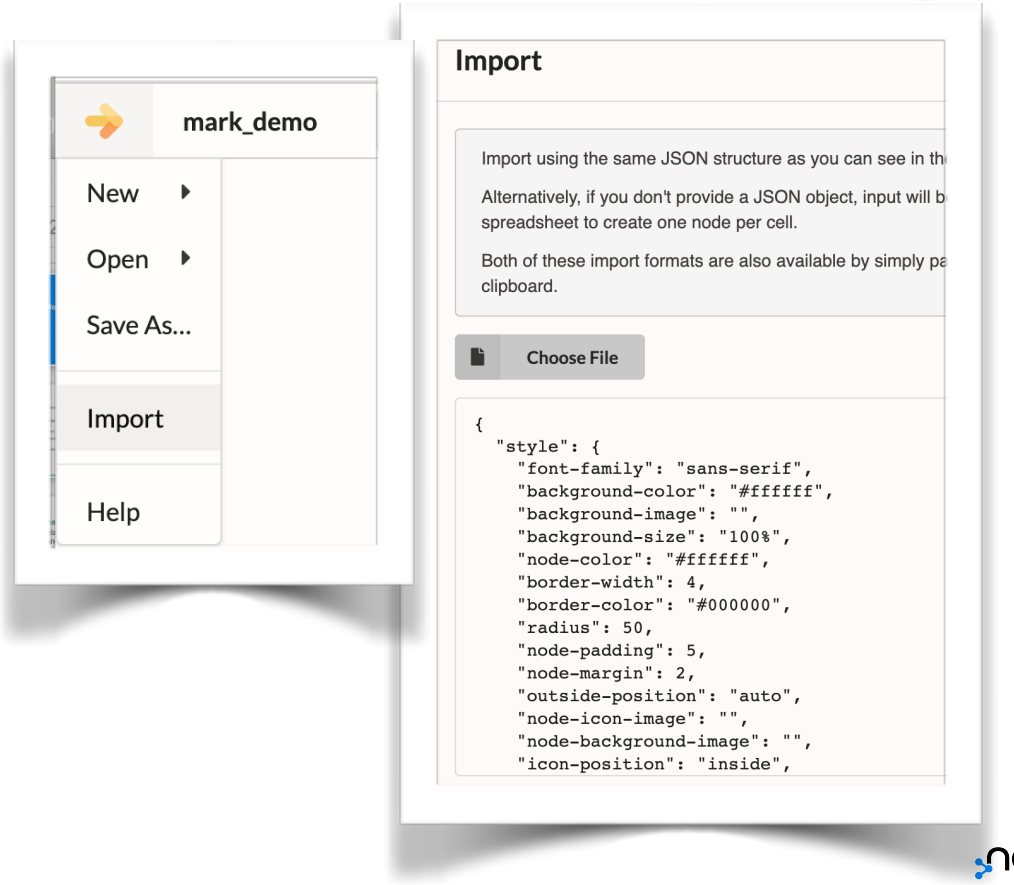
Lab 1.4 - Exporting a Model to JSON

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.



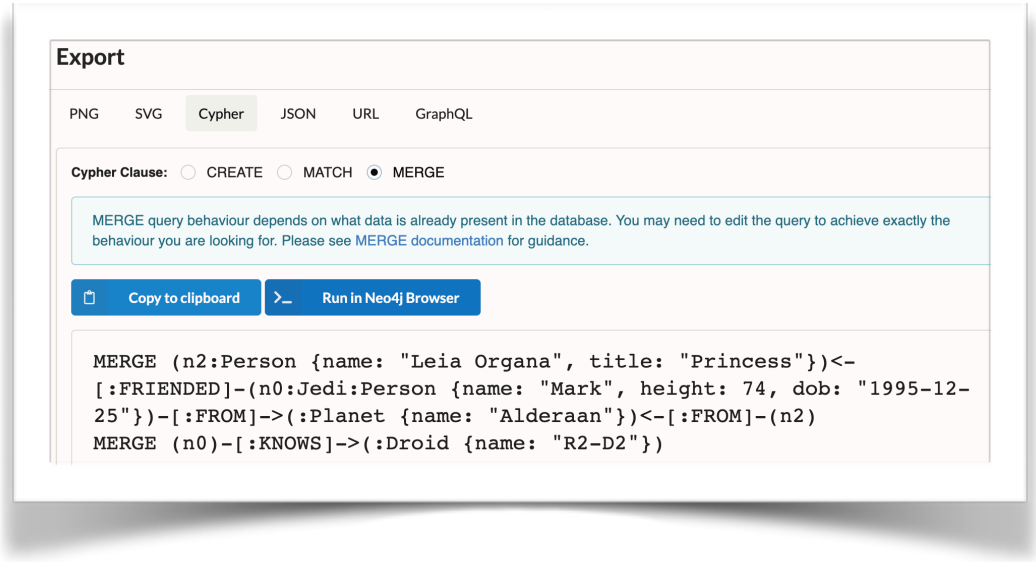
Lab 1.5 - Import a Previously Exported Model

1. Click on the yellow arrow at the top left corner
2. Select the Import item
3. Choose the JSON file to be uploaded. Use the json file “starwars_intro_model.json” from the downloaded zip file.
4. View the new nodes / relationships.



Lab 1.6 - Exporting A Model to Cypher

1. Click on the Download/Export button
2. Select the Cypher tab
3. Select the CREATE option
4. Review the Cypher code
5. Select the MERGE option
6. Compare this Cypher with the previous



Lab 1.7 - Importing the data into Neo4j using the browser

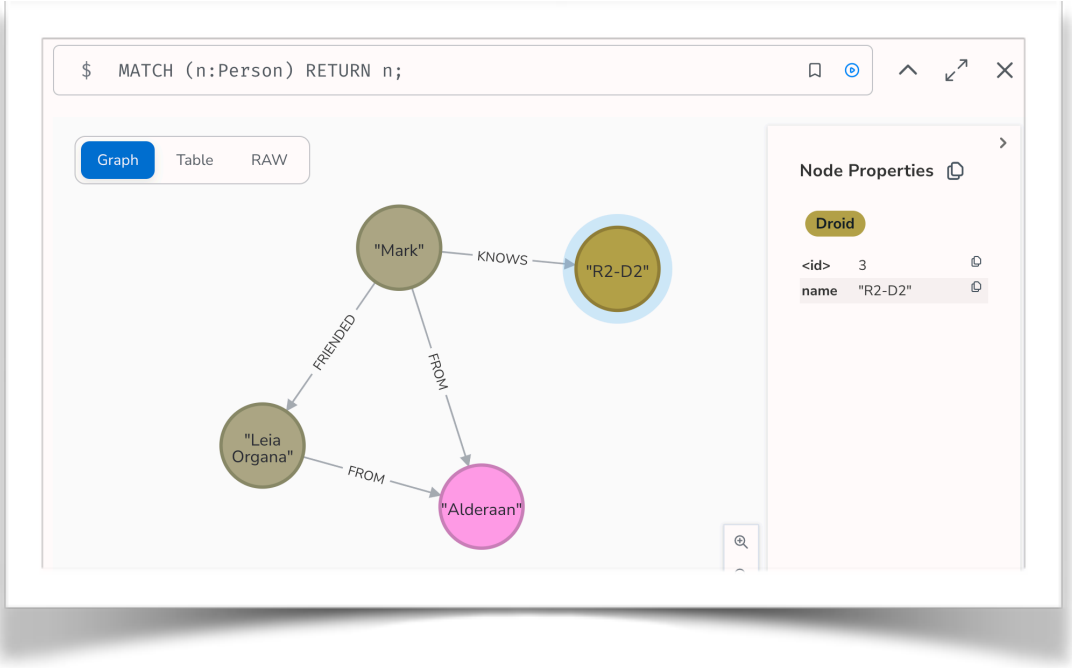
1. Copy the Cypher text that was generated by the Export window
2. Switch to your browser and bring up the Aura Developer Workbench tab
3. Paste the code into the execution window
4. Click on the small blue arrow to execute the command

```
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 1.8 - Viewing the data in a Neo4j Browser

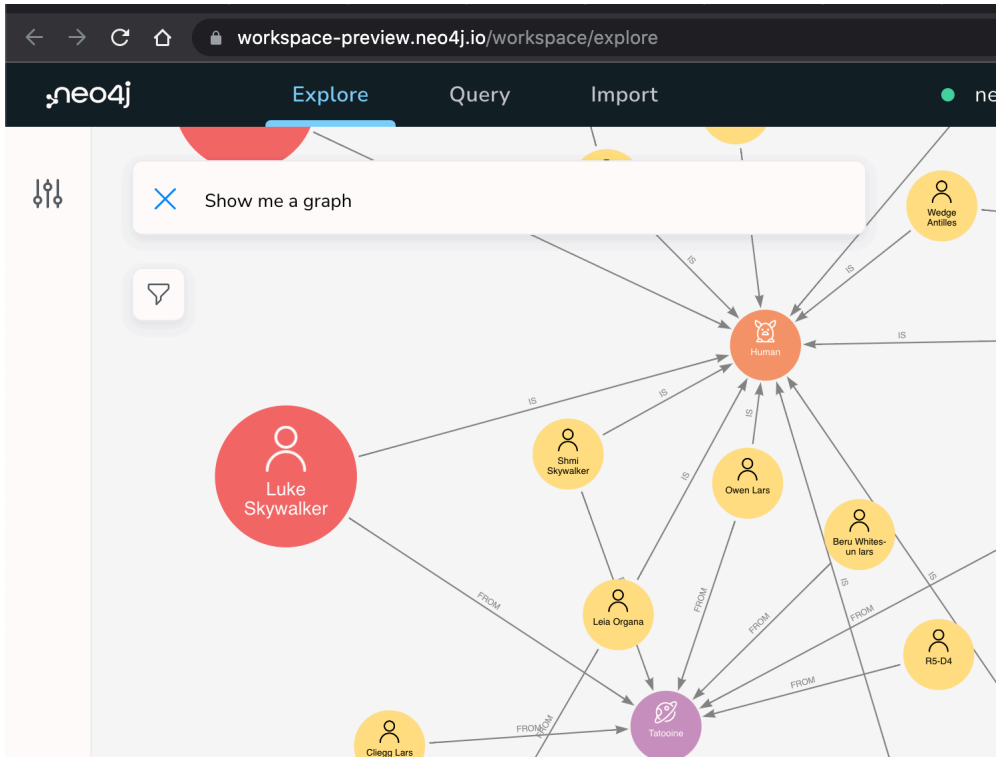
1. Run a simple Cypher command to view your data
2. Double-click on the nodes to see related nodes appear.

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



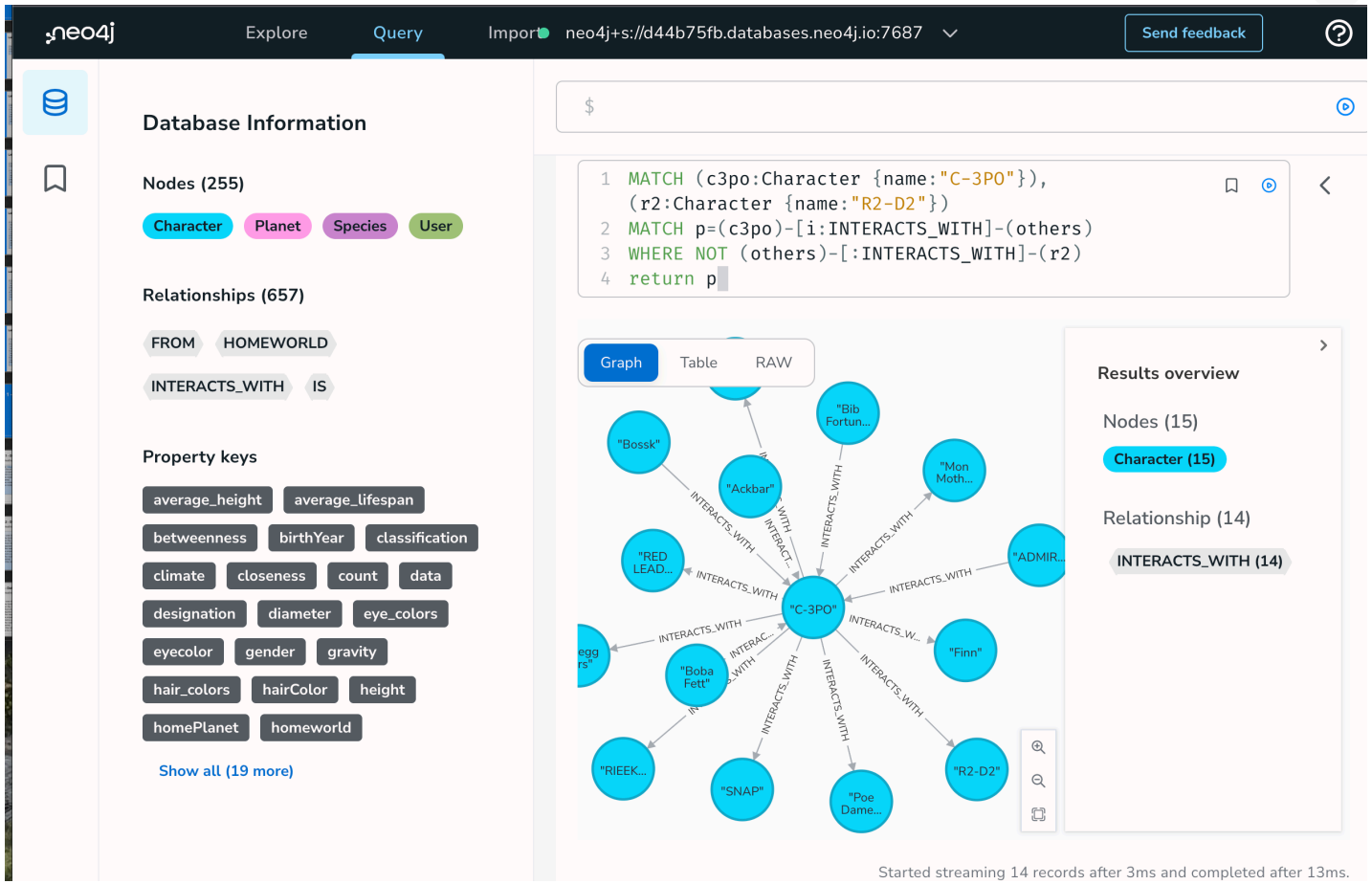
Neo4j Workspace

Explore, Query, Import with Neo4j Workspace

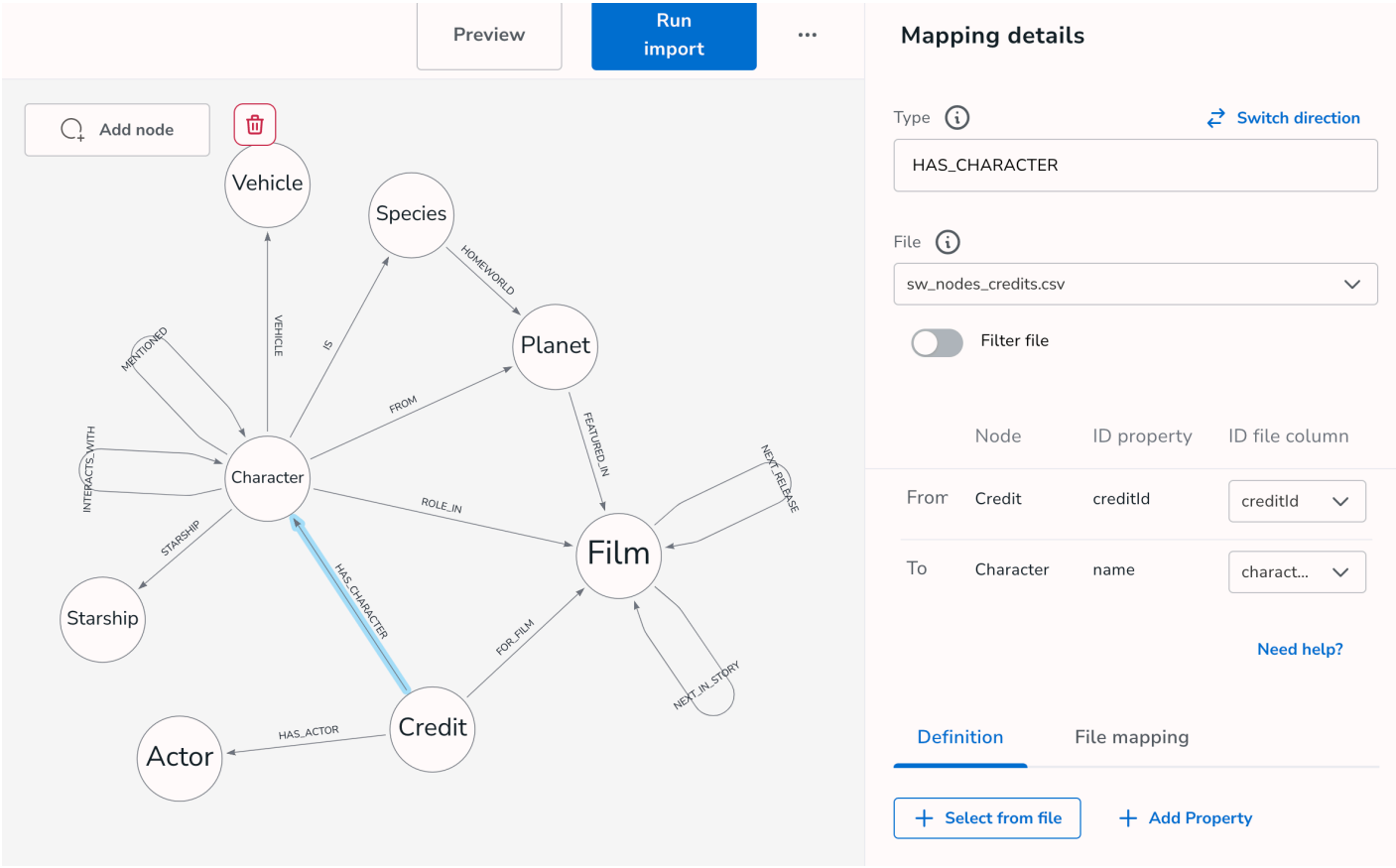


- Easy-to-use work surface (beta)
- Import CSV files
- Edit/Run Cypher Commands
- Visually explore the graph without using Cypher

Run Cypher Commands -Workspace - Query



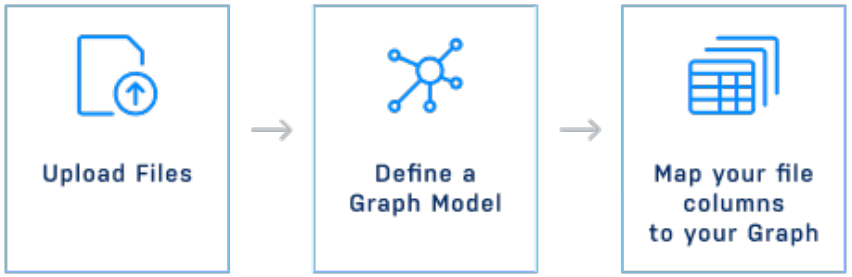
Import CSV files -Workspace - Importer



Neo4j Data Importer

Small no-code UI for loading flat file data

- Beta on AuraDB
- Not designed to replace ETL tools or production imports
- Support for small flat-file inputs (CSV, TSV)
- The ability to sketch out a graph model and map your input data
- Loads into any Neo4j database reachable from your machine
- Data is only kept in your web browser; the tool doesn't use a server-side component
- Self-hosted may use this website - https://data-importer.graphapp.io/

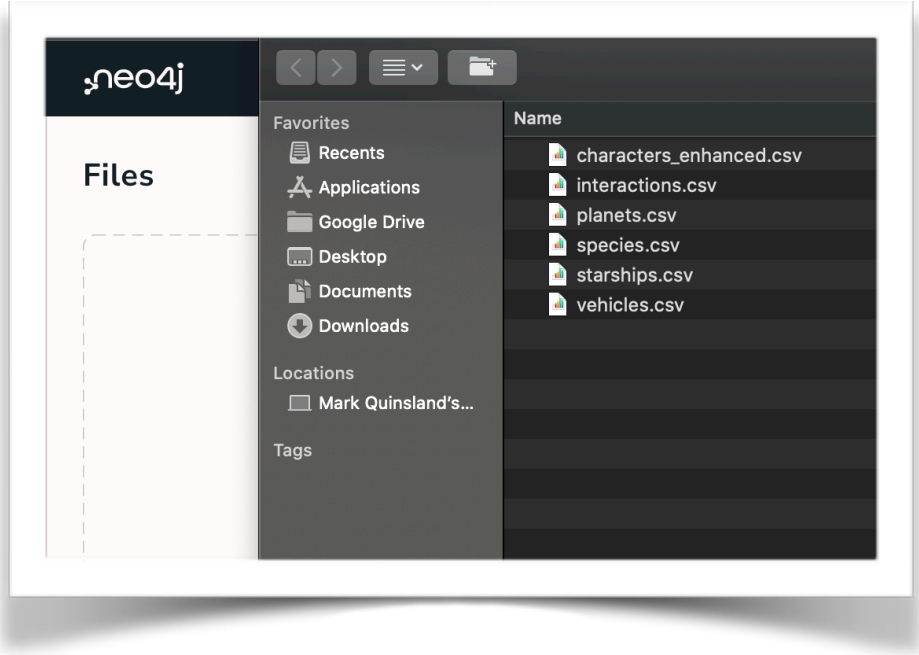


Lab 2 - Using Workspace Importer

Lab 2.1 - Using Workspace Importer

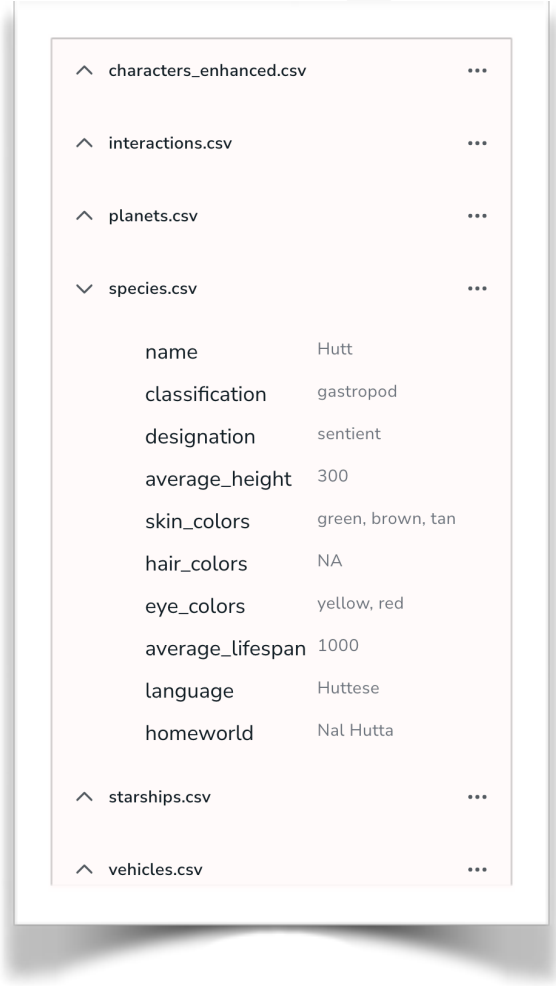
Selecting CSV files to Import

- Download the data files from
- https://github.com/mquinz/star_wars_demo/blob/master/data/star_wars_data.zip
- Unzip the files
- Click on the + Add files link on the Workspace Import tab or drag-and-drop the files onto the files tab.



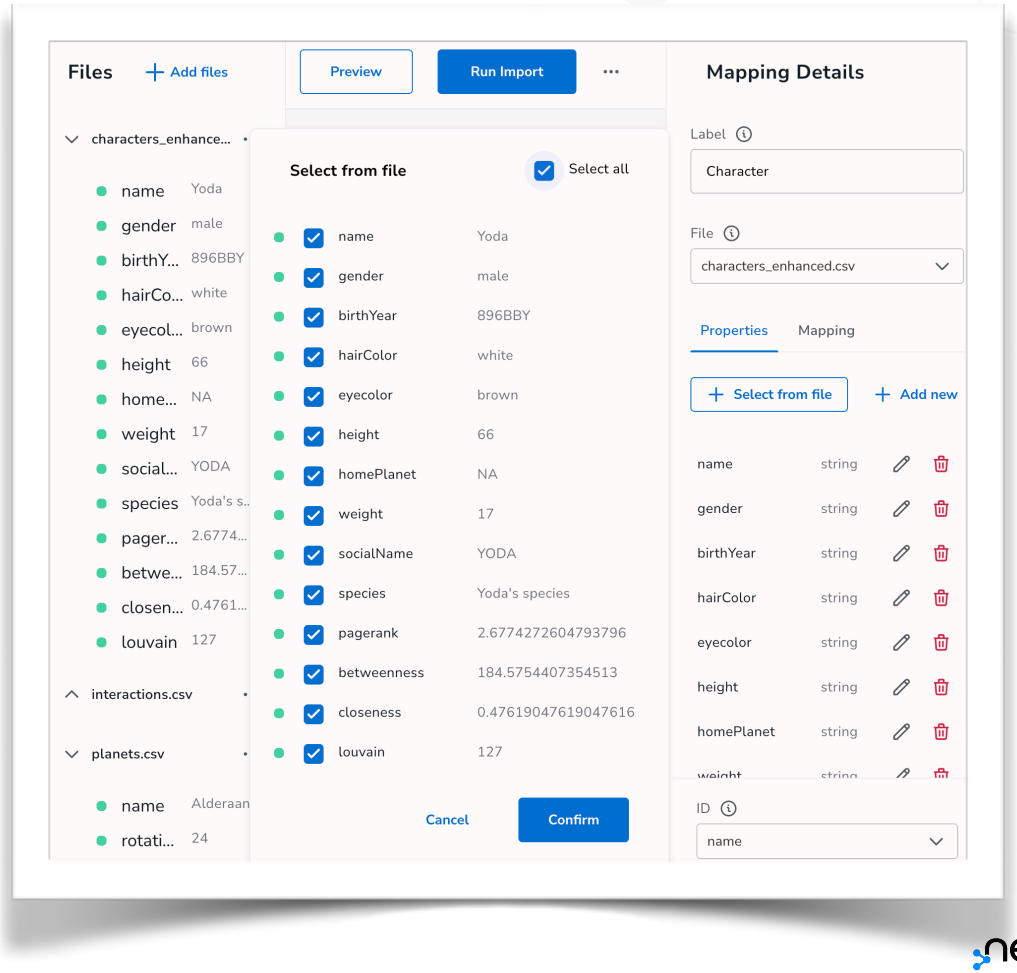
Lab 2.2 - Previewing Data from CSV Files

- 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.



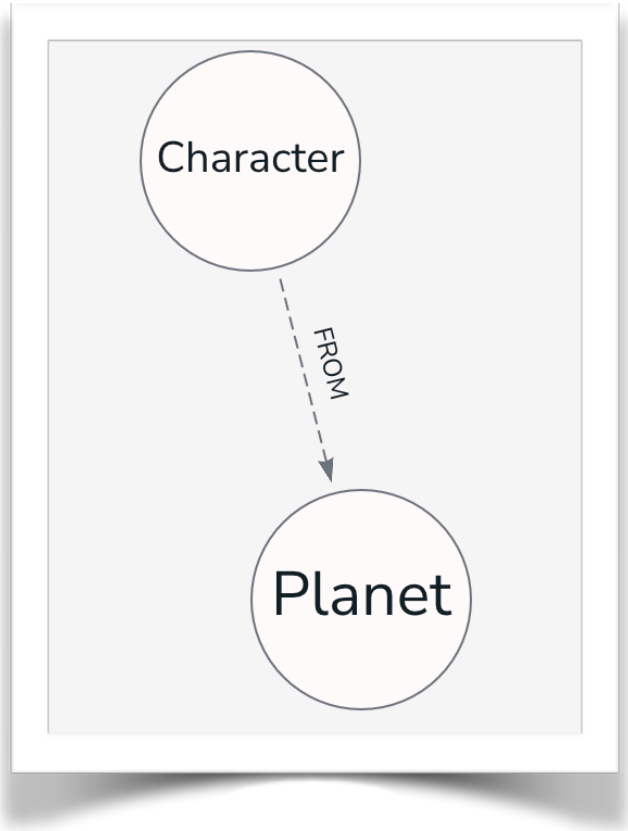
Lab 2.3 - Adding Nodes from CSV File

- 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



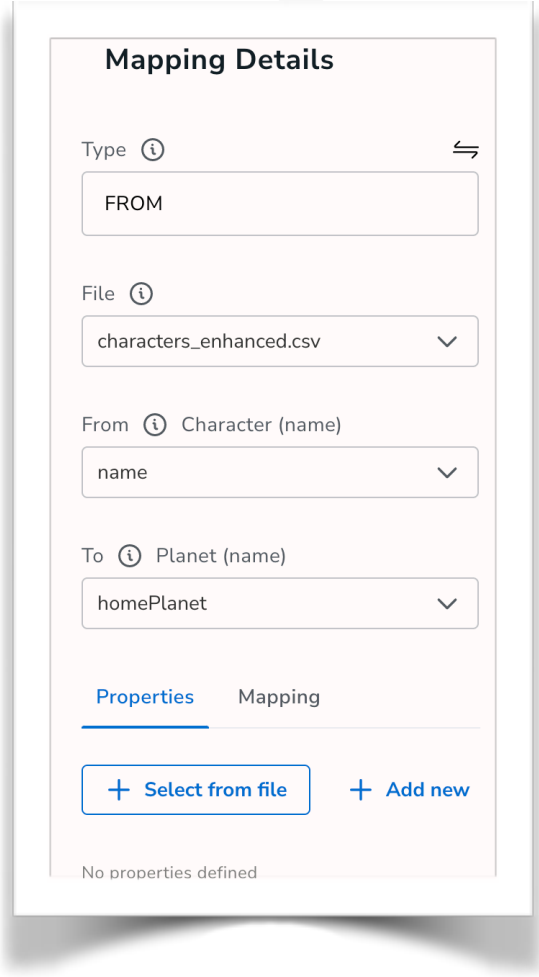
Lab 2.4 - Creating Relationships Between Nodes

- 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



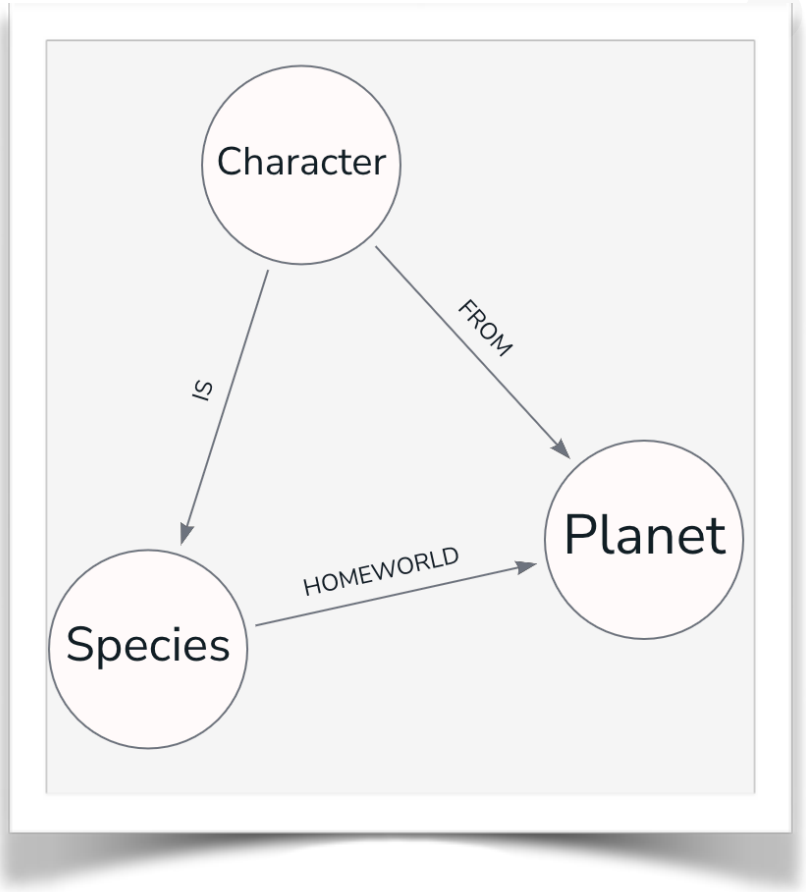
Lab 2.5 - Configuring New Relationships

- 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.



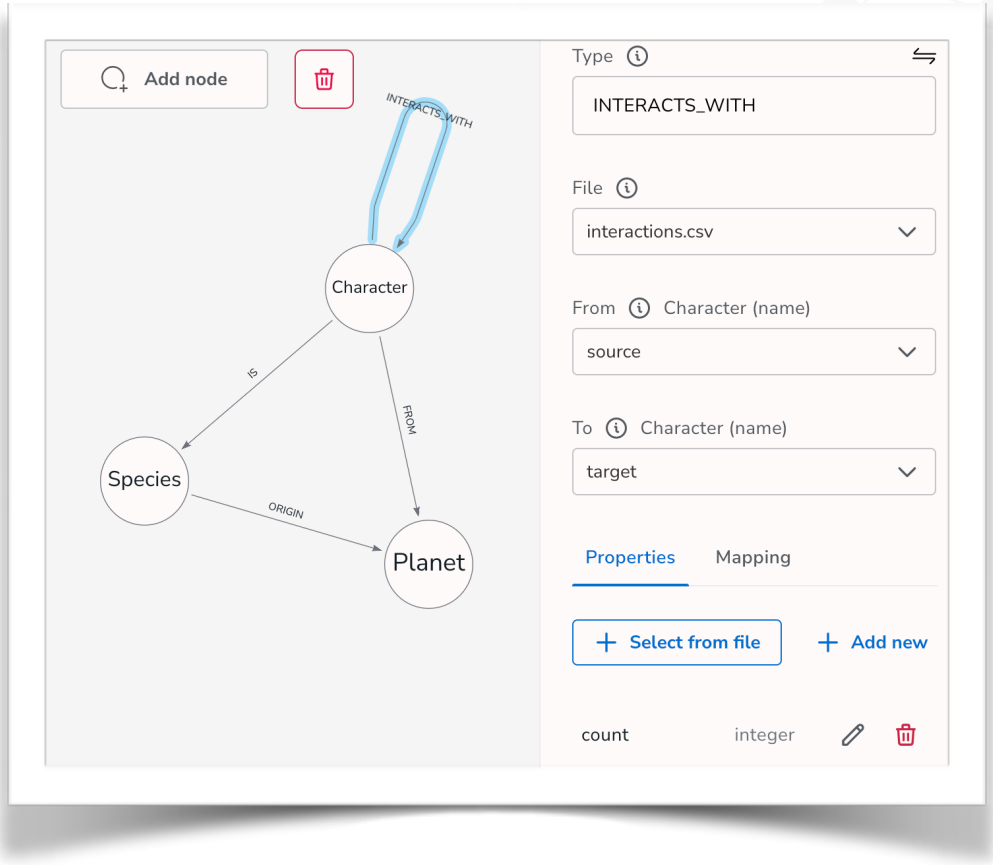
Lab 2.6 - Load Species Nodes and Relationships

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



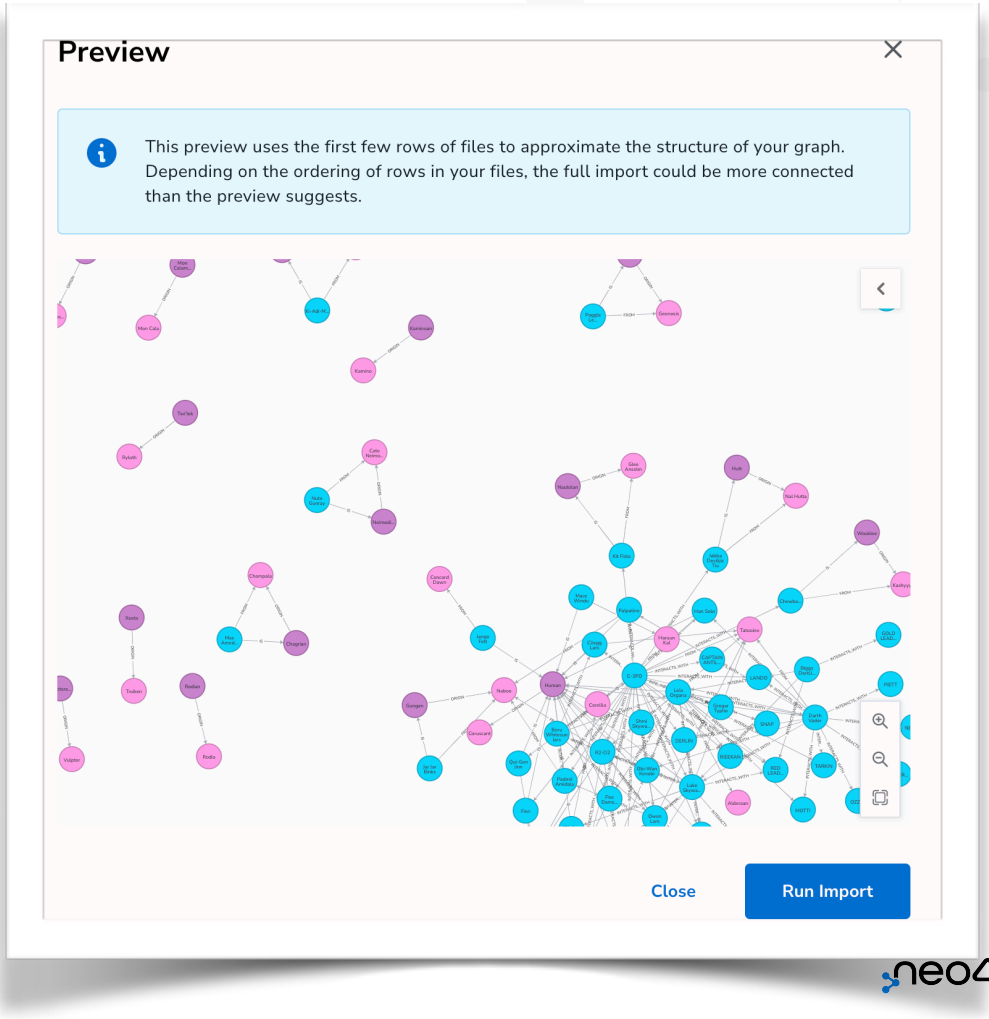
Lab 2.7 - Creating Self-Join Relationships for Characters

- 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



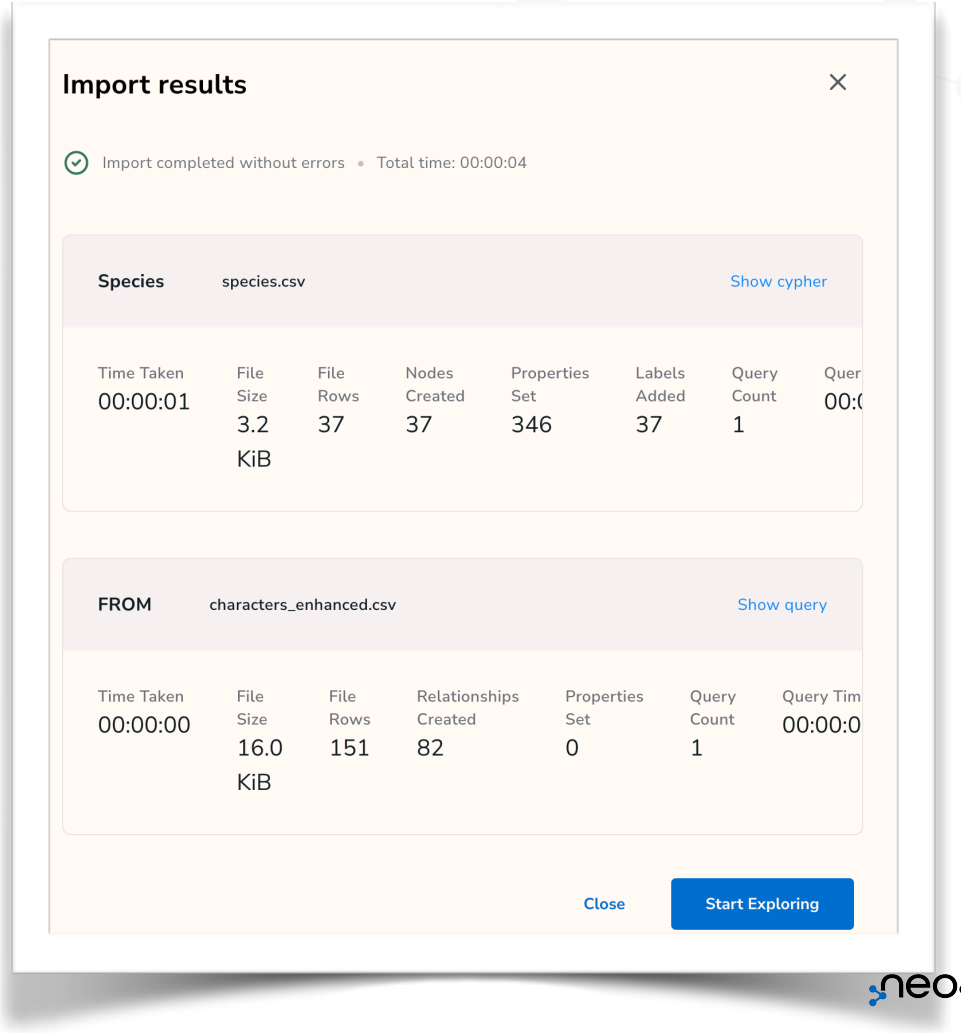
Lab 2.8 - Preview the Import

- 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.



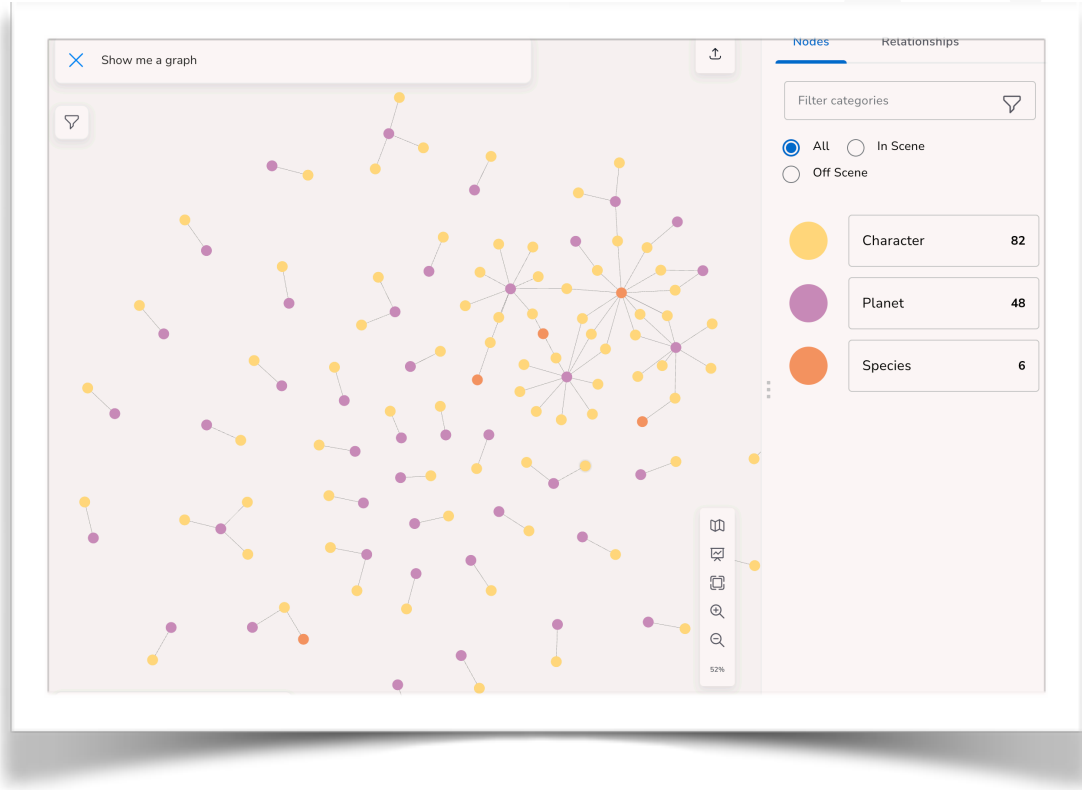
Lab 2.9 - Run the Import and Review Results

- 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.



Lab 2.9.1 - Explore your Graph in Bloom

- 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

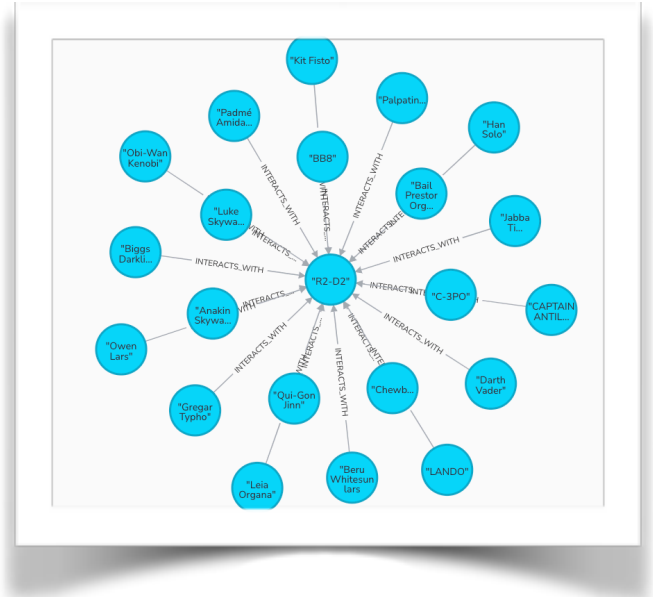
```
// Return all Characters from Alderaan
MATCH (c:Character)-[:FROM]->(p:Planet)
WHERE p.name = "Alderaan"
RETURN c
```

```
// 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

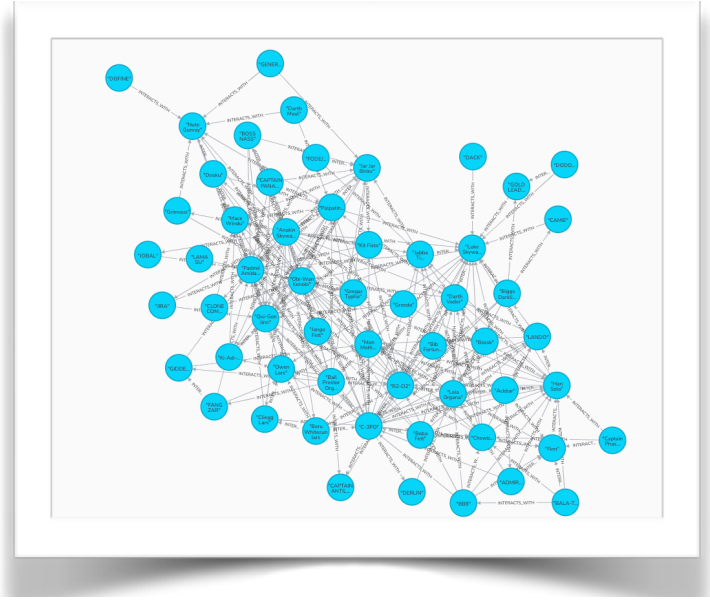
```
// Return all Characters that interacted with R2-D2
MATCH (c:Character)<-[INTERACTS_WITH]-(other:Character)
WHERE c.name = "R2-D2"
RETURN other
```



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
```

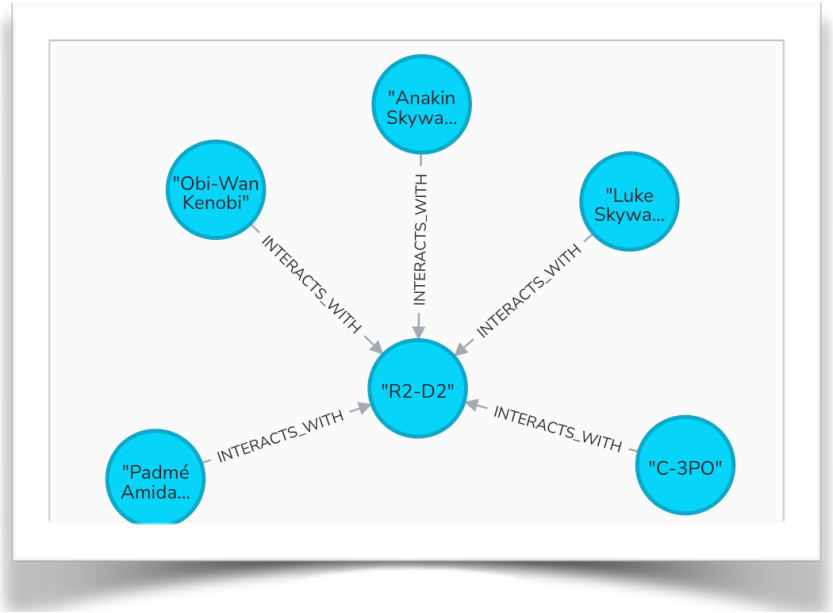


neo4j

Lab 3.6 - Basic Queries - The Match Statement

Relationship Properties Can Also be Used

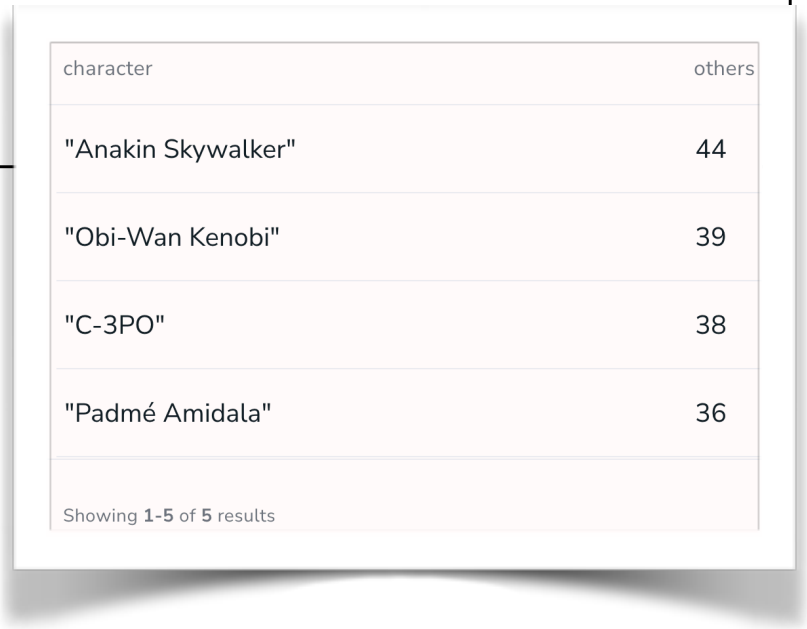
```
// Who does R2-D2 interact with the most
MATCH p=(c:Character)-[i:INTERACTS_WITH]-(other:Character)
WHERE c.name = "R2-D2"
RETURN p ORDER BY i.count DESC LIMIT 5
```



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
"Padmé Amidala"	36

Showing **1-5** of **5** results

Lab 3.8 - Basic Queries - The Match Statement

String Comparison Used as a Filter

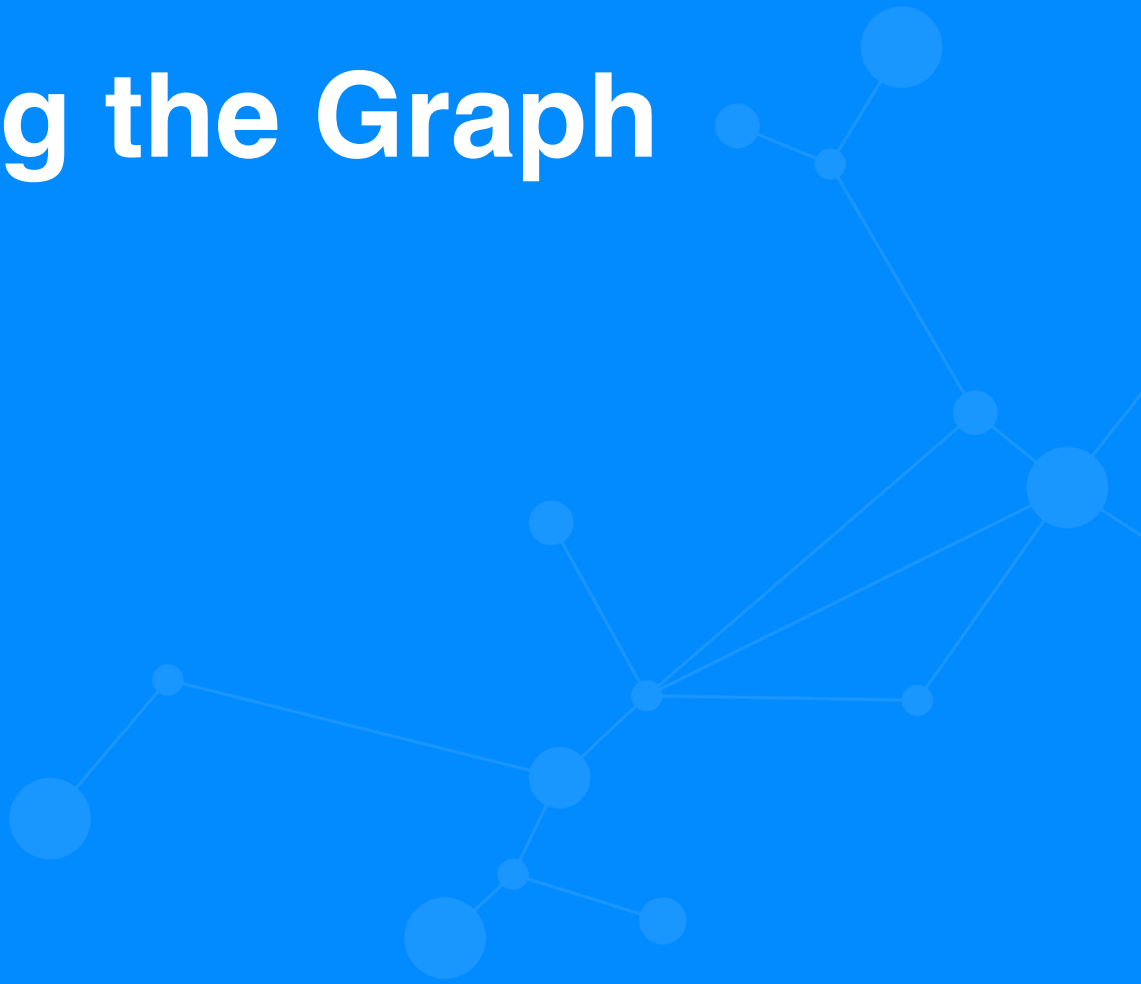
```
// Which characters have Darth as part of their name
MATCH (c:Character)
WHERE c.name STARTS WITH "Darth"
RETURN c
```

Lab 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
```

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

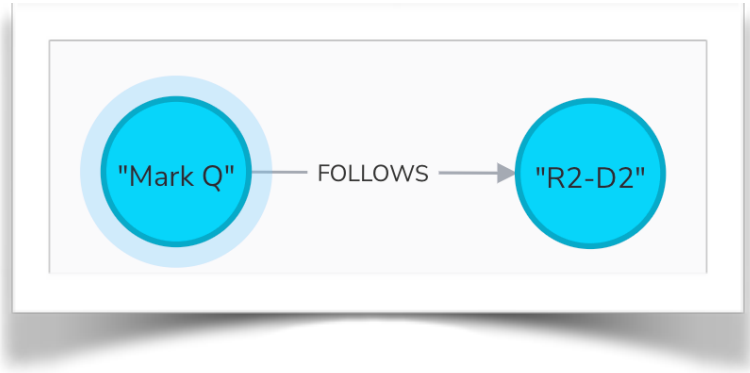
```
// Carefully create a new Character using MERGE
MERGE (c:Character {name:"Mark Q"})
ON CREATE SET c +=
{height:74,hair_colors:"Fleshtone"}
RETURN c
```

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
```



Exercise 4.3.1 - Creating a New Node

Using the MERGE Statement

- Create a New node with a name of your choosing using a MERGE statement.
- Add another property by adding a key/value pairs to the map
- Run the statement again to ensure that the record

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

neo4j

Exercise 4.3.2 - Creating a New Relationship

Using the MERGE Statement

- Create a relationship from your new node to an existing Film

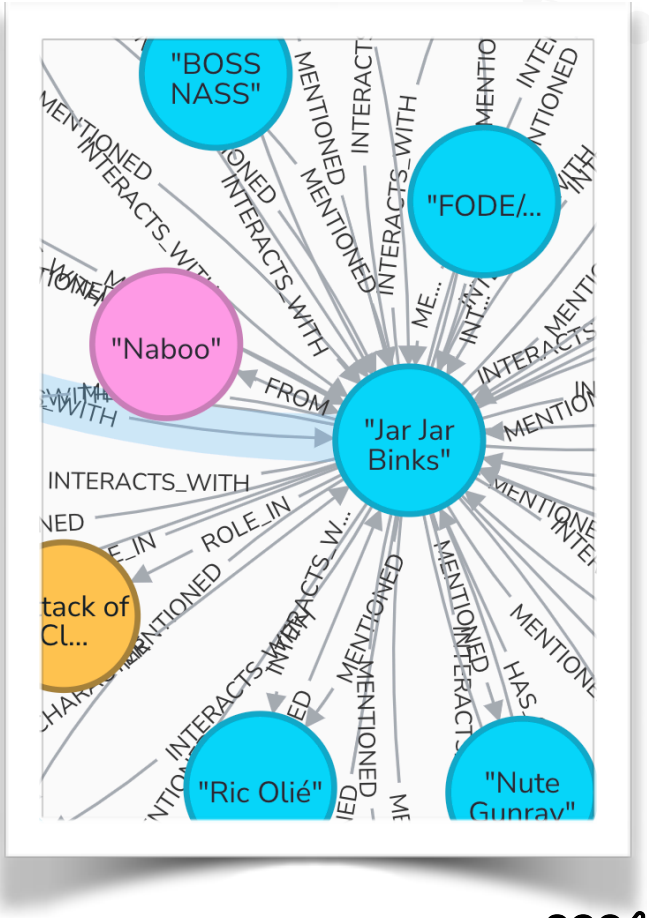
```
// Add your new Jedi to a Film
MATCH (c:Character {name:"Mark Q"}),
MATCH (f:Film {name:"The Empire Strikes Back})
MERGE (c)<-[r:HAS_CHARACTER]-(f)
RETURN c, r, f
```

Lab 4.5 - Update Queries - Deleting a Node

Using the DELETE Statement

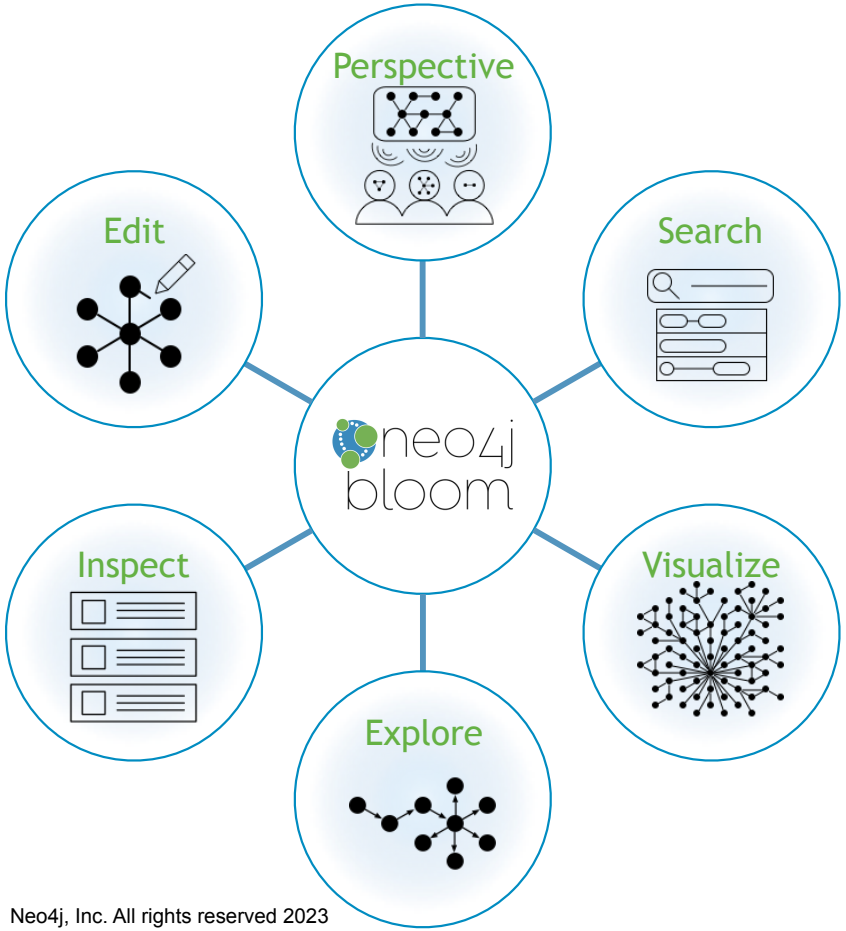
- Nodes may be deleted only if they do not have any relationships.
- DETACH DELETE will automatically remove any relationships so the node may be deleted.
- USE EXTREME CAUTION

```
// Find the Node to be deleted
MATCH (c:Character WHERE c.name = 'Jar Jar Binks')
// Delete the object and all of its relationships
DETACH DELETE c
```



Neo4j Bloom

Explore & Collaborate with Neo4j Bloom



Explore Graphs Visually

Prototype Concepts Faster

Collaborate Across Teams

Neo4j Bloom's Intuitive User Interface

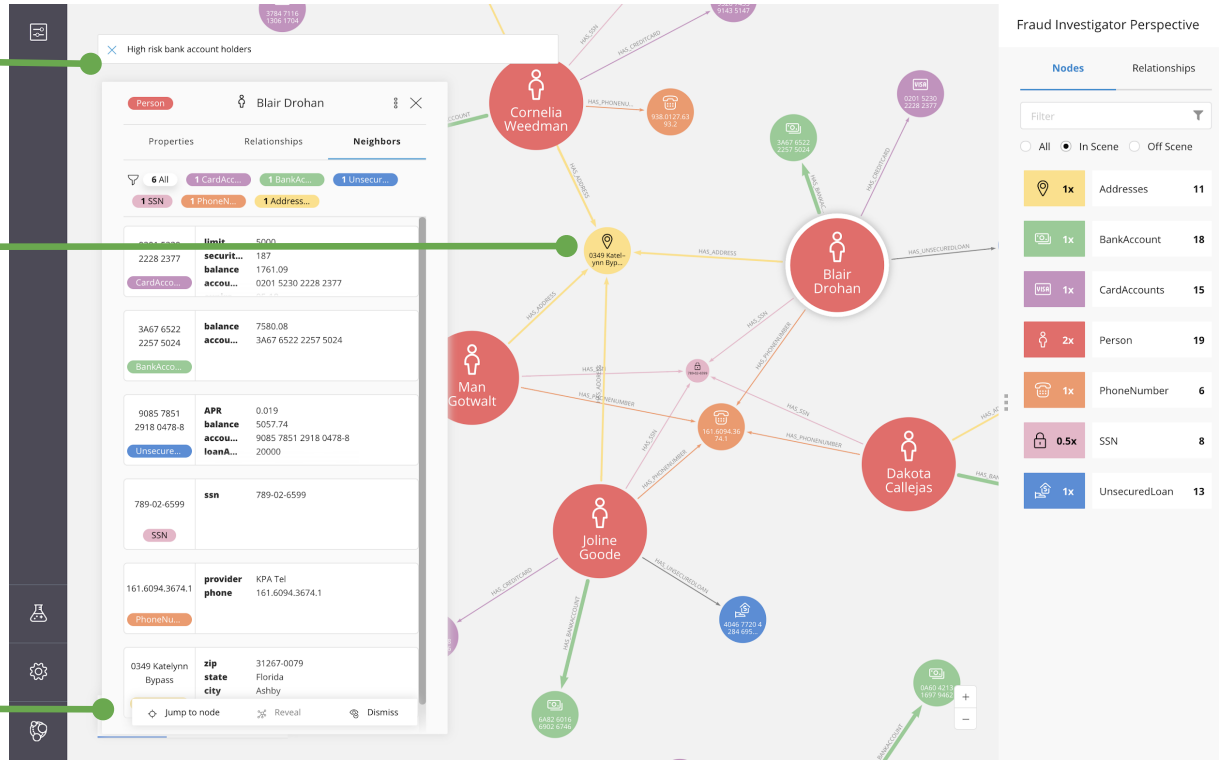
Search with type-ahead suggestions

Flexible Color, Size and Icon schemes

Visualize, Explore and Discover

Pan, Zoom and Select

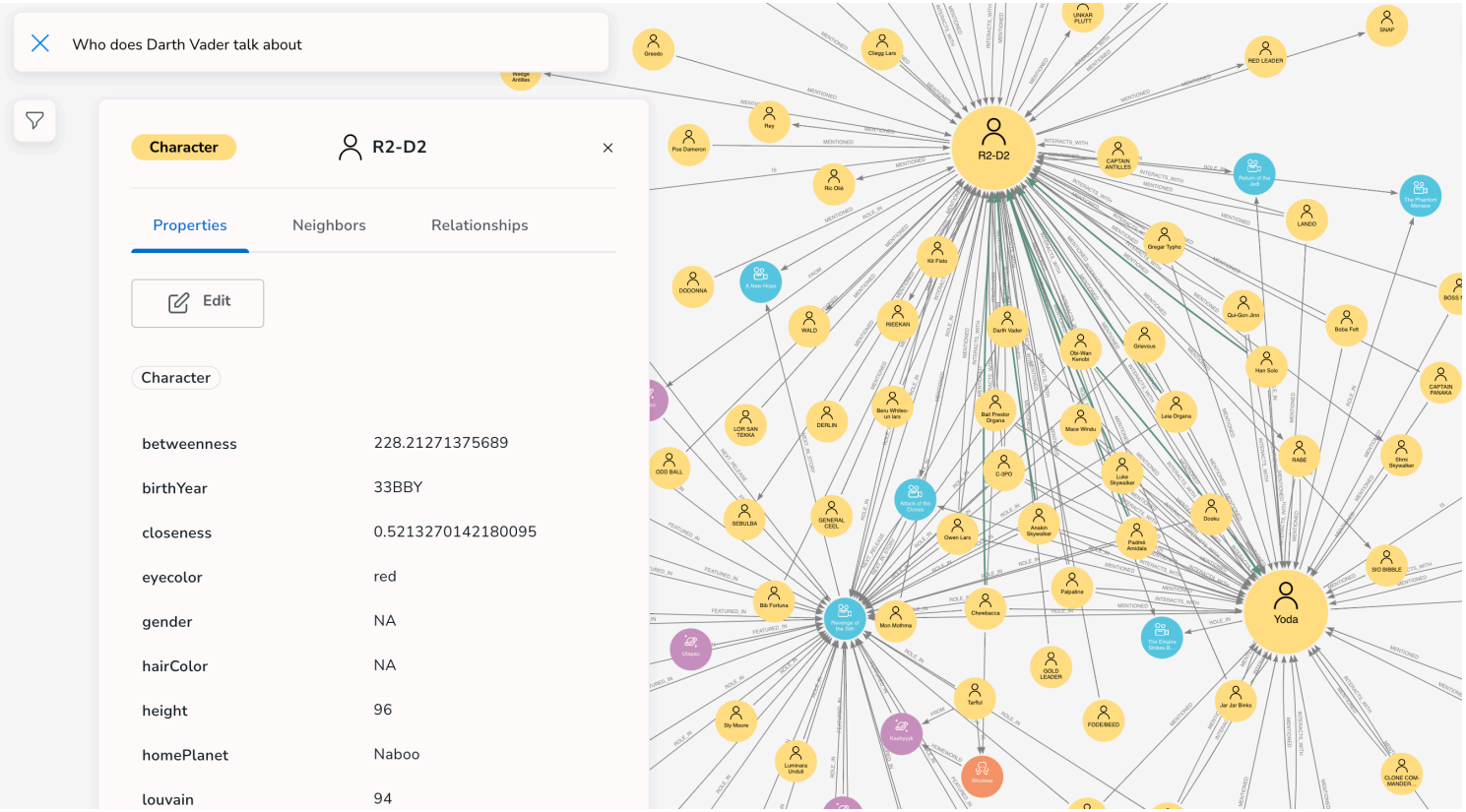
Property Browser and editor



Lab 5 - Exploring the Graph with Bloom

Lab 5.1 - Exploring the Graph Using Bloom

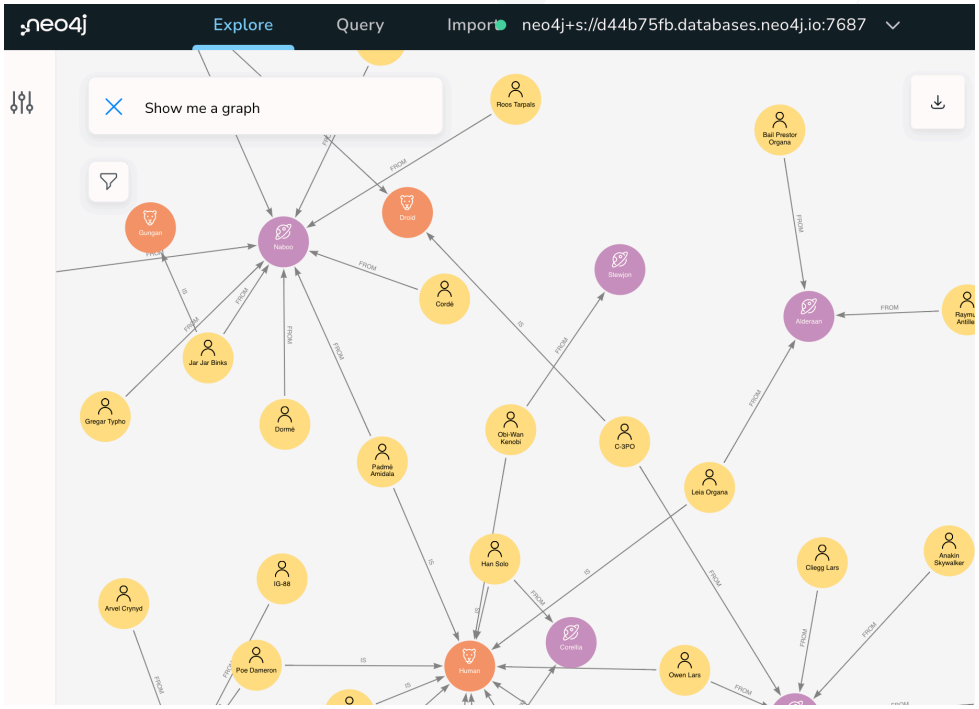
Explore the graph without writing code (unless you want to)



Lab 5.2 - Exploring Your DB using Bloom

Switch to Explore Tab

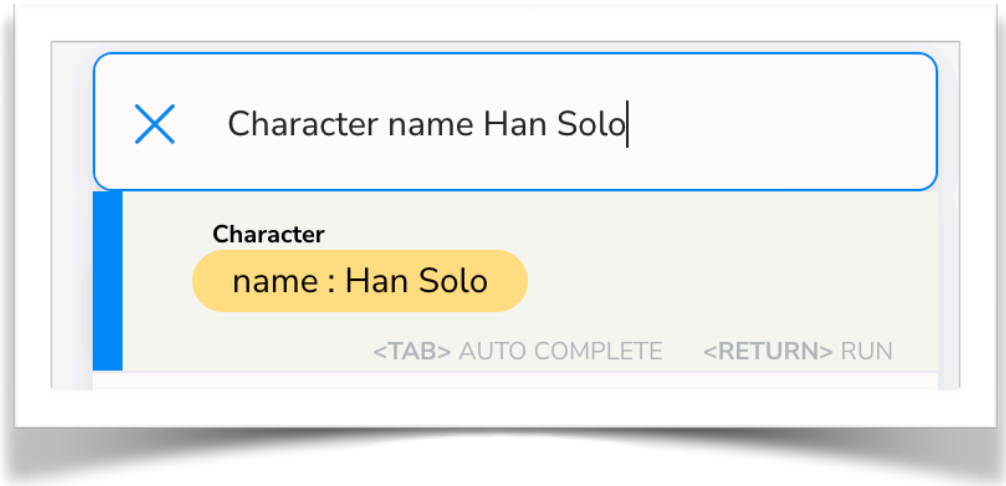
- Clicking on the Explore tab will invoke Neo4j Bloom.
- Bloom starts with a sample query to give an idea of the overall graph.

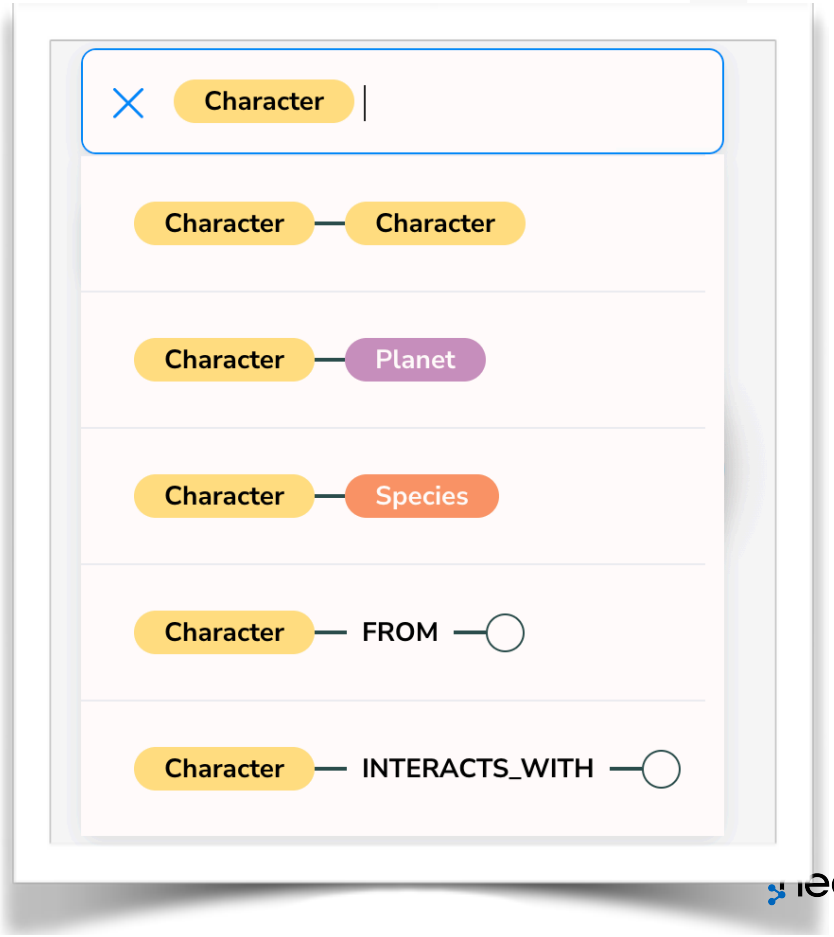


Lab 5.3 - Exploring Your DB using Bloom

Pattern Searching using NLP

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

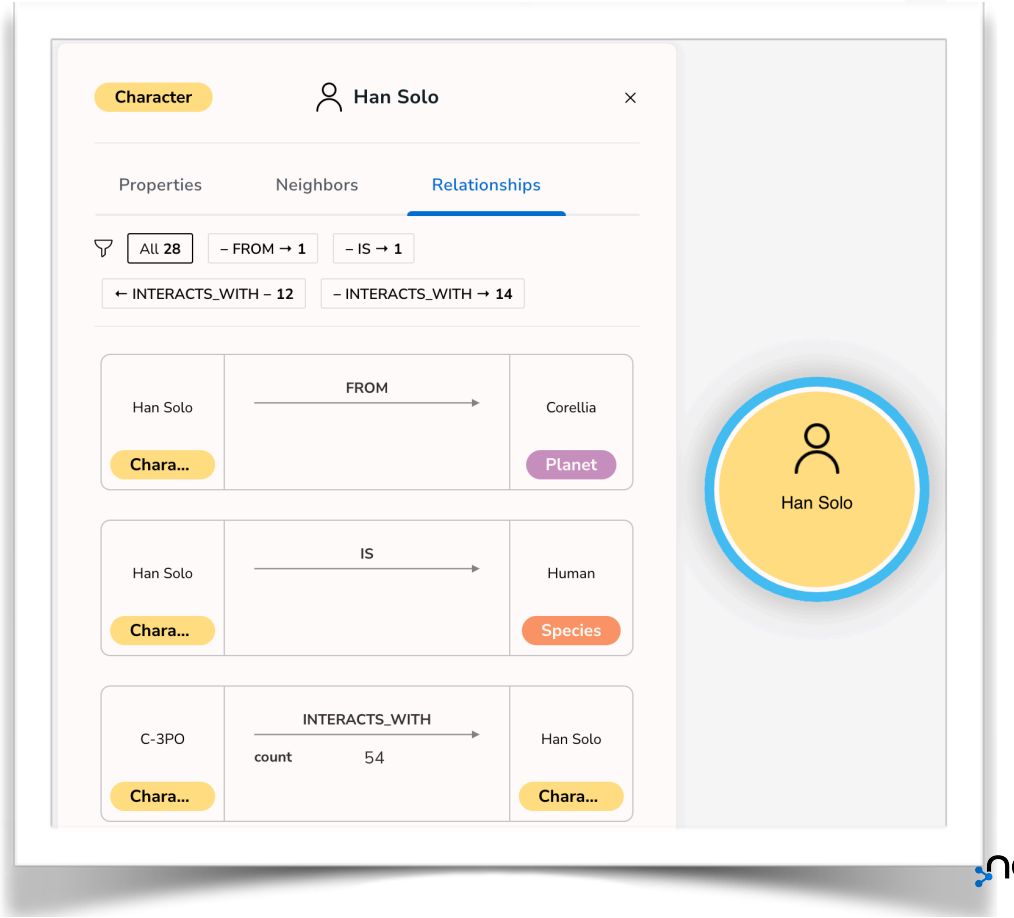




Lab 5.4 - Exploring Your DB using Bloom

Inspect Nodes to see properties and relationships

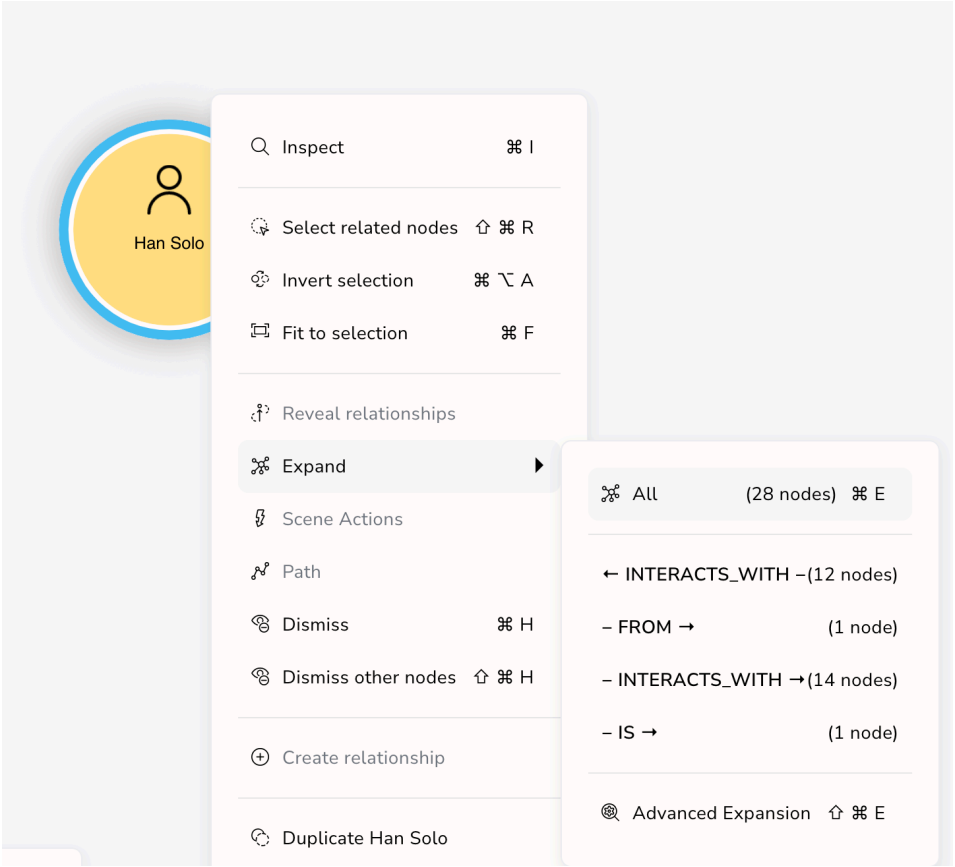
- Double-click on a node to inspect a node.
- Properties may be viewed and updated
- Related nodes can be viewed in the Neighbors tab or Relationships tab.



Lab 5.5 - Exploring Your DB using Bloom

Expanding Nodes to show related nodes

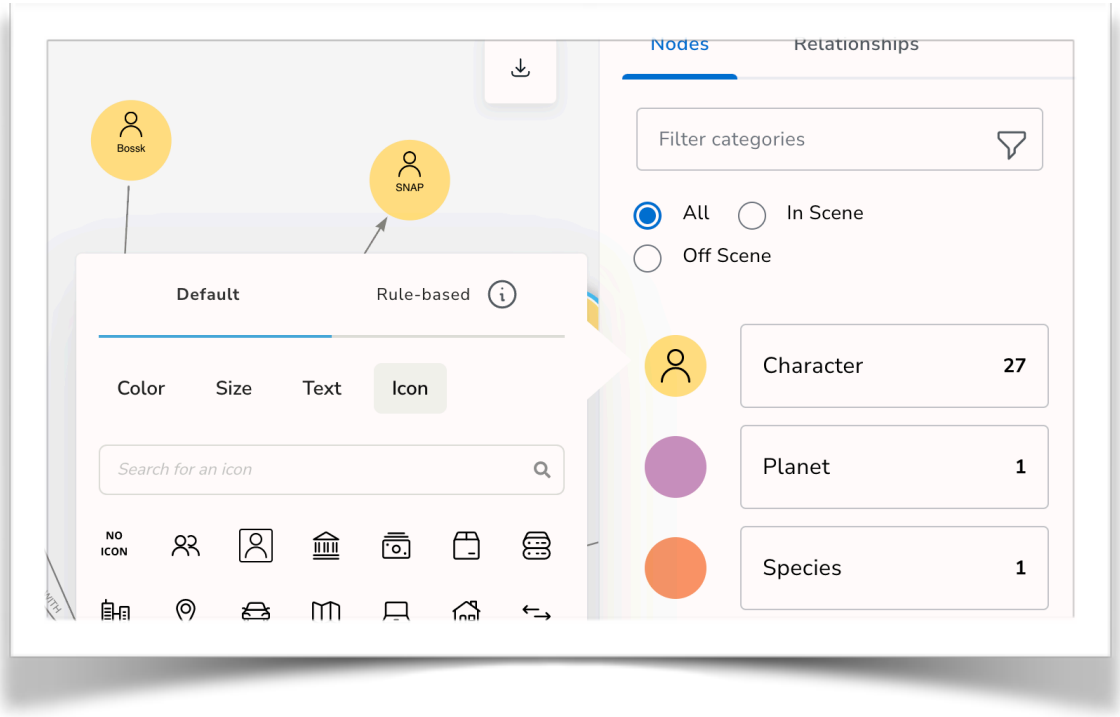
- Right click on a node to show pop-up menu for a node.
- Choose ‘Expand’ to show related nodes. Choose some or all relationships



Lab 5.6 - Exploring Your DB using Bloom

Customizing the Display

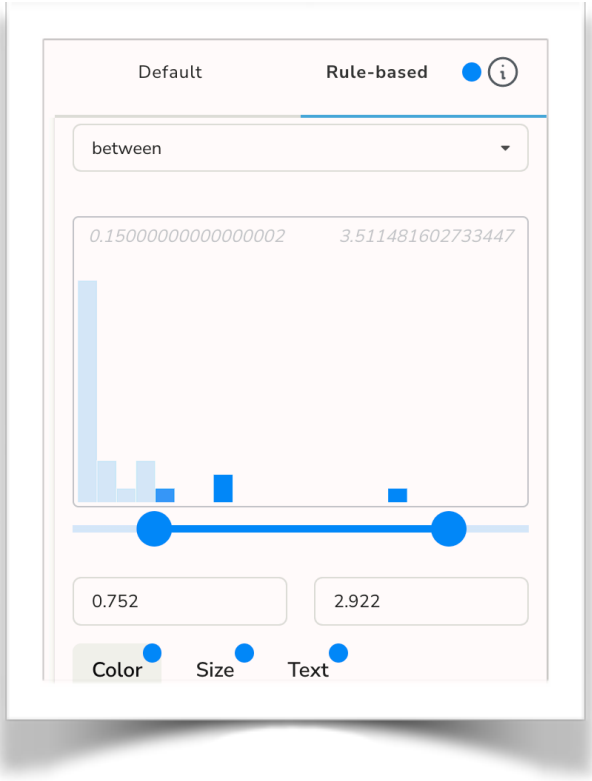
- Click on a Label to show display options.
- Color, size, text, and icons can all be selected.
- Rules can be used to highlight certain nodes

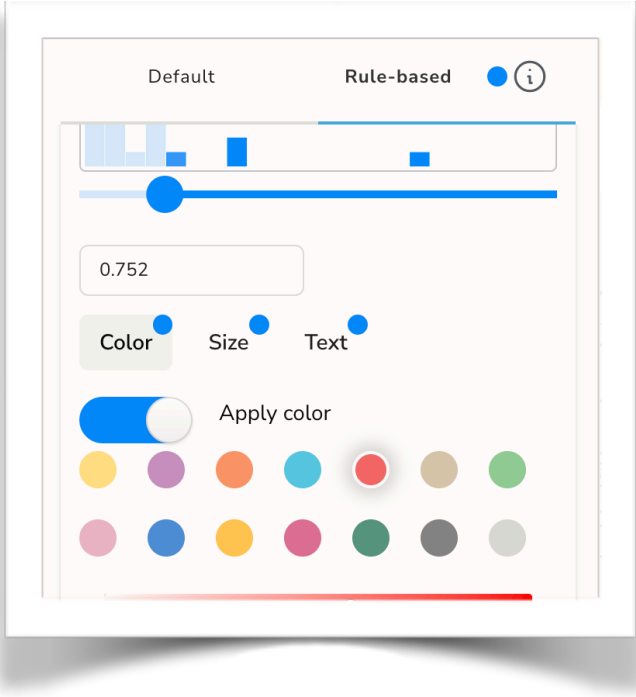


Lab 5.7- Exploring Your DB using Bloom

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

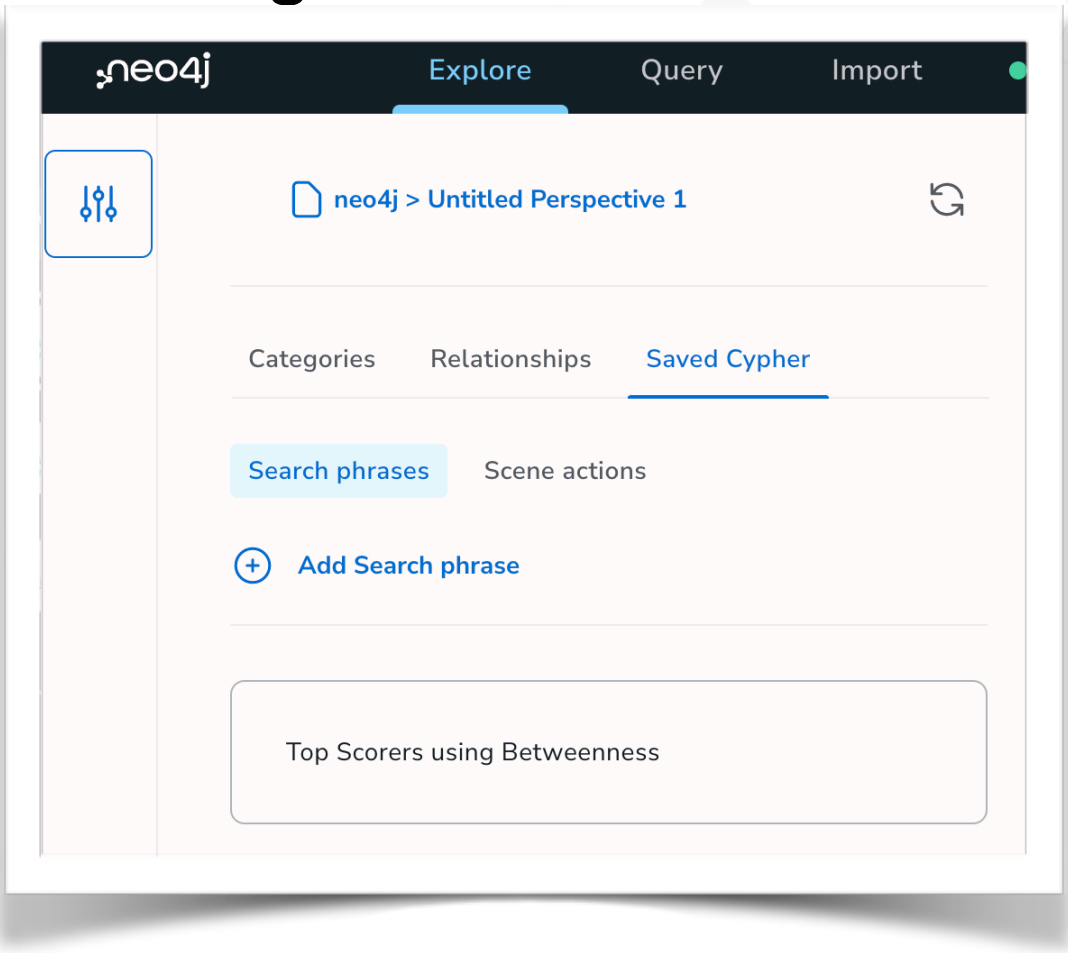




Lab 5 - Exploring Your DB using Bloom

Using Saved Cypher

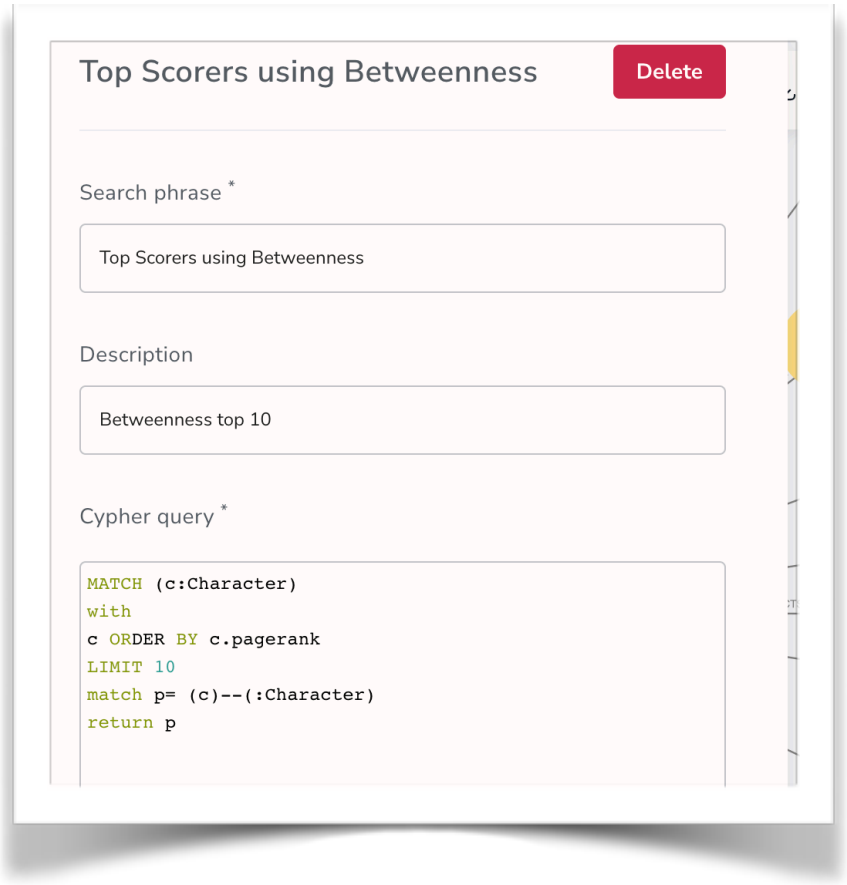
- Click on the Perspectives Icon (upper left corner)
- Click on Saved Cypher / Search Phrases
- Add a search phrase



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

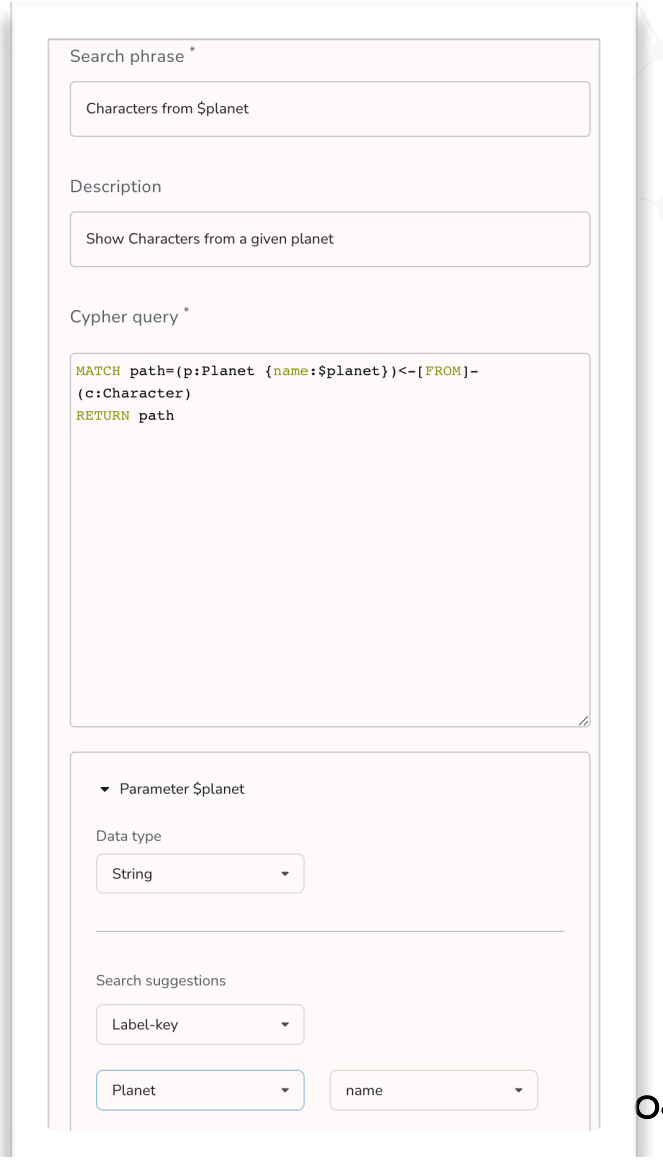


Lab 5.8 - Parameterized Queries

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:

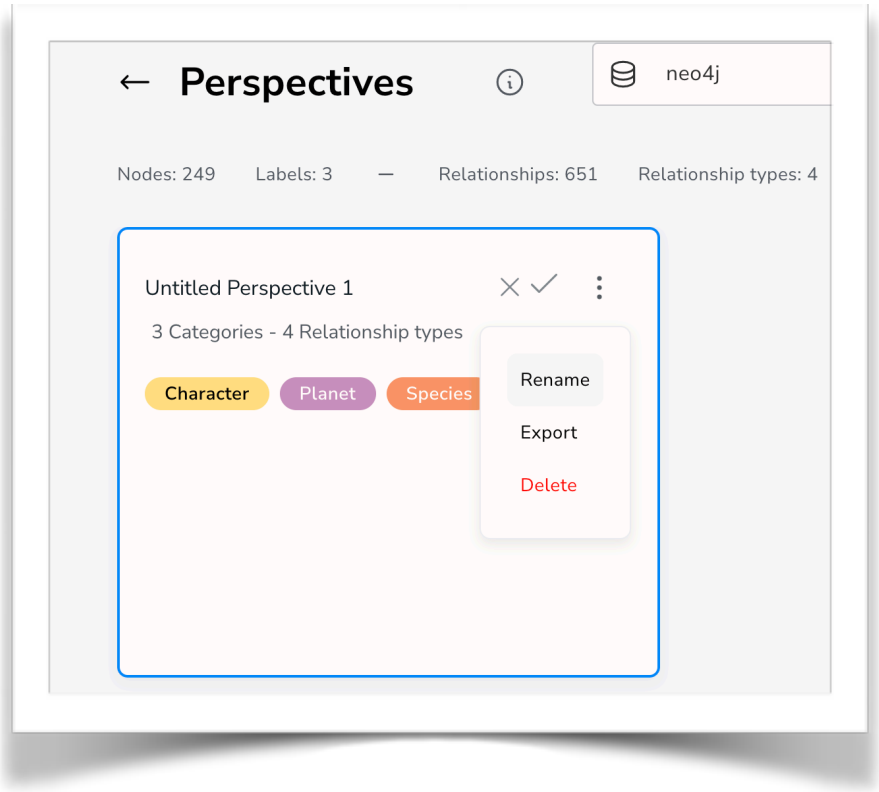
  MATCH path=(p:Planet {name: $planet})<-[FROM]-(c:Character) RETURN path
- Define how Bloom should suggest parameter values



Lab 5.9- Export Perspectives

Use JSON to persist & share perspectives

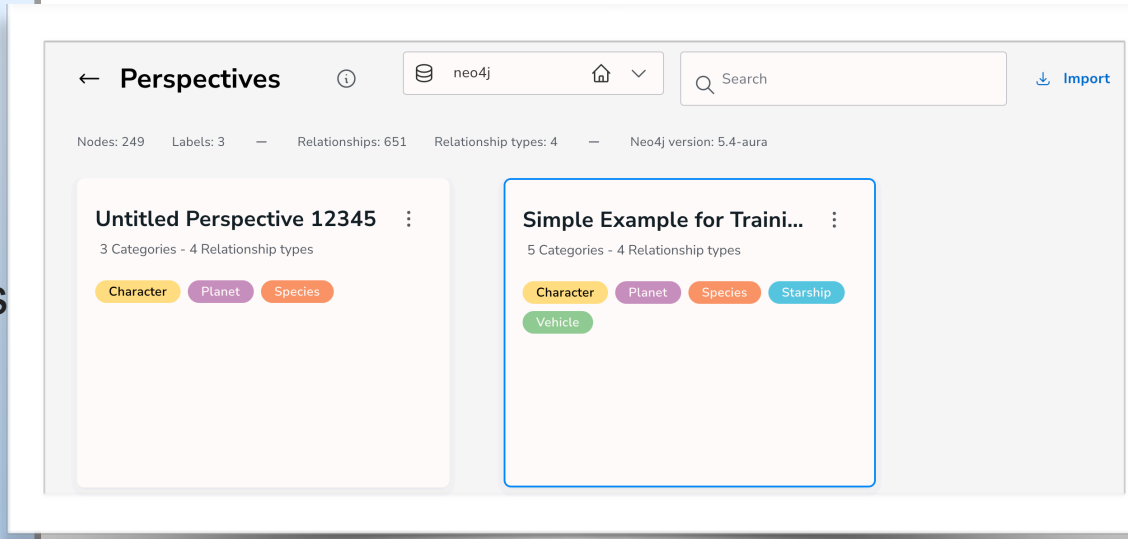
- Perspectives may be exported to JSON files for persistence and versioning
- Export your perspective to a JSON file (optional)



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



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]:

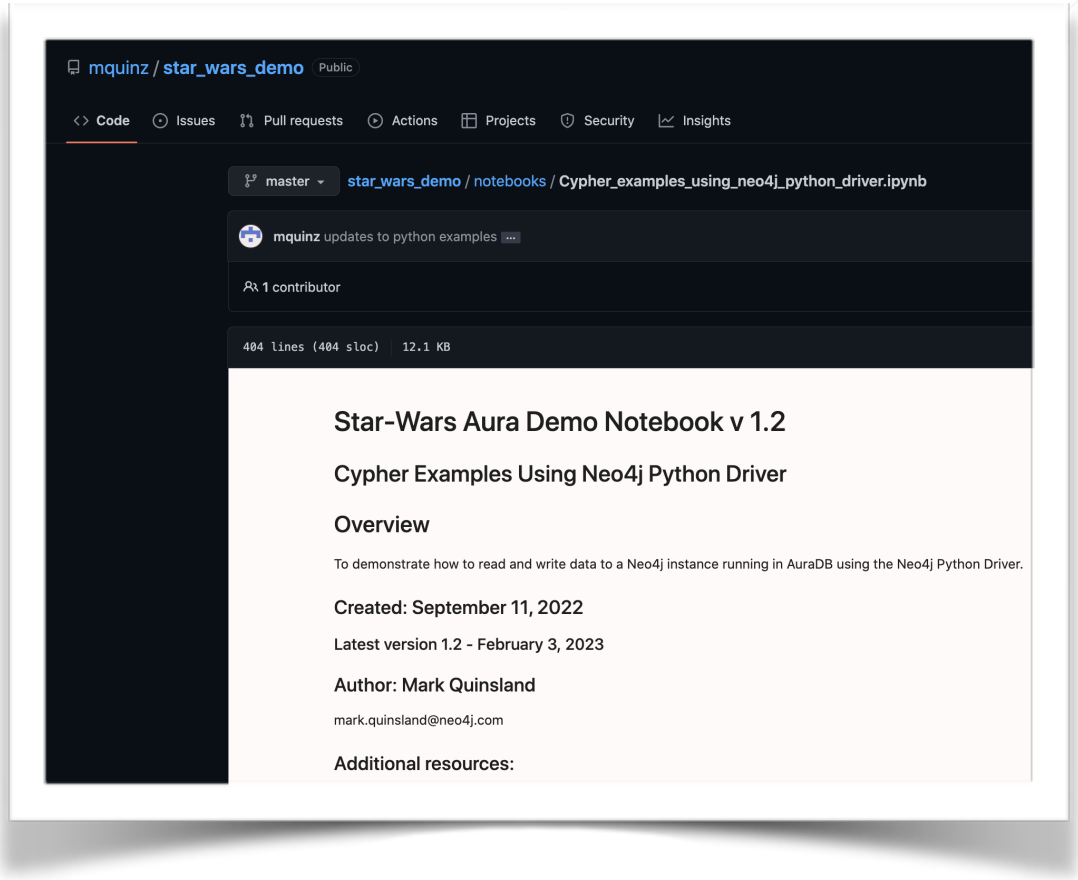
	otherName	interactionCount
0	Chewbacca	77
1	Leia Organa	69
2	C-3PO	54

https://github.com/mquinz/star_wars_demo/blob/master/notebooks/Cypher_examples_using_neo4j_python_driver.ipynb

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



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'])
```


neo4j

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

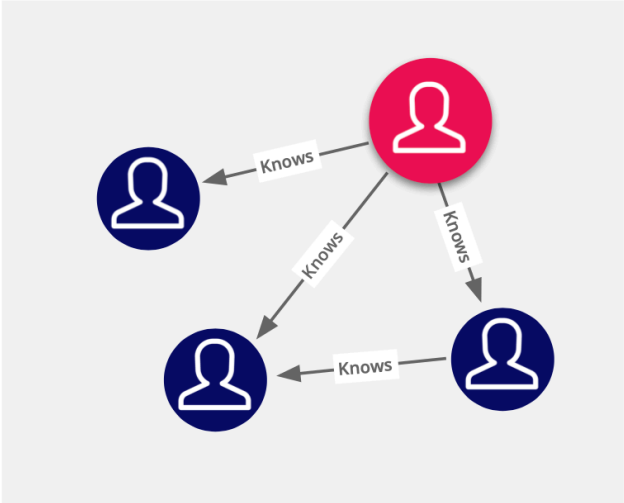
Insert Nodes and Relationships Using Batches

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

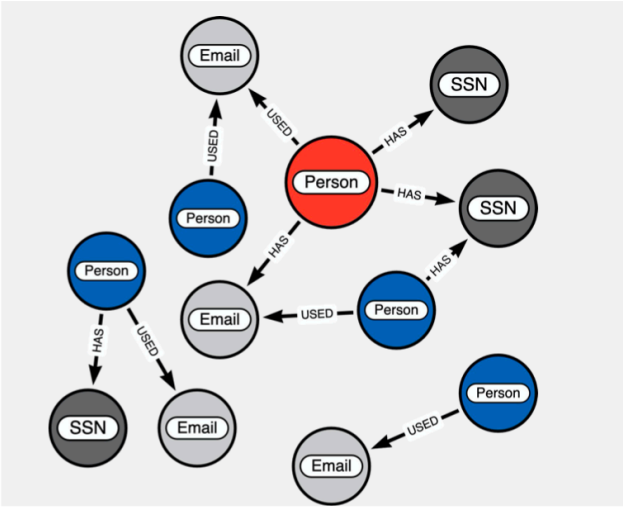
```
params = { }

# 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)
```

Neo4j Graph Data Science

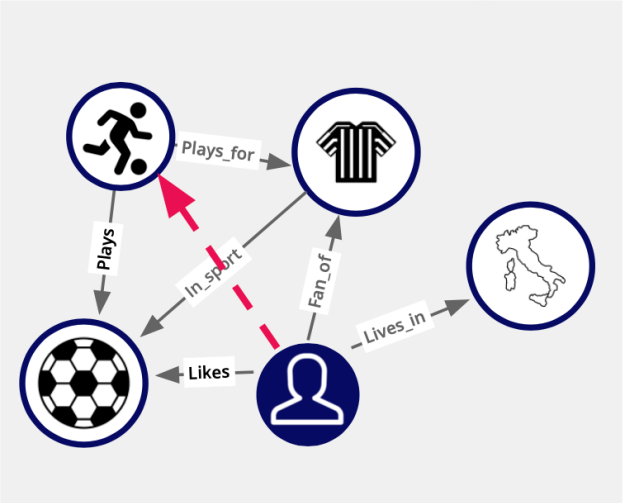
Use Graph Data Science to Answer:



What's important?



What's unusual?



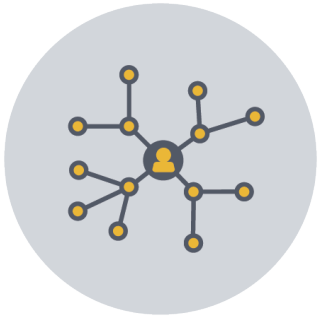
What's next?

neo4j

Graph Algorithm Categories

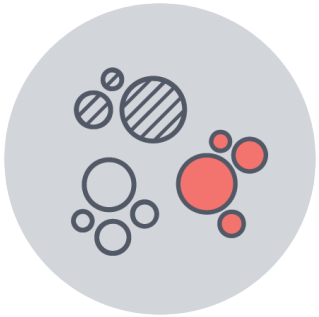
Pathfinding and Search

Finds optimal paths or evaluates route availability and quality.



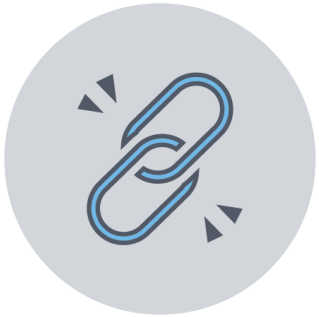
Centrality

Determines the importance of distinct nodes in the network.



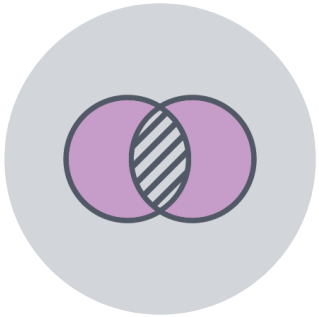
Community Detection

Detects group clustering or partition.



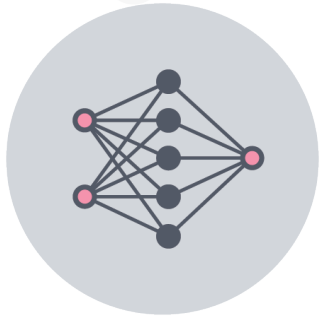
Heuristic Link Prediction

Estimates the likelihood of nodes forming a future relationship.



Similarity

Evaluates how alike nodes are by neighbors and relationships.



Embeddings

Learns graph topology to reduce dimensionality for ML

neo4j

Graph Centrality Algorithms Assist in Analysis

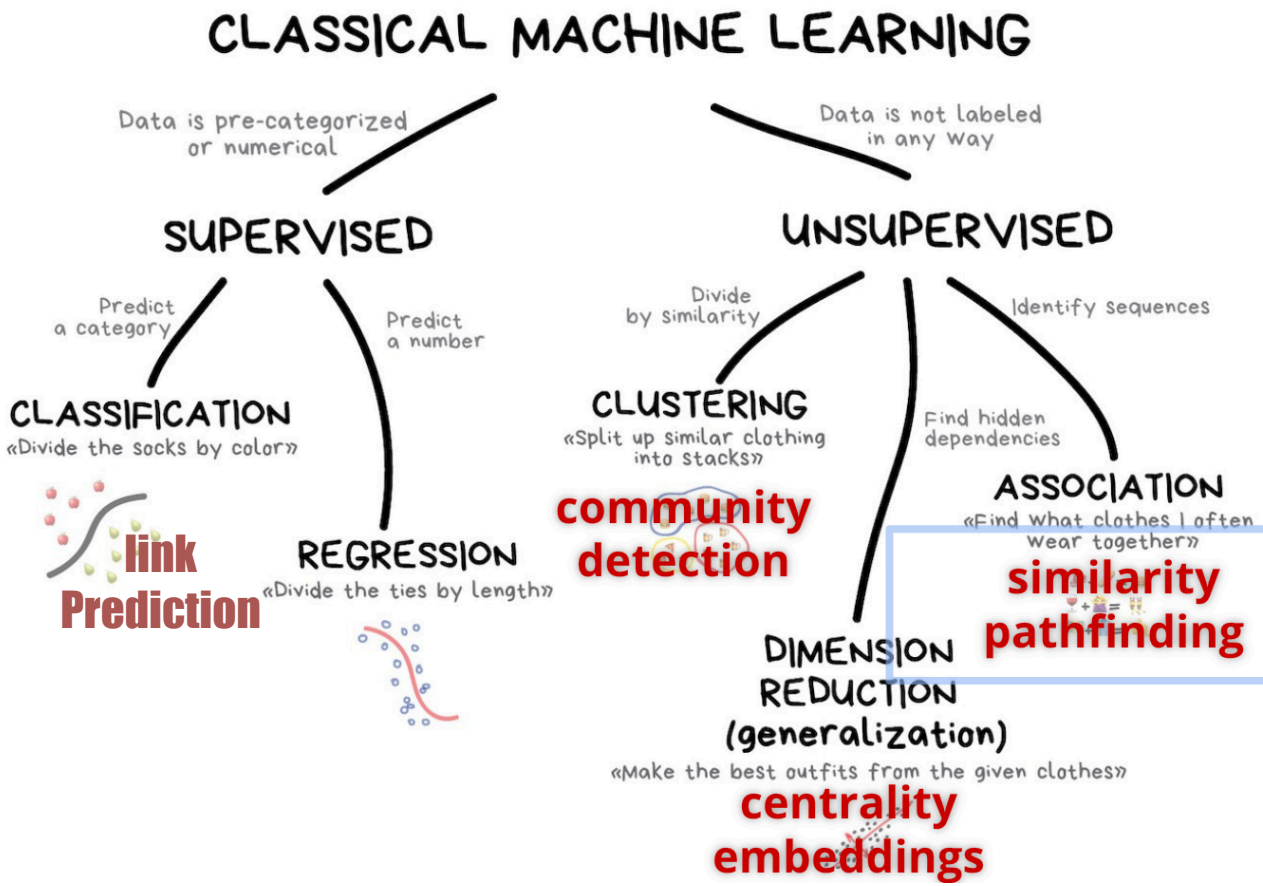
- Closeness algorithms identify:
  - Highly-connected nodes
  - Built-in redundancy
- Betweenness algorithms identify:
  - Strategically-connected nodes
  - Single points of failure
  - Potentially vulnerable nodes

neo4j

Neo4j GDS - Part of your ML Pipeline

Machine learning can be

- **unsupervised**: unlabelled, identifying patterns, *or*
- **supervised**: labeled data, predicting values



Neo4j Graph Data Science - End User Experience

Built for Data Scientists

- **Native Python Client**
- 65+ Pretuned Algos
- Single API for load, analysis & writeback
- Production tier graduation

Connect to Graph Data Science (GDS)

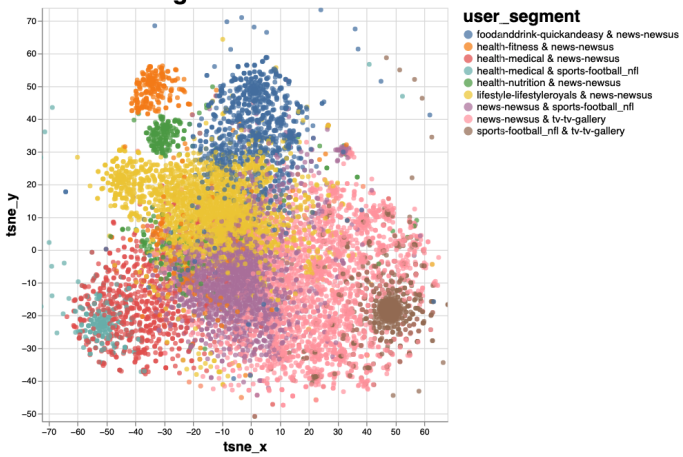
```
In [3]: from graphdatascience import GraphDataScience

        # Use Neo4j URI and credentials according to your setup
        gds = GraphDataScience(HOST, auth=(USERNAME, PASSWORD), aura_ds=True)
```

Apply GDS for Automated Graph Feature Engineering

FastRP transforms the graph of interconnected user activity into embedding features for EDA & ML applications

```
In [4]: g, _ = gds.graph.project('proj',['User','News'], {'CLICKED': {'orientation': 'UNDIRECTED'}})
        gds.fastRP.mutate(g,mutateProperty='segmentEmbedding',embeddingDimension=128,randomSeed=7474)
        gds.graph.writeNodeProperties(g, ['segmentEmbedding'], ['User'])

Out[4]: writeMillis                     1212
        graphName                       proj
        nodeProperties       [segmentEmbedding]
        propertiesWritten               750434
        Name: 0, dtype: object
```

User Segmentation Clusters



Let’s Do Something Amazing Together