

PROJECT 3: NEO4J

Made by:
Marina Gómez Rey
María Ángeles Magro Garrote

INDEX

EXERCISE 1	3
2.1.	3
2.2.	4
2.3.	4
2.4.	5
2.5.	6
2.6.	7
2.7.	8
2.8.	8
EXERCISE 2: CREATE A GRAPH DATABASE	10
2.1.	10
2.2.	13
2.3.	13
2.4.	14
2.5.	15
2.6.	15
2.7.	16
2.8.	17
2.9.	18

EXERCISE 1

2.1.

Write a query to display the schema of your database.

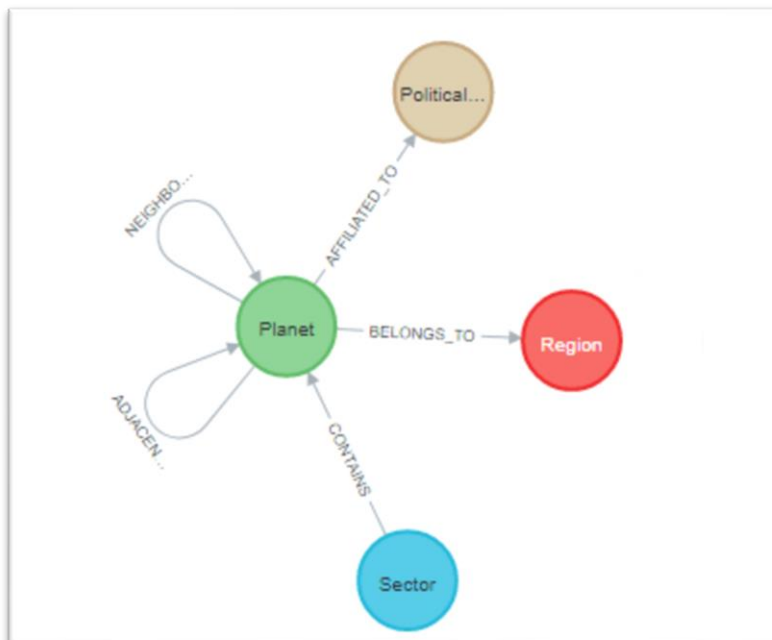
In order to retrieve a schema of our data, the tool “db.schema.visualization” is used.

As seen in the result, the relationships among different objects are obtained: planets can be adjacent to or neighborhood to another planet, a planet can be affiliated to political systems as well as they can belong to a region or contain a sector.

Query:

CALL db.schema.visualization

Result:



2.2.

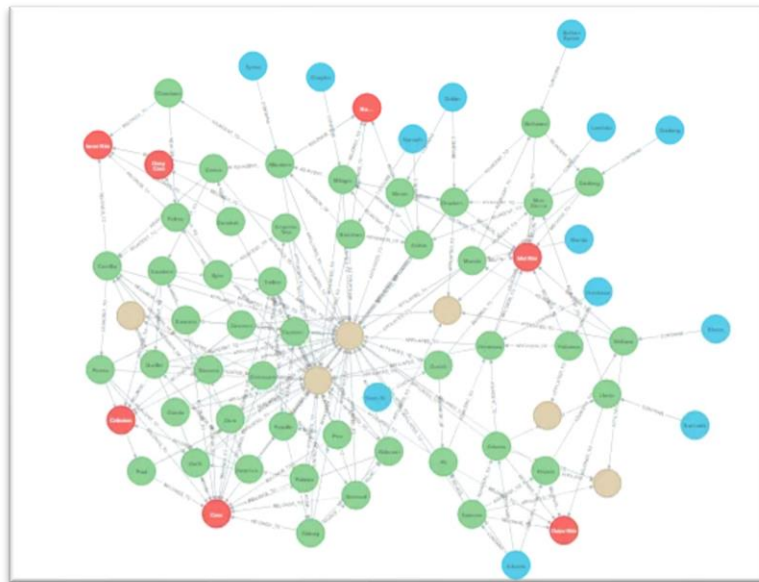
Recover all nodes and relationships (display graph).

In order to retrieve all the nodes, all of them are matched and then returned. All the nodes are shown with all the relationships they have with other nodes. That is why a big graph is obtained.

Query:

```
MATCH (n) RETURN n
```

Result:



2.3.

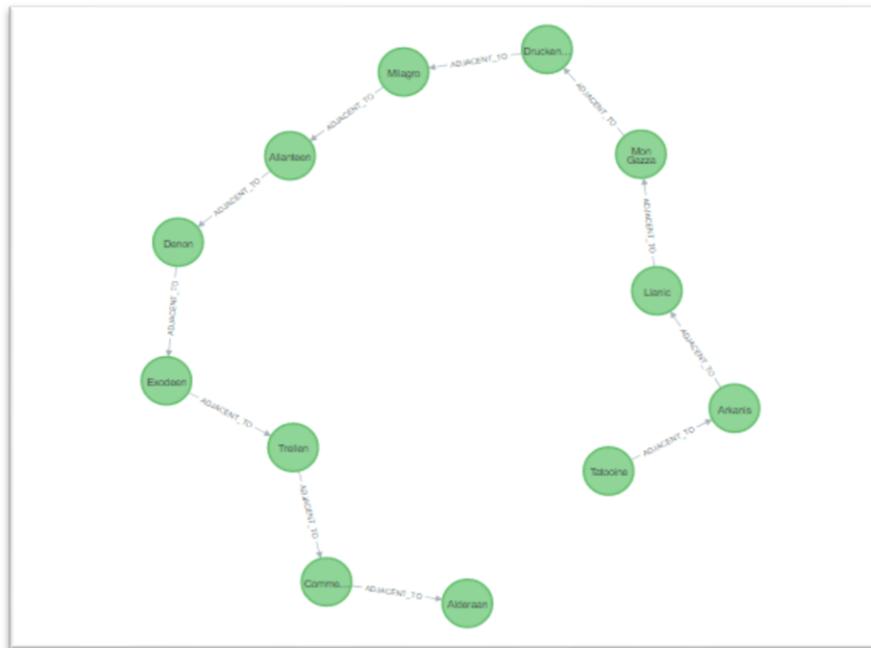
Find the way to "Alderaan" (return the graph with the best route), starting from the planet "Tatooine".

First, to obtain the best route, the function “shortestPath” is used. The start point will be “Tatooine” and then the route will continue following planets that are adjacent or neighbors of the previous planet. The route will finish once “Alderaan” has been reached. This route will be saved using the variable p and will be then returned.

Query:

```
MATCH p=shortestPath((p1:Planet{name:'Tatooine'})-  
[r:ADJACENT_TO|NEIGHBOR_OF*] - (p2:Planet{name:'Alderaan'}))  
RETURN p
```

Result:



2.4.

Returns a list indicating which are the dominant political forces in the galaxy and how many affiliated planets each of them has, in descending order.

The political forces have been obtained throughout the relationship affiliated to. Taking into account that some political systems are repeated more than once, in the RETURN it has been used the function “distinct” so they do not appear more than once. Then, in order to order them from more important to less important, the count of the planets that each system has has been saved and used to order it in ORDER BY. As the statement says, the order must be descending so the DESC is applied.

Query:

```
MATCH (p:Planet)-[:AFFILIATED_TO]->(ps:Political_System)
RETURN distinct ps.system as Political_System, count(p) as Affiliated_Planets
ORDER BY Affiliated_Planets DESC
```

Result:

Political_System	Affiliated_Planets
"Galactic Empire"	35
"Galactic Republic"	29
"Confederacy of Independent Systems"	5
"Hutt Clan"	4
"Jedi Order"	2
"Neutral"	1

2.5.

Returns a list where indicate which sectors, regions and how many planets are in the Galactic Empire better established, in descending order.

First, as it has been asked for sectors, regions and planets, they are all retrieved in the MATCH, saving them by “r” (Region), “p” (Planet) and “s” (Sector).

Afterwards, the following condition is applied in the WHERE statement: they must be affiliated to the Galactic Empire.

Again, the same region could be obtained more than once, so the distinct function is used. Then, apart from retrieving the name of the sector, the count of the planets in that area is returned.

As the statement asks to order them following how well they are established, the ORDER BY is used, ordering the result by the number of planets in descending order.

Query:

```
MATCH (r:Region)-[:BELONGS_TO]-(p:Planet)-[:CONTAINS]-(s:Sector)
WHERE (p)-[:AFFILIATED_TO]->(:Political_System {system: 'Galactic Empire'})
RETURN distinct r.name as Region, s.name as Sector, count(distinct p) as Planets
ORDER BY Planets DESC
```

Result:

Region	Sector	Planets
"Outer Rim"	"Arkanis"	4
"Mid Rim"	"Doldur"	2
"Expansion Region"	"Narvath"	2
"Mid Rim"	"Trans-Nebular"	1
"Expansion Region"	"Tynna"	1
"Mid Rim"	"Manda"	1
"Mid Rim"	"Herdessa"	1
"Expansion Region"	"Chaykin"	1

2.6.

Find all the planets that belongs to the region "Inner Rim", return the graph

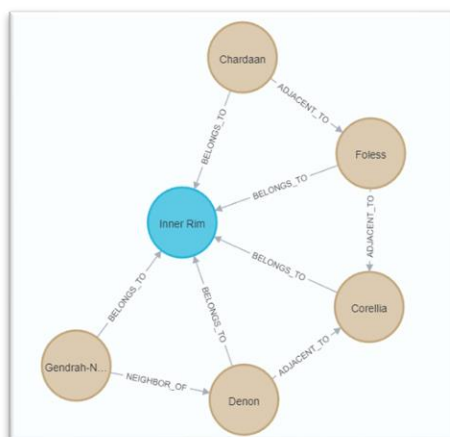
The only condition applied in the MATCH is that the planet must belong to the region "Inner Rim".

If the RETURN * is used, all the relationships will be shown, including the "Inner Rim" in the graph, which is a more visual representation: all the relationships among the planets will be shown such as adjacent to or neighbor of, apart from the "belongs_to" with the Inner Rim.

Query:

```
MATCH (p)-[:BELONGS_TO]->(r:Region{name:'Inner Rim'})
RETURN *
```

Result:



2.7.

Return the graph that represents the trajectory from the "Trans-Nebular" sector, to reach the planet "Pii".

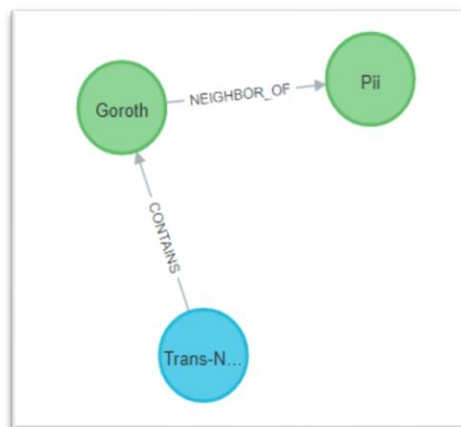
First, the “Trans-Nebular” sector is saved using the variable s1. Then, the planet “Pii” is saved in p1.

Then, the function “ShortestPath” is used again but this time, the path could go through anywhere, as the start point is a sector, not a planet. So the “*” is used to indicate this, and s1 in the (start) and p1 (end) are also included in the function. This path is saved in p, which is then returned.

Query:

```
MATCH(s1:Sector{name:'Trans-Nebular'}),(p1:Planet{name:'Pii'}), p=ShortestPath((s1)-[*]-(p1))
RETURN p
```

Result:



2.8.

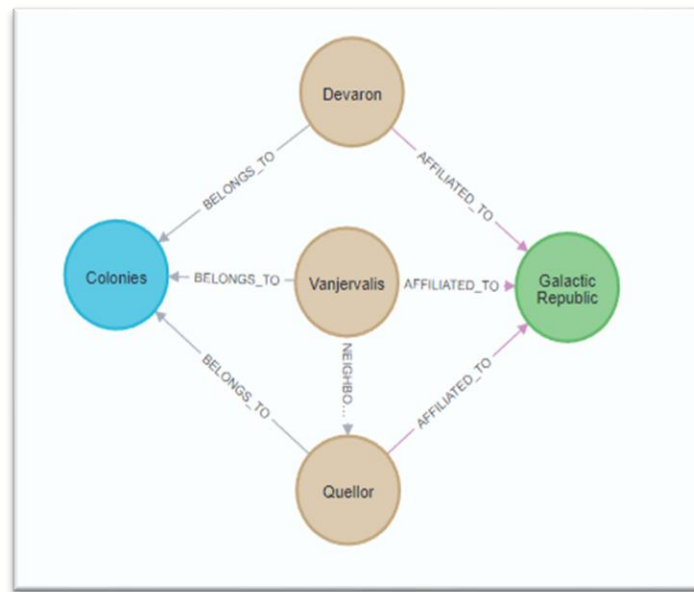
Return the graph that represents the planets that are part of the "Colonies" sector that are under the power of the "Galactic Republic" political system.

There are two main conditions: the planets must be part of “Colonies” and of “Galactic Republic”. So, the planet must be conditioned with the relationship `Belong_to` and `Affiliated_to`. This information is saved in p (which are the planets that satisfy both conditions), which is then returned.

Query:

```
MATCH (r:Region{name:"Colonies"})<-[:BELONGS_TO]-(p:Planet)-[:AFFILIATED_TO]-> (ps:Political_System {system:"Galactic Republic"})
RETURN p
```


Result:



EXERCISE 2: CREATE A GRAPH DATABASE

2.1.

Write the script, insert data (nodes, relationships, properties) into the graph, display the screenshot of the right menu (Database Information) and output (Table).

The insertion of the data is done with the create function. In it, the Person and Movie objects are created. The person includes the name, lastname, nick and account and the movie, the title, date (of the premiere) and the budget. Then, there are some relationships that must be added, the ACTED_IN (between actors and the films they acted in, it includes the role they had in the film), DIRECTED (between the person that directed a movie and the movie), RECOMMENDED (between a person and the movie they recommend, it also includes “to” that includes to who the recommendation is for) and the FRIEND_OF (between two people that are friends, it includes the date in which they became friends).

- Insert data

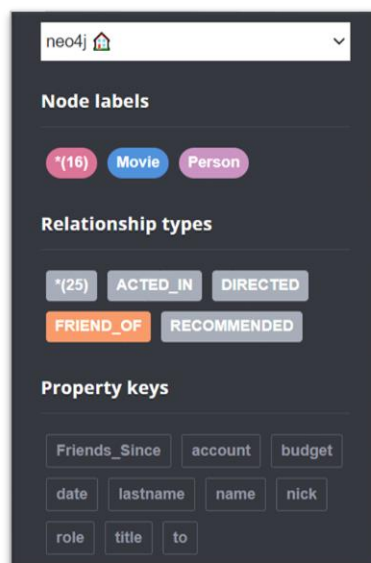
CREATE

```
(Alejandro:Person {name: 'Alejandro', lastname: 'García', nick: 'Alejandrito', account: 'Ale03'}),
(JamesMc:Person {name: 'James', lastname: 'McAvoy', nick: 'Jamie', account: 'JMCA3'}),
(JamesM:Person {name: 'James', lastname: 'Marsden', nick: 'JamesM', account: 'JMen7'}),
(Paula:Person {name: 'Paula', lastname: 'Gómez', nick: 'Paulaa', account: 'La_Pau'}),
(Keanu:Person {name: 'Keanu', lastname: 'Reeves', nick: 'KeaRee', account: 'Keanu09'}),
(Jorge:Person {name: 'Jorge', lastname: 'Sánchez', nick: 'George', account: 'Jorgi8'}),
(Laura:Person {name: 'Laura', lastname: 'Ruiz', nick: 'Lau', account: 'Lauu2'}),
(Melania:Person {name: 'Melania', lastname: 'Ulloa', nick: 'Mela', account: 'Mela87'}),
(Lana:Person {name: 'Lana', lastname: 'Wachowski', nick: 'Lanita', account: 'Lana23'}),

(Chad:Person {name: 'Chad', lastname: 'Stahelski', nick: 'GChad', account: 'ChadS'}),
(Ian:Person {name: 'Ian', lastname: 'McShane', nick: 'Iaan', account: 'IanMS'}),
(The_Matrix:Movie {title: 'The Matrix', date: '23/06/1999', budget: 63000000}),
(John_Wick:Movie {title: 'John Wick', date: '24/10/2014', budget: 20000000}),
(The_Avengers:Movie {title: 'The Avengers', date: '27/04/2012', budget: 220000000}),
(Split:Movie {title: 'Split', date: '26/09/2018', budget: 9000000}),
(Sonic:Movie {title: 'Sonic', date: '21/02/2020', budget: 9000000}),
(JamesMc)-[:ACTED_IN {role: 'Main actor'}]->(Split),
(JamesM)-[:ACTED_IN {role: 'Main actor'}]->(Sonic),
(Keanu)-[:ACTED_IN {role: 'Main actor'}]->(The_Matrix),
```

(Keanu)-[:ACTED_IN {role: 'Main actor'}]->(John_Wick),
 (Ian)-[:ACTED_IN {role: 'Main actor'}]->(John_Wick),
 (Lana)-[:DIRECTED]->(The_Matrix),
 (Chad)-[:DIRECTED]->(John_Wick),
 (Lana)-[:RECOMMENDED {to: 'Jorge'}]->(John_Wick),
 (Lana)-[:RECOMMENDED {to: 'Jorge'}]->(The_Matrix),
 (Jorge)-[:RECOMMENDED {to: 'Alejandro'}]->(The_Avengers),
 (Jorge)-[:RECOMMENDED {to: 'Lana'}]->(Sonic),
 (Laura)-[:RECOMMENDED {to: 'Chad'}]->(The_Matrix),
 (Laura)-[:RECOMMENDED {to: 'Chad'}]->(Sonic),
 (Alejandro)-[:RECOMMENDED {to: 'Paula'}]->(The_Matrix),
 (Alejandro)-[:RECOMMENDED {to: 'Laura'}]->(The_Avengers),
 (Alejandro)-[:RECOMMENDED {to: 'Jorge'}]->(John_Wick),
 (Alejandro)-[:FRIEND_OF{Friends_Since:"13/11/2019"}]->(Paula),
 (Alejandro)-[:FRIEND_OF{Friends_Since:"20/03/2017"}]->(Laura),
 (Alejandro)-[:FRIEND_OF{Friends_Since:"05/06/2015"}]->(Jorge),
 (Lana)-[:FRIEND_OF{Friends_Since:"24/01/2010"}]->(Jorge),
 (Melania)-[:FRIEND_OF{Friends_Since:"13/11/2019"}]->(Paula),
 (Jorge)-[:FRIEND_OF{Friends_Since:"05/06/2015"}]->(Alejandro),
 (Jorge)-[:FRIEND_OF{Friends_Since:"10/10/2018"}]->(Lana),
 (Jorge)-[:FRIEND_OF{Friends_Since:"05/10/2018"}]->(Chad),
 (Laura)-[:FRIEND_OF{Friends_Since:"20/07/2013"}]->(Chad)

- Right menu (database information)



TABLE

<pre>{ "identity": 0, "labels": ["Person"], "properties": { "nick": "Alejandrito", "name": "Alejandro", "account": "Ale03", "lastname": "Garcia" }, "elementId": "0" }</pre>	<pre>{ "identity": 1, "labels": ["Person"], "properties": { "nick": "Jamie", "name": "James", "account": "JMCA3", "lastname": "McAvoy" }, "elementId": "1" }</pre>	<pre>{ "identity": 2, "labels": ["Person"], "properties": { "nick": "JamesM", "name": "James", "account": "JMen7", "lastname": "Marsden" }, "elementId": "2" }</pre>	<pre>{ "identity": 3, "labels": ["Person"], "properties": { "nick": "Paulaa", "name": "Paula", "account": "La_Pau", "lastname": "Gómez" }, "elementId": "3" }</pre>
<pre>{ "identity": 4, "labels": ["Person"], "properties": { "nick": "KeaRee", "name": "Keanu", "account": "Keanu09", "lastname": "Reeves" }, "elementId": "4" }</pre>	<pre>{ "identity": 5, "labels": ["Person"], "properties": { "nick": "George", "name": "Jorge", "account": "Jorgi8", "lastname": "Sánchez" }, "elementId": "5" }</pre>	<pre>{ "identity": 6, "labels": ["Person"], "properties": { "nick": "Lau", "name": "Laura", "account": "Lauu2", "lastname": "Ruiz" }, "elementId": "6" }</pre>	<pre>{ "identity": 7, "labels": ["Person"], "properties": { "nick": "Mela", "name": "Melania", "account": "Mela87", "lastname": "Ulloa" }, "elementId": "7" }</pre>
<pre>{ "identity": 8, "labels": ["Person"], "properties": { "nick": "Lanita", "name": "Lana", "account": "Lana23", "lastname": "Wachowski" }, "elementId": "8" }</pre>	<pre>{ "identity": 9, "labels": ["Person"], "properties": { "nick": "GChad", "name": "Chad", "account": "ChadS", "lastname": "Stahelski" }, "elementId": "9" }</pre>	<pre>{ "identity": 10, "labels": ["Person"], "properties": { "nick": "Iaan", "name": "Ian", "account": "IanMS", "lastname": "McShane" }, "elementId": "10" }</pre>	<pre>{ "identity": 11, "labels": ["Movie"], "properties": { "date": "23/06/1999", "title": "The Matrix", "budget": 63000000 }, "elementId": "11" }</pre>
<pre>{ "identity": 12, "labels": ["Movie"], "properties": { "date": "24/10/2014", "title": "John Wick", "budget": 20000000 }, "elementId": "12" }</pre>	<pre>{ "identity": 13, "labels": ["Movie"], "properties": { "date": "27/04/2012", "title": "The Avengers", "budget": 220000000 }, "elementId": "13" }</pre>	<pre>{ "identity": 14, "labels": ["Movie"], "properties": { "date": "26/09/2018", "title": "Split", "budget": 9000000 }, "elementId": "14" }</pre>	<pre>{ "identity": 15, "labels": ["Movie"], "properties": { "date": "21/02/2020", "title": "Sonic", "budget": 9000000 }, "elementId": "15" }</pre>

2.2.

Write a query to display the schema of your database.

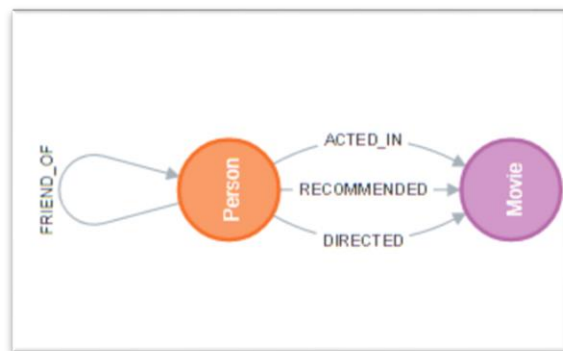
In order to retrieve a schema of our data, the tool “db.schema.visualization” is used.

As seen in the graph obtained, the people are related to being friends. Apart from that, a person is related with a movie because they can be the director, an actor of it or have recommended it.

Query:

CALL db.schema.visualization

Result:



2.3.

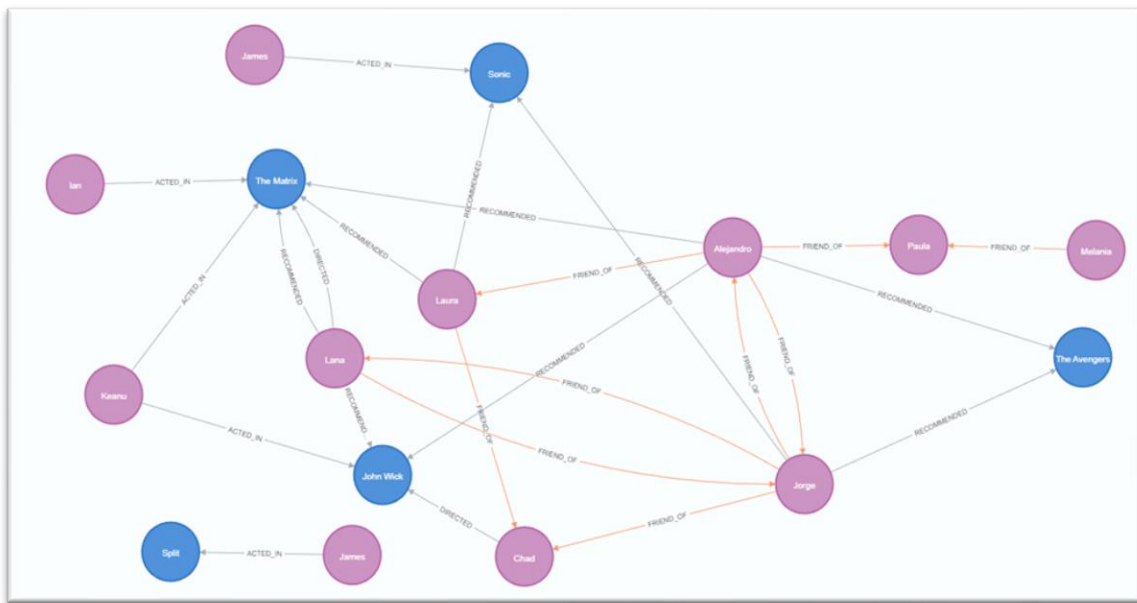
Recover all nodes and relationships (display graph).

In order to retrieve all the nodes, all of them are matched and then returned. All the nodes are shown with all the relationships they have with other nodes.

Query:

MATCH (n) RETURN n

Result:



2.4.

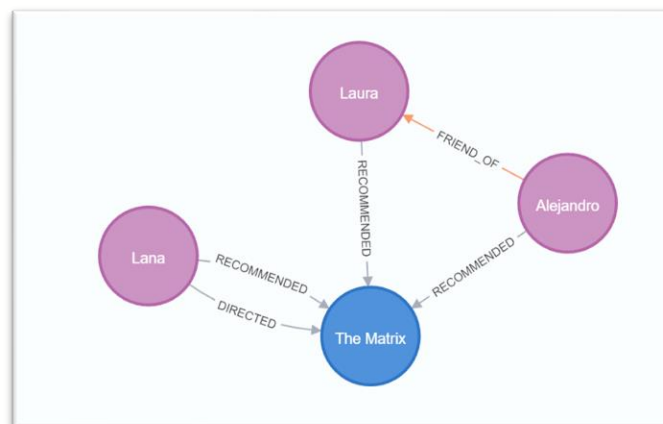
Recover all the people on the network who recommend the writer "The Matrix".

To find the recommendations, we have the RECOMMENDED relationship, so we establish the people as p (as only the Person class has the recommended relation, we do not need to specify that p is a person, if we did that the result would be correct but the graph would not be as visual) and relate it to the movie whose title is "The Matrix". Then, we return the result.

Query:

MATCH (p)-[:RECOMMENDED]->(m:Movie{title:"The Matrix"}) RETURN *

Result:



2.5.

Retrieve all movies that "Keanu Reeves" acted in and return their titles.

To find the films where a specific person acted, we use the ACTED_IN relation and establish the path between a person (who in this case has as name “Keanu” and as last name “Reeves”) and a movie. Then , we return the titles of the movies that are given as a result.

Query:

```
MATCH (p:Person{name:"Keanu", lastname: "Reeves"})-[:ACTED_IN]->(m:Movie)
RETURN m.title as Titles
```

Result:

Titles	
1	"The Matrix"
2	"John Wick"

2.6.

Return all "Alejandro" friends and what movies he has recommended.

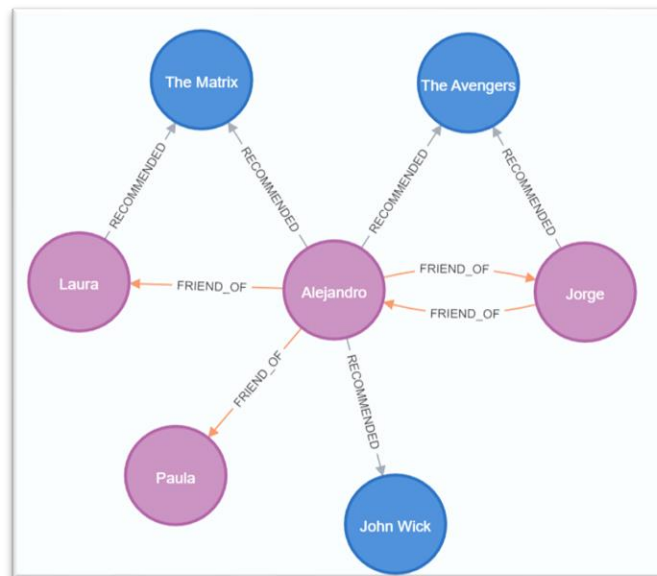
To find Alejandro’s friends and recommended movies, we do two matches, the first one establishes the movies he recommended with that relationship, and the other that returns the friends with the FRIEND_OF relation. Finally, we return * as that gives as a result a really clear graph with the relationships Alejandro has.

On top of that, if instead of the graph we go to the table, we can see the keys and properties of each of the results, so if we go to the r (Recommended) one, we can see who was recommended that film by Alejandro. To see it clearer, that was underlined.

Query:

```
MATCH (p:Person{name:"Alejandro"})-[r:RECOMMENDED]->(m:Movie)
MATCH (p)-[:FRIEND_OF]->(p2:Person)
RETURN *
```

Result:



m	p	p2	r
<pre>{ "identity": 12, "labels": ["Movie"], "properties": { "date": "24/10/2014", "title": "John Wick", "budget": 20000000 }, "elementId": "12" }</pre>	<pre>{ "identity": 0, "labels": ["Person"], "properties": { "nick": "Alejandrino", "name": "Alejandro", "account": "Ale03", "lastname": "García" }, "elementId": "0" }</pre>	<pre>{ "identity": 5, "labels": ["Person"], "properties": { "nick": "George", "name": "Jorge", "account": "Jorgi8", "lastname": "Sánchez" }, "elementId": "5" }</pre>	<pre>{ "identity": 15, "start": 0, "end": 12, "type": "RECOMMENDED", "properties": { "to": "Jorge" }, "elementId": "15", "startNodeElementId": "0", "endNodeElementId": "12" }</pre>

2.7.

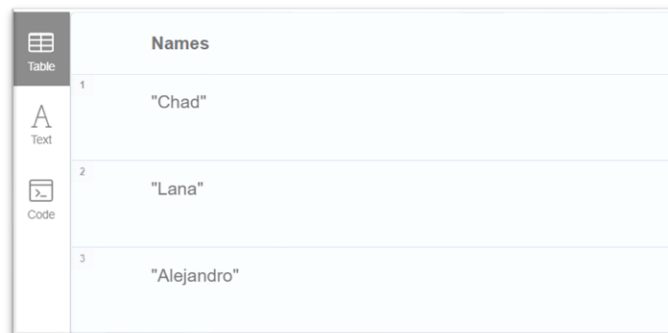
List the friends of "Alejandro's" friends.

To find the friends of the friends of a person, first we need to find the friends of that person. To do that, we specify the person we want to find the friends of and establish the relationship. Once that is done, we do the same thing but with the p2 (friends that have been found) and find the FRIENDS_OF relationship. Those are the friends of the friends so their names are given as a result (distinct as they may be repeated).

Query:

```
MATCH (p:Person{name:"Alejandro"})-[:FRIEND_OF]->(p2:Person), (p2)-[:FRIEND_OF]->(p3:Person)
RETURN DISTINCT(p3.name) as Names
```

Result:



A screenshot of a database query result interface. On the left, there is a vertical toolbar with three icons: a table icon labeled 'Table', a text icon labeled 'Text', and a code icon labeled 'Code'. The main area displays a table with the title 'Names'. The table has three rows, numbered 1, 2, and 3 in the first column. The second column contains the names 'Chad', 'Lana', and 'Alejandro' respectively.

	Names
1	"Chad"
2	"Lana"
3	"Alejandro"

2.8.

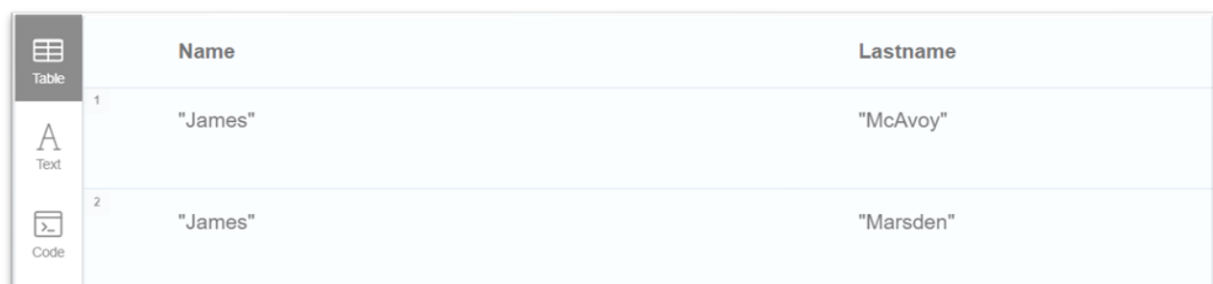
Retrieve all actors whose name begins with "James", returning their names.

To create this query we search for objects type Person where the attribute name starts with James (not equal as the name may be composite, for example "James William"). Then, we return the name and the surname to distinguish them as their names obviously are James.

Query:

```
MATCH (p:Person) WHERE p.name STARTS WITH "James"
RETURN p.name as Name, p.lastname as Lastname
```

Result:



A screenshot of a database query result interface. On the left, there is a vertical toolbar with three icons: a table icon labeled 'Table', a text icon labeled 'Text', and a code icon labeled 'Code'. The main area displays a table with two columns: 'Name' and 'Lastname'. The table has two rows, numbered 1 and 2 in the first column. The second column contains the names 'James' and 'James' respectively. The third column contains the surnames 'McAvoy' and 'Marsden' respectively.

	Name	Lastname
1	"James"	"McAvoy"
2	"James"	"Marsden"

2.9.

Retrieve all directors, their movies, and people who acted in the movies, returning the name of the director, the number of actors the director has worked with, and the list of actors.

To begin, we have to establish the relationship between the directors and the movies and then, the actors who participated in those movies. With that being done, we can already return the asked entities. First, the name of the director, then, with the count function, the number of the actors that contributed to the director's films and finally, we can get the names of the actors with the collect function that returns them in a list. Also, the distinct function is necessary as the same actor can act in different films but we only need to count it once.

Query:

```
MATCH (d:Person)-[:DIRECTED]->(m:Movie)<-[:ACTED_IN]-(a:Person)
RETURN d.name as Director, COUNT(a.name) as Number, collect(distinct(a.name)) as
Actors
```

Result:

	Director	Number	Actors
1	"Lana"	1	["Keanu"]
2	"Chad"	2	["Ian", "Keanu"]