Introduction to Graph Databases

Neo4j

Alejandro Vaisman
avaisman@itba.edu.ar

Neo4j – Graph Data Science

Useful Libraries (revisited)

1. APOC (Awsome Procedures on Cypher)

- Contains many different procedures that extend the capabilities of Neo4j.
- Provides features not covered by Cypher
- Exposes functions (returning a single value) and procedures (producing a result stream) related to:
  - Extensions of Cypher with, for instance, dynamic labels or property keys and periodic commits for all operations
  - Graph refactoring (cloning nodes, changing a relationship's starting or ending node, and so on)
  - Managing collections and lists
  - Database introspection (graph schema, types of properties, and so on)
  - Import from/export to files in different formats (JSON, XML, and so on)
- To install, download the right version (this is VERY important, must be the exact version) and copy it to the Plugins folder.
- In neo4j.conf:

```
dbms.security.procedures.unrestricted = apoc.*, …
dbms.security.procedures.allowlist =  apoc.*,  …
```

Useful Libraries

2. Graph Data Science (GDS previously graph-algo)

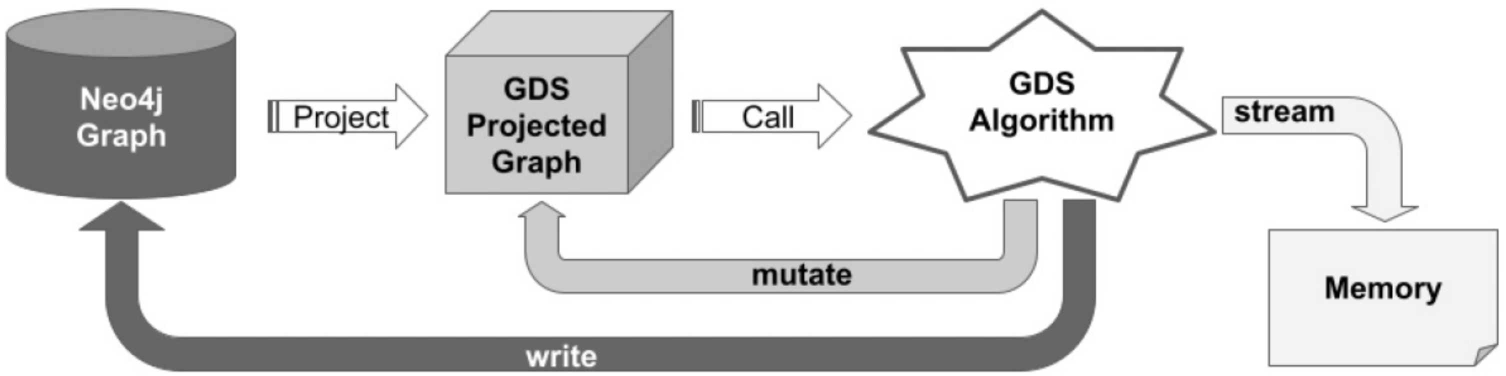
- Contains tools to be used in a data science project using data stored in Neo4j:
  - Path-related algorithms: Dijkstra, A*, etc.
  - Graph algorithms
    - Centrality
    - Community detection
    - Similarity
  - Machine learning (ML) models and pipelines: allow feature extraction using graph embedding
  - Python client: allows GDS to be called from Python, without using Cypher
- To install, download the right version and copy it into the Plugins folder.
- In neo4j.conf:

```
dbms.security.procedures.unrestricted=apoc.*,gds.*,n10s.*,.....
dbms.security.procedures.allowlist=apoc.coll.*,apoc.load.*,gds.*, apoc.*, n10s.*, …
```

Useful Libraries

2. Graph Data Science (cont.)

- Four modes of returning results:
  - Stream
  - Write
  - Mutate
  - Stats



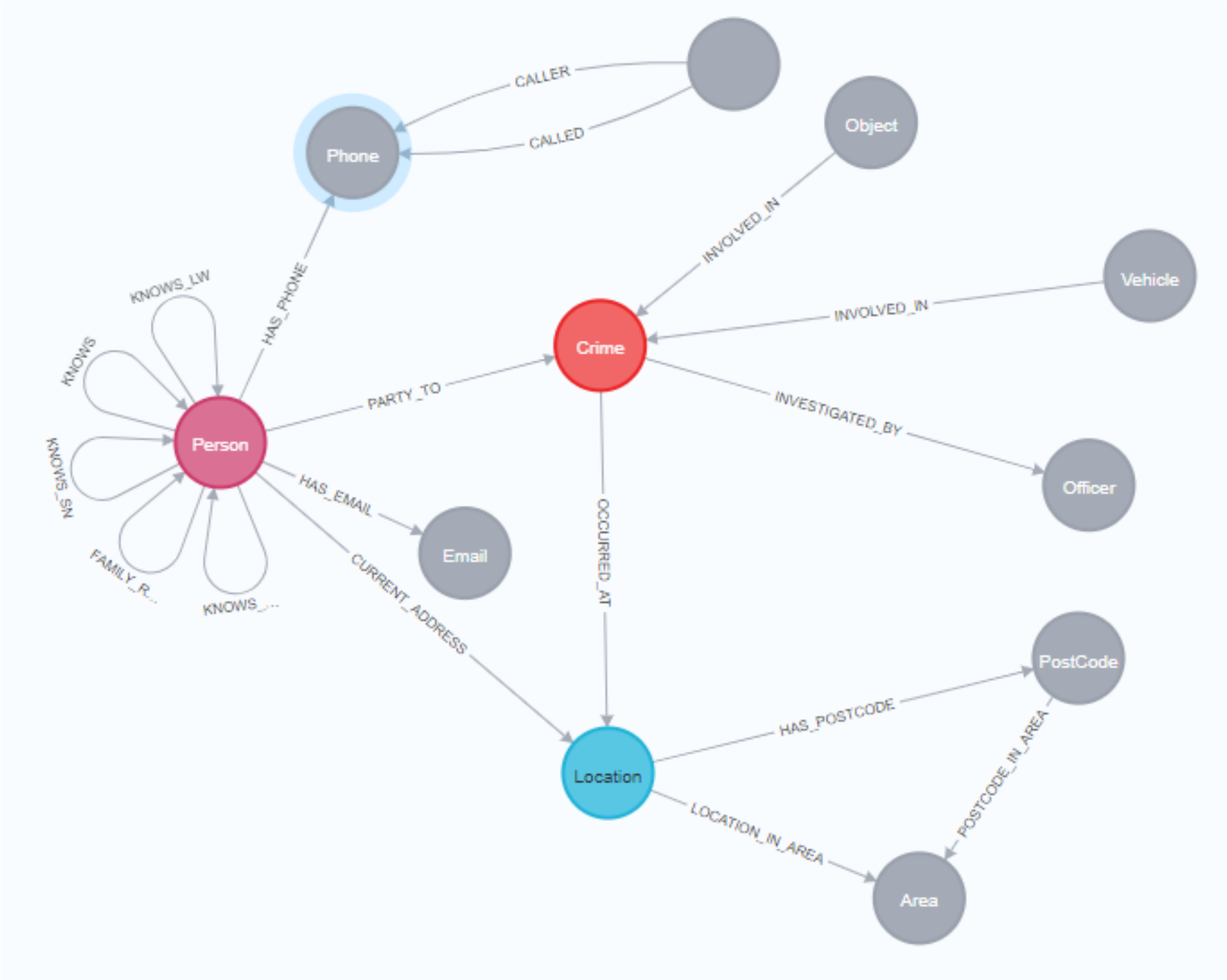
Problem 5 – Advanced queries and GDS – Crime

- The POLE data model is a standard approach in security use cases, that can also be applied in other areas
- Typical POLE use cases: Policing, Counter Terrorism, Border Control / Immigration, Insurance Fraud Investigations
- Crime data: downloaded from public sources (http://data.gov.uk), up to the block or street level and crime, **by month only** (not day or timestamp). **Only crime and location data,** not personal data
- This example uses street crime data **for Greater Manchester, UK from August 2017**
- Crime IDs, longitude, latitude, crime type, street/locale name, and last outcome values, taken from the public street crime data files. UK postcodes were retrieved from a public API using Longitude and Latitude,
- **Randomly generated or curated data was used for other entities in the database** (vehicles, officers, people, phone numbers, phone calls, emails, day of the month, etc.).

Problem 5 – Crime

Relationships: 'KNOWS', FAMILY_REL (related to), KNOWS_LW (lives with), KNOWS_PHONE (has a related phone call), and KNOWS_SN (social network).

Location is associated with Postcode and Area. In the UK, Postcodes are split into two sections - M1 1AA ('M1 1AA' is the Postcode, and 'M1' is the area). Postcode is typically limited to a street or a few blocks and Area, may cover a town or city neighborhood).



Problem 5 – Crime

1. Types of crimes

```
MATCH (c:Crime)
RETURN c.type AS crime_type, count(c) AS total
ORDER BY count(c) DESC
```

crime.db$ MATCH (c:Crime) RETURN c.type AS crime_type, count(…

Table

Text

Code

	crime_type	total
1	"Violence and sexual offences"	8765
2	"Public order"	4839

2. Top locations of crimes

```
MATCH (l:Location)<-[:OCCURRED_AT]-(:Crime)
RETURN l.address AS address,
       l.postcode AS postcode, count(l) AS total
ORDER BY count(l) DESC
LIMIT 15
```

	address	postcode	total
1	"Piccadilly"	"M1 1LU"	166
2	"Shopping Area"	"M60 1TA"	111
3	"Prison"	"M60 9AH"	48

Problem 5 – Crime

3. Crimes near an address

```
MATCH (l:Location {address: '1 Coronation Street', postcode: 'M5 3RW'})
WITH point(l) AS myaddress
MATCH (x:Location)-[:HAS_POSTCODE]->(p:PostCode), (x)<-[:OCCURRED_AT]-(c:Crime)
WITH x, p, c, point.distance(point(x), myaddress)  AS distance WHERE distance < 500
RETURN x.address AS address, p.code AS postcode, count(c) AS crime_total,
          COLLECT(distinct(c.type)) AS crime_type, distance
ORDER BY distance
LIMIT 10
```

	address	postcode	crime_total	crime_type
1	"1 Coronation Street"	"M5 3RW"	3	["Bicycle theft", "Violence and sexual offer
2	"158 Gloucester Street"	"M5 3SG"	1	["Public order"]
3	"147 West Crown Avenue"	"M5 3WT"	3	["Public order", "Shoplifting"]

Problem 5 – Crime

4. Crimes investigated by Inspector Morse

```
MATCH (o:Officer { 'Chief Inspector'  surname: 'Morse'})<-[i:INVESTIGATED_BY]-(c:Crime)
RETURN *
```

5. Drug crimes investigated by officer Larive

```
MATCH (c:Crime {last_outcome: 'Under investigation', type: 'Drugs’}) - [i:INVESTIGATED_BY] ->
          (o:Officer {surname: 'Larive’})
RETURN *
```

Problem 5 – Crime

6. Are pairs of persons associated with the Drug Crimes in the previous query, somehow connected in the graph?

We look for **all shortest paths** between them, having 3 or fewer hops along all types of 'KNOWS' relationships. We ignore the direction of the relationships, since we are not interested in which direction they point.

```
MATCH (c:Crime{last_outcome: 'Under investigation', type: 'Drugs'} ) - [:INVESTIGATED_BY] - >
        (o:Officer {badge_no: '26-5234182'}), (c)<-[:PARTY_TO]-(p:Person)
WITH COLLECT(p) AS persons
UNWIND persons AS p1
UNWIND persons AS p2
WITH * WHERE  p1.nhs_no < p2.nhs_no
MATCH path = allshortestpaths((p1)-[:KNOWS|KNOWS_LW|
        KNOWS_SN|FAMILY_REL|KNOWS_PHONE*..3]-(p2))
RETURN distinct [p in NODES(path)|(p.name+' '+p.surname)]
```

	[p in NODES(path)|(p.name+' '+p.surname)]
1	["Jack Powell", "Brian Morales", "Phillip Williamson", "Raymond Walker"]
2	["Jack Powell", "Alan Ward", "Phillip Williamson", "Raymond Walker"]
3	["Jack Powell", "Alan Ward", "Kathleen Peters", "Raymond Walker"]

Problem 5 – Crime

7. People who know someone who is involved in a crime

```
MATCH (p:Person) - [:KNOWS] - (friend) - [:PARTY_TO] -> (:Crime)
            WHERE NOT (p) - [:PARTY_TO] -> (:Crime)
RETURN    p.name AS name, p.surname AS surname, p.nhs_no AS id, count(distinct friend) AS dangerousFriends
ORDER BY dangerousFriends DESC
LIMIT 5
```

8. Number of persons who know someone who is involved in a crime, or who know a person who knows a person involved in a crime

```
MATCH (p:Person)-[:KNOWS*1..2]-(friend)-[:PARTY_TO]->(:Crime) WHERE NOT (p:Person)-[:PARTY_TO]->(:Crime)
RETURN friend.name AS suspectname, friend.surname AS suspectsurname,
            friend.nhs_no AS id, count(distinct p) AS friends
ORDER BY friends DESC
LIMIT 5
```

Problem 5 – Crime

9. List the names of the persons who know persons involved in crimes, living at less than 10 km from each other, and the distance between them

```
MATCH (vp:Person)-[k:KNOWS]-(fovp)-[pt:PARTY_TO]->(:Crime),
        (vp)-[:CURRENT_ADDRESS]->(vpaddress),
        (fovp)-[:CURRENT_ADDRESS]->(fovpaddress)
        WHERE  vp.nhs_no <> fovp.nhs_no AND NOT (vp)-[:PARTY_TO]->(:Crime)
WITH vp as vulnerable, fovp as suspect, vpaddress as vulnerableaddress, fovpaddress as suspectaddress
WITH vulnerable, suspect,  point(vulnerableaddress)  AS p1, point(suspectaddress) AS p2
WITH vulnerable.name as Name, suspect.name as Suspect, point.distance(p1,p2) as Distance
WHERE Distance < 10000
RETURN distinct Name, Suspect, Distance
```

Problem 5 – Crime

10. Shortest paths between vulnerable persons that are not suspects or have not committed a crime but who know people that have participated in a crime.

```
MATCH (p:Person)-[:KNOWS]-(friend)-[:PARTY_TO]->(:Crime)   WHERE NOT (p:Person)-[:PARTY_TO]->(:Crime)
WITH   p, count(distinct friend) AS dangerousFriends ORDER BY dangerousFriends DESC LIMIT 5
WITH  COLLECT (p) AS people
UNWIND people AS p1
UNWIND people AS p2
WITH * WHERE p1.nhs_no <> p2.nhs_no
MATCH   path = shortestpath((p1)-[:KNOWS*]-(p2))
RETURN [p in NODES(path)|(p.name+' '+p.surname)]
```

Problem 5 – Crime

11. List the names, ids and the number of number of acquanintances of the persons who are not involved in a crime, live with relatives who are not involved in a crime, but have friends involved in a crime.

```
MATCH (p:Person)-[:FAMILY_REL]-(relative)-[:KNOWS]-(famFriend)-[:PARTY_TO]-> (:Crime),
      (p)-[:CURRENT_ADDRESS]->(:Location)<-[:CURRENT_ADDRESS]-(relative)
      WHERE NOT (p:Person)-[:PARTY_TO]->(:Crime) AND
            NOT (relative)-[:PARTY_TO]->(:Crime)
RETURN p.name AS name, p.surname AS surname,
       p.nhs_no AS id, count(DISTINCT famFriend) AS DangerousFamilyFriends
ORDER BY DangerousFamilyFriends DESC
LIMIT 5
```

Graph Data Science Algorithms

GDS Workflow

- To use GDS we need to create a projected graph. **Projected graphs are stored in memory and not persisted**

- The projected graph may include
  - Only a certain node label(s)
  - Only a certain relationship type(s)
  - Only certain properties
  - New relationships computed on the fly
  - New properties computed on the fly

- Two ways of creating a projected graph:
  - **Native projection**: Nodes, relationships and properties are selected from the Neo4j database
  - **Cypher projection:** Entities are filtered from Neo4j or created on the fly, using Cypher queries

Graph Data Science

GDS Workflow

- Technique:
    - Create one or more projected graphs, selecting nodes and relationships from the data stored in Neo4j
    - Run one or multiple algorithms on this projected graph, retrieving the results either by streaming them, storing them in the projected graph, or persisting them by writing them back into the Neo4j database
- Projected graphs are created with:
    - For native projection `gds.graph.project`
    - For Cypher projection `gds.graph.project.cypher`

Graph Data Science

- Native projection

```
CALL gds.graph.project(
  <graphName>,
  <nodeProjectionConfig>,
  <relationshipProjectionConfig> )
```

- Cypher projection

```
CALL gds.graph.project.cypher(
 "projectedGraphCypher",
 "<Cypher query to select nodes>",
 "<Cypher query to select relationships>" )
```

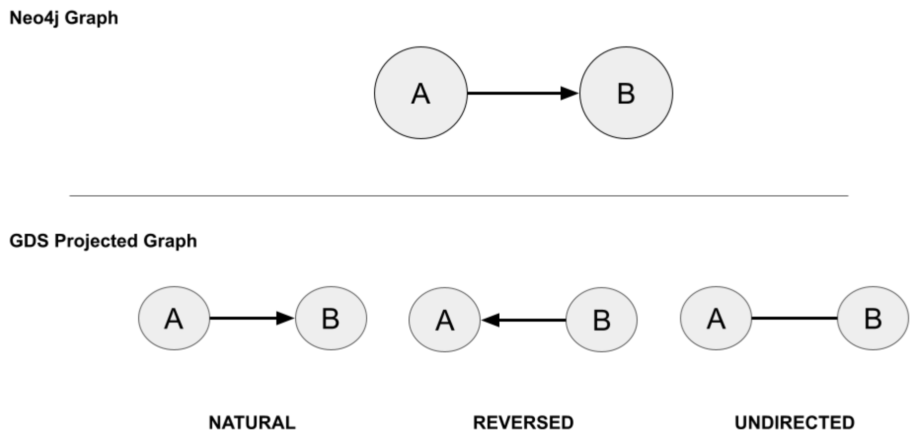
- We will use the above expressions in the examples that follow

Graph Data Science

- **Native projection**

```
CALL gds.graph.project(
  <graphName>,
  <nodeProjectionConfig>,
  <relationshipProjectionConfig> )
```

- Some algorithms require a projected graph to be undirected. We can control the orientation of the relationships in a projected graph with the **orientation** property.
- Default for projected graphs: the same orientation as in the source graph, called NATURAL
- We can reverse the original graph orientation, using orientation = REVERSE, or change to UNDIRECTED



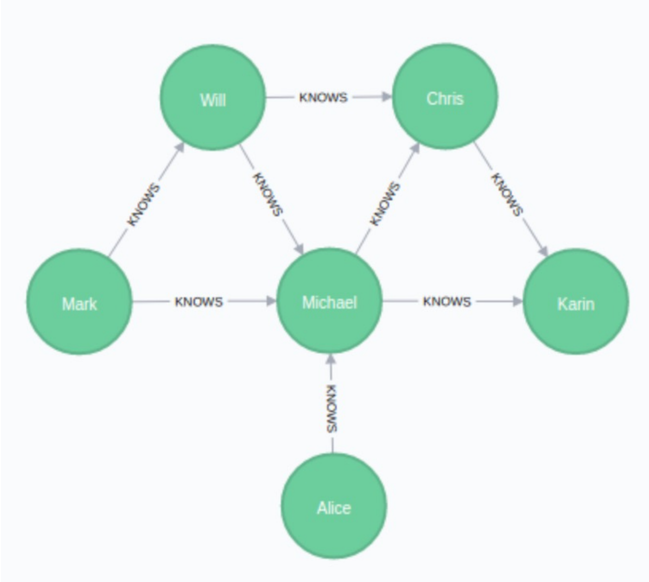
Problem 5 – Graph Data Science Algorithms - Crime

- We now project **Person** nodes and **KNOWS** relationships in the in-memory graph, which we call ”social"

```
CALL gds.graph.project ('social',  'Person', {KNOWS: {orientation:'UNDIRECTED'}})
```

To check the projected graphs: Call gds.graph.list. The triangle count algorithm requires undirected graphs

12. The triangle count algorithm returns 'triangles' of connected nodes. The algorithm counts the number of triangles for each node in the graph. A triangle is a set of three nodes where each node has a relationship to the other two**. In graph theory terminology, this is sometimes referred to as a 3-clique**.



- In the example, Michael is in 3 triangles, Karin in one, Chris in 2

Problem 5 – Graph Data Science Algorithms - Crime

12. The triangle count algorithm

In this case, the algorithm finds groups of three Persons where every node in the group 'KNOWS' the others ('A' knows 'B' knows 'C' knows 'A'). Now, we can identify Person nodes in the ”social" graph, who are members of the largest number of triangles.

```
CALL gds.triangleCount.stream('social')  YIELD nodeId, triangleCount as triangles
WITH gds.util.asNode(nodeId) AS node, triangles
RETURN node.name AS name,
        node.surname AS surname,
        node.nhs_no AS id, triangles
ORDER BY triangles DESC
LIMIT 10;
```

	name	surname	id	triangles
1	"Deborah"	"Ford"	"838-45-9343"	10
2	"Phillip"	"Perry"	"884-33-9676"	9
3	"Peter"	"Bryant"	"245-63-7539"	8

Problem 5 – Graph Data Science Algorithms - Crime

13. List the people in each triangle (in total or in pairs)

```
CALL gds.alpha.triangles ('social')  YIELD  nodeA,  nodeB, nodeC
WITH  gds.util.asNode(nodeA) AS node1, gds.util.asNode(nodeB)  as node2, gds.util.asNode(nodeC) as node3
RETURN node1.name + ' '  + node1.surname, node2.name+ ' ' + node2.surname, node3.name+ ' ' +
node3.surname, count(*)
```

node1.name + ' ' + node1.surname	node2.name+ ' ' + node2.surname	node3.name+ ' ' + node3.surname	count(*)
"Mary Young"	"Stephanie Hughes"	"Pamela Gibson"	1
"Wanda Webb"	"Kevin Hawkins"	"Jennifer Gray"	1
"Phyllis Murray"	"Joshua Black"	"Wanda Weaver"	1

Problem 5 – Graph Data Science Algorithms - Crime

13. List the people in each triangle (in total or in pairs)

```
CALL  gds.alpha.triangles ('social') YIELD  nodeA,  nodeB, nodeC
WITH gds.util.asNode(nodeA) AS node1, gds.util.asNode(nodeB)  as node2, gds.util.asNode(nodeC) as node3
WITH apoc.coll.sort([node1.name,node2.name,node3.name]) as col
// Builds collections of three nodes and sorts them for grouping
RETURN col, count(*)
```

col	count(*)
["Mary", "Pamela", "Stephanie"]	1
["Bobby", "Brian", "Craig"]	1
["Linda", "Philip", "Rebecca"]	1

Problem 5 – Graph Data Science Algorithms - Crime

13. List the people in each triangle (in total or in pairs)

```
CALL  gds.alpha.triangles ('social') YIELD  nodeA,  nodeB, nodeC
WITH gds.util.asNode(nodeA) AS node1, gds.util.asNode(nodeB)  as node2, gds.util.asNode(nodeC) as node3
WITH apoc.coll.sort([node1.name,node2.name,node3.name]) as col
WITH apoc.coll.pairs(col) as col1 // apoc.coll.pairs([1,2,3]) returns [1,2],[2,3],[3,null]
UNWIND col1 as  col2
RETURN col2,count(*) ORDER BY count(*) DESC
```

col2	count(*)
["Phillip", *null*]	10
["Peter", *null*]	5
["Rachel", *null*]	4
["Peter", "Phillip"]	4

Problem 5 – Graph Data Science Algorithms - Crime

14. The previous query was run against the **entire graph.** We can use the same algorithm on a sub-graph containing only people who are associated with crimes. This returns a different set of triangles, consisting only of people associated with crimes who appear in communities/clusters.

Project a subgraph that contains only people associated with crimes:

```
MATCH (p0:Person)-[:PARTY_TO]->(:Crime)
WITH    COLLECT (DISTINCT p0.nhs_no) as criminalPartyIds
MATCH (s:Person)-[r:KNOWS]->(t:Person)
WHERE (s.nhs_no  IN criminalPartyIds) AND (t.nhs_no  IN criminalPartyIds)  //only connected criminals
WITH     gds.graph.project( 'crime-associates', s, t,  {sourceNodeLabels:labels(s), targetNodeLabels:labels(t),
         relationshipType:type(r)}, {undirectedRelationshipTypes:[type(r)]}) AS g
RETURN g.graphName AS graph, g.nodeCount AS nodes, g.relationshipCount AS rels
```

Problem 5 – Graph Data Science Algorithms - Crime

15. Now we run Query 12 against this graph and compare results

```
CALL gds.triangleCount.stream('crime-associates')
YIELD nodeId, triangleCount as triangles
WITH gds.util.asNode(nodeId) AS node, triangles
RETURN node.name AS name, node.surname AS surname, node.nhs_no AS id, triangles
ORDER BY triangles DESC
LIMIT 5;
```

	name	surname	id	triangles
1	"Deborah"	"Ford"	"838-45-9343"	10
2	"Phillip"	"Perry"	"884-33-9676"	9
3	"Peter"	"Bryant"	"245-63-7539"	8

Query 12

	name	surname	id	triangle
1	"Phillip"	"Williamson"	"337-28-4424"	4
2	"Brian"	"Morales"	"335-71-7747"	3
3	"Alan"	"Ward"	"881-20-2396"	3

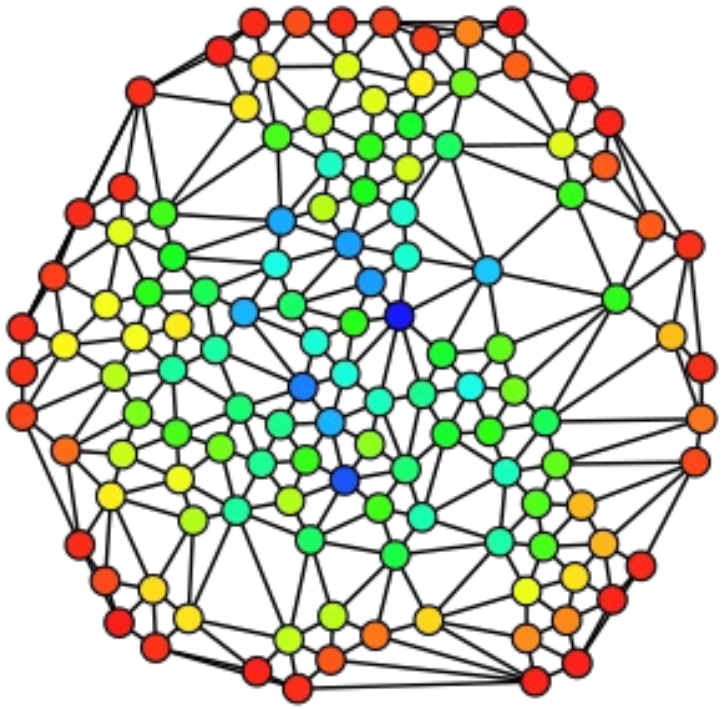
Query 15

Problem 5 – Graph Data Science Algorithms - Crime

16. The betweenness centrality *calculates the nodes in the shortest paths between all pairs of nodes in a graph.* Each node receives a score, **based on the number of shortest paths that pass through the node**. The higher this number, the higher the betweenness centrality score. The idea is to identify central nodes or 'bridge' nodes between communities in the graph.

```
CALL gds.betweenness.stream('social')  YIELD nodeId, score AS centrality
WITH gds.util.asNode(nodeId) AS node, centrality
RETURN node.name AS name, node.surname AS surname,
         node.nhs_no AS id, toInteger(centrality) AS score
ORDER BY centrality DESC LIMIT 10;
```

	name	surname	id	score
1	"Annie"	"Duncan"	"863-96-9468"	5275
2	"Ann"	"Fox"	"576-99-9244"	5116
3	"Amanda"	"Alexander"	"893-63-6176"	4599

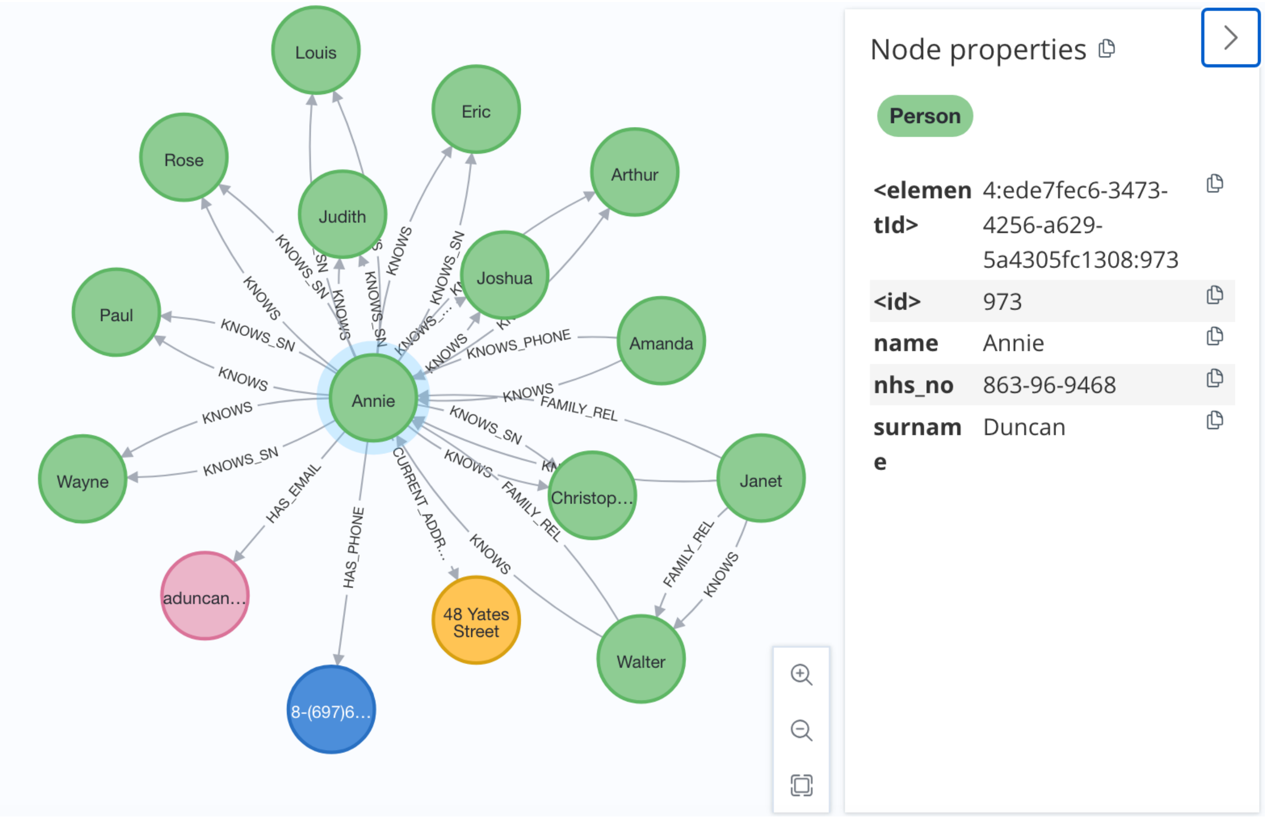


Problem 5 – Graph Data Science Algorithms - Crime

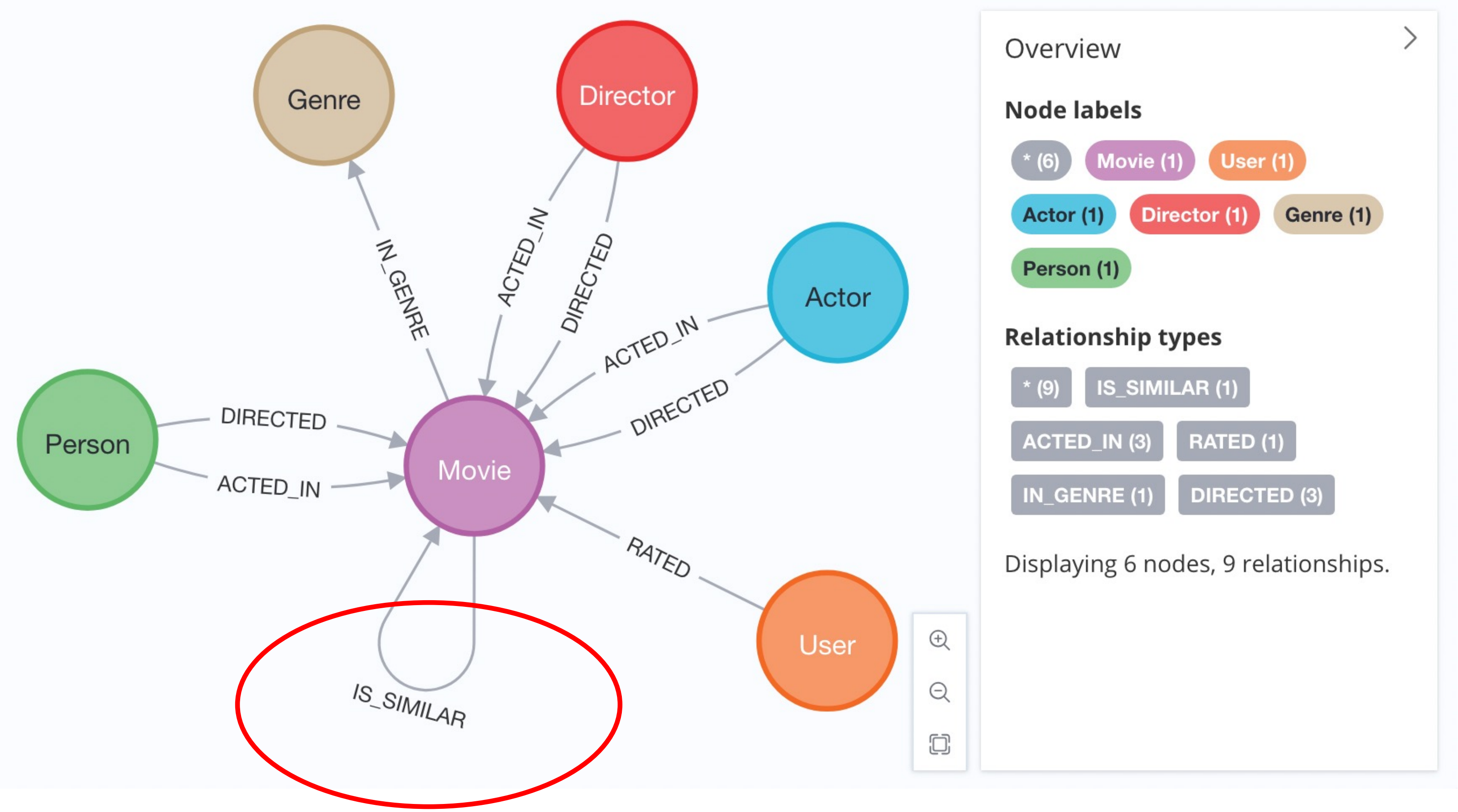
17. Compute the most influential node based on the centrality algorithm

```
CALL gds.betweenness.stream('social')
YIELD nodeId, score AS centrality
WITH gds.util.asNode(nodeId) AS node, centrality
WITH node, toInteger(centrality) AS score
ORDER BY centrality DESC
LIMIT 1 // this returns the most influential node
MATCH (node) -[r]-(s)
RETURN node,s
```

We can see that the most central node is Annie Duncan.



Problem 6 – Recommender Systems - Movies



Problem 6 – Recommender Systems

- Recommender Systems systems: **Collaborative Filtering and Content-Based Filtering**
- **Content-Based Filtering**
  - An information retrieval method that uses item features to select and return items relevant to a user's query. This method often takes features of other items in which a user expresses interest into account.
- **Collaborative Filtering**
  - An information retrieval method that recommends items to users based on how other users with similar preferences and behavior have interacted with that item

Problem 6 – Recommender Systems

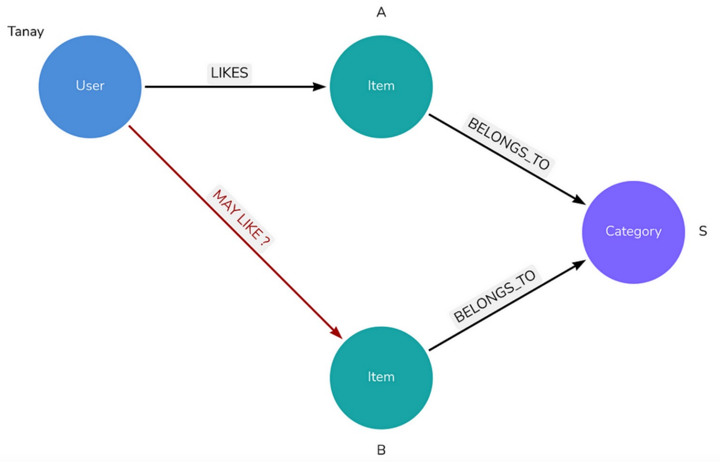
- Recommender Systems systems: **Content-Based and Collaborative Filtering**

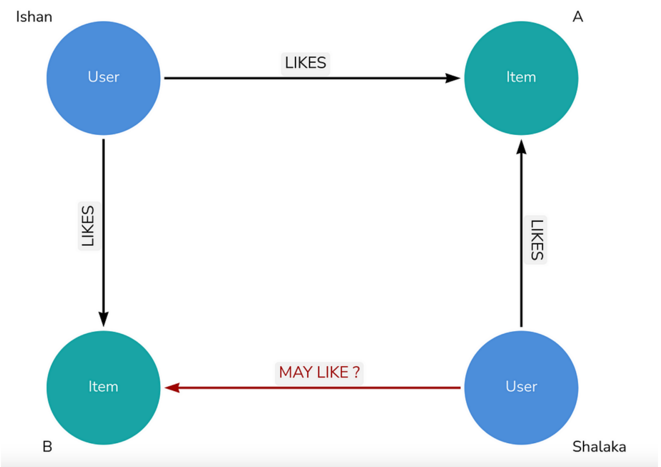
- **Content-Based Filtering**

  - People generally prefer to eat food belonging to specific cuisine(s) or like to watch movies of a specific genre(s).
  - If a user has liked a comedy movie, then there is a possibility that the user will like another movie that also belongs to the comedy genre.

- **Collaborative Filtering**

  - Users collaborate on a single entity, i.e., order the same item(s), watch the same movie(s), etc.
  - Similarity based on the fact that they are interested in the same things and have some similar interests.





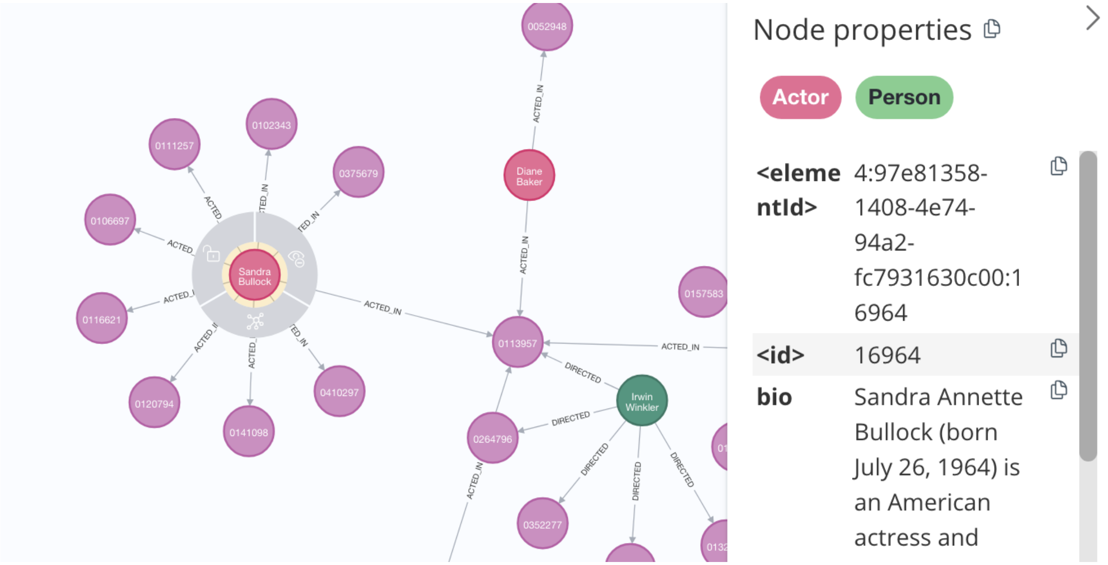
Problem 6 – Recommender Systems

1. Collaborative filtering – For each movie, find the number of users the users who rated that movie and had also rated 'Crimson Tide

MATCH (m:Movie {title: 'Crimson Tide'})<-[:RATED]- (u:User)-[:RATED]->(rec:Movie)
WITH rec, COUNT(*) AS usersWhoAlsoRated
ORDER BY usersWhoAlsoRated DESC LIMIT 25
RETURN rec.title AS title, usersWhoAlsoRated

2. Content-based filtering – List the movies similar to 'The Net' based on co-actors, director and genre.

MATCH p=(m:Movie {title: 'Net, The'}) –
[:ACTED_IN|IN_GENRE|DIRECTED*2]-()
RETURN p LIMIT 25



Problem 6 – Content-based Filtering

3. Similarity Based on Common Genres - Find movies most similar to “Inception” based on shared genres

```
MATCH (m:Movie)-[:IN_GENRE]->(g:Genre) <-[:IN_GENRE]-(rec:Movie) WHERE m.title = 'Inception'
WITH rec, COLLECT (g.name) AS genres, count(*) AS commonGenres
RETURN rec.title, genres, commonGenres
ORDER BY commonGenres DESC
LIMIT 10;
```

	rec.title	genres
1	"Patlabor: The Movie (Kidô keisatsu patorebâ: The Movie)"	["Crime", "Drama", "Mystery", "Sci-Fi", "Thril
2	"Strange Days"	["Crime", "Drama", "Mystery", "Sci-Fi", "Thril
3	"Watchmen"	["Drama", "Mystery", "Sci-Fi", "Thriller", "IMA

Problem 6 – Content-based Filtering

4. Personalized Recommender Systems Based on Genres – Find the movies with the same genre than the ones a given user has rated, sorted by score, where the score of each movie is the sum of all the genres in common with the movies that the user has rated.

```
MATCH (u:User {name: 'Omar Huffman'})-[r:RATED] -> (m:Movie), (m)-[:IN_GENRE]->(g:Genre)<-[:IN_GENRE]-
        (rec:Movie) WHERE NOT EXISTS{ (u)-[:RATED]->(rec) }
WITH rec, g.name as genre, count(*) AS count
 //movies not rated by Omar, their genre g, and the # of movies of genre g g rated by Omar (only title shown)
```

rec.title	genre
"Boxtrolls, The"	"Adventure"
"Boxtrolls, The"	"Adventure"

		genre	count
1	"Boxtrolls, The"	"Adventure"	9
2	"The Book of Life"	"Adventure"	9

Problem 6 – Content-based Filtering

4. Personalized Recommender Systems Based on Genres – Find the movies with the same genre than the ones a given user has rated, sorted by score, where the score of each movie is the sum of all the genres in common with the movies that the user has rated.

```
MATCH (u:User {name: 'Omar Huffman'})-[r:RATED] -> (m:Movie), (m)-[:IN_GENRE]->(g:Genre)<-[:IN_GENRE]-
        (rec:Movie) WHERE NOT EXISTS{ (u)-[:RATED]->(rec) }
WITH rec, g.name as genre, count(*) AS count
 //movies not rated by Omar, their genre g, and the # of movies of genre g g rated by Omar
WITH rec, COLLECT([genre, count]) AS scoreComponents  // same movies, with their genres and the count above
RETURN rec.title AS movie, reduce(s=0, x in scoreComponents | s+x[1]) AS score,
        rec.year AS year, scoreComponents
ORDER BY score DESC LIMIT 10
```

movie	score	year	scoreComponents
"Motorama"	38	1991	[["Adventure", 9], ["Comedy", 5], ["Fantasy", 4], ["Drama", 7], ["Crime", 2], ["Thriller", 6], ["Sci-Fi", 5]]
"Rubber"	37	2010	[["Adventure", 9], ["Comedy", 5], ["Drama", 7], ["Action", 5], ["Crime", 2], ["Thriller", 6], ["Horror", 2], ["Western", 1]]
"Interstate 60"	36	2002	[["Adventure", 9], ["Comedy", 5], ["Fantasy", 4], ["Drama", 7], ["Thriller", 6], ["Sci-Fi", 5]]

Problem 6 – Content-based Filtering

5. Weighted Content Algorithm – Given a movie a user has watched, find the first 25 movies to recommend to this user, based on a score that accounts for the number of actors (a weight of 3), genre (a weight of 5) and directors (a weight of 4) they have in common.

```
MATCH (m:Movie) WHERE m.title = 'American President, The'
MATCH (m)-[:IN_GENRE]->(g:Genre)<-[:IN_GENRE]-(rec:Movie) // similar movies by common genres
WITH m, rec, count(*) AS gs // # of common genres for each movie matching the genre of 'The American President'
OPTIONAL MATCH (m)<-[:ACTED_IN]-(a)-[:ACTED_IN]->(rec)
WITH m, rec, gs, count(a) AS as //common actors
OPTIONAL MATCH (m)<-[:DIRECTED]-(d)-[:DIRECTED]->(rec)
WITH m, rec, gs, as, count(d) AS ds //common directors
RETURN rec.title AS movie, (5*gs)+(3*as)+(4*ds) AS score
ORDER BY score DESC LIMIT 25
```

Problem 6 – Content-based Filtering - Similarity

6. Content-Based Similarity Metrics - The Jaccard index is a number between 0 and 1 that indicates how similar two sets are. The Jaccard index of two identical sets is 1. If two sets do not have a common element, then the Jaccard index is 0. The Jaccard is calculated by dividing the size of the intersection of two sets by the size of the union of the two sets. **Compute the movies similar to "Inception" based on Jaccard similarity of genres.**

```
MATCH (m:Movie {title:'Inception'})-[:IN_GENRE]-> (g:Genre)<-[:IN_GENRE]-(other:Movie)
WITH m, other, count(g) AS intersection, COLLECT(g.name) as common
// mg: genres of inception; og: genres of movies sharing at least one in common with inception
WITH m, other, intersection, common,  [(m)-[:IN_GENRE]->(mg) | mg.name] AS set1,
                                       [(other)-[:IN_GENRE]->(og) | og.name] AS set2
WITH m, other, intersection, common, set1, set2, set1+[x IN set2 WHERE NOT x IN set1] AS union
RETURN  ((1.0*intersection)/size(union)) AS jaccard, m.title, other.title, common, set1,set2
ORDER BY jaccard DESC LIMIT 25
```

jaccard	m.title	other.title	common	set1
0.8571428571428571	"Inception"	"Strange Days"	["Crime", "Drama", "Mystery", "Sci-Fi", "Thriller", "Action"]	["Crime", "Dra
0.8571428571428571	"Inception"	"Watchmen"	["Drama", "Mystery", "Sci-Fi", "Thriller", "IMAX", "Action"]	["Crime", "Dra

Problem 6 – Recommender Systems - Similarity

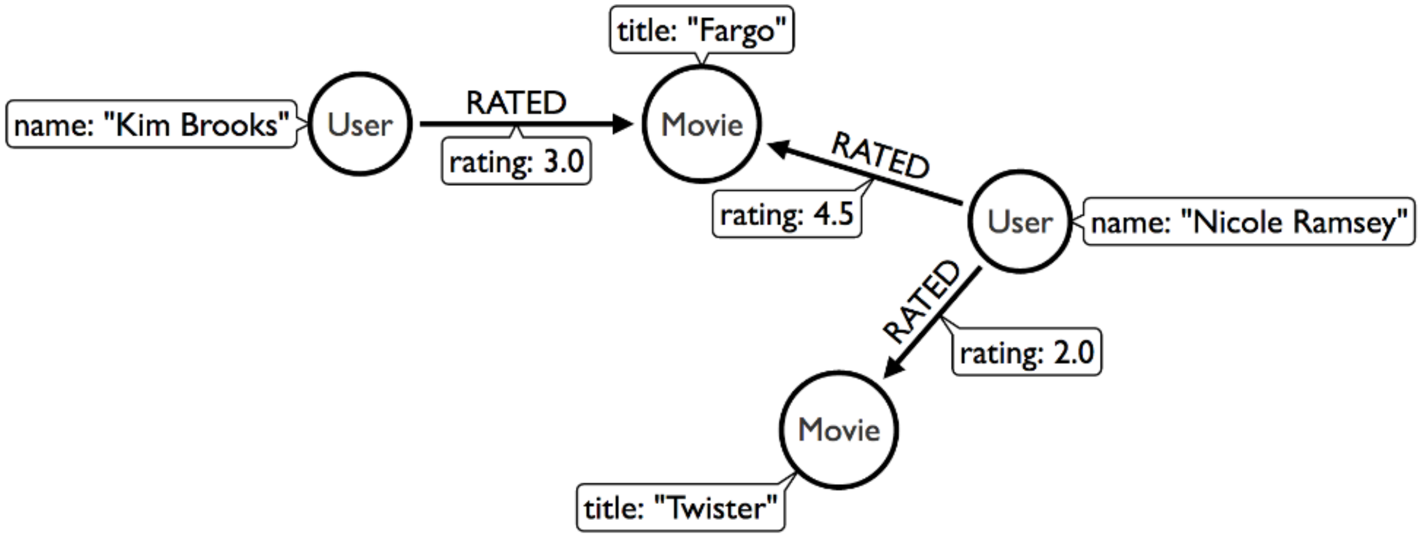
7. Content-Based Similarity Metrics - Compute the movies similar to "Inception" based on Jaccard similarity of all features (actors, director, genre)

```
MATCH (m:Movie {title: 'Inception'}) -
        [:IN_GENRE|ACTED_IN|DIRECTED]- (t) <-[:IN_GENRE|ACTED_IN|DIRECTED]-(other:Movie)
WITH    m, other, count(t) AS intersection, collect(t.name) AS common,
        [(m)-[:IN_GENRE|ACTED_IN|DIRECTED]- (mt) | mt.name]   AS set1,
        [(other)-[:IN_GENRE|ACTED_IN|DIRECTED]-(ot) | ot.name] AS set2
WITH m, other, intersection, common, set1, set2,   set1 + [x IN set2 WHERE NOT x IN set1] AS union
RETURN  ((1.0*intersection)/size(union)) AS jaccard, m.title, other.title, common, set1,set2
ORDER BY jaccard DESC LIMIT 25
```

jaccard	m.title	other.title	set1
0.4166666666666667	"Inception"	"Sherlock: The Abominable Bride"	["Crime", "Drama", "Mystery", "Sci-Fi", "Thriller", "IMAX", "Actic
0.35294117647058826	"Inception"	"Watchmen"	["Crime", "Drama", "Mystery", "Sci-Fi", "Thriller", "IMAX", "Actic
0.35294117647058826	"Inception"	"Strange Days"	["Crime", "Drama", "Mystery", "Sci-Fi", "Thriller", "IMAX", "Actic

Problem 6 – Collaborative Filtering

1. Find users similar to a user in the network
2. Assuming that similar users have similar preferences, what are the movies those similar users like?



Problem 6 – Collaborative Filtering

8. Simple Collaborative Filtering. Compute the movies not rated by a given user, but rated by someone else

```
MATCH (u:User {name: 'Cynthia Freeman'})-[:RATED]->  (:Movie) <- [:RATED]- (peer:User)
MATCH (peer)-[:RATED]->(rec:Movie) WHERE NOT EXISTS { (u)-[:RATED]->(rec) }
RETURN distinct rec.title, rec.year, rec.plot LIMIT 25
```

rec.title	rec.year	rec.plot
""Great Performances" Cats"	*null*	*null*
"$9.99"	2008	"A stop-motion an
"'Hellboy': The Seeds of Creation"	2004	"In-depth docume

Problem 6 – Collaborative Filtering

9. Simple Collaborative Filtering. Compute users that gave at least one rating similar to the one given by our user. Then, compute the ratings given by those users to movies not rated by our target user such that these ratings are higher than 4.8

```
MATCH (u:User {name: 'Cynthia Freeman'})-[r1:RATED]-> (:Movie)<-[r2:RATED]-(peer:User)
WHERE abs(r1.rating-r2.rating) < 2 // similarly rated
WITH distinct u, peer
MATCH (peer)-[r3:RATED]->(rec:Movie) WHERE r3.rating > 4.8 AND NOT EXISTS { (u)-[:RATED]->(rec) }
WITH rec, count(*) as freq, avg(r3.rating) as rating
RETURN rec.title, rec.year, rating, freq, rec.plot
ORDER BY rating DESC, freq DESC  LIMIT 25
```

rec.title	rec.year	rating	freq	rec.plot
"Pulp Fiction"	1994	5.0	127	"The lives of two mob
"Star Wars: Episode IV - A New Hope"	1977	5.0	112	"Luke Skywalker joins
"Forrest Gump"	1994	5.0	98	"Forrest Gump, while

Problem 6 – Collaborative Filtering

10. Simple Collaborative Filtering. For a given user, in which genres he had given a higher-than-average rating? Use this to score similar movies.

```
MATCH (u:User {name: 'Andrew Freeman'})-[r:RATED]->(m:Movie) // compute mean rating
WITH u, avg(r.rating) AS mean
// find genres with higher than average rating and their number of rated movies
MATCH (u)-[r:RATED]->(m:Movie)  -[:IN_GENRE]->(g:Genre) WHERE r.rating > mean
WITH u, g, count(*) AS score // The user and the # of movies rated by him, by genre
// find movies with the genres above,  not yet  watched by our user
MATCH (g) <- [:IN_GENRE] - (rec:Movie) WHERE NOT EXISTS { (u)-[:RATED]->(rec) }
RETURN rec.title AS recmovie, rec.year AS year,  sum(score) AS sscore, collect(DISTINCT g.name) AS genres
ORDER BY sscore DESC LIMIT 10
```

	recmovie	year	sscore	genres
1	"Mars Needs Moms"	2011	155	["IMAX", "Sci-Fi", "Action", "Adventure", "Cor
2	"Wonderful World of the Brothers Grimm, The"	1962	155	["Adventure", "Romance", "Musical", "Fantasy

Problem 6 – Collaborative Filtering

Collaborative Filtering – Similarity Metrics.

We use similarity metrics to quantify how similar two users or two items are. Jaccard similarity is not enough to consider weights for movie ratings. We use cosine similarity in this case.

Cosine Similarity - The cosine similarity of two users indicates how similar two users' preferences for movies are. Users with a high cosine similarity will have similar preferences. (the cosine of the angle between two vectors)

$$similarity(A, B) = \frac{A \cdot B}{\|A\| \times \|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \times \sqrt{\sum_{i=1}^{n} B_i^2}}$$

Problem 6 – Collaborative Filtering

11. Collaborative Filtering – Cosine Similarity - Find similar users using cosine similarity

```
MATCH (p1:User )-[x:RATED] -> (m:Movie) <- [y:RATED]-(p2:User)
WITH    p1, p2, count(m) AS numbermovies, sum(x.rating * y.rating) AS xyDotProduct,
            COLLECT(x.rating) as xRatings, COLLECT(y.rating) as yRatings
             WHERE numbermovies > 10
WITH p1, p2, xyDotProduct,
         sqrt(reduce(xDot = 0.0, a IN xRatings | xDot + a^2)) AS xLength,
         sqrt(reduce(yDot = 0.0, b IN yRatings | yDot + b^2)) AS yLength
RETURN p1.name, p2.name, xyDotProduct / (xLength * yLength) AS sim
ORDER BY sim DESC
LIMIT 100;
```

p1.name	p2.name	sim
"Jonathan Cobb"	"Jackie Bradford"	1.0
"Jackie Bradford"	"Jonathan Cobb"	1.0
"Kathleen Cordova"	"Michael Johnson"	0.999634

Problem 6 – Collaborative Filtering

12. Collaborative Filtering – Cosine Similarity - Find the users with the most similar preferences to Cynthia Freeman, using the Cosine Similarity algorithm in the Neo4j Graph Data Science Library

```
MATCH (p1:User {name:'Cynthia Freeman'})-[x:RATED]->(movie)<-[x2:RATED]-(p2:User)
WHERE p2 <> p1
WITH p1, p2,COLLECT(x.rating) AS p1Ratings, COLLECT(x2.rating) AS p2Ratings WHERE size(p1Ratings) > 10
RETURN p1.name AS from, p2.name AS to,
        gds.similarity.cosine(p1Ratings, p2Ratings) AS similarity
ORDER BY similarity DESC
```

	from	to	similarity
1	"Cynthia Freeman"	"Roy Sweeney"	0.9897493266497985
2	"Cynthia Freeman"	"Jessica Leblanc"	0.9859211437470902
3	"Cynthia Freeman"	"Lori Cooper"	0.9854212747359109

Problem 6 – Collaborative Filtering

Collaborative Filtering – Pearson Similarity.

Pearson similarity, or Pearson correlation, is a similarity metric, appropriate for product Recommender Systemss because it takes into account the fact that different users will have different **mean ratings**.

The Pearson correlation coefficient is a measures the linear correlation between two sets of data. It is the ratio between the covariance of two variables and the product of their standard deviations.

It accounts for the fact that some users will tend to give higher ratings than others, because it considers differences w.r.t. the mean

$$\frac{\sum_{i=1}^{n}(A_i - \bar{A})(B_i - \bar{B})}{\sqrt{\sum_{i=1}^{n}(A_i - \bar{A})^2 \sum_{i=1}^{n}(B_i - \bar{B})^2}}$$

Problem 6 – Collaborative Filtering

13. Collaborative Filtering – Pearson Similarity. Find users most similar to Cynthia Freeman, according to Pearson similarity, using the [Pearson Similarity algorithm] in the Neo4j Graph Data Science Library.

```
MATCH (p1:User{name:'Cynthia Freeman'} )-[x:RATED]->(movie)<-[x2:RATED]-(p2:User)
WHERE p2 <> p1
WITH p1, p2, collect(x.rating) AS p1Ratings, collect(x2.rating) AS p2Ratings  WHERE size(p1Ratings) > 10
RETURN p1.name AS from,  p2.name AS to,
            gds.similarity.pearson(p1Ratings, p2Ratings) AS similarity
ORDER BY similarity DESC
```

	from	to	similarity
1	"Cynthia Freeman"	"Katelyn Morgan"	0.9119502320886002
2	"Cynthia Freeman"	"Jessica Leblanc"	0.7940236214913066
3	"Cynthia Freeman"	"Guy Davis"	0.737948455797911

Problem 6 – Collaborative Filtering

Neighborhood-Based Recommender Systems. kNN – K-Nearest Neighbors

We now want to answer this question:

"Who are the 10 users with tastes in movies most similar to mine? What movies that I haven't watched yet, have they rated highly?"

For this, we need to compute the K-Nearest Neighbors

Compute the kNN-based movie recommender system using Pearson similarity

Problem 6 – Collaborative Filtering

14. Collaborative Filtering – Compute the kNN movie-based recommendation using Pearson similarity

. First find the ten users most similar to Cynthia. Then, recommend her the movies rated by these users.

```
MATCH (p1:User {name: 'Cynthia Freeman'})-[x:RATED]->(movie)<-[x2:RATED]-(p2:User) WHERE p2 <> p1
WITH p1, p2, COLLECT(x.rating) AS p1Ratings, COLLECT(x2.rating) AS p2Ratings WHERE size(p1Ratings) > 10
WITH  p1,  p2, gds.similarity.pearson (p1Ratings, p2Ratings) AS pearson  ORDER BY pearson DESC LIMIT 10
MATCH (p2)-[r:RATED]->(m:Movie) WHERE NOT EXISTS( (p1)-[:RATED]->(m) )
RETURN m.title, SUM(pearson * r.rating) AS score  ORDER BY score DESC LIMIT 25
```

m.title	score
"Silence of the Lambs, The"	27.121527468609806
"Forrest Gump"	26.22376269405463

Problem 6 – Stats, Mutate and Write Modes

Content-based Filtering based on similarity

We first check the values of the metrics we are going to use

```
MATCH (n)
WITH n, ['year','imdbRating','runtime'] AS metrics
UNWIND metrics as metric
// for each metric and movie, the value of the metric
WITH metric, n[metric] AS value, n.title as title
        WHERE n.title IS NOT NULL
RETURN metric, value, title
```

metric	value	title
"year"	1995	"Toy Story"
"imdbRating"	8.3	"Toy Story"
"runtime"	81	"Toy Story"
"year"	1995	"Jumanji"
"imdbRating"	6.9	"Jumanji"

Problem 6 – Stats, Mutate and Write Modes

Content-based Filtering based on similarity

Compute the overall statistics

```
MATCH (n)
WITH n, ['year','imdbRating','runtime'] AS metrics
UNWIND metrics as metric
WITH metric, n[metric] AS value, n.title as title
        WHERE n.title IS NOT NULL
RETURN  metric, min(value) AS minValue,
             percentileCont(value, 0.25) AS percentile25,
             percentileCont(value, 0.50) AS percentile50,
             percentileCont(value, 0.75) AS percentile75,
             max(value) AS maxValue
```

metric	minValue	percentile25	percentile50	percentile75	maxValu
"year"	1902	1984.0	1997.0	2006.0	2016
"imdbRating"	1.6	6.1	6.9	7.5	9.6
"runtime"	2	93.0	102.0	115.0	910

Problem 6 – Stats, Mutate and Write Modes

Content-based Filtering based on similarity

Create a project graph with the three properties: 'year','imdbRating','runtime'

```
CALL gds.graph.project('Movies', {Movie: { properties: ['year','imdbRating','runtime'] }}, '*');
```

Only **numeric** properties can be projected (e.g., if we add "title" we get an error". Then, we analyze the data distribution. We compute the statistic values of these properties.

Problem 6 – Stats, Mutate and Write Modes

15. Content-based Filtering – Node similarity – Stats mode

We analyze the similarity distribution using KNN (https://neo4j.com/docs/graph-data-science/current/algorithms/knn/)\

The **k-nearest neighbors graph** (**k-NNG**) is a graph in which two vertices p and q are connected by an edge, if the distance between p and q is among the k-th smallest distances from p to other objects from P.

The Neo4j K-Nearest Neighbors algorithm computes a distance value for all node pairs, and creates new relationships between each node and its k nearest neighbors. The distance is calculated based on node properties. The algorithm compares given properties of each node. The k nodes where these properties are most similar are the k-nearest neighbors. The similarity of two neighbors is the mean of the similarities of the individual properties.

The algorithm only compares a sample of all possible neighbors on each iteration. The initial set of neighbors is selected at random. This is controlled with the configuration parameter sampleRate. TopK: The number of neighbors to find for each node.

Here is the algorithm that Neo4j implements: https://dl.acm.org/doi/abs/10.1145/1963405.1963487

Problem 6 – Stats, Mutate and Write Modes

15. Content-based Filtering – Node similarity – Stats mode

```
CALL gds.knn.stats("Movies", {nodeProperties:{runtime:"EUCLIDEAN", imdbRating:"EUCLIDEAN",
        year:"Jaccard"}, topK:15, sampleRate: 0.5, randomSeed:42, concurrency:1 } ) YIELD similarityPairs,
        similarityDistribution, nodePairsConsidered, configuration
RETURN configuration, nodePairsConsidered, similarityPairs, similarityDistribution
```

configuration	nodePairsConsidered	similarityPairs	similarityDistribution
{ "randomSeed": 42, "jobId": "20bae780-90f6-4551- bd86-bc8c3065dc7b", "deltaThreshold": 0.001,	14846917	136875	{ "min": 0.3333320617675781, "p5": 0.6206340789794922, "max": 1.000007629394531,

Problem 6 – Stats, Mutate and Write Modes

15. Content-based Filtering – Stream mode

```
CALL gds.knn.stream("Movies",
        { nodeProperties: {runtime:"EUCLIDEAN", imdbRating:"EUCLIDEAN", year:"Jaccard"},
        topK: 10, similarityCutoff: 0.85, sampleRate: 0.5, randomSeed:42, concurrency:1})
        YIELD node1, node2, similarity
RETURN left(gds.util.asNode(node1).title, 30) as movie, left(gds.util.asNode(node2).title,30) as
            similarTo,round(similarity,4)
```

"10 Things I Hate About You"	"Bowfinger"	0.8519
"10 Things I Hate About You"	"Zenon: Girl of the 21st Centur"	0.8519
"10 Years"	"Big Year, The"	0.9697

Problem 6 – Mutate and Write Modes

17. Content-based Filtering – Node similarity – Mutate mode

We now store the similarity in the in-memory graph

```
CALL gds.knn.mutate("Movies” { nodeProperties:{runtime:"EUCLIDEAN",
                                               imdbRating:"EUCLIDEAN",
                                               year:"Jaccard"},
        topK: 10, mutateRelationshipType: "IS_SIMILAR",
        mutateProperty: "similarity", similarityCutoff: 0.85,
        sampleRate:1, randomSeed:42, concurrency:1} )
```

Problem 6 – Mutate and Write Modes

18. Content-based Filtering – Node similarity – Write mode

We now WRITE this in-memory properties into the original graph, as follows:

```
CALL gds.graph.writeRelationship("Movies", "IS_SIMILAR", "similarity" )
```

Now, IS_SIMILAR is in the graph with property “similarity”