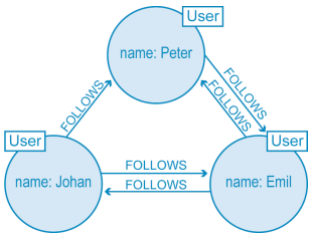
Lecture 5: Graph Databases

What is a Graph Database

- A graph is a pictorial representation of a set of objects where some pairs of objects are connected by links. It models data in the form of a graph
- It is composed of two elements – nodes and relationships
- The nodes of a graph depict the entities (person, place, thing, category or other piece of data) while the relationships depict the association of these entities
- Each relationship represents how two nodes are associated. For example, the two nodes CAKE and DESSERT would have the relationship IS A TYPE pointing from cake to dessert (cake is a type of dessert).
- Below is an example of a graph database used by Twitter:

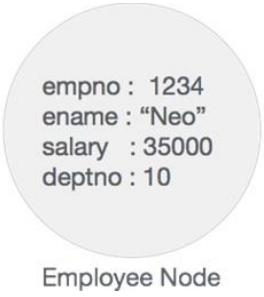


- Each node is labeled USER and belongs to a single person and is connected with relationships describing how each user is connected. Peter and Emil follow each other, Emil and Johan also follow each other. Johan follows Peter but Peter does not follow Johan.
- Graphs are used for social networks, fraud detection and asset management.
- Neo4j is a popular graph database. Other graph databases are Oracle NoSQL database, OrientDB, HypherGraphDB, GraphBase, InfiniteGraph and AlegroGraph
- Today most of the data exists in the form of the relationship between different objects and more often the relationship between the data in more valuable than the data itself
- Relational databases do not store the relationships between the data only the data.
- There are many advantages to using a graph database including:
  - Flexible data model
  - Realtime insights
  - High availability
  - Connected and semi structured data representation
  - Easy retrieval
  - Cypher query language (CQL)– a declarative query language to represent the graph visually. Commands are easy to learn
  - No joins
- To learn more about graph database, we will be using Neo4j

Neo4j Graph Database

- Access Neo4j
  - Access the console online (no installation): Go to http://console.neo4j.org/

    - Download a desktop version: Go to Access Neo4j https://neo4j.com/sandbox-v2/
        - If the system asks for an organization, type CUNY or CITY TECH
- Neo4j Information and Tutorials:
    - https://www.tutorialspoint.com/neo4j/index.htm
    - https://www.quackit.com/neo4j/tutorial/neo4j_create_a_node_using_cypher.cfm
    - https://neo4j.com/docs/developer-manual/current/introduction/
- Neo4j Graph Database has the following building blocks:
    - Nodes, Properties, relationships, labels, data browser
- A node is a fundamental unit of a Graph. It contains properties with key-value pairs as shown below:



- Property is a key-value pair to describe graph nodes and relationships
- Relationships are another major building bock of a Graph Database. Relationships connect two nodes. Below emp and dept are two different nodes. "Works-For" is a relationship between emp and dept nodes. The arrow shows that employee works for department (emp is the start node and dept is the end node). This is also known as an incoming relationship to the dept node and an outgoing relationship to the emp node.

- Like nodes, relationships contain properties as key-value pairs. Below "Works-For" relationship has one property as key-value pair (ID=123) which is the Id of this relationship.

- Label associates a common name to a set of nodes or relationships. A node or relationship can contain one or more labels. We can create new labels to existing nodes or relationships. We can remove the existing labels from the existing nodes or relationships. In the figure above, the labels are emp, dept and "Works-For"
- Neo4j Data Browser is used to execute CQL commands and view the output. It has a $ prompt. You type the commands after the dollar sign and click the execute button to run the commands.
- MATCH command is used to search the data with a specified pattern. Returns data.

Create Command

- CREATE command is used to create nodes, relationships and properties
- Create command can be used to create simple nodes without properties or relationships
  - Create (sample) return(sample); - creates a single node, semicolon optional
  - Create (sample1), (sample2) - creates two nodes
    Return sample1, sample2; - return can be without parenthesis
- Create command can be used to create properties
  - Create(Clooney:actor{name: "George Clooney", yob: 1961, pob: "England"}) return Clooney;
  - Create(Smith:actor{name: "Will Smith", yob:1968, pob: "US"}) return Smith;
  - Create(Damon:actor{name: "Matt Damon", yob:1970, pob: "US"}) return Damon;
  - Create(Almal:spouse{name: "Almal Clooney", yob:1978, pob: "England"}) Return Almal;
  - Create(Jada:spouse{name: "Jada Smith", yob:1971, pob:"US"}) return Jada;
  - Create(Luciana:spouse{name: "Luciana Barroso", yob:1976, pob:"Argentine"}) return Luciana;
- Start command allows you to see all the nodes and their relationships
  - Start n=node(*) return n;

Match Command

- To create relationships use the Match command. Relationships are presented within the square braces "[]" and depending on the direction of the relationship, it is placed between hyphen "-" and arrow "->"
  - Match (a:actor), (b:spouse) where a.name="George Clooney" and b.name="Almal Clooney" Create (a)-[r:married_to]->(b) return a,b;
  - Match (a:actor), (b:spouse) where a.name="Will Smith" and b.name="Jada Smith" Create (a)-[r:married_to]->(b) return a,b;
  - Match (a:actor), (b:spouse) where a.name="Matt Damon" and b.name="Luciana Barroso" Create (a)-[r:married_to]->(b) return a,b;
  - Start n=node(*) return n;
- Create complicated relationships
  - match (a:actor),(b:spouse) where a.name='George Clooney' and b.name="Almal Clooney" create (b)-[r:spouse_of]->(a) return a,b
  - match (a:actor),(b:spouse) where a.name='Matt Damon' and b.name="Luciana Barroso" create (b)-[r:spouse_of]->(a) return a,b
  - match (a:actor),(b:spouse) where a.name='Will Smith' and b.name="Jada Smith" create (b)-[r:spouse_of]->(a) return a,b

    - START n=node(*) RETURN n
- Create more complicated relationships
    - match (a:actor), (b:actor) where a.name="Matt Damon" and b.name="George Clooney" create (a)-[r:friend_of]->(b) return a,b;
    - START n=node(*) RETURN n;
    - match (a:spouse), (b:spouse) where a.name="Luciana Barroso" and b.name="Almal Clooney" create (a)-[r:friend_of]->(b) return a,b;
    - START n=node(*) RETURN n;
    - match (a:spouse), (b:actor) where a.name="Jada Smith" and b.name="Matt Damon" create (a)-[r:friend_of]->(b) return a,b;
    - START n=node(*) RETURN n;
- You can also use the Match command to show all the nodes and relationships
    - Match (n) return n;
- Change the label of a property, for example if you created “Jada Smith” as an actor instead of a spouse, you would use the following to change the property label from actor to spouse but first you would need to know the ID for “Jada Smith” by scrolling over the node for “Jade Smith”
    - Match (n:spouse) where ID(n)=6 remove n:spouse set n:spouse
- You can use Match to search for specific information, for within specific nodes. Below you want to know which spouse was born in Argentine
    - MATCH (spouse) WHERE spouse.pob = "Argentine" RETURN spouse
- You can use Match to search for more general information. Below you want ot know who was born in the US can be spouse or actor
    - MATCH (n) WHERE n.pob = "US" RETURN n
    - Match can also search for relationships, below you want to know who is married to whom
        - MATCH (a:spouse)-[r:married_to]->(b:actor) RETURN a,b;
    - Match can also search for relationships, below you want to know who is married to Matt Damon
        - MATCH (a:spouse)-[r:married_to]->(b:actor) WHERE b.name = "Matt Damon" RETURN a
- Match can be used to add properties
    - Create(Hemsworth:actor{name:"Chris Hemsworth", yob:1983, pob:"Australia"}) return Hemsworth;
    - Start n=node(*) return n
    - Now add a property to Hemsworth
        - MATCH (n) WHERE n.name = "Chris Hemsworth" SET n.owns = "Porsche"
        - Start n=node(*) return n
        - run the mouse over Chris Hemsworth
    - Use Match to change the information contained by a property
        - MATCH (n) WHERE n.name = "Chris Hemsworth" SET n.owns = "Audi"
        - Start n=node(*) return n

    - run the mouse over Chris Hemsworth

Count

- Match can be used with Count to give you a count of all the nodes
    - MATCH (n) RETURN count(n)
- Match can be used with Count to give you a count of all relationships as well
    - MATCH ()-->() RETURN count(*);

Order By

- Match can be used with ORDER BY to place output in ascending order (default)
    - MATCH (n) RETURN n.name, n.pob ORDER BY n.name
- Match can be used with ORDER BY to place output in descending order
    - MATCH (n) RETURN n.name, n.pob, n.yob ORDER BY n.yob desc

Limit

- Match can be used to limit the output
    - MATCH (n) RETURN n.name, n.pob, n.yob ORDER BY n.yob limit 3
- Match can be used to skip the output lines and display after the skipped lines
    - MATCH (n) RETURN n.name, n.pob, n.yob ORDER BY n.yob desc
    - MATCH (n) RETURN n.name, n.pob, n.yob ORDER BY n.yob desc skip 3

Delete

- Start n=node(*) return n
- Match can be used to delete a node by the label
    - MATCH (Almal:spouse {name: "Almal Clooney"}) DETACH DELETE Almal
    - Start n=node(*) return n
- Match can be used to delete a node by ID
    - Match (n) where ID(n)=6 DETACH DELETE n
    - Start n=node(*) return n
- Match can be used to delete a node and all its relationships
    - MATCH (n { name: 'Will Smith'}) DETACH DELETE n
    - Start n=node(*) return n
- Match can be used to delete just a relationships
    - MATCH (n { name: 'Matt Damon'}) – [r:married_to]->() DELETE r
    - Start n=node(*) return n
    - MATCH (n { name: 'George Clooney'}) – [r:friend_of]->() DELETE r
    - Start n=node(*) return n
- Match can be used to delete everything
    - MATCH (n) DETACH DELETE n
    - start n=node(*) return n;