



Neo4j

Data Engineering



Agenda

- Graph Fundamentals
- Business Cases
- Intro to Neo4j
- Discussion
- Activity

Why Not SQL?

- Relational algebra that is widely used in SQL does not always work for certain database types.
- Most common types of NoSQL databases:
 - NoSQL graph
 - NoSQL document
 - NoSQL key-value

NoSQL Graph

Nodes (vertices)

- Can have labels for classification purposes
- Attributes
 - Key-value pairs

Relationships (edges)

- Type
- Direction
- Attributes
 - Key-value pairs

Graph Feature Engineering

- Types of graph-related features
 - Graph features
 - Graph algorithm features
- Feature engineering is for:
 - AL (artificial intelligence)
 - ML (machine learning)
 - DL (deep learning)

Types of Graphs

Random

- Flat
- No patterns
- All nodes have the same probability of being attached to each other

Small - World

- High degree of local clustering
- Short average path lengths
- Hub and spoke
- No node more than a few relationships away from any other node

Scale - Free

- Hub and spoke in multiple scale
- Power-law distribution
- Change in one quantity results in relatively proportional change in another quantity.

Graph Characteristics

- Connected, disconnected
- Weighted, unweighted
- Directed, undirected
- Acyclic, cyclic
- Trees

Connected vs Disconnected

Issues with disconnected nodes

- May not be analyzed in most graph algorithms
An island of connected nodes disconnected from the main graph
- May not be analyzed in most graph algorithm
- Can cause algorithms to get stuck with no way out

Weighted vs Unweighted

Issues

- If an algorithm requires weights, unweighted relationships would not be considered
- If entire graph is unweighted, weighted algorithms are not appropriate

Directed vs Undirected

Directed: relationship has a direction

Undirected: relationship does not have a direction

Issues

- If an algorithm requires direction, undirected graphs would not be appropriate.
- If an algorithm does not consider direction, and we have directed graph, reconsider

Cyclic vs Acyclic (Trees)

- Cyclic: has cycles-path from a node back to itself
- Acyclic: no cycles- no path from any node back to itself
- Issues

If we have cycles, some algorithms can get stuck

Lots of common algorithms require acyclic.

Trees, Spanning Trees

- Tree: acyclic graph
 - Computer science: directed or undirected
 - Formal discrete mathematics: undirected only
- Spanning tree
 - All nodes in a graph
 - Relationships removed to remove cycles
- Minimum spanning tree
 - Connects all nodes with minimum hops
- Numerous spanning tree algorithms

Graph Density

- Maximum density = $(\text{nodes} [\text{nodes} - 1]) / 2$
- Actual density = $(2 * \text{relationships}) / (\text{nodes} * [\text{nodes} - 1])$
- Sparse = low density
- Issues
 - High level of density
 - Identify and peel off layers
 - High level of sparsity
 - See if we can add relationships

K-Partite Graphs

- Monopartite
 - One node type, one relationship type
- Bipartite
 - Two sets, nodes from one set only connect to nodes in the other set
- k-partite
 - Number of node types
 - In real world, most graphs have a high k value

K-Partite Graphs

- Monopartite
 - One node type, one relationship type
- Bipartite
 - Two sets, nodes from one set only connect to nodes in the other set
- k-partite
 - Number of node types
 - In real world, most graphs have a high k value

Random Network Cases

- Interactions involving social security numbers
- All demographics across the USA
- Births
- Deaths
- Paychecks
- Government assistance programs
- Retirement
- Disabled

Small-World Network Cases

- Social network
- Lots of small local networks
 - Most people connected to fewer than 500 other people
- Despite the small local networks
 - Most people are a few hops from anyone on the planet
 - Six degrees of separation

Scale-Free Network Cases

- World Wide Web
- Hub and spoke in multiple scales
- Each scale up of hub and spoke
 - Power law
 - Proportional scale up

All Roads in the USA

- Connected, disconnected
 - Highly connected over 48 states and DC
 - Disconnected nodes: Alaska, Hawaii, Puerto Rico, Guam, etc.
- Weighted, unweighted
 - Weight freeways and interstate highways highest
 - Weight gravel roads lowest
 - Weight traffic congestion periods, construction, special events

All Roads in the USA

- Directed, undirected
 - Two-way streets
 - One-way streets, freeways, entrance and exit ramps
- Cyclic, acyclic
 - Highly cyclic: many places, numerous ways to get from A to B
- Trees
 - Create spanning trees of major roads
 - Use the full graph to get you to and from the spanning tree

All Roads in the USA

- Directed, undirected
 - Two-way streets
 - One-way streets, freeways, entrance and exit ramps
- Cyclic, acyclic
 - Highly cyclic: many places, numerous ways to get from A to B
- Trees
 - Create spanning trees of major roads
 - Use the full graph to get you to and from the spanning tree

All Roads in the USA

- Density
 - High density in cities
 - Low density in rural areas with small towns
 - High sparsity in remote rural areas
- k-partite
 - High k value—numerous node and relationship types

All Roads in the USA

- Density
 - High density in cities
 - Low density in rural areas with small towns
 - High sparsity in remote rural areas
- k-partite
 - High k value—numerous node and relationship types

Path Algorithms

- Breadth-first search/depth-first search
- Eulerian circuits/Hamiltonian circuits
- Shortest path (with A* and Yen's variations)
- All pairs shortest path
- Single source shortest path
- Minimum spanning tree
- Random walk

Path Algorithms

- Searching trees
- Breadth-first search
 - Visit sibling nodes first before visiting child nodes
- Depth-first search
 - Visit child nodes first before visiting sibling nodes

All Pairs Shortest Path

Find the shortest path between all pairs of nodes

Minimum Spanning Tree

Find a tree structure path that will visit all nodes in the graph with smallest cost

Random Walk

Given a path size, randomly walk through nodes until we hit the path size

NoSQL Graph Path Algorithms

Map App

- Find nearby places of interest
- Solution: use breadth-first search

Game Simulation

- Simulate moves several levels deep to find the best move
- Solution: use a depth-first search

Driving Directions

- Find driving directions between A and B
- Solution: use a shortest path algorithm with weights for road, traffic lights, traffic, construction, etc.

NoSQL Graph Path Algorithms

Unexpected Traffic Jam

- We are a GPS routing company
- A traffic jam occurs
- We want to route our customers around the traffic
- Solution: *all pairs shortest path* to find all routes around the traffic jam

NoSQL Graph Path Algorithms

Home to Freeway

- We are a GPS routing company
- Our customers want to get from their home to the freeway as quickly as possible to get to work
- Numerous ways to get to the freeway
- Solution: *single source shortest path* with weights for road size, traffic lights, traffic, construction, etc. to find all routes from home to freeway

NoSQL Graph Path Algorithms

Laying Fiber

- We are a telecom company wanting to lay new fiber cabling to business and homes around the city
- Minimize cost for laying fiber: implies laying the smallest amount of fiber
- Maximize internet speed for our customers: implies each customer has the shortest path to data center
- Solution: minimum spanning tree

NoSQL Graph Path Algorithms

Explore Graph Structure

- We have a dataset we are using for AI
- We want to enhance our AI training by also creating a graph for some elements
- Since a graph is a secondary structure, not a primary structure, we need to statistically explore it
- Solution: random walks

Discussion

Consider the following problem. How would you design the algorithm to schedule deliveries:

We want to deliver a product from one location near the city center to customer locations up to 30 miles from the city center

We have several delivery vehicles

We will only deliver Monday through Friday

From 6am to 9am

Traffic within a 5 mile radius of the city center is clogged

Inbound traffic from 30 miles out is clogged

From 4pm to 7pm

Traffic within a 5 mile radius of the city center is clogged

Outbound traffic up to 30 miles out is clogged

For security reasons, we don't want to deliver before 6am or after 10pm. We will leave it on the porch on odd hours.

Assume the deliver truck has an onboard mounted laptop computer. What could we do after each delivery is made as the driver is starting to drive to the next delivery?

What is Neo4j?

Neo4j is a graph database management system. the data element Neo4j stored are nodes, edges connecting them, and attributes of nodes and edges

- Use Cypher as declarative graph query language for graph databases.
- Cypher provides a visual way of matching patterns and relationships.
- Syntax: (nodes) - [:RELATIONSHIP] -> (other nodes)

Cypher and SQL: key Differences

- Cypher is schema-flexible: users are not required to use a fixed schema to represent data;
- Cypher ends with the return clause whereas SQL begins with what a user wants to return.

```
SELECT movie.name  
FROM movie  
WHERE movie.rating > 7
```

```
MATCH (movie:Movie)  
WHERE movie.rating > 7  
RETURN movie.title
```

```
SELECT actors.name  
FROM actors  
      LEFT JOIN acted_in ON acted_in.actor_id = actors.id  
      LEFT JOIN movies ON movies.id = acted_in.movie_id  
WHERE movies.title = "The Matrix"
```

```
MATCH (actor:Actor)-[:ACTED_IN]->(movie:Movie {title: 'The Matrix'})  
RETURN actor.name
```

Neo4j Sandbox

A free, cloud-based instance of Neo4j that can be used to learn about Neo4j, test ideas, or play with pre-built data examples.

```
SELECT movie.name  
FROM movie  
WHERE movie.rating > 7
```

```
MATCH (movie:Movie)  
WHERE movie.rating > 7  
RETURN movie.title
```


```
SELECT actors.name  
FROM actors  
      LEFT JOIN acted_in ON acted_in.actor_id = actors.id  
      LEFT JOIN movies ON movies.id = acted_in.movie_id  
WHERE movies.title = "The Matrix"
```

```
MATCH (actor:Actor)-[:ACTED_IN]->(movie:Movie {title: 'The Matrix'})  
RETURN actor.name
```

Neo4j Sandbox sandbox.neo4j.com



New Project

Name	Status	
 Movies	Running Expires in about 3 days	<div data-bbox="1271 659 1406 703">Open</div> <div data-bbox="1439 670 1458 685">▼</div>


Select a project



☐ For Developers (17) ☐ For Data Scientists (8)

Featured Dataset






Beginner For Developers ☒

Movies

A guide to common graph query patterns involving connections between movies, actors, and directors.




For Developers ☐

OpenStreetMap

A graph solution to the shortest-path problem with Cypher involving points of interest and routing of Central Park in New York City.

Libraries Enabled: GraphQL




Beginner For Data Scientists ☐

Graph Data Science

Leverage Neo4j Graph Data Science library to explore graph algorithms for analytics and feature engineering.

Libraries Enabled: Graph Data Science



For Developers New ☐

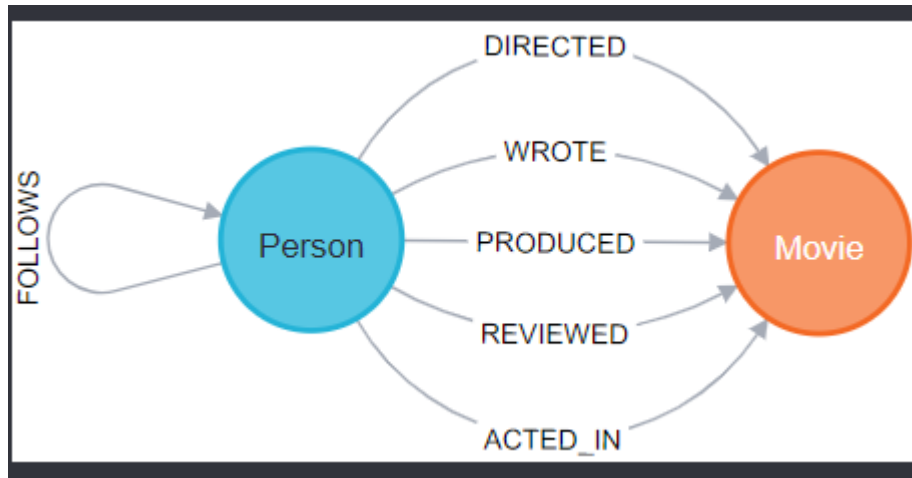
ICIJ Offshoreleaks

The Offshore leaks dataset and guide from the International Consortium of Investigative Journalists (ICIJ).

Nodes

A node in graph database is similar to a row in a relational database. The figure below shows 2 kinds of nodes – **person** and **movie**.

() presents a node. i.e. (p: Movie),
where p is a variable and movie is the type of node it is referring to

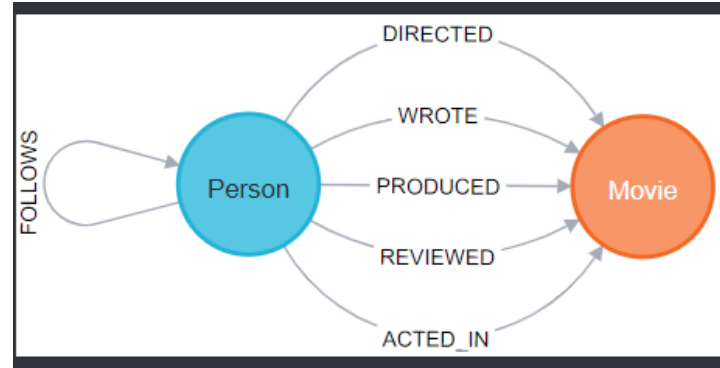


Relationship

Two nodes can be connected with relationship. In the below figure:
ACTED_IN, REVIEWED, PRODUCED, WROTE, DIRECTED are all relationships connecting the corresponding types of nodes.

Use [] represents a relationship. [w: DIRECTED] where w is a variable and DIRECTED is the relationship it is referring to.

Note that 2 nodes can be connected with more than one relationships.



Labels

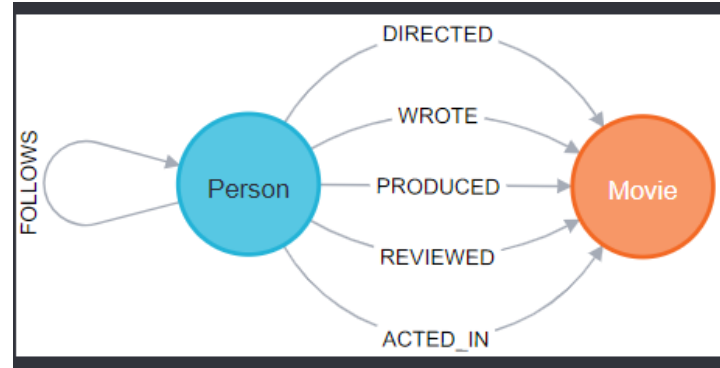
Labels is a name or identifier of a Node or a Relationship. In the image below Movie and Person are Node labels and ACTED,etc are Relationship types.

Labels are prefixed with a colon :

```
MATCH (p:Person)
```

```
RETURN p
```

```
LIMIT 20
```




Properties

Properties are name-value pairs that are used to add attributes to nodes and relationships

```
MATCH (m:Movie)
RETURN m.title, m.released, m.tagline
```

```
MATCH (p:Person)
RETURN p.name, p.born
```

<pre>1 MATCH (m:Movie) 2 RETURN m.title, m.released, m.tagline</pre>			
 Table	m.title	m.released	m.tagline
1	"The Matrix"	1999	"Welcome to the Real World"
2	"The Matrix Reloaded"	2003	"Free your mind"
3	"The Matrix Revolutions"	2003	"Everything that has a beginning has an end"
4	"The Devil's Advocate"	1997	"Evil has its winning ways"
5	"A Few Good Men"	1992	"In the heart of the nation's capital, in a courthouse of the U.S. government, one mi
6	"Top Gun"	1986	"I feel the need, the need for speed."

Create a Node

```
CREATE (p:Person {name: 'Jake Lee'})
```

```
RETURN p
```

Create a new person node with property name having value Jake Lee

Match and Where Claus

```
MATCH (p:Person)
```

```
WHERE p.name = 'Tom Hanks'
```

Merge Clause

The MERGE clause is used to either 1) match the existing nodes and bind them or 2) create new node or nodes and bind them

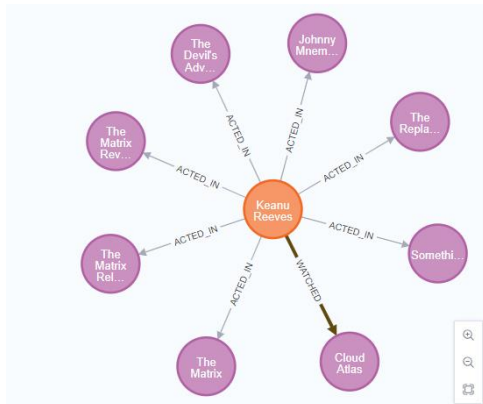
```
MERGE (p:Person {name: 'Jake Lee'})  
  
ON CREATE SET p.createdAt = timestamp()  
  
ON MATCH SET p.lastLoggedInAt = timestamp()  
  
RETURN p
```

It creates the Person node if it doesn't exist. If it exists, then it will set the property lastLoggedInAt to the current timestamp.

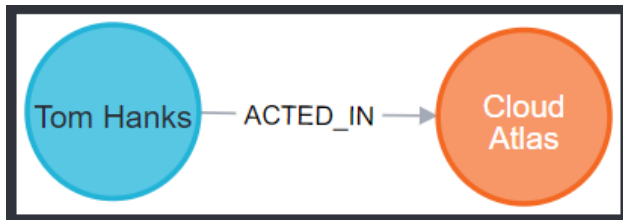
Create a Relationship

```
MATCH (p:Person), (m:Movie)
WHERE p.name = "Keanu Reeves" AND m.title = "Cloud Atlas"
CREATE (p) - [w:WATCHED] -> (m)
```

```
MATCH (p:Person{name: "Keanu Reeves"}) - [r] -> (m:Movie)
RETURN p, r, m
```



Relationship Types



In the above picture, the Tom Hanks node is said to have an outgoing relationship while the Cloud Atlas node is said to have an incoming relationship.

Relationships always have a direction. However, you only have to pay attention to the direction where it is useful.

To denote an outgoing or an incoming relationship in cypher, we use \rightarrow or \leftarrow .

```
MATCH (p:Person) -[r:ACTED_IN]-> (m:Movie)
RETURN p, r, m
```

More Cypher Queries

- Finding who directed Cloud Atlas movie

```
MATCH (m:Movie {title: 'Cloud Atlas'})<-[d:DIRECTED]-(p:Person)
RETURN p.name
```

- Finding all people who have co-acted with Tom Hanks in any movies:

```
MATCH (tom:Person {name: "Tom Hanks"})-[:ACTED_IN]->(:Movie)<-
[:ACTED_IN]-(p:Person)
RETURN p.name
```

- Finding all people related to the movie Cloud Atlas in any way

```
MATCH (p:Person)-[relatedTo]-(m:Movie {title: "Cloud Atlas"})
RETURN p.name, type(relatedTo)
```

More Cypher Queries

- Finding Movies and Actors that are 3 hops away from Kevin Bacon

```
MATCH (p:Person {name: 'Kevin Bacon'})-[*1..3]-  
      (hollywood)  
RETURN DISTINCT p, Hollywood
```

Interpretation:

- To find a node labeled as “person” with the name “ Kevin Bacon”.
- Traverses relationships to any node labeled as ‘Hollywood’ (*) up to 3 hops [*1..3]
- Returns distinct pairs of nodes representing Kevin Bacon and other nodes labeled as ‘Hollywood’. Each pair is unique.

What did you learn today ?