Graph Data Model and Neo4j Basics

Abdu Alawini

University of Illinois at Urbana-Champaign

CS411: Database Systems

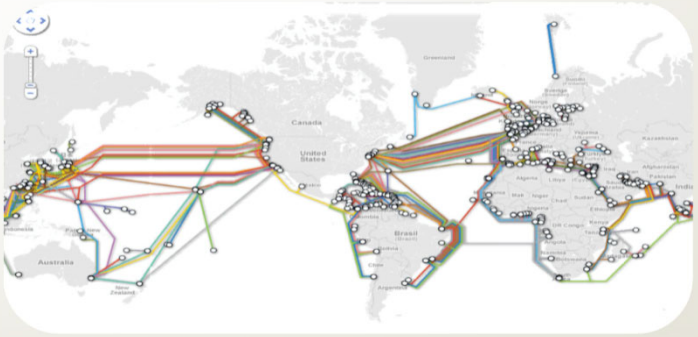
Learning Objectives

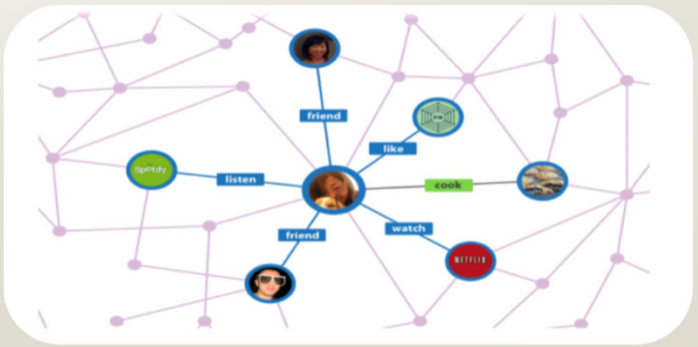
After this lecture, you should be able to:

- Discuss the Labeled Property Graph Data Model
- Write basic nodes and relationships Cypher queries.
- Use Cypher commands to manipulate Neo4J graph data.

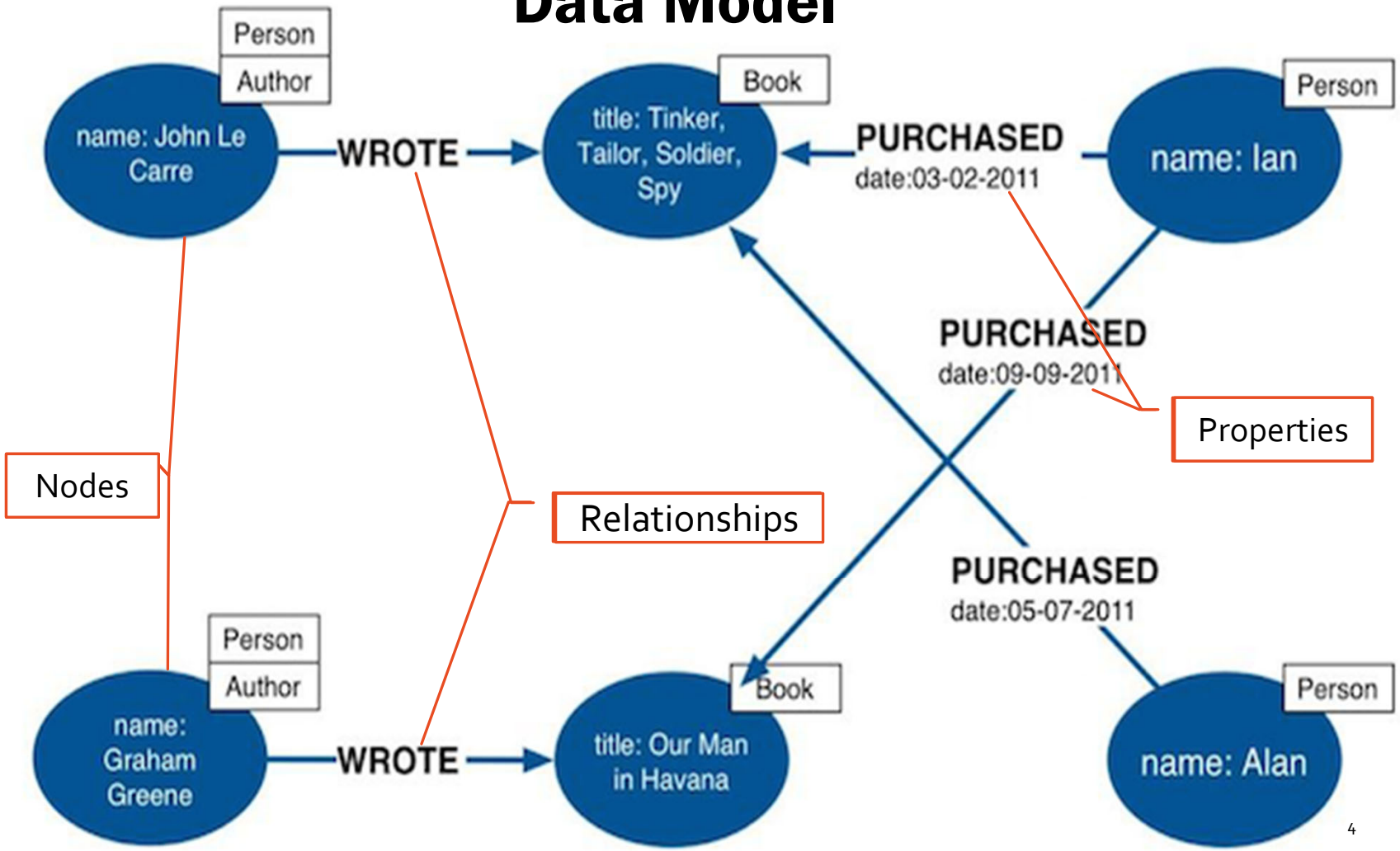
Neo4j: Graph Database

- Many types of data can be represented as graphs
  - Road networks, with intersections as nodes and road segments as edges
  - Computer networks, with computers as nodes and connections as edges
  - Social networks, with people/postings as nodes and edges as relationship (e.g. friends, likes, created, …)
- Graph databases store relationships and connections as first-class entities: “Property Graph Model”





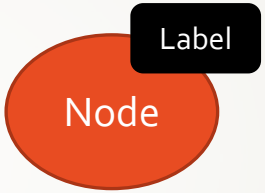
Labeled Property Graph Data Model

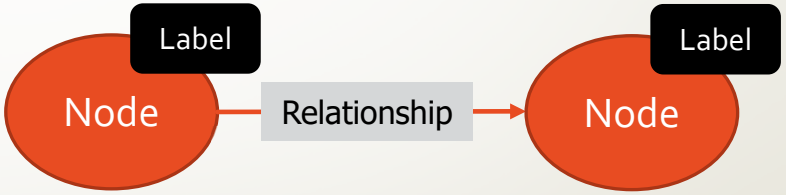


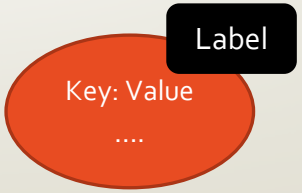
https://neo4j.com/developer/graph-database/

Neo4j Nodes and Relationships

- Nodes
  - have a system-assigned id
  - can have key/value properties
  - there is a reference node
    ("starting point" into the node space)
- Relationships (Edges)
  - have a system-assigned id
  - are directed
    (but can be traversed in either direction)
  - have a type – can have key/value properties
- Key/value properties
  - values always stored as strings
  - support for basic types and arrays of basic types







The Power of Graph Databases

- **Performance**
  - Graph databases have better performance when dealing with connected data vs. relational databases and NOSQL
- **Flexibility**
  - No need to define schema upfront
- **Agility**
  - graph data model is schema-free
  - testable graph database's API
  - query language

Graph DB and application evolve in an agile fashion

Cypher: A Graph Query Language

- SQL-like syntax
- Declarative pattern-matching graph query language
- Query a graph DB to find data (Nodes, Relationships, subgraphs) that matches a specific pattern
  - uses ASCII to specify a patterns

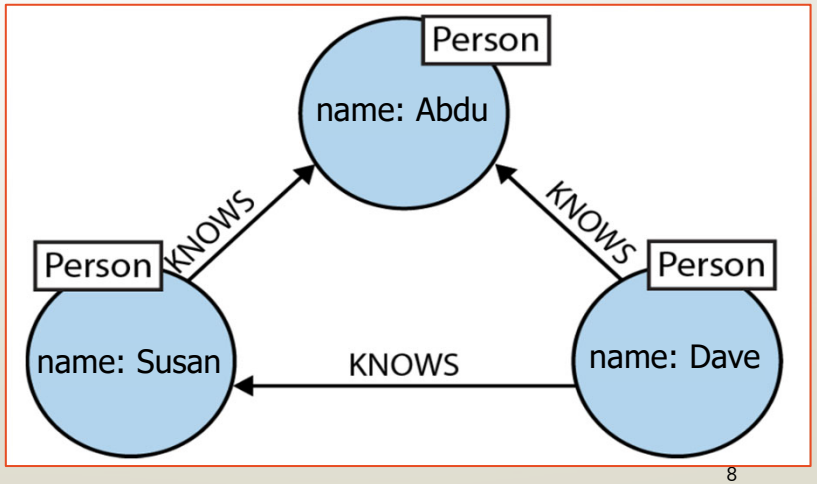
Cypher Basics

- **Patterns** describes *path* that connects a node (or set of nodes)
- **Identifiers** allow us to refer to the same node more than once when describing a pattern
- In Cypher, graphs are described using *specification by example*.

Variables (Identifiers)

Cypher Pattern

```
(x:Person {name: 'Abdu'})
<-[:KNOWS]-(y:Person {name: 'Dave'})
-[:KNOWS]->(z:Person {name: 'Susan'})
-[:KNOWS]->(x)
```



Cypher Queries

- Cypher is comprised of three main concepts
  - **`MATCH`**: graph pattern to match, bound to the starting points
  - **`WHERE`**: filtering criteria
  - **`RETURN`**: what to return
- Implemented using the Scala programming language

Querying Nodes

- Nodes are surrounded with parentheses

```
()
(matrix)
(:Movie)
(matrix:Movie)
(matrix:Movie {title: "The Matrix"})
(matrix:Movie {title: "The Matrix", released: 1997})
```

- Nodes queries

```
MATCH (node:Label)
WHERE node.property = {value}
RETURN node.property
```


Source: https://neo4j.com/developer/cypher-query-language/

Examples

- Find ”Apollo 13” movie

```
MATCH (n:Movie {title:"Apollo 13"})
RETURN n.released;
```

n.released
1995

- Find 1990’s movies

```
MATCH (n:Movie)
WHERE n.released >= 1990 AND n.released <=2000
RETURN n;
```

"n"
{"tagline":"Welcome to the Real World","title":"The Matrix","released":"1999"}
{"tagline":"At odds in life... in love on-line.","title":"When Harry Met Sally","released":"1998"}
{"tagline":"In every life there comes a time when that thing you dream becomes that thing you do","title":"That Thing You Do","released":"1996"}
{"tagline":"Pain heals, Chicks dig scars... Glory lasts forever","title":"The Replacements","released":"2000"}

Querying Relationships

Relationships are basically an arrow --> between two nodes.

- relationship-types -[:KNOWS]-> , <-[: LIKE]-
- a variable name -[rel:KNOWS]-> before the colon
- additional properties -[rel:KNOWS {since:2018}]->
- structural information for paths of variable length -[:KNOWS*..4]->

```
MATCH (n1:Label1)-[rel:TYPE]->(n2:Label2)
WHERE rel.property = {value}
RETURN rel.property, type(rel)
```


Source: https://neo4j.com/developer/cypher-query-language/

Example

```
MATCH (m:Movie)<-[r:ACTED_IN]-(p:Person)
WHERE r.roles = "Morpheus"
RETURN p.name, m.title;
```

"p.name"	"m.title"
"Laurence Fishburne"	"The Matrix"
"Laurence Fishburne"	"The Matrix Reloaded"
"Laurence Fishburne"	"The Matrix Revolutions"

Patterns

- Nodes and relationship expressions are the building blocks for more complex patterns.
- Patterns can be written continuously or separated with commas.
- You can refer to variables declared earlier or introduce new ones.

Types of Pattern Matching

- friend-of-a-friend

```
(user)-[:KNOWS]-(friend)-[:KNOWS]-(foaf)
```

- shortest path:

```
path = shortestPath( (user)-[:KNOWS*..5]-(other) )
```

- collaborative filtering

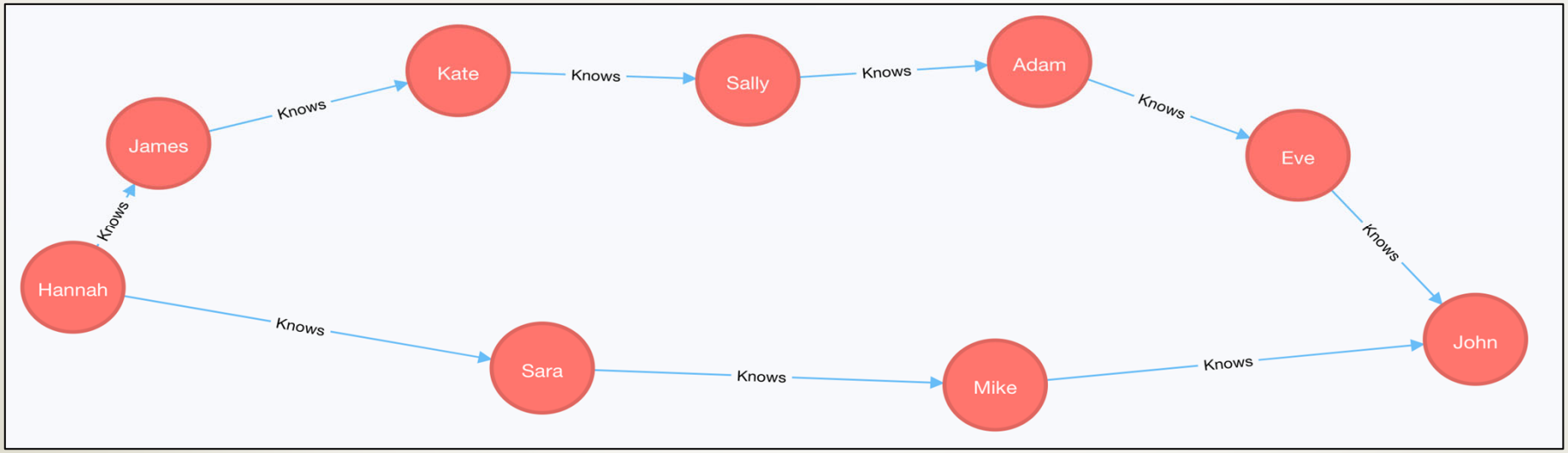
```
(user)-[:PURCHASED]->(product)<-[:PURCHASED]-()-[:PURCHASED] ->(otherProduct)
```

- tree navigation

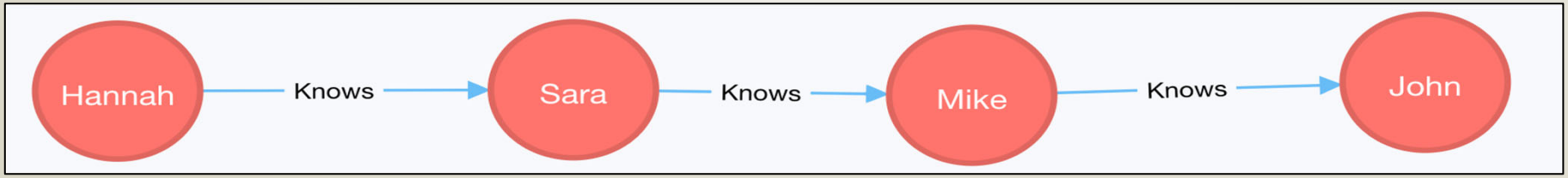
```
(root)<-[:PARENT*]-(leaf:Category)-[:ITEM]->(data:Product)
```

Source: https://neo4j.com/developer/cypher-query-language/

Shortest path example




```
MATCH x = shortestPath( (s:Student {name:"Hannah"})-[:Knows*..10]->(s1:Student {name:"John"}))
RETURN x;
```



Collaborative Filtering Example

```
MATCH (p:Person)-[:ACTED_IN]->(m:Movie)<-[:DIRECTED]-(p)
RETURN p.name, m.title;
```

p.name	m.title
Tom Hanks	That Thing You Do
Clint Eastwood	Unforgiven
Danny DeVito	Hoffa

Outline

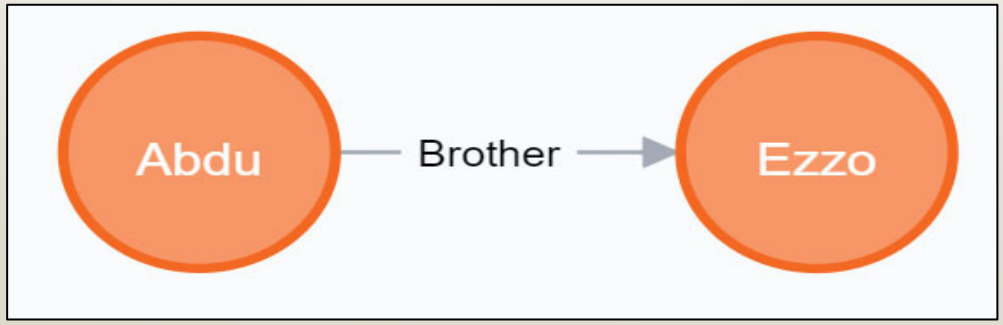
- ✓ Map-reduce
- ✓ Introduction to Neo4J
- Cypher: A Graph Query Language
  - ✓ Querying Nodes and Relationships using Patterns
  - Manipulating Graph Data

Manipulating Graph Data

Create statement creates nodes and relationships specified in the pattern

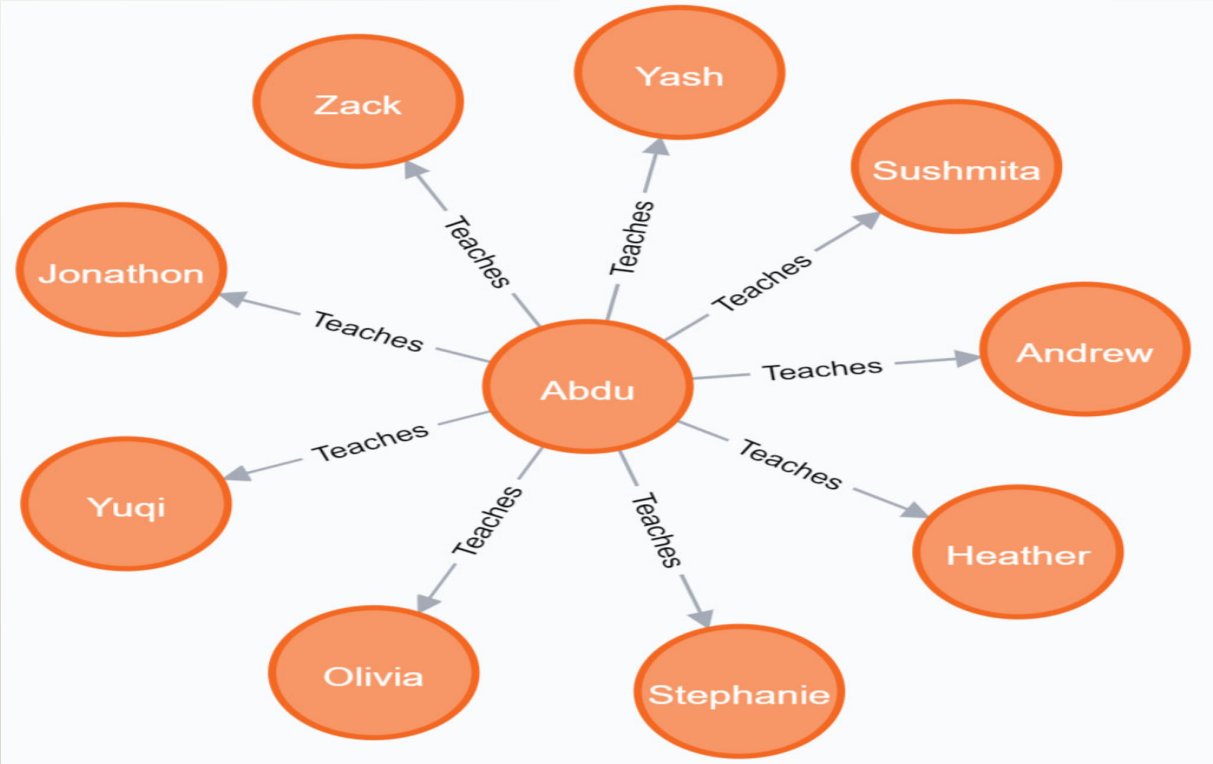
```
1 CREATE (var:Person {name:"Abdu"})
2 RETURN var;
```


```
1 CREATE path = (:Person {name:"Abdu", age:21})-[:Brother]->(:Person {name:"Ezzo", age:55})
2 RETURN path;
```



FOREACH Statement

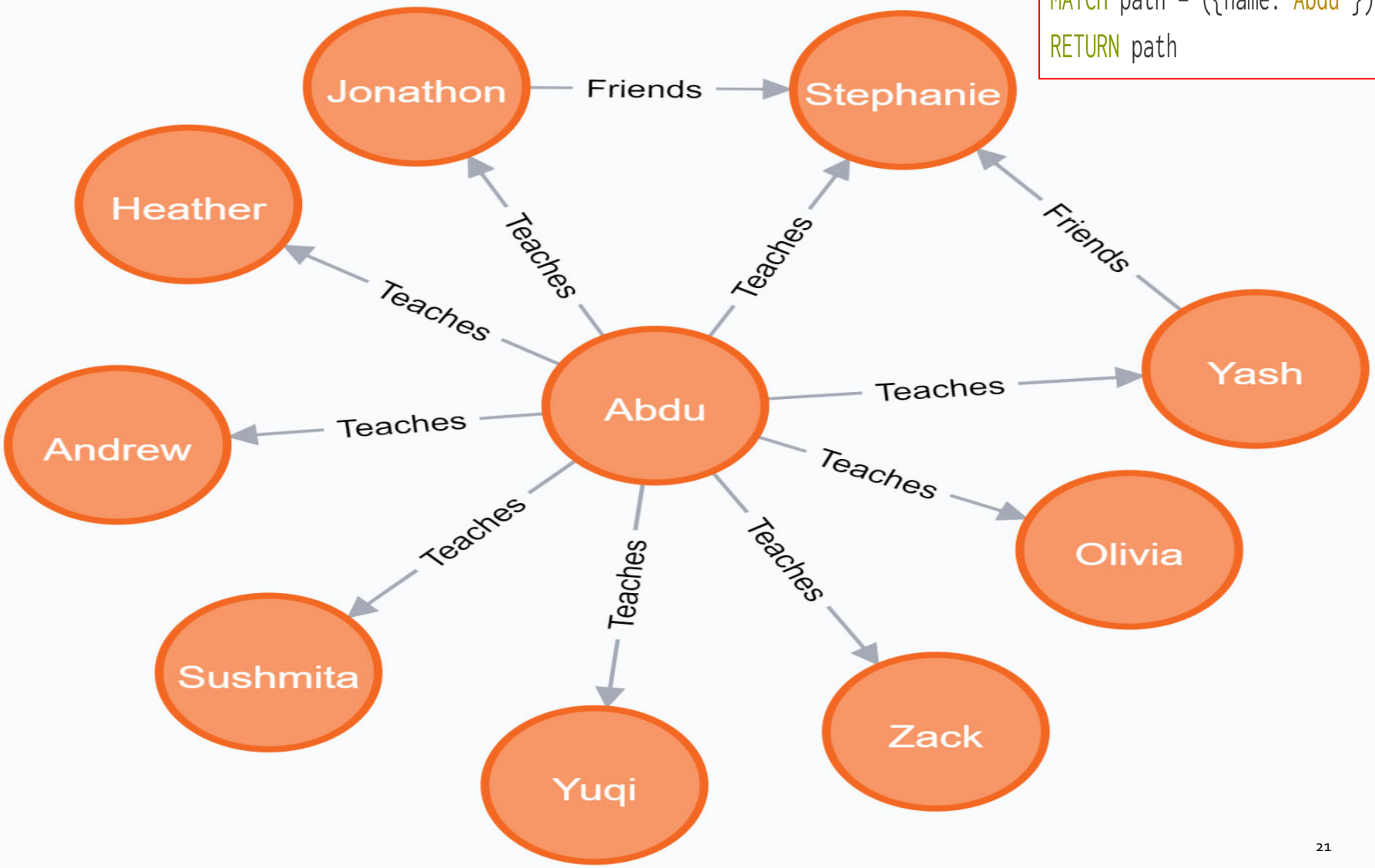
```
1 CREATE(me:Person {name: 'Abdu'})
2 FOREACH (student in ['Jonathon','Zack','Yash', 'Stephanie', 'Olivia',
  'Sushmita','Heather','Yuqi','Andrew'] | CREATE (me)-[:Teaches]->(:Person {name:student}))
```



Creating Relationships

```
MATCH (a:Person {name:"Lamya"})
MATCH (b:Person {name:"Devin"})
MATCH (c:Person {name: "Vidisha"})
CREATE (a)-[:Friends]->(b)<-[:Friends]-(c)
RETURN a,b,c
```

```
MATCH path = ({name:"Abdu"})--()
RETURN path
```



DELETE, REMOVE and SET Commands

To delete a node:

```
MATCH (p:Person {name:"Abdu"})
DELETE p;
```

To delete a node and all its relationships:

```
MATCH (a {name:"Abdu"})
DETACH DELETE a;
```

To remove a property:

```
MATCH (abdu { name:'Abdu' })
REMOVE abdu.age
RETURN abdu
```

To update or add a property:

```
MATCH (n { name: 'Abdu' })
SET n.name = 'Abdussalam'
RETURN n
```

Update a property using FOREACH

```
MATCH p =(begin)-[*]->(END )
WHERE begin.name = 'A' AND END .name = 'D'
FOREACH (n IN nodes(p)| SET n.marked = TRUE )
```

Completing patterns: Merge

- acts like a combination of MATCH or CREATE
- checks for the existence of data first before creating it

```
MATCH (m:Person {name:"Abdu"})
MERGE (m)-[:Works_with]->(s:Person {name:"Susan"})
RETURN m,s
```

ON CREATE, ON MATCH

```
MERGE (a:Person { name: 'Abdu' })
ON CREATE SET a.created = timestamp()
ON MATCH SET a.lastSeen = timestamp()
RETURN a.name, a.created, a.lastSeen
```

Running the above command for the first time

"a.name"	"a.created"	"a.lastSeen"
"Abdu"	"1510526671422"	null

Running the same command for the second time

"a.name"	"a.created"	"a.lastSeen"
"Abdu"	"1510526671422"	"1510526738428"

Source: https://neo4j.com/developer/cypher-query-language/

Neo4j Resources

1. Neo4j Tutorial: https://www.tutorialspoint.com/neo4j/index.htm
2. Video Tutorials: https://neo4j.com/blog/neo4j-video-tutorials/?_ga=2.57983406.580712586.1555337212-902296776.1553382068
3. GraphGists are teaching tools which allow you to explore how data in a particular domain would be modeled as a graph and see some example queries of that graph data
   - https://neo4j.com/graphgists/
4. Awesome user-defined procedures: https://github.com/neo4j-contrib/neo4j-apoc-procedures