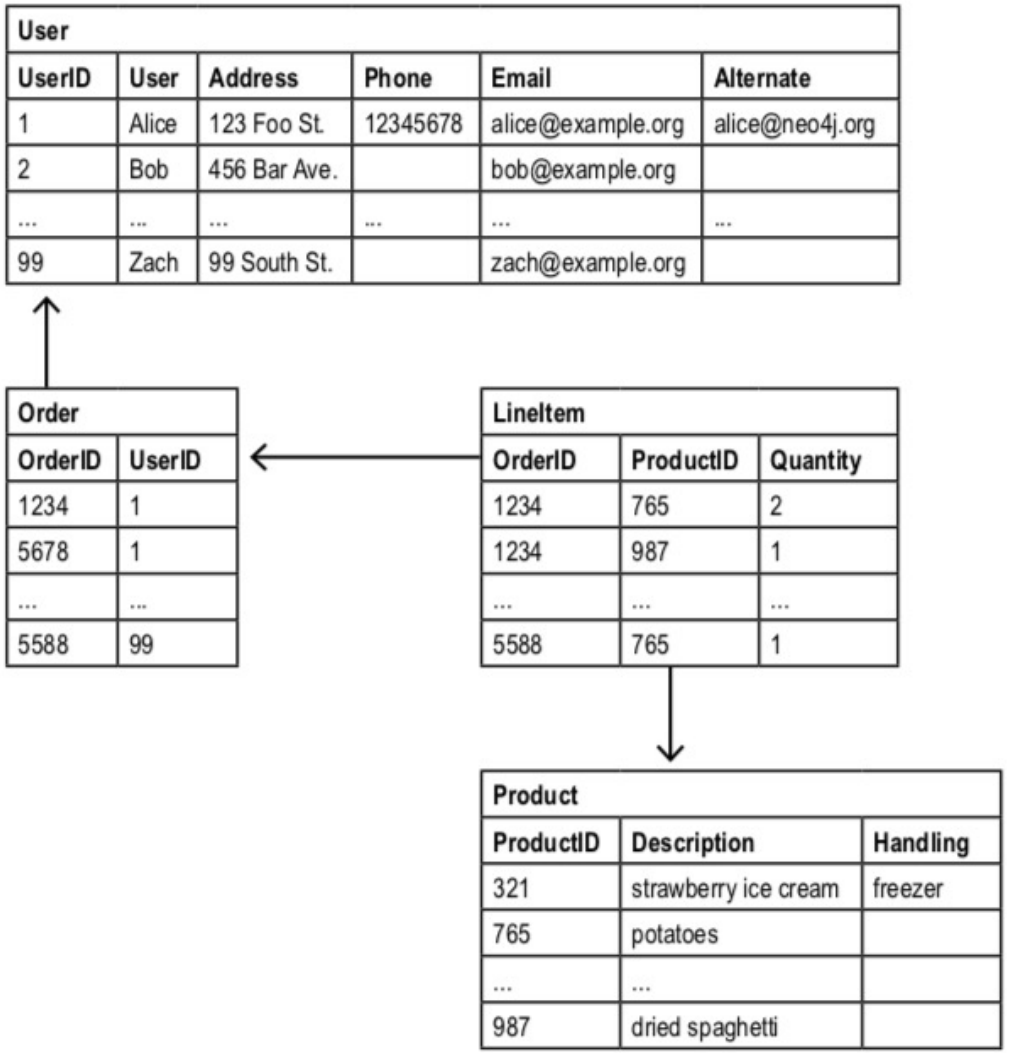
Graph Database

COMP9312_24T2

UNSW
SYDNEY

Drawbacks of Relational Databases

- **Schema are inflexible**
  - Missing values
  - Business Requirements change quickly
- **Inefficient**
- Consider the E-Commerce example
  - What items did a customer buy?
  - Which customers bought this product?
  - A basic query for recommendation: Which customers buying **this** product also bought **that** product?
- NoSQL database faces the similar issue



User

UserID	User	Address	Phone	Email	Alternate
1	Alice	123 Foo St.	12345678	alice@example.org	alice@neo4j.org
2	Bob	456 Bar Ave.		bob@example.org	
...	...	...	...	...	...
99	Zach	99 South St.		zach@example.org	

Order

OrderID	UserID
1234	1
5678	1
...	...
5588	99

LineItem

OrderID	ProductID	Quantity
1234	765	2
1234	987	1
...	...	...
5588	765	1

Product

ProductID	Description	Handling
321	strawberry ice cream	freezer
765	potatoes	
...	...	
987	dried spaghetti	

Drawbacks of Relational Databases (Cont.)

id	node1	node2
0	0	1
1	2	4
2	1	3
3	1	2
4	5	2
5	3	4

...

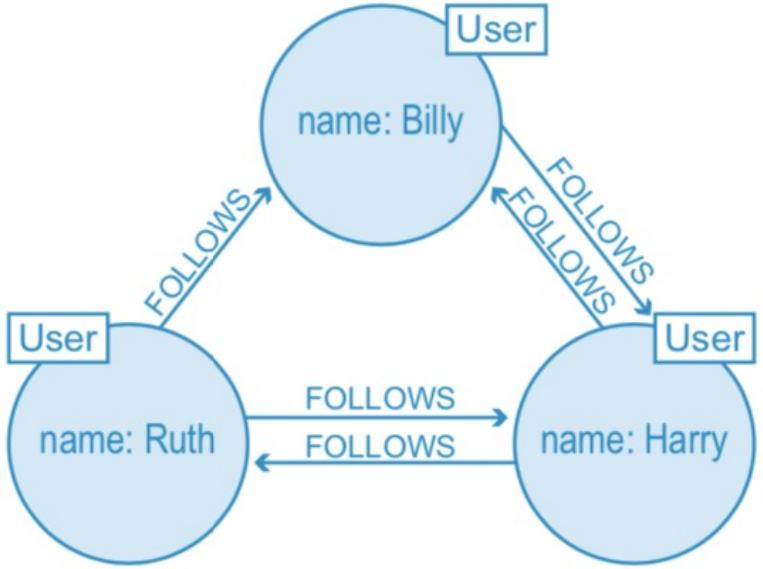
Time complexity to find neighbors of a node?

Time complexity to find all common neighbors of two nodes?

How about BFS? Triangles?
at least needs many interaction operations in RDBMS~

What is a Graph Database?

- A database consists of **entities** and their **relationships**
- An entity is modelled as a **node** (with arbitrary number of attributes).
- A relationship is modelled as an **edge** (possibly with labels or weights)
- No background of graph theory is needed to query a graph database
- More intuitive to understand than a relational database management systems (RDBMS)

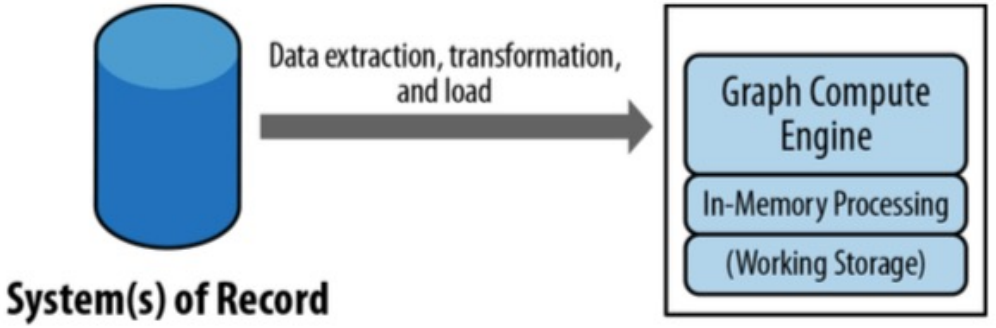


Why we care about Graph Database~

- **Performance**
  - ❑ Traditional Joins are inefficient
  - ❑ Billion-scale data are common, e.g., Facebook social network , Google web graph
- **Flexibility**
  - ❑ Real-world entities may not have a fixed schema. It is not feasible to design 1000 attributes for a table.
  - ❑ Relationships among entities can be arbitrary. It is not feasible to use 1000 tables to model 1000 types of relationships.
- **Agility**
  - ❑ Business requirements changes over time
  - ❑ Today’s development practices are agile, test-driven

How a graph database works

- **Graph Storage**
  - ❑ Usually use the native graph structure, e.g., adjacency lists.
  - ❑ Efficient and easy to develop graph algorithms.
- **Graph Processing Engine**
  - ❑ Algorithms and queries supported based on the graph storage
  - ❑ Native graph processing is more efficient



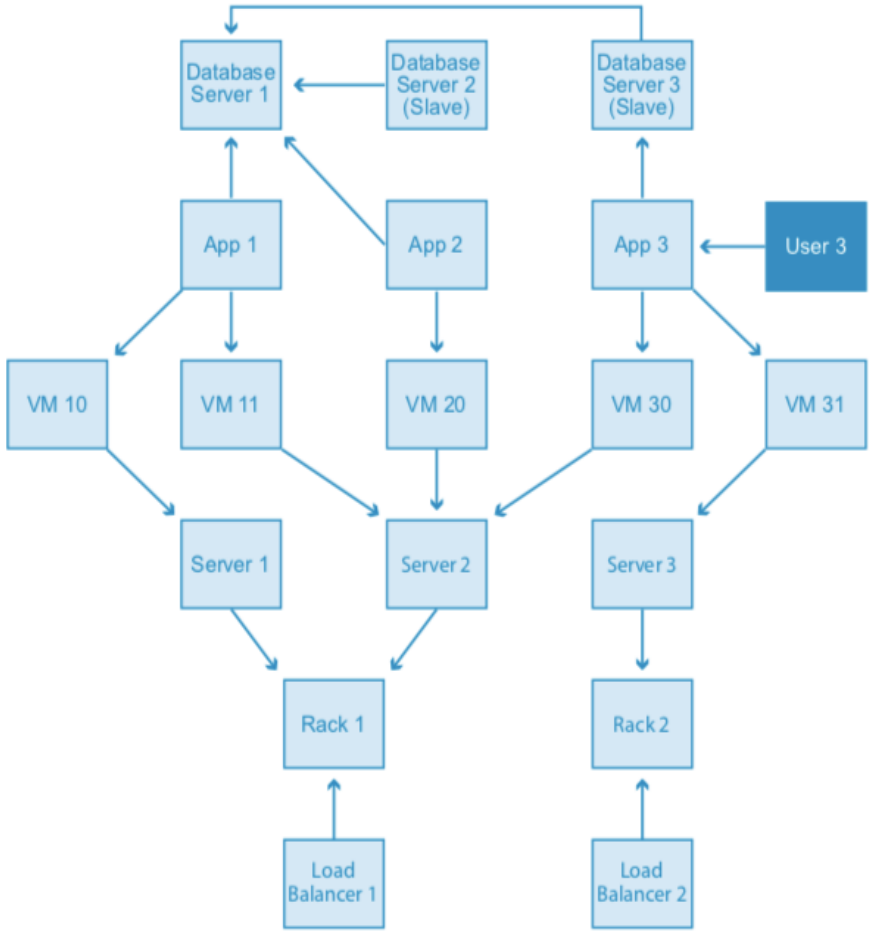
Graph DB VS RDBMS

- **An Example:**
  - ❑ Data: a social network of 1,000,000 people each with approximately 50 friends
  - ❑ Query: to find friends-of-friends connections to a depth of five degrees.
- **Efficiency Comparison:**

Depth	RDBMS execution time(s)	Neo4j execution time(s)	Records returned
2	0.016	0.01	~2,500
3	30.267	0.168	~110,000
4	1543.505	1.359	~600,000
5	Unfinished	2.132	~800,000

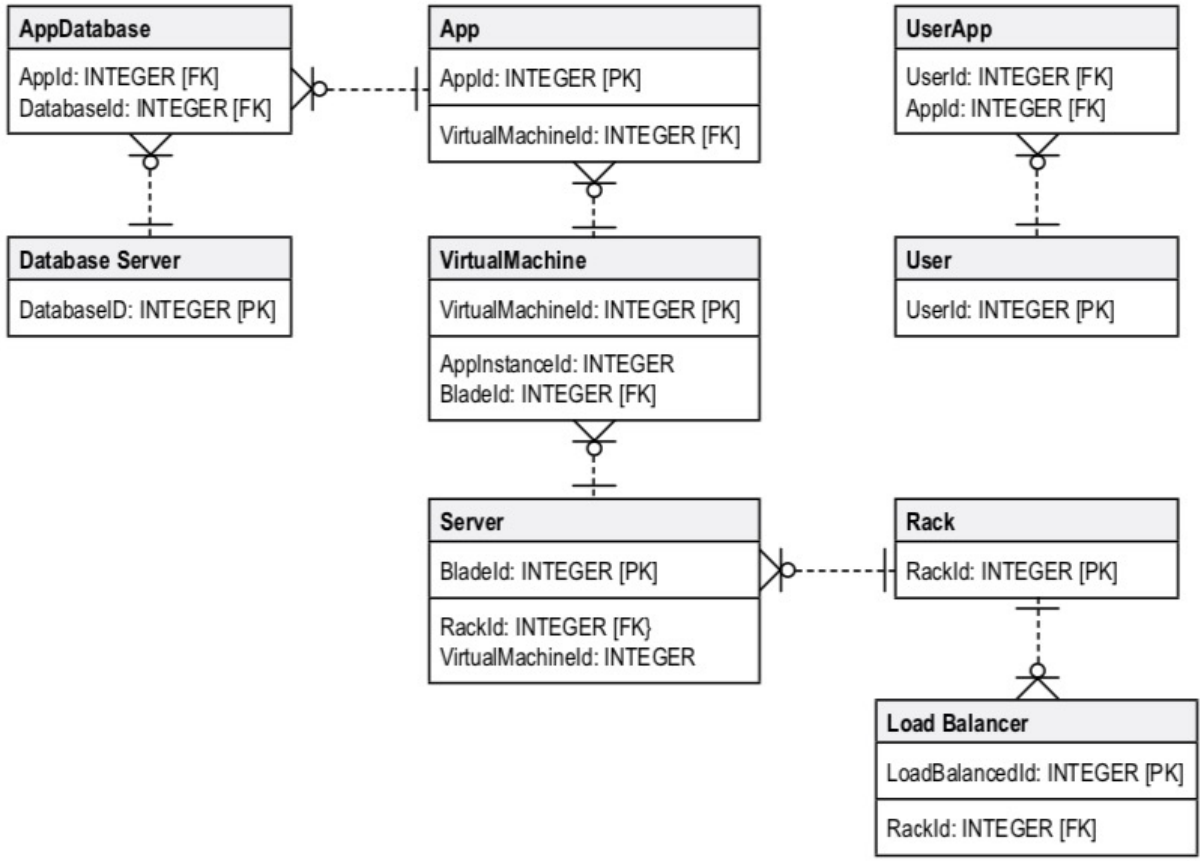
Data Modelling: RDBMS vs Graph DB

- An Example: In this data center management domain, several data centers support a few applications using infrastructure like virtual machines and load balancers.
- The “whiteboard” form is shown on the right



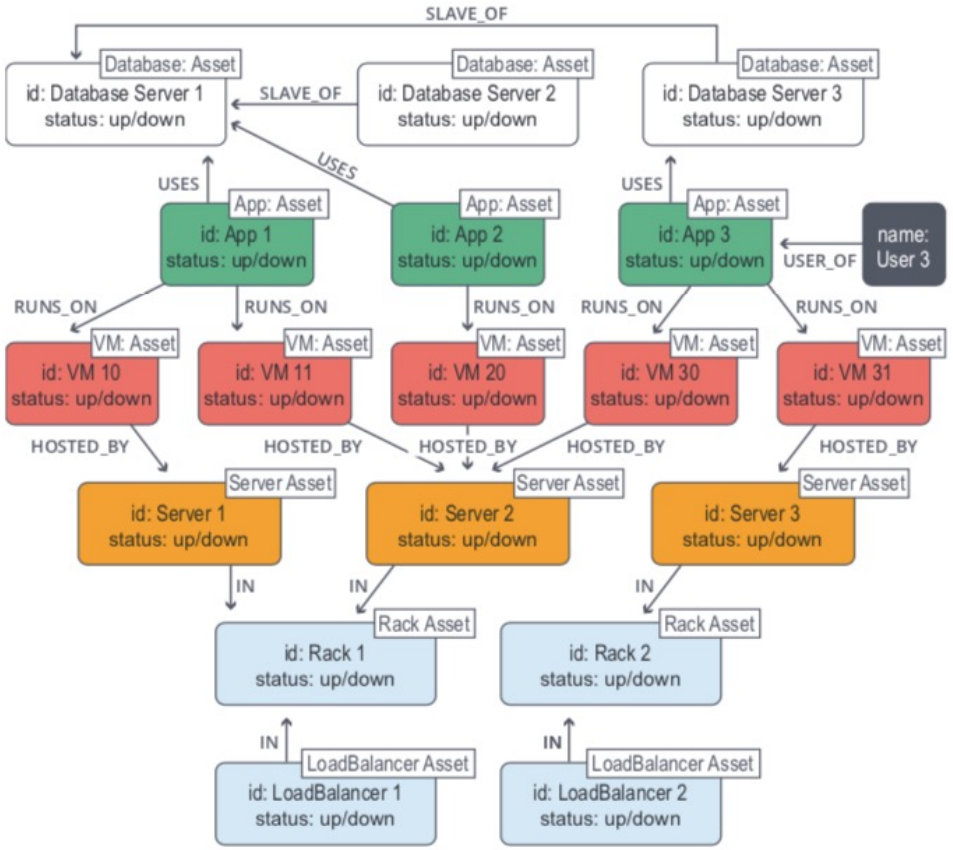
Data Modelling in RDBMS

- **Data Model in RDBMS**
  - ❑ Aim: From initial whiteboard to relations
  - ❑ Step 1: design schema for each table (consider data redundancy, efficiency, …)
  - ❑ Step 2: design primary key (PK) and foreign key (FK)
  - ❑ Step 3: insert data for each table following the schema
  - ❑ Step 4: query the RDBMS using SQL
  - ❑ Needs careful modelling



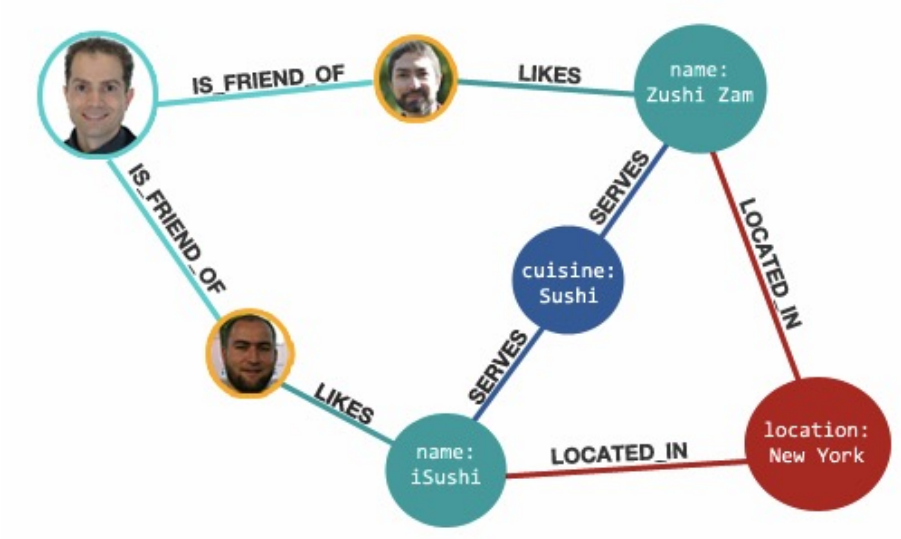
Data Modelling in Graph DB

- **Data Model in Graph DB**
  - ❑ Aim: From initial whiteboard to Graph DB
  - ❑ Step 1: insert data for entities and relationships
  - ❑ Step 2: query the Graph DB
  - ❑ Looks just as what they are on the whiteboard
  - ❑ No schema but highly expressive.
  - ❑ New types of data can be easily integrated
  - ❑ We need a query language



Cypher: the graph query language in Neo4j

Optional



An example of Cypher:
Find Sushi restaurants in New York that the friends of Philip like

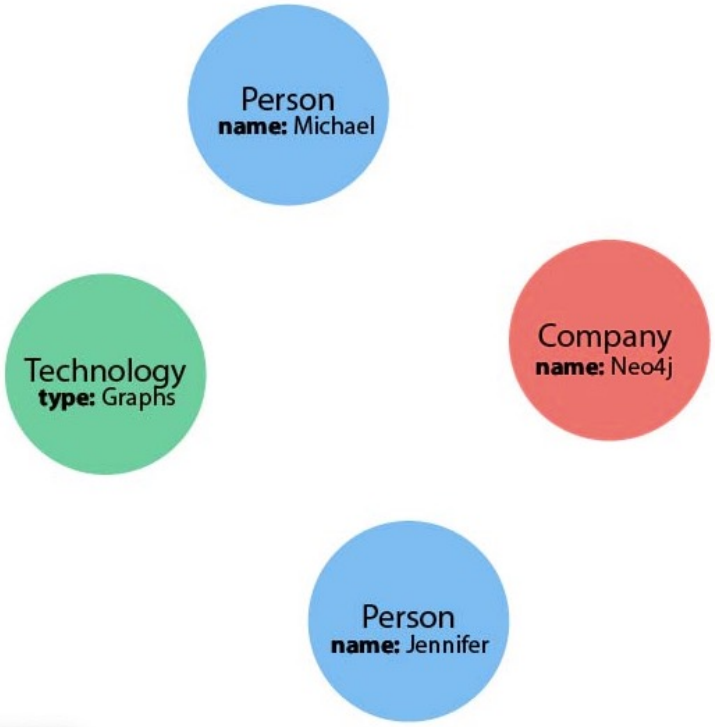
```
MATCH (person:Person)-[:IS_FRIEND_OF]->(friend),
      (friend)-[:LIKES]->(restaurant:Restaurant),
      (restaurant)-[:LOCATED_IN]->(loc:Location),
      (restaurant)-[:SERVES]->(type:Cuisine)

WHERE person.name = 'Philip'
  AND loc.location = 'New York'
  AND type.cuisine = 'Sushi'

RETURN restaurant.name, count(*) AS occurrence
ORDER BY occurrence DESC
LIMIT 5
```

Representing Nodes in Cypher

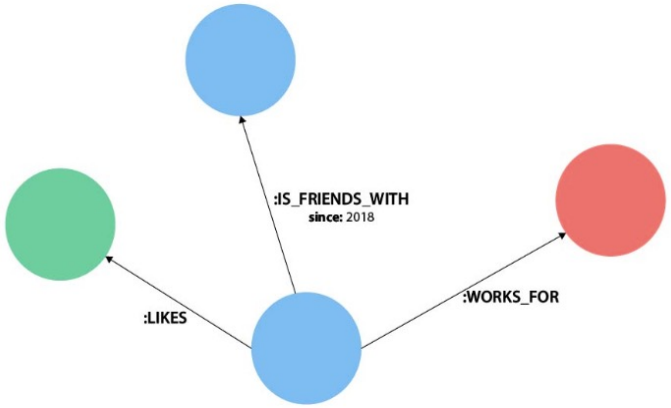
Optional



- () //anonymous node (no label or variable) can refer to any node in the database
- (p:Person) //using variable p and label Person
- (:Technology) //no variable, label Technology
- (work:Company) //using variable work and label Company

Representing Relationships

Optional



- -- or --> or <-- //anonymous relationship
- -[rel]-> //using variable rel to denote a relationship of any label
- -[rel:LIKES]-> //using variable rel to denote a relationship of label LIKES
- -[:LIKES]-> //denote a relationship of label LIKES

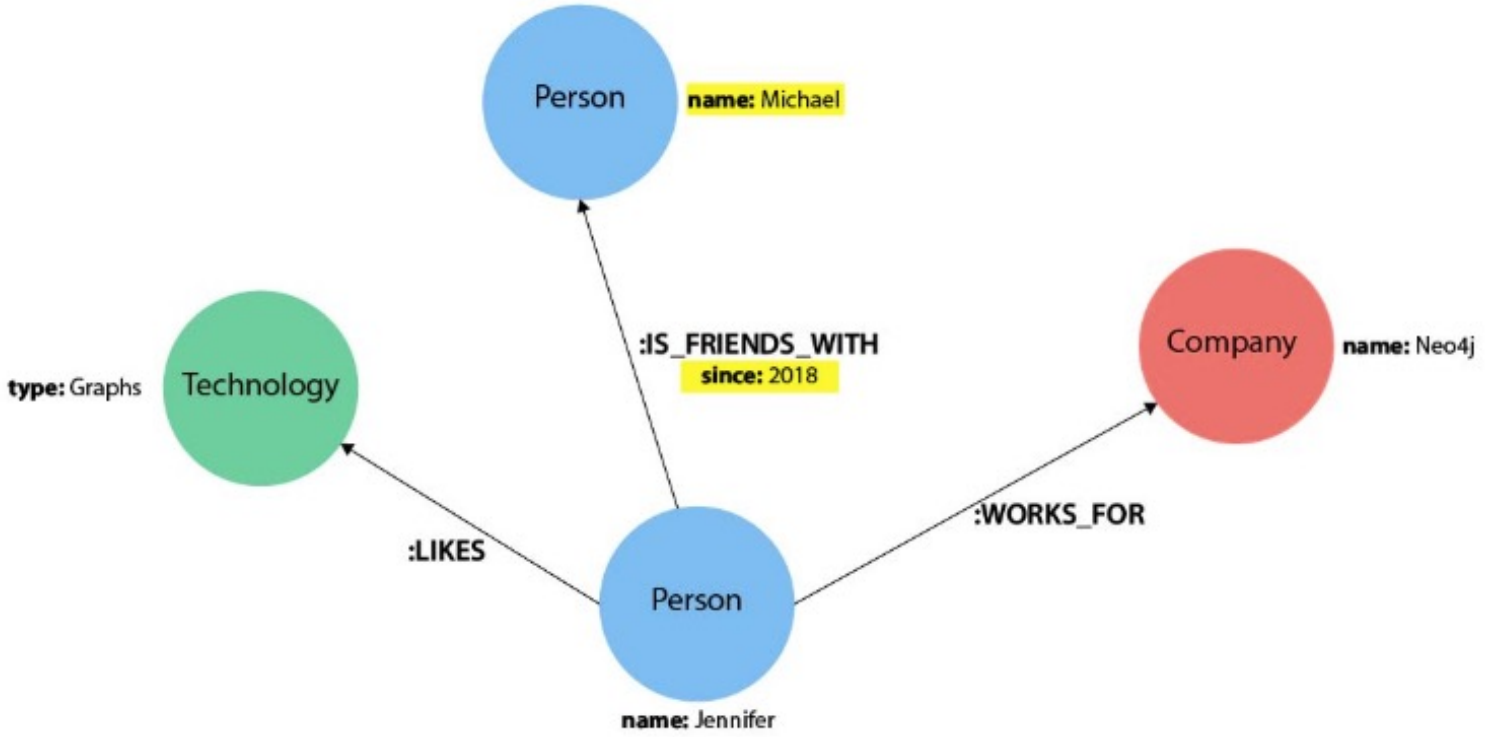
```
//data stored with this direction
CREATE (p:Person)-[:LIKES]->(t:Technology)

//query relationship backwards will not return results
MATCH (p:Person)<-[:LIKES]-(t:Technology)

//better to query with undirected relationship unless sure of direction
MATCH (p:Person)-[:LIKES]-(t:Technology)
```

Node or Relationship Properties

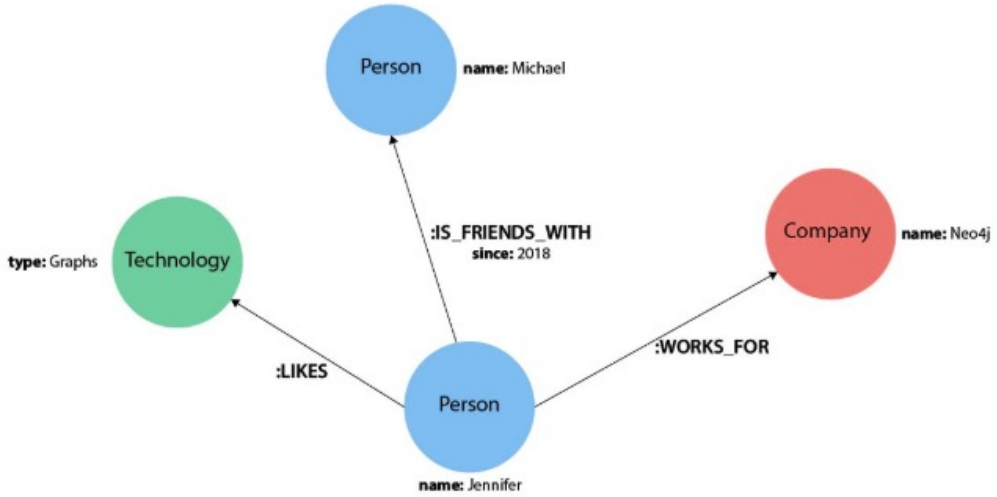
Optional

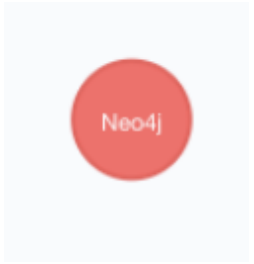


- Node property: `(p:Person {name: 'Jennifer'})`
- Relationship property: `-[rel:IS_FRIENDS_WITH {since: 2018}]->`

Find nodes by relationships

Optional

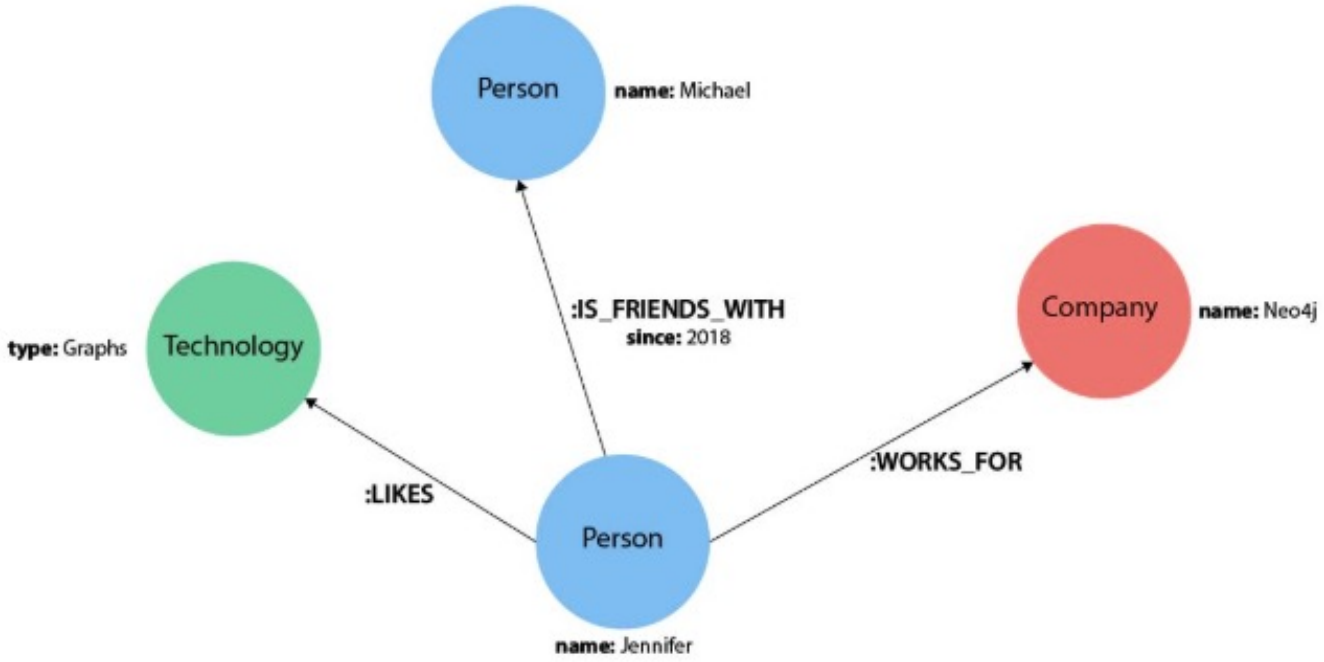





```
MATCH (:Person {name: 'Jennifer'})-[:WORKS_FOR]->(company:Company)
RETURN company
```

Create a node

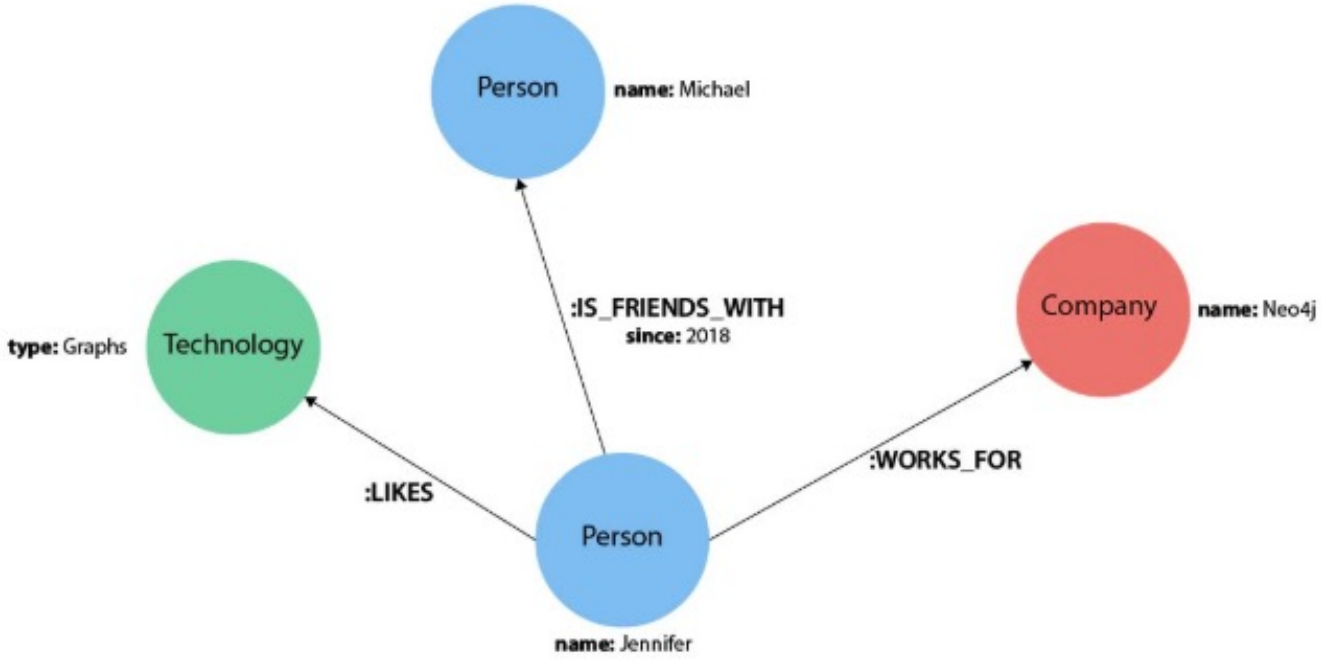
Optional




```
CREATE (friend:Person {name: 'Mark'})
RETURN friend
```

Create a relationship

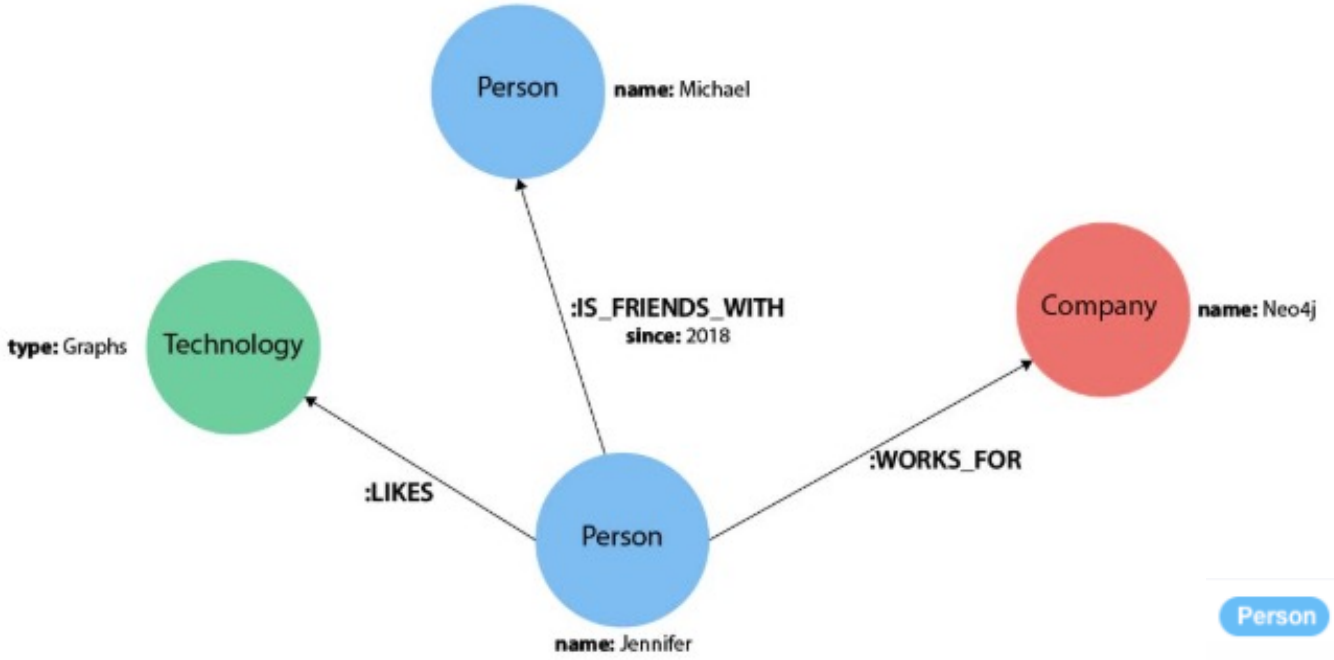
Optional




```
MATCH (jennifer:Person {name: 'Jennifer'})
MATCH (mark:Person {name: 'Mark'})
CREATE (jennifer)-[rel:IS_FRIENDS_WITH]->(mark)
```

Create/modify a node property

Optional

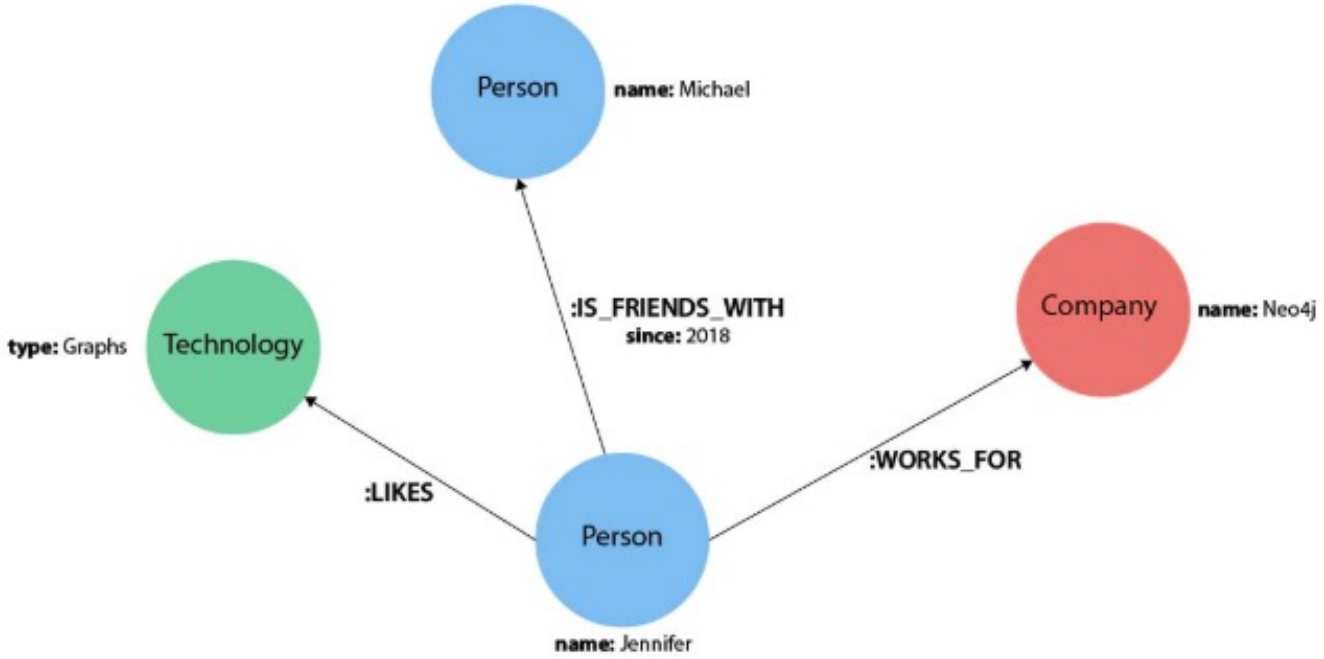


Person <id>: 0 birthdate: "1980-01-01" name: Jennifer

```
MATCH (p:Person {name: 'Jennifer'})
SET p.birthdate = date('1980-01-01')
RETURN p
```

Create/modify a relationship

Optional



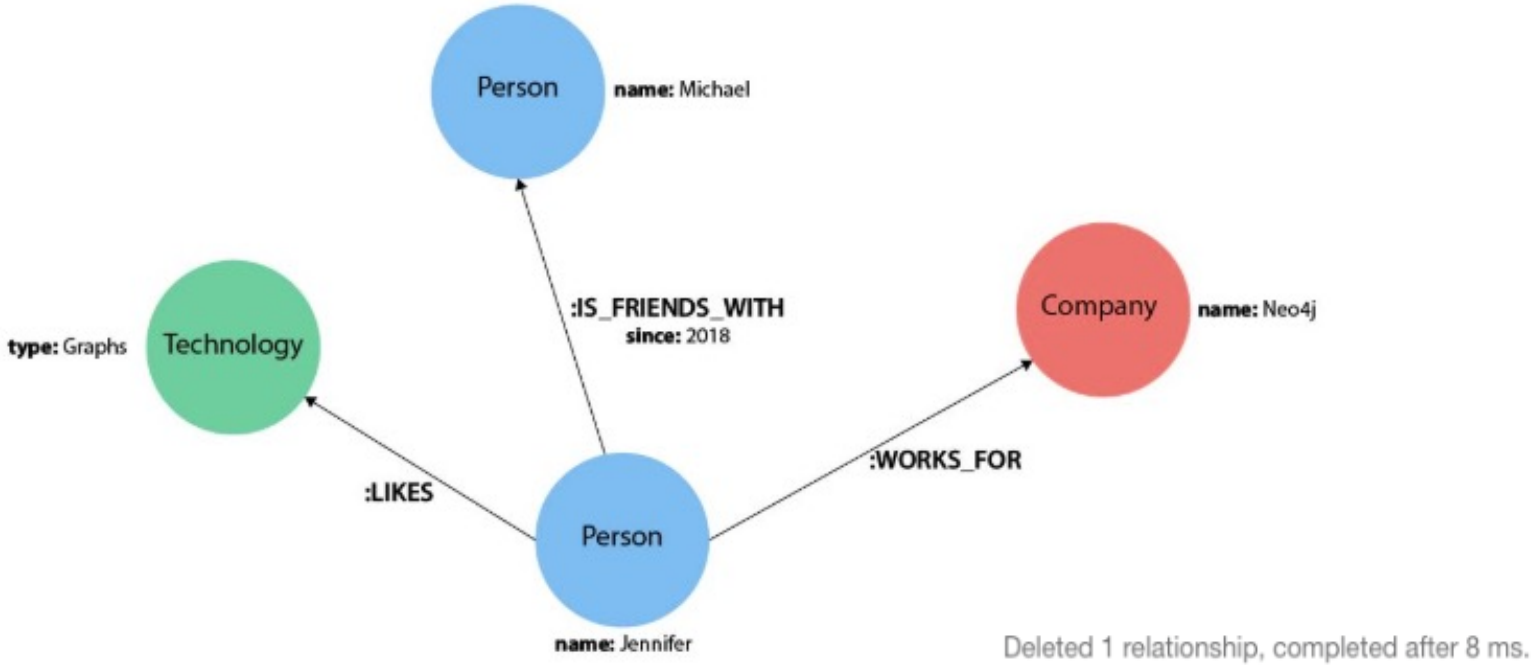
rel

```
{
  "startYear": "2018-01-01"
}
```

```
MATCH (:Person {name: 'Jennifer'})-[rel:WORKS_FOR]-(:Company {name: 'Neo4j'})
SET rel.startYear = date({year: 2018})
```

Delete a relationship

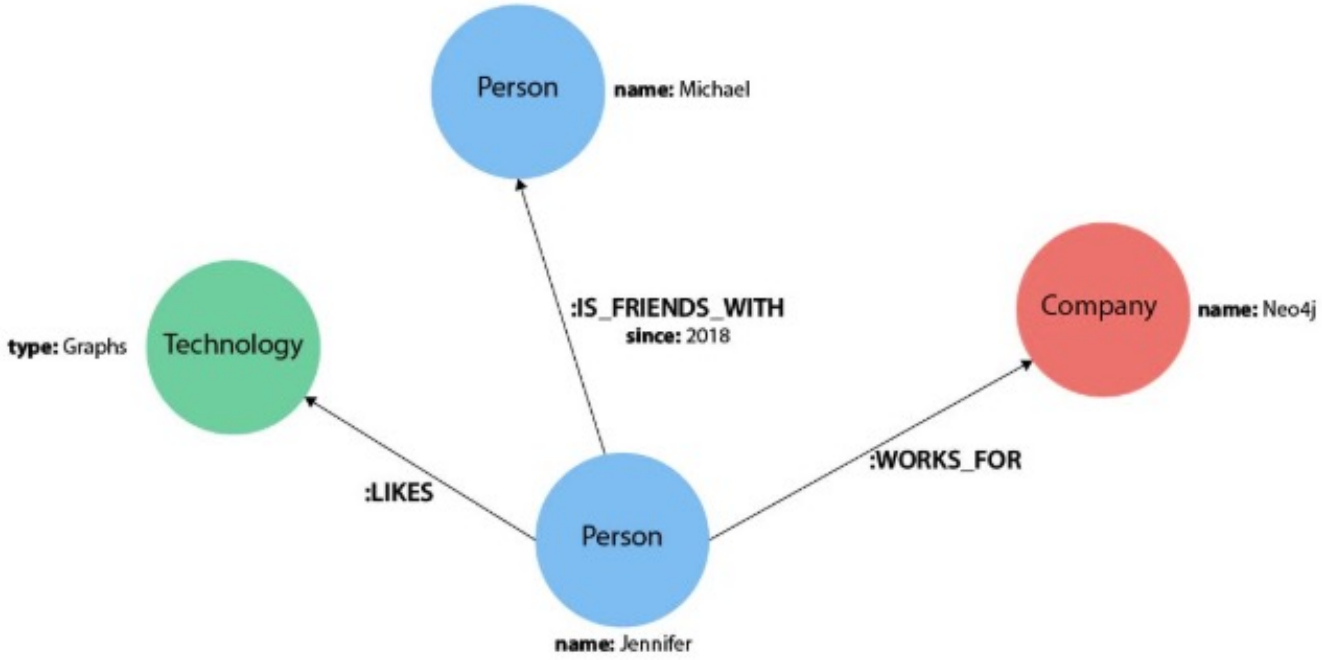
Optional




```
MATCH (j:Person {name: 'Jennifer'})-[r:IS_FRIENDS_WITH]->(m:Person {name: Michael'})
DELETE r
```

Delete a node

Optional



Delete a node with relationship

```
MATCH (m:Person {name: Jennifer'})
DETACH DELETE m
```

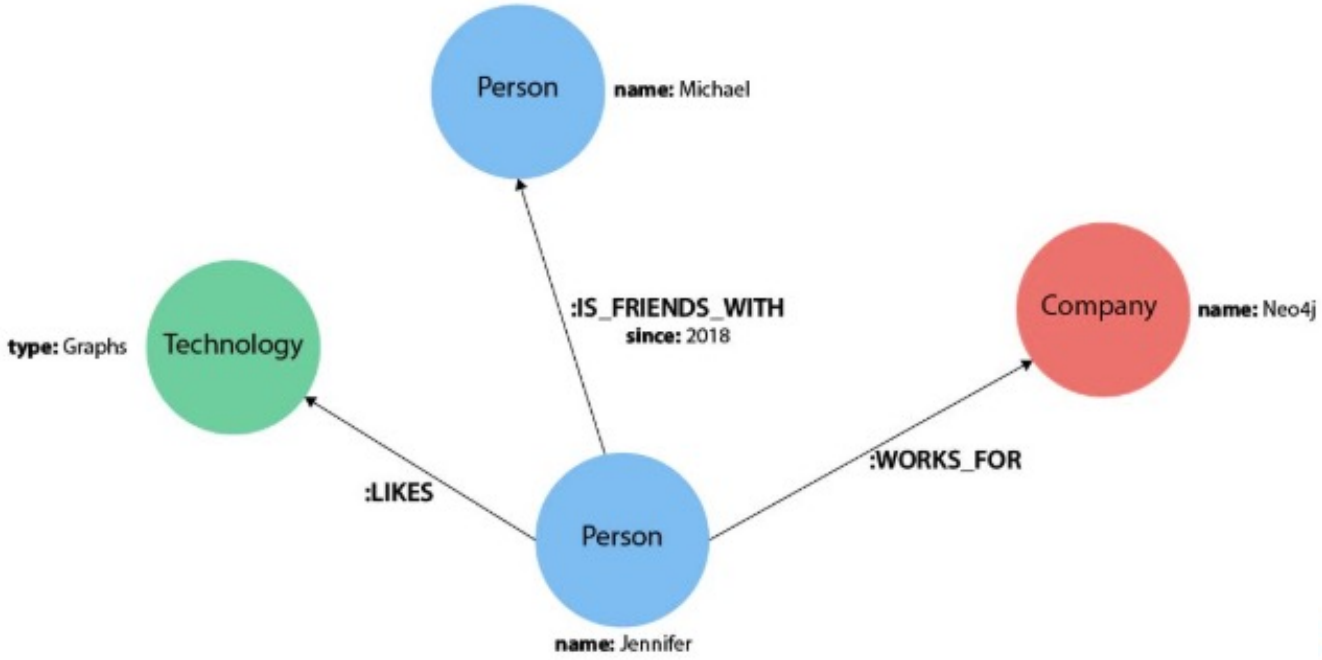
Delete a node without any relationship

Deleted 1 node, completed after 1 ms.

```
MATCH (m:Person {name: 'Mark'})
DELETE m
```

Delete property

Optional




```
//delete property using REMOVE keyword
MATCH (n:Person {name: 'Jennifer'})
REMOVE n.birthdate

//delete property with SET to null value
MATCH (n:Person {name: 'Jennifer'})
SET n.birthdate = null
```

Other operations

Create/modify/delete nodes/edges/properties

Merge nodes

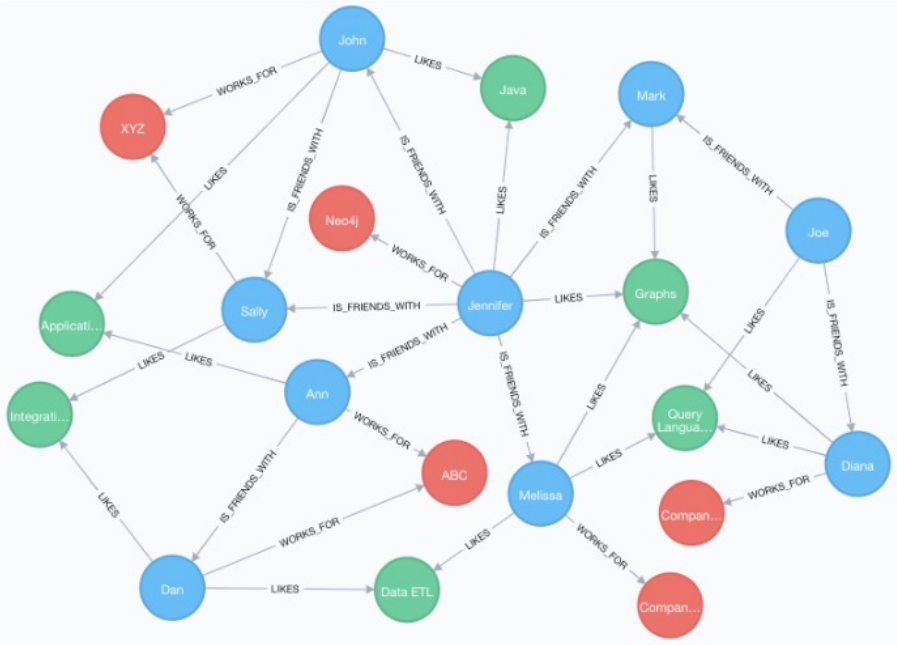
Selection

Where

. . .

Some Complex Patterns

Optional

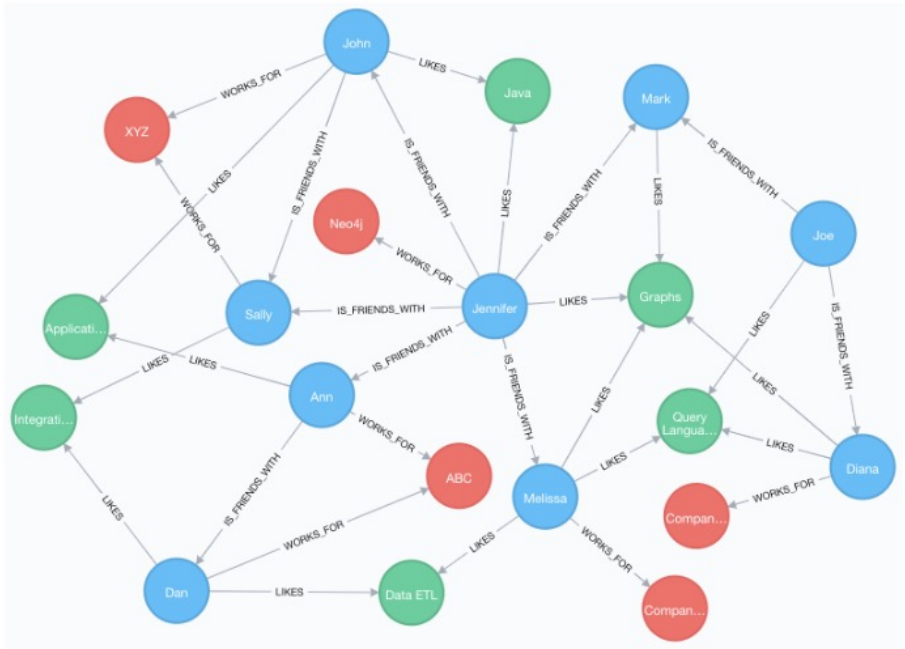



```
//Query1: find which people are friends of someone who works for Neo4j
MATCH (p:Person)-[r:IS_FRIENDS_WITH]->(friend:Person)
WHERE exists((p)-[:WORKS_FOR]->(:Company {name: 'Neo4j'}))
RETURN p, r, friend

//Query2: find Jennifer's friends who do not work for a company
MATCH (p:Person)-[r:IS_FRIENDS_WITH]->(friend:Person)
WHERE p.name = 'Jennifer'
AND NOT exists((friend)-[:WORKS_FOR]->(:Company))
RETURN friend.name
```

Some Complex Patterns (Cont.)

Optional

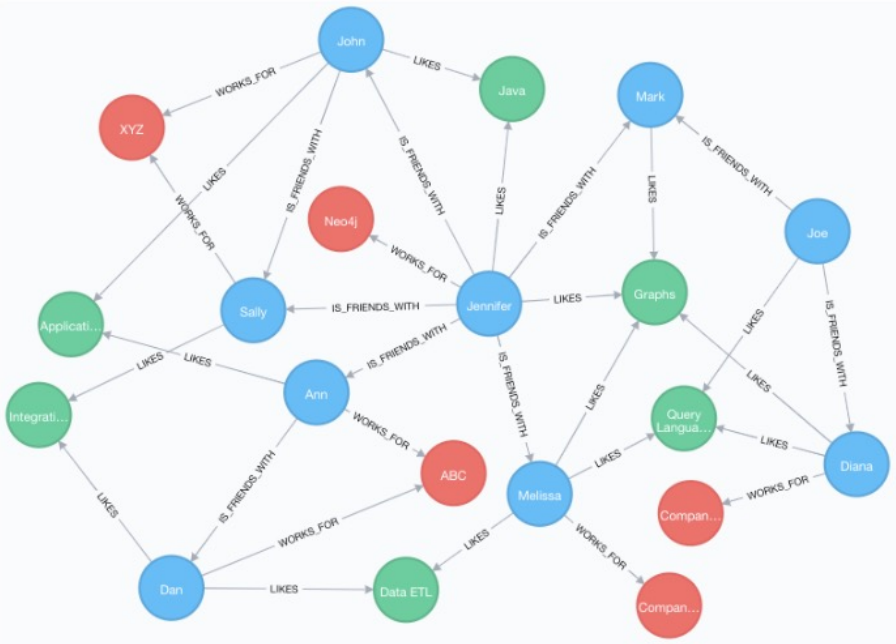


p.name
"Diana"
"Mark"
"Melissa"

```
//Find who likes graphs besides Jennifer
MATCH (j:Person {name: 'Jennifer'})-[r:LIKES]-(graph:Technology {type: 'Graphs'})-[r2:LIKES]-(p:Person)
RETURN p.name
```

Some Complex Patterns (Cont.)

Optional

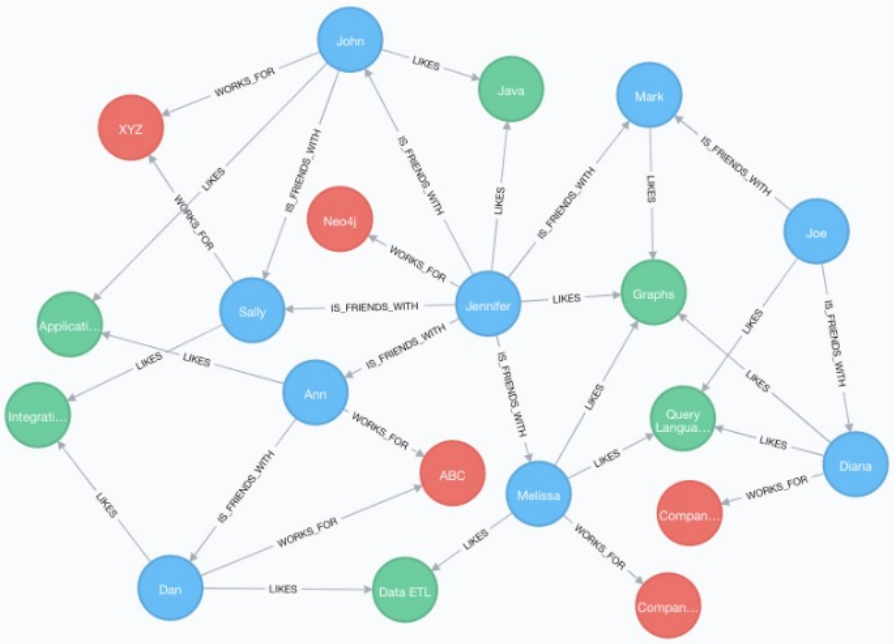


p.name
"Mark"
"Melissa"

```
//Find who likes graphs besides Jennifer that she is also friends with
MATCH (j:Person {name: 'Jennifer'})-[:LIKES]->(:Technology {type: 'Graphs'})<-[:LIKES]-(p:Person),
        (j)-[:IS_FRIENDS_WITH]-(p)
RETURN p.name
```

Aggregating values

Optional



p.name	friend
"Sally"	["John", "Jennifer"]
"Dan"	["Ann"]
"John"	["Sally", "Jennifer"]
"Diana"	["Joe"]
"Jennifer"	["Sally", "John", "Ann", "Mark", "Melissa"]
"Ann"	["Dan", "Jennifer"]
"Mark"	["Joe", "Jennifer"]
"Joe"	["Diana", "Mark"]
"Melissa"	["Jennifer"]

```
MATCH (p:Person)-[:IS_FRIENDS_WITH]->(friend:Person)
RETURN p.name, collect(friend.name) AS friend
```

Algorithms supported by Neo4J

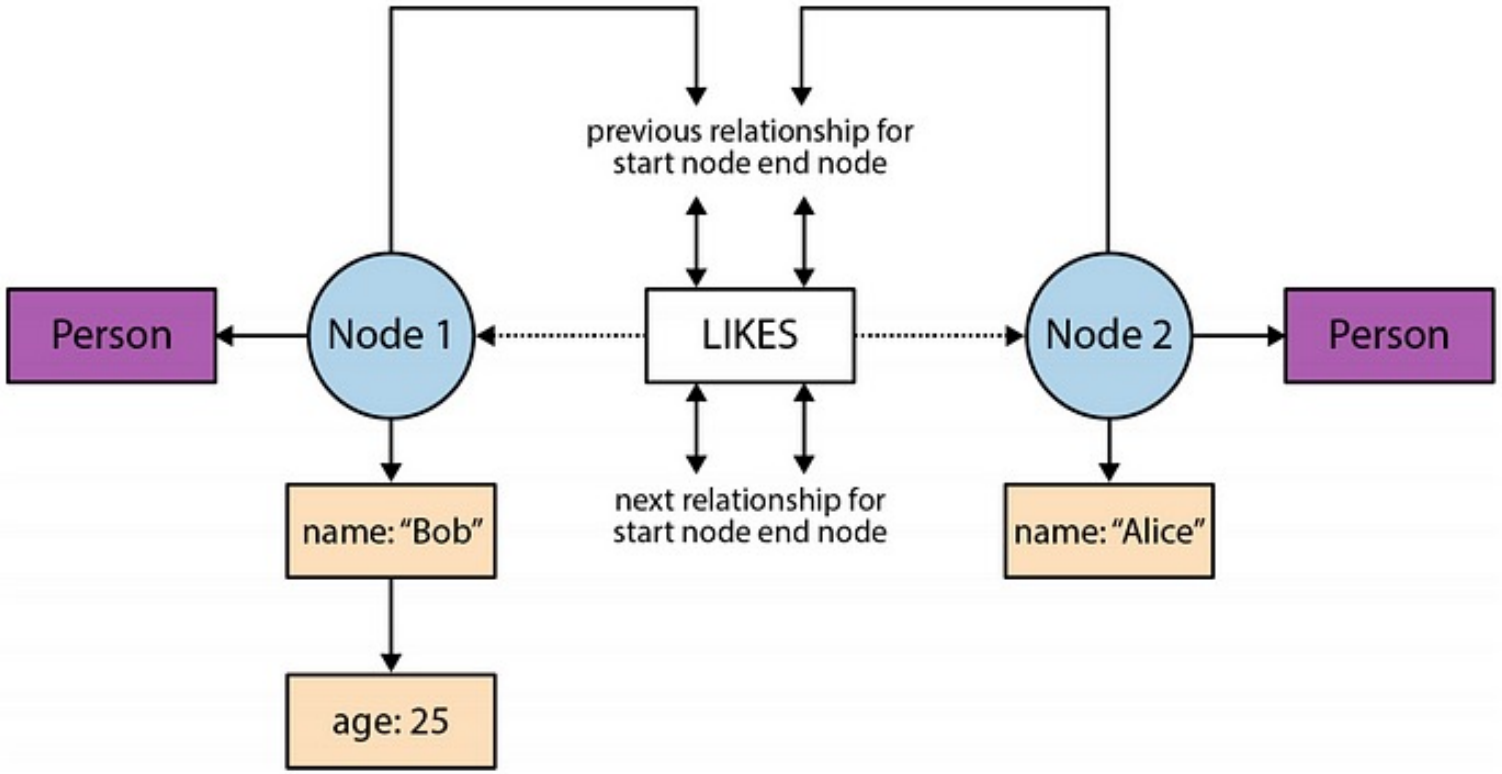
Optional

- Syntax overview
- Centrality
- Community detection
- Similarity
- Path finding
- DAG algorithms
- Node embeddings
- Topological link prediction
- Pregel API

https://neo4j.com/docs/graph-data-science/current/algorithms/

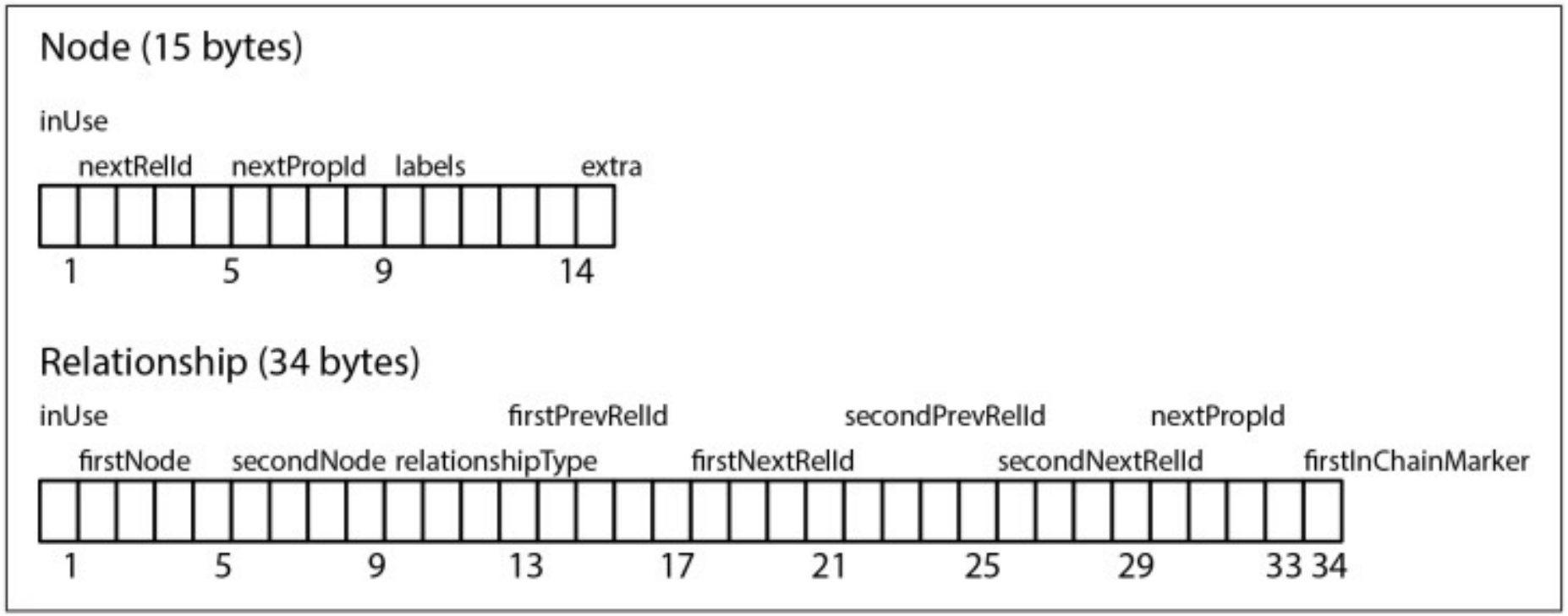
Internal Structure

Optional



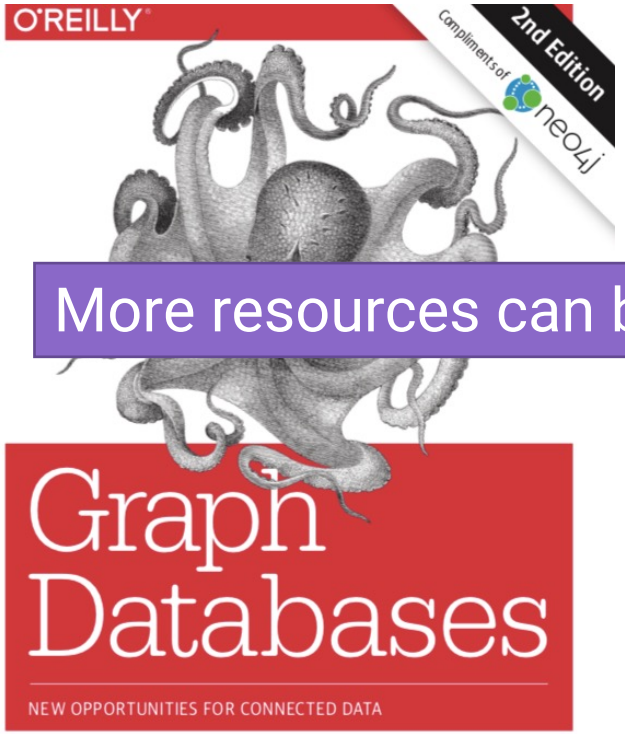
Internal Structure (cont)

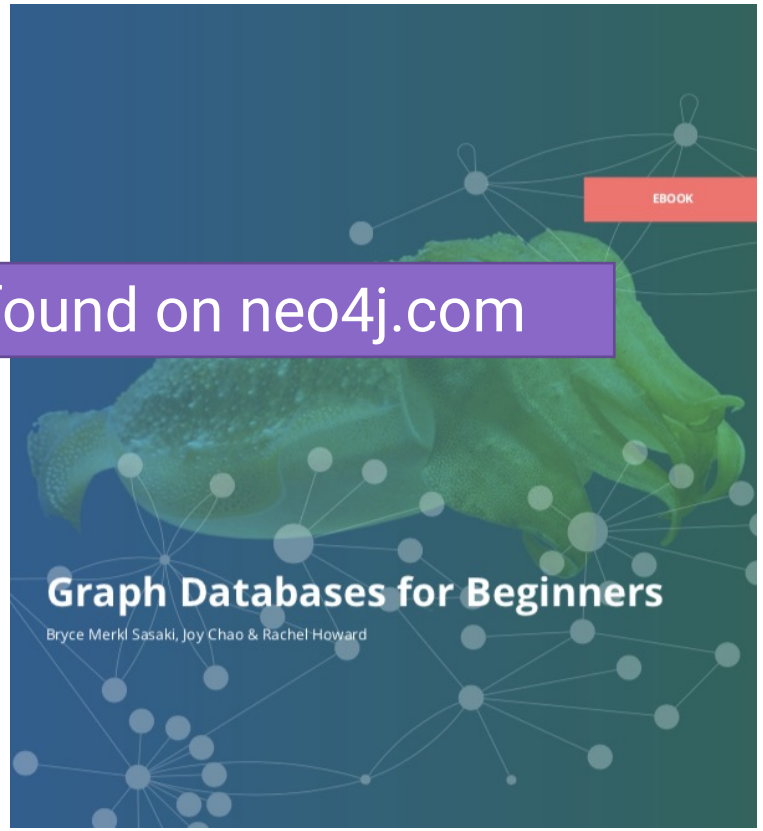
Optional



Resources

Optional





More resources can be found on neo4j.com