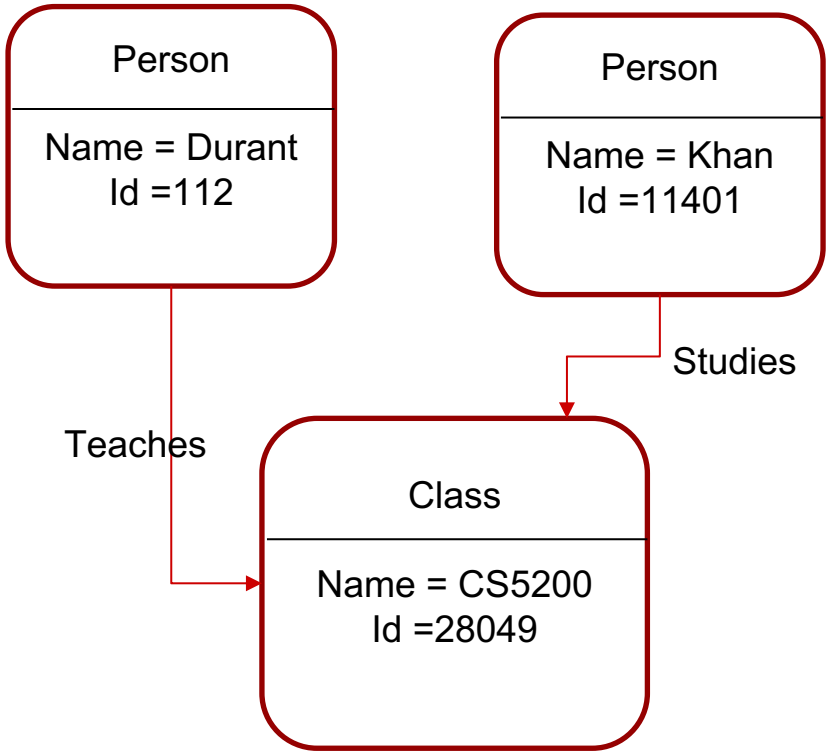
Neo4j

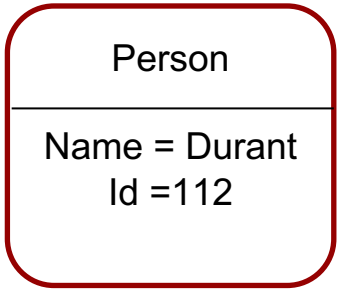
REFERENCE: https://neo4j.com/docs/getting-started/

Neo4j is a graph database



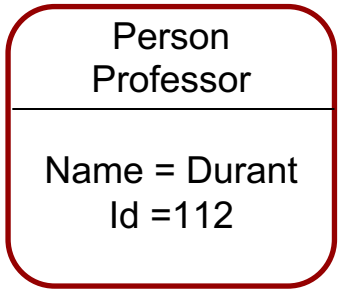
- A graph consists of nodes and edges
- An entity is represented as a node
- An association is represented as an edge and is called a relationship.
- Entities can have labels; labels have a grouping effect on the nodes

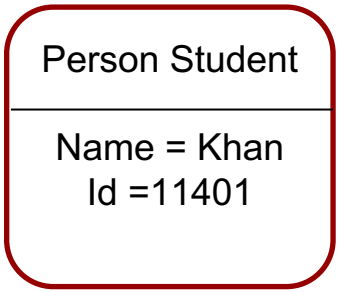
Simplest graph

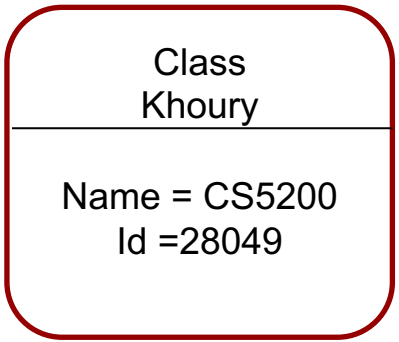


- The simplest graph is a node

Additional labels for an entity

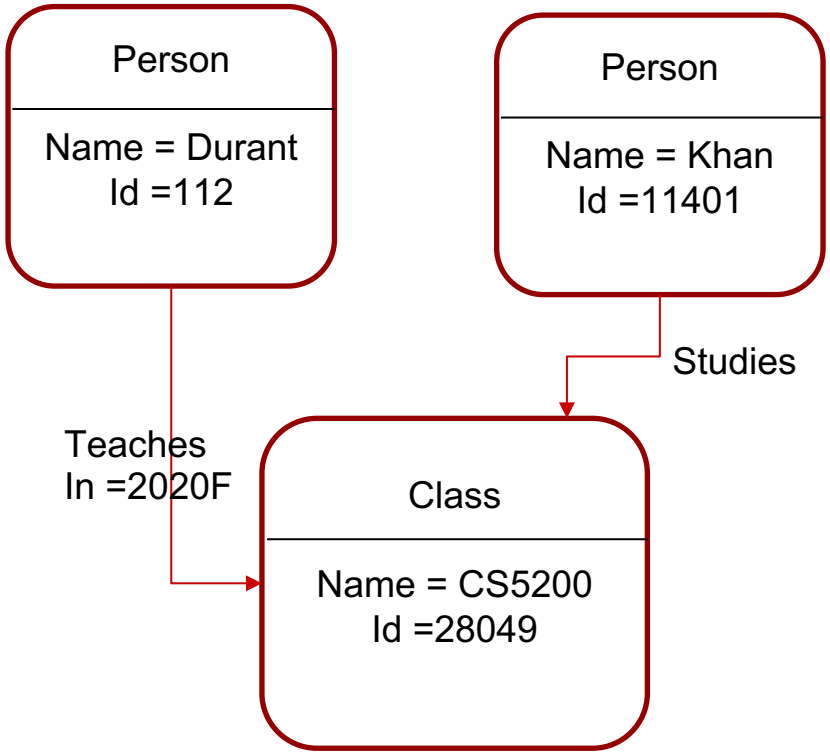






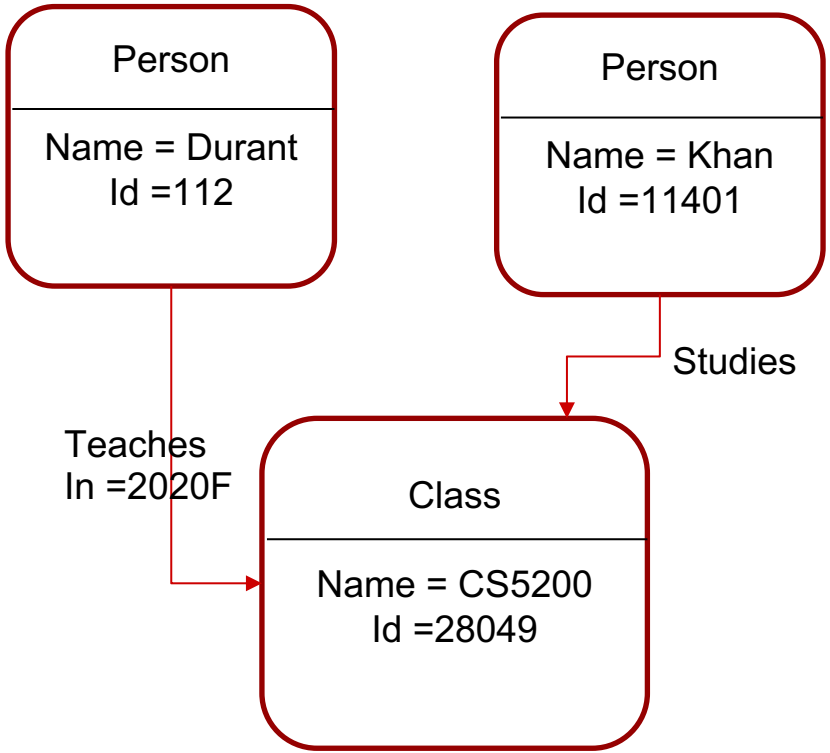
- We can add multiple labels to an entity
- Each label can be used to identify a group of nodes
- Operations can be performed over groups

Relationship connects two nodes



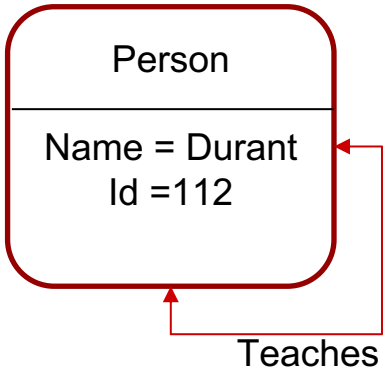
- Relationships define structure in the graph
- A relationship has one and only one type
- In our example, Teaches and Studies are relationship types
- Relationships can have properties as demonstrated by our In property

Relationship direction matters



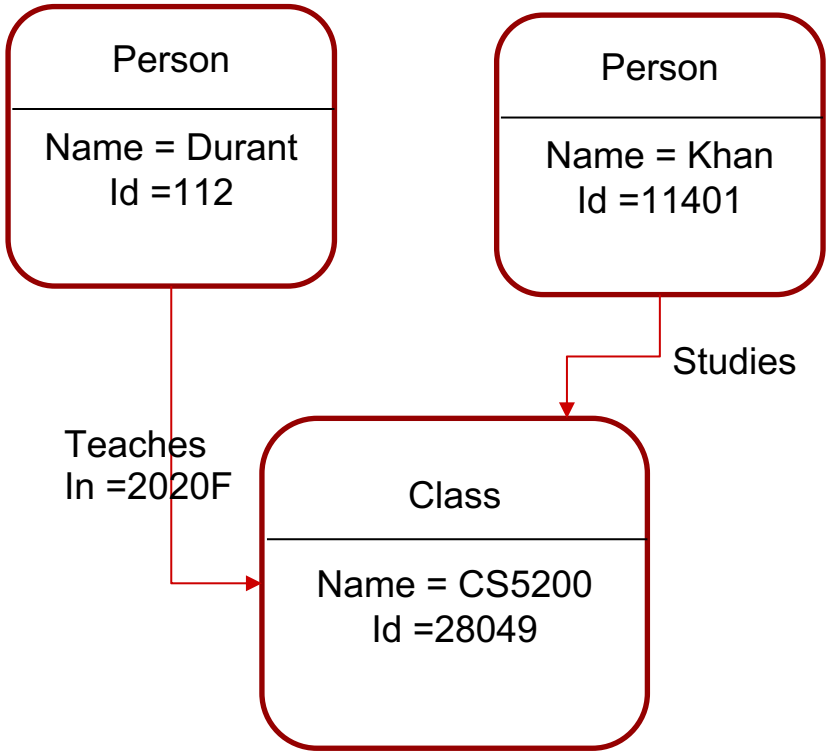
- The Durant node has an outgoing edge
- The CS5200 node has 2 incoming edges
- Typically, on one relationship between 2 nodes need to be represented
- Example: “CS5200 is taught by” need not be represented

Nodes can have relationships with itself



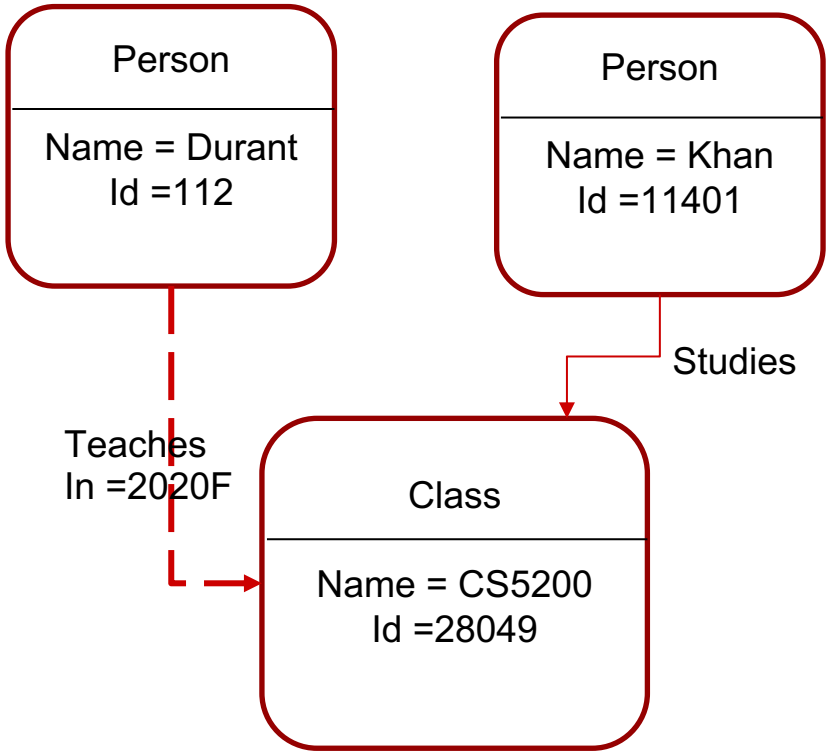
- Recursive relationships are allowed in Neo4j
- In our example, The Durant node can teach herself

Nodes can have properties



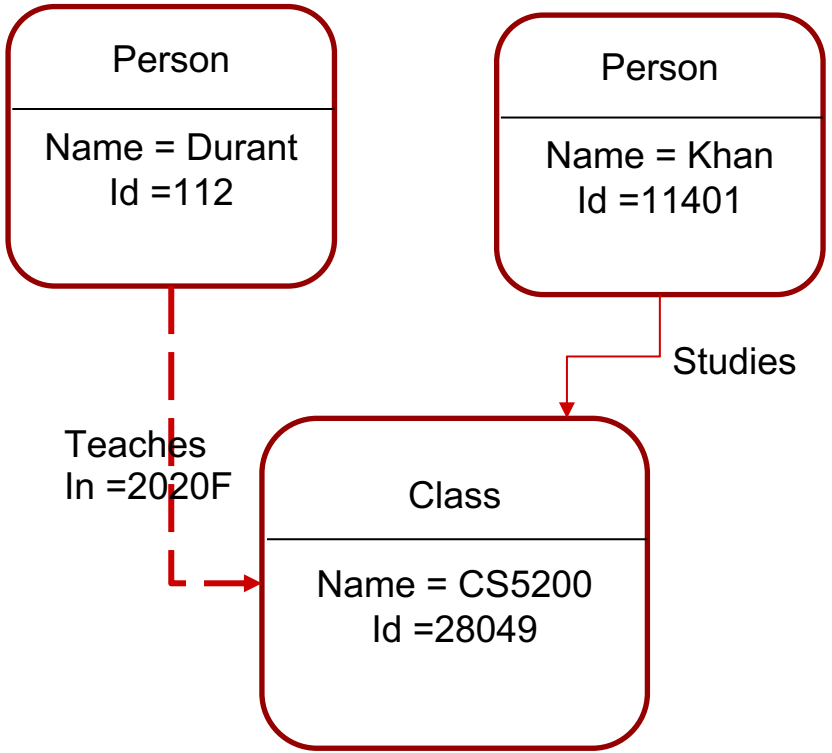
- Properties are name, value pairs
- The value part is not limited to a specific domain
- Any value can be associated with any property name

A schema in Neo4j



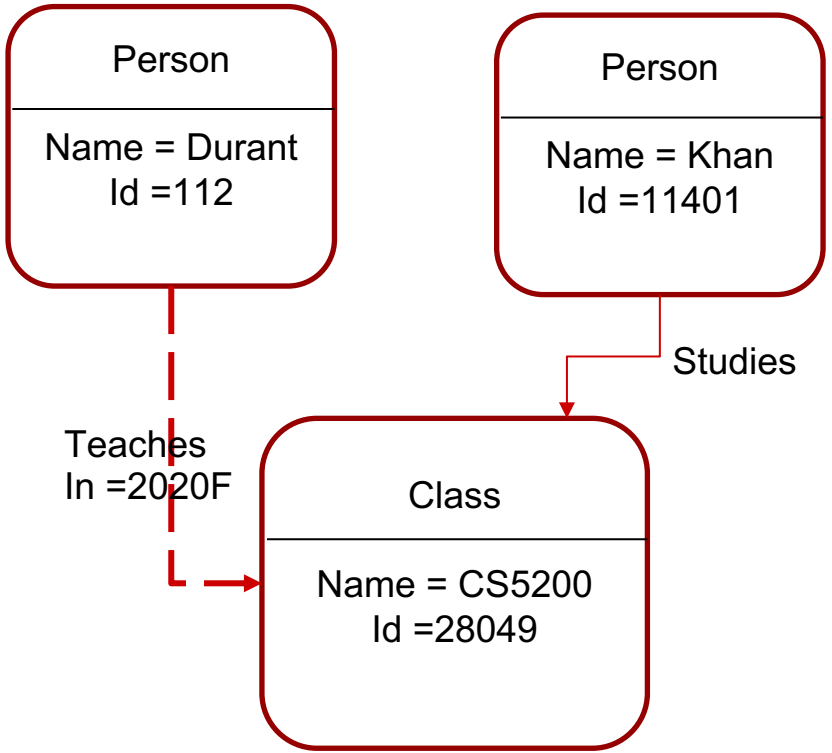
- Defines indexes and constraints
- A schema is optional in Neo4j
- Indexes are data structures that speed up a traversal
- Constraints are rules that the data must adhere to

A traversal is a path through the graph



- We answer questions on the data via a traversal of the graph
- Traversal is visiting a collection of nodes by following relationships
- To answer the query: who teaches CS5200 in 2020F, we would start at CS5200 and follow the appropriate relationship

Cypher, Neo4j's query language



- Cypher allows a user to specify a query for a specific graph
- The query can be used to extract or store data
- Cypher query language specifies nodes, as well as criteria to satisfy
- Like SQL, it is declarative

Cypher

The simplest query is a description of a node. A node is surrounded by braces (). It can be general or specific. For example (CS5200)

You can provide more detail to limit the nodes part of the result

CREATE a node command

Create a specific node with specific properties
Syntax
CREATE (variable:Label {property1: “value”[,…]})
Example
CREATE (Stillson:Person { name: ”Gary", from: ”Norway",})
CREATE clause to create data
() indicate a node
Variable – name of node being created
Label – group variable belongs to
{} Property list

Creating a relationship

Syntax
(*node*)-[:*Relationship_name*]->(*node*)

Example
(js)-[:KNOWS]->(ir)

CREATE create nodes and relationships

```
CREATE (js:Person { name: "Johan", from: "Sweden", learn:
  "surfing" }),
(ir:Person { name: "Ian", from: "England", title: "author" }),
(rvb:Person { name: "Rik", from: "Belgium", pet: "Orval" }),
(ally:Person { name: "Allison", from: "California", hobby:
  "surfing" }),
(ee)-[:KNOWS {since: 2001}]->(js),(ee)-[:KNOWS {rating: 5}]-
  >(ir),
(js)-[:KNOWS]->(ir),(js)-[:KNOWS]->(rvb),
(ir)-[:KNOWS]->(js),(ir)-[:KNOWS]->(ally),
(rvb)-[:KNOWS]->(ally)
```

Example: Specifying a pattern

```
(keanu:Person:Actor {name:  "Keanu Reeves"} )
-[role:ACTED_IN     {roles: ["Neo"] } ]->
(matrix:Movie       {title: "The Matrix"} )
```

For conciseness, these patterns can be assigned to variables

```
acted_in = (:Person)-[:ACTED_IN]->(:Movie)
```

Use relationships to identify a subgraph

Find all of Gary's friends

```
MATCH (ee:Person)-[:KNOWS]-(friends)
WHERE ee.name = ”Gary" RETURN ee, friends
```

MATCH command

Find a specific node with specific properties

Syntax

MATCH (variable:Label WHERE criteria RETURN ret_value)

Example

MATCH (ret:Person WHERE name=”Gary” RETURN ret)

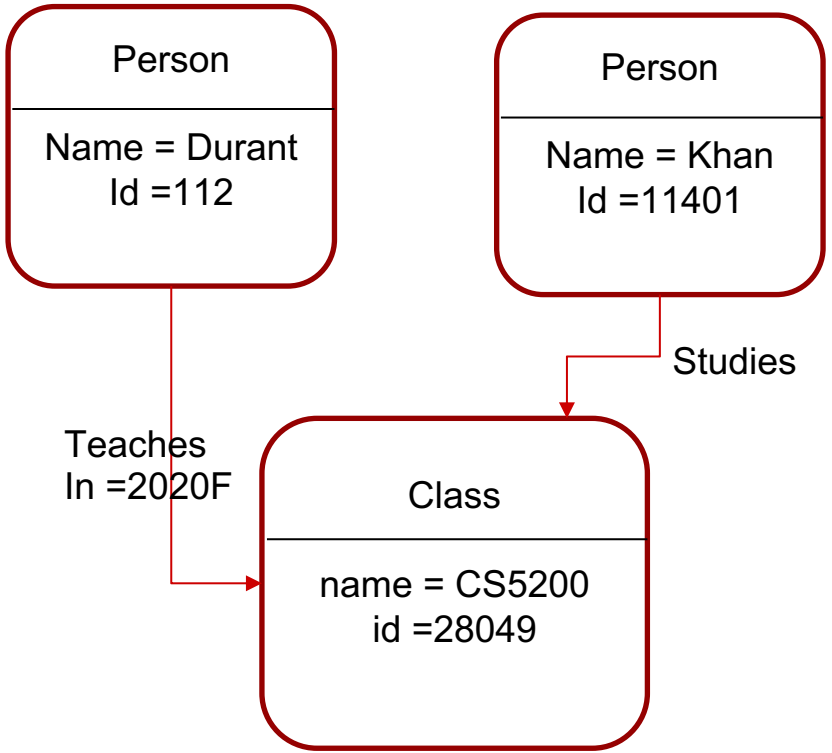
- MATCH: clause to specify nodes or relationships
- WHERE: Criteria to satisfy
- RETURN: value, values to be returned

RETURN command

Specify that data should be returned to a user

```
CREATE (p:Person { name:”Gary Snail", born:1980 })
RETURN p
```

Create this graph




```
CREATE (a:Person {
name:”Durant",
  id:112 })-[r:Teaches { in:
[”2020F"]}]->(c:class {
class:”CS5200",id:28049 })

CREATE (d:Person {
name:”Khan", id:11401 })-
[:Studies]->(c)
RETURN a,d,r,m
```

Extending a graph

To extend the graph with new information, we first match the existing connection points and then attach the newly created nodes to them with relationships.

```
MATCH (p:Person { name:”Durant" })
CREATE (c:Class { title:”CS3200",id:12345 })
CREATE (p)-[r:TEACHES{ In: [‘2021S']}]->(c)
RETURN p,r,c
```

Notice the use of variables in this expression.

MATCH and CREATE combined

Sometimes you are not sure if the data has been added yet to your graph. The MERGE command allows you to specify objects, if not yet created to create them, then match them for other following commands.

Example:

```
MERGE (c:Class { title:”CS3200" })
ON CREATE SET c.id = 12345
RETURN c
```

Aggregating data

You can generate an aggregate per a group by returning a value with the aggregate

Example:
MATCH (:Person)
RETURN count(*) AS people

Returns the number person nodes as the variable people

Other aggregate functions: count, sum, avg, min, and max

Grouping data for aggregates

Data is aggregated in the RETURN statement

Example:
MATCH (:Person)
RETURN role, count(*) AS people

Returns the number of people for each distinct role value

Ordering the returning data

Data is ordered with the ORDER BY statement

Example:
MATCH (:Person)
RETURN ORDER BY id [ASC|DESC]

Returns the person nodes ordered by the id property

There is also the SKIP and LIMIT keyword to specify where to start returning values and how many to return

Combine results

UNION clause allows you to combine results from multiple commands

Create intermediate results

WITH clause allows you to compute an intermediate result and use it in a subsequent command

RDB Concepts to Neo4j

RDBMS	Neo4j	Notes
Database	Graph	Both system have the USE command.
Table, View	Nodes with same Type	Graph is not rigid in its structure
Row	Node	Node is not rigid in its structure
Column	Property	Property is data associated with a node
Index	Index	Same data structure
Join	traversal	Trace from one node to another (slightly different)
Foreign key	Relationship (edge)	Link 2 objects together
Partition	Shard (Fabric)	User can specify the portion of data to be stored on a specific server

Neo4j DB features

- Neo4j is ACID-compliant
- Dynamic schema
  - No DDL
- Graph-based database
- Cypher Query language available via an API
- Choice on consistency
  - Strong consistency
    - Once an object updated, all subsequent reads get update
  - Eventual consistency
    - Replica copies will be updated at an optimal time
- Data retrieved by traversals is not protected from modification by other transactions

Summary

- Neo4j
  - Graph oriented data store, flexible schema, provides a query language via an API, consistent reads if data is set up for strong consistency
  - Provides transactions, relationships and traversal of one node to another in order to answer a user query