Graph Databases

Immanuel Trummer
itrummer@cornell.edu
www.itrummer.org

Outlook: Beyond Relational Data

- Graph data
- Data streams
- Spatial data

Outlook:
Beyond Relational Data

- **Graph data**
- Data streams
- Spatial data

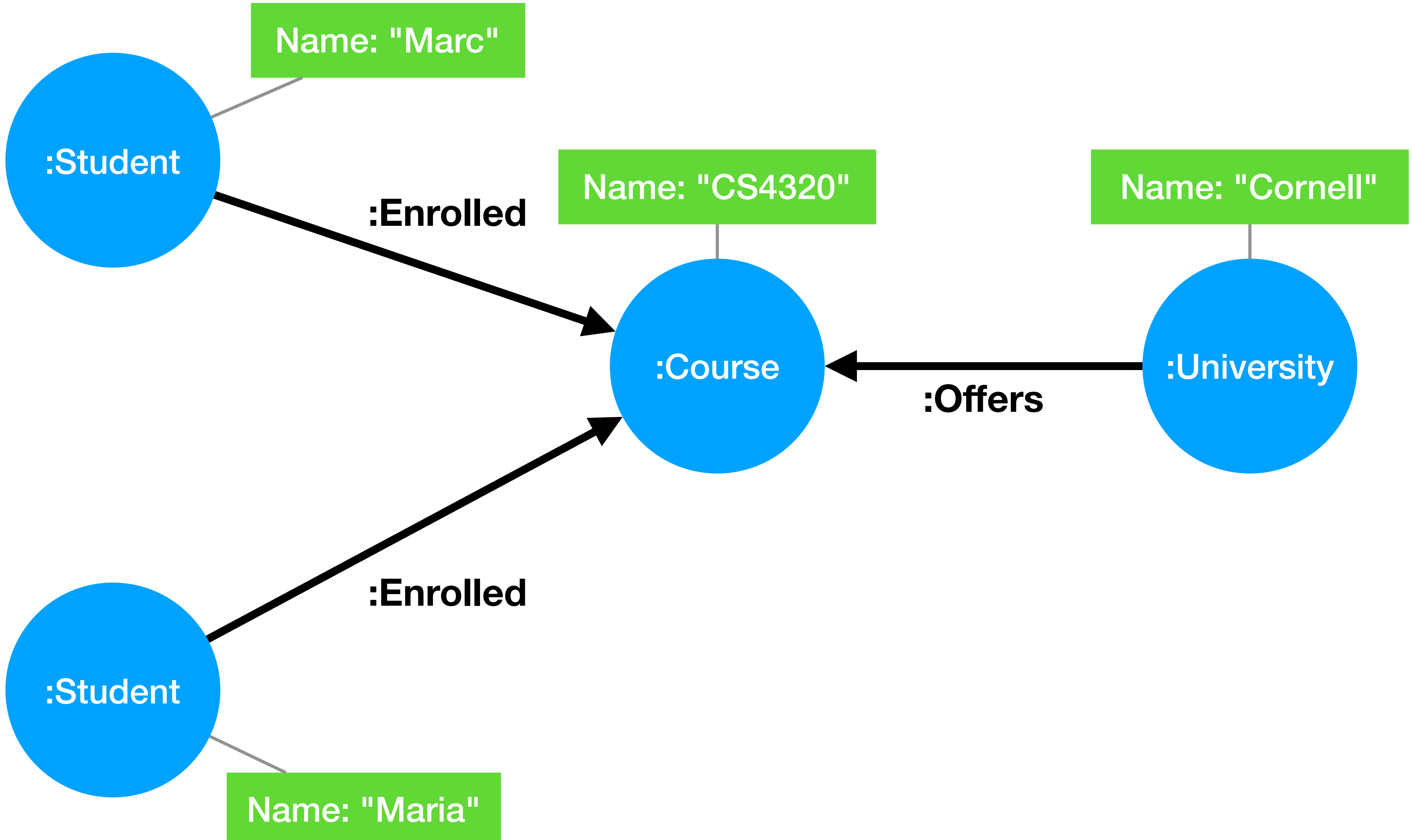
Reading List

- "*Graph Databases*" by I. Robinson et al.
- "*Graph Databases Comparison: AllegroGraph, ArangoDB, InfiniteGraph, Neo4J, and OrientDB*"
  by Fernandes and Bernardino.
- http://www.Neo4j.com

Graph Data

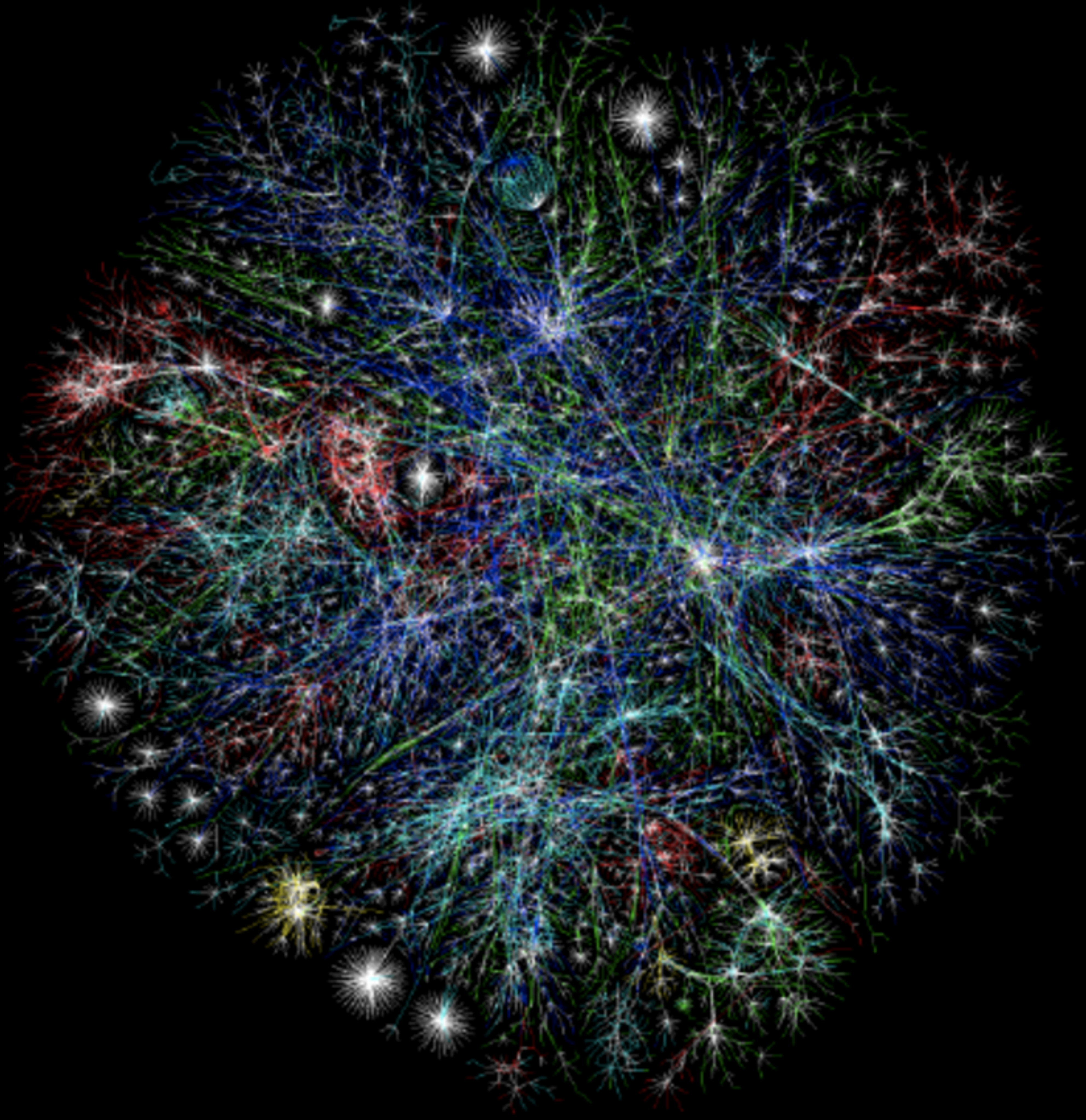
- A set of **nodes** and a set of **edges** connecting nodes
- Nodes and edges can be associated with **labels**
- Nodes and edges can be associated with **properties**

Example (Toy) Graph



Motivation

- **Social** networks
- **Knowledge** graphs
- **Communication** graphs
- **Road** networks
- ...



Communication Structure of the Internet as Graph.

Visualization of Facebook Connections as Graph.

NYC Metro Graph

NYC Metro Graph

All Paths?

How to represent Graph as Relational DB?

Relational Representation

- CREATE TABLE **Stations**(
  StationID int primary key, name text);

- CREATE TABLE **Connected**(
  StationID1 int, StationID2,
  primary key (StationID1, StationID2),
  foreign key (StationID1) references Stations(StationID1),
  foreign key (StationID2) references Stations(StationID2)
  );

Query: Find Paths from Port Authority to NYU?

Find Paths from P to N

- SELECT * from Connected C1
  join Connected C2 on (C1.stationid2 = C2.stationid1)
  join Connected C3 on (C2.stationid2 = C3.stationid1)
  ... join Connected Cn ...
  WHERE C1.name = 'Port Authority'
  and Cn.name = 'NYU'

- Retrieves paths with a fixed length (can iterate)

- (Can be solved better with advanced SQL features)

(Intermediate) Conclusions

- Storing graph data in relational DBMS is **possible**
- But querying graphs via vanilla SQL is **inconvenient**
- Also, may increase **efficiency** by graph specialization

Graph Database Systems

FlockDB

JanusGraph

OrientDB®

ArangoDB

Dgraph

neo4j

Graph Database Systems

FlockDB

JanusGraph

OrientDB®

ArangoDB

Dgraph

neo4j

Cypher

- **Graph query language** used by Neo4j
  - Allows **creating/updating** nodes and relationships
  - Allows **searching** graphs for complex patterns
  - **Aggregation**, filtering, sub-queries etc.
  - Inspired by **SQL** in some aspects

Creating Nodes

- **CREATE ()**
  Create node without labels or properties

- **CREATE (:Student)**
  Create node labeled as student, no properties

- **CREATE (:Student {name : 'Marc'})**
  Create node labeled as student, name set to 'Marc'

Finding Nodes

- **MATCH (m:Student {name : 'Marc'})**
  - Finds nodes labeled as "Student"
  - Name property must be set to "Marc"
  - Match result is assigned to variable m
  - Variable m can be used in remaining query

Creating Relationships

- **MATCH (a:Student {name: 'Marc'}),
  (b:Course {name: 'CS4320'})
  CREATE (a)-[:Enrolled {semester: 'FS20'}]->(b)**

    - Matches a to students with name "Marc"
    - Matches b to courses with name "CS4320"
    - Inserts edge from a to b with label "Enrolled"
    - Edge has property "semester" set to "FS20"

Updating Nodes

- **MATCH (m:Student {name: 'Marc'})**
  SET m:Alumnus
  - Changes label of Marc from Student to Alumnus
- **MATCH (m:Student {name: 'Marc'})**
  SET m.name = 'Marcus'
  - Changes value of name property to "Marcus"

Finding Relationships

- **MATCH (a:Student {name: 'Marc'}) -[e:Enrolled {semester: 'FS20'}]- (b:Course {name: 'CS4320'})**
    - Find edges connecting nodes a and b such that
        - Node a is a student with name 'Marc'
        - Node b is a course with name 'CS4320'
        - Edge labeled "Enrolled", property semester is "FS20"
    - Assign resulting edges to variable e

Updating Relationships

- **MATCH (a:Student {name: 'Marc'})
  -[e:Enrolled {semester: 'FS20'}]-
  (b:Course {name: 'CS4320'})
  SET e.semester = 'FS21'**

  - Get edge representing enrollment of Marc in CS4320

  - Update value of semester property to "FS21"

Deletions

- **MATCH (a:Student {name: 'Marc'})**
  DELETE a

  - Deletes students with name "Marc" from the database

(Demo)

Exercise: Create Graph DB

- Create a graph DB representing the following situation
- **Ithaca** and **Binghamton** are cities located in NY state
- **Cornell** University is located in Ithaca
- Cornell offers a course with name "**CS4320**"

Pattern-Based Retrieval

- **MATCH (:Student {name: 'Marc'})**
  -[:friendsWith]-> (s:Student)
  RETURN s

  - Returns all friends (students) of student Marc

Pattern-Based Retrieval

- **MATCH (:Student {name: 'Marc'}) -[:friendsWith*]-> (s:Student) RETURN s**
  - Returns all friends (students) of Marc, **their friends, the friends of their friends, etc.**

Pattern-Based Retrieval

- **MATCH (:Student {name: 'Marc'})**
  -[:friendsWith*0..2]-> (s:Student)
  RETURN s

  - ***Any suggestions: what does this (probably) return?***

Aggregation

- **MATCH (:Student {name: 'Marc'}) -[:friendsWith]-> (:Student) RETURN count(*)**
    - Count number of friends of Marc

Complex Patterns

- **MATCH (s1:Student) -[:friendsWith]->(s2:Student), (s1)-[:Enrolled]->(c:Course), (s2)-[:Enrolled]->(c) WHERE s1.name IN ['Marc', 'Maria'] AND NOT c.name = 'CS4320' RETURN s2**

  - Friends of Marc and Maria who have at least one course in common with them, excluding CS4320

Retrieve Courses Taken by At Least One Student Who Also Takes CS4320!

(Initial Example)

- **MATCH (s1:Station) -[:Connected*]- (s2:Station) RETURN ***

Data Layout

- In-memory data layout is optimized for **fast traversals**
- **Nodes** stored with label, properties, and edge references
  - Node stores list of incoming and outgoing edges
- **Edges** stored with label, properties, and node references

Query Processing

- Query plans composed from **standard operators**
  - Most known from SQL: filtering, projection, ...
  - A few graph-specific operators (e.g., shortest path)
- Can use **indices** to retrieve specific nodes/edges
- Query plans are selected via cost-based **optimization**

Transaction Processing

- Neo4j supports **read-committed** isolation by default
- Acquire locks manually to achieve **higher isolation** level
- Uses logging to persistent storage to achieve **durability**
- Overall: can support **ACID**