Graph Query Languages

ANNA QUERALT, OSCAR ROMERO

(FACULTAT D’INFORMÀTICA DE BARCELONA)

Foundations of Graph Query Languages

Declarative languages to query the graph

- Typically, it matches to an **extended version of pattern matching**
- For pattern matching, every current graph database engine chooses a **fixed semantics**
  - Homomorphism
  - Isomorphism
    - Strict
    - No-repeated-node
    - No-repeated-edge

APIs providing implementation of graph metrics or (label-constrained) shortest-path

- Depending on the metric or algorithm, it maps to adjacency, reachability or pattern matching

Types of Queries

In graph databases we can distinguish certain types of queries, since each of them maps to a different access plan:

- Adjacency queries
  - Neighbourhood queries require accessing the basic data structure and navigate it (i.e., find a node and follow its edges)
- Regular path queries (or navigational graph patterns)
  - Combine pattern matching and reachability: require specific graph-oriented algorithms
- Complex graph patterns
  - Additional expressivity: Grouping, aggregations, set operations (union, difference, etc.), inequalities, ...

Adjacency Queries

Depend on the internal database structures

- Time to find a node or an edge depends on how the graph data structures are implemented (thus, different performance for each database)
  - See the graph databases session

- Once a node / edge is found, time to find its adjacent / incident neighbours

Navigational Graph Patterns

*Navigational graph patterns (**NGP**s)* or *regular path queries (**RPQ**s)* refer to an extended algorithm typically implemented in graph databases that mixes pattern matching and reachability

RPQs **extend** the BGP definition by allowing regular expressions on edges to describe path queries as part of the pattern. A path is described as:

$$x \xrightarrow{\alpha} y \text{ over } G$$

- ***x*** and ***y*** are nodes in *G*
- ***α*** is a regular expression over *Lab* (the set of labels in *G*)

Path Queries

The regular expressions evaluated differ from language to language. The most usual ones are:

- (*) Kleene star and ($^+$) Kleene plus
- (○) Concatenation friend concat lives -> friend then lives
- ($^-$) Inverse
- (|) Union

… and combinations of them

Activity

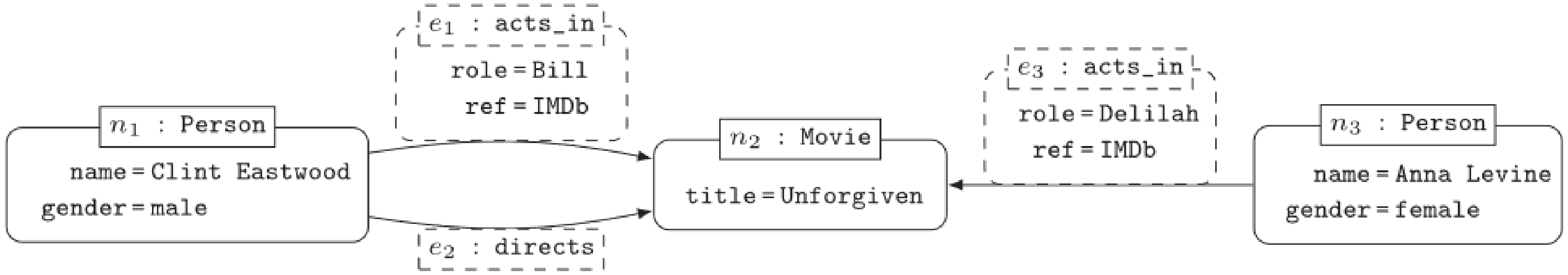
In notebook 1

Objective: Understand how RPQs extend pattern matching

Assume a graph containing relationships and nodes like the ones shown below

- Define a RPQ including expressions from the previous slide to find *all co-actors of all actors*
  - Which are the solutions you will get?
- Define a RPQ to retrieve *all actors you can reach by (transitivetely) following the co-actoring relationship, at least once*
- Define a RPQ to find *all persons that participate in the same movie*

In all cases, think whether the RPQs requested could be represented by means of a BGP.



Complex Graph Patterns

RPQs are equivalent to conjunctive queries without projections (i.e., joins and equality selections)

However, database languages (typically based on the relational algebra) are richer than that

GraphQL was the first graph algebra extending RPQs with relational-like operators

- Union
- Difference
- Left-outer join / Optional
- Selection / Filter – considering inequalities on properties

GraphQL was the first formal graph language presented (2008) and included RPQs and complex graph patterns

He et al. Graphs-at-a-time: Query Language and Access Methods for Graph Databases. SIGMOD'08 (https://people.csail.mit.edu/tdanford/6830papers/he-graphs-at-a-time.pdf)

Cypher

NEO4J’S QUERY LANGUAGE

Neo4J Data Model

It is a graph!

- Use nodes to represent entities
- Use relationships to represent the semantic connection between entities
- Use node properties to represent entity attributes plus any necessary entity metadata such as timestamps, version numbers, etc.
- Use relationship properties to represent connection attributes plus any necessary relationship metadata, such as timestamps, version numbers, etc.

Unique constraints (~PK) can be asserted

- `CREATE CONSTRAINT ON (book:Book) ASSERT book.isbn IS UNIQUE`
- An index is also added to the property (in all the nodes with the indicated label)

Cypher

Created by Neo4j

- Nowadays, standard de facto adopted by other graph databases (OpenCypher)

High-level, declarative language

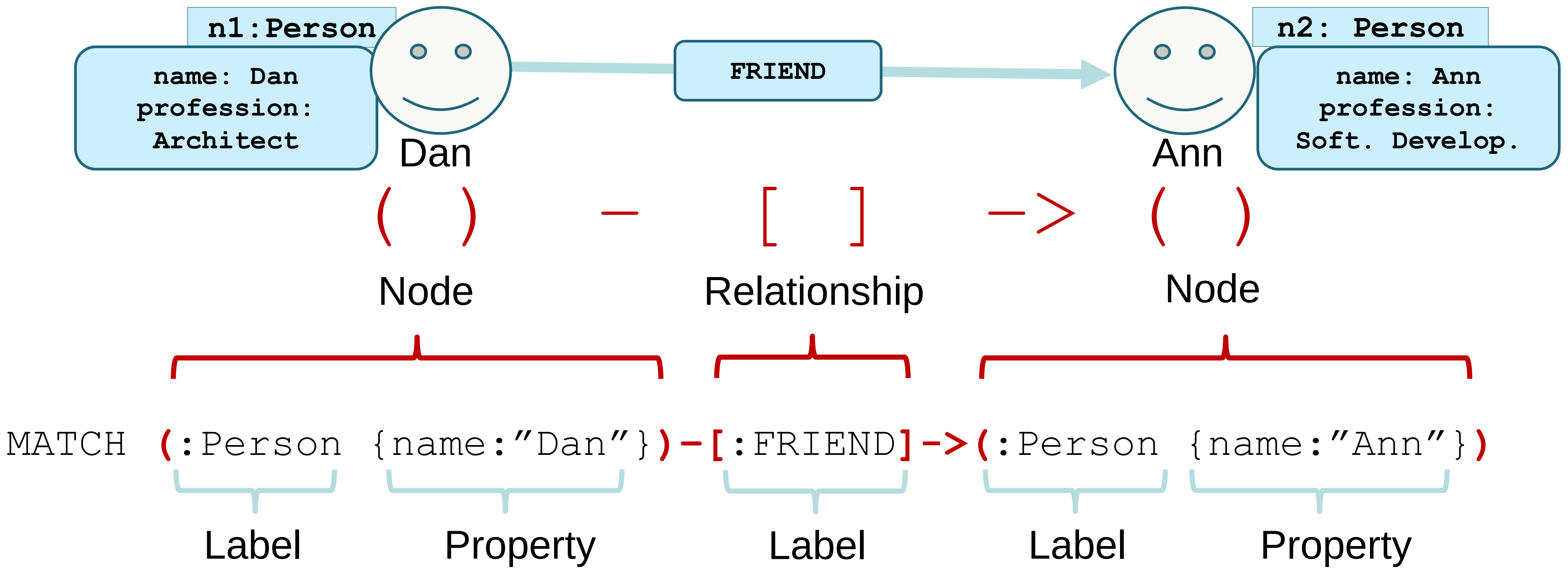
- It is both a data definition language (DDL) and a data manipulation language (DML)

Allows navigational graph patterns

- Except concatenation (as an operator)

It applies pattern matching under **no-repeated-edge isomorphism semantics**

Cypher: Intuition



Cypher Clauses

- DML:
  - MATCH: The graph **pattern** (bgp / rpq) to match
  - WHERE: Additional **constraints / filtering** criteria
  - WITH: Allows query parts to be **chained** together, piping the results from one to the next
  - RETURN: What to include in the query **result** set

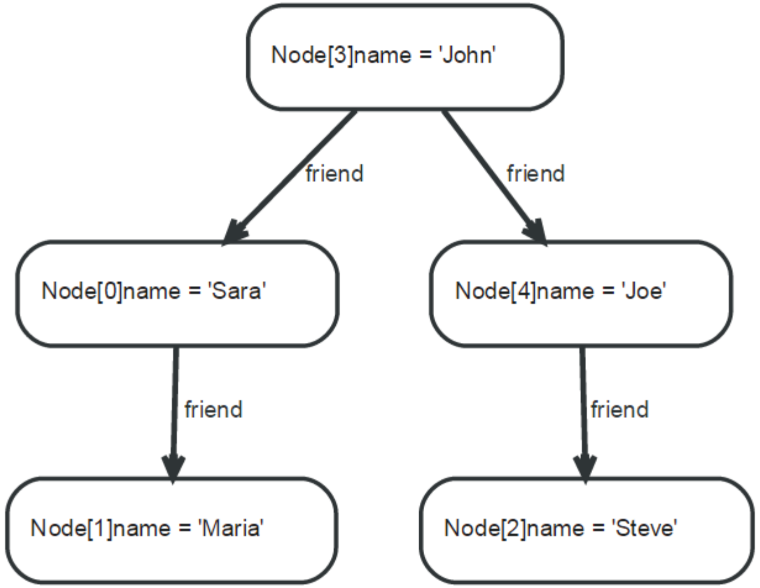
- DDL:
  - CREATE (or MERGE): **Creates** nodes and relationships (if they don't exist)
  - DELETE: **Removes** nodes, relationships and properties
  - SET: Set **values** to properties

- DML and DDL clauses can be combined in a single query

https://neo4j.com/docs/cypher-manual

https://neo4j.com/docs/cypher-refcard

Cypher: Example



Variable   variable

```
MATCH (john {name: 'John'})-[:friend]->()-[:friend]->(fof)
RETURN john, fof
```

john	fof
Node[3]{name:"John"}	Node[1]{name:"Maria"}
Node[3]{name:"John"}	Node[2]{name:"Steve"}

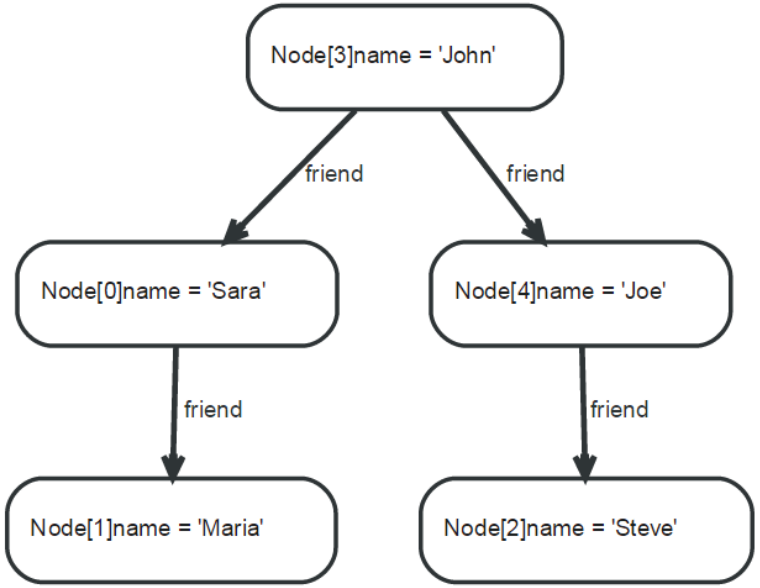
2 rows

Activity

In notebook -2

Objective: Basics on Cypher

Given the following graph, write the Cypher query for the next statements:



MATCH (n)
RETURN n

1) Return all nodes

2) Return all edges

MATCH ()-[r]->()
RETURN r

3) Return all friends of 'John'

MATCH (john:Person {name: 'John'})-[:FRIEND]-(friend)
RETURN friend

MATCH (sara:Person {name: 'Sara'})--(adjacent)
RETURN adjacent

4) Return all nodes adjacent to 'Sara'

5) Return all nodes reachable from 'Sara' in any direction

MATCH (sara {name: 'Sara'})-[*]-(reachable)
RETURN reachable

6) Return the friends of those persons that have 2 friends

MATCH (p) - [:FRIEND] -> (f1), (p) - [:FRIEND] -> (f2)
RETURN p,f1,f2

Cypher: Group By and Aggregates

```
MATCH (n)-[r]->() RETURN n, count(*);
```

Returns the number of nodes related to each node n

List of distinct nodes along with the count of outgoing relationships from each node

```
MATCH (n) RETURN n.name, count(*);
```

Groups n by name and returns the number

List of distinct names along with the count of occurrences of each name in the graph

```
MATCH (david {name: 'David'})--(otherPerson)-->()
WITH otherPerson, count(*) AS num_fof
WHERE num_fof > 1
RETURN otherPerson.name
```

Returns the name(s) of the person(s) connected to 'David' with more than one outgoing relationship

Aggregates: https://neo4j.com/docs/cypher-manual/current/functions/aggregating

Cypher: Pipelines

Cypher applies a data pipeline, where each stage is a MATCH-WHERE-WITH…RETURN

It allows the definition of aliases to be passed between stages

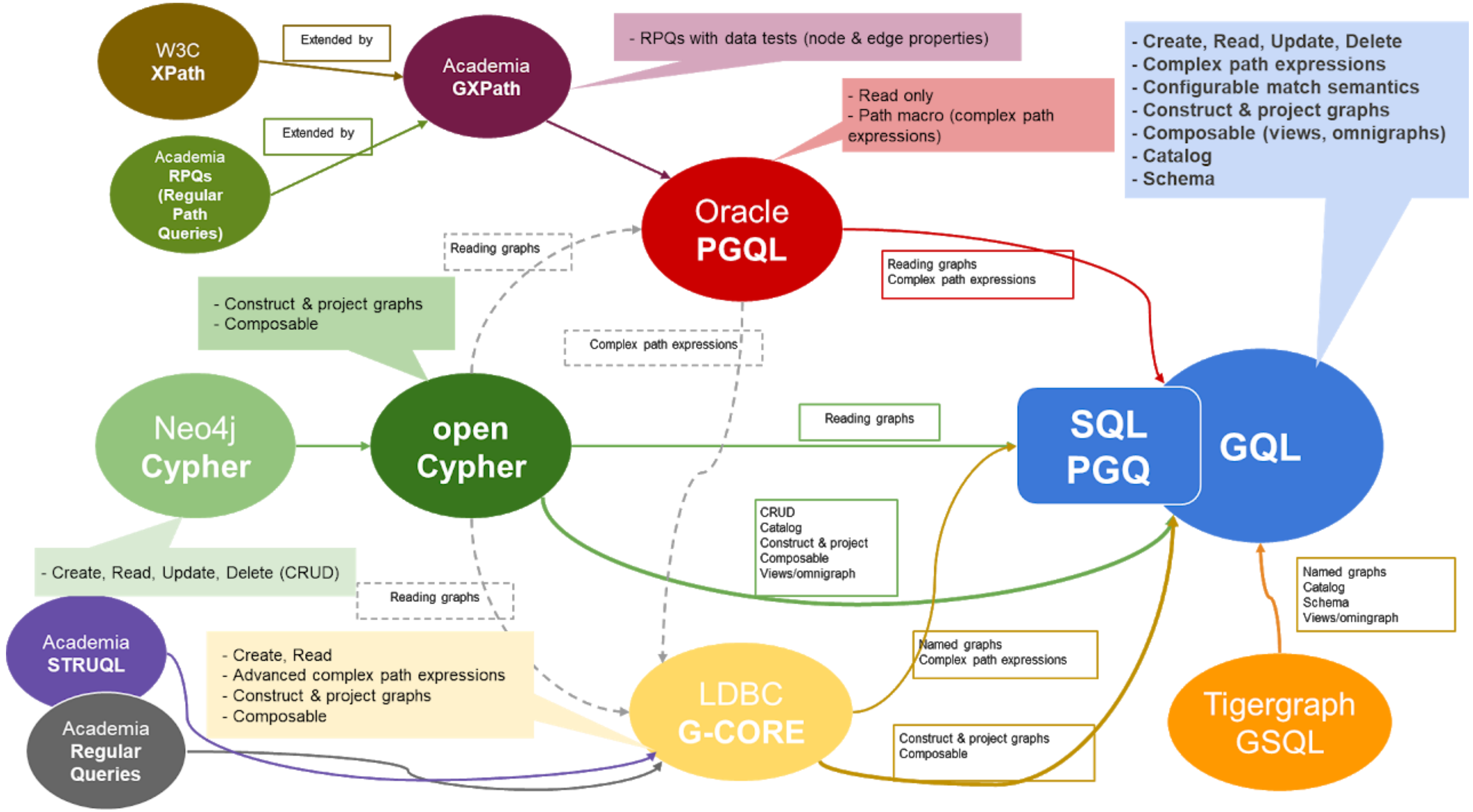
```
MATCH (m:Movie)<-[r:RATED]-()
WITH m, avg(r.rating) AS rating   //inline processing + passing of data
ORDER BY rating DESC LIMIT 5
MATCH (m)<-[:ACTED_IN]-(a:Person)
RETURN m.title, collect(a.name) AS cast, rating
```

Returns the title, cast (as a list), and rating of the 5 movies with the highest ratings

GQL

Graph Query Language

There is currently a big effort towards standardization: https://www.gqlstandards.org/



Summary

Graph languages have been strongly formalized

- Computational complexity deeply studied

Navigational pattern matching as keystone

- Pattern matching
- Reachability

Complex graph patterns

- Extends navigational pattern matching with relational-like operators
- Necessary to unleash the power of graphs for data integration, OLAP or advanced data analytics

Most popular languages

- Cypher (and OpenCypher), Gremlin, GraphQL
- Unfortunately, no standard yet (but coming soon…)

Thanks! Any Question?