Graph Query Languages

ANNA QUERALT

FACULTAT D’INFORMÀTICA DE BARCELONA

Reminder: Basic graph operations

Adjacency

$$\text{Adjacency}(n) = \bar{N}$$
$$n_i \in \bar{N} \iff \exists e_1 \mid \rho(e_1) = (n_i, n) \vee \rho(e_1) = (n, n_i)$$

Reachability

- Path Queries (RPQs)
  - *x*, *y* represent nodes
  - α is a regular expression over *Lab*

$$x \xrightarrow{\alpha} y$$

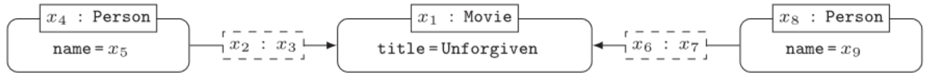
Regular expression operators

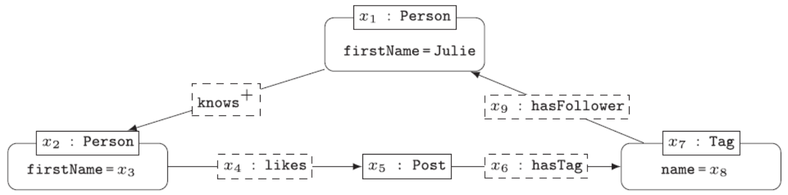
*	Kleene star
+	Kleene plus
.	Concatenation
|	Union
-	Inverse

… and combinations of them

Pattern Matching

- Basic Graph Patterns (BGPs)
  - Graphs where variables can appear in place of any constant
- Navigational Graph Patterns (NGPs)
  - BGPs where edge labels can be RPQs





Types of Graph Query Languages

Imperative languages (APIs) providing traversal operations, or implementations of graph metrics (algorithms)

- Depending on the metric/algorithm, they correspond to adjacency, path queries or graph patterns (or combinations of them)

Declarative languages to query the graph

- Typically, their syntax corresponds to that of Navigational Graph Patterns (NGPs)
  - Combination of pattern matching and reachability
- For pattern matching, every current graph database engine chooses a **fixed semantics**
  - Homomorphism
  - Isomorphism
    - Strict (no-repeated-anything)
    - No-repeated-node
    - No-repeated-edge

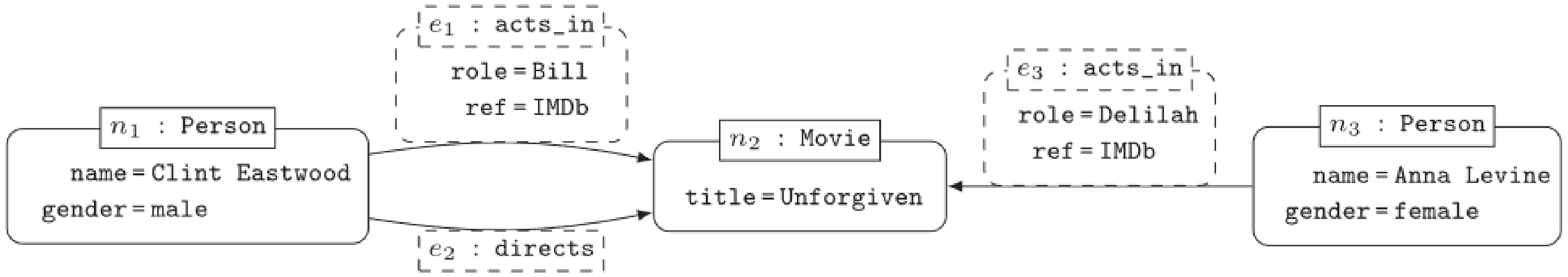
Activity

Objective: Identify the most efficient operation to solve a given query

Assume a graph containing relationships and nodes like the ones shown below (the graph may contain more nodes/eges, possibly with different labels)

Define (in the form of a NGP):

- An adjacency query
- A reachability query (a RPQ) that is not an adjacency query
- A Pattern Matching query (a BGP or a NGP) that is not an adjacency or a reachability query



Complex Graph Patterns (CGPs)

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

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

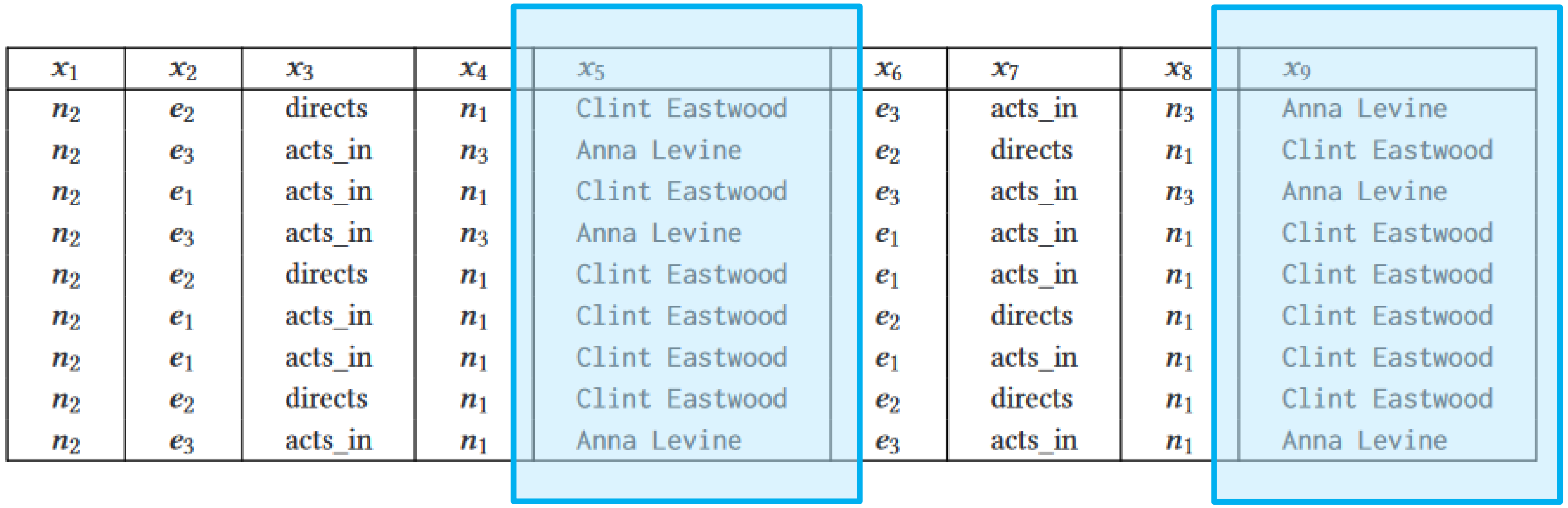
Graph query languages usually combine patterns with relational-like operators

- Projection
- Selection / Filter – considering inequalities on properties
- Union
- Difference
- Left-outer join / Optional
- Grouping, aggregations

It's like post-processing the results of pattern matching (always applied AFTER)

Complex Graph Patterns (CGPs)

Results of pattern matching:



x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9
n_2	e_2	directs	n_1	Clint Eastwood	e_3	acts_in	n_3	Anna Levine
n_2	e_3	acts_in	n_3	Anna Levine	e_2	directs	n_1	Clint Eastwood
n_2	e_1	acts_in	n_1	Clint Eastwood	e_3	acts_in	n_3	Anna Levine
n_2	e_3	acts_in	n_3	Anna Levine	e_1	acts_in	n_1	Clint Eastwood
n_2	e_2	directs	n_1	Clint Eastwood	e_1	acts_in	n_1	Clint Eastwood
n_2	e_1	acts_in	n_1	Clint Eastwood	e_2	directs	n_1	Clint Eastwood
n_2	e_1	acts_in	n_1	Clint Eastwood	e_1	acts_in	n_1	Clint Eastwood
n_2	e_2	directs	n_1	Clint Eastwood	e_2	directs	n_1	Clint Eastwood
n_2	e_3	acts_in	n_1	Anna Levine	e_3	acts_in	n_1	Anna Levine

Projection

An orthogonal choice of semantics appears now:

- **Set semantics:** no duplicate rows in the result
- **Bag semantics:** result may contain duplicates
- Examples: homomorphism-based set semantics (SPARQL), isomorphism-based bag semantics (Cypher), …

Cypher

NEO4J’S QUERY LANGUAGE

Neo4J Data Model

It is a graph!

- Use nodes to represent entities
- Use relationships to represent the 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.

Uniqueness constraints (~PK) can be asserted (only for node properties)

- `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)

Indexes can be created both in node and edge properties

You can use "PKs" and indexes in the Lab

Cypher

Created by Neo4j

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

High-level, declarative language

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

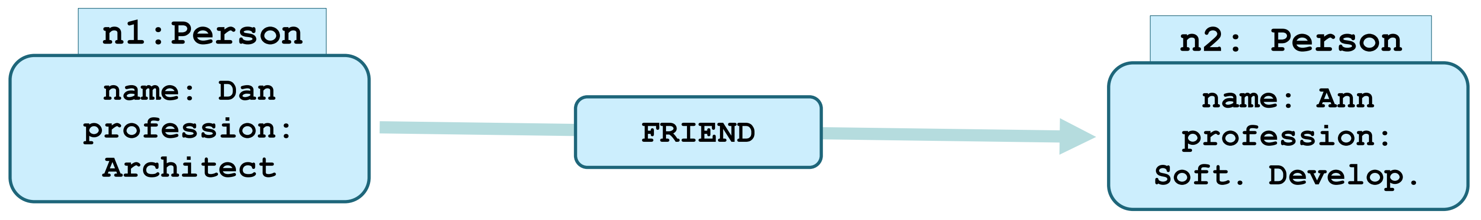
Its syntax allows to express navigational graph patterns

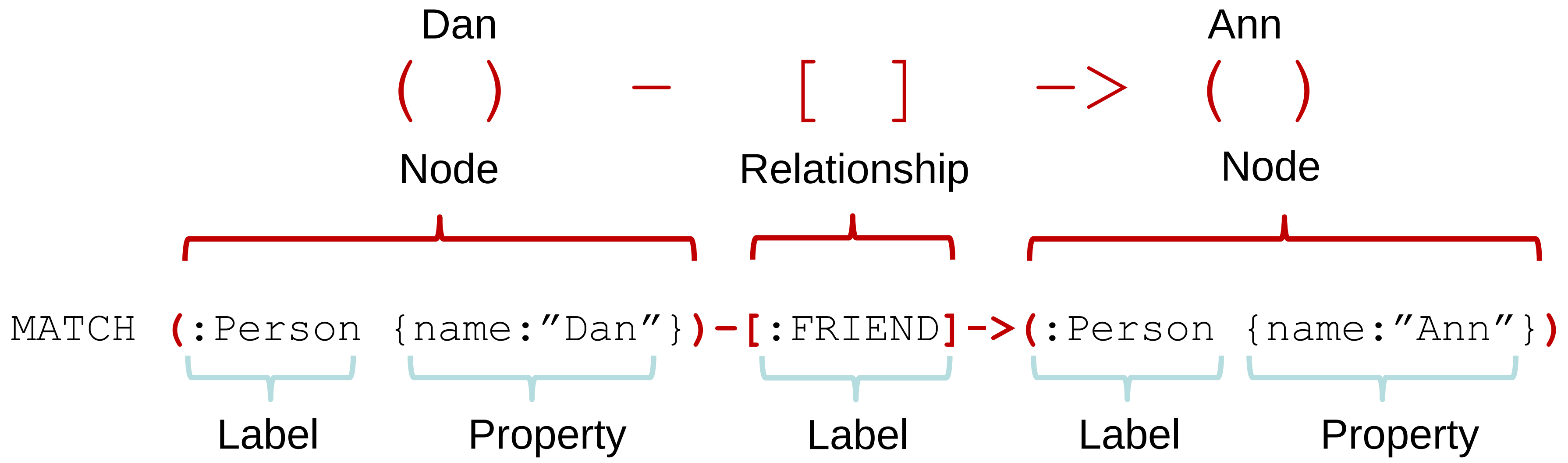
- Except concatenation (as an operator)

Cypher syntax to express paths does not directly correspond to the operations in regular expressions

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

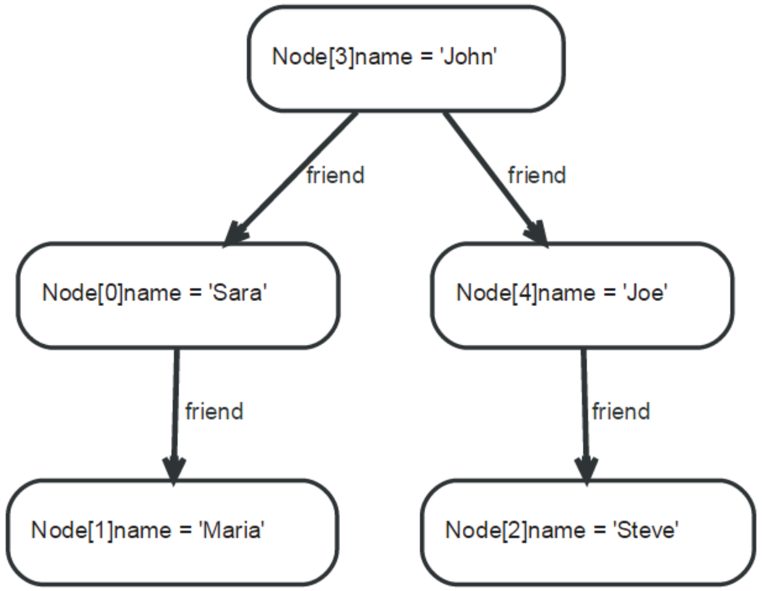
Cypher: Intuition





```
MATCH (:Person {name:"Dan"})-[:FRIEND]->(:Person {name:"Ann"})
```

Cypher: Example



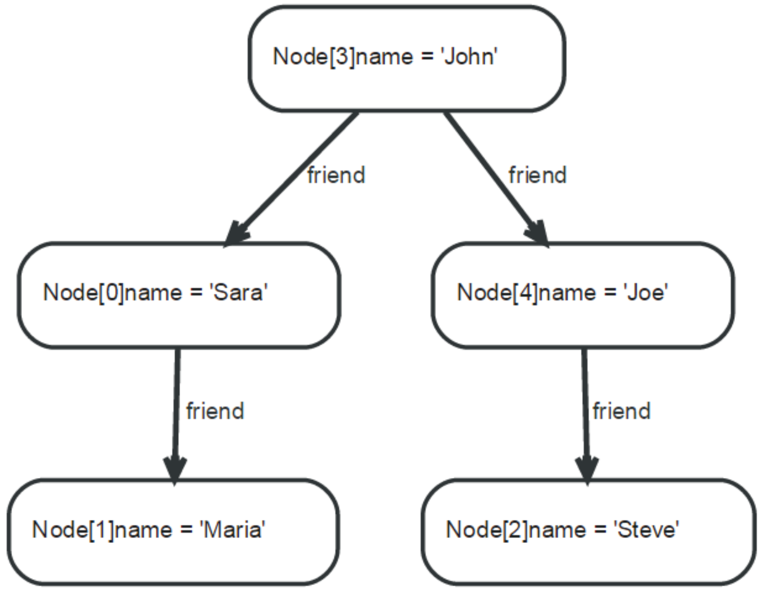
```
MATCH (john {name: 'John'})-[:friend]->()-[:friend]->(fof)
RETURN john, fof
```

Is this query a BGP, a NGP, or a CGP?

Activity

Objective: Basics on Cypher

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



1) Return all nodes

2) Return all edges

3) Return all friends of 'John'

4) Return all nodes adjacent to 'Sara'

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

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

Cypher Clauses

- DML:
  - MATCH: The graph **pattern** (BGP or NGP) 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 (MERGE only 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

You will need this for the Lab (for Evolving and for Recommender)

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

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

Cypher: Group By and Aggregates

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

Returns the number of nodes related to each node n

"n" is the grouping key,
"count" is the operation
on the resulting groups

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

Groups n by name and returns the number

```
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

This is how you can handle subqueries in Cypher

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

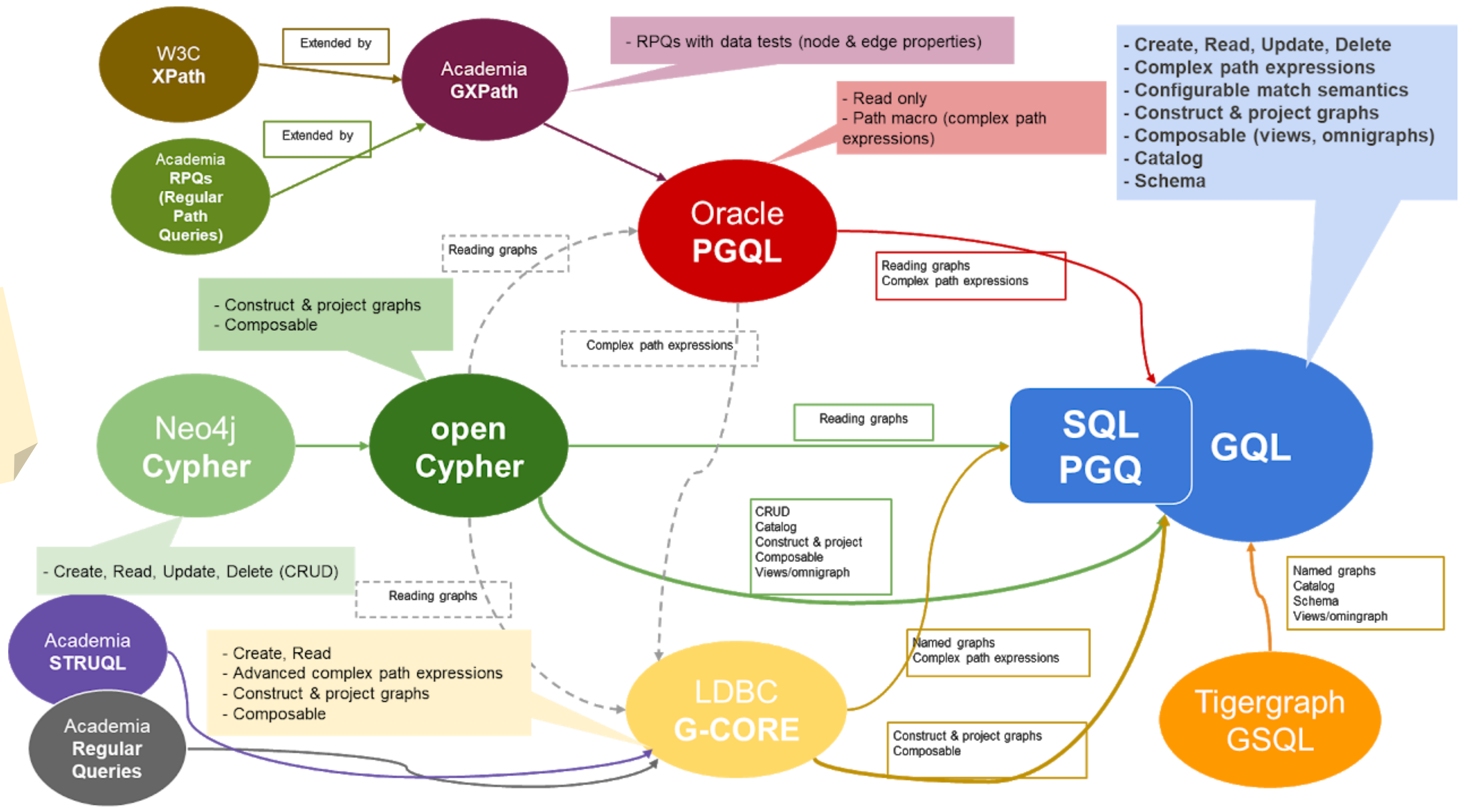
It allows the definition of aliases and processing to be passed between stages

```
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 (ISO Standard): https://www.gqlstandards.org/



OpenCypher was the de-facto standard for many years

Summary

Navigational Graph Patterns as keystone

- Pattern matching + Reachability

Complex Graph Patterns

- Extend 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) for property graphs, SPARQL for knowledge graphs
- GQL standard recently published

Thanks! Any Question?