Data Design and Modeling

Università della Svizzera italiana

Neo4J

Marco Brambilla

@marcobrambi

marco.brambilla@usi.ch

What is Neo4j

Developed by Neo Technologies

Most Popular Graph Database

Implemented in Java

Open Source

(www.neo4j.org)

Salient features of Neo4j

Neo4j is schema free – Data does not have to adhere to any convention

ACID – atomic, consistent, isolated and durable for logical units of work

Easy to get started and use

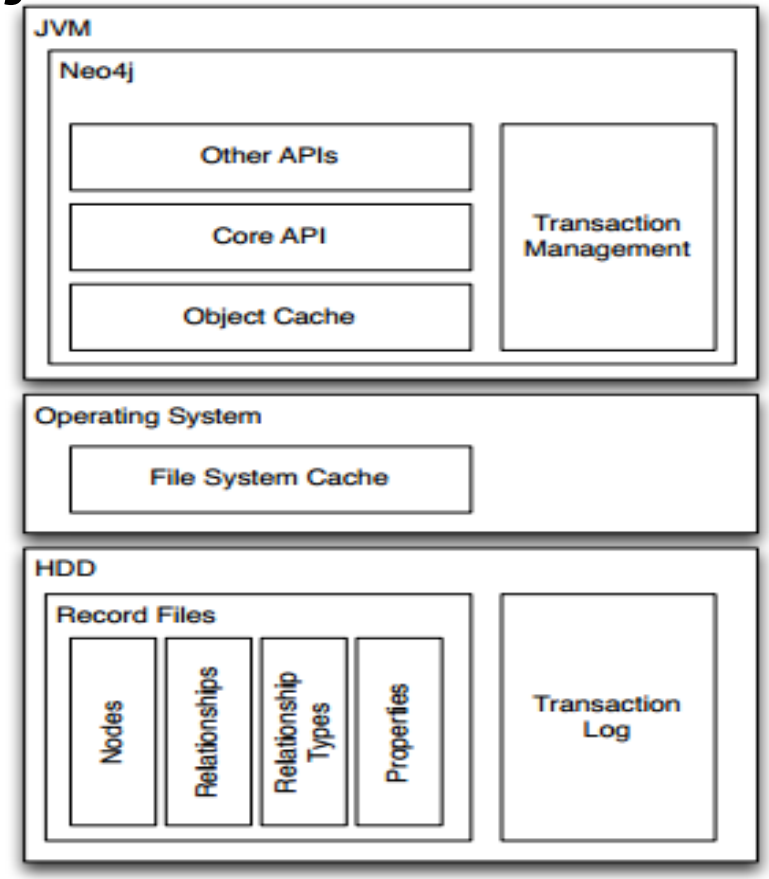
Well documented and large developer community

Support for wide variety of languages

Java, Python, Perl, Scala, Cypher, etc

(Mistry, Deep, 2013)

Neo4j Software Architecture



Purpose

Meant to be an operational DB, not specifically for analytics

- ACID
- Efficient on nodes
- Not so efficient in whole-graph analysis

Data Model

Nodes – with labels (type) and attributes

Edges

Indexes

Cypher

Query Language for Neo4j

Declarative language

Easy to formulate queries based on relationships

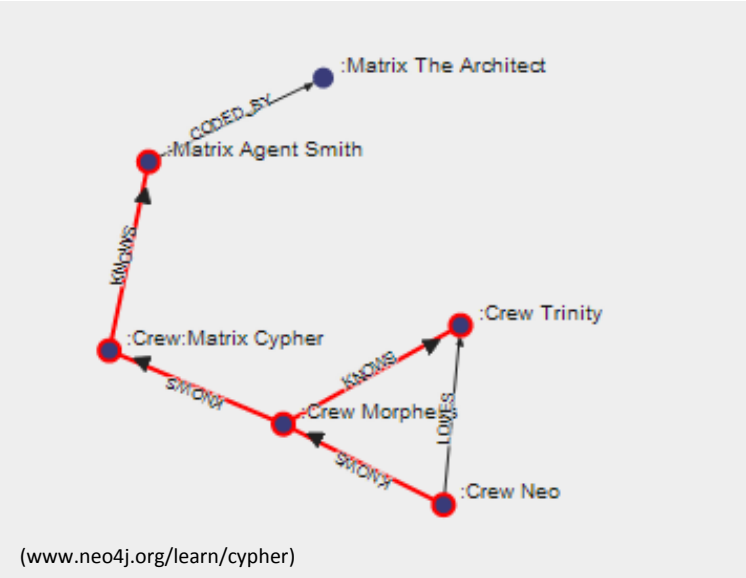
Many features stem from improving on pain points with SQL such as join tables

(Hunger, Michael 2013)

Cypher – data creation

```
CREATE (Neo:Crew { name:'Neo' })
```

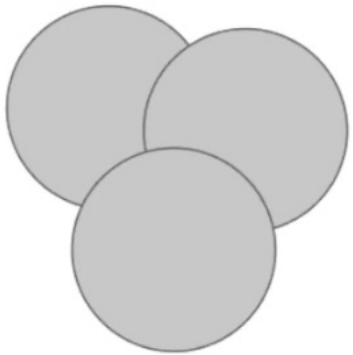
```
(Neo)-[:KNOWS]->(Morpheus)
```



(www.neo4j.org/learn/cypher)

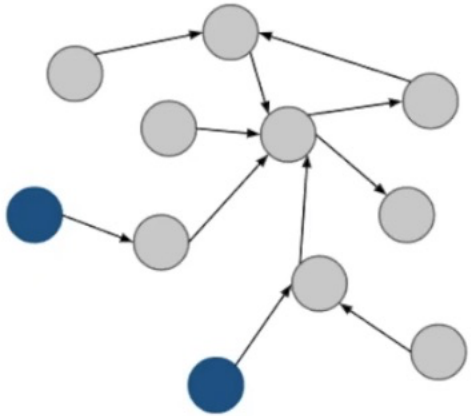
Cypher - indexes

CREATE INDEX ON :Customer(customerID);



Relational

Use index scans to look up rows in tables and join them with rows from other tables



Graph

Use indexes to find the starting points for a query.

Cypher - constraints

```
CREATE CONSTRAINT ON (c:Customer)
ASSERT c.customerID IS UNIQUE;
```

Cypher - Queries

START

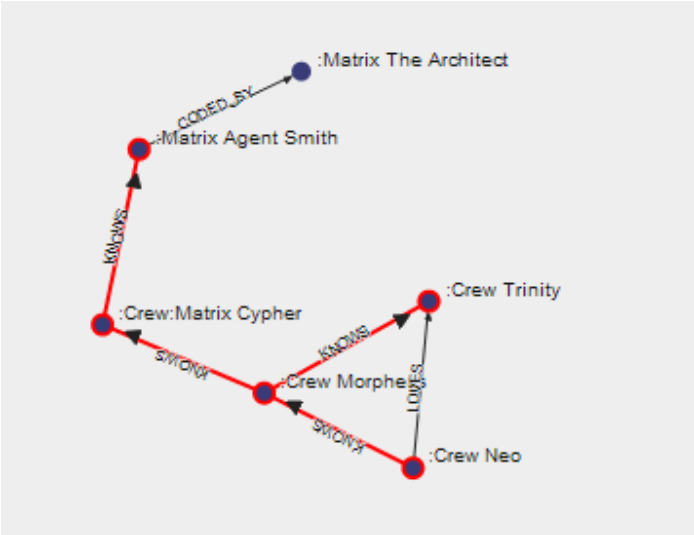
MATCH Pattern Matching

WHERE Expressions, Predicates

RETURN Output

Cypher

```
Query:
MATCH (n:Crew)-[r:KNOWS*]-m
WHERE n.name='Neo'
RETURN n AS Neo,r,m
```



Neo	r	m
{name:"Neo"}	[(0)-[0:KNOWS]->(1)]	(1:Crew {name:"Morpheus"})
{name:"Neo"}	[(0)-[0:KNOWS]->(1), (1)-[2:KNOWS]->(2)]	(2:Crew {name:"Trinity"})
{name:"Neo"}	[(0)-[0:KNOWS]->(1), (1)-[3:KNOWS]->(3)]	(3:Crew:Matrix {name:"Cypher"})
{name:"Neo"}	[(0)-[0:KNOWS]->(1), (1)-[3:KNOWS]->(3), (3)-[4:KNOWS]->(4)]	(4:Matrix {name:"Agent Smith"})

(www.neo4j.org/learn/cypher)

General Query Format

```
MATCH (user)-[:FRIEND]-(friend)
WITH user, count(friend) AS friends
ORDER BY friends DESC
SKIP 1     LIMIT 3
RETURN user
```

Aggregation can be used (count).

WITH separates query parts explicitly, to declare the variables for the next part.

SKIP skips results at the top and LIMIT limits the number of results.

Patterns

(n:Person)

Node with Person label.

(n:Person:Swedish)

Node with both Person and Swedish labels.

(n:Person {name: $value})

Node with the declared properties.

()-[r {name: $value}]-()

Matches relationships with the declared properties.

(n)-->(m)

Relationship from n to m.

(n)--(m)

Relationship in any direction between n and m.

(n:Person)-->(m)

Node n labeled Person with relationship to m.

Patterns

(m)<-[:KNOWS]-(n)

Relationship of type KNOWS from n to m.

(n)-[:KNOWS|:LOVES]->(m)

Relationship of type KNOWS or of type LOVES from n to m.

(n)-[r]->(m)

Bind the relationship to variable r.

(n)-[*1..5]->(m)

Variable length path from 1 to 5 rels. from n to m.

(n)-[*]->(m)

Variable length path of any number of rels. from n to m

(n)-[:KNOWS]->(m {property: $value})

A relationship of type KNOWS from a node n to a node m with the declared property.

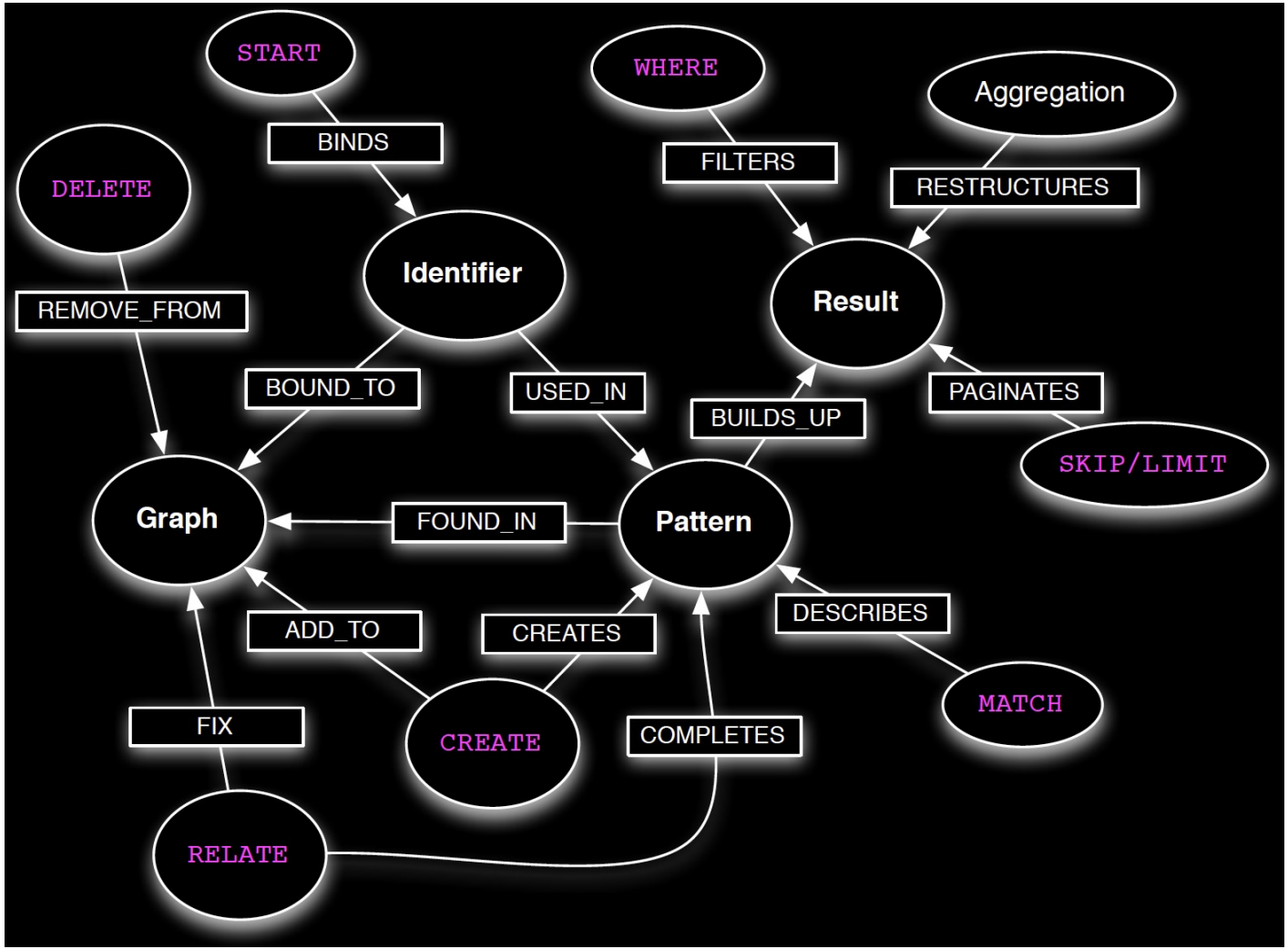
Paths

shortestPath((n1:Person)-[*..6]-(n2:Person))
Find a single shortest path.

allShortestPaths((n1:Person)-[*..6]->(n2:Person))
Find all shortest paths.

size((n)-->()-->())
Count the paths matching the pattern.

Core operators and impact



Hints

Use parameters instead of literals when possible. This allows Cypher to re-use your queries instead of having to parse and build new execution plans.

Always set an upper limit for your variable length patterns. It's easy to have a query touch all nodes in a graph by mistake.

Return **only the data you need**. Avoid returning whole nodes and relationships

Use **PROFILE / EXPLAIN** to analyze the performance of your queries.

References

https://neo4j.com/

https://neo4j.com/docs/cypher-refcard/3.2/