Querying Data

Roland Guijt
www.rmgsolutions.nl
@rolandguijt

pluralsight
hardcore dev and IT training

Agenda

Data Modelling

Other Language Elements

MATCH RETURN

Advanced Syntax

What is Cypher?

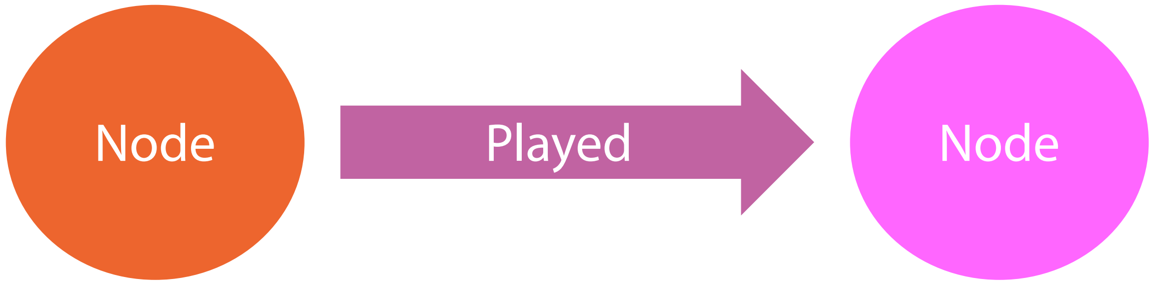
What Is Cypher?

- Graph query language
- Declarative
- Easy on the brain
- Pattern matching
- Clauses

Cypher Is About Pattern Matching

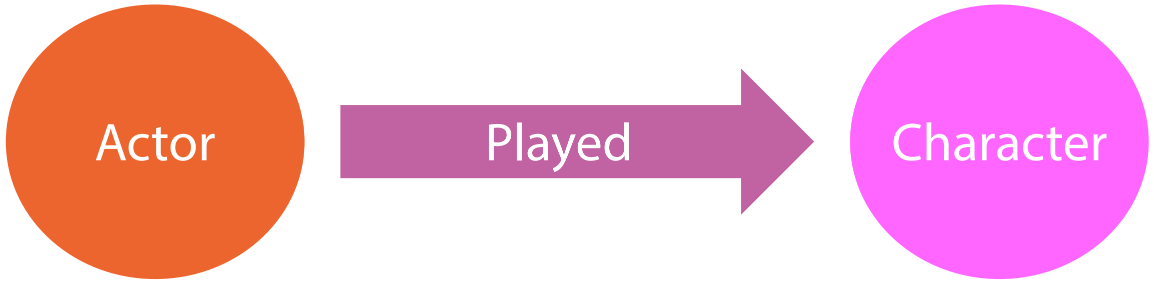
Recipe to make a query:

- Think of a whiteboard friendly pattern or structure you would like to retrieve
- Translate into ASCII art
- Surround by clauses



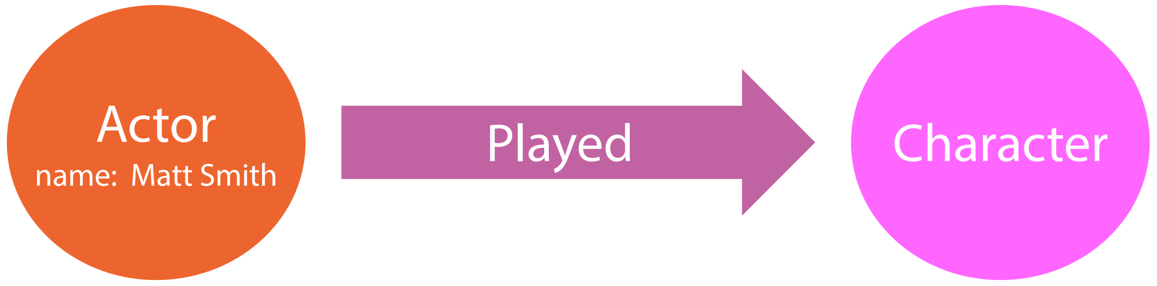
() –[:PLAYED]->()

Cypher Is About Pattern Matching



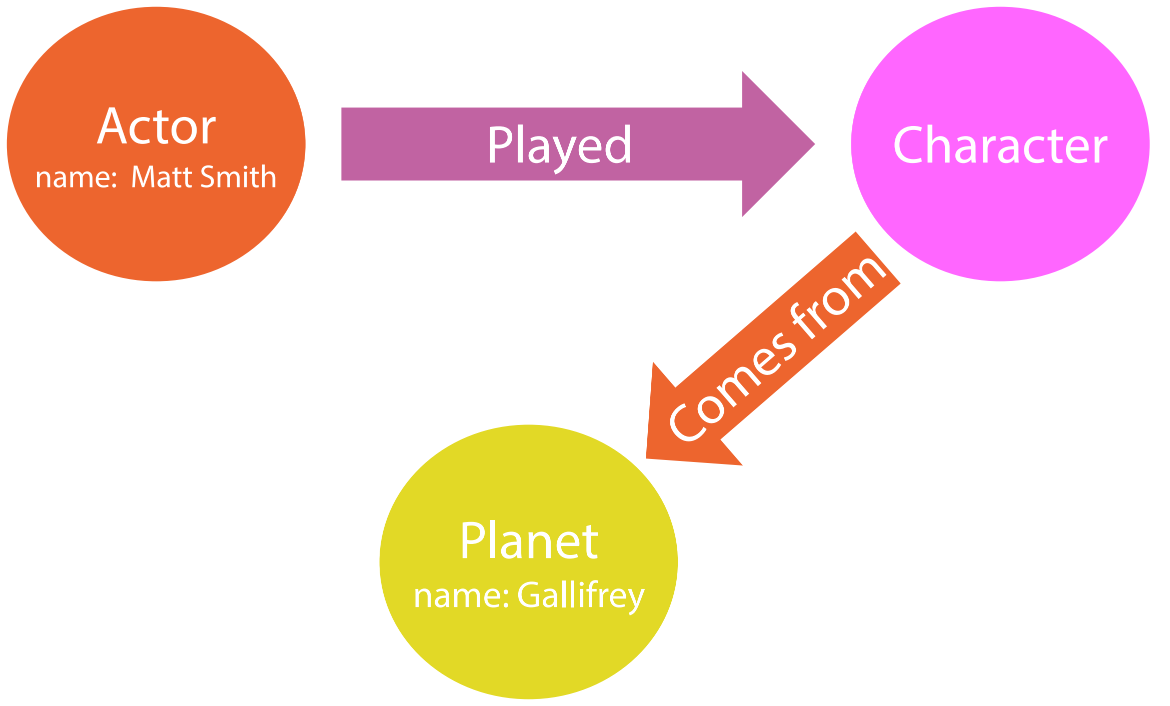
(:Actor) –[:PLAYED]->(:Character)

Cypher Is About Pattern Matching



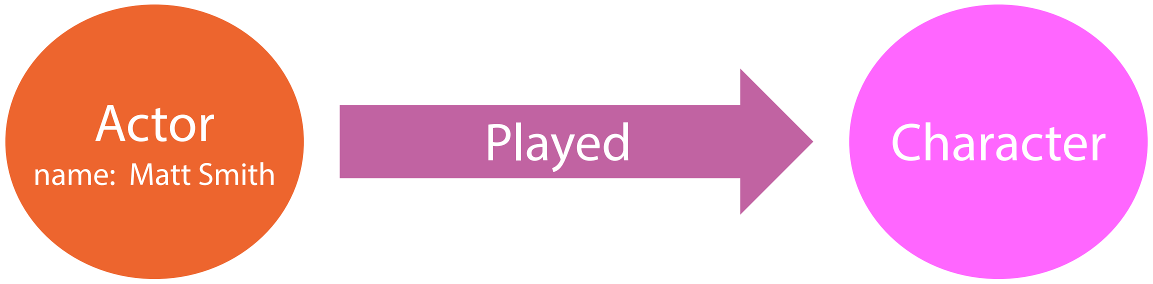
(:Actor{name:'Matt Smith'}) –[:PLAYED]->(:Character)

Cypher Is About Pattern Matching



(:Actor{name:'Matt Smith'}) –[:PLAYED]->
(:Character)-[:COMES_FROM]->(:Planet{name:'Gallifrey'})

The MATCH and RETURN Clauses



(:Actor{name:'Matt Smith'}) –[:PLAYED]->(:Character)

(:Actor{name:'Matt Smith'}) –[:PLAYED]->(c:Character)

MATCH

(:Actor{name:'Matt Smith'}) –[:PLAYED]->(c:Character)

RETURN c

Query Examples: 2 Loose Ends

Return the name properties of all nodes with the Label property and put them side by side with the name properties of all nodes that are on the other end of the regenerated_to relation

```
MATCH (actors:Actor)-[:REGENERATED_TO]-> (others)

RETURN actors.name, others.name;
```

Query Examples: More Complex

Collect all nodes with the Character label which have the enemy_of relation with the Doctor. Check if they have a comes_from relation with nodes with a Planet label. Return the name of the planets along with the number of occurances

```
MATCH (:Character{name:'Doctor'})<-[:ENEMY_OF]-(:Character)-
[:COMES_FROM]->(p:Planet)

RETURN p.name as Planet, count(p) AS Count;
```

Query Examples: More Complex

Give me all the episodes the character Amy Pond and the Actor Matt Smith were in. List the enemies of the Doctor that were in that episode beside it.

```
MATCH (:Actor{name:"Matt Smith"}) -[:APPEARED_IN]-> (ep:Episode)
<-[:APPEARED_IN]- (:Character{name:'Amy Pond'}),

(ep) <-[:APPEARED_IN]-(enemies:Character) <-[:ENEMY_OF]-
(:Character{name:'Doctor'})

RETURN ep AS Episode, collect(enemies.name) AS Enemies;
```

Where

- Filters result set

```
MATCH
(:Actor{name:'Matt Smith'}) –[:PLAYED]->(c:Character)
RETURN c

MATCH
(a:Actor) –[:PLAYED]->(c:Character)
WHERE a.name = 'Matt Smith'
RETURN c
```

Order By

- Orders result set
- Supports multiple properties
- Use DESC to reverse order

```
MATCH
(a:Actor) –[:PLAYED]->(c:Character)
WHERE a.name = ‘Matt Smith’
RETURN c
ORDER BY c.name
```

Skip and **Limit**

- Limits result set

```
MATCH
(:Actor{name:'Matt Smith'}) –[:PLAYED]->(c:Character)
RETURN c
LIMIT 10
SKIP 5
```

Union

- Glues result sets together
- Use UNION ALL to include duplicates

```
MATCH (a:Actor)
RETURN a.name
UNION
MATCH (c:Character)
RETURN c.name
```

With

- Manipulate result set for the rest of the query
- Can have ORDER BY clause

```
MATCH
(a:Actor)
WITH a.name AS name, count(a) AS count
ORDER BY name
WHERE count > 10
RETURN name
```

Start (legacy)

- Was used to access legacy indexes
- Provide a starting point for the pattern

Predicates

- Return true or false for a given input
- Input can be properties or patterns
- Mostly used in WHERE clause
- ALL, ANY, NONE, SINGLE, EXISTS

```
MATCH
(a:Actor)
WHERE EXISTS ((a)-[:PLAYED]->())
RETURN a.name
```

Scalar Functions

- Return a single value
- LENGTH, TYPE, ID, COALESCE, HEAD,
- LAST, TIMESTAMP, TOINT, TOFLOAT, TOSTRING

```
MATCH
p = (:Actor)-[:PLAYED]->(:Character)
RETURN LENGTH(p)
```

Collection Functions

- Return collections of ‘things’
- NODES, RELATIONSHIPS, LABELS
- EXTRACT, FILTER, TAIL
- RANGE, REDUCE

```
MATCH
p = (:Actor)-[:PLAYED]->(:Character)
RETURN NODES(p)
```

Mathematical Functions

- ABS
- ACOS
- ASIN
- ATAN
- COS
- COT
- DEGREES
- EXP
- FLOOR
- ROUND
- SQRT
- Etc.

String Functions

- STR
- REPLACE
- SUBSTRING
- LEFT
- RIGHT
- LTRIM
- RTRIM
- TRIM
- LOWER
- UPPER
- SPLIT

Advanced Syntax: Directionless Relationships

```
MATCH
(:Episode)-[:PREVIOUS]-(e:Episode)
RETURN e
```

Advanced Syntax: No Relationship Defined

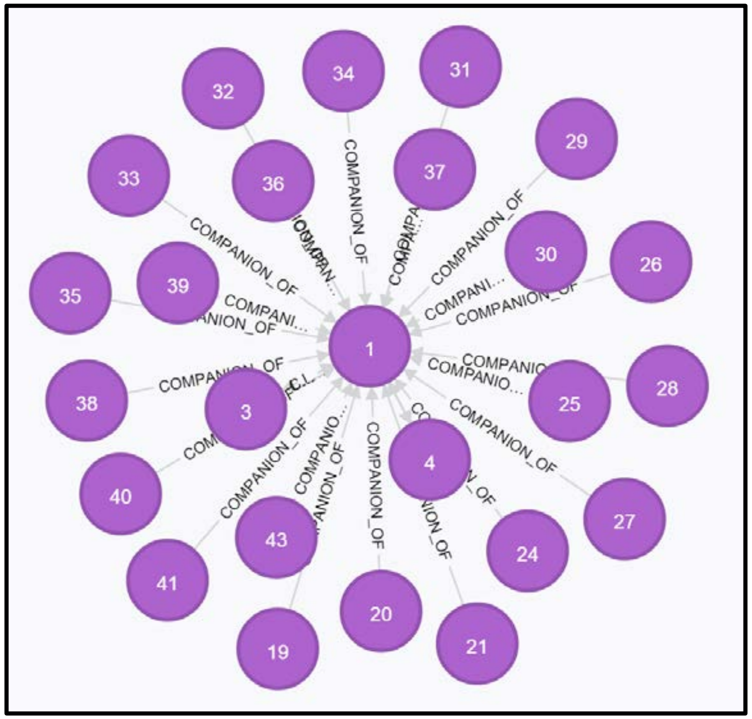
```
MATCH
(:Episode)-->(e:Episode)
RETURN e
```

Advanced Syntax: No Relationship Name

```
MATCH
(:Actor)-[]->()-[]->(p:Planet)
RETURN p
```

Advanced Syntax: Number of Hops

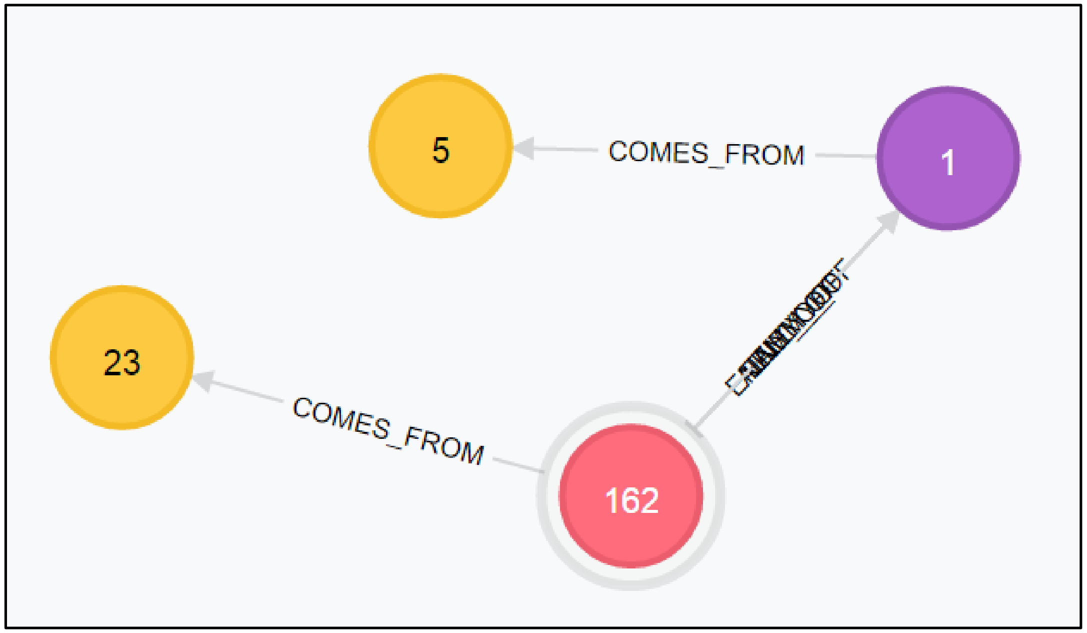
```
MATCH
(:Actor)-[*2]->(p:Planet)
RETURN p
```



```
MATCH
(c:Character)-
[:COMPANION_OF*1..2]-
(:Character)
RETURN c
```

Advanced Syntax: Shortest Path

```
MATCH (earth:Planet { name:"Earth" }),
(gallifrey:Planet { name:"Gallifrey" }),
p = shortestPath((earth)-[*..15]-(gallifrey))
RETURN p
```



Advanced Syntax: Optional MATCH

```
MATCH (a:Character)
OPTIONAL MATCH
(a)-[r:COMES_FROM]->()
RETURN r
```

Summary

- Cypher is a powerful, declarative query language for Neo4j.
- It uses patterns to query data.
- Cypher’s main clauses are MATCH and RETURN.
- There are more SQL-like clauses like WHERE.
- Many powerful functions to be used in query complement the language.
- Going beyond the basic syntax opens up even more powerful query possibilities.

What’s Next?

- Manipulating data