Cypher Query Language

- declarative graph query language
- pattern oriented
- allows to focus on your application domains instead of getting lost in technicalities
    - focuses on the clarity of expression *what* to retrieve from a graph and now *how* to retrieve it
    - very expressive
- relatively small but yet still very powerful
- easily read and understood by *developers, db professionnals* and *business stakeholders*

Basic Query Structure

Graph traversal / Data retrieval

MATCH clause

- how it really works is that you specify pattern that represent an example on how to traverse the graph
- indicates the pattern which should be used to traverse the graph
- placeholders may be present to capture and use different information within the query
    - use those variables throughout the query
- to find data for specific nodes and relationships in an existing dataset, we specify the property values explicitely:

```
MATCH (emil: Person {name:'Emil'})-[:KNOWS]->(ian:Person {name:'Ian'})
```

WHERE clause

may be bundled with a `WHERE` clause

```
MATCH (n1:NodeLabel1)-->(n2:NodeLabel2)
```

- adds constraint to a match pattern or to filter results that passes through the `MATCH` clause

RETURN clause

used to state what a query should return

Examples

```
MATCH (cust:Customer)-[:PURCHASED]->(:Order)-[o:ORDERS]->(p:Product),
      (p)-[:PART_OF]->(c:Category {categoryName:"Produce"})
RETURN DISTINCT cust.contactName as CustomerName, SUM(o.quantity) AS
TotalProductsPurchased
```

- *cust*, *o*, *p*, *c* are *placeholders* that captures the informations
- use of pattern matching to specify a graph traversal
- (node_placeholder:NodeLabel)-[:RELATIONSHIP_NAME]->(node_placeholder2:NodeLabel {propertyName: "property"})

```
MATCH (js:Person)-[:KNOWS]-()-[:KNOWS]-(surfer)
WHERE js.name = "Johan" AND surfer.hobby = "surfing"
RETURN DISTINCT surfer
```

- use of WHERE clause to enforce constraint or filter information

Graph manipulation

CREATE

- creates all the new part of a pattern : (*create a new :Bar node and a :FOO relationship that is associated with a node having id = 1*)

```
MATCH (n) WHERE id = 1 CREATE (n)-[:FOO]->(b:Bar)
```

MERGE

Will `MATCH`or `CREATE` (only if it doesn't exists)

Aggregation functions

cypher has various built-in functions like `count()` `sum()` `avg()` `min()`

Indexes / Constraints

- uniquely idenfifying object defined in Neo4j
- mainly used by `MERGE` queries to ensure nodes are uniquely created
- much more like SQL

```
CREATE CONSTRAINT on (u:User) ASSERT u.id IS UNIQUE;
CREATE INDEX on :User(name);
```

and then users are uniquely identified through *name*

Examples

```
MATCH (neo:Database {name:"Neo4j"})
MATCH (johan:Person {name:"Johan"})
CREATE (johan)-[:FRIEND]->(:Person:Expert {name:"Max"})-[:WORKED_WITH]->
(neo)
```

```
MATCH (you {name:"You"}), (expert)-[:WORKED_WITH]->(db:Database
{name:"Neo4j"}),
  p = shortestPath( (you)-[:FRIEND*..5]-(expert) )
RETURN p,db
```

SQL vs Cypher

```
SELECT DISTINCT co_actor.name
FROM person AS keanu
  JOIN acted_in AS acted_in1 ON acted_in1.person_id = keanu.id
  JOIN acted_in AS acted_in2 ON acted_in2.movie_id = acted_in1.movie_id
  JOIN person AS co_actor
    ON acted_in2.person_id = co_actor.id AND co_actor.id <> keanu.id
WHERE keanu.name = 'Keanu Reeves';
```

vs

```
MATCH (keanu:Person)-[:ACTED_IN]->(movie:Movie),
(coActor:Person)-[:ACTED_IN]->(movie)
WHERE keanu.name = 'Keanu Reeves'
RETURN DISTINCT coActor.name;
```

- notice how more expressive the cypher query is
- much more related on the application domain and not necessarily the technicalities involved in the SQL query

Common pitfalls

- Although very expressive, not sure a particular graph if fit for purpose
- Easy to make mistakes if misused

```
MATCH (:Person)-[:KNOWS]-()-[:KNOWS]->(:Person)
```

- the `()` specifies any specific nodes that associated with 2 `KNOWS`relation; but they can be anything if no label specified, not necessarily `:Person`
- binds so naturally that sometimes queries might hide logic errors
  - (one might assume that only a :Person can be associted with 2 :KNOWS relationship), however one could add a totally different node with 2 :KNOWS relationship and it would still be matched by this query.

Cypher	SQL	XQuery
MATCH (:Person)-[:KNOWS]-()-[:KNOWS]->(:Person)	SELECT ...	//node/path/pattern
WHERE cond = condval	WHERE cond = condval	//node/path/patern[@attr='attr_value']
CREATE ...	INSERT INTO ...	N/A
MATCH ... SET n = 1	UPDATE ... SET n = 1 WHERE ...	N/A

---

http://neo4j.com/docs/stable/cypher-introduction.html

https://www.airpair.com/neo4j/posts/getting-started-with-neo4j-and-cypher

http://neo4j.com/developer/cypher-query-language/#_about_cypher