Neo4j Cypher CRUD

Cypher basic syntax

- () Node
- {} Properties
- [] Relationships

- MATCH [some set of nodes and/or relationships]
  WHERE [some set of properties holds]
  RETURN [some set of results captured by the MATCH and WHERE clauses]

- MATCH (n) RETURN n;
  Returns every node in the database

Create

- CREATE (tt:Chocolate {name:“Trinitario Treasure", cocoa: 0.71, broad: “Ghana”, manufacturer: “Jacque Torres”, country: “USA”})
- CREATE (cem:Publication {name: “Chocolate Expert Monthly”})
- CREATE (tt)-[r:review {rating: 2, year: 2006}]->(cem)
- CREATE (tri:Bean {name:”Trinitario”})
- CREATE (tt)-[b:bean_type]->(tri)

Add more nodes

- CREATE (amedei:Company {name: “Amedei”, country: “Italy”})
- CREATE (tb:Chocolate {name:“Toscano Black", cocoa: 0.70})
- CREATE (md:Chocolate {name:“Madagascar", cocoa: 0.70})
- CREATE (blend:Bean {name:”Blend”})
- CREATE (amedei)-[p:produce]->(tb)
- CREATE (amedei)-[p:produce]->(md)
- CREATE (tb)-[b:bean_type]->(blend)

Schemaless Social

- CREATE (p:Person {name: "Alice"})
- MATCH (p:Person {name: "Alice"}),(c:Chocolate {name: “Trinitario Treasure"})
- CREATE (p)-[r:likes]->(c)

- CREATE (p: Person {name: "Tom"})
- MATCH (p:Person {name: "Tom"}),
- (pub:Publication {name: “Chocolate Expert Monthly"})
- CREATE (p)-[r:trusts]->(pub)

- CREATE (p:Person:Critic {name: "Patty"})
- MATCH (p1:Person {name: "Patty"}),(p2:Person {name: "Tom"})
- CREATE (p1)-[r:friends]->(p2)
- CREATE (:Person {name: “Patty”})-[r:friends]->(:Person {name: “Alice”})

Read

- All nodes stored in the database
- MATCH (n) RETURN n;

- Nodes with label “Person” and property “name” with value “Tom”
- MATCH (n:Person { name: “Tom" }) RETURN n;

- All matching nodes that are related, regardless of the direction (notice the lack of an arrow) and returning the nodes from both sides.
- MATCH (a)--(b) RETURN a, b;

- Same as before but returning the relationship
- MATCH (a)-[r]-(b) RETURN a, r, b;

- Nodes connected with relationship friends
- MATCH (a)-[:friends]->(b) RETURN a, b;

Optional Match

- Return the path (instead of individual nodes).
- MATCH p=(a)-[:friends]->(b) RETURN p;

- You can use multiple MATCH clauses in a query :
- MATCH (a:Person {name: ‘Tom'})
- MATCH (b:Person {name: ‘Hanks'})
- RETURN a, b;

- Optional match returns the match if it’s there, if not it’ll return `null`, so the query still works.
- MATCH (a:Person {name: ‘Tom})
- OPTIONAL MATCH (b:Person {name: ‘Hanks'})
- RETURN a, b;

- You can also use the optional flag to return potential relationships for a node. Like:
- MATCH (a:Person {name: ‘Tom'})
- OPTIONAL MATCH (a)-->(x)
- RETURN a, x;
- For all of the `Person` labeled nodes with the `name` of `Chris` both the nodes that do and don’t have relationships will be returned.

Where clause

- Similar to SQL
- MATCH (n:Chocolate) WHERE n.cocoa > .7 RETURN n;

- You can also combine WHERE with AND, OR, and NOT
- MATCH (n:Person)
- WHERE n.age > 18 AND (n.name = 'Chris' OR n.name = “Tom") AND (n)-[:RELATED {relation:"brother"}]-() RETURN n;

- MATCH (n:Chocolate)
- WHERE n.broad = ‘Ghana' OR n.broad IS NULL
- RETURN n
- ORDER BY n.name

- Regular expressions using `=~` followed by the pattern:
- MATCH (n)
- WHERE n.name =~ '(?i)^[a-d].*'
- RETURN n

Update

- MATCH (n { name: ‘Tom' })
- SET n.age = 35
- RETURN n

- MATCH (n { name: ‘Tom' })
- SET n += { username: ‘tomh'}

Delete

- CREATE (e: EphemeralNode {name: "short lived"})
- MATCH (c:Chocolate {name: “Trinitatio Treasure"}),
- (e:EphemeralNode {name: "short lived"})
- CREATE (c)-[r:short_lived_relationship]->(e)

- MATCH ()-[r:short_lived_relationship]-() DELETE r
- MATCH (e:EphemeralNode) DELETE e

- Delete entire graph
- MATCH (n) OPTIONAL MATCH (n)-[r]-() DELETE n, r

Indexes

- Can help speed up queries

- You can create an index on that type/property combination like this:
- CREATE INDEX ON :Chocolate(name);

- You can easily remove indexes at any time:
- DROP INDEX ON :Chocolate(name);

Constraints

- *Constraints* help sanitize data inputs by preventing writes that don't satisfy a specified criteria
- To ensure that every Chocholate node in the graph have a unique name, for example, you could create this constraint:
- CREATE CONSTRAINT ON (c:Chocolate) ASSERT c.name IS UNIQUE;

- To remove it include the entire constraint statement:
- DROP CONSTRAINT ON (c:Chocolate) ASSERT c.name IS UNIQUE;

- You cannot apply a constraint to a label that already has an index, and if you do create a constraint on a specific label/property pair, an index will be created automatically. So *usually* you'll only need to explicitly create a constraint *or* an index.

Limit and Skip

- MATCH (n)
- RETURN n
- ORDER BY n.name LIMIT 3

- MATCH (n)
- RETURN n
- ORDER BY n.name SKIP 1

- MATCH (n)
- RETURN n
- ORDER BY n.name
- SKIP 1 LIMIT 5

Other functions

- count(*), count(n), count(DISTINCT n)
- length(p)
- type (r)
- id(n), labels(n)
- nodes(p)
- collect – create array with return values
- timestamp()

Unwind

- UNWIND takes collections of nodes, or arrays of data and splits them into individual rows.
- Data must be aliased for the query to work.

- UNWIND ['Chris', 'Kyle', 'Andy', 'Dave', “Tom”] AS x
- RETURN x

- This would return all of the names in individual rows, rather than as a collection as they were passed

For today…

- After following the movies example (excluding the clean up)
  - https://neo4j.com/developer/cypher/guide-cypher-basics/
  - :play movie-graph
- Write a query that shows the movies that Tom Hanks has acted in and the directors that have directed them
- Upload the graph result to ICON
- Clean up