Database Management Systems

Course 236363

Tutorial 8: NoSQL Neo4j

Slides by Roni Licher and Shoval Lagziel
Outline

• Neo4j and Cypher
• CREATE
• QUERY
• SET
• WITH
• Questions
Graph Database, Think Different!

- Neo: age 29, name: Neo, knows Morpheus, loves Trinity
- Trinity: age 12 years, name: Trinity, knows Neo, loves Neo
- Morpheus: name: Morpheus, occupation: Total badass, rank: Captain, knows Neo, knows Cypher, knows The Architect, discloses public
- Cypher: last name: Reagan, name: Cypher, knows Morpheus, discloses secret
- Agent Smith: language: C++, name: Agent Smith, version 1.0b, coded by The Architect
Graph Database, Think Different!

- Nodes
  - Have labels
  - Properties (key:value)

- Edges – directed
  - Have type
  - Properties (key:value)
Neo4j

- Graph Database
  - Implemented in Java and Scala
- Open Source
- Started in 2003
- The most popular graph database
We found Neo4j to be literally thousands of times faster than our prior MySQL solution, with queries that require 10-100 times less code. Today, Neo4j provides eBay with functionality that was previously impossible.

Volker Pacher, Senior Developer, eBay
Neo4j and Cypher

Neo4j
Graph database

Cypher
Query language
Cypher - Create

• Create a node with a property:
  – CREATE (n { name: 'Andres' })

• Create a labeled node:
  – CREATE (n : Person)

• Create a labeled node with properties:
  – CREATE (n : Person
    { name: 'Andres',
      title: 'Developer' })

• To return the created labeled nodes:
  – RETURN n
• Create nodes with relationship:

```cypher
CREATE
  (p:Person {name: "Israel"})-[r:Says]->
  (m:Message {name: "Hello World!"})
RETURN p, m, r
```

• It is also possible to add relationships between existing nodes.
MATCH [Nodes and relationships]
WHERE [Boolean filter statement]
RETURN [DISTINCT] [statements [AS alias]]
ORDER BY [Properties] [ASC\DESC]
SKIP [Number] LIMIT [Number]
Query all nodes with label Person with the name Israel:

MATCH (a : Person)
WHERE a.name = 'Israel'
RETURN a
• Query all edges of type SAYS connecting between Person and Message:

MATCH (a:Person)-[r:SAY]->(b:Message)
RETURN a,r,b
For nodes:
(a)
(a:Nlabel)
(a { prop:'value' } )
(a:Nlabel { prop:'value' } )
Use MATCH

• For edges:
  (a)--(b)
  (a)-->(b)
  (a)<--(b)
  (a)-->(
  (a)-[r]->(b)
  (a)-[:Rtype]->(b)
  (a)-[:R1|:R2]->(b)

• For more than two nodes:
  (a)-->(b)<--(c), (a)-->(b)-->(c)
Use MATCH

• For edges by distance:
  – (a)-[:Rtype*2]-(b) – 2 hops of type Rtype.
  – (a)-[:Rtype* ]-(b) – any number of hops of type Rtype.
  – (a)-[:Rtype*2..10]-(b) – 2-10 hops of Rtype.
  – (a)-[:Rtype* ..10]-(b) – 1-10 hops of Rtype.
  – (a)-[:Rtype*2.. ]-(b) – at least 2 hops of Rtype

• For pathways:
  \[ p = (a)-[\ast]\rightarrow(b) \]
MATCH (a:Person),(b:Person)
WHERE a.name = 'Node A' AND
b.name = 'Node B'
WITH a,b
CREATE (a)-[r:RELTYPE]->(b)
RETURN r
Cypher Operators

- Arithmetic operations:
  +, -, *, /, %, ^ (power)
- Compare:
  =, <>, <, >, >=, <=, =~ (Regex), IS NULL, IS NOT NULL
Cypher Operators

• Logical:
  AND, OR, XOR, NOT

• Strings:
  STARTS WITH, ENDS WITH, CONTAINS
  Concatenate +

• For collections:
  concatenate with +,
  IN to check is an element exists in a collection.
WHERE others.name IN ['Andres', 'Peter']
WHERE user.age IN range (18,30)
WHERE n.name =~ 'Tob.*'
WHERE n.name =~ '(?i)ANDR.*' - (case insensitive)
WHERE (n)-->()
WHERE NOT (n)-->()
WHERE exists(a.name)
WHERE b.name? = 'Bob'
(Returns all nodes where name = 'Bob' plus all nodes without a name property)
Collections operations

- MATCH (user)
  RETURN count(user)

- MATCH (user)
  RETURN count(DISTINCT user.name)

- MATCH (user)
  RETURN collect(user.name)
  (Collection from the values, ignores NULL)

- MATCH (user)
  RETURN avg(user.age)
  (Average numerical values. Similar functions are sum, min, max.)
Functions

- **On paths:**
  - MATCH shortestPath( (a)-[*]-(b) )
  - MATCH allShortestPath( (a)-[*]-(b) )
  - Length(path) – The path length or 0 if not exists.
  - relationships(p) - Returns all relationships in a path.
  - nodes(p) – nodes of the path p

- **On collections:**
  - WHERE ANY (x IN a.array WHERE x = "MAMAN")
    – at least one
  - WHERE ALL (x IN nodes(p) WHERE x.age > 30)
    – all elements
  - WHERE SINGLE (x IN nodes(p) WHERE var.eyes = "blue") – Only one
• **Change or add** properties:
  
  MATCH (n { name: 'Andres' })
  SET n.position = 'Developer', n.surname = 'Taylor'

• Copy all properties from another node:
  (Remove all the properties of the receiving element)
  
  MATCH (at { name: 'Andres' }), (pn { name: 'Peter' })
  SET at = pn
• Set labels to an existing node:
  MATCH (n { name: 'Emil' })
  SET n :Swedish:Israeli
  RETURN n

  (add two labels ‘Swedish’ and ‘Israeli’)
• Manipulate the result sequence before it is passed on to the following query parts.

• Usage of WITH:
  – Limit the number of entries that are then passed on to other MATCH clauses.
  – Introduce aggregates which can then be used in predicates in WHERE.
MATCH (david { name: "David" })-->(otherPerson)-->()

WITH otherPerson, count(*) AS foaf

WHERE foaf > 1

RETURN otherPerson

• What will be returned?

• Persons connected to David with more than one outgoing edge.
Question

- dname
- dcity
- frequents
- drinker
- pub
- pname
- pcity
- likes
- serves
- beer
- bname
- btype
In which cities there is a served beer Yossi likes?

MATCH (:drinker { dname: "Yossi" })-[:likes]->(:beer)<-[:serves]-(p :pub)
RETURN p.pcity
Who are the drinkers that all the pubs serve a beer they like?

MATCH (p : pub)
WITH collect(p) as Pubs
MATCH (d : drinker)
WHERE ALL (p in Pubs WHERE (p)-[:serves]->(:beer)<-[:likes]-(d) )
RETURN d
Studies: Connects between Student and Course, contains semester and grade properties.
Teaches: Connects between Lecturer to Course, contains semester and classroom properties.
a. Write Cypher query which returns:
   Names of Students who study all the courses.

MATCH (c:Course)
WITH collect(c) AS courses
MATCH (s:Student)
WHERE ALL (x in courses WHERE (s)-[:Studies]->(x))
RETURN s.name
MATCH (s:Student)-[:Studies*2..4]-(:Student{Name:"Roy"})
RETURN DISTINCT s.name
a. We define a distance function between two different students as follows:
   1. Students A and B are in distance 1 if they have learned a common course.
   2. Students A and B are in distance \( n > 1 \) if \( n \) is the smallest number such that there is a Student C, and A is at \( n-1 \) distance from C and C is 1 from B.
   3. If no such \( n \) exists we will define the distance to be 0.

Write a Cypher query that returns:
The distance between two students whose ID is 12345 and 67890

MATCH p=shortestPath((s1:Student {ID:'12345'})-[:Studies*]-s2:Student {ID:'67890'}))
RETURN length(p)/2
b. Names of all lecturers who taught at least 3 subjects. (You can assume there are no duplicates in the graph)

MATCH (l:Lecturer)-[:Teaches]->(c:Course)
WITH l, count(c) as numcourses
WHERE numcourses >= 3
RETURN l.name
Learn more...

Check Neo4j online version:

http://console.neo4j.org/

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

<table>
<thead>
<tr>
<th>Neo</th>
<th>r</th>
<th>m</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0:Crew {name:&quot;Neo&quot;})</td>
<td>[(0)-[0:KNOWS]-&gt;(1)]</td>
<td>(1:Crew {name:&quot;Morpheus&quot;})</td>
</tr>
<tr>
<td>(0:Crew {name:&quot;Neo&quot;})</td>
<td>[(0)-[0:KNOWS]-&gt;(1), (1)-[3:KNOWS]-&gt;(3)]</td>
<td>(3:Crew:Matrix {name:&quot;Cypher&quot;})</td>
</tr>
<tr>
<td>(0:Crew {name:&quot;Neo&quot;})</td>
<td>[(0)-[0:KNOWS]-&gt;(1)]</td>
<td>(1)-[3:KNOWS]-&gt;(3)</td>
</tr>
<tr>
<td>(0:Crew {name:&quot;Neo&quot;})</td>
<td>[(0)-[0:KNOWS]-&gt;(1), (1)-[2:KNOWS]-&gt;(2)]</td>
<td>(2:Crew {name:&quot;Trinity&quot;})</td>
</tr>
</tbody>
</table>

Query took 28 ms and returned 4 rows.

You can modify and query this graph by entering statements in the input field at the bottom. For some syntax help hit the Help button. If you want to share your graph, just do it with Share.
Learn more...

Download Neo4j for free:

http://neo4j.com/download/
Learn more...

Read the Neo4j manual:
http://neo4j.com/docs/stable/

Cypher tutorials:
http://neo4j.com/developer/cypher-query-language/

More Neo4j developers tutorials:
http://neo4j.com/developer/get-started/