Managing Data on the World Wide Web

Neo4j - Introduction
Overview

1. Introduction

2. Cypher
While there is a value in individual data items, in some cases the relationships between them carry the most interesting information:
- information about people vs. information about the relationships between them
- Web documents vs. the web graph (where web pages are connected by links)

Although graphs can be represented as relational tables, graph analytics queries may result in poor performance.
Graph Databases

• A graph database is a database management system with Create, Read, Update, and Delete (CRUD) methods that expose a graph data model

• Neo4j is an open-source graph database implemented in Java
  – The most popular graph database (see this link)
  – ACID-compliant
  – Supports read-committed isolation
  – Enables explicit locking of nodes and edges
• Everything is stored in form of either an edge, a node or an attribute
  – An edge is a synonym for undirected relationship
  – All relationships in a Neo4j graph are directed
  – Attribute is a synonym for property
• Each node and edge can have any number of attributes
Nodes can be labelled
- A label marks a node as a member of a named subset
- Therefore, can be used to narrow searches
- Nodes may have zero, one, or more labels

Each edge must have exactly one relationship type
• Relationships are equally well traversed in either direction
  – This means that there is no need to add duplicate relationships in the opposite direction (with regard to traversal or performance)
Here, friendship is asymmetric.

To find Alice’s friends we have first to perform an index lookup, at cost $O(\log n)$.

If we wanted to find out who is friends with Alice, we would have to perform multiple index lookups – one for each node that is potentially friends with Alice.
In Neo4j, each node maintains direct references to its adjacent nodes
– Query times are independent of the total size of the graph

To find Alice’s friends using a graph, we simply follow her outgoing FRIEND relationships, at $O(1)$ cost each

To find who is friends with Alice, we follow all of Alice’s incoming FRIEND relationships to their source, at $O(1)$ cost each
Overview

1. Introduction

2. Cypher
A declarative graph query language for Neo4j

- We express what to retrieve from a graph, not on how to retrieve it

Has similarities to SQL

- E.g., WHERE, ORDER BY

Queries are built up using various clauses:

- MATCH: The graph pattern to match
- RETURN: What to return

Cypher can be used for both querying and updating your graph
• The simplest form: ()
  – Represents an anonymous node

• A node with an identifier: (m)
  – By adding an identifier we can refer to that node

• More example:
  (m:Movie)
  (m:Movie {title: "The Matrix"})
  (m:Movie {title: "The Matrix", released: 1997})
The arrowhead can be dropped if the direction is not important: -[role]-
  – Note that in Neo4j relationships always have a direction

The arrowhead can be at either end (but not both): <-[role:ACTED_IN]>
Patterns

• By combining the syntax for nodes and relationships we form **patterns**
• Patterns are used to match desired graph structures

```
(keanu:Person:Actor {name: "Keanu Reeves"} )
 -[role:ACTED_IN {roles: ["Neo"]}] ->
  (matrix:Movie {title: "The Matrix"} )
```

```
(:Person) -[:LIVES_IN]-> (:City) -[:PART_OF]-> (:Country)
```
CREATE (a:Person { name: "Tom Hanks", born: 1956 })
  -[r:ACTED_IN { roles: ["Forrest"]}]->
    (m:Movie { title: "Forrest Gump", released: 1994 })
CREATE (d:Person { name: "Robert Zemeckis", born: 1951 })
  -[:DIRECTED]-(m)
RETURN a, d, r, m
A MATCH statement will search for the patterns we specify and return one row per successful pattern match.

1. MATCH (m:Movie) RETURN m
2. MATCH (m:Movie { title: "The Matrix" }) RETURN m
3. MATCH (p:Person { name:"Tom Hanks" })-[r:ACTED_IN]-(m:Movie) RETURN m.title, r.roles
4. MATCH (actor)-[r:ACTED_IN*2]-(co_actor) RETURN r

The second example is equivalent to:

MATCH (m:Movie) WHERE m.title = "The Matrix" RETURN m
• Query processing:
  – Parse and validate the query
  – Generate the execution plan
  – **Locate the initial node(s)**
  – Select and traverse relationships
  – Change and/or return values

• Before traversal can begin, Neo4j must know one or more starting nodes

• For a graph query to run fast, we don’t need indexes
  – We only need them to find our starting points

• Adding index to actors by name:

  CREATE INDEX ON :Actor(name)
References

- [http://neo4j.com/docs/stable](http://neo4j.com/docs/stable)
- [Neo4j Javadocs](http://neo4j.com/docs/stable)
- [Neo4j Bookshelf](http://neo4j.com/docs/stable)
  - Recommended: Graph Databases 2nd Edition (Free Book)