Cypher query optimisation - Utilising known properties of nodes

1 post by 1 author

Michael Anslow Apr 10

Other recipients: Michael...@gmail.com

Setup:

Neo4j and Cypher version 2.2.0.
I'm querying Neo4j as an in-memory instance in Eclipse created `TestGraphDatabaseFactory().newImpermanentDatabase();`.
I'm using this approach as it seems faster than the embedded version and I assume it has the same functionality.
My graph database is randomly generated programmatically with varying numbers of nodes.

Background:

I generate cypher queries automatically. These queries are used to try and identify a single 'target' node. I can limit the possible matches of the queries by using known 'node' properties. I only use a 'name' property in this case. If there is a known name for a node, I can use it to find the node id and use this in the start clause. As well as known names, I also know (for some nodes) if there are names known not to belong to a node. I specify this in the where clause.

The sorts of queries that I am running look like this...

START

nvari = node(5)

MATCH

(target:C5)-[:IN_LOCATION]->(nvara:LOCATION),
(nvara:LOCATION)-[:CONNECTED]->(nvarb:LOCATION),
(nvara:LOCATION)-[:CONNECTED]->(nvarc:LOCATION),
(nvard:LOCATION)-[:CONNECTED]->(nvarc:LOCATION),
(nvard:LOCATION)-[:CONNECTED]->(nvare:LOCATION),
(nvare:LOCATION)-[:CONNECTED]->(nvarf:LOCATION),
(nvarg:LOCATION)-[:CONNECTED]->(nvarf:LOCATION),
(nvarg:LOCATION)-[:CONNECTED]->(nvarh:LOCATION),
(nvari:C4)-[:IN_LOCATION]->(nvarg:LOCATION),
(nvarj:C2)-[:IN_LOCATION]->(nvarg:LOCATION),
(nvare:LOCATION)-[:CONNECTED]->(nvark:LOCATION),
(nvarm:C3)-[:IN_LOCATION]->(nvarg:LOCATION),

WHERE

NOT(nvarj.Name IN ['nf']) AND NOT(nvarm.Name IN ['nb','nj'])

RETURN DISTINCT target

Another way to think about this (if it helps), is that this is an isomorphism testing problem where we have some information about how nodes in a query and target graph correspond to each other based on restrictions on labels.

Question:

With regards to optimisation:

1. Would it help to include relation variables in the match clause? I took them out because the node variables are sufficient to distinguish between relationships but this might slow it down?
2. Should I restructure the match clause to have match/where couples including the where clauses from my previous example first? My expectation is that they can limit possible bindings early on. For example...

```
START

nvari = node(5)

MATCH

(nvarj:C2)-[:IN_LOCATION]->(nvarg:LOCATION)

WHERE NOT(nvarj.Name IN ['nf'])

MATCH

(nvarm:C3)-[:IN_LOCATION]->(nvarg:LOCATION)

WHERE NOT(nvarm.Name IN ['nb','nj'])

MATCH

(target:C5)-[:IN_LOCATION]->(nvara:LOCATION),
(nvara:LOCATION)-[:CONNECTED]->(nvarb:LOCATION),
(nvara:LOCATION)-[:CONNECTED]->(nvarc:LOCATION),
(nvard:LOCATION)-[:CONNECTED]->(nvarc:LOCATION),
(nvard:LOCATION)-[:CONNECTED]->(nvare:LOCATION),
(nvare:LOCATION)-[:CONNECTED]->(nvarf:LOCATION),
(nvarg:LOCATION)-[:CONNECTED]->(nvarf:LOCATION),
(nvarg:LOCATION)-[:CONNECTED]->(nvarh:LOCATION),
(nvare:LOCATION)-[:CONNECTED]->(nvark:LOCATION)

RETURN DISTINCT target
```

On the side:

3. (Less important but still an interest) If I make each relationship in a match clause an optional match except for relationships containing the target node, would cypher essentially be finding a maximum common sub-graph between the query and the graph data base with the constraint that the MCS contains the target node?

Thanks a lot in advance! I hope I have made my requirements clear but I appreciate that this is not a typical use-case for Neo4j.