What are best practices for using optional matches in Neo4j 2.2 Cypher queries?

3 posts by 1 author  

bi...@levelstory.com Apr 8 

After some more testing, I don't think I understand how to use OPTIONAL MATCH statements correctly.

I have a query to find a bunch of contacts that a user has access to within a team.

```
  PROFILE
  MATCH (:user { id:"foobar" })-[:CONTACT]->(uc:contact)-[:CONTACT]->(ur:role)
  WHERE HAS(ur.teamid)
  WITH uc, ur

  MATCH (c:contact {teamid: uc.teamid})-[:CONTACT]->(cr:role)
  WHERE (ur.name IN ["Owners", "Admins"]) OR
        (ur.name = "Employees" AND NOT cr.name = "Clients") OR
        (cr:projectrole AND (uc)-[:CONTACT*1..3]->(:projectrole)<-
[:CONTACT*0..2]-(cr)) OR
        (c.id = uc.id)
  WITH DISTINCT c, {accessedby: uc.id, accessedrole:ur.name} AS r

  MATCH (c)-[:CONTACT]->(cr:role {teamid: c.teamid})
  WITH c, {
    accessedrole: r.accessedrole, accessedby: r.accessedby,
    role: cr.name
    } AS r

  RETURN c, r LIMIT 10
```

After running it a few times to make sure that the statement is cached, it takes 587 db hits and 118 ms to execute. I then try and add a simple OPTIONAL MATCH query to return all of the phone numbers associated with the contacts. There are 670 nodes in the (c) collection and I'm trying to just say 'hey, if the contact has some phone numbers associated with them, get me those phone numbers'. The part in bold below is what was added.

```
  PROFILE
  MATCH (:user { id:"foobar" })-[:CONTACT]->(uc:contact)-[:CONTACT]->(ur:role)
  WHERE HAS(ur.teamid)
  WITH uc, ur

  MATCH (c:contact {teamid: uc.teamid})-[:CONTACT]->(cr:role)
  WHERE (ur.name IN ["Owners", "Admins"]) OR
        (ur.name = "Employees" AND NOT cr.name = "Clients") OR
        (cr:projectrole AND (uc)-[:CONTACT*1..3]->(:projectrole)<-
[:CONTACT*0..2]-(cr)) OR
        (c.id = uc.id)
  WITH DISTINCT c, {accessedby: uc.id, accessedrole:ur.name} AS r

  MATCH (c)-[:CONTACT]->(cr:role {teamid: c.teamid})
  WITH c, {
    accessedrole: r.accessedrole, accessedby: r.accessedby,
    role: cr.name
    } AS r

  OPTIONAL MATCH (c)-[:PHONE]->(p:phone)
  WITH c, {
```

```
    accessedrole: r.accessedrole, accessedby: r.accessedby, role: r.role,
    phones: EXTRACT(x IN COLLECT(p) | { id: x.id, object: x.object,number:
x.number, description: x.description })
    } AS r

  RETURN c, r LIMIT 10
```

This new query now requires 48,000 db hits and 947ms to complete. Somehow finding a collection of 670 contacts took 100ms but seeing if those same contacts had phone numbers took 10x longer and took 100x the db hits. The problem that I am running into is that I also need to look for optional addresses and tags which altogether is taking over a minute to complete.

Can anybody help me re-write this query to be in line with best practices? What is the right strategy for matching a node along with optional child nodes?

Thanks!

bi...@levelstory.com Apr 8

Images are not displayed
Display images in this post - Always display images from bi...@levelstory.com- Always display images in Neo4j

Here is another interesting case as I try things out.

This query takes 475ms and 66k db hits to execute:

```
  PROFILE
  MATCH (:user { id:"foobar" })-[:CONTACT]->(uc:contact)-[:CONTACT]->(ur:role)
  WHERE HAS(ur.teamid)
  WITH uc, ur

  MATCH (c:contact {teamid: uc.teamid})-[:CONTACT]->(cr:role)
  WHERE (ur.name IN ["Owners", "Admins"]) OR
        (ur.name = "Employees" AND NOT cr.name = "Clients") OR
        (cr:projectrole AND (uc)-[:CONTACT*1..3]->(:projectrole)<-
[:CONTACT*0..2]-(cr)) OR
        (c.id = uc.id)
  WITH DISTINCT c, {accessedby: uc.id, accessedrole:ur.name} AS r

  MATCH (c)-[:CONTACT]->(cr:role {teamid: c.teamid})
  WITH c, {
    accessedrole: r.accessedrole, accessedby: r.accessedby,
    role: cr.name
    } AS r

  OPTIONAL MATCH (c)-->(p:phone)
  WITH c, {
    accessedrole: r.accessedrole, accessedby: r.accessedby, role: r.role,
    phones: EXTRACT(x IN COLLECT(p) | { id: x.id, object: x.object })
    } AS r

  OPTIONAL MATCH (c)-->(a:address)
  WITH c, {
    accessedrole: r.accessedrole, accessedby: r.accessedby, role: r.role,
    phones: r.phones,
    addresses: EXTRACT(x IN COLLECT(a) | { id: x.id, object: x.object, street:
x.street, locality: x.locality,
```

```
        region: x.region, postcode: x.postcode, country: x.country, description:
x.description, neighborhood: x.neighborhood,
        latlnglocation: x.latlnglocation
      })
    } AS r

  OPTIONAL MATCH (c)-->(t:tag)
  WITH c, {
    accessedrole: r.accessedrole, accessedby: r.accessedby, role: r.role,
    phones: r.phones, addresses: r.addresses,
    tags: EXTRACT(x IN COLLECT(t) | {id: x.id, object: x.object, name: x.name})
    } AS r

  RETURN c, r LIMIT 10
```

However, add in the bold clause below and the query now take 8200ms (20x) and 6 million db hits (100x), even though not a single contact matches against the new optional query.

```
  PROFILE
  MATCH (:user { id:"foobar" })-[:CONTACT]->(uc:contact)-[:CONTACT]->(ur:role)
  WHERE HAS(ur.teamid)
  WITH uc, ur

  MATCH (c:contact {teamid: uc.teamid})-[:CONTACT]->(cr:role)
  WHERE (ur.name IN ["Owners", "Admins"]) OR
        (ur.name = "Employees" AND NOT cr.name = "Clients") OR
        (cr:projectrole AND (uc)-[:CONTACT*1..3]->(:projectrole)<-
[:CONTACT*0..2]-(cr)) OR
        (c.id = uc.id)
  WITH DISTINCT c, {accessedby: uc.id, accessedrole:ur.name} AS r

  MATCH (c)-[:CONTACT]->(cr:role {teamid: c.teamid})
  WITH c, {
    accessedrole: r.accessedrole, accessedby: r.accessedby,
    role: cr.name
    } AS r

  OPTIONAL MATCH (c)-->(i:invite)
  WITH c, {
    accessedrole: r.accessedrole, accessedby: r.accessedby, role: r.role,
    isinvited: NOT i IS NULL
    } AS r

  OPTIONAL MATCH (c)-->(p:phone)
  WITH c, {
    accessedrole: r.accessedrole, accessedby: r.accessedby, role: r.role,
    isinvited: r.isinvited,
    phones: EXTRACT(x IN COLLECT(p) | { id: x.id, object: x.object })
    } AS r

  OPTIONAL MATCH (c)-->(a:address)
  WITH c, {
    accessedrole: r.accessedrole, accessedby: r.accessedby, role: r.role,
    isinvited: r.isinvited, phones: r.phones,
    addresses: EXTRACT(x IN COLLECT(a) | { id: x.id, object: x.object, street:
x.street, locality: x.locality,
        region: x.region, postcode: x.postcode, country: x.country, description:
x.description, neighborhood: x.neighborhood,
        latlnglocation: x.latlnglocation
      })
    } AS r
```

```
OPTIONAL MATCH (c)-->(t:tag)
WITH c, {
  accessedrole: r.accessedrole, accessedby: r.accessedby, role: r.role,
  isinvited: r.isinvited, phones: r.phones, addresses: r.addresses,
  tags: EXTRACT(x IN COLLECT(t) | {id: x.id, object: x.object, name: x.name})
  } AS r

RETURN c, r LIMIT 10
```

You also end up with some pretty interesting profile output:

---

bi...@levelstory.com Apr 9

I uploaded the full plan details for the second query to http://i.imgur.com/TUmd2AF.png.
