Neo4j 2.2 Cypher - Cost planner not using indexes for computed values, index hints not working

New issue

Open **bunkat** opened this issue Apr 14, 2015 · 8 comments

bunkat commented Apr 14, 2015

The COST planner is unable to use indexes for basic MATCH statements, instead choosing to do expensive NodeByLabelScan operations. This seems to be a problem with using indexes on computed values (cr.teamid) vs with parameters ({p0}) or static strings ("foobar"). I also noticed that the COST planner uses a NodeIndexSeek operation instead of a SchemaIndex operation (for example, when finding the user in the below queries). It is unclear what the difference between the two is.

The RULE planner uses the SchemaIndex correctly in both instances.

Query

```
PROFILE
PLANNER RULE
MATCH (u:user {id: "foobar"})
WITH u

MATCH (u)-[:CONTACT]->(c:contact)-[:CONTACT]->(cr:role)
WHERE HAS(cr.teamid)
WITH c, cr

MATCH (p:project)
WHERE p.teamid = cr.teamid
RETURN p
```

Result

Labels

cypher

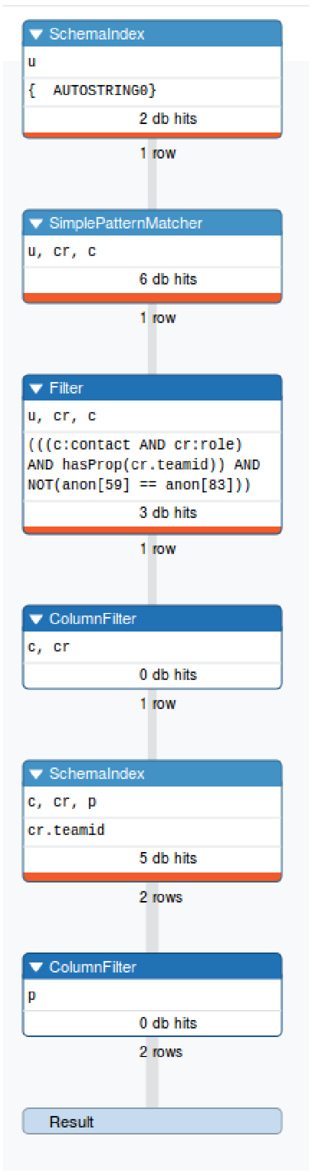
Milestone

No milestone

Assignee

No one assigned

3 participants



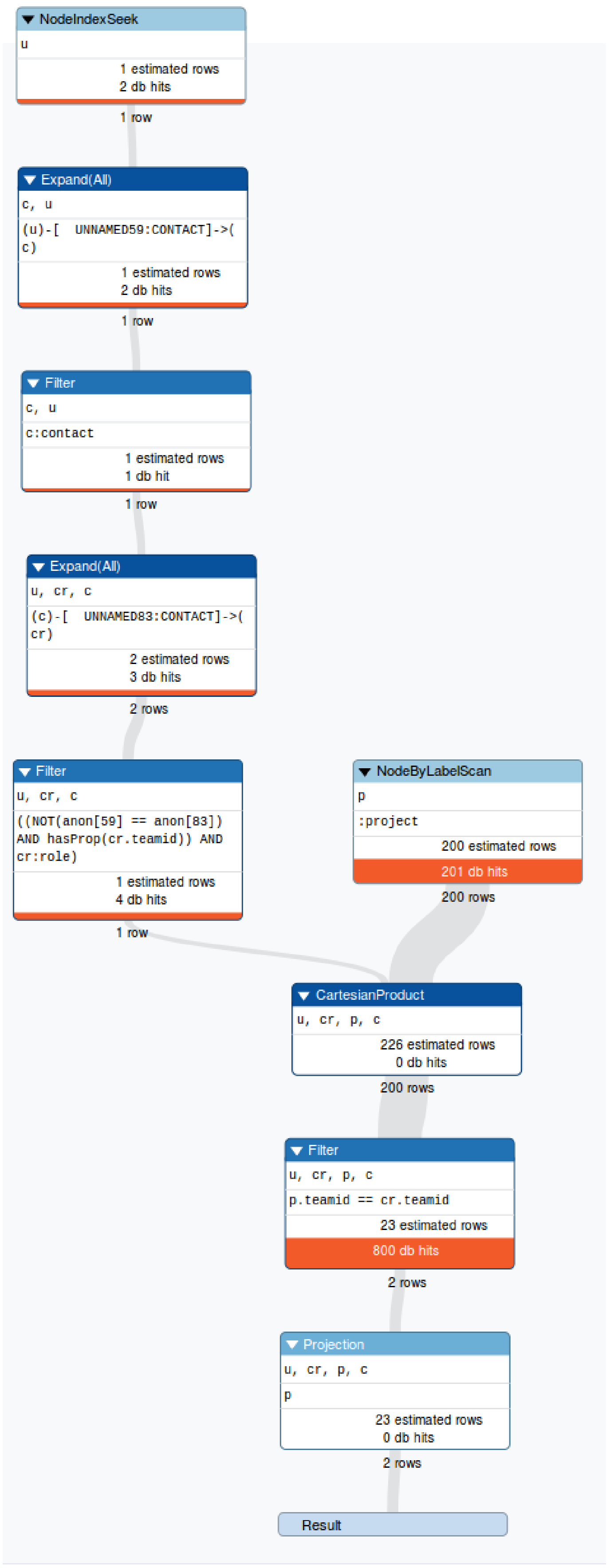
Query

```
PROFILE
PLANNER COST
MATCH (u:user {id: "foobar"})
WITH u

MATCH (u)-[:CONTACT]->(c:contact)-[:CONTACT]->(cr:role)
WHERE HAS(cr.teamid)
WITH c, cr

MATCH (p:project)
WHERE p.teamid = cr.teamid
RETURN p
```

Result


NodeIndexSeek
u
1 estimated rows
2 db hits
1 row
Expand(All)
c, u
(u)-[UNNAMED59:CONTACT]->(c)
1 estimated rows
2 db hits
1 row
Filter
c, u
c:contact
1 estimated rows
1 db hit
1 row
Expand(All)
u, cr, c
(c)-[UNNAMED83:CONTACT]->(cr)
2 estimated rows
3 db hits
2 rows
Filter
u, cr, c
((NOT(anon[59] == anon[83]) AND hasProp(cr.teamid)) AND cr:role)
1 estimated rows
4 db hits
1 row
NodeByLabelScan
p
:project
200 estimated rows
201 db hits
200 rows
CartesianProduct
u, cr, p, c
226 estimated rows
0 db hits
200 rows
Filter
u, cr, p, c
p.teamid == cr.teamid
23 estimated rows
800 db hits
2 rows
Projection
u, cr, p, c
p
23 estimated rows
0 db hits
2 rows
Result

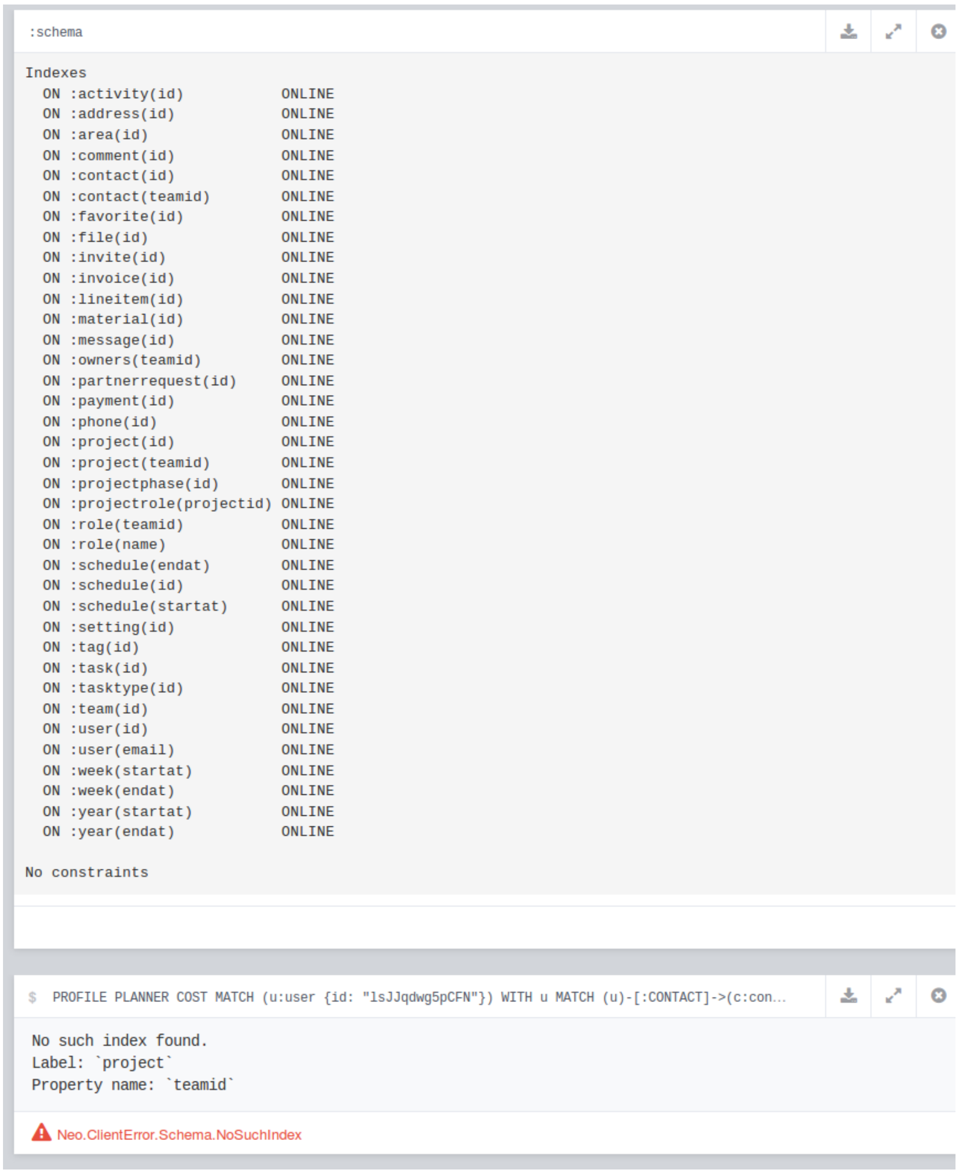
Query

```
PROFILE
PLANNER COST
MATCH (u:user {id: "foobar"})
WITH u

MATCH (u)-[:CONTACT]->(c:contact)-[:CONTACT]->(cr:role)
WHERE HAS(cr.teamid)
WITH c, cr

MATCH (p:project)
USING INDEX p:project(teamid)
WHERE p.teamid IN [cr.teamid]
RETURN p
```

Result



```
:schema

Indexes
  ON :activity(id)            ONLINE
  ON :address(id)             ONLINE
  ON :area(id)                ONLINE
  ON :comment(id)             ONLINE
  ON :contact(id)             ONLINE
  ON :contact(teamid)         ONLINE
  ON :favorite(id)            ONLINE
  ON :file(id)                ONLINE
  ON :invite(id)              ONLINE
  ON :invoice(id)             ONLINE
  ON :lineitem(id)            ONLINE
  ON :material(id)            ONLINE
  ON :message(id)             ONLINE
  ON :owners(teamid)          ONLINE
  ON :partnerrequest(id)      ONLINE
  ON :payment(id)             ONLINE
  ON :phone(id)               ONLINE
  ON :project(id)             ONLINE
  ON :project(teamid)         ONLINE
  ON :projectphase(id)        ONLINE
  ON :projectrole(projectid)  ONLINE
  ON :role(teamid)            ONLINE
  ON :role(name)              ONLINE
  ON :schedule(endat)         ONLINE
  ON :schedule(id)            ONLINE
  ON :schedule(startat)       ONLINE
  ON :setting(id)             ONLINE
  ON :tag(id)                 ONLINE
  ON :task(id)                ONLINE
  ON :tasktype(id)            ONLINE
  ON :team(id)                ONLINE
  ON :user(id)                ONLINE
  ON :user(email)             ONLINE
  ON :week(startat)           ONLINE
  ON :week(endat)             ONLINE
  ON :year(startat)           ONLINE
  ON :year(endat)             ONLINE

No constraints
```

```
$ PROFILE PLANNER COST MATCH (u:user {id: "lsJJqdwg5pCFN"}) WITH u MATCH (u)-[:CONTACT]->(c:con…

No such index found.
Label: `project`
Property name: `teamid`

Neo.ClientError.Schema.NoSuchIndex
```

craigtaverner commented Apr 15, 2015 Collaborator

The short answer is that you are right that the cost planner is not optimised for this case where the index lookup would be based on a dynamic or computed value. Another way of wording it is that it is not optimised for a 'foreign key join'. In this case you could have used a relationship to connect the (cr:role) and (p:project) nodes. The connection is by the teamid, and you would perhaps use a (cr)-[:team]-(p) relationship. If the value of the teamid is important, you could put a property on this relationship or make an intermediate (t:team) node.

Getting back to your query though, it should be possible to teach the cost model to consider using an index for even this case. I would think of that as a performance optimization. However, I'm curious to know what the performance comparison of the RULE and COST queries was? Was RULE much faster than COST? Even though COST did not use the schema index, it is possible that the NodeByLabelScan would have been fast enough anyway if the number of projects is small.

One thing that does look like a bug here is the error message for the index hint. It seems to claim that no such index exists, which is not the case. So we seem to have a bug and a feature request:

- bug - wrong error message on index hint
- feature request - performance optimization to use schema index on dynamic property (Apply/NodeIndexSeek instead of CartesianProduct/NodeByLabelScan)

bunkat commented Apr 15, 2015

For the queries I posted:

```
PROFILE
MATCH (u:user {id: "foobar"})
WITH u

MATCH (u)-[:CONTACT]->(c:contact)-[:CONTACT]->(cr:role)
WHERE HAS(cr.teamid)
WITH c, cr

MATCH (p:project)
WHERE p.teamid = cr.teamid
RETURN p

Cypher version: CYPHER 2.2, planner: RULE. 224 total db hits in 87 ms (10 run avg).
Cypher version: CYPHER 2.2, planner: COST. 1098 total db hits in 117 ms (10 run avg).
```

The COST planner takes 4.9x db hits and 1.34x as long to execute. This was just a simple demonstrative query example with a test database of 150 projects. Other queries take 4x to 10x longer to execute when using the COST planner and all of them will just continue to get slower over time due to the NodeByLabelScan operations.

While I could add more and more relationships to try and overcome these limitations it becomes very hard to manage. Especially since trying to avoid this type of foreign key join would mean that every child object would need to have an additional 10+ relationships so that I could always quickly get to the associated team or project (since there are role hierarchies 5 deep for both teams and projects that would all need these new connections). Plus, none of this is needed for the RULE planner.

In general it would be helpful to have more information on when the COST planner is expected to be the better option. I found the following quote in the blog post but it makes no mention on how the indices for the COST planner need to be used or other drawbacks:

> So, for which types of queries does Ronja bring about the best improvements?
> We have conducted extensive benchmarking tests, and, empirically, queries
> containing multiple nodes that can be reached via indices and/or pattern predicates
> show the most improvement.

It's unfortunate, but dropping in Neo4j 2.2 with default behavior has introduced a lot of new performance bottlenecks that made the database unfit for production use in my case. At this point, I just always force the RULE planner and see a lot better and more consistent performance.

jexp commented Apr 15, 2015 Collaborator

Hey Bill, sorry you ran into this, it's a bug which will be fixed.

Currently computed expressions are not taken into account for index lookups in the cost planner.

But you can force it to use an index, I'd be interested in the comparison between cost and rule then:

```
PROFILE
MATCH (u:user {id: "foobar"})
WITH u

MATCH (u)-[:CONTACT]->(c:contact)-[:CONTACT]->(cr:role)
WHERE HAS(cr.teamid)
WITH c, cr

MATCH (p:project)
using index p:project(teamid)
WHERE p.teamid = cr.teamid
RETURN p
```

bunkat commented Apr 15, 2015

That was the other bug that I hit. You can't use index hints with the COST planner.

Query as suggested fails to compile

```
PROFILE
MATCH (u:user {id: "foobar"})
WITH u

MATCH (u)-[:CONTACT]->(c:contact)-[:CONTACT]->(cr:role)
WHERE HAS(cr.teamid)
WITH c, cr

MATCH (p:project)
using index p:project(teamid)
WHERE p.teamid = cr.teamid
RETURN p
```

> Cannot use index hint in this context. Index hints require using a simple equality comparison or IN condition in WHERE (either directly or as part of a top-level AND). Note that the label and property comparison must be specified on a non-optional node (line 9, column 3 (offset: 164))
> " using index p:project(teamid)"

Switching to an IN expression, query fails to find index

```
PROFILE
MATCH (u:user {id: "foobar"})
WITH u

MATCH (u)-[:CONTACT]->(c:contact)-[:CONTACT]->(cr:role)
WHERE HAS(cr.teamid)
WITH c, cr

MATCH (p:project)
using index p:project(teamid)
WHERE p.teamid IN [cr.teamid]
RETURN p
```

> No such index found.
> Label: `project`
> Property name: `teamid`

```
Indexes
  ....
  ON :project(teamid)        ONLINE
```

Using the RULE planner instead, indexes work as expected

```
  PROFILE
  PLANNER RULE
  MATCH (u:user {id: "foobar"})
  WITH u

  MATCH (u)-[:CONTACT]->(c:contact)-[:CONTACT]->(cr:role)
  WHERE HAS(cr.teamid)
  WITH c, cr

  MATCH (p:project)
  using index p:project(teamid)
  WHERE p.teamid IN [cr.teamid]
  RETURN p
```

> Cypher version: CYPHER 2.2, planner: RULE. 224 total db hits in 87 ms.

By the way, this is actually a breaking change since this type of query will attempt to use the COST planner by default and then fail on `using index` throwing an error. Suddenly queries that worked fine in 2.1.8 will be broken when the database is upgraded to 2.2.1.

jexp commented Apr 15, 2015 Collaborator

Sorry for that, really frustrating.

```
PROFILE
  MATCH (u:user {id: "foobar"})
  WITH u

  MATCH (u)-[:CONTACT]->(c:contact)-[:CONTACT]->(cr:role)
  WHERE HAS(cr.teamid)
  WITH c, cr, cr.teamid as teamid

  MATCH (p:project)
  using index p:project(teamid)
  WHERE p.teamid = teamid
  RETURN p
```

bunkat commented Apr 15, 2015

> Cypher version: CYPHER 2.2, planner: RULE. 224 total db hits in 94 ms (10 run avg).

Using the index hint (did not know that node properties weren't allowed...) brings the performance of the COST planner in line with the RULE planner for this example. It still isn't faster, but they are basically equivalent.

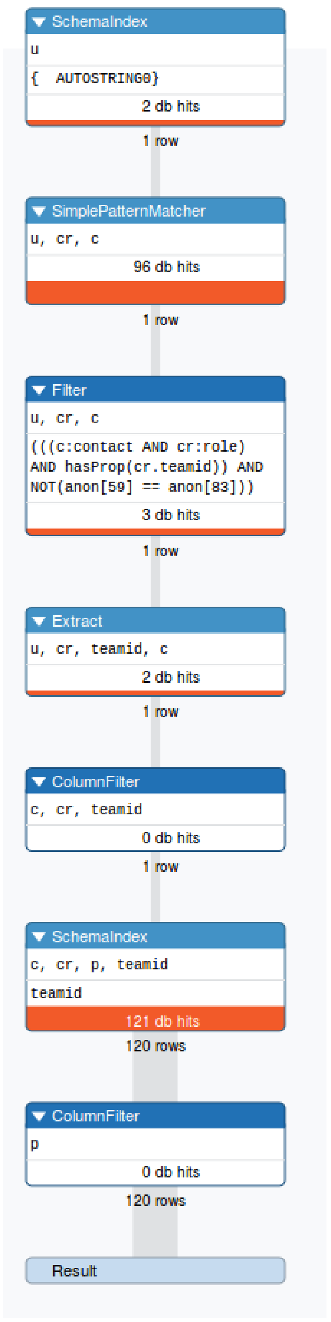
jexp commented Apr 15, 2015 Collaborator

Node properties and other expressions actually are allowed, it has to be fixed.

Ya for this simple query I doubt that the performance would be much different. Feel free to share the profile output for both planners.

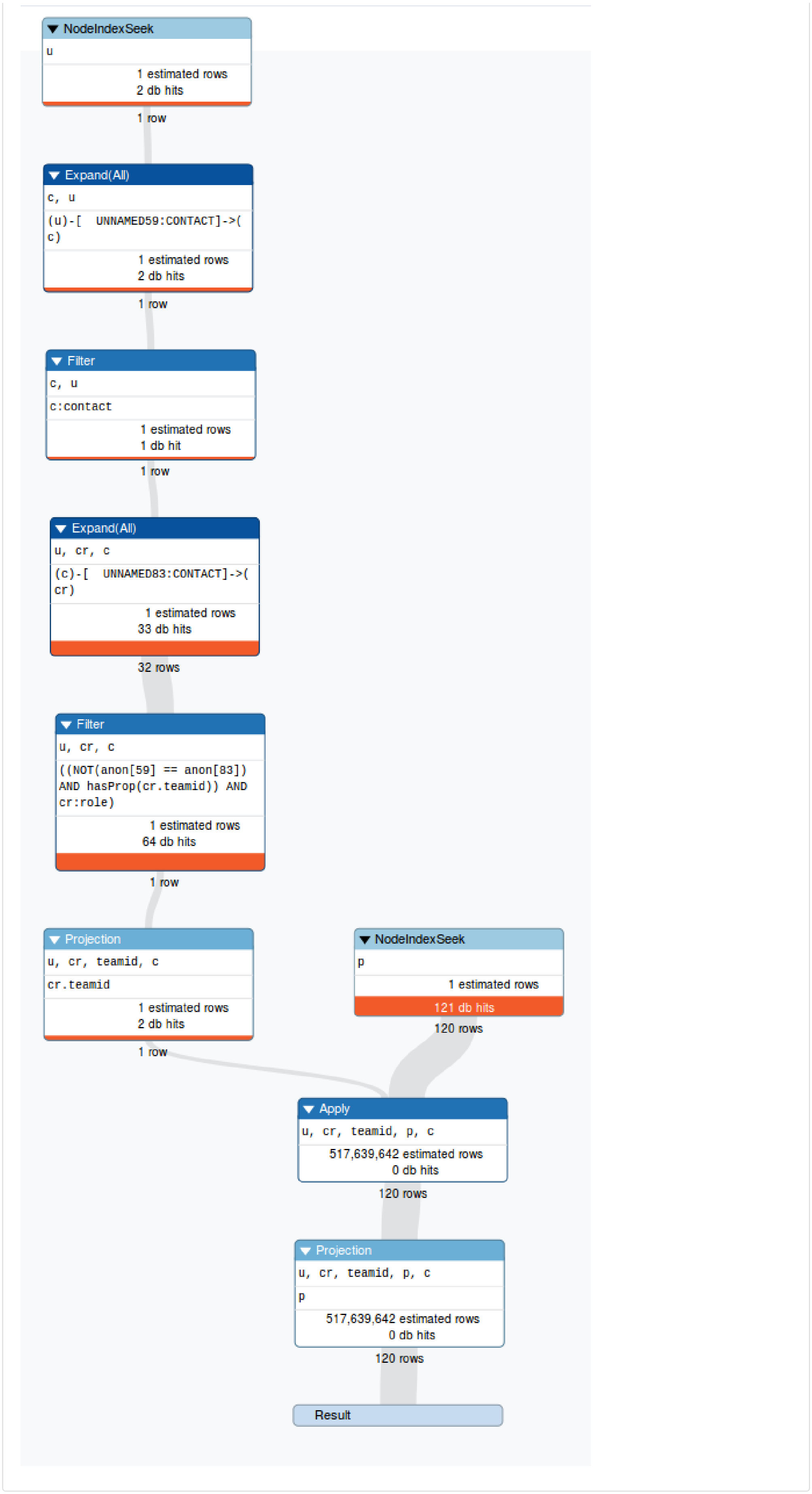
bunkat commented Apr 15, 2015

RULE



COST

The final Apply and Projection steps look a little funny, but I guess that is just the COST planner doing the join?



```
NodeIndexSeek
u
1 estimated rows
2 db hits
1 row

Expand(All)
c, u
(u)-[  UNNAMED59:CONTACT]->(
c)
1 estimated rows
2 db hits
1 row

Filter
c, u
c:contact
1 estimated rows
1 db hit
1 row

Expand(All)
u, cr, c
(c)-[  UNNAMED83:CONTACT]->(
cr)
1 estimated rows
33 db hits
32 rows

Filter
u, cr, c
((NOT(anon[59] == anon[83])
AND hasProp(cr.teamid)) AND
cr:role)
1 estimated rows
64 db hits
1 row

Projection
u, cr, teamid, c
cr.teamid
1 estimated rows
2 db hits
1 row

NodeIndexSeek
p
1 estimated rows
121 db hits
120 rows

Apply
u, cr, teamid, p, c
517,639,642 estimated rows
0 db hits
120 rows

Projection
u, cr, teamid, p, c
p
517,639,642 estimated rows
0 db hits
120 rows

Result
```

jexp added the **cypher** label May 29, 2015