Duality through Inversion: Syntax and Semantics of Cypher for multiple Graphs

Stefan Plantikow

stefan.plantikow@neotechnology.com

Lead openCypher Language Group (CLG)
Neo Technology

opencypher.org | opencypher@googlegroups.com
About this talk

• This is a proposal on how to add support for working with multiple graphs to the Cypher property graph query language, as part of the openCypher project.

• It is informational only: none of the ideas and proposed features presented in the following material are a part of the Cypher standard nor are they available from Neo Technology in any product offering, nor does this presentation represent any commitment that Neo Technology is going to provide such features in the future.
Why Multiple Graphs?

- Management of multiple graphs inside a GDBMS
  - Federation across organizational boundaries
  - Natural Sharding and Partitioning
    - e.g. by country, region etc.
  - Structuring the graph data set for operational purposes
  - System graph
  - Access control
  - Snapshots
  - Versioning
Why Multiple Graphs?

• **Views:** Transform, filter, aggregate graphs inside the GDBMS
  • \(<G> \rightarrow <G'>\)
  • Application data provisioning
  • Incremental maintenance of aggregates
  • Analytical processing in big data systems

• **Graph Visualization**
  • How to return relevant entities in a systematic fashion?
Why Multiple Graphs?

- Inter-graph operations
  - What is the difference graph between now and yesterday?
  - How do these two cliques of social graphs intersect?
  - How to contract parts of a larger graph to see the bigger picture?
  - Updating graphs

- Graphs as a modeling tool
  - How do we represent a route in the graph? (e.g. Travel trips, Bus routes)
  - How do we relate larger parts of the same graph to each other? (e.g. Fraud ring tracking)
Support for multiple graphs impacts...

**Physical Model** Where are graphs (nodes, relationships) stored? How are they addressed?

**Logical Model** How to add discrete multiple graphs to the Property Graph Model?

**Language** How are graphs represented in Cypher? As values? Between operators?

**Client** How are graphs returned to the client?

**Services** Existing higher-order features and services
Physical model: Multiple Graph Spaces

Graph Space I (persisted to disk)

Graph Space II (in-memory session)

Graph Space III (snapshot on disk)

GDBMS A (e.g. a cluster)

GDBMS B (e.g. a single host)
Existence ≠ Containment

An entity **exists** in a single *associated* graph space.

A node or a relationship **is contained** in at least one or more graphs.
Logical Model: Graphs as entity containers

• Graphs are entity containers
• May have
  • Address
  • Labels(?)
  • Properties

• Graphs may contain nodes
• Graphs may contain relationships
  (including start and end nodes)
• Each node or relationship must be contained
  in at least one graph
Towards a language for multiple graphs

Given model with multiple graphs which are

- addressable,
- have properties,
- have labels,
- and are possibly updateable.

How do we change Cypher systematically?
How do we change Cypher systematically?

- Cypher already operates on a graph
- Let's invert existing operations
- To find dual operations
- And invent dual syntax for them

"Duality through Inversion"
Which operations?

Let's analyze and invert

- RETURN
- MATCH (a)-[r]->(b)
- WITH
- THEN

In terms of

- Table Effect (Read, Amend, ...)
- Graph Effect (Read, Load, ...)
- Change in cardinality
RETURN => ...
RETURN => LOAD GRAPH

Load Graph

LOAD GRAPH ...

Preserve Table

1 : 1
LOAD GRAPH

Insights

- Cypher operators (clauses) already work on tabular and graph data
- Inverting an existing clause leads to interesting, rich new behaviour
- Proposed new clause

LOAD GRAPH AT "graph://graphology.com/europe/uk"
MATCH => ...

Read Graph

MATCH (a)-[r]->(b)

Amend Table from Graph

1 : n

Assumption: Pattern does not reference fields from the table
MATCH => RETURN GRAPH

Return Graph from Table

Consume Table

\[ n : 1 \]

Variation: Use RETURN GRAPH to end a query
## Tables from Graphs

It's easy to construct tables from a graph... but what's the inverse?

**MATCH (a)-->(b) WITH a, b ...**

<table>
<thead>
<tr>
<th>GRAPH</th>
<th>MATCHES</th>
<th>RECORDS</th>
<th>TABLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>(#1)--&gt;(#2)</td>
<td>a: #1, b: #2</td>
<td>a: #1, b: #2</td>
</tr>
<tr>
<td>#1</td>
<td>(#1)--&gt;(#3)</td>
<td>a: #1, b: #3</td>
<td>a: #1, b: #3</td>
</tr>
<tr>
<td>#3</td>
<td>(#3)--&gt;(#2)</td>
<td>a: #3, b: #2</td>
<td>a: #3, b: #2</td>
</tr>
<tr>
<td>#3</td>
<td>(#3)--&gt;(#4)</td>
<td>a: #3, b: #4</td>
<td>a: #3, b: #4</td>
</tr>
<tr>
<td>#4</td>
<td>(#4)--&gt;(#2)</td>
<td>a: #4, b: #2</td>
<td>a: #4, b: #2</td>
</tr>
</tbody>
</table>
Graphs from Tables

...a graph is a set of pattern matches!

WITH a, r, b RETURN GRAPH FROM (a)-[r]->(b) ...

TABLE

<table>
<thead>
<tr>
<th>a: #1, r: #5, b: #2</th>
</tr>
</thead>
<tbody>
<tr>
<td>a: #1, r: #6, b: #3</td>
</tr>
<tr>
<td>a: #3, r: #7, b: #2</td>
</tr>
<tr>
<td>a: #3, r: #8, b: #4</td>
</tr>
<tr>
<td>a: #4, r: #9, b: #2</td>
</tr>
</tbody>
</table>

RECORDS

<table>
<thead>
<tr>
<th>a: #1, r: #5, b: #2</th>
</tr>
</thead>
<tbody>
<tr>
<td>a: #1, r: #6, b: #3</td>
</tr>
<tr>
<td>a: #3, r: #7, b: #2</td>
</tr>
<tr>
<td>a: #3, r: #8, b: #4</td>
</tr>
<tr>
<td>a: #4, r: #9, b: #2</td>
</tr>
</tbody>
</table>

MATCHES

<table>
<thead>
<tr>
<th>(#1)-[#5]-&gt;(#2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(#1)-[#6]-&gt;(#3)</td>
</tr>
<tr>
<td>(#3)-[#7]-&gt;(#2)</td>
</tr>
<tr>
<td>(#3)-[#8]-&gt;(#4)</td>
</tr>
<tr>
<td>(#4)-[#9]-&gt;(#2)</td>
</tr>
</tbody>
</table>

GRAPH

#1 -> #2
#2 -> #3
#3 -> #4
#4 -> #5
#5 -> #6
#6 -> #7
#7 -> #8
#8 -> #9
#9 -> #1
MATCH

MATCH
p=(a)-[r]->(b)

RETURN
RETURN
GRAPH
GRAPH
FROM
FROM

MATCH

MATCH
p=(a)-[r]->(b)

RETURN
RETURN
GRAPH
GRAPH
FROM
FROM

bound

bound
verify exists in graph
verify exists in graph

unbound

unbound
match in graph
match in graph

path binding

path binding
p=...
bind as given
bind as given

bare path/entity

bare path/entity
p
verify exists in graph
verify exists in graph

mirror/copy

mirror/copy
(=b)
ensure equivalence
ensure equivalence

Syntax Ideas?
## Graph/Graphlet Duality

<table>
<thead>
<tr>
<th></th>
<th>&quot;big&quot; graphs</th>
<th>&quot;small&quot; graphlets</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Occurrence</strong></td>
<td>Between operators</td>
<td>Embedded into records</td>
</tr>
<tr>
<td><strong>Construction</strong></td>
<td>Set of pattern instances</td>
<td>Pattern instance</td>
</tr>
<tr>
<td><strong>Access</strong></td>
<td>Via clauses</td>
<td>Via patterns</td>
</tr>
<tr>
<td></td>
<td></td>
<td>As values (but only for paths!)</td>
</tr>
</tbody>
</table>
Graphlet Patterns

MATCH g=(a)-[r]->(b) + (c) + (d)-[r2]->(c)

\( g \) is of type GRAPHLET, every PATH is a GRAPHLET

- union: \( g_1 + g_2 \)
- difference: \( g_1 - g_2 \)
- intersection: \( g_1 \& g_2 \)
- exclusive union: \( g_1 \wedge g_2 \)
- annotation: \( g \{ :\text{Label prop: 12} \} \)
Graphlet Fusion

MATCH  \( g=(a)-[r]->(b) + (c) + (d)-[r2]->(c) \)

Aggregation functions for graphlets \( g \)

- **union**: \( \text{union}(g) \)
- **intersection**: \( \text{intersect}(g) \)
- ...
Generic Graph Construction

```
RETURN GRAPH FROM <pat>
```

may be generalized into

```
RETURN GRAPH FROM g=<pat> THROUGH union(g)
```

Alternative: Use Update Syntax! `GRAPH FROM { CREATE () ... }`
WITH => ...

Preserve Graph

Transform Table

1 : 1
WITH => WITH GRAPH

Transform Graph

Preserve Table
1 : 1

G

T

G’

T
MATCH (c:City {name: "London"})
MATCH (c)<-[LIVES_IN]-(a)-[LOVES]-(b)-[LIVES_IN]->(c)

WITH GRAPH FROM (x)-[KNOWS]-(y)
WHERE NOT EXISTS { (x)-[LOVES]-(y) }

MATCH (a)-(single_friend)-(b)
RETURN *
And THEN?

Pass on Graph

Pass on Table
Query Composition with THEN

No inversion needed, **everything** is always passed on, one issue though:

- **RETURN * THEN . . .**
  Discards the graph and switches back to the default graph

- How to pass on the current graph when otherwise returning records?
  **RETURN WITH GRAPH * THEN ...**
Query Combinators

Requires argument queries to end in **RETURN GRAPH**

- **UNION GRAPH**
- **INTERSECT GRAPH**
- **EXCLUSIVE UNION GRAPH**
- ...

opencypher.org | opencypher@googlegroups.com
Inverted Syntax Summary

- LOAD GRAPH AT ...
- WITH GRAPH FROM ... THROUGH ...
- RETURN GRAPH FROM ... THROUGH ...
- RETURN WITH GRAPH ...
Duality Through Inversion

- Table-Graph-Duality
- Graph-Graphlet-Duality
- Load-Return-Duality
- Match-Return-Graph-Duality

Yet: Composition-Non-Duality
Where do graphs come from?

So far, we've looked at declarative graph construction and transformation

- How to manage **persistent** graphs?
- How to create **persistent** graphs?
- How to delete **persistent** graphs?
- ...

This is another duality: Virtual vs Persistent Graphs!
Graph Pattern Overview

- GRAPH AT "url"
- GRAPH FROM <pat> THROUGH <pat-aggr>
- EMPTY GRAPH
- CURRENT GRAPH
- DEFAULT GRAPH
- Persistent graphs?
Managing Persistent Graphs

CREATE GRAPH <MyGraph:Label {name: 15}>
AT "url"
FROM g=(a)-[r]->(b)
THROUGH union(g)

ADD/MOVE GRAPH g TO "url"
REMOVE GRAPH g FROM "url"

DELETE GRAPH g
Working with Persistent Graphs

MATCH GRAPH <MyGraph:Label {name: 15}> ... 
WITH GRAPH <MyGraph:Label {name: 15}> FROM ... THROUGH ... 

LOAD GRAPH MyGraph 
MATCH IN GRAPH ... 
SET MyGraph.name = 15 
REMOVE MyGraph.name 
ADD TO GRAPH ... 
REMOVE FROM GRAPH ... 
ADD MyGraph:SocialNetwork 
REMOVE MyGraph:Experiment
Graphs vs. Graphlets

- MATCH  g=...  g has Cypher type GRAPHLET
- MATCH  GRAPH  <G>  G has type GRAPH

Not exactly the same

- Graphs as values are handles/references
- Different operations (Immutable graphlets vs Mutable graphs)
- Graphs may be coerced into graphlets though
Let's compare languages: SQL

Transform Table

SQL
Let's compare languages: CYPHER 3.x
Let's compare languages: CYPHER vNEXT

Multiple graphs require a **fundamental** change to "Projection Graphs".
Summary

- CYPHER vNEXT with support for multiple graphs is coming
- It implies a foundational change to the whole language:
  
  Cypher changes from a table transformation language...
  ...into a graph transformation language!

- We work on this in the context of Cypher for Apache Spark
- Many open points: CIPs, Details, Aggregation, Views ...

- **Please help us build this and get involved, e.g. via the oCIG calls!**
Thank you!

Questions?